Modul za obavljanje korisnički definiranih programa u sustavu za automatsko ocjenjivanje programskog kôda Edgar

Vinožganić, Jakov

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1315

MODUL ZA OBAVLJANJE KORISNIČKI DEFINIRANIH PROGRAMA U SUSTAVU ZA AUTOMATSKO OCJENJIVANJE PROGRAMSKOG KÔDA EDGAR

Jakov Vinožganić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1315

MODUL ZA OBAVLJANJE KORISNIČKI DEFINIRANIH PROGRAMA U SUSTAVU ZA AUTOMATSKO OCJENJIVANJE PROGRAMSKOG KÔDA EDGAR

Jakov Vinožganić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1315

Pristupnik:	**Jakov Vinožganić (0036541835)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Igor Mekterović

Zadatak: **Modul za obavljanje korisnički definiranih programa u sustavu za automatsko ocjenjivanje programskog kôda Edgar**

Opis zadatka:

Edgar je informacijski sustav za testiranje studenata i učenje razvijen na Fakultetu elektrotehnike i računarstva. Osnovna funkcionalnost mu je mogućnost strojnog ocjenjivanja programskih pitanja odnosno evaluaciju programskog kôda u različitim programskim jezicima kao što su SQL, C, Java, itd. Naravno, pritom se prikupljaju različiti podatci, od praćenja samog pisanja ispita do rezultata na ispitima. Edgar omogućuje brojne izvještaje i vizualizacije koje pružaju uvid u podatke, ali ne omogućuje nastavniku da ih prilagođava i mijenja. Također, neke složenije analize, poput analize vršnjačkog ocjenjivanja, su izvedene putem skripti u programskom jeziku R koje rade nad podatcima koje je potrebno ručno pribaviti iz baze podataka u obliku CSV datoteka. Potrebno je proučiti i klasificirati kakve potrebe u smislu izračunavanja i ulaznih podataka imaju korisnici Edgara i različiti Edgarovi moduli (npr. detekcija plagijata, izračun težine pitanja, izvještaji o kvaliteti ispita, rudarenje podataka, itd.). Potom proučiti arhitekturu sustava Edgar s naglaskom na podsustav "CodeRunner" koji omogućuje izvršavanje proizvoljnog, potencijalno malicioznog, programskog koda u zaštićenoj okolini (engl. sandbox). Konačno, napraviti prototip sustava koji omogućuje definiciju i obavljanje poslova u zakazanom periodu. Posao uključuje definiciju ulaznih podataka koji bi se pribavljali iz Edgarove baze podataka, programa koji će raditi nad tim podatcima i izlaznim rezultatima. Za obavljanje je potrebno koristiti postojeći CodeRunner podsustav. Voditi dnevnik obavljanja poslova, omogućiti uređivanje i pokretanje posla na zahtjev, te eventualne notifikacije o statusu obavljanja posla. Donijeti ocjenu ostvarenog pristupa te smjernice za budući razvoj.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

Uvod

Edgar je napredni informacijski sustav razvijen na Fakultetu elektrotehnike i računarstva (FER) s ciljem automatiziranog testiranja znanja studenata. Ovaj složeni sustav, koji spada u klasu automatiziranih sustava za programsku provjeru znanja (engl. *APAS*, *Automated Programming Assessment System*), osmišljen je za strojno ocjenjivanje programskih zadataka. *APAS* sustavi, kao što je *Edgar*, omogućuju automatsku evaluaciju koda, smanjujući potrebu za ručnim ispravljanjem zadataka i osiguravajući brzu povratnu informaciju korisnicima, u ovom slučaju studentima. Edgar podržava različite programske jezike poput C-a, Jave, i SQL-a, omogućujući brzo i precizno ocjenjivanje rješenja, što značajno olakšava rad nastavnicima i povećava učinkovitost obrazovnog procesa.

Razvoj *Edgara* motiviran je potrebom za smanjenjem vremena i truda potrebnog za ručno ocjenjivanje programskih zadataka. Ručno ocjenjivanje zahtjeva značajne resurse, a s obzirom na različite načine rješavanja programskih problema, ručna provjera može biti složena i dugotrajna. Na sveučilištima i fakultetima s velikim brojem studenata, poput FER-a, kašnjenje povratnih informacija može otežati praćenje napretka studenata i prilagodbu nastavnih metoda njihovim potrebama. *Edgar* rješava ove probleme automatizacijom ocjenjivanja te time omogućuje nastavnicima fokus na druge važne zadatke.

Edgarov sustav ocjenjivanja temelji se na unaprijed definiranim testnim slučajevima koji se koriste za provjeru točnosti studentskog koda. Na primjer, za zadatke u C++-u, sustav koristi unaprijed definirane ulazne podatke i očekivane izlaze za testiranje ispravnosti algoritama. Sustav automatski izvršava studentski kod i uspoređuje dobivene rezultate s očekivanim. Ovaj pristup studentima pruža pravovremenu povratnu informaciju o njihovom radu.

Direktnu pomoć nastavnika ili asistenata u stvarnom vremenu nudi *ticketing* sustav. On omogućuje studentima postavljanje pitanja u vezi zadataka, čime se poboljšava njihovo razumijevanje i učinkovitost tijekom rješavanja problema. Ovaj je sustav posebno koristan tijekom ispita, kada studenti mogu naići na tehničke poteškoće ili nejasnoće vezane uz semantiku samog zadatka. Studenti putem sustava mogu postaviti svoja pitanja, a pomagači odgovaraju na njih u stvarnom vremenu, osiguravajući da studenti uvijek imaju potrebnu podršku kako bi mogli nastaviti s rješavanjem bez većih zastoja. Važno je napomenuti da

sustav nije dizajniran za pružanje pomoći u samom rješavanju problema. Umjesto toga, pomagači odgovaraju na tehnička pitanja ili daju pojašnjenja vezana uz specifikacije zadatka. Na primjer, studenti mogu pitati o pravilnom načinu korištenja određenih funkcija u programskom jeziku, razjasniti nesporazume oko uvjeta zadatka ili dobiti informacije o dostupnim resursima i alatima. Time se osigurava da studenti imaju jasno razumijevanje onoga što se od njih traži, dok se istovremeno održava integritet ispita. *Ticketing* sustav također omogućuje nastavnicima praćenje vrsta pitanja koja studenti postavljaju, što može pružiti vrijedne uvide u područja gdje studenti najčešće nailaze na probleme. Na taj način, nadležni mogu prilagoditi nastavu kako bi unaprijed adresirali te poteškoće, što dugoročno poboljšava kvalitetu obrazovanja.

Sustav *Edgar* također uključuje, danas neizostavne, napredne mehanizme za praćenje aktivnosti studenata tijekom rješavanja zadataka, laboratorijskih vježbi i ispita kako bi se osigurala poštenost i spriječile ilegalne radnje. Alat bilježi neke od radnji koje mogu ukazivati na potencijalno varanje, kao što su izlazak iz preglednika te promjene u fokusu na zaslonu. Svaka takva radnja evidentira se s vremenskim oznakama, omogućujući nastavnicima detaljan uvid u ponašanje studenata tijekom ispita. Na ovaj način, sustav pomaže u prepoznavanju i dokumentiranju svih sumnjivih aktivnosti koje bi mogle ukazivati na pokušaje varanja ili prepisivanja. Ove informacije nastavnicima pružaju dodatni alat za održavanje integriteta evaluacijskog procesa i osiguranje da su rezultati ispita odraz stvarnih znanja i vještina studenata.

Neke složenije funkcionalnosti, poput analize vršnjačkog ocjenjivanja, provode se putem skripti u programskom jeziku R. Ove skripte rade nad podacima koje je potrebno ručno preuzeti iz baze podataka u obliku CSV datoteka, čime se omogućuje dodatna fleksibilnost i prilagodba u analizi rezultata.

U ovom je završnom radu razvijen modul za obavljanje korisnički definiranih skripti nad podatcima iz sustava za automatsko ocjenjivanje programskog kôda (*Edgar).* Modul je izveden kao web-aplikacija. Modul omogućuje automatizirani dohvat proizvoljnih podataka nad kojim se onda obavljaju skripte napisane u programskom jeziku R. Nudi jednostavan i intuitivan način za kreiranje koraka obrade podataka te pregled međurezultata i konačnih rezultata analiza. Sustav vodi detaljan dnevnik izvođenja svih koraka te time osigurava transparentno praćenje procesa i rezultata. Na ovaj način, modul omogućuje korisnicima *Edgara* da obavljaju vlastite analize nad podatcima na elegantan i automaziran način.

1. Statistička obrada podataka i vizualizacija podataka u Edgaru

1.1. Analiza podataka

Pri svim funkcionalnostima sustava *Edgar*, kao što su rješavanje zadataka, laboratorijskih vježbi, ispita te nadgledanje aktivnosti studenata, prikupljaju se različiti podatci. Ovi podatci uključuju detalje o vremenu provedenom na rješavanju zadataka, uspjeh u ispunjavanju različitih testnih slučajeva, učestalost pristupa sustavu, te specifične radnje poput promjena fokusa na zaslonu tijekom ispita. Uz to, sustav prikuplja informacije o rezultatima domaćih zadaća, ispita, laboratorijskih vježbi i sl.

Jedna od ključnih prednosti *Edgara* je njegova sposobnost generiranja raznih izvještaja i vizualizacija temeljenih na prikupljenim podacima. Ovi izvještaji nastavnicima pružaju detaljan uvid u uspješnost studenata, identifikaciju čestih grešaka te praćenje napretka tijekom vremena. Vizualizacije uz pomoć dijagrama omogućuju brže i intuitivnije razumijevanje podataka, olakšavajući nastavnicima donošenje informiranih odluka o prilagodbi nastavnih metoda i materijala.

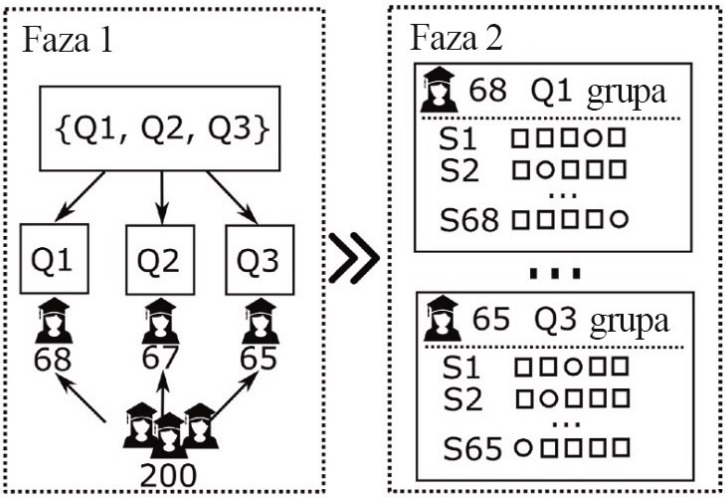
Posebno je korisno što sustav omogućava prikaz statistike za sve ispite održane tijekom akademske godine. Na primjer, kutijasti dijagrami (engl. *boxplot*) prikazuju distribuciju bodova svake provjere znanja, pružajući vizualni prikaz raspona rezultata studenata, uključujući minimalne i maksimalne vrijednosti, donji i gornji kvartil te medijan. Klikom na bilo koji dijagram prikazuju se detalji o odabranom ispitu, uključujući postotak prolaznosti, vrijeme pisanja i predavanja ispita te histogram s razdiobom bodova. Histogrami pružaju dodatni uvid u raspodjelu bodova među studentima, uključujući oznake za srednju vrijednost i medijan, što omogućuje detaljnije razumijevanje performansi studenata.

Linijski dijagrami (engl. *line charts*) mogu se koristiti za praćenje napretka studenata kroz vrijeme, prikazujući promjene bodova iz ispita u ispit ili iz vježbe u vježbu. Stupčasti dijagrami (engl. *bar charts*) omogućuju usporedbe između različitih grupa studenata ili različitih zadataka te time olakšavaju identifikaciju specifičnih područja gdje su potrebne intervencije ili dodatne upute.

Međutim, unatoč naprednim analitičkim mogućnostima, *Edgar* ne omogućuje nastavnicima prilagodbu i izmjenu generiranih izvještaja i vizualizacija. Sustav nudi unaprijed definirane formate izvještaja, što može ograničiti fleksibilnost u analizi podataka prema specifičnim potrebama svakog nastavnika ili kolegija. Ova ograničenja mogu predstavljati izazov kada je potrebno detaljnije istražiti određene aspekte studentskih performansi ili prilagoditi analize za specifične istraživačke projekte. Stoga, nastavnici koji trebaju dodatne prilagodbe izvještaja moraju izvoziti podatke iz sustava u druge analitičke alate kako bi proveli dublje analize. Primjerice, podatci se mogu preuzeti u obliku CSV datoteka i dalje obrađivati u alatima kao što su *Excel* ili u specijaliziranom statističkom softveru poput R-a, koji je namijenjen statističkoj analizi i vizualizaciji podataka, ili Pythona, koji iako je programski jezik za opću namjenu, nudi snažne biblioteke za statističku analizu [1].

1.2. Vršnjačko ocjenjivanje

Analiza vršnjačkog ocjenjivanja unutar sustava *Edgar* provodi se s pomoću skripti napisanih u programskom jeziku R, koje se temelje na ručno izvučenim podacima iz baze podataka u obliku CSV datoteka. Ovaj postupak omogućava dubinsku analizu performansi i pristranosti ocjenjivanja među studentima. Proces započinje ažuriranjem točnih odgovora na kalibracijske testove.



Slika 1.1. Postavke vršnjačkog ocjenjivanja: Faza jedan uključuje dodjelu tri nasumično dodijeljena pitanja za 200 studenata koji polažu ispit, grupiranih u tri skupine pitanja. U drugoj fazi studenti ocjenjuju pet radova: četiri nasumično odabrana rada svojih kolega i jedan kalibrirani rad (označen krugom), prethodno ocijenjen od strane nastavnika [3]

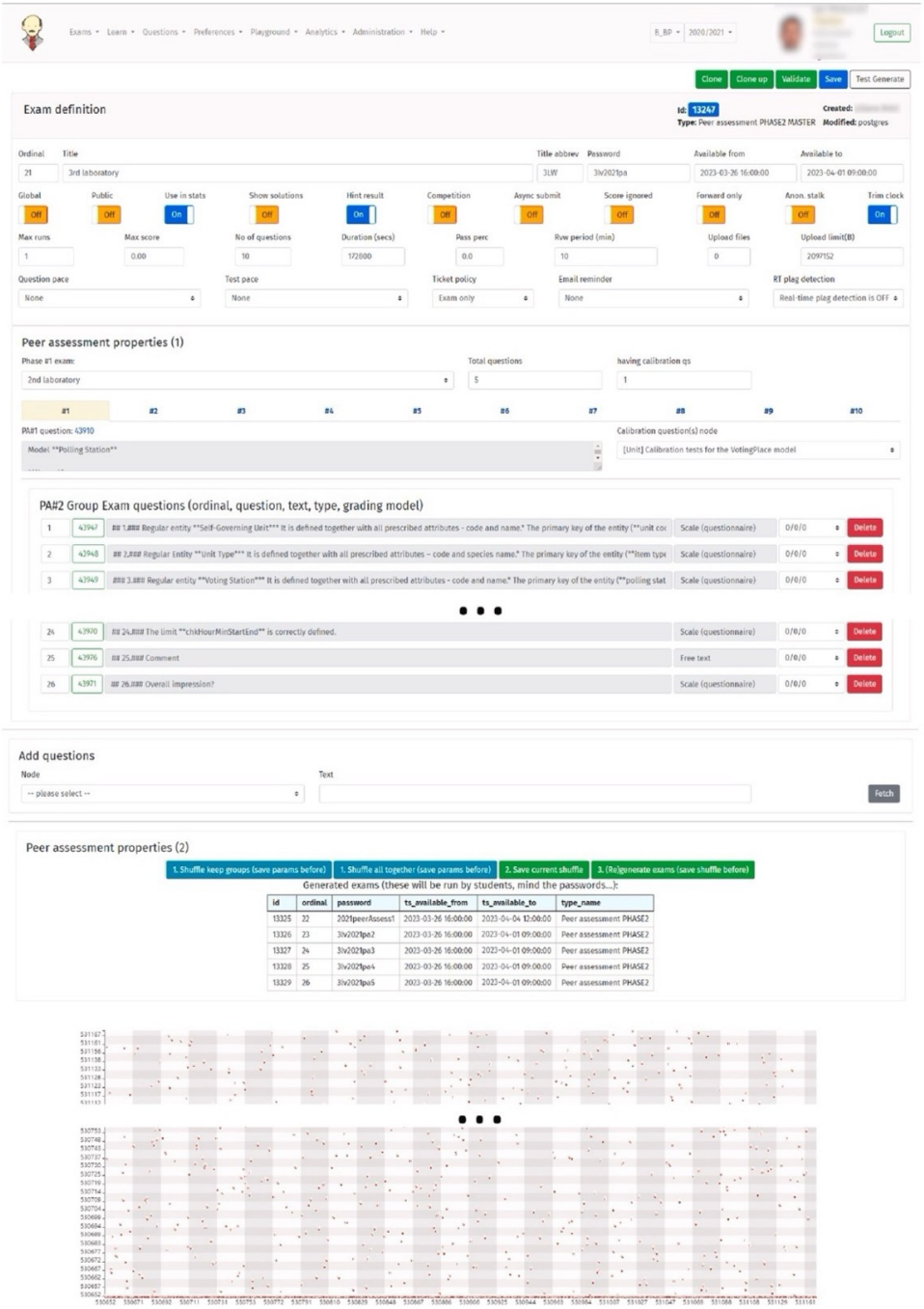
Sažeto objašnjenje postupka za analizu i dodjeljivanje ocjene u postupku vršnjačkog ocjenjivanja u koracima:

1. Nastavnik ili asistent treba izmijeniti parametre u skriptama za izdvajanje podataka, obaviti izvođenje tih skripti kako bi se generirala odgovarajuća CSV datoteka te preuzeti tu datoteku na svoje računalo. Ta datoteka se zatim pohranjuje u projektni direktorij R.
2. Pokreće se R skripta koja učitava prethodno generiranu CSV datoteku i provodi analize nad podacima. Ova skripta generira dvije ključne CSV datoteke koje sadrže informacije o pristranosti ocjenjivanja i rezultate analize. Na temelju tih rezultata, generiraju se SQL naredbe koje se koriste za ažuriranje ocjena u bazi podataka sustava *Edgar*.
3. Stvaranje privremene tablice u bazi podataka na temelju jedne od generiranih CSV datoteka, što omogućava lakše manipuliranje podacima. Zatim se ponavlja proces izmjene parametara i izvođenja skripti za izdvajanje podataka za drugi korak analize, preuzimanje nove CSV datoteke te pokretanje odgovarajuće R skripte koja provodi daljnju analizu. Ova druga analiza generira dodatne CSV datoteke koje se koriste za generiranje SQL naredbi za ažuriranje ocjena i davanje povratnih informacija studentima o pristranosti u ocjenjivanju.

Ukratko, u prvom koraku analize („*Phase #1"*), definira se test s jednim fiksnim pitanjem slobodnog teksta. Studenti mogu vidjeti pitanje, unijeti svoj odgovor te priložiti datoteke. Nakon predaje testa, njihove se datoteke komprimiraju i mogu se preuzeti pojedinačno ili u cjelini. Nakon toga, u drugom koraku („*Phase #2*"), studenti ocjenjuju radove svojih kolega i kalibracijske primjere koje su pripremili nastavnici. S pomoću kalibracijskih primjera dolazi se do ocjene studentove pristranosti (engl. *bias*) te se na temelju toga može ocijeniti sposobnost studentova ocjenjivanja tuđih radova. Ovaj proces uključuje definiranje „*master PA*" testa, generiranje stvarnih *PA testova* te distribuciju lozinki studentima. Studenti ocjenjuju radove putem zasebnih „*PA Phase #2*" testova, a ocjene se generiraju na temelju unaprijed definiranih pitanja i kriterija ocjenjivanja.

Tijekom cijelog postupka koristi se niz različitih analitičkih alata i SQL skripti s vrlo složenim upitima (engl. *queries*) kako bi se osigurala točnost i pravednost ocjenjivanja. Osoba koja izvršava ovaj postupak mora biti dobro upoznata s tehnologijama koje se koriste,

poput SQL-a i programskog jezika R, te mora detaljno razumjeti prethodno objašnjeni postupak. To zahtijeva tehničko znanje i sposobnost precizne primjene analitičkih metoda, kako bi se osigurala točnost i pravednost u ocjenjivanju studentskih radova [2][3].



Slika 1.2. Postavke faze dva ispita u *Edgaru*: nakon odabira ispita iz faze jedan, sustav dinamički generira grupe (#1-#10) na temelju pitanja iz prvog ispita. Postavke uključuju definiranje broja radova za ocjenjivanje (5), kalibracijska pitanja (1) i set pitanja (obično „*Likertova skala*“) za svaku grupu. Na primjer, Grupa #1 sadrži 26 pitanja, uključujući ne-*Likertova* pitanja “Komentar” i “Opći dojam”. Postavke također omogućuju nasumičnu dodjelu parova radova i ocjenjivača ("miješanje") za određeni broj radova. [3]

2. Postojeće funkcionalnosti analize podataka i potencijalni dodatni zahtjevi korisnika

Kako bi se osigurala optimalna učinkovitost i kvaliteta obrazovnog procesa u sustavu *Edgar*, važno je analizirati postojeće funkcionalnosti sustava te identificirati potencijalne nadogradnje i nove funkcionalnosti koje bi korisnici mogli zahtijevati. Nastavnici i asistenti trebaju imati pristup preciznim i pravovremenim informacijama koje im omogućuju efikasno praćenje i evaluaciju studentskih aktivnosti. *Edgar* trenutno nudi određene funkcionalnosti za analizu podataka, kao što su izračun težine pitanja i izvještaji o kvaliteti ispita. Ove funkcionalnosti su korisne, ali postoji mogućnost daljnjeg unapređenja kako bi se bolje zadovoljile potrebe korisnika. S druge strane, funkcionalnosti poput dubinske analize podataka još nisu integrirane u sustav, iako bi mogle značajno unaprijediti analitičke mogućnosti *Edgara*. U nastavku će se razmotriti postojeće funkcionalnosti, njihovi trenutni kapaciteti i potencijalna unapređenja, kao i prijedlozi za nove analitičke alate.

Kratak uvid u postojeće funkcionalnosti te potencijalne nadogradnje:

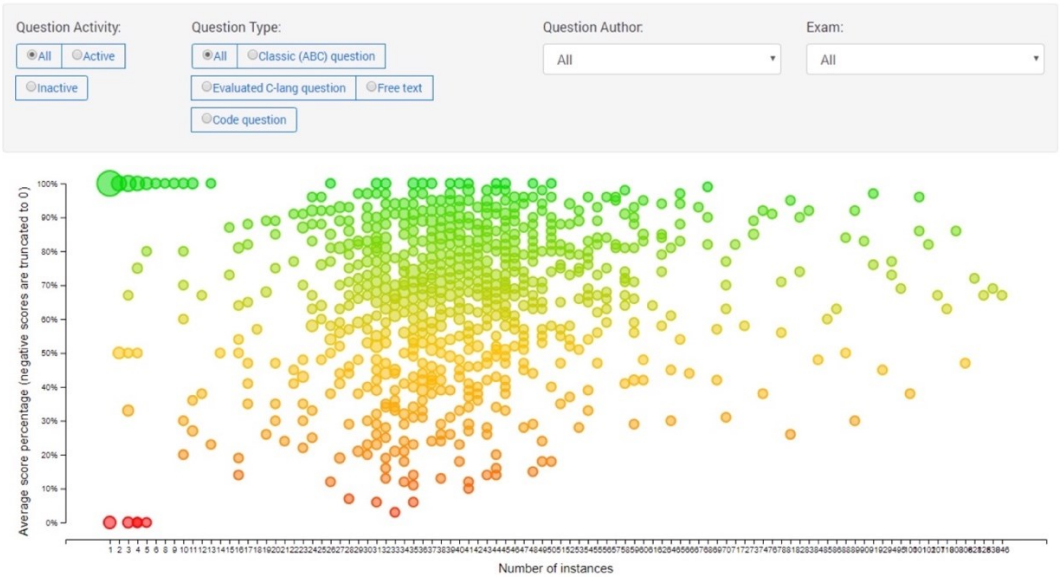
1. **Izračun težine pitanja:** U sustavu postoji analiza težine pitanja. Ona omogućava nastavnicima pregled uspješnosti rješavanja pojedinih zadataka od strane studenata, što im pruža uvid u složenost pitanja i prilagođavanje zadataka potrebama studenata. Moguće je dodatno unaprijediti ovu funkcionalnost primjenom naprednih statističkih modela i drugih metoda koje omogućuju detaljniju analizu težine zadataka.
2. **Izvještaji o kvaliteti ispita:** *Edgar* pruža osnovni uvid u rezultate ispita putem vizualnih prikaza. Nastavnicima je omogućen pregled rezultata ispita, postotak prolaznosti, te distribuciju ocjena među studentima. Unapređenja bi uključivala detaljniju analizu kvalitete ispita, kao što su izračun indeksa lakoće, analiza slučajnog pogotka, fleksibilna analiza podskupina studenata itd.
3. **Dubinska analiza podataka (engl. *data mining*):** Premda *Edgar* trenutno ne podržava funkcionalnosti za dubinsku analizu podataka, implementacija ovih tehnika mogla bi značajno unaprijediti analizu i praćenje studentskih aktivnosti. Ona uključuje korištenje tehnika strojnog učenja i statističke analize kako bi se iz velikih skupova podataka izvukli korisni obrasci i predikcije.

2.1. Izračun težine pitanja

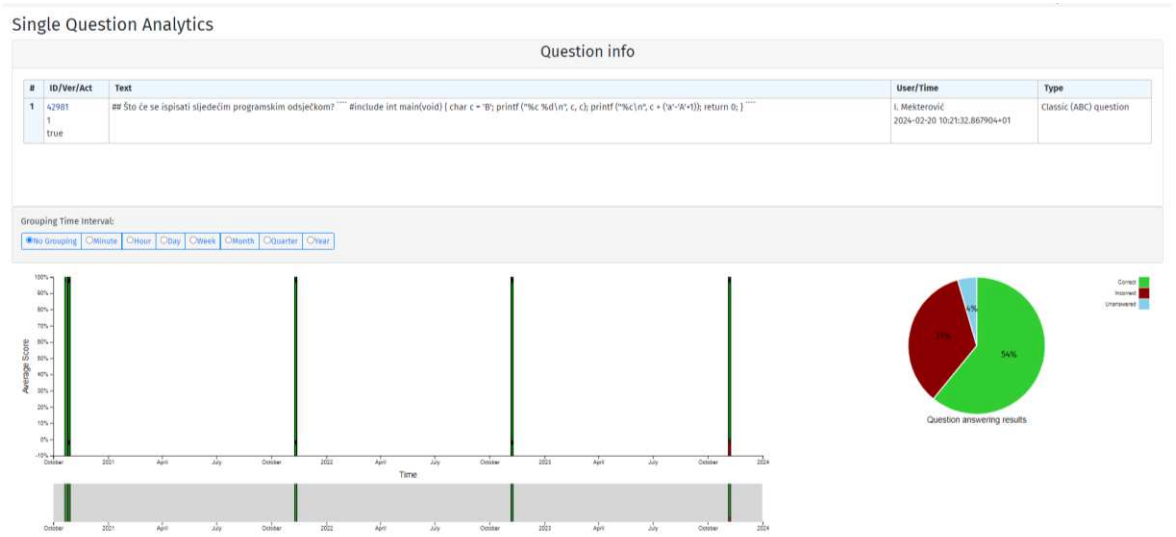
Izračun težine zadataka u sustavu *Edgar* ključan je za osiguranje da su zadaci koje studenti rješavaju prikladno izazovni i pravedno bodovani. Ova funkcionalnost omogućava nastavnicima da kalibriraju težinu zadataka, što je od velikog značaja pri procjeni studentskih vještina i znanja. Ove se procjene automatski generiraju na temelju povijesti rješavanja zadataka, što nastavnicima pruža uvid u to kako studenti pristupaju različitim vrstama problema.

U *Edgaru* postoji funkcionalnost za analizu težine pitanja, dostupna nastavnicima putem opcije *„Question Analytics“*. Tamo se prikazuje dijagram koji prikazuje instance zadataka. Klikom na bilo koju točku na dijagramu, otvara se detaljan prikaz informacija o tom pitanju. Na ovaj način, oni mogu dobiti pregled o uspješnosti rješavanja pojedinih zadataka od strane studenata.

Jedna od ključnih metrika je postotak bodova za pitanje, koji se izračunava prosjekom postotaka bodova svakog pitanja. Negativni bodovi zamjenjuju se s nulom kako bi se dobio realniji prikaz uspješnosti. Na primjer, ako su tri studenta riješila jedno pitanje s bodovima 1.0, 0.5 i -0.25, konačni postotak za to pitanje izračunava se kao prosjek niza brojeva 1.0, 0.5 i 0.0, što iznosi 0.5 [3].



Slika 2.1. Prikaz analize skupa pitanja [3]



Slika 2.2. Prikaz analize pojedinačnog pitanja

Nakon odabira točke na mjehuričastom dijagramu (engl. *bubble chart*), otvara se detaljna analiza pitanja putem opcije *„Single Question Analytics“*. Ovo omogućava nastavnicima da vide detalje svakog pojedinog pitanja, uključujući tekst pitanja, autora, vrijeme postavljanja te rezultate odgovora. Vizualni prikazi, poput tortnih dijagrama (engl. *pie charts*) i stupčastih dijagrama, pružaju uvid u uspješnost studenata na određenom pitanju, uključujući postotak točnih odgovora, netočnih odgovora te slučaja kada odgovor uopće nije odabran, te distribuciju bodova kroz vrijeme.

Ovo je nastavnicima od iznimne koristi, jer omogućava brzo i intuitivno praćenje uspješnosti studenata. Pomaže u identificiranju zadataka koji su preteški ili prelagani, omogućavajući im prilagodbu materijala kako bi bolje odgovarali potrebama studenata. Također, pomaže u prepoznavanju pitanja koja možda nisu jasno postavljena ili su problematična, što može dovesti do njihovog poboljšanja u budućnosti.

Premda je ova funkcionalnost uistinu korisna, postoji prostora za unapređenje. Trenutno, sustav ne omogućava dubinsku analizu pitanja koja bi mogla uključivati naprednije statističke modele i prilagodljive parametre izračuna, poput teorije odgovora na zadatke (engl. IRT - *Item Response Theory*) te drugih naprednijih tehnika. Na primjer, iz baze podataka mogli bi se izvući detaljni podaci o zadacima, uključujući informacije o broju pokušaja rješavanja, točnosti odgovora, vremenu provedenom na rješavanju i broju odabranih opcija. Ovi podaci zatim bi se obradili putem niza ulančanih skripti koje bi koristile različite algoritme za analizu težine zadataka. Svaka skripta mogla bi primijeniti

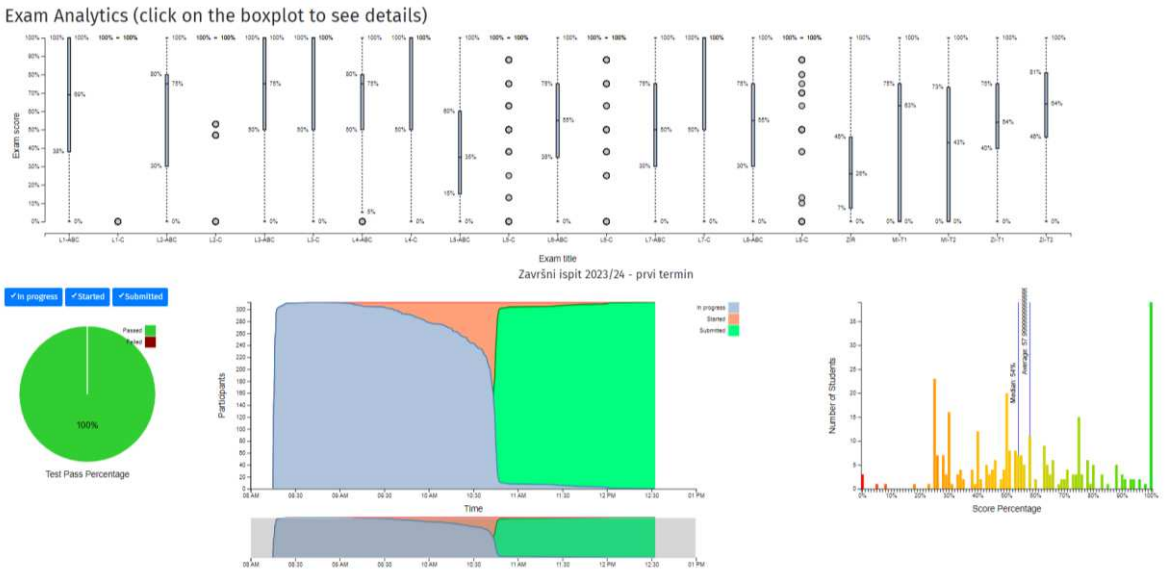
specifične metode, poput analize distribucije odgovora ili prilagodbe težine pitanja prema naprednim modelima. Na taj način, rezultati iz jedne skripte poslužili bi kao ulaz za sljedeću[3].

2.2. Izvještaji o kvaliteti ispita

Trenutno *Edgar* nudi dvije opcije za analizu ispita: ispitnu analitiku (engl. *exam analytics*) te analitika distribucije rezultata ispita (engl. *exam score distribution analytics*) - po svim grupama odjednom te po određenim odabranim grupama.

2.2.1. Ispitna analitika

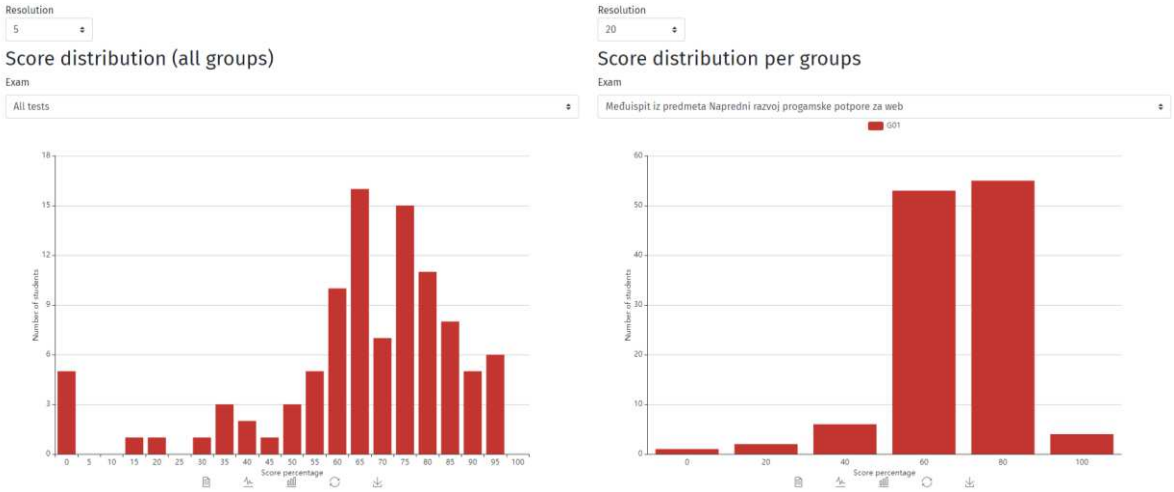
Opcija ispitne analitike omogućava pregled rezultata ispita kroz različite vizualne prikaze. Na kutijastom su dijagramu prikazane sve postojeće provjere znanja vezane za odabrani kolegij, kao što su domaće zadaće, laboratorijske vježbe i ispiti. Klikom na određenu provjeru na tom dijagramu, prikazuju se dodatni dijagrami s analizama specifičnim točno za tu provjeru. Tortni dijagram prikazuje postotak prolaznosti ukupnog broja studenata koji je sudjelovao na toj provjeri. Na složenom površinskom dijagramu (engl. *stacked area chart*) može se vidjeti kada su studenti započeli, rješavali i predavali rješenja te provjere. Zadnje, histogram prikazuje postotak točne riješenosti te provjere.



Slika 2.3. Prikaz analize završnog ispita iz nekog predmeta

2.2.2. Analitika distribucije rezultata ispita

S pomoću stupčastih i linijskih dijagrama, s opcijom mijenjanja između ta dva, nastavnici imaju u uvid pregled distribucije bodova studenata, s dodatnom opcijom pregleda po grupama (engl. *per class groups*). Ovi dijagrami omogućavaju nastavnicima uvid u distribuciju ocjena među svim studentima, identificiranje grupa s različitim razinama uspješnosti te analizu razlika u uspjehu među grupama.



Slika 2.4. Prikaz analize distribucije rezultata međuispita iz kolegija Napredni razvoj programske potpore za web

Iako ove opcije pružaju osnovni uvid u rezultate ispita, nastavnici bi imali velike koristi od detaljnije analize. Na primjer, izvlačenjem podataka s određenog kolegija iz baze podataka i analizom s pomoću R skripti moglo bi se generirati nekoliko ključnih statističkih pokazatelja:

1. **Indeks lakoće**: Ovaj indeks mjeri prosječnu težinu pitanja, odnosno koliko su pitanja bila laka ili teška za studente. Izračunava se kao prosječni postotak točnih odgovora na pitanje.
2. **Vrijednost slučajnog pogotka**: Procjenjuje vjerojatnost da student slučajno pogodi točan odgovor, što je posebno važno kod pitanja s višestrukim izborom (engl. *multiple choice questions*).

3. **Detaljnija fleksibilna analiza podskupina**: Omogućava nastavnicima da analiziraju rezultate specifičnih podskupina studenata, npr. prema demografskim podacima ili prethodnom uspjehu.
4. ***Cronbach alfa***: Ovaj pokazatelj mjeri unutarnju konzistentnost ispita, odnosno koliko su pitanja na ispitu koherentna i pouzdana. Viši *Cronbach alfa* indeks ukazuje na višu pouzdanost ispita.
5. **Analiza trendova kroz vrijeme**: Prati promjene u uspješnosti studenata tijekom semestra, što može pomoći u identifikaciji razdoblja kada su studenti pod većim stresom ili imaju poteškoće s određenim dijelovima gradiva.

Primjenom nekih od ovih pokazatelja, nastavnici bi mogli provesti detaljniju analizu kvalitete ispita, identificirati pitanja koja nisu adekvatna te unaprijediti strukturu i sadržaj ispita. Ovi podaci mogu poslužiti za kontinuirano poboljšanje kvalitete obrazovnog procesa, omogućujući precizniju i objektivniju evaluaciju studentskih postignuća [1].

2.3. Dubinska analiza podataka

Dubinska analiza podataka u robusnom (engl. *robust*) edukacijskom sustavu poput *Edgara* moglo bi unaprijediti analizu i praćenje studentskih aktivnosti te omogućiti nastavnicima dublji uvid u obrazovne trendove i ponašanje studenata. Ova metoda koristi različite tehnike strojnog učenja i statističke analize kako bi se iz velikih skupova podataka izvukli korisni obrasci i predikcije.

Primjena dubinske analize podataka mogla bi se provoditi kroz ove korake:

1. **Prikupljanje podataka**: Podaci bi se kontinuirano prikupljali iz baze podataka sustava *Edgar*, uključujući ocjene studenata, njihovo sudjelovanje u aktivnostima, vrijeme provedeno na rješavanju zadataka te rezultate ispita. Ovi podaci mogu biti strukturirani (npr. ocjene, brojevi bodova) i nestrukturirani (npr. tekstualni odgovori).
2. **Predobrada podataka**: Prikupljeni podaci bi se morali očistiti i pripremiti za analizu. Ovo uključuje uklanjanje nepotpunih ili nevažnih podataka, normalizaciju vrijednosti, te transformaciju podataka u odgovarajuće formate za analizu. Predobradom se osigurava da podaci budu dosljedni i spremni za analitičke procese.

3. **Analitički modeli**: Korištenjem različitih analitičkih modela i algoritama, podaci bi se analizirali kako bi se identificirali obrasci i trendovi. Na primjer, klasifikacijski algoritmi poput stabala odluke (engl. *decision trees*), slučajnih šuma (engl. *random forest*) i naivnog Bayesovog klasifikatora (engl. *naive Bayes classifier*) mogli bi se koristiti za predikciju akademskih postignuća studenata na temelju njihovih dosadašnjih rezultata i ponašanja.
4. **Vizualizacija podataka**: Rezultati analize mogli bi se vizualizirati kroz različite dijagrame. Ovi vizualni prikazi omogućuju nastavnicima jednostavan uvid u ključne informacije poput uspjeha studenata, učestalost određenih grešaka, te identifikaciju rizičnih skupina studenata koji bi mogli trebati dodatnu podršku.
5. **Generiranje izvještaja**: Automatizirani sustav mogao bi generirati redovite izvještaje koji bi se dostavljali nastavnicima, omogućujući im pravovremeno reagiranje na uočene probleme. Ovi izvještaji mogli bi uključivati analizu trendova kroz vrijeme, usporedbe između različitih skupina studenata, te preporuke za poboljšanje obrazovnih strategija.

Primjenom ovih koraka, dubinska analiza podataka mogla bi značajno unaprijediti efikasnost i kvalitetu obrazovnog procesa u *Edgaru*, omogućujući nastavnicima dublji uvid u studentske aktivnosti i bolje i informiranije odluke [4].

3. Potreba za automatiziranom analizom podataka

Kao napredni edukacijski sustav, *Edgar* svakodnevno pohranjuje i obrađuje ogromne količine podataka generiranih aktivnostima brojnih studenata i nastavnika. Kao što je već navedeno, ovi podaci obuhvaćaju širok spektar informacija - od rješavanja zadataka i laboratorijskih vježbi do rezultata ispita i različitih interakcija unutar obrazovnog procesa. Iako *Edgar* već pruža osnovne analize tih podataka, često su potrebne detaljnije i složenije analize koje postojeći sustav ne može u potpunosti podržati. Nastavnici se stoga suočavaju s izazovom ručnog dohvaćanja i obrade podataka, što je neefikasno i ponekad nesigurno.

Trenutne analize, premda su korisne, vrlo su jednostavne i pružaju osnovne povratne informacije. Kada nastavnici trebaju detaljniju i kompleksniju analizu, suočavaju se s nekoliko problema. Prije svega, mnogi nastavnici nemaju direktan pristup bazi podataka, što otežava dohvaćanje potrebnih podataka. Čak i oni koji imaju pristup često se susreću s izazovima vezanim za ručno izvođenje analiza. Ručno dohvaćanje podataka i izvođenje analiza zahtjeva određeno tehničko znanje, zna biti vremenski zahtjevno te je podložno greškama. Osim toga, direktan pristup bazi podataka može biti opasan zbog potencijalnog rizika od slučajnih, ili čak neovlaštenih izmjena ili brisanja podataka, što bi moglo narušiti integritet cijelog sustava.

Kako bi se ovi problemi riješili, predlaže se razvoj sustava koji bi omogućio nastavnicima da kreiraju i pohranjuju vlastite skripte za analizu podataka. Ovaj sustav bi omogućio stvaranje i pohranjivanje „cjevovoda“ (engl. *pipeline*)[1] koje bi automatski provodile analize nad najnovijim, uvijek ažurnim podacima. Taj prethodno opisani *pipeline* će se u ostatku rada nazivati „poslom“. Takav pristup bi imao brojne prednosti. Prvo, omogućio bi nastavnicima veću fleksibilnost i preciznost u analizi podataka, prilagođavajući analize specifičnim potrebama pojedinih kolegija ili istraživačkih projekata. Drugo, automatizirane analize bi značajno smanjile vrijeme potrebno za obradu podataka, omogućujući nastavnicima da se usmjere na druge važne zadatke, poput pripreme nastave i individualne podrške studentima. Još neke od prednosti ovog sustava su povećanje točnosti i dosljednosti u analizi podataka. Automatizirane analize smanjuju rizik od ljudske pogreške

[1] niz koraka za obradu podataka ili izvršavanje zadataka koji se automatski provode unaprijed definiranim redoslijedom

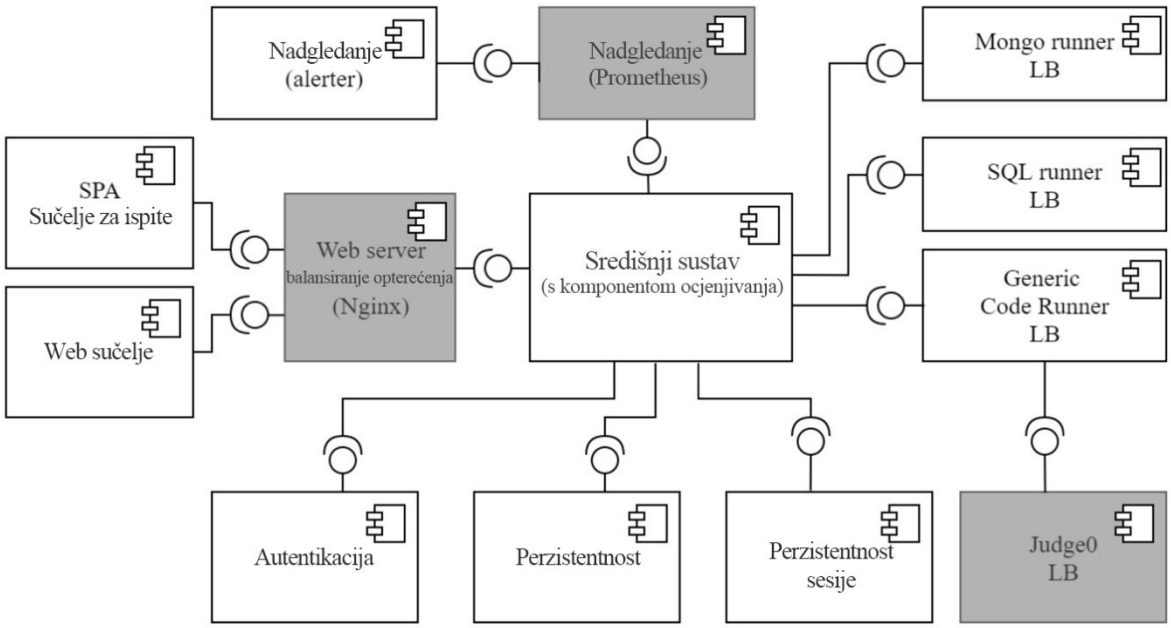
i osiguravaju da se svi podaci obrađuju prema istim kriterijima. Isto tako, omogućuju nastavnicima praćenje napretka studenata kroz vrijeme, identificiranje obrazovnih trendova i potencijalnih problema u ranoj fazi te prilagodbu nastavnih metoda prema potrebama studenata.

Jedan od primjera uspješne implementacije automatiziranih *pipeline* sustava može se pronaći u području znanstvenih istraživanja, gdje se takvi sustavi koriste za obradu velikih skupova podataka i izvođenje kompleksnih analiza. Na primjer, istraživanje o sustavu *„IntelliGenes“* pokazala je kako se primjenom naprednih algoritama strojnog učenja može značajno unaprijediti proces otkrivanja biomarkera, omogućujući istraživačima brzo i precizno analiziranje podataka te generiranje izvještaja. Slično, u području *fMRI* istraživanja, *WFU Pipeline* pruža automatiziranu obradu podataka, smanjujući potrebu za ručnim radom i povećavajući učinkovitost analize. Implementacija sličnog sustava u *Edgaru* mogla bi značajno unaprijediti obrazovni proces te pružiti nastavnicima moćan alat za analizu i interpretaciju podataka [5].

4. Arhitektura sustava *Edgar*

4.1. Cjelokupna arhitektura sustava

Sustav *Edgar* posjeduje modularnu[2] i skalabilnu arhitekturu, što omogućuje raspodjelu različitih dijelova sustava na više servera te horizontalnu skalabilnost. Napisan je u *Node.js-u* i može raditi na svim glavnim operacijskim sustavima. Njegova je arhitektura prikazana je na slici 4.1. Komponente označene s "LB" mogu se horizontalno skalirati te se s njima može balansirati opterećenje (engl. *load balancing*)[3], a komponente treće strane (engl. *third party*) označene su sivom bojom.



Slika 4.1. Prikaz modularne arhitekture sustava *Edgar* [6]

Edgar se sastoji od više opcionalnih i obaveznih modula. Temeljni sustav i sustav za pohranu podataka su obavezni moduli. Taj je temeljni sustav implementiran kao web-aplikacija, a može se skalirati horizontalno pomoću *Nginx load balancera* koji ravnomjerno

[2] podijeljenost sustava na nezavisne komponente koje se mogu zasebno razvijati, testirati i održavati

[3] metoda ravnomjerne distribucije mrežnog prometa preko skupa resursa koji podržavaju neku aplikaciju

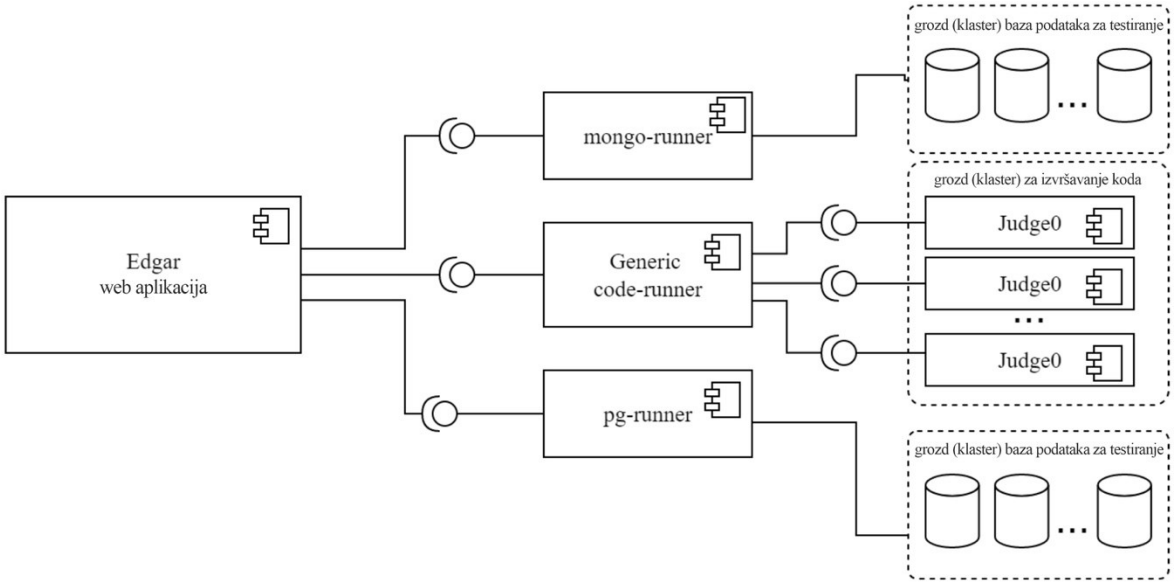
raspoređuje mrežni promet između više poslužitelja kako bi se osigurala visoka dostupnost i optimalna iskorištenost resursa. Za pohranu podataka *Edgar* koristi *PostgreSQL* za glavnu bazu podataka i *MongoDB* za privremene podatke sesije i razne događaje. Ova mu arhitektura omogućuje dinamičko prilagođavanje broja čvorova sustava i postavljanje novih verzija aplikacije bez prekida rada.

Prednji dio sustava uključuje jednostraničnu web-aplikaciju za studente koja im omogućuje pisanje ispita, domaćih zadaći, običnih i laboratorijskih vježbi, kao i klasičnu, višestraničnu web-aplikaciju za nastavnike i studente za druge zadatke. Obje aplikacije imaju moderan i responzivan *GUI* (engl. *graphical user interface*). Sustav autentifikacije podržava različite opcije, uključujući lokalnu autentifikaciju i autentifikaciju putem vanjskih pružatelja usluga identiteta koristeći *SAML* ili *OAuth2* protokol [6].

4.2. CodeRunner

Edgarov podsustav za evaluaciju programskih rješenja sastoji se od različitih "pokretača" (engl. *runners*) koji omogućuju izvršavanje koda u različitim programskim jezicima (slika 4.1.1.):

- ***SQL-runner***: Ovaj pokretač prihvaća SQL naredbe, izvršava ih na konfiguriranoj PostgreSQL bazi podataka koristeći definirano vremensko ograničenje i vraća skup zapisa ili poruku o grešci. Sve naredbe se izvršavaju unutar transakcije koja se uvijek poništava, stvarajući time sigurnu izoliranu okolinu za testiranje.
- ***Mongo-runner***: Ovaj pokretač prihvaća *MapReduce* i *find* upite, izvršava ih na *MongoDB* bazi podataka i vraća odgovarajući *JSON* (engl. *JavaScript Object Notation*) ili poruku o grešci. Zbog specifičnosti *MongoDB*-a, ovaj pokretač parsira primljene upite i koristi *Mongo API* za njihovo izvršavanje.
- **Generički *(engl. Generic) code-runner***: Ovaj pokretač prihvaća kod u skoro bilo kojem programskom jeziku (C, C#, C++, Java, Python, itd.) i niz ulaznih podataka, izvršava ih i vraća odgovarajući niz izlaznih podataka i grešaka, zajedno s metapodacima o procesu izvršavanja. Ovaj se pokretač oslanja na *Judge0 API* za izvršavanje koda.



Slika 4.2. Arhitektura podsustava *CodeRunner*: vanjski pokretači koda apstrahiraju programski jezik/paradigmu i balansiraju opterećenje [2]

Fokus ovog poglavlja je na *Generic code-runneru,* kojeg će se, zbog jednostavnosti, nadalje oslovljavati samo s *CodeRunner*.

CodeRunner omogućuje izvršavanje proizvoljnog, potencijalno malicioznog programskog koda u zaštićenoj okolini (engl. *sandbox*). On djeluje kao most između sustava *Edgar* i *„sandbox“* okoline, gdje se izvršava proizvoljni kod te se osigurava da potencijalno štetan kod ne može naštetiti sustavu. Omogućuje evaluaciju programskih zadataka u različitim programskim jezicima kao što su C, Python, JavaScript, PHP, Java i mnogi drugi. Osim što na ulaz prihvaća proizvoljan kod u jednom od brojnih podržanih jezika, *CodeRunner* prima i niz ulaznih podataka, izvršava ih te vraća odgovarajući niz izlaznih podataka i grešaka, zajedno s bogatim metapodacima o procesu izvršavanja.

Sustav koristi paradigme dinamičke analize za evaluaciju programskog koda, gdje se program tretira kao „crna kutija“ (engl. *black box*). Program se prevodi, izvršava i testira s različitim ulazima, a rezultati se uspoređuju s očekivanim izlazima. *Edgar* se oslanja na *Judge0 API* za sigurno izvršavanje koda. *Judge0* je besplatni i otvoreni API za izvršavanje i ocjenjivanje nepouzdanog izvornog koda. O njemu će se detaljnije pričati u sljedećem poglavlju [2].

4.3. Judge0

Kao što je već prethodno spomenuto, *Edgar* koristi *Judge0 API* za sigurno izvršavanje koda. *Judge0* je besplatni i otvoreni sustav za online izvršavanje koda koji je razvio student FER-a Herman Došilović. Od svog nastanka 2016. godine, postao je neizostavan alat za više od 3000 studenata FER-a, te je do lipnja 2020. obradio više od 8 milijuna programa širom svijeta [7]. Danas su ti brojevi značajno veći.

Jedan od ključnih aspekata sustava *Judge0* jest sigurnost. Zbog mogućnosti pokretanja proizvoljnog koda, svaki poslani kod smatra se nepouzdanim. Kako bi se osigurala sigurnost, Judge0 koristi *isolate*, dokazani mehanizam za obavljanje u izoliranoj okolini (engl. *sandboxing*) koji koristi značajke *Linux kernela* kao što su polja imena (engl. *namespaces*) i kontrolne grupe (engl. *control groups*). Ovaj pristup osigurava da se studentski kod prevodi i izvršava u izoliranoj okolini, čime se štiti sustav od potencijalno malicioznih aktivnosti.

Judge0 je dizajniran s naglaskom na skalabilnost. Ona se postiže s pomoću tri strategije: vertikalnog skaliranja (povećanje resursa i broja *CEE*-ova (engl. *Code Execution Engine*) na jednom računalu), horizontalnog skaliranja (replikacija *OCES*-a (engl. *Online Code Execution System*) na više računala s balansiranjem opterećenja) i kombiniranog skaliranja (skaliranje specifičnih dijelova sustava korištenjem *Docker* kontejnera). Ove strategije omogućuju efikasno rukovanje velikim brojem simultanih podnesaka, što je ključno za rad s velikim brojem studenata.

Judge0 nudi jednostavan *JSON web API* s dvije glavne pristupne točke:

- stvaranje novog podneska (engl. *submission*)
- dohvaćanje statusa podneska

Prva pristupna točka omogućuje slanje izvornog koda i identifikatora prevodioca/interpretera, dok druga vraća rezultate izvršenja kao što su izlaz, standardne pogreške, vrijeme izvršenja i korištenje memorije. Ovaj jednostavan pristup omogućuje lako integriranje ovog sustava u različite web aplikacije, uključujući e-learning platforme i online kompajlere.

Arhitektura sustava *Judge0* je modularna i skalabilna. Prvo, korisnik šalje zahtjev web API-ju za stvaranje novog podnošenja pružajući izvorni kod i željeni jezik. Web API zatim stvara novi zapis podnošenja u bazi podataka i dodaje ga u red. *CEE* kontinuirano preuzima sljedeće podnošenje iz reda, inicijalizira *„sandbox“* tj. izoliranu okolinu, priprema datoteke,

prevodi kod, izvršava ga, parsira metapodatke nakon izvršenja i ažurira zapis podnošenja u bazi podataka. Korisnik te rezultate može dohvatiti putem API-ja.

Ovaj sustav omogućuje asinkrono izvršavanje, gdje korisnik ne zna kada će podnošenje biti završeno, te sinkrono izvršavanje, gdje korisnik odmah dobiva rezultate nazad. Asinkrono izvršavanje pokazalo se korisnijim kada je broj podnošenja znatno veći od broja podnošenja koja se mogu istovremeno izvršiti [7].

5. Opis korištenih tehnologija

Prilikom izrade web-aplikacije *Edgar-Sync*, primarno je korišten programski jezik JavaScript, odnosno njegova nadogradnja TypeScript. JavaScript je najpopularniji programski jezik za razvoj web-aplikacija. Koristi se za stvaranje dinamičnih i interaktivnih web stranica, omogućujući bogate korisničke interakcije i ažuriranja u stvarnom vremenu bez potrebe za osvježavanjem stranice. TypeScript je nadskup JavaScripta koji uvodi statičko tipiziranje (engl. *static typing*), što omogućava bolju kontrolu nad kodom i ranije otkrivanje grešaka. Uz TypeScript, programeri mogu koristiti postojeći JavaScript kod dok postepeno dodaju tipove za poboljšanje čitljivosti i održavanja koda. TypeScript također podržava moderne JavaScript značajke, kao što su „async/await“, klase i moduli, čime se povećava produktivnost i kvaliteta koda [8].

NestJS je napredni *Node.js* okvir dizajniran za izradu skalabilnih i učinkovitih poslužiteljskih aplikacija. *Node.js* je platforma izgrađena na „*V8 JavaScript engine“* koja omogućava izvršavanje JavaScript koda izvan preglednika. *NestJS* omogućava razvoj aplikacija visokih performansi koristeći jednostavnu i intuitivnu arhitekturu. Razvijen koristeći TypeScript, on koristi modularnu arhitekturu koja omogućava organizaciju koda u samostalne module, što olakšava održavanje i proširivost aplikacija. Jedna od njegovih ključnih značajki je *„dependency injection“*, tehnika koja omogućava da jedan objekt pruži ovisnosti drugom objektu umjesto da ih on sam kreira, što poboljšava mogućnost testiranja i strukturu koda. *NestJS* također podržava različite vrste aplikacija, uključujući *REST API*-e (engl. *Representational State Transfer Application Programming Interfaces*), *GraphQL API*-e, mikroservise i aplikacije u stvarnom vremenu, čime se pokriva širok spektar poslužiteljskih potreba [9].

NPM (*Node Package Manager*) je alat za upravljanje paketima u JavaScriptu koji omogućava instalaciju, ažuriranje, verzioniranje i uklanjanje paketa. U svojem registru sadrži tisuće paketa razvijenih od strane globalne zajednice, od kojih su mnogi dostupni besplatno pod različitim licencama. Pri započinjanju novog projekta, *NPM* kreira datoteku *package.json* u glavnom direktoriju projekta koja bilježi sve korištene pakete i njihove verzije. Preuzeti paketi pohranjuju se u direktorij *node_modules*, a njihovo uključivanje u sustav za verzioniranje izvornog koda (engl. *source version control system*) nije potrebno jer se mogu ponovno preuzeti i instalirati na temelju zapisa iz *package.json* datoteke.

Swagger je alat za dizajniranje, dokumentiranje i testiranje „RESTful“ API-ja. Koristeći OpenAPI specifikaciju, Swagger omogućava automatsko generiranje interaktivne dokumentacije koja programerima olakšava razumijevanje i korištenje API-a. Uz *Swagger*, programeri mogu definirati krajnje točke API-a, metode, parametre i odgovore, što omogućava standardizaciju i bolju komunikaciju između razvojnih timova. *Swagger UI* pruža korisničko sučelje za testiranje API-a, omogućavajući jednostavno slanje zahtjeva i pregled odgovora.

Quasar je napredni okvir temeljen na *Vue.js* za izradu responzivnih korisničkih sučelja i progresivnih web-aplikacija (*PWA*). *Vue.js* je JavaScript okvir za izradu korisničkih sučelja koji omogućava brzu i reaktivnu izradu web aplikacija. *Quasar* omogućava razvoj aplikacija koje se mogu pokretati na različitim platformama, uključujući web, mobilne uređaje i desktop računala, koristeći jedinstveni skup izvornog koda. On također pruža bogat skup komponenti za brzo i učinkovito razvijanje aplikacija, uz podršku za moderne web-tehnologije kao što su *„lazy loading“*, *„tree shaking“*, i *„cache busting“*. Također, *Quasar CLI* (engl. *Command Line Interface*) omogućava jednostavno pokretanje projekata i upravljanje njima, čime se smanjuje vrijeme potrebno za postavljanje razvojnih okruženja [10].

PostgreSQL je napredni objektno-relacijski sustav za upravljanje bazama podataka (ORDBMS) poznat po svojoj stabilnosti i skalabilnosti. Pruža bogat skup funkcionalnosti, uključujući podršku za *ACID* (engl. *Atomicity, Consistency, Isolation, Durability*) transakcije, napredno zaključavanje, replikaciju i podršku za razne tipove podataka. *PostgreSQL* je idealan za izradu kompleksnih aplikacija koje zahtijevaju robusne baze podataka. Također, omogućava ekstenzibilnost putem proceduralnih jezika poput *PL/pgSQL*, što omogućava prilagodbu baza podataka specifičnim potrebama aplikacija. U ovom je projektu korišten *NPM* paket *Kysely* [11], koji je *„type-safe*“ graditelj SQL upita za TypeScript.

MongoDB je ne-relacijska, NoSQL baza podataka koja koristi fleksibilan, dokumentno-orijentirani model podataka. Umjesto pohranjivanja podataka u tablice, kao u relacijskim bazama podataka, MongoDB koristi kolekcije koje sadrže dokumente. Dokumenti se pohranjuju u *BSON* (engl. *Binary JSON*) formatu, što omogućava brzo izvođenje upita i jednostavno skaliranje. Ne zahtijeva eksplicitno definiranje sheme podataka, što omogućava veću fleksibilnost u radu s nestrukturiranim podacima. *MongoDB* posjeduje upitni jezik za osnovne radnje (engl. *CRUD, Create, Read, Update, Delete*) s mogućnostima filtriranja po

atributima dokumenata. Također, podržava horizontalno skaliranje s pomoću raslojavanja (engl. *sharding*) i replikacije, čime se osigurava visoka dostupnost i otpornost na greške. Ova baza podataka je idealna za aplikacije koje obrađuju velike količine nestrukturiranih podataka ili zahtijevaju fleksibilnu strukturu podataka. U sustavu se koristi za pohranu podataka o sjednici između poslužitelja i klijenta, pohranu trenutnih odgovora studenata te za vođenje dnevnika događaja na testovima, lekcijama i vježbama.

Mongoose je objektno-dokumentni „maper“ (*ODM*) za *MongoDB* i *Node.js*. Pruža strukturu i pojednostavljuje interakciju s *MongoDB*, omogućujući definiranje shema i validaciju podataka. Također omogućava programerima da koriste *MongoDB* na objektno-orijentirani način, olakšavajući rad s podacima kroz bogat skup funkcionalnosti za upravljanje podacima, uključujući *„query building“*, *„middleware“* i validaciju [12].

MinIO je visoko učinkovit sustav za pohranu objekata, kompatibilan s *Amazon S3 API*-em. Dizajniran je za pohranu velikih količina nestrukturiranih podataka kao što su slike, video zapisi i dnevnički zapisi (engl. *logs*). Pruža jednostavno postavljanje i visoku dostupnost, čime je idealan za aplikacije koje zahtijevaju robusno rješenje za pohranu podataka. Podržava horizontalno skaliranje i koristi tehnologije poput *„Erasure Coding“* za osiguravanje otpornosti na greške [13].

R je programski jezik i okruženje za statističku analizu i grafiku. Koristi se široko u znanstvenim istraživanjima, analizi podataka i strojnom učenju zbog svoje moćne funkcionalnosti za manipulaciju podacima i vizualizaciju. Podržava širok spektar statističkih i grafičkih tehnika, te ima bogat ekosustav paketa koji omogućavaju proširenje osnovnih funkcionalnosti za specifične potrebe analize. Osim toga, jezik *R* ima snažnu zajednicu korisnika i programera koji svakodnevno doprinose razvoju novih paketa i alata [14].

Docker je platforma za isporuku aplikacija koja koristi virtualizaciju na razini operacijskog sustava kako bi omogućila izolaciju aplikacija u kontejnere. Kontejneri su lagani, prijenosni i omogućuju sigurno okruženje za razvoj, testiranje i proizvodnju aplikacija. Omogućava pakiranje aplikacije i svih njezinih ovisnosti u jedan kontejner, što osigurava da aplikacija radi dosljedno bez obzira na okruženje u kojem se izvršava. Pruža alate za jednostavno upravljanje kontejnerima, uključujući njihovo pokretanje, zaustavljanje i skaliranje. *Docker* također podržava *docker-compose*, alat koji omogućava definiranje i upravljanje softverom koji se sastoji od više kontejnera [15].

6. Oblikovanje i implementacija rješenja

Korisnici sustava *Edgar*, kako studenti tako i nastavnici, često se suočavaju s potrebom za kompleksnijom i detaljnijom analizom podataka. Iako *Edgar* nudi osnovne mogućnosti analize, složenije potrebe često nadilaze njegove trenutne kapacitete, što rezultira potrebom za ručnim dohvaćanjem i obradom podataka, što je izrazito neefikasno i sklono pogreškama. Kako bi se prevladali ovi nedostaci i unaprijedio se proces analize podataka, razvijen je sustav *Edgar-Sync*. Kao što je već navedeno u uvodu, ovaj sustav koristi skripte u programskom jeziku R za automatiziranu analizu proizvoljnih podataka, omogućuje jednostavno kreiranje koraka za obradu podataka te pregled međurezultata i konačnih rezultata analiza. Vođenje detaljnog dnevnika izvođenja svih analiza osigurava transparentno praćenje procesa i rezultata te time ubrzava proces analize podataka korisnicima *Edgara*. U ovom će se poglavlju detaljno analizirati arhitektura ovog sustava, nadogradnje na podsustavima *Edgara* te pregledati sve funkcionalnosti koje sustav nudi.

6.1. Arhitektura sustava *Edgar-Sync*

Sustav *Edgar-Sync* temelji se na robusnoj i skalabilnoj arhitekturi koja omogućuje automatiziranu analizu podataka koristeći skripte napisane u programskom jeziku *R*. Njegova arhitektura sastoji se od nekoliko ključnih komponenti koje zajedno osiguravaju efikasno izvršavanje i praćenje analiza.

6.1.1. Pozadinski servis i API

Glavni dio ovog sustava čini pozadinski servis koji je implementiran koristeći *NestJS*. Ovaj servis sadrži nekoliko bitnih funkcionalnosti:

- **Logika za izvođenje posla (cjevovoda):** Ova komponenta omogućuje definiranje, pohranu i izvršavanje složenih analiza podataka. Korisnici mogu automatizirati proces analize kreiranjem poslova koji specificiraju korake potrebne za analizu, uključujući dohvaćanje podataka, njihovu obradu, analizu i generiranje rezultata. Logika poslova osigurava da se svi ovi koraci izvršavaju u ispravnom redoslijedu, te omogućuje praćenje i upravljanje svakim korakom procesa.

- **Spremanje i izvođenje zakazanih poslova:** Pozadinski servis upravlja zakazanim poslovima putem naprednog sustava za zakazivanje zadataka. Korisnici mogu definirati periodične zadatke koji se automatski izvršavaju prema određenom rasporedu, čime se osigurava redovito ažuriranje i analiza podataka bez potrebe za ručnom intervencijom. Ovaj sustav za zakazivanje podržava različite frekvencije izvršavanja, od svakodnevnih do mjesečnih zadataka, te omogućuje fleksibilno upravljanje resursima.
- **Evidencija (engl. logging):** Svaki izvršeni posao te njegovi koraci detaljno se evidentiraju, uključujući vremenske oznake (engl. *timestamps*) i druge relevantne podatke. Evidentiranje omogućuje transparentno praćenje svih aktivnosti unutar sustava, olakšava dijagnostiku i rješavanje problema te pruža povijesni zapis svih izvršenih analiza. Ova funkcionalnost je ključna za osiguranje integriteta podataka i održavanje visoke razine pouzdanosti sustava.
- **API komunikacija:** Pozadinski servis izlaže API koji omogućuje komunikaciju s prednjim dijelom (sučeljem) aplikacije. Putem ovog API-ja, klijenti mogu dobivati sve potrebne podatke u vezi poslova koji se izvršavaju na upit te zakazanih poslova i evidencija. API je dizajniran tako da podržava razne operacije poput stvaranja, ažuriranja, brisanja i dohvaćanja podataka (*CRUD* – engl. *Create, Read, Update, and Delete*), te omogućuje sučelju interakciju s pozadinskim servisom na siguran i efikasan način.

Pozadinski servis koristi *PostgreSQL* bazu podataka za pohranu definicija zakazanih poslova, što omogućuje strukturirano i pouzdano upravljanje podacima. Osim toga, koristi *MinIO* spremnik objekata za pohranu svih skripti, međurezultata i konačnih rezultata, kao i deskriptivnih evidencija vezanih uz izvršavanje (zakazanih) poslova tj. poslova koji se izvršavaju odmah te poslova koji se izvršavaju automatski u zadanim vremenskim intervalima.

6.1.2. Prednji dio (sučelje) aplikacije

Prednji dio sustava *Edgar-Sync* implementiran je koristeći *Quasar*, napredni okvir temeljen na *Vue.js*-u. Ova aplikacija pruža korisnicima intuitivno i responzivno korisničko sučelje koje omogućuje jednostavno kreiranje, upravljanje i pregledavanje poslova. Kroz sučelje aplikacije, korisnici mogu izvršavati poslove, zakazati nove, uređivati ih i brisati,

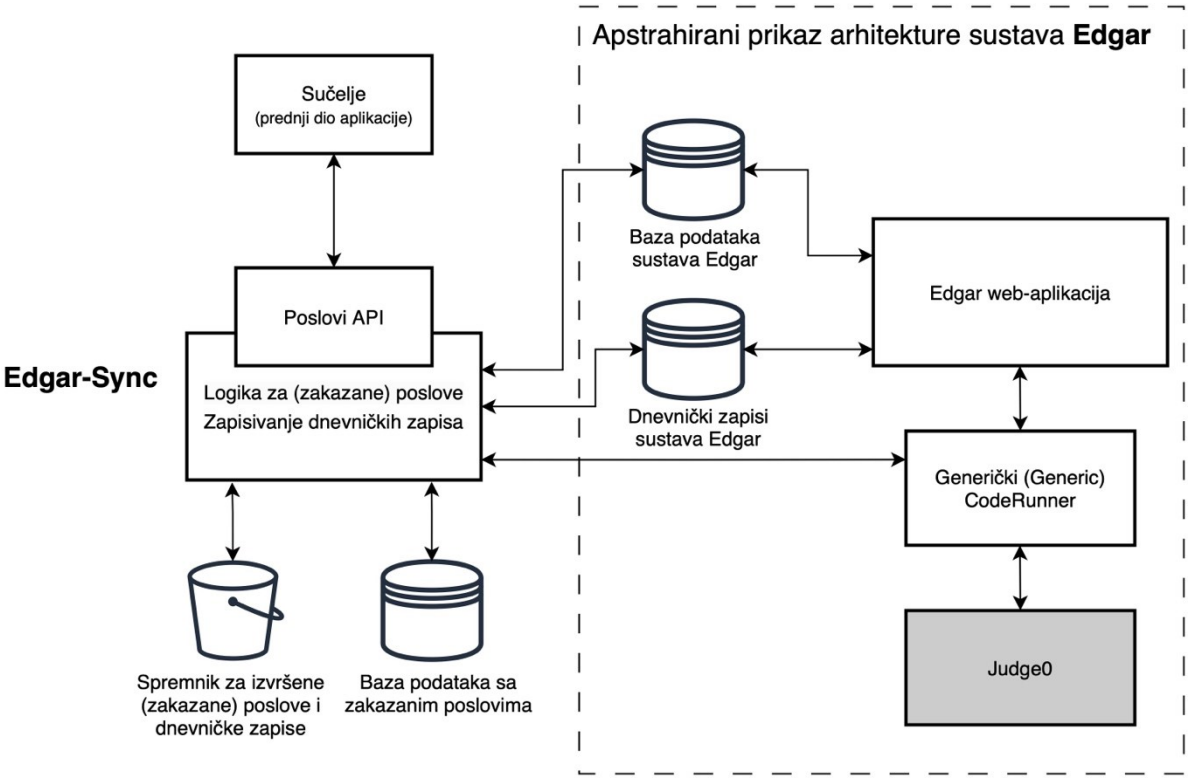
pregledavati međurezultate i konačne rezultate analiza te pratiti povijest izvršavanja zadataka.

6.1.3. Integracija sa sustavom *Edgar*

Sustav *Edgar-Sync* ima direktan pristup bazama podataka sustava *Edgar*, što omogućuje učinkovito korištenje postojećih podataka u analitičkim procesima. Konkretno, *Edgar-Sync* pristupa *Edgarovoj*:

- ***PostgreSQL* bazi**: U njoj se pohranjuju svi podaci vezani za studente, nastavnike, ispite, domaće zadaće i druge relevantne informacije.
- ***MongoDB* bazi**: Sadrži evidencije o instancama rješavanja ispita, informacije o sesijama (engl. *sessions*), napretku studenata u učenju određenih lekcija i druge dinamičke podatke.

Jedna od ključnih komponenti ovog sustava je njegova integracija s *Edgarovim* podsustavom *CodeRunner*. Tijekom izvođenja poslova na upit i zakazanih poslova, predefinirane *R* skripte i podaci izvučeni iz *Edgarovih* baza podataka šalju se sustavu CodeRunner. Ovaj sustav, u kombinaciji s *Judge0* API-jem, obrađuje te podatke i vraća rezultate natrag.



Slika 6.1. Prikaz arhitekture sustava *Edgar-Sync* koja uključuje i apstrahirani prikaz *Edgarove* arhitekture te međusobno dijeljenje resursa (podataka) i *CodeRunner* servisa

6.1.4. Spremanje definicija zakazanih poslova

Za pohranu definicija zakazanih poslova koje definiraju korisnici koristi se relacijska *PostgreSQL* baza podataka. Te se definicije pohranjuju u tablicu naziva *„scheduledJobs“*. Struktura ove tablice osmišljena je tako da omogućuje učinkovito upravljanje i praćenje zakazanih poslova.

Tablica 6.1. Atributi tablice *„scheduledJobs“*

Naziv	Tip	Opis
id	cijeli broj	Primarni ključ tablice, automatski se povećava za svaki novi zapis
uuid	jedinstvena identifikator (znakovni niz)	Jedinstveni identifikator zapisa, generira se automatski

name	znakovni niz	Naziv zakazanog posla
steps	JSONB format (znakovni niz)	Koraci potrebni za izvršenje posla, pohranjeni u JSONB formatu
cronjob	znakovni niz	Izraz koji definira učestalost izvršavanja posla, koristi *cron* sintaksu
email	znakovni niz	E-mail adresa korisnika povezanog s poslom, koristi se za obavijesti
created	vremenska oznaka bez vremenske zone	Vrijeme kreiranja zapisa, automatski se dodjeljuje pri stvaranju
lastmodified	vremenska oznaka bez vremenske zone	Vrijeme zadnje izmjene zapisa, automatski se ažurira pri svakoj promjeni entiteta tablice

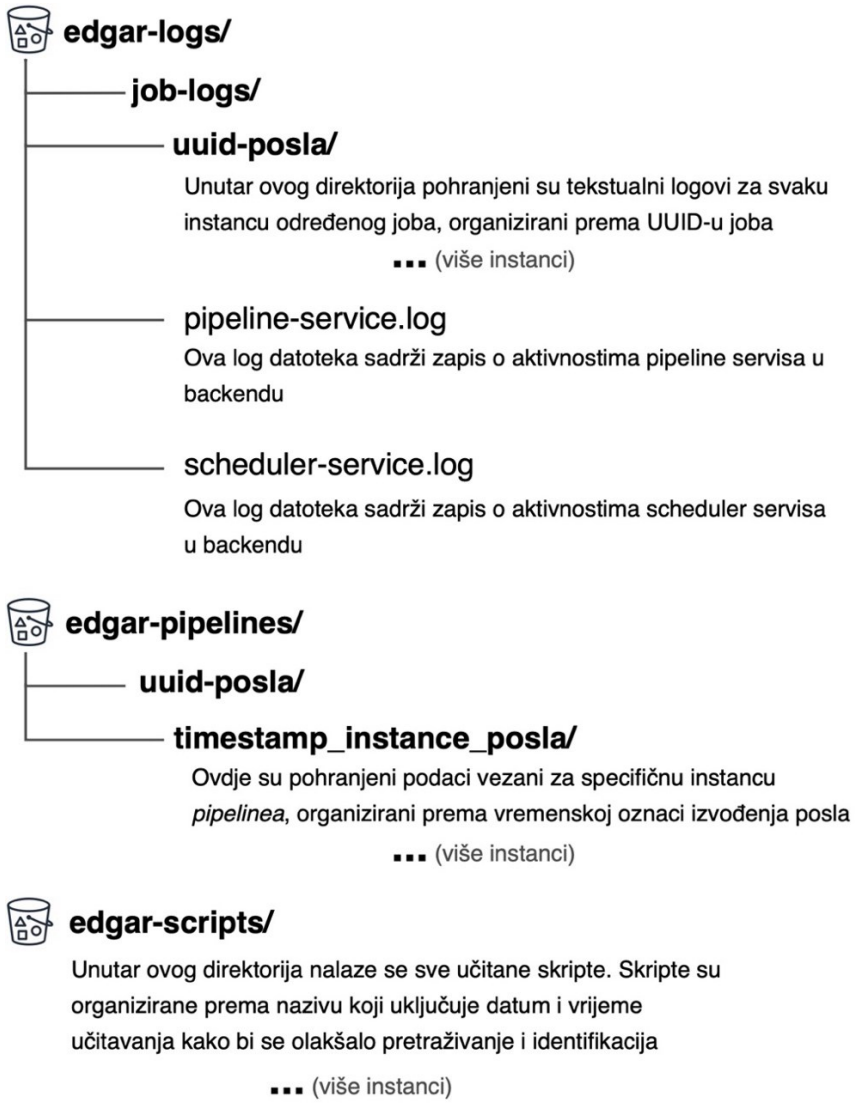
Svi korisnici sustava mogu nad ovom tablicom raditi *CRUD* operacije, što znači da mogu kreirati, ažurirati, brisati i dohvaćati zakazane poslove. Ovime se osigurava da svi relevantni podaci vezani za zakazane poslove budu centralizirano pohranjeni u bazi podataka, što omogućuje jednostavan pristup, učinkovitu obradu i analizu podataka.

Osim toga, mogućnost korištenja *cron* sintakse za definiranje učestalosti izvršavanja poslova omogućuje korisnicima precizno planiranje i automatizaciju svojih zadataka, čime se dodatno povećava učinkovitost rada sustava. Integracija sa sustavom za automatsko slanje e-mailova osigurava da korisnici budu pravovremeno obaviješteni o statusu svojih poslova.

6.1.5. Spremanje rezultata instanci poslova i evidencija njihovih izvršavanja

Ovaj sustav koristi *MinIO* spremnik (binarnih) objekata za pohranu svih podataka vezanih za svaku pojedinačnu instancu izvršavanja poslova odnosno zakazanih poslova.

MinIO je poznat po izuzetnoj brzini čitanja i pisanja podataka, što omogućuje učinkovitu obradu velikih količina podataka u stvarnom vremenu. Ovaj sustav visokih performansi osigurava da svi podaci budu brzo dostupni i spremni za analizu.



Slika 6.2. Struktura spremnika objekata koji služi za spremanje instanci izvršenih poslova i evidencija njihovih izvršavanja

Spremnik se sastoji od tzv. *bucketa*, koji služe za organiziranje i pohranu podataka. U ovom se sustavu koriste se tri glavna *bucketa*: „*edgar-logs“*, *„edgar-pipelines“* i *„edgar-scripts“*. U *bucketu „edgar-logs“* pohranjuju se svi dnevnički zapisi vezani uz izvršavanje poslova i aktivnosti servisa. *„edgar-pipelines“* sadrži podatke o instancama izvršenih poslova, organizirane prema vremenskim oznakama izvršenja, dok se u *„edgar-scripts“*

pohranjuju sve učitane skripte, organizirane prema datumu i vremenu učitavanja kako bi se olakšalo pretraživanje i identifikacija.

Ovaj se spremnik koristi za spremanje svih podataka vezanih za svaku pojedinačnu instancu izvršavanja posla, odnosno zakazanog posla. Na primjer, ako je posao zakazan da se izvršava svaki dan u ponoć, u direktorij *„job-logs“* unutar spremnika stvara se poddirektorij s imenom koje se sastoji od jedinstvenog identifikatora (*UUID*) tog posla. Unutar tog poddirektorija kreiraju se poddirektoriji s vremenskim oznakama kada je posao izvršen. Ti poddirektoriji sadrže podatke izvučene iz baze podataka, sve skripte vezane za taj posao te međurezultate i konačne rezultate izvršenja.

Za svaki zakazani posao koji se izvrši, sustav također pohranjuje detaljnu evidenciju u obliku dnevničkih zapisa koja sadrži opis obavljenih aktivnosti po koracima. Evidencija uključuje detaljan opis svakog koraka izvršenja posla, vremenske oznake kada je svaki korak započet i dovršen, te konačnu poruku o uspješnom obavljanju posla. U slučaju neuspjeha, dnevnički zapis sadrži detaljan opis greške koja se dogodila, uključujući točan korak gdje je došlo do greške.

```
2024-05-30_16:35:00 [SUCCESS] - PgGetStudentTestResults: Successfully executed query and uploaded test-
results-11810-155.csv to Minio
2024-05-30_16:35:00 [INFO] - ExecuteRScript: Script csv_test-2024-05-25T20-29-50.R read from Minio
2024-05-30_16:35:00 [INFO] - ExecuteRScript: Data file e0700b6b-77fd-4673-8b9c-f739e5930213/2024-05-30_16:35/db-
recordsets/test-results-11810-155-2024-05-30_16:35.csv read from Minio
2024-05-30_16:35:00 [INFO] - ExecuteRScript: Script copied to e0700b6b-77fd-4673-8b9c-f739e5930213/2024-05-30_
16:35/scripts/csv_test-2024-05-25T20-29-50.R
2024-05-30_16:35:00 [INFO] - ExecuteRScript: Data file e0700b6b-77fd-4673-8b9c-f739e5930213/2024-05-30_16:35/db-
recordsets/test-results-11810-155-2024-05-30_16:35_new.csv zipped and encoded
2024-05-30_16:35:01 [INFO] - ExecuteRScript: Request object created for CodeRunner
2024-05-30_16:35:03 [INFO] - ExecuteRScript: Script executed with status ID 14
2024-05-30_16:35:03 [INFO] - ExecuteRScript: Zip buffer decoded
2024-05-30_16:35:03 [SUCCESS] - ExecuteRScript: Successfully executed script and uploaded results
2024-05-30_16:35:03 [INFO] - ExecuteRScript: Script csv_test2-2024-05-25T20-29-52.R read from Minio
2024-05-30_16:35:03 [INFO] - ExecuteRScript: Data file e0700b6b-77fd-4673-8b9c-f739e5930213/2024-05-30_
16:35/results/file_modified-2024-05-30_16:35:03.csv read from Minio
2024-05-30_16:35:03 [INFO] - ExecuteRScript: Script copied to e0700b6b-77fd-4673-8b9c-f739e5930213/2024-05-30_
16:35/scripts/csv_test2-2024-05-25T20-29-52.R
2024-05-30_16:35:03 [INFO] - ExecuteRScript: Data file e0700b6b-77fd-4673-8b9c-f739e5930213/2024-05-30_
16:35/results/file_modified-2024-05-30_16:35:03_new.csv zipped and encoded
2024-05-30_16:35:03 [INFO] - ExecuteRScript: Request object created for CodeRunner
2024-05-30_16:35:06 [INFO] - ExecuteRScript: Script executed with status ID 14
2024-05-30_16:35:06 [INFO] - ExecuteRScript: Zip buffer decoded
2024-05-30_16:35:06 [SUCCESS] - ExecuteRScript: Successfully executed script and uploaded results
```

Slika 6.3. Primjer dnevničkih zapisa uspješnog izvršavanja zakazanog posla. Spremljeno u spremniku: *„edgar-logs/job-logs/e0700b6b-77fd-4673-8b9c-f739e5930213/2024-05-30_16:35.log“*

6.2. Nadogradnja sustava *Judge0* i *CodeRunner*

U svrhu unapređenja sustava *Edgar-Sync*, provedene su nadogradnje sustava *Judge0* i *CodeRunner*. Cilj ovih nadogradnji je omogućiti podršku za izvršavanje R i R Markdown skripti unutar okruženja Judge0, čime se proširuju analitičke mogućnosti sustava. U ovom se poglavlju detaljno opisuje postupak i rezultati provedenih nadogradnji.

Manipulacijom atributa *„run_cmd“* tablice *„languages“* u *Judge0* sustavu omogućeno je izvršavanje R skripti, čime se otvaraju brojne nove mogućnosti poput naprednih analiza, vizualizacije podataka i sl. Opisan je način na koji se može kreirati i upravljati direktorijima za pohranu dnevničkih zapisa i datoteka nastalih tijekom izvršavanja skripti. Taj se postupak može prilagoditi za različite potrebe analize podataka. Tablica *„languages“* proširena je dodavanjem dva nova retka za *R* i *R Markdown (RMD)* skripte. Međutim, umjesto korištenja *gzip* kompresije, odlučeno je da će se koristiti *zip* kompresija kako bi se olakšalo rukovanje komprimiranim datotekama.

Tablica 6.2. Novododana dva retka u tablici *„languages“* unutar sustava *Judge0*

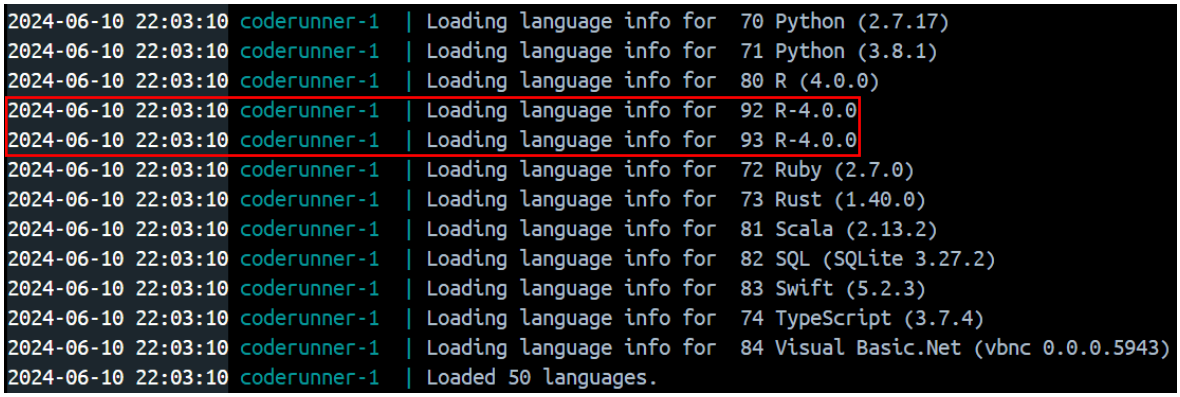
id	name	run_cmd	source_file

92	R-4.0.0.	`/usr/local/r-4.0.0/bin/R -e "`
		`dir.create('logs');`
		`dir.create('files');`
		`files_before = list.files('.');`
		`sink('logs/log.txt');`
		`library('base64enc');`
		`library('zip'); source('script.R');`
		`files_after = list.files('.');`
		`files_new =`
		`setdiff(files_after,files_before);`
		`files_new_path =`
		`paste('./files',files_new, sep='/');`
		`file.copy(files_new, files_new_path);`
		`sink();`
		`zip('source.zip', c('logs', 'files'));`
		`print(base64encode('source.zip'));`
		`"`
			script.R

93	R-4.0.0.	/usr/local/r-4.0.0/bin/R -e "
		dir.create('logs');
		dir.create('files');
		files_before = list.files('.');
		sink('logs/log.txt');
		library('base64enc');
		library('zip');
		rmarkdown::render('script.Rmd');
		files_after = list.files('.');
		files_new =
		setdiff(files_after,files_before);
		files_new_path =
		paste('./files',files_new, sep='/');
		file.copy(files_new,files_new_path);
		sink();
		zip('source.zip',c('logs', 'files'));
		print(base64encode('source.zip'));
		"
			script.Rmd

Ovim je proširenjem omogućeno izvršavanje *R* i *R Markdown (RMD)* skripti unutar *Judge0* sustava. Provedene nadogradnje uključuju kreiranje direktorija za pohranu dnevničkih zapisa i datoteka prije pokretanja skripti. Prvo se stvara popis datoteka u trenutnom direktoriju, nakon čega se pokreće predana *R* ili *R Markdown* skripta. Po završetku izvršavanja, uspoređuju se početni i završni popisi datoteka kako bi se identificirale novonastale datoteke. Te se datoteke zatim premještaju u posebne direktorije te komprimiraju i enkodiraju u *Base64* format [16].

Kako bi sustav *Edgar-Sync* mogao slati skripte ovog tipa na izvršavanje u *Judge0*, bilo je potrebno integrirati ove promjene s *CodeRunner* podsustavom. *CodeRunner* koristi *Judge0* za sigurno izvršavanje koda, te stoga mora biti „svjestan“ novih jezika i skripti koje može izvršavati. Prilikom svakog pokretanja kontejnera sustava *CodeRunner*, sustav povlači sve retke iz tablice *„languages“* u *Judge0* i ubacuje ih u svoju internu tablicu. Tako, *CodeRunner* prepoznaje sve jezike i pripadajuće skripte koje Judge0 može izvršavati. Tako, kada sustav *Edgar-Sync* šalje zahtjev za izvršavanje *R* ili *R Markdown* skripti, *CodeRunner* može pravilno interpretirati i proslijediti te zahtjeve *Judge0* sustavu.



Slika 6.5. Dio dnevničkih zapisa iz kontejnera *CodeRunner* sustava na kojima se vide automatsko učitana dva nova „jezika“

6.3. Definiranje niza koraka za obradu podataka

Glavna funkcionalnost sustava Edgar-Sync jest omogućiti korisnicima definiranje prilagođenih nizova koraka za obradu podataka. Uz to dolaze već prethodno navedene funkcionalnosti definiranja niza koraka obrade podataka uz pomoć skripti, spremanje rezultata u *MinIO* spremnik i pregledavanje istih, te opcionalno obavještavanje korisnika o izvršavanju posla e-mailom.

6.3.1. Primjer upotrebe funkcionalnosti koja kroz niz koraka obrađuje podatke

Nastavnik želi analizirati napredak studenata kroz različite semestre kako bi identificirao trendove u uspješnosti i prilagodio nastavne metode. To uključuje prikupljanje podataka o ocjenama studenata po semestrima, usporedbu njihovih rezultata te generiranje detaljnih izvještaja s vizualizacijama.

Prvi korak je dohvaćanje relevantnih podataka iz baze podataka sustava *Edgar*. To uključuje izvlačenje informacija o studentima, njihovim rezultatima i semestrima u kojima su polagali određene vježbe. Podaci se prikupljaju iz tablica koje sadrže informacije o studentima, vježbama i kolegijima kao što su *„exercise_student“*, *„course“*, *„student“* i sl.

Nakon prikupljanja podataka, slijedi obrada koja uključuje grupiranje podataka po semestru i studentu te izračun prosječnih ocjena po semestrima. Ova obrada omogućuje dobivanje uvida u napredak svakog studenta kroz različite semestre.

Sljedeći korak je vizualizacija podataka. Kreiraju se grafički prikazi koji ilustriraju trendove u uspješnosti studenata po semestrima. Vizualizacije pomažu nastavnicima da lako interpretiraju rezultate i uoče obrasce napretka. Ovaj i prošli korak ostvaruju se s R skriptama.

Na kraju, generira se detaljan izvještaj koji sadrži grafove i komentare te time omogućava nastavnicima da temeljito analiziraju rezultate. Izvještaj također pruža uvid u potencijalne probleme i olakšava buduću prilagodbu nastavnih metoda kako bi se poboljšali ishodi učenja. Ovaj se korak ostvaruje s *RMD* (*R Markdown*) skriptom koja generira *HTML* datoteku s jasnim i čitljivim prikazom podataka.

6.3.2. Izvršavanje posla na upit u stvarnom vremenu

Kad korisnik pristupi kartici s nazivom *Pipeline*, otvara se sučelje za kreiranje prilagođenog posla koji se izvršava odmah. Ova je opcija korisna korisniku kad želi odmah dobiti rezultate svojih predefiniranih koraka za analizu podataka ili kad prije definiranja zakazanog posla žele isprobati hoće li njihov posao raditi i hoće li vraćati ispravne podatke. Prvi korak u ovom procesu uvijek se sastoji od definiranja koraka (u obliku kartice) u kojem se određuje koji podaci se dohvaćaju iz baze. Trenutno, sustav nudi tri predefinirana parametrizirana upita:

- Rezultati ispita studenata (*„Students Exam Results“*): Ovaj upit dohvaća podatke o studentima i njihovim rezultatima na određenom ispitu, koristeći ID ispita *(„Test ID“*) i ID kolegija (*„Course ID“*) kao parametre.
- Studenti na kolegiju *(„Enrolled students“*): Ovaj upit prikuplja informacije o studentima upisanim na određeni kolegij u akademskoj godini, koristeći ID kolegija (*„Course ID“*) i akademsku godinu *(„Academic Year“*) kao parametre.
- Detaljni dnevnik ispita („*Exam log details*“): Ovaj upit dohvaća podatke o instancama ispita iz *MongoDB* baze podataka, koristeći ID instance ispita kao parametar.

Osim ova tri predefinirana upita, korisnici imaju mogućnost pisanja prilagođenih upita u *Edgarovu PostgreSQL* bazu podataka koristeći opciju *„Custom SQL Query“*. Ova funkcionalnost pruža značajnu fleksibilnost te omogućava korisnicima da dohvate podatke iz bilo koje tablice unutar baze te kreiraju kompleksne upite koji zadovoljavaju specifične potrebe analize. Na pozadinskom servisu implementirana je validacija SQL upita, koja automatski prepoznaje i prijavljuje pogreške u sintaksi upita. Ako korisnik unese neispravan SQL upit, sustav će generirati detaljan zapis o grešci u dnevničkim zapisima, objašnjavajući što nije u redu s upitom. Trenutno ne postoji direktna zaštita protiv potencijalno zlonamjernih radnji korisnika, kao što su izmjene ili brisanje podataka u bazi. Optimalno rješenje za ovaj problem bilo bi implementirati sigurnosne mjere na razini same baze podataka, gdje bi se definirali korisnici s ograničenim dopuštenjima kako bi se spriječile neovlaštene radnje.

Nakon definiranja početnog koraka, korisnik odabire format u kojem želi dohvatiti podatke iz baze (npr. CSV, JSON), što je bitno za sljedeće korake u obradi podataka.

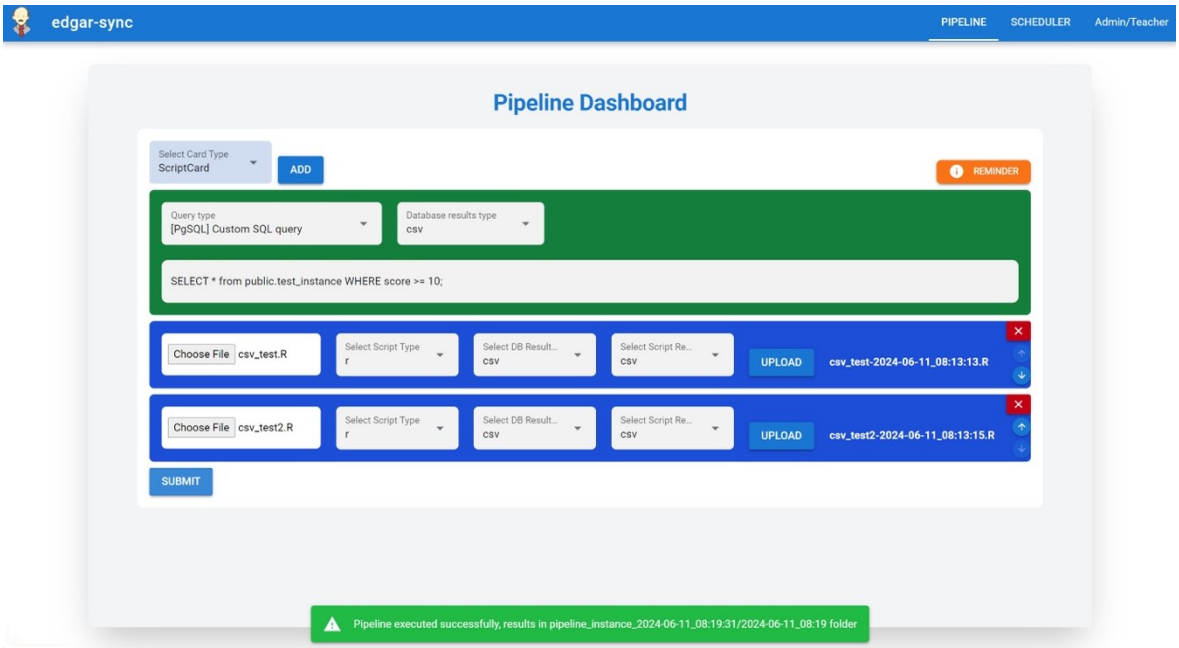
Sljedeći korak nakon definiranja upita za dohvaćanje podataka iz baze, jest specifikacija jednog ili više koraka za obradu tih podataka, koje nazivamo „skriptnim koracima“. Ovdje se definira:

- Učitavanje skripte za obradu podataka: Svaki „skriptni korak“ mora uključivati učitavanje skripte putem gumba *"Upload"*. To je ključni korak jer bez učitane skripte posao neće moći biti uspješno izvršen. Skripte mogu biti u formatu .R ili .Rmd.
- Odabir tipa podataka iz prethodnog koraka: U prvom „skriptnom koraku“ to se odnosi na tip podataka u kojima korisnik želi dobiti rezultate iz baze (CSV ili JSON). U sljedećim skriptnim koracima, ovo se odnosi na tip rezultata iz prethodnog skriptnog koraka. Ovaj odabir je također jako bitan jer osigurava da svaki korak ima ispravan ulazni format za obradu podataka.
- Odabir formata rezultata koje skripta generira: Svaki „skriptni korak“ mora specificirati format rezultata koje skripta daje (JSON, CSV, HTML). To omogućuje sljedećem koraku da pravilno interpretira i obradi te rezultate. Ovo je posebno važno kada se definira niz uzastopnih skripti jer svaki korak mora biti kompatibilan s izlaznim formatom prethodnog koraka.

Ti „skriptni koraci“, tj. koraci koji definiraju obradu podataka skriptama, mogu se mijenjati, reorganizirati i brisati prema potrebi.

Na kraju, klikom na gumb *"Submit"*, korisnik biva upozoren s porukama koje ga podsjećaju da provjeri jesu li svi rezultati iz baze podataka, rezultati skripti i tipovi skripti ispravni te jesu li sve skripte učitane. Korisnik tada može odustati od podnošenja ili podnijeti zahtjev za izvršavanje posla.

Nakon što korisnik podnese taj zahtjev, sustav generira obavijest o njegovom uspješnom izvršenju. Korisnik tada može otići u određeni direktorij unutar MinIO spremnika gdje može pronaći rezultate njegovog izvršavanja te pregledati sve relevantne dnevničke zapise kako bi dobio uvid u detalje procesa obrade.



Slika 6.6. Prikaz konfiguracije posla koji se izvodi samo na upit te obavijest o uspješnom izvršavanju i lokaciji gdje se nalaze rezultati izvršavanja. Korišten *„Custom SQL Query“* i dva „skriptna koraka“ s dvije različite ulančane *R* skripte.

6.3.3. Kreiranje novog zakazanog posla

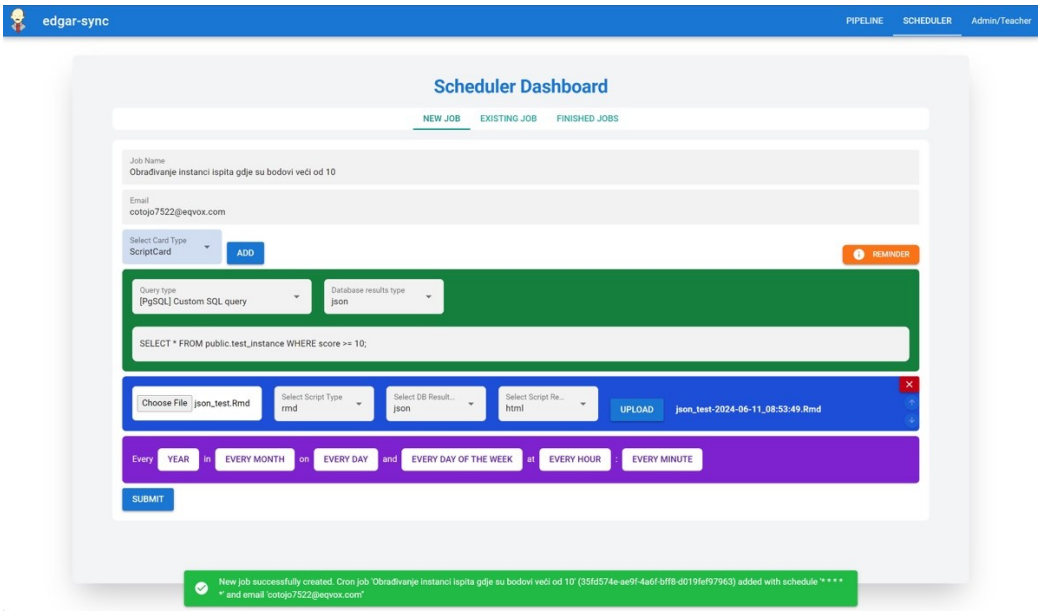
Ova je funkcionalnost proširenje prethodno opisanih mogućnosti sustava *Edgar-Sync*. Zakazani posao je unaprijed definirani niz koraka za obradu podataka kojemu se dodaje vremenska komponenta za automatsko izvršavanje u određenim intervalima. Osim

koraka za dohvaćanje podataka iz *Edgarovih* baza podataka i definiranja „skriptnih koraka“, prilikom kreiranja novog zakazanog posla potrebno je (ili je opcionalno) unijeti još nekoliko podataka:

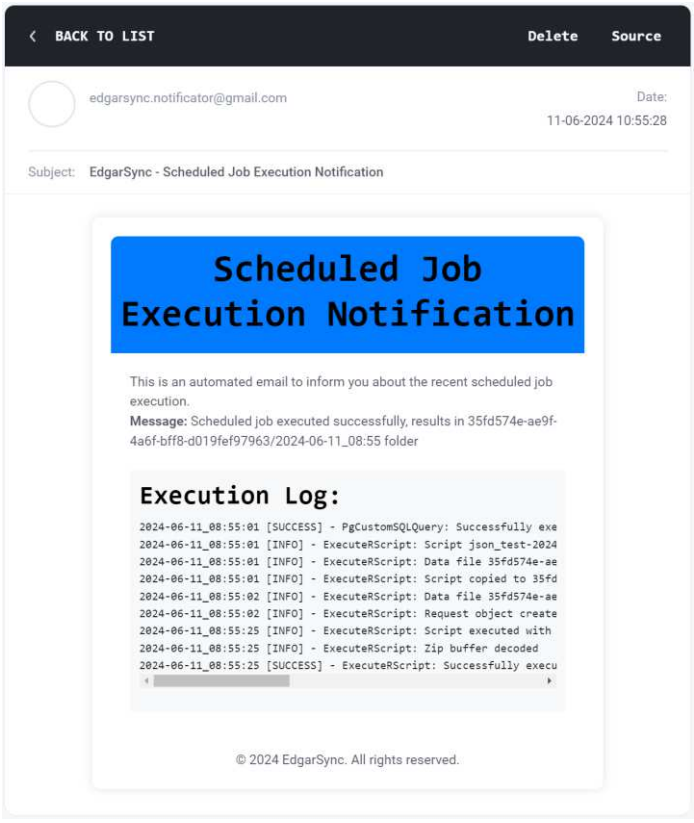
- Ime posla (*„Job Name“*): Ovdje korisnik unosi proizvoljno ime posla po kojem će ga kasnije moći identificirati. Ime posla olakšava organizaciju i upravljanje različitim zadacima unutar sustava.
- E-mail adresa: Ovo je opcionalno polje u koje korisnik može unijeti svoju e-mail adresu. Ako se adresa unese, korisnik će biti obaviješten svaki put kada se posao izvrši, bilo to uspješno ili neuspješno. U e-mail poruci korisnik dobiva dnevničke zapise te lokaciju rezultata u spremniku *MinIO*.
- *cron* izraz (*„cron expression“*): *cron* izraz je niz znakova koji definiraju intervale izvršavanja zakazanog posla. Ovi se izrazi koriste za zakazivanje ponovljenih zadataka u točno određenim vremenskim intervalima. U kontekstu ovog sustava, *Cron* izraz omogućuje precizno definiranje vremena i frekvencije izvršavanja zakazanih poslova. Primjeri ovakvih izraza uključuju svakodnevne zadatke, zadatke koji se izvršavaju svaki sat, svaki tjedan, ili čak svaku minutu, ovisno o potrebama korisnika. Na primjer, ovo može biti korisno kad se podaci u nekoj tablici svakodnevno ažuriraju, a korisnik za *cron* izraz postavi „(0 0 * * *)“, što znači da se posao izvršava svaki dan u ponoć (00:00). Kreiranjem zakazanog posla s ovom postavkom, korisnik će svaki dan moći vidjeti rezultate analize koja se obavila u ponoć.

Nakon što se unesu svi potrebni podaci, korisniku se klikom na gumb *"Submit"* otvara upozorenje sa sličnom porukom kao i u prošloj navedenoj funkcionalnosti. Nakon potvrde, posao se kreira i sprema u *PostgreSQL* bazu podataka sustava *Edgar-Sync* u tablicu *scheduledJobs*. Svaki put kada se sustav pokrene ili resetira, svi zapisi iz te tablice automatski se učitavaju u aplikaciju i aktiviraju kako bi se nastavili izvršavati prema definiranim intervalima.

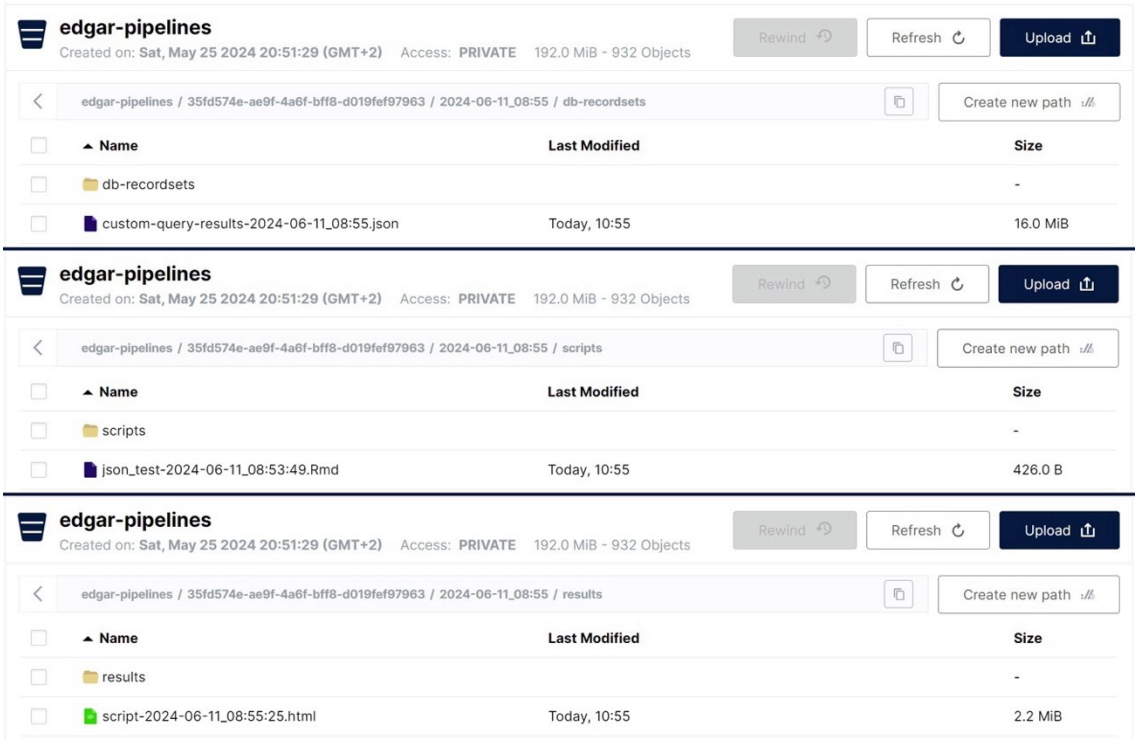
Ovaj upravo kreirani posao kasnije korisnik može urediti ili obrisati, ali o toj funkcionalnosti će biti riječi u sljedećem poglavlju. Kreiranjem zakazanih poslova, korisnici mogu automatizirati redovite zadatke analize podataka, što značajno poboljšava efikasnost rada i smanjuje potrebu za ručnom intervencijom.



Slika 6.7. Prikaz uspješno kreiranog zakazanog posla s imenom „Obrađivanje instanci ispita gdje su bodovi veći od 10“, postavljenom e-mail adresom te *cron* izrazom na izvršavanje posla svake minute.



Slika 6.8. E-mail poruka s lokacijom rezultata izvršavanja zakazanog posla i njegovim dnevničkim zapisima



Slika 6.9. Prikaz spremljenih podataka za izvršavanje posla iz MinIO spremnika

6.4. Uređivanje i brisanje zakazanih poslova

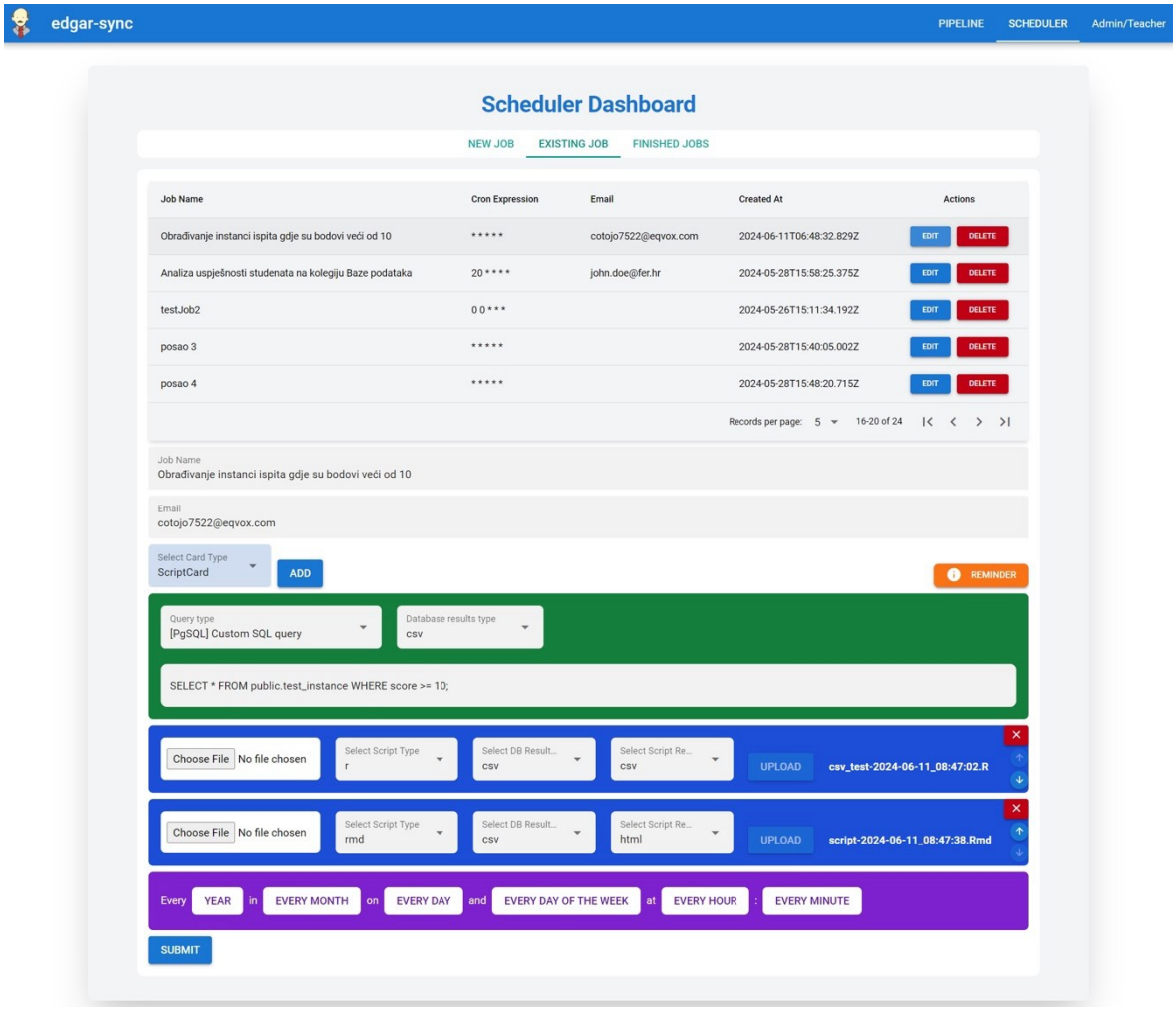
Odabirom kartice *Existing Job* na *Scheduler Dashboardu*, prikazuje se tablica svih postojećih zakazanih poslova s mogućnošću uređivanja i brisanja istih.

Klikom na gumb *Edit* pored određenog zakazanog posla, otvara se komponenta koja služi za kreiranje poslova, ali sada ispunjena već prethodno unesenim definiranim podacima za taj posao. Korisnici mogu mijenjati bilo koji aspekt posla, uključujući naziv posla, e-mail adresu za obavijesti, *cron* izraz za raspored izvršavanja, te korake unutar posla. Nakon unosa izmjena, klikom na gumb *Submit*, zakazani posao se ažurira u sustavu, a korisnik dobiva odgovarajuću poruku o uspješnom uređivanju.

Ako korisnik želi trajno ukloniti neki zakazani posao, može to učiniti klikom na gumb *Delete*. Ova akcija ne samo da uklanja posao iz *schedulera*, što znači da se više nikada neće izvršiti, već ga također briše iz baze podataka, čime se osigurava da su svi povezani podaci trajno uklonjeni.

Važnost ove funkcionalnosti leži u njenoj sposobnosti da korisnicima omogući upravljanje zakazanim poslovima. Naime, tijekom korištenja sustava može doći do promjena

u potrebama korisnika, kao što su promjene u učestalosti izvršavanja analiza, dodavanje novih koraka unutar postojećih poslova ili ažuriranje kontakt informacija. Mogućnost jednostavnog uređivanja i brisanja poslova omogućava korisnicima da brzo reagiraju na te promjene bez potrebe za kreiranjem novih poslova od početka. Na primjer, ako korisnik primijeti da neki zakazani posao generira neprecizne rezultate zbog promjene u strukturi podataka, može jednostavno urediti njegove korake kako bi prilagodio analizu novim uvjetima. S druge strane, ako neki posao postane suvišan ili nepotreban, može ga jednostavno obrisati.



Slika 6.10. Prikaz tablice postojećih zakazanih poslova te uređivanja prethodno kreiranog posla

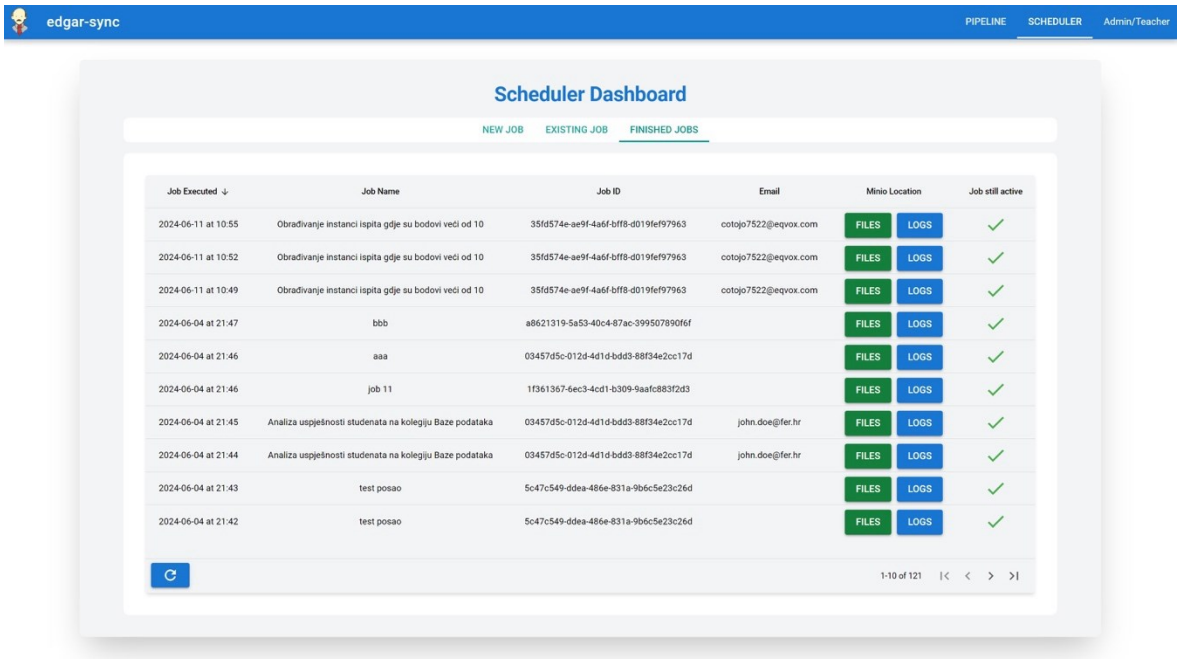
6.5. Dnevnik obavljanja poslova

Posljednja funkcionalnost koju nudi sustav *Edgar-Sync* jest dnevnik obavljanja poslova, odnosno kontrolna ploča prikaza svih instanci izvršenja zakazanih poslova.

Odabirom kartice *Finished Jobs* na *Scheduler Dashboardu*, prikazuje se tablica s popisom svih obavljenih poslova. Tablica je zadano (engl. *default*) sortirana od najnovijih prema najstarijima. Ova tablica sadrži nekoliko ključnih informacija za svaku instancu izvršenog posla. Prikazano je vrijeme kada je posao izvršen, naziv posla, jedinstveni identifikator (*UUID*) posla, e-mail adresa korisnika (ako je unesena), te informacija je li posao još uvijek aktivan. Uz to, korisnici imaju pristup dvama gumbima - *Files* i *Logs.* Klikom na gumb *Files*, otvara se nova kartica u web-pregledniku koja vodi do točne lokacije svih datoteka vezanih za tu instancu posla u *MinIO* spremniku. Na isti način, klikom na gumb *Logs*, korisnici mogu pregledati detaljne zapise svih aktivnosti povezanih s izvršenjem posla.

Tablica također omogućuje sortiranje podataka prema bilo kojem stupcu, bilo silazno, bilo uzlazno. Ovo je osobito korisno kada korisnici žele kategorizirani prikaz izvršenih poslova, primjerice, prikaz poslova grupiran po e-mail adresama. Uz mogućnosti sortiranja, tablica sadrži i gumb za osvježavanje podataka. Ovaj gumb dohvaća najnovije podatke o instancama izvršenih poslova u stvarnom vremenu, čime se osigurava da korisnici uvijek imaju pristup najnovijim informacijama. Ova funkcionalnost je posebno korisna kada se između dva dohvaćanja podataka obave novi poslovi, omogućujući korisnicima da odmah vide rezultate bez potrebe za ručnim osvježavanjem stranice.

Važnost dnevnika obavljanja poslova leži u njegovoj sposobnosti da korisnicima pruži detaljan uvid u povijest svih izvršenih aktivnosti unutar sustava. Transparentnost podataka omogućuje praćenje svih izvršenja, olakšava dijagnostiku eventualnih problema te pruža povijesni zapis svih aktivnosti.



Slika 6.11. Prikaz tablice dnevnika poslova. Sortirana od najnovijih do najstarijih izvršenja poslova

7. Zaključak

Kad bi ušao u stalnu uporabu, *Edgar-Sync* bi zasigurno olakšao analizu podataka za nastavnike i asistente. On podržava automatizaciju procesa analize podataka, ulančano izvođenje složenih analitičkih skripti te transparentno praćenje rezultata. Ovaj sustav omogućava nastavnicima da se fokusiraju na obrazovni proces bez potrebe za ručnim izvršavanjem analitičkih zadataka, što rezultira bržim i učinkovitijim procesom obrade podataka. Također, sustav vodi detaljan dnevnik svih aktivnosti, čime se osigurava transparentnost i mogućnost povratne analize.

Iako je trenutna implementacija funkcionalna i obuhvaća ključne aspekte automatizacije procesa analize podataka, definitivno ima prostora za daljnja poboljšanja. Prvo i najvažnije, potrebno je omogućiti korisnicima potpunu nezavisnost od MinIO spremnika, integriranjem funkcionalnosti za direktno dohvaćanje datoteka i dnevničkih zapisa unutar aplikacije. Nadalje, integracija Python skripti mogla bi unaprijediti analitičke mogućnosti zbog široke upotrebe i fleksibilnosti Pythona. Uvođenje autentifikacije putem *AAI@EduHr*[4] povećalo bi sigurnost i kontrolu pristupa sustavu, osiguravajući da samo ovlašteni korisnici mogu pristupati osjetljivim podacima. Osim navedenih poboljšanja, preporučljivo je razmotriti razvoj dodatnih funkcionalnosti koje bi unaprijedile korisničko iskustvo i efikasnost sustava. Na primjer, sustav za generiranje prilagodljivih izvještaja, automatska analiza kvalitete podataka i detekcija anomalija itd. Integracija dodatnih analitičkih alata i metoda, kao što su napredni statistički modeli i strojno učenje, mogla bi pružiti dublje uvide u podatke i podržati donošenje informiranih odluka.

Konačno, *Edgar-Sync* mogao bi poslužiti kao vrijedan alat za nastavnike u analizi podataka iz *Edgara*, no daljnjim razvojem i unapređenjima može se postići još veća efikasnost i korisnost. Nastavnici bi tada mogli u potpunosti iskoristiti potencijal svojih podataka i unaprijediti kvalitetu obrazovnog procesa na temelju detaljnih i preciznih analiza.

[4] sustav koji omogućuje sigurno, pouzdano i efikasno upravljanje elektroničkim identitetima za pristup mrežnim resursima u znanosti i visokom obrazovanju u Hrvatskoj

Literatura

[1] Marija Kompar, *Statistički pokazatelji i vizualizacija uspješnosti ispita u Edgaru,* 2023.

[2] Igor Mekterović, Ljiljana Brkić, Marko Horvat, *Scaling Automated Programming Assessment Systems,* 2023.

[3] Ljiljana Brkić, Igor Mekterović , Melita Fertalj, Darko Mekterović, *Peer assessment methodology of open-ended assignments: Insights from a two-year case study within a university course using novel open source system,* 2024.

[4] C. Romero, S. Ventura, *Educational data mining and learning analytics: An updated survey,* 2024., arXiv:2402.07956 [cs.HC]

[5] William DeGroat, Dinesh Mendhe, Atharva Bhusari, Habiba Abdelhalim, Saman Zeeshan, Zeeshan Ahmed, *IntelliGenes: a novel machine learning pipeline for biomarker discovery and predictive analysis using multi-genomic profiles*, Bioinformatics, Volume 39, Issue 12, December 2023, btad755, poveznica: https://doi.org/10.1093/bioinformatics/btad755

[6] Igor Mekterovic, Ljiljana Brkic, Boris Milašinovic, (Member, IEEE), Mirta Baranovic, (Member, IEEE), *Building a Comprehensive Automated Programming Assessment System*, 2020.

[7] Herman Zvonimir Došilović, Igor Mekterović, *Robust and Scalable Online Code Execution System*, 2020.

[8] TypeScript, *What is TypeScript?,* poveznica: https://www.typescriptlang.org

[9] NodeJS, *NodeJS Documentation – Introduction and Philosophy*, poveznica: https://docs.nestjs.com

[10] Quasar, poveznica: https://quasar.dev

[11] Kysely, *Kysely Docs*, poveznica: https://kysely.dev/docs/intro

[12] Jesse Hall, *Getting Started with MongoDB & Mongoose*, 2022., poveznica: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/

[13] MinIO, Documentation: *MinIO Object Storage for Linux*, poveznica: https://min.io/docs/minio/linux/index.html#

[14] R, *What is R?*, poveznica: https://www.r-project.org/about.html

[15] Docker, *Docker overview*, poveznica: https://docs.docker.com/guides/docker-overview/

[16] Petar Benjak, *Prototip integracije sustava za strojno učenje u sustav Edgar*, 2022.

Sažetak

Modul za obavljanje korisnički definiranih programa u sustavu za automatsko ocjenjivanje programskog kôda Edgar

Sustav *Edgar*, koji koriste studenti i nastavnici, često zahtijeva napredniju i detaljniju analizu podataka od one koju trenutno nudi. Kako bi se riješio problem ručnog dohvaćanja i obrade podataka, razvijen je *Edgar-Sync*. On automatizira analizu podataka dohvaćajući podatke iz *Edgarove* baze te pružajući mogućnost ulančavanja proizvoljnog broja R skripti koje te podatke obrađuju, omogućujući veću fleksibilnost i prilagodljivost korisnicima. Ovaj sustav omogućuje jednostavno kreiranje, pregled i upravljanje zakazanim poslovima te poslovima koji se izvode na upit koji se sastoje od tih koraka te njihovo detaljno praćenje putem dnevnika izvršavanja. Time se poboljšava učinkovitost i preciznost analize podataka, pružajući korisnicima alat za unapređenje obrazovnog procesa i donošenje informiranih odluka temeljenih na pouzdanim podacima.

Ključne riječi: analiza, obrada, podaci, zakazani posao, cjevovod, baza podataka, dnevnik izvršavanja, evidencija, CodeRunner, Judge0

Abstract

Module for executing user-defined programs in Edgar automated programming assessment system

Edgar system, used by both students and teachers, often requires more advanced and detailed data analysis than it currently offers. To address the issue of manual data retrieval and processing, *Edgar-Sync* was developed. It automates data analysis by fetching data from the *Edgar* database and providing the ability to chain an arbitrary number of R scripts to process this data, thus offering greater flexibility and adaptability for users. This system allows for easy creation, review, and management of scheduled jobs as well as on-demand jobs consisting of these steps, and their detailed tracking through execution logs. This improves the efficiency and accuracy of data analysis, providing users with a tool to enhance the educational process and make informed decisions based on reliable data.

Keywords: analysis, processing, data, scheduled job, pipeline, database, execution log, records, CodeRunner, Judge0

Popis oznaka i kratica

FER	Fakultet elektrotehnike i računarstva
APAS	Automated Programming Assessment System
SQL	Structured Query Language
CSV	Comma Separated Values
JSON	JavaScript Object Notation
PA	Peer Assessment
IRT	Item Response Theory
API	Application Programming Interface
REST	Representational State Transfer
ID	Identification
UUID	Universally Unique Identifier
HTML	HyperText Markup Language