SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 902

OPĆENAMJENSKI SERVIS ZA DETEKCIJU PLAGIJATA

Ivan Bilobrk

Zagreb, srpanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 902

OPĆENAMJENSKI SERVIS ZA DETEKCIJU PLAGIJATA

Ivan Bilobrk

Zagreb, srpanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

DIPLOMSKI ZADATAK br. 902

Pristupnik:	**Ivan Bilobrk (0036531335)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	prof. dr. sc. Igor Mekterović

Zadatak: **Općenamjenski servis za detekciju plagijata**

Opis zadatka:

Edgar je informacijski sustav koji omogućuje provođenje testova u kojima studenti predaju programski kod u različitim programskim jezicima (Java, C, SQL, itd.), ali i u formi slobodnog teksta. Programski kod se automatski pokreće, provjerava i ocjenjuje. Studenti tipično pišu u ograničenim vremenskim intervalima, ne uvijek pod fizičkim nadzorom i ponekad dolazi do slučajeva preuzimanja tuđih rješenja. Edgar se koristi na različitim predmetima i razina i opseg traženog programskog koda varira te su negdje očekivane velike sličnosti autentičnog koda, a negdje nisu i zato je potrebno razviti sustav detekcije prepisivanja koji će biti podesiv kako bi se mogao prilagoditi svakom predmetu, a eventualno čak i testu ili pitanju. Napraviti pregled pristupa i alata za detekciju plagijata s naglaskom na detekciju plagijata programskog koda. Osmisliti i napraviti pozadinski servis koji u zakazanom vremenu pretražuje definirane skupove podataka u potrazi za plagijatima. Servis mora biti podesiv kako bi mogao raditi s različitim podatcima i algoritmima. Osim podataka pohranjenih u Edgaru (studentski ispiti i odgovarajući ispiti u prethodnim akademskim godinama) potrebno je omogućiti i dodatne vanjske izvore podatak kao što su git repozitoriji ili eksplicitno zadane kolekcije. Sustav treba izložiti jednostavno programsko sučelje (API) putem kojega je moguće definirati poslove detekcije kao i preuzeti rezultate. Napraviti pokaznu integraciju s web-aplikacijom gdje će korisnik moći putem grafičkog sučelja definirati poslove i pregledati rezultate. Demonstrirati rad sustava na podatcima iz Edgara u više akademskih godina u kombinaciji s vanjskim podatcima. Razmotriti mogućnosti vizualizacije podataka o sličnosti studentskih rješenja. Razmotriti i komentirati utjecaj velikih jezičnih modela na domenu detekcije plagijata. Donijeti ocjenu ostvarenog pristupa te smjernice za budući razvoj.

Rok za predaju rada: 4. srpnja 2025.

Sadržaj

Uvod

Sve veći broj studenata koji pohađaju predmete računarstva, kao i smanjenje broja osoblja potaklo je razvoj računalnih sustava koji olakšavaju pisanje provjera znanja te automatiziraju njihovo ispravljanje. Ovakvi sustavi pisanje provjera znanja podižu na novu razinu za nastavnike i studente. Nastavnici se više ne moraju brinuti o podjeli ispitnih papira te prikupljanju rješenja. Ispitni materijali dostupni su u bilo kojem trenutku uz pristup internetu. Tradicionalni ispiti u papirnatom obliku podložni su gubljenju, oštećenju, a nerijetko su studentska rješenja napisana na teško čitljiv način. Studentima ovi sustavi pomažu za vrijeme ispita gdje se očekuju opsežni tekstualni odgovori te naročito tamo gdje je potrebno napisati isječak programskog koda strukturiranog na prikladan način. Naime, mogućnosti kao što su brisanje, kopiranje i micanje dijelova teksta zahtijevaju značajno manje napora nego kada se ista stvar radi na papiru. Na Fakultetu elektrotehnike i računarstva razvijen je sustav Edgar kroz koji studenti mogu pisati provjere znanja, vježbati zadatke i predavati zadaće u raznim oblicima. Velika prednost je što Edgar podržava i zadatke na koje se očekuje programski kod kao odgovor, a isti studenti mogu pokretati preko web sučelja prije same predaje kako bi se uvjerili u ispravnost. Iako poželjno, studente nije moguće imati pod nadzorom tijekom svih oblika provjera, kao što su recimo domaće zadaće, te je za očekivati da će određeni postotak studenata tu priliku iskoristiti za međusobno prepisivanje i plagiranje rješenja. Studentima su danas na raspolaganju brojni alati, izvorno napravljeni za rješavanje problema tijekom programiranja, a koji se mogu iskoristiti i u neželjene svrhe tijekom provjera znanja. Neki od tih alata su *online* forumi, javno dostupni repozitoriji koda te veliki jezični modeli. Kako bi se osigurala visoka kvaliteta studija te zajamčila ravnopravnost među svim studentima, nužno je što više pokušaja plagiranja detektirati. U okviru ovog diplomskog rada napravljen je pregled alata i tehnika za detekciju plagijata kao i sustav koji omogućuje nastavnicima koji koriste Edgar na svojim predmetima detekciju plagiranih rješenja nakon njihove predaje. Osim usporedbe programskih kodova, servis omogućuje i usporedbu rješenja u formatu čistog teksta te detekciju rješenja koja su nastala korištenjem umjetne inteligencije i velikih jezičnih modela. U okviru diplomskog rada razvijena je i pokazna web-aplikacija koja prikazuje sve mogućnosti sustava te omogućuje jednostavno konfiguriranje poslova usporedbe studentskih radova kao i pregled rezultata usporedbe.

1. Pregled alata i tehnika za detekciju plagijata

1.1. Zahtjevi alata za detekciju plagijata

Prema [1] čak 50 % do 79 % studenata odluči se za plagiranje barem jednom tijekom svog studija. Na predmetima s puno studenata ili velikim zadaćama, detekcija plagijata je izazovan zadatak. Alati kojima se nastavnici koriste stoga moraju zadovoljavati temeljna dva načela:

1. Robusnost na metode modificiranja plagiranog rješenja kako bi se izbjegla detekcija.
2. Precizno i točno detektiranje plagiranih rješenja.

Ova dva zahtjeva su povezana, ali ne u potpunosti identična. Prvi zahtjev govori da alat mora biti što otporniji na razne tehnike obfusciranja rješenja kako bi se izbjeglo njegovo detektiranje. Napretkom alata za detekciju plagijata, napredovale su i tehnike i alati za obfuskaciju. Neke od metoda obfuskacije programskog koda su izmjena imena varijabli, unošenje koda koji nema nikakvu funkciju (engl. *dead code*), pretvaranje *for* petlji u *while*, izmjena poretka funkcija i slično. Kvaliteta alata za detekciju plagijata mora osigurati da je vrijeme potrebno za modificiranje tuđeg rješenja i predavanje kao vlastitog, u cilju izbjegavanja detektiranja, veće ili jednako vremenu potrebnom za razvijanje vlastitog rješenja. S vremenom su razvijeni i alati koji uspješno iskorištavaju navedene ranjivosti, a opisani su u dijelu 1.2.3.

Preciznost u detektiranju plagijata odnosi se na sposobnost alata da detektira plagijat tamo gdje on uistinu postoji, ali isto tako i da što manje studentskih rješenja označava kao plagijat kada oni to nisu. Lažno pozitivne detekcije plagiranih rješenja mogu predstavljati značajan gubitak vremena za nastavnike koji ionako sva rješenja označena kao plagijat moraju ručno pregledati.

1.2. Pregled alata i tehnika za detekciju plagijata

Tehnike koje se koriste u alatima za detekciju plagijata mogu se prema [3] i [4] podijeliti na sljedeći način:

a) Usporedba studentskih kodova na osnovu atributa koje programski kod posjeduje, poznato još i pod nazivom tehnike zasnovane na otisku (engl. *fingerprint*). Za svaki program jednostavno bi se prikupile sve metrike te usporedile sa svim ostalim metrikama drugih programskih kodova.

   Prvi poznati alat za detekciju plagijata kojeg je napravio Ottenstein [6] nastao je upravo korištenjem ove tehnike. Ottenstein je koristio Halsteadove [5] metrike:

   - broj jedinstvenih operatora
   - broj jedinstvenih operanada
   - broj pojavljivanja operatora
   - broj pojavljivanja operanada.

   Navedeni alat koristio se za detekciju plagijata u programskom jeziku ANSI-FORTRAN, a za današnje potrebe smatra se prejednostavnim te nedovoljno kompleksnim za napredne detekcije plagijata.

b) Usporedba znakovnih nizova koji čine studentske kodove.

   Ovu tehniku alati mogu primjenjivati na jednostavniji način tako da koriste algoritme koji direktno uspoređuju dva znakovna niza te kao rezultat dobiju mjeru sličnosti u obliku postotka ili recimo broj operacija koje se moraju provesti da dva znakovna niza budu jednaka. Neki od poznatijih algoritama su: Levenshteinova udaljenost, Hammingova udaljenost, Jaro–Winkler i Ratcliff/Obershelp. Navedeni algoritmi su korisni za usporedbu čistog teksta, ili programskih kodova studenata proizašlih iz jednostavnijih zadataka. Primjena naprednijih algoritama usporedbe koda nije prikladna kada se kao rješenja zadataka očekuju relativno jednostavni isječci koda. U tom slučaju varijabilnost studentskih rješenja je minimalna i bilo koji napredniji algoritam bi detektirao previše lažno pozitivnih plagijata. Teškoća pronalaženja istinskih plagijata u slučaju jednostavnih zadataka jednako je izazovan problem i za nastavnika koji ručno detektira plagijate.

Kompliciraniji način primjene ove tehnike uključuje dva koraka: generiranje tokena (najmanja značajna jedinica programskog koda) pomoću prevoditelja za određeni programski jezik te pronalaženje sličnih sekvenci tokena u studentskim kodovima. Ovaj način je pouzdaniji jer generiranjem tokena automatski se izbjegavaju određeni pokušaji zaobilaženja označavanja rješenja kao plagijat. Naime, prilikom generiranja tokena, imena varijabli, funkcija, razreda i ostalog se gube, a isto tako određeni dijelovi koda mogu rezultirati istim tokenima. Jedan primjer je taj da *for* i *while* petlje poprimaju isti token. Iz ovoga je sada i jasno zašto ove metode nisu prikladne za usporedbu jednostavnih programskih isječaka. Slijed generiranih tokena na jedinstven način predstavlja studentsko rješenje. Svi nizovi tokena se međusobno uspoređuju tražeći podnizove koji su isti. Za pronalaženje sekvenci sličnih tokena, alati uglavnom koriste Running-Karp-Rabin Greedy String Tiling (RKR-GST) i Winnowing algoritme. Veliki broj podudarnih podnizova između dva studentska rješenja ukazuje da su ta dva rješenja potencijalni plagijati. Neki od alata koji primjenjuju ovu tehniku su: JPlag [8], Plague [9], YAP3 (Yet Another Plague) [10] i Sherlock [11].

c) Usporedba studentskih kodova kao parametriziranih nizova.

Autor B. Baker [12] je 1995. godine, suočen s problemom dupliciranja koda u velikim projektima, napravio alat Dup koji detektira identične ili skoro identične isječke koda u nekom projektu. Algoritam se temelji na postojanju dva skupa simbola, $\sum$ i Π, gdje $\sum$ predstavlja konačan skup jednostavnih simbola (operator pridruživanja '=', razne zagrade '{', '}' i slično), a Π predstavlja konačan skup simbola kojima se originalni programski kod parametrizira. Dva parametrizirana znakovna niza S i T su podudarna (*p-match*) ako postoji takva injektivna funkcija f, da vrijedi:

$$f(S) = T.$$

Primjer:

$\sum = \{a, b, c\}$,

$\Pi = \{x, y, w\}$,

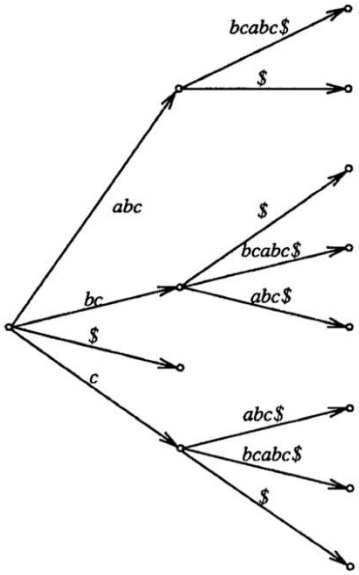
S = axaybxycby,

T = ayavbyvcbv

Primjenom injektivne funkcije koja svako pojavljivanje x i y zamjenjuje s y i v, na znakovni niz S dobije se niz koji je identičan nizu T.

Na jednostavnom primjeru relativno je lako pronaći za dva niza podudarne podnizove, no kod opsežnih programskih kodova trebalo bi značajno više vremena.

Ovaj problem rješava se konstruiranjem parametriziranih sufiksnih stabala (slika 1).



Slika 1. Primjer parametriziranog sufiksnog stabla za niz abcbcabc$

Pronalazak podudarnih podnizova u parametriziranom sufiksnom stablu značajno je ubrzalo rad programa.

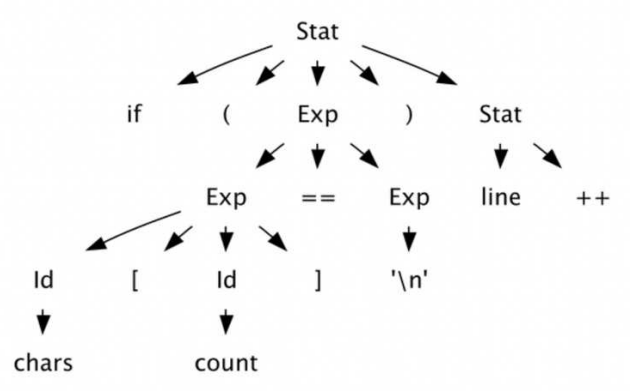
Ova tehnika usporedbe kodova ne primjenjuje se više u velikoj mjeri zbog nemogućnosti pronalaska naprednijih plagijata koji nisu samo izmjena naziva varijabli.

d) Usporedba grafova koji predstavljaju programski kod.

Za svaki programski isječak moguće je generirati razne grafove kao što su graf ovisnosti u programu, graf funkcijskih poziva i sintaksno stablo (slika 2). Eksperimentalno su razvijeni i brojni alati koji primjenjuju ove tehnike usporedbe koda smatrajući da grafovi, za razliku od čistih znakovnih nizova ili tokena, mogu prenijeti više informacija o samoj strukturi programa te tako zaobići apsolutno sve pokušaje izbjegavanja detekcije plagijata. Alati konstruiraju grafove na temelju koda te vodeći se izomorfizmom grafova računaju broj koraka potrebnih da dva grafa budu potpuno jednaka. Što je taj broj manji, time su dva koda sličnija.

Holmes [13] je alat namijenjen detekciji plagijata u jeziku Haskell. Kombinira usporedbu tokena i graf funkcijskih poziva.

GPlag [14] je alat namijenjen detekciji plagijata u jezicima C, C++ i Java. Za svaki programski isječak generira graf ovisnosti.



Slika 2. Primjer sintaksnog stabla

Iako su se tehnike usporedbe programskih grafova pokazale uspješnima čak i kad dva koda nisu direktna kopija, ne postoje dovoljno dobri alati koji koriste ovu tehniku i koji bi se mogli primjenjivati za usporedbu tisuće studentskih kodova u raznima programskim jezicima. Razlog leži u tome što provjera izomorfnosti grafova spada u kategoriju NP teških problema, a provjera izomorfnosti podgrafova u NP potpun problem. Algoritmi koji se koriste za provjeru izomorfnosti imaju kvazi – polinomijalnu složenost, a u najgorem slučaju eksponencijalnu. Alati koji su navedeni razvijeni su uglavnom u istraživačke potrebe.

e) Usporedba programskih kodova koristeći latentnu semantičku analizu (LSA).

LSA je tehnika pretraživanja relevantnih informacija iz velikih skupova podataka, tj. programskih kodova u ovom slučaju [15]. Tehnika prvo kreira matricu u kojoj su pobrojana pojavljivanja izraza programskih kodova za svaki programski kod. Na broj pojavljivanja opcionalno se primjenjuje i težinska funkcija koja osigurava da izrazi bitni prilikom detekcije plagijata dobiju više na važnosti prilikom izračuna sličnosti. Na dobivenu matricu primjenjuje se algoritam dekompozicije na singularne vrijednosti (SVD) čime se dobije više matrica, manjih dimenzija od originalne, ali koje pokazuju sličnosti između programskih kodova u raznim dimenzijama. Cilj SVD-a je smanjiti količinu šuma u podatcima te ostaviti samo one sličnosti koje su uistinu bitne za provjeru plagijata. SVD se temelji na pretpostavci da se dijelovi koda

koji obavljaju istu funkciju, ali nisu identično napisani, pojavljuju u istom okruženju. Smatra se da među izrazima postoje tranzitivne ovisnosti. To znači da LSA uspoređuje dva programska koda tako da pokuša pronaći isječke koda koji obavljaju istu funkcionalnost, ali su kod studenata napisani na malo drugačiji način. Prednost LSA je ta što je neovisna o programskom jeziku te se može koristiti i za usporedbu čistog teksta. Zanimljiva mogućnost je i to da nakon SVD-a je moguće kreirati tekstualne upite, a kao rezultat se dobiju svi dokumenti koji sadrže slične izraze. Alat PlaGate [15], razvijen u istraživačke svrhe, primjenjuje upravo ovu tehniku.

f) Usporedba programskih kodova na temelju ponašanja programa tijekom izvođenja.

Ova tehnika koristi simboličko izvršavanje programa kako bi prikupila sve informacije o tome na koji način određeni program koristi i modificira podatke te kako program surađuje s okolinom. Koristeći simboličko izvršavanje, moguće je analizirati apsolutno sve moguće tokove izvršavanja programa. Osim toga, za svaki program se može identificirati na koji način se stog i gomila koriste tijekom izvršavanja programa, kao i koje vanjske metode se pozivaju. Alat BPlag [1] iskorištava navedene metrike za usporedbu programskih kodova napisanih u Javi. Java je objektno orijentiran jezik te BPlag nastoji ponašanje objekata tijekom izvođenja prikazati u obliku grafa ovisnosti interakcija programa (engl. *program interaction dependency graph* - PIDG). Svaki programski kod BPlag na kraju predstavi kao skup nekoliko PIDG-ova gdje svaki PIDG nastoji prikazati određenu vrstu ponašanja programa. PIDG-ovi se dalje međusobno uspoređuju kako bi se odredile sličnosti između dvaju studentskih rješenja.

Cilj ovog diplomskog rada je iskoristiti jedan od alata koji primjenjuje neku tehniku detekcije plagijata i time automatizirati proces usporedbe kodova. Alat koji će se koristit mora zadovoljavati sljedeće uvjete:

- mogućnost usporedbe više od tisuću studentskih rješenja u jednom pokretanju
- mogućnost usporedbe studentskih rješenja organiziranih kao projekti koji se sastoje od više datoteka
- podrška za programske jezike: Java, C, C++, JavaScript, TypeScript, Python, C#
- podrška za usporedbu čistog teksta
- mogućnost integracije u projekt kao vanjska programska biblioteka

- mogućnost definiranja koda koji se treba izostaviti prilikom provjere plagijata (engl. *basecode*)
- izlazni format koji zadovoljava potrebe vizualizacije detektiranih plagijata.

Iako navedenih tehnika ima mnogo, većina alata koji su dovoljno kvalitetni, podržavaju veliki broj programskih jezika te su dovoljno robusni da mogu usporediti više od tisuću studentskih rješenja koriste tehniku usporedbe tokena generiranih iz programskog koda. Svi ostali alati razvijeni su u istraživačke svrhe, više se ne održavaju te su napisani korištenjem zastarjelih tehnologija što bi otežalo njihovu primjenu. Najpoznatiji i najviše korišteni alati danas su JPlag i MOSS, a princip njihova rada opisan je u sljedećim poglavljima.

1.2.1. JPlag

JPlag [8] je alat za detekciju plagijata, napisan u Javi, razvijen 1996. godine na Sveučilištu Karlsruhe. Temelji se na pronalaženju podudarnih nizova tokena u studentskim kodovima. Podržava 19 jezika za programiranje i modeliranje, uključujući i usporedbu čistog teksta. Uspješno se nosi s velikim brojem studentskih rješenja, a podržava i rješenja koja se sastoje od više datoteka. Dostupan je za korištenje kao Java biblioteka, a za jednostavnije potrebe može se koristiti i preko naredbene linije. Parametri koje JPlag prima prilikom pokretanja provjere plagijata su:

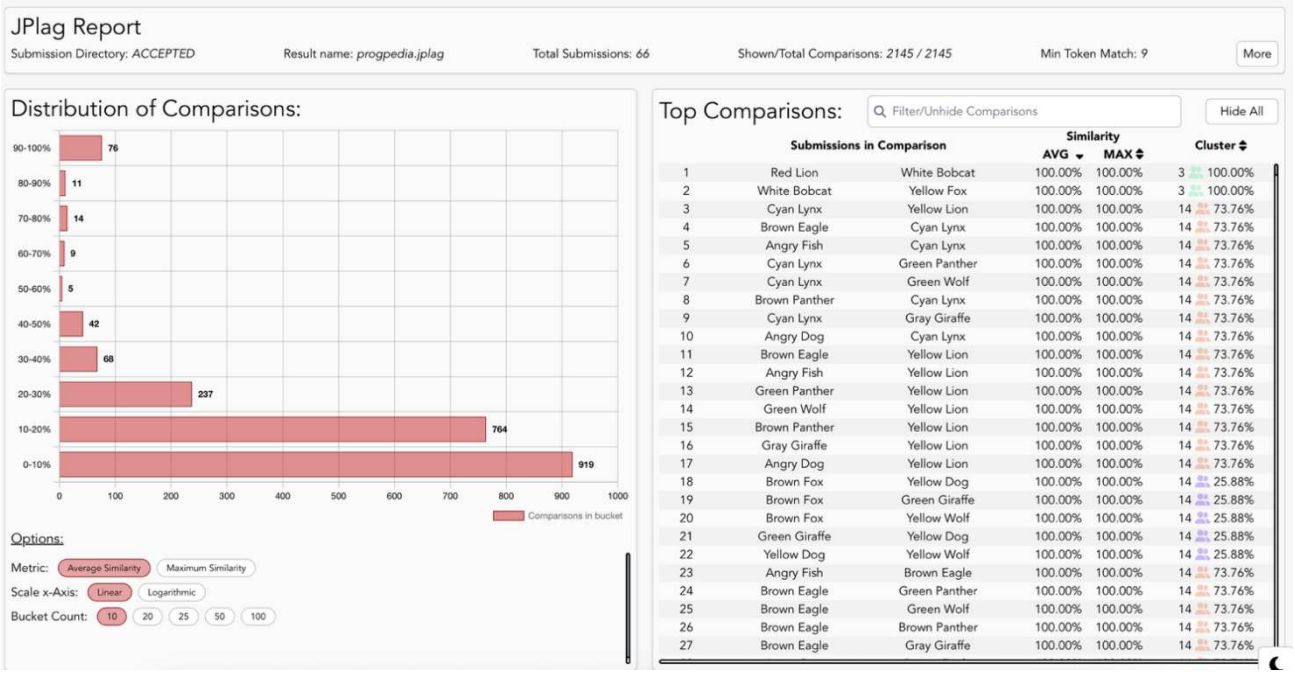
- programski jezik
- najmanji broj tokena koji trebaju biti isti u oba studentska rješenja kako bi se taj dio koda označio kao podudaran (engl. *minimal match length* - MML) - što je ovaj broj manji, to će JPlag pronaći više sličnih dijelova koda, ali uz opasnost od detektiranja previše lažno pozitivnih plagijata
- putanju do direktorija sa studentskim rješenjima iz trenutačne akademske godine
- putanju do direktorija sa studentskim rješenjima od prošlih akademskih godina
    - JPlag radi na principu da studentska rješenja uspoređuje svaki sa svakim, ali rješenja iz ovog skupa neće uspoređivati međusobno, već će se ona uspoređivati samo s rješenjima iz trenutačne akademske godine. Na ovaj način JPlag značajno smanjuje trajanje detekcije.
- putanju do direktorija s kodom koji se mora izostaviti prilikom provjere plagijata

    - Ako studentska rješenja sadrže dijelove koda koji se također nalaze u ovom direktoriju, ti će se dijelovi koda zanemariti prilikom izračuna sličnosti rješenja. Korisna opcija kada se prije početka zadaće svim studentima daje početni kod koji je potrebno modificirati.
- sufiksi datoteka koje se trebaju uzeti u obzir prilikom provjere plagijata
- ime poddirektorija čiji se sadržaj neće uzeti u obzir prilikom provjere plagijata
- minimalna sličnost dvaju studentskih rješenja nužna da se ta usporedba pojavi u konačnom rezultatu
- najveći dopušteni broj usporedbi u konačnom rezultatu
- opcije vezane uz grupiranje sličnih rješenja.

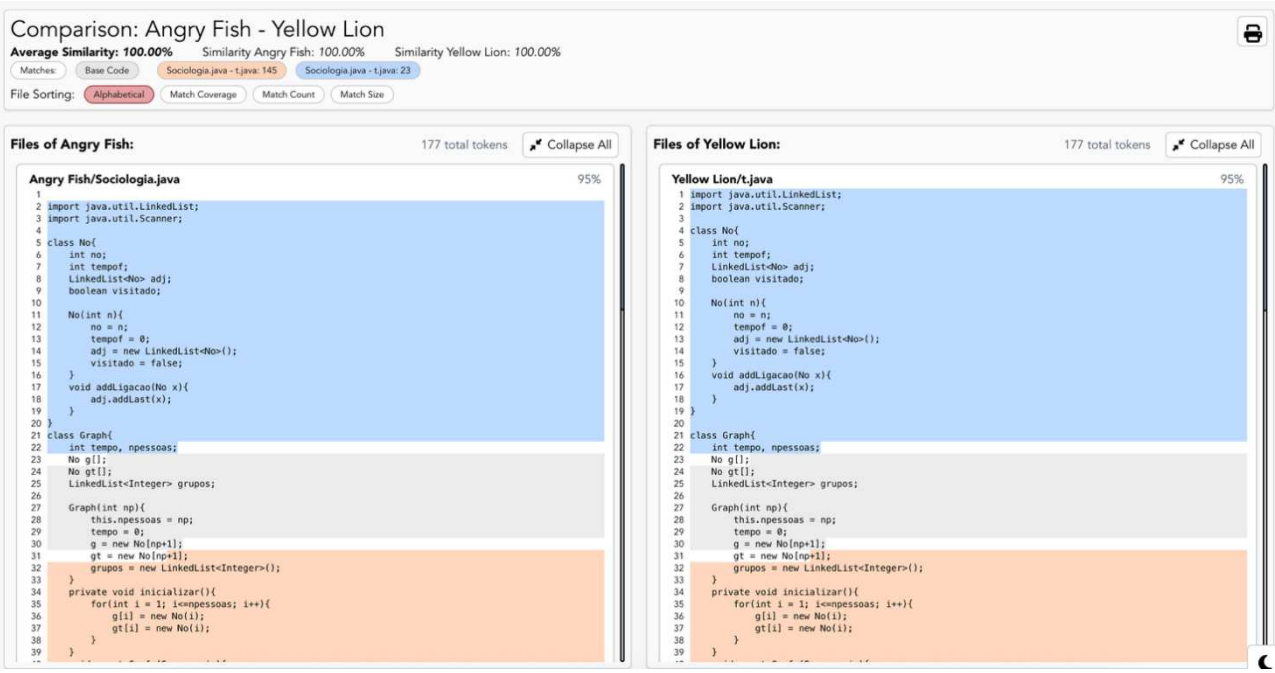
JPlag generiranje tokena studentskih rješenja, njihovu normalizaciju i usporedbu, provodi paralelno. Nikad se u istom trenutku ne izvode paralelno normalizacija i generiranje tokena nekih rješenja već sami poslovi unutar pojedinih koraka. Naime, ovi zadatci su u potpunosti neovisni jedan od drugih te se smatraju primjerom trivijalno paralelnog problema. Jednom kad je provjera gotova, JPlag generira datoteku u kojoj su pohranjeni podatci o svim sličnostima studentskih rješenja. Datoteka se potom prenese na JPlag web aplikaciju koja nudi razne opcije vizualizacije rezultata. Na slici 3 vidi se početna stranica vidljiva nakon prijenosa datoteke s rezultatima usporedbe. JPlag nudi uvid u distribuciju sličnosti rješenja, kao i popis svih rezultata međusobne usporedbe poredanih prema sličnosti, počevši od najveće. Pritiskom na rezultat usporedbe između dva studenta, otvara se stranica (slika 4) na kojoj se studentski projekti ručno mogu usporediti jedan pokraj drugog, a linije koda za koje JPlag smatra da su međusobno kopirane označene su bojom. JPlag također kreira i grupe studenata za koje smatra da su surađivali što je vidljivo na slici 5. JPlag trenutačno podržava aglomerativno hijerarhijsko grupiranje te spektralno grupiranje koristeći algoritam K-sredina. Korisniku se omogućava implementiranje vlastitog algoritma grupiranja kojeg onda JPlag koristi prilikom stvaranja grupa. Aglomerativno grupiranje radi hijerarhiju grupa koja se može prikazati u obliku dendrograma (slika 9). Dendrogram je dijagram u obliku stabla koji prikazuje kako su studentska rješenja grupirana. Izgradnja dendrograma kreće tako što je na početku svako studentsko rješenja svoja grupa, tj. list stabla. Između takvih grupa pronađe se par grupa ili rješenja koja su slična koristeći neku mjeru sličnosti i sljedeće opcije povezivanja rješenja: jednostruka povezanost, potpuna povezanost ili prosječna povezanost rješenja. Vodeći se pretpostavkom da su grupe rješenja kompaktne i prirodno dobro

odvojene, svejedno je koja se od navedenih metoda koristi. Ako bi se grupe prikazale grafički, jednostruko povezivanje daje dugačke, lančaste grupe, a potpuno povezivanje manje i zbijenije grupe. Prosječna povezanost je kombinacija ovih dviju te se preporučuje koristiti u slučaju nesigurnosti izgleda grupa. Najbliže grupe ili rješenja se spajaju u novu grupu te se cijeli proces ponavlja sve dok se ne dobije samo jedna grupa.

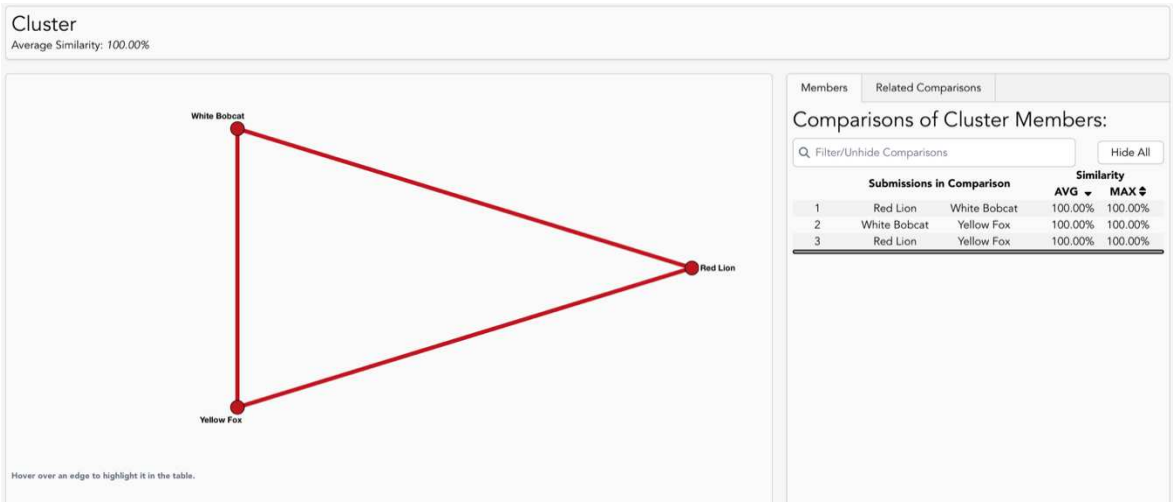
Grupiranje koristeći algoritam K-sredina temelji se na odabiru k grupa gdje svaka grupa ima svoj centroid koji se na početku nasumično odabire, a s vremenom se pomiče u prostoru kako bi se poboljšalo grupiranje (slika 7). Algoritmi grupiranja ne znaju unaprijed broj grupa u prostoru studentskih rješenja stoga se algoritam K-sredina pokreće više puta s različitom vrijednosti k, sve dok se ne nađe optimalno grupiranje. Algoritam K-sredina nešto je brži od algoritma aglomerativnog grupiranja, ali postoji mogućnost da zapne u lokalnom suboptimalnom rješenju. JPlagove zadane postavke koriste algoritam aglomerativnog grupiranja.



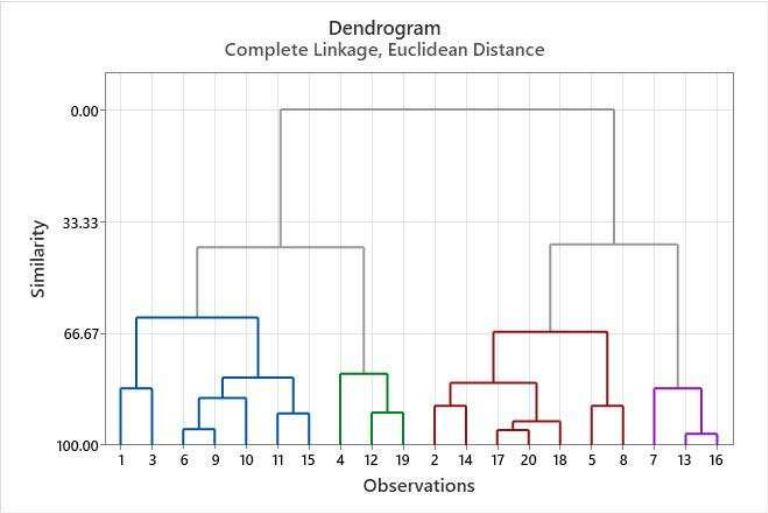
Slika 3. JPlag prikaz rezultata usporedbe



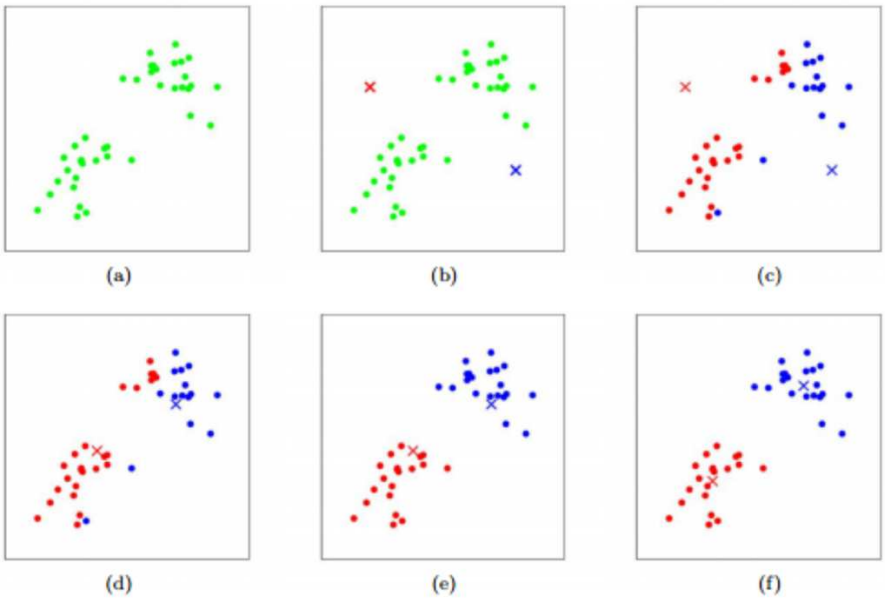
Slika 4. JPlag usporedba dva studentska projekta



Slika 5. JPlag grupiranje studentskih rješenja



Slika 6. Općeniti dendrogram koji bi nastao grupiranjem studentskih rješenja [25]



Slika 7. Grupiranje algoritmom K-sredina gdje se vidi kako algoritam od iteracija a do f mijenja položaj centroida sve dok ne bude dovoljno dobar [26]

Prvi korak u radu JPlaga je generiranje tokena koristeći prevoditelj za odgovarajući programski jezik. Prevoditelji kao izlaz daju sintaksno stablo koje JPlag linearizira dobivajući tako sekvencu tokena. Na slici 8 vidi se primjer generiranih tokena za isječak Java koda. JPlag ignorira sve tokene nepotrebne za usporedbu, kao što su prazni red, razmaci ili komentari. Za svaki generirani token JPlag pamti poziciju u originalnom kodu gdje se nalazi kod koji je povezan s tim tokenom. Na ovaj način JPlag prilikom vizualizacije rezultata usporedbe može označiti slične dijelove koda.

	Java source code	Generated tokens
1	`public class Count {`	BEGINCLASS
2	`  public static void main(String[] args)`	VARDEF,BEGINMETHOD
3	`            throws java.io.IOException {`	
4	`    int count = 0;`	VARDEF,ASSIGN
5		
6	`    while (System.in.read() != -1)`	APPLY,BEGINWHILE
7	`      count++;`	ASSIGN,ENDWHILE
8	`    System.out.println(count+" chars.");`	APPLY
9	`  }`	ENDMETHOD
10	`}`	ENDCLASS

Slika 8. Primjer JPlag generiranja za isječak Java koda

Generiranje tokena automatski sprječava sljedeće pokušaje zaobilaženja detekcije plagijata:

a) ubacivanje ili promjena komentara

b) modifikacija ispisa programa

c) promjena imena varijabli, metoda i razreda

d) grupiranje ili odvajanje deklaracija varijabli

e) promjena modifikatora vidljivosti iz *public* u *private* i obrnuto

f) promjena konstantnih vrijednosti.

Generiranje tokena jedini je korak u radu JPlaga koji je ovisan o korištenom programskom jeziku stoga autori JPlaga taj korak zovu i *front-end* proces. Velika prednost ovakvog pristupa je što je dodavanje podrške za nove jezike jednostavno bez potrebe za modificiranjem glavnog algoritma usporedbe tokena.

Sljedeći korak u radu JPlaga je pronaći slične tokene u različitim studentskim rješenjima. JPlag prije ovog koraka unaprijed generira sve moguće parove studentskih rješenja koja se moraju usporediti. Za pronalazak podudarnih tokena koristi se Running-Karp-Rabin Greedy String Tiling (RKR-GST) algoritam. Prilikom usporedbe studentskih rješenja A i B, JPlag osigurava da se svaki token iz rješenja A može upariti samo s jednim tokenom iz rješenja B te neovisnost pozicije tokena za uspješno uparivanje. Ove karakteristike su nužne kako bi JPlag uspio pronaći slične dijelove koda bez obzira na redoslijed tokena. JPlag računa *hash* vrijednost svih postojećih podnizova čija duljina odgovara najmanjem broju tokena koji trebaju biti isti u oba studentska rješenja kako bi se neki dio koda označio kao podudaran. JPlag uspoređuje *hash* vrijednosti rješenja A i B, svaki sa svakim, te kad pronađe istu *hash* vrijednost provodi usporedbu stvarnih tokena koji stoje iza *hash* vrijednosti nastojeći pronaći

što dulji niz istih tokena. Ako se za neki token kasnije ispostavi da pripada duljem nizu sličnih tokena, onda se prvo podudaranje ignorira, a novo uzima u obzir. Složenost JPlag algoritma u najgorem slučaju je O (n^3), a u prosječnom O (n^2) gdje je n broj znakova većeg rješenja.

Mjeru sličnosti, *sim,* JPlag računa koristeći izraz (1) gdje pokrivenost odgovara sumi duljina svih pronađenih sličnih dijelova koda.

$$sim = \frac{2*pokrivenost}{|A|+|B|} \quad (1)$$

Jednostavan primjer rada JPlaga na programima sa slike 9 temelji se na pretpostavci da je minimalna duljina preklapanja tokena, M=2, te da svaka linija programskog koda generira jedan token. Generirani tokeni za oba programa vidljivi su na slici 10. Prema tome, programu A pripadaju tokeni 1, 2, 3, 4, 5, 6 i 7, a programu B 1, 2, 6, 7. Sljedeći korak u radu JPlaga je generiranje *hasheva* svih podnizova generiranih tokena čija duljina je u ovom slučaju 2. Program A ima 8 podnizova, a program B 3. Podnizovi se rade kao klizeći prozori. Podnizovi i njihovi *hashevi*, koji su radi jednostavnosti u ovom slučaju nasumični brojevi, vidljivi su na slici 11. Za svaki *hash* dodatno se pamti i pozicija tokena u programskom kodu za koji je *hash* vezan. Sljedeći korak algoritma je usporedba svih mogućih parova *hasheva* na način da se iterira po *hashevima* manjeg programa pokušavajući pronaći iste *hasheve* u većem programu. Jednom kada se pronađu dva ista *hasha*, JPlag provjerava identičnost tokena programskog koda koji stoje iza tih *hasheva*. Za ovaj primjer, u prvoj iteraciji algoritma utvrdit će se da su isti *hashevi* 103 koji odgovaraju podnizovima [1, 2]. Greedy String Tiling nastavlja dalje te utvrđuje da hashevi 208 i 205 nisu isti te u tom trenutku je podniz [1, 2] najdulji pronađeni slijed tokena. Algoritam nakon ovoga ne staje jer je moguće da će naići na dulje preklapanje tokena za *hash* 103 iz programskog koda B. U sljedećem koraku algoritam preskače *hasheve* 205, 307, 409 i 506 iz programskog koda A te dolazi ponovno do istog *hasha* 103 u oba koda, ali sada na drugačijoj poziciji. U ovom slučaju algoritam utvrđuje da su i sljedeći par podnizova [2, 6] i [6, 7] isti te nakon toga staje jer je došao do kraja programa B. Sada je slijed tokena koji čine podnizove [1, 2], [2, 6] i [6, 7] označen kao preklapajući. Broj pokrivenih tokena je 4, a duljine programa A i B su 9, odnosno 4 tokena. Prema formuli (1), izračunata sličnost je 61.5 %.

```
Program A:

a = 1
b = 2
x = 5
y = 6
z = x + y
a = 1
b = 2
c = a + b
print(c)

Program B:

a = 1
b = 2
c = a + b
print(c)
```

Slika 9 Primjer dvaju sličnih programa za demonstraciju JPlag algoritma

a = 1	1
b = 2	2
x = 5	3
y = 6	4
z = x + y	5
c = a + b	6
print(c)	7

Slika 10 JPlag generirani tokeni na temelju programskog koda sa slike 9

```
Program A:      Program B:

[1,2] = 103     [1,2] = 103
[2,3] = 205     [2,6] = 208
[3,4] = 307     [6,7] = 613
[4,5] = 409
[5,1] = 506
[1,2] = 103
[2,6] = 208
[6,7] = 613
```

Slika 11 Podnizovi programa sa slike 9 i njihovi pripadajući *hashevi*

1.2.2. MOSS (Measure of Software Similarity)

MOSS [16] je alat namijenjen detekciji plagijata razvijen 1994. godine na sveučilištu Stanford. Dostupan je kao web servis koji prima datoteke koje čine studentska rješenja, a kao rezultat vraća niz HTML dokumenata pomoću kojih se vizualno prikazuju rezultati

provjere plagijata. Za korištenje servisa, MOSS nudi Perl skriptu, a s vremenom su razvijeni i klijenti koji tu istu skriptu implementiraju u Javi, C-u i Pythonu.

Za razliku od JPlaga MOSS podržava značajno manji broj mogućnosti konfiguriranja prije pokretanja provjere:

a) programski jezik
b) putanju do koda koji se treba ignorirati prilikom provjere plagijata
c) minimalni broj ponavljanja nakon čega se neki odsječak koda ignorira prilikom provjere plagijata – ako se neki odsječak koda pojavljuje kod velikog broja studenata vjerojatno je riječ o općepoznatom kodu.

Skripta za pokretanje MOSS-a sva studentska rješenja prenosi na poslužitelje Sveučilišta u Stanfordu.

Podržava 26 jezika za modeliranje i programiranje. Moss, kao i JPlag radi na principu usporedbe tokena koji čine programski kod, stoga nije osjetljiv na razne oblike pokušaja zaobilaženja detekcije plagijata navedenih u prošlom dijelu. Na slici 12, vidi se na primjeru isječka programa napisanog u C-u kakve tokene generira.

```
int hello = 0;
return hello;
```

```
TYP_INT ID EQ NUM SEMI
RET ID SEMI
```

Slika 12. Primjer MOSS generiranih tokena

U sljedećem koraku, sekvenca generiranih tokena dijeli se u preklapajuće k-grame, gdje je k interni i korisniku nedostupan parametar - prag šuma. Što je ovaj broj veći, to će MOSS-u trebati više istih tokena poredanih jedan do drugog kako bi MOSS taj dio koda označio kao plagiran. Za svaki k-gram, MOSS računa *hash* vrijednosti. Za tokene generirane na slici 12, *hash* vrijednosti k-grama su vidljive na slici 13, pod pretpostavkom da parametar k ima vrijednost 3.

```
30 15 56 83 71 10
```

Slika 13. MOSS *hash* vrijednosti k-grama

Nakon generiranja *hash* vrijednosti, MOSS provodi winnowing algoritam koji za svako studentsko rješenje stvara otisak (engl. *fingerprint*). Prvi korak stvaranja otiska je da se sve dobivene *hash* vrijednosti u prethodnom koraku grupiraju u preklapajuće prozore duljine *w* (2).

$$w = t - k + 1$$

t = prag zajamčenog podudaranja – minimalna duljina tokena koji će sigurno biti označeni kao sličnima prilikom provjere plagijata (2)

Kao posljedicu ovakvog odabira veličine prozora, svaki podniz tokena u nekom studentskom rješenju ima barem jedan pripadajući prozor *w* te pronalaskom barem jedne iste *hash* vrijednosti iz prozora u oba rješenja, MOSS bi utvrdio sličnost. Primjer generiranih preklapajućih prozora vidljiv je na slici 14. Konačni otisak ulaznog rješenja čine samo najmanje *hash* vrijednosti pojedinog prozora s pravilom da ako dva susjedna prozora imaju istu najmanju vrijednost, samo jedna se uzima u obzir. Otisak za navedeni primjer vidljiv je na slici 15. Za svaki generirani otisak, MOSS prati poziciju u originalnom rješenju na koju se *hash* vrijednost odnosi, što je bitno kasnije za vizualizaciju rezultata. MOSS pomoću *hash* vrijednosti i informacija o poziciji gradi indeks koji je sličan invertiranom indeksu kojeg internetske tražilice koriste i koji sadrži informacije o položaju riječi u dokumentu. Nakon ovoga, za svako studentsko rješenje se ponovno gradi otisak te se pomoću indeksa traže ostala studentska rješenja koja imaju iste *hash* vrijednosti. Ovim pristupom, MOSS izbjegava direktnu usporedbu svih mogućih parova studentskih rješenja čiji broj je kvadratno ovisan o broju studenata. Tek kad se pomoću indeksa nađu potencijalno slična rješenja, stvaraju se parovi rješenja koja će se detaljnije usporediti.

```
(30 15 56) (15 56 83) (56 83 71) (83 71 10)
```

Slika 14. MOSS preklapajući prozori *hash* vrijednosti

```
(15 56 10)
```

Slika 15 MOSS otisak programskog koda

Primjer konačnog rezultata MOSS usporedbe vidljiv je na slici 19.

Na primjeru koda A i B sa slike 9 MOSS bi također, uz pretpostavku korištenja istog prevoditelja, generirao tokene sa slike 10. Neka su parametri k = 3, t = 5 te w prema (2), 3. K-grami koje MOSS generira za kod A i B kao i njihove *hash* vrijednosti vidljive su na slici 16. Na temelju *hash* vrijednosti računaju se preklapajući prozori duljine w = 3 za programe A i B koji su vidljivi na slici 17. Na temelju generiranih prozora za svaki program moguće je dobiti pripadajuće otiske vidljive na slici 18. Program B u ovom slučaju ima manje od 3

hash vrijednosti te se u tom slučaju za njegov otisak uzimaju obje vrijednosti. Sljedeći korak je izgradnja indeksa na temelju otisaka za svaki program zbog brže pretrage istih otisaka među različitim studentskim rješenjima. U ovom primjeru jasno je da je jedino otisak 20615 isti u oba programa. Ovaj otisak prekriva prve tri linije programa B koje su iste u programu A. U ovom slučaju se vidi i razlika u odnosu na JPlag koji bi u ovom slučaju nastavio uspoređivati tokene sve dok ne bi naišao na različite te bi označio i zadnju liniju programa B kao istu iz programa A, dok MOSS samo traži iste otiske bez gledanja u tokene. Jasno je da u slučaju duljeg programa B, bi i zadnja linija imala mogućnost biti detektirana kao ista.

```
Program A:              Program B:

[1,2, 3] = 10406        [1,2, 6] = 10409
[2,3, 4] = 20608        [2,6, 7] = 20615
[3,4, 5] = 30814
[4,5, 1] = 41007
[5,1, 2] = 51208
[1,2, 6] = 10409
[2,6, 7] = 20615
```

Slika 16 MOSS K-grami i pripadne *hash* vrijednosti

```
Program A:

(10406, 20608, 30814), (20608, 30814, 41007), (30814, 41007, 51208)
(41007, 51208, 10409), (51208, 10409, 20615)

Program B:

(10409, 20615)
```

Slika 17 MOSS preklapajući prozori *hash* vrijednosti

```
Program A:

(10406, 20608, 30814, 10409)

Program B:

(10409, 20615)
```

Slika 18 MOSS otisci programa iz primjera sa slike 17

Slika 19. MOSS primjer rezultata usporedbe

1.2.3. Metode obrane od pokušaja zaobilaženja detekcije plagijata

Alati koji detekciju plagijata temelje na provjeri podudarnih sekvenci tokena otporni su na većinu jednostavnih napada navedenih u dijelu 1.2.1., a koji se u velikoj mjeri temelje na preimenovanju programskih elemenata. Napredniji napadač koji želi izbjeći da njegovo rješenje bude detektirano kao plagijat mogao bi primijeniti tehniku promjene redoslijeda koda ili ubacivanja linija koda koje nemaju svrhu. Ubačene linije koda mogu pozivati metode koje se koriste u programu, a čije pozivanje više puta neće imati negativne posljedice. Moguće je i istoj varijabli pridružiti vrijednost više puta ako je ta operacija idempotentna. Promjena redoslijeda koda efektivan je napad samo ako se pomiču manji dijelovi koda, manji od minimalne duljine tokena za koje alat tvrdi da će sigurno detektirati kao slične. Ako je pomaknuti dio koda veći od te vrijednosti, alat će detektirati da je riječ o pomaku cijelog bloka koda te ispravno detektirati plagijate. Općenito, napadi ubacivanja koda ili promjene redoslijeda, opasni su za alate jer uzrokuju promjenu sekvenci tokena te se potencijalno jednaki podnizovi neće moći detektirati. Alat Mossad [2] napravljen je u svrhu iskorištavanja ove ranjivosti. Kao ulaz prima studentsko rješenje koje je potrebno modificirati te niz korisnički definiranih linija koda. Kombinacijom koda koji se već nalazi u studentskom rješenju s vanjskim definiranim linijama koda, Mossad modificira predano rješenje umetanjem i promjenom redoslijeda koda sve dok sličnost između predanog rješenja i novo generiranog ne padne ispod korisnički definiranog praga. Kod koji se umeće uglavnom je kod koji ne doprinosi radu programa (engl. *dead code*). Umetanje koda koji se

temelji na dosadašnjem kodu napisanom u studentskom rješenju provodi se rjeđe jer je alatu teško prepoznati kod koji se može duplicirati bez da se promjeni rad programa. Promjena redoslijeda još je izazovniji zadatak jer alat mora paziti na međusobne ovisnosti između dijelova koda kako ne bi promijenio funkcionalnost programa ili potencijalno onemogućio njegovo prevođenje. Na slici 20 vidi se primjer modifikacije jednostavnog programa alatom Mossad, a na slici 21 i promjene u generiranim tokenima dvaju programa. Mossad se pokazao uspješnim u zaobilaženju JPlag i MOSS detekcija. Zbog etičkih razloga, implementacija nije javno dostupna, ali istraživanje je potaklo unaprjeđenje u radu alata koji detekciju plagijata temelje na usporedbi niza tokena. Metoda koja se pokazala uspješnom je normalizacija sekvenci tokena [17].

```
1  void printSquares() {              void printSquares() {
2   int i = 1;                         int i = 1;
3                              (+)     boolean debug = false;
4   while (i <= 10) {                  while (i <= 10) {
5    int square = i * i;                int square = i * i;
6    println(square);          (~)      i++;
7    i++;                      (~)      println(square);
8   }                                  }
9  }                                  }
```

Slika 20. Mossad modifikacija programa

#	Original Tokens	→	Variant Tokens
1	method start		method start
2	variable		variable
3		(+)	variable
4	loop start		loop start
5	variable		variable
6	apply	(~)	assignment
7	assignment	(~)	apply
8	loop end		loop end
9	method end		method end

Slika 21. Mossad promjene u generiranim tokenima

Normalizacija tokena je metoda koju JPlag provodi nakon prevođenja, a prije same usporedbe tokena.

Proces normalizacije minimalno je ovisan o programskom jeziku tako da od prevoditelja za svaki generirani token prikuplja sljedeće dodatne informacije:

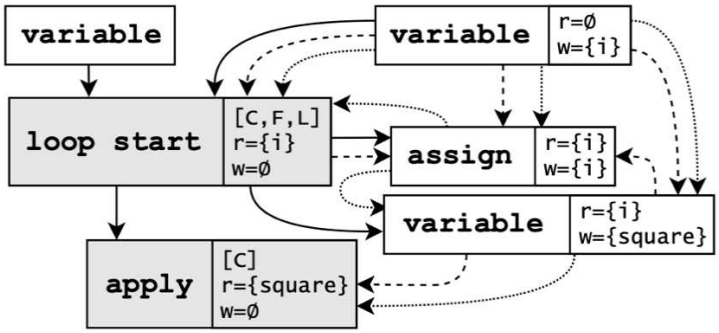
a) Pripadnost tokena skupini visoko kritičnih tokena. Primjer visoko kritičnih tokena su oni koji pozivaju funkcije ili pokreću programske petlje. To su uglavnom svi tokeni koji pokreću obavljanje neke radnje u kodu, a ne oni koji sudjeluju u pridruživanju vrijednosti nekoj varijabli. U primjeru sa slike 20 to bi bili tokeni koji pokreću programske petlje.
b) Nužnost očuvanja pozicije tokena. U primjeru sa slike 20 nužno je paziti da svi tokeni koji se nalaze unutar *while* petlje ostanu unutar nje.
c) Popis drugih tokena o kojima token ovisi. Svaki token ima listu ostalih tokena o kojem ovisi te je ta lista podijeljena u listu za pisanje i čitanje. U primjeru sa slike 20, token koji označava početak petlje, u svojoj listi za čitanje ima token *i* jer vrijednost tog tokena direktno ovisi o broju ponavljanja petlje.
d) Zastavica koja označava da token sudjeluje u definiranju bloka koda koji se ponavlja više puta. U primjeru sa slike 21, tokeni *loop start* i *loop end,* oboje imaju ovu zastavicu postavljenu na *true,* što JPlagu govori da između ta dva tokena postoje drugi tokeni čiji redoslijed se možda ne mora u potpunosti poštivati.

Sljedeći korak u procesu normalizacije je konstrukcija grafa normalizacije tokena (TNG – engl. *token normalization graph*). TNG je vrsta grafa ovisnosti programa s tom razlikom da čvorovi grafa nisu dijelovi programskog koda već tokeni. Nakon prikupljanja semantičkih informacija o tokenima, generacija TNG-a i svi ostali koraci normalizacije neovisni su o programskom jeziku. Tokeni koji čine čvorove TNG-a povezani su s tri tipa bridova:

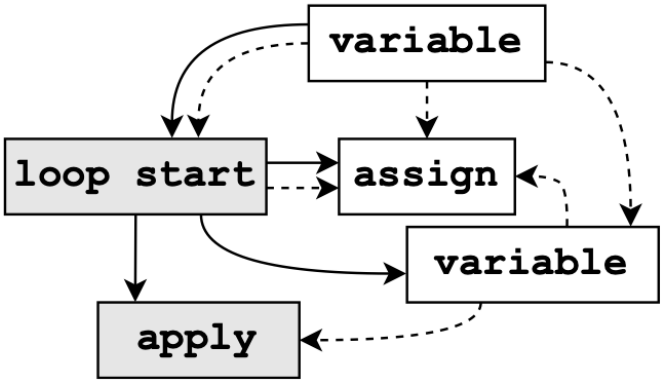
a) bridovi koji označavaju da neki dio koda piše u varijablu koju drugi dio koda čita
b) bridovi koji označavaju nužnost očuvanja poretka između dva dijela koda radi očuvanja vrijednosti varijable
c) bridovi koji označavaju nužnost očuvanja poretka između dva dijela koda radi očuvanja programske logike koja ne uključuje očuvanje vrijednosti varijable.

Na slici 22 vidi se primjer TNG-a za plagirani program sa slike 20. Pune strelice su bridovi tipa a), isprekidane strelice su bridovi tipa b) te istočkane strelice su bridovi tipa c). Tokeni *loop start* i *apply* su u ovom primjeru visoko kritični tokeni. Uklanjanje mrtvog koda provodi se uklanjanjem svih čvorova iz kojih se ne može doći u čvorove koji su označeni kao visoko kritični koristeći bridove tipa a). TNG koji se dobije ovim procesom vidljiv je na slici 23.

Nakon uklanjanja mrtvog koda i sortiranja koda unutar petlje koristeći topološko sortiranje, za plagirani program sa slike 20 dobije se niz tokena vidljiv na slici 24. Usporedbom tokena sa slike 24 i 21 vidljivo je da nema razlike u generiranim tokenima za dva vizualno različita programa.



Slika 22. Primjer TNG-a



Slika 23 Primjer TNG-a nakon uklanjanja mrtvog koda

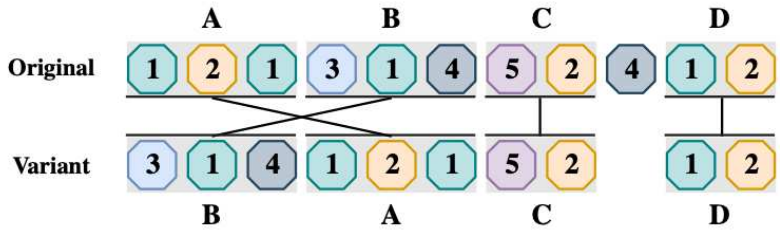
```
method start
  variable

 loop start
  variable
    apply
 assignment
  loop end
method end
```

Slika 24. Primjer konačnog niza tokena nakon provedene normalizacije

Normalizacija tokena se pokazala kao vrlo učinkovita metoda te u istraživanju provedenom u [17] za plagirana studentska rješenja koja su imala prosječnu sličnost manju od 20 %, nakon provedene normalizacije tokena, sličnost se povećala na preko 95 %.

Trenutačna JPlagova najnovija verzija uvodi još jedan način obrane od obfuskacije koda. Spajanje sekvenci sličnih tokena pokazalo se uspješnom metodom obrane i od alata kao što su Mossad, ali i od velikih jezičnih modela koji se mogu koristiti u svrhu automatskog obfusciranja koda ili generiranja cijelog koda iz nule. Veliki jezični modeli predstavljaju značajan problem jer za razliku od dosadašnjih automatiziranih obfuskatora koda za koje se točno zna na koji način mijenjaju kod, veliki jezični modeli kod mogu obfuscirati na veliki broj različitih načina u kratkom periodu. Na slici 25 vidljive su sekvence A, B, C i D tokena koje su iste u dva rješenja. JPlagov algoritam će spojiti susjedne sekvence udaljene minimalno određeni broj tokena te duljine veće od minimalno propisane. U primjeru sa slike, to su sekvence C i D. U slučaju većeg programskog koda, navedeni algoritam bi se nastavio sve dok ne bi bilo više sekvenci koje zadovoljavaju navedene uvjete. Koristeći navedeni algoritam prema studiji obavljenoj u [18], plagiranim rješenjima nastalim koristeći Mossad sličnost je porasla za 30 % do 44 %, a kod onih nastalim koristeći velike jezične modele između 9 % i 16 %.



Slika 25 JPlag susjedne sekvence tokena

Lista navedenih prijetnji alatima za detekciju plagijata praktički je neograničena, osobito napretkom velikih jezičnih modela. Koristeći tehnike navedene u ovom poglavlju moguće je smanjiti izloženost njima, ali treba biti svjestan da napredniji plagijator i dalje ima šansu ne biti detektiran. Ovo ne znači da nema smisla koristiti alate kao što su JPlag jer većina plagijata se nalazi na uvodnim računarskim predmetima te se velik dio plagijatora niti ne trudi uvelike zamaskirati tuđe rješenje kao vlastito. Prema trenutačnom stanju [17], istraživači su pouzdani u rad JPlaga u toj mjeri da ako student uspije zaobići detekciju plagijata da samim time već ima dovoljno znanja potrebnog da sam implementira traženi problem.

2. Servis za automatsku detekciju plagijata

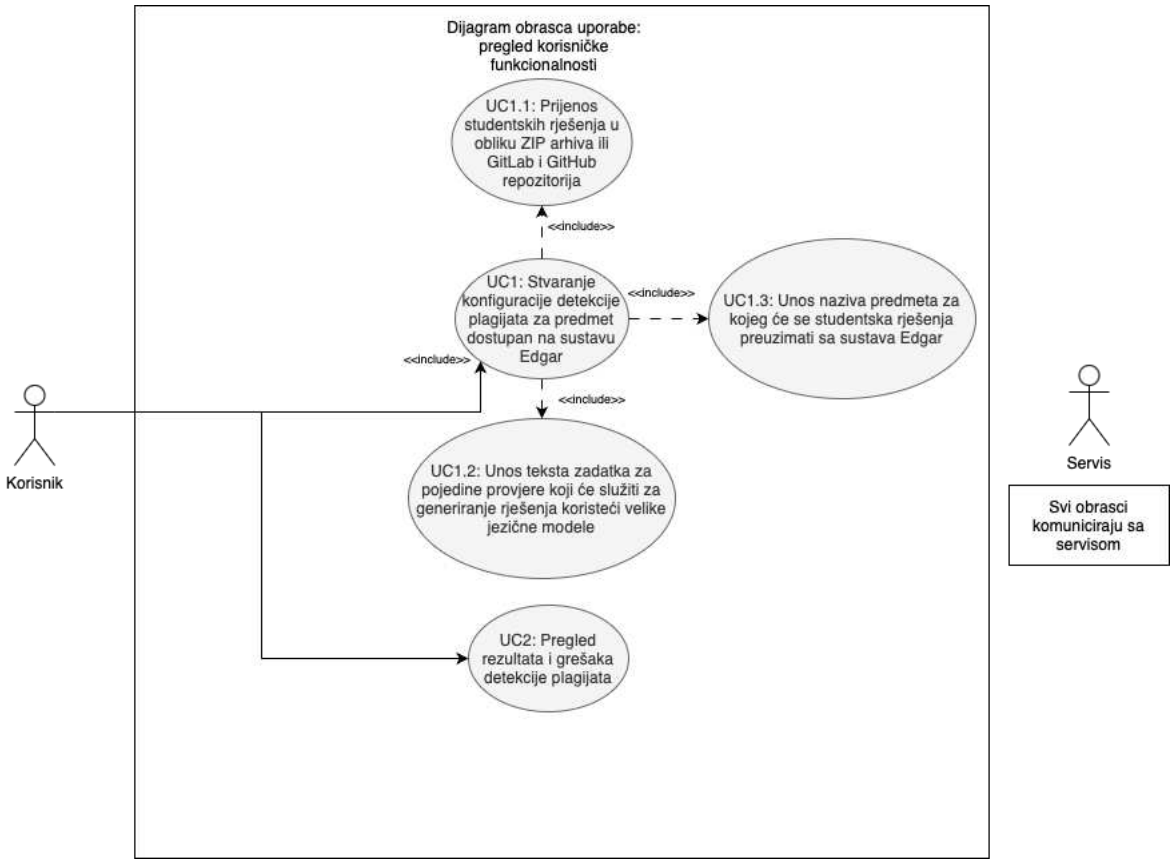
Za potrebe automatske detekcije plagijata u ovom radu koristi se alat JPlag. JPlag, za razliku od svih ostalih alata nabrojanih u prethodnom poglavlju, otvorenog je koda, napisan u Javi te i dalje u redovnom razvoju s nekoliko novih izdanja godišnje. MOSS, iako vrlo popularan, nije dovoljno dobar jer je dostupan samo kao web servis s ograničenim brojem usporedba po danu.

2.1. Funkcionalni zahtjevi sustava

Funkcionalni zahtjevi predstavljaju sve usluge koje programski proizvod mora pružiti korisnicima te definiraju kako sustav reagira na određene ulazne poticaje. Cilj ovog servisa je omogućiti nastavnicima što lakšu detekciju plagijata, posebice ako za potrebe provjera znanja, laboratorijskih vježbi i domaćih zadaća koriste sustav Edgar. Iz ovog cilja proizlaze sljedeći funkcionalni zahtjevi:

1. Sustav mora omogućiti automatsku detekciju plagijata nad unaprijed definiranim skupom studentskih rješenja nastalih tijekom trenutačne ali i prethodnih akademskih godina.
2. Sustav mora omogućiti korisniku definiranje programskog koda čije se pojavljivanje u studentskim rješenjima ignorira prilikom izračuna sličnosti.
3. Sustav mora omogućiti korisniku definiranje skupa studentskih rješenja prijenosom ZIP arhiva, unosom poveznice na GitHub ili GitLab repozitorije i unosom naziva predmeta iz sustava Edgar.
4. Sustav mora omogućiti provjeru sličnosti studentskih rješenja s unaprijed generiranim rješenjem od strane velikih jezičnih modela.
5. Sustav mora omogućiti korisniku stvaranje konfiguracije za pojedine predmete, definirajući pritom pravila prema kojima će se detekcija plagijata obavljati.
6. Sustav mora omogućiti korisniku pregled rezultata detekcije plagijata i mogućih pogrešaka prilikom detekcije.

Na dijagramu sa slike 26 vidljive su sve funkcionalnosti sustava iz perspektive korisnika.



Slika 26. Dijagram obrasca uporabe, pregled korisničke funkcionalnosti

2.2. Nefunkcionalni zahtjevi sustava

Nefunkcionalni zahtjevi opisuju svojstva koja sustav mora imati, a ne funkcionalnost koju pruža korisniku.

Ovaj sustav mora:

1. Biti otporan na pogreške u slučaju da se pojedini programski kodovi studenata ne mogu prevesti. Pogreška prevođenja manjeg dijela rješenja ne smije uzrokovati pogrešku detekcije plagijata nad svim definiranim rješenjima za pojedinu provjeru znanja.
2. Biti skalabilan tako da posao vezan uz detekciju plagijata se može obavljati istodobno na više poslužitelja.

2.3. Model podataka

2.3.1. Korištene baze podataka

Za potrebe sustava izabrane su dvije baze podataka: MongoDB i Redis.

MongoDB [19] je NoSQL baza podataka koja podatke pohranjuje u obliku kolekcije gdje svaka kolekcija ima više dokumenata. Podatci se interno pohranjuju u BSON obliku (binary JSON) te svaki dokument izgleda kao JSON objekt. Mongo spada u tip baze podataka koje nemaju striktnu shemu, dobro se nosi s velikim količinama podataka te se lako skalira. Mongo baza podataka se činila kao logičan izbor s obzirom na to da rezultati koje JPlag alat vrati nakon provedene detekcije plagijata su veliki JSON objekti. SQL baze podataka sa striktnom shemom nisu pogodne za pohranu ovakvih podataka. Ako je u trenutačnoj akademskoj godini na nekoj provjeri znanja sudjelovalo n studenata, onda ukupan broj parova koje JPlag generira je $\binom{n}{2}$. Broj parova raste ako se studentska rješenja trenutačne akademske godine uspoređuju s onima iz prošlih akademskih godina. Ako studentskih rješenja iz prethodnih akademskih godina ima m, onda ukupan broj parova koji se moraju provjeriti je $\binom{n}{2} + n * m$. Važno je napomenuti da se rješenja iz prethodnih akademskih godina ne uspoređuju međusobno, što značajno ubrzava proces provjere. Provjera plagijata koja je obuhvatila 143 studentskih rješenja iz trenutačne akademske godine, bez prethodnih akademskih godina, rezultirala je s 10153 JSON objekta koje je potrebno pohraniti radi kasnije vizualizacije. Predmeti na početnim godinama studija imaju i preko 500 studenata, što za Mongo ne predstavlja problem.

Redis [20] je izuzetno brza baza podataka koja sve podatke čuva u RAM-u. Podatke pohranjuje u obliku parova ključ-vrijednost. Jedna od mogućnosti napravljenog servisa je uživo praćenje napretka detekcije plagijata. Korisnik u svakom trenutku putem web-aplikacije zna koja studentska rješenja se trenutačno prevode, normaliziraju ili međusobno uspoređuju. S obzirom na to da rješenja ima puno te se pojedini koraci odvijaju paralelno za očekivati je da će opterećenje na bazu podataka biti veliko. Inicijalno je, za potrebe podataka koji moraju biti dostupni tijekom provedbe detekcije plagijata, bila korištena baza Mongo, ali testiranjem na velikom broju rješenja utvrđen je značajan pad performansi sustava do te mjere da se detekcija nije mogla do kraja provesti bez greške. Redis bez problema podržava traženi broj operacija u sekundi, sve operacije izvršava sekvencijalno na jednoj dretvi te nema problema s time da više dretvi servisa istovremeno ažurira istu kolekciju.

2.3.2. Podatkovni entiteti za definiranje konfiguracije predmeta

Kako bi korisnik mogao obaviti provjeru plagijata potrebno je napraviti konfiguraciju koja je povezana s predmetom u sustavu Edgar. Svaka konfiguracija ima atribute navedene u tablici 1.

Ime atributa	Podatkovni tip atributa	Opis atributa
id	String	Identifikator konfiguracije generiran od strane Mongo baze.
name	String	Naziv konfiguracije.
subjectName	String	Naziv predmeta.
resourcesYears	Long	Broj akademskih godina za koje će se preuzeti studentska rješenja sa sustava Edgar i koristiti za provjeru plagijata.
projects	List<StudentProject>	Projekti vezani uz predmet, opisani u tablici 2.
scheduleType	ScheduleType	Enumeracija koja označava učestalost ažuriranja studentskih rješenja s vrijednostima opisanim u tablici 3.
userName	String	Naziv korisnika koji je definirao konfiguraciju.
suffixes	List<String>	Popis nastavaka datoteka koje se trebaju uzimati u obzir prilikom detekcije plagijata.
languages	List<String>	Popis programskih jezika u kojima su studentska rješenja implementirana, može uključivati i prirodni jezik.
lastUpdateTime	Date	Vrijeme zadnjeg ažuriranja studentskih rješenja.

subjectId	String	Identifikator predmeta pomoću kojeg se konfiguracija referencira na predmet u sustavu Edgar.
disallowedFiles	List<String>	Naziv datoteka i direktorija koji se ne uzimaju u obzir prilikom detekcije plagijata u pojedinim studentskim rješenjima.

Tablica 1. Podatkovni model konfiguracije predmeta

Ime atributa	**Podatkovni tip atributa**	**Opis atributa**
name	String	Naziv provjere znanja.
resources	List<Resources>	Studentska rješenja koja će biti korištena za detekciju plagijata, opisani u tablici 4.
taskText	String	Tekst zadatka koji će se predati velikom jezičnom modelu za generiranje rješenja te time detektirati plagijate nastale korištenjem umjetne inteligencije.
resultIds	List<String>	Nasumično generirani identifikatori rezultata usporedbe, služe kao strani ključevi prilikom dohvata svih rezultata usporedbe, opisanih u tablici 7, za trenutnu provjeru znanja.

Tablica 2. Podatkovni model provjere znanja na predmetu

Naziv enumeracije	Opis
DAILY	Studentska rješenja se ažuriraju svaki dan.
WEEKLY	Studentska rješenja se ažuriraju svaki tjedan.
EVERY_TWO_WEEKS	Studentska rješenja se ažuriraju svaki drugi tjedan.
MONTHLY	Studentska rješenja se ažuriraju svaki mjesec.

Tablica 3. Enumeracija koja definira učestalost ažuriranja studentskih rješenja

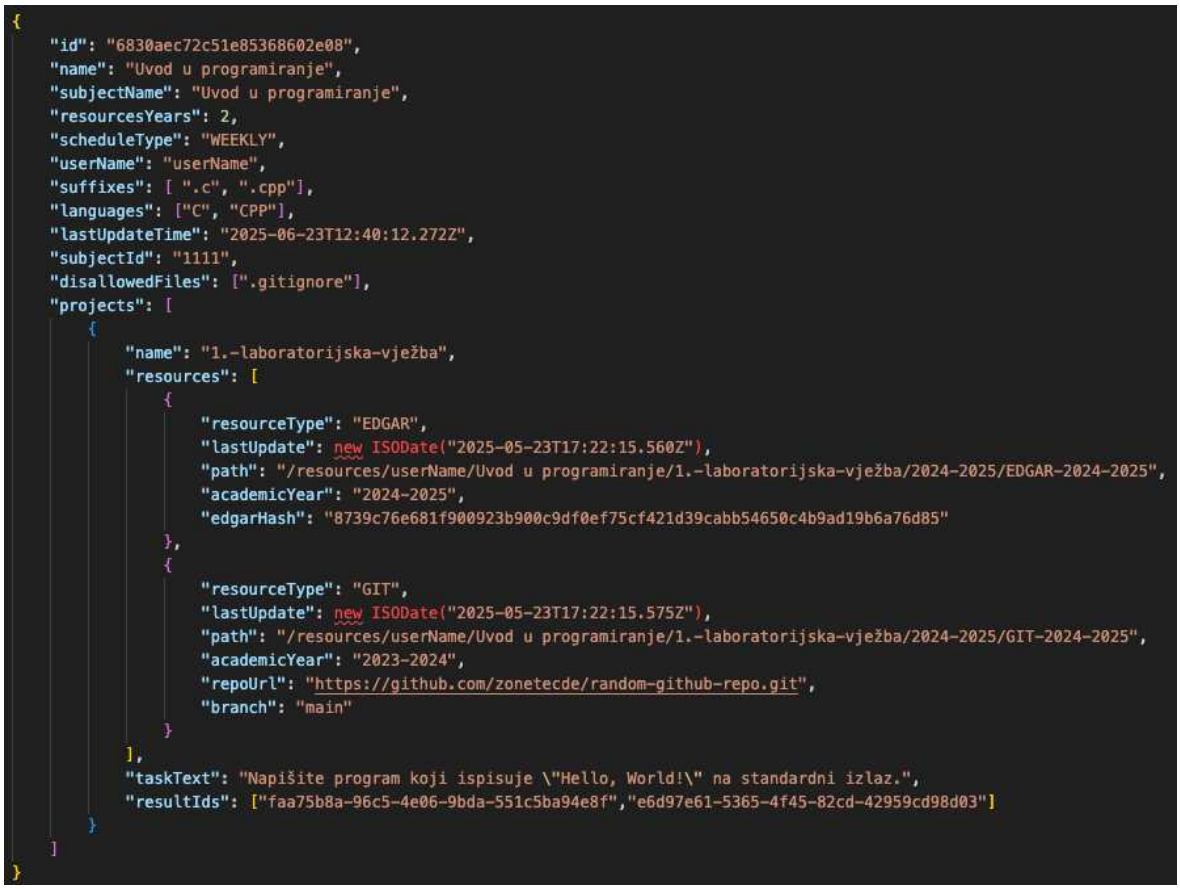
Ime atributa	Podatkovni tip atributa	Opis atributa
resourceType	ResourceType	Enumeracija koja označava o kakvom tipu resursa je riječ, opisano u tablici 5.
path	String	Putanja na kojoj se resurs nalazi na disku, koristi se za sve tipove resursa osim za one koji se dohvaćaju s Edgara.
academicYear	String	Akademska godina kojoj resurs pripada.
edgarHash	String	Sažetak resursa, jedinstveno identificira resurs na Edgaru te se koristi prilikom preuzimanja resursa s Edgara.
repoURL	String	Služi kao poveznica na GitHub ili GitLab repozitorij ako je riječ o takvom tipu resursa.
branch	String	Naziv grane na GitHub ili GitLab repozitoriju koja će se koristiti za preuzimanje resursa.

Tablica 4. Podatkovni model resursa korištenog prilikom detekcije plagijata

Naziv enumeracije	Opis
EDGAR	Resurs se preuzima s Edgara.
GIT	Resurs se preuzima s GitHuba ili GitLaba.
ZIP	Resurs korisnik prenosi u obliku ZIP arhive.
WHITELIST	Resurs se koristi kao programski kod koji se ignorira prilikom detekcije plagijata.
AI	Rješenje generirano uz pomoć umjetne inteligencije za detekciju studentskih rješenja napravljenih uz pomoć velikih jezičnih modela.

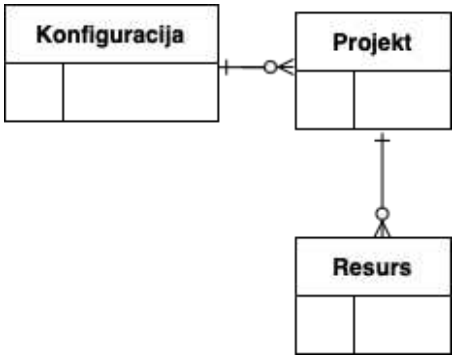
Tablica 5. Enumeracija tipova resursa

Nakon opisanih svih podatkovnih modela, primjer jedne konfiguracije u JSON obliku dostupan je na slici 27.



Slika 27 Primjer konfiguracije za predmet u JSON obliku

Iz ovih tablica proizlazi da se konfiguracija zapravo odnosi na predmet koji je upisan u sustavu Edgar. Svaka konfiguracija može imati više projekata koji su zapravo provjere znanja te svaki projekt ima više resursa, odnosno studentskih rješenje prikupljenih iz raznih izvora. Bitno je napomenuti da se jedan resurs odnosi na više pojedinačnih studentskih rješenja te da su grupirani ovisno o izvoru resursa. Dijagram relacijskih odnosa vidljiv je na slici 28.



Slika 28. Dijagram relacijskih odnosa entiteta za konfiguraciju detekcije plagijata

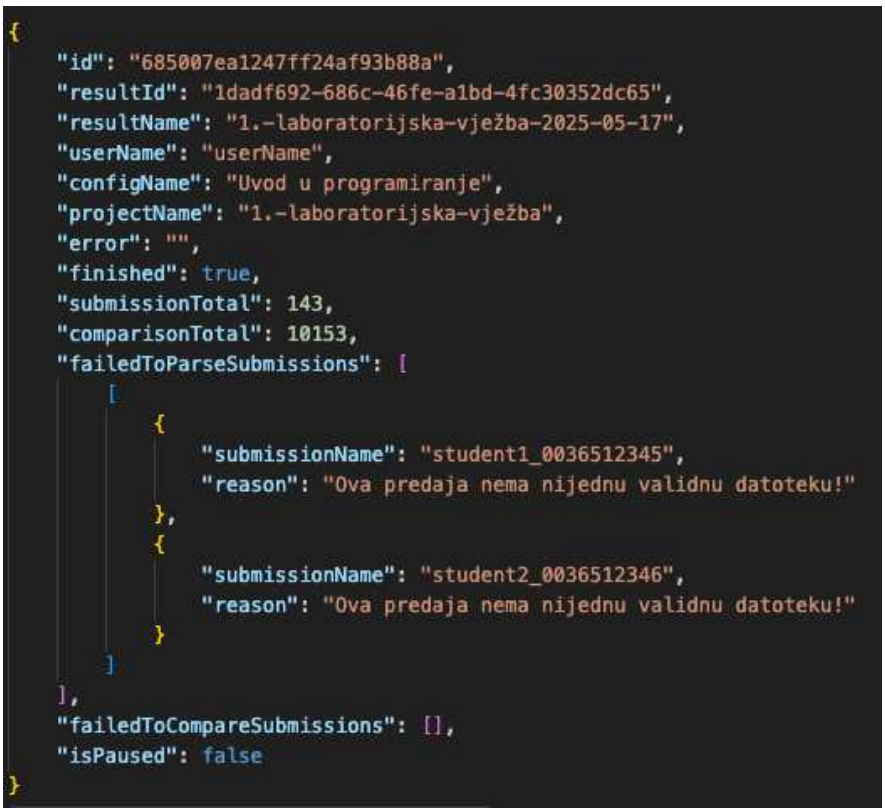
2.3.3. Podatkovni entiteti za pohranu JPlag rezultata

JPlag alat u svojoj originalnoj verziji rezultate pohranjuje u ZIP arhivu koja ima puno JSON datoteka. Svaka JSON datoteka odgovara jednom studentskom paru koji je uspoređen te sadrži podatke o sličnosti kao što su početak i kraj sličnih dijelova koda. Ove JSON datoteke se koriste za prikaz rezultata usporedbe vidljivog na slici 4. Osim toga, JPlag generira jednu glavnu JSON datoteku koja sadrži podatke o svim postotnim sličnostima između studentskih rješenja te se ta datoteka koristi za prikaz vidljiv na slici 3. Sustav u kojem bi korisnik morao preuzimati ZIP arhive te ih kasnije prenositi na WEB aplikaciju za prikaz rezultata nije prihvatljiva opcija. JSON datoteke koje JPLag generira su dugačke i teške za razumjeti, stoga ovaj sustav pamti JPlag rezultat nastao prije samog generiranja JPlag JSON datoteka. Objekti koje sustav pamti veliki su JSON objekti, no jednostavnijeg oblika i potpuno prilagođeni za kasniji prikaz rezultata u web aplikaciji. Svaki put kad se pokrene detekcija plagijata sustav pamti detalje o opcijama s kojima je JPlag pokrenut u obliku objekta opisanog u tablici 6.

Ime atributa	Podatkovni tip atributa	Opis atributa
id	String	Identifikator generiran od strane Mongo baze.
resultId	String	Nasumično generiran identifikator koji se koristi za dohvat rezultata usporedbe.
resultName	String	Naziv pokrenute detekcije plagijata, u slučaju automatske detekcije plagijata nasumično generiran uključujući vrijeme pokretanja.
userName	String	Naziv korisnika za kojeg se pokreće detekcija plagijata.
configName	String	Naziv konfiguracije za koju se pokreće detekcija plagijata.
projectName	String	Naziv projekta za kojeg se pokreće detekcija plagijata.
error	String	Poruka u slučaju pogreške prilikom detekcije plagijata.
finished	Boolean	Zastavica za indikaciju završenosti detekcije plagijata.
submissionTotal	Int	Ukupan broj studentskih rješenja.
comparisonTotal	Int	Ukupan broj uspoređenih studentskih parova rješenja.
failedToParseSubmissions	List<String>	Nazivi studentskih rješenja koje JPlag nije mogao prevesti.
failedToCompareSubmissions	List<String>	Nazivi parova studentskih rješenja koja se nisu mogla međusobno usporediti.
isPaused	Boolean	Zastavica koja je istinita u slučaju pauzirane detekcije plagijata.

Tablica 6. Podatkovni model objekta za spremanje detalja pokrenute detekcije plagijata

JSON primjer objekta za pohranu detalja pokrenute detekcije plagijata dostupan je na slici 29.



Slika 29 Primjer JSON objekta za pohranu detalja JPlag konfiguracije završene detekcije plagijata

Za svaku završenu detekciju plagijata u Mongo bazu se sprema objekt opisan u tablici 7.

Ime atributa	**Podatkovni tip atributa**	**Opis atributa**
id	String	Identifikator generiran od strane Mongo baze.
resultId	String	Nasumično generiran identifikator rezultata usporedbe, referenciran iz objekta opisanog u tablici 6.
topComparisonIds	List<String>	Referenca na identifikatore objekta tipa Comparison, koristi se za prikaz rezultata usporedbe prikazanih na slici 3.

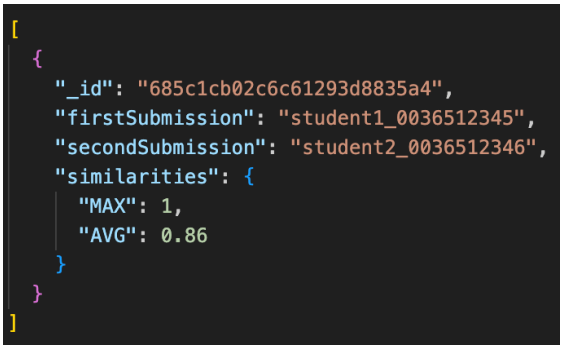
clusterIds	List<String>	Referenca na objekte tipa Cluster koji sadrže sve podatke o studentima koji su zajedno surađivali na projektu.
submissionComparisonIds	List<String>	Referenca na objekte tipa SubmissionComparison koji sadrže sve podatke nužne za prikaz rezultata usporedbe prikazanih na slici 4.
submissionFileIndex	SubmissionFileIndex	Objekt koji za svaku datoteku studentskih rješenja sadrži broj JPlag generiranih tokena. Koristi se za izračun postotne sličnosti.

Tablica 7. Podatkovni model objekta korištenog za pohranu rezultata usporedbe

Primjer JSON objekta koji se koristi za pohranu rezultata usporedbe dostupan je na slici 30.

Slika 30 Primjer JSON objekta za pohranu rezultata detekcije plagijata

Objekti tipa *Comparison* koriste se za prikaz sa slike 3, a primjer jednog takvog objekta dostupan je na slici 31. Objekti tipa *Cluster* koriste se za prikaz grupiranja sa slike 5, a primjer jednog takvog objekta dostupan je na slici 32. Objekti tipa *SubmissionComparison* koriste se za pohranu informacija o dijelovima koda koji su slični u dva studentska rješenja te uključuje informacije kao što su početak i kraj linija i stupaca koda koji imaju iste tokene. Primjer ovog objekta vidljiv je na slici 33.



Slika 31 Primjer JSON objekta *Comparison*

Slika 32 Primjer JSON objekta *Cluster*

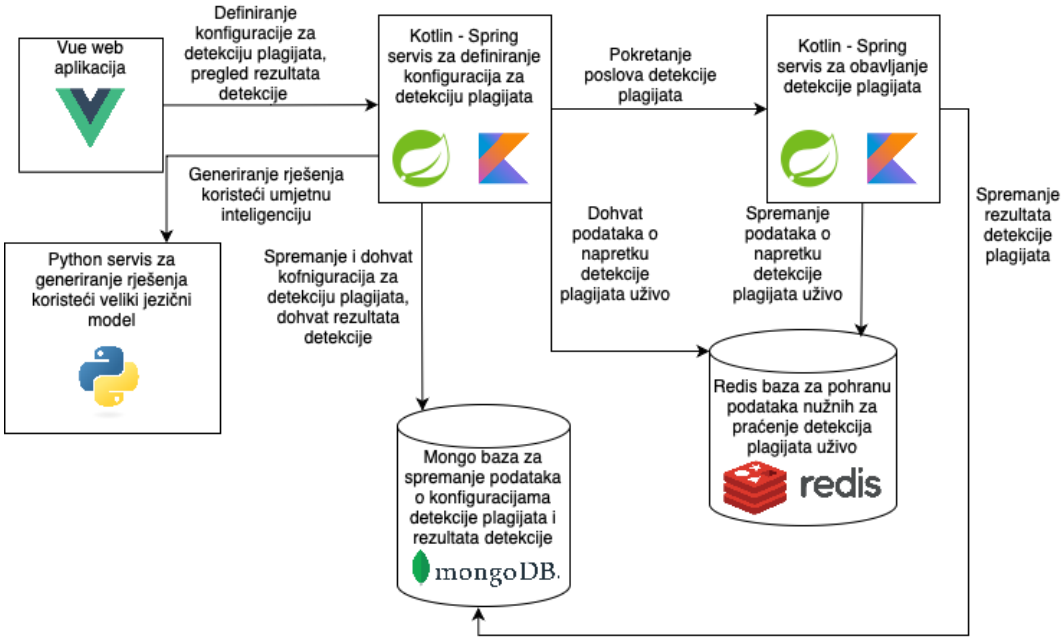
Svi dosad navedeni podatci spremaju se u Mongo bazu. Jedna od funkcionalnosti servisa je mogućnost praćenja napretka detekcije plagijata. Korisniku su dostupni podatci o tome koja studentska rješenja se trenutačno prevode, normaliziraju ili međusobno uspoređuju. Za ove potrebe korišten je Redisov skup (engl. set) tip podatka, a kao ključ pristupa koristi se jedinstveni identifikator svake detekcije plagijata, *resultId.*

```
[
  {
    "_id": "685c1f158ce7725e3d0856b1",
    "firstSimilarity": 0.05803571428571429,
    "firstSubmissionId": "student1_0036512345",
    "firstSubmissionPath": "resources/userName/Operacijska istraživanja/1.-laboratorijska-vježba/2024-2025/EDGAR-2024-2025/student1_0036512345",
    "matches": [
      {
        "firstFileName": "student1_0036512345/upload_0_2labos/2labos/server.js",
        "secondFileName": "student2_0036512346/upload_0_lab2_andrej_lovei/main map/server.js",
        "startInFirst": {
          "lineNumber": 9,
          "column": 6,
          "tokenListIndex": 14
        },
        "endInFirst": {
          "lineNumber": 28,
          "column": 11,
          "tokenListIndex": 26
        },
        "startInSecond": {
          "lineNumber": 6,
          "column": 6,
          "tokenListIndex": 11
        },
        "endInSecond": {
          "lineNumber": 16,
          "column": 8,
          "tokenListIndex": 23
        },
        "tokens": 13
      }
    ],
    "resultName": "816f4ba6-cfc1-49ba-8551-639ece04fd6d",
    "searchKey": "student1_0036512345-student2_0036512346",
    "secondSimilarity": 0.0327455919395466,
    "secondSubmissionId": "student2_0036512346",
    "secondSubmissionPath": "resources/userName/Operacijska istraživanja/1.-laboratorijska-vježba/2024-2025/EDGAR-2024-2025/student2_0036512346",
    "similarities": {
      "MAX": 0.05803571428571429,
      "AVG": 0.04186795491143317
    }
  }
]
```


Slika 33 Primjer JSON objekta *SubmissionComparison*

2.4. Arhitektura i pregled funkcionalnosti sustava

Implementirani sustav sastoji se od pet glavnih komponenti čija su povezanost i suradnja opisani dijagramom na slici 34.



Slika 34. Arhitekturni dijagram sustava

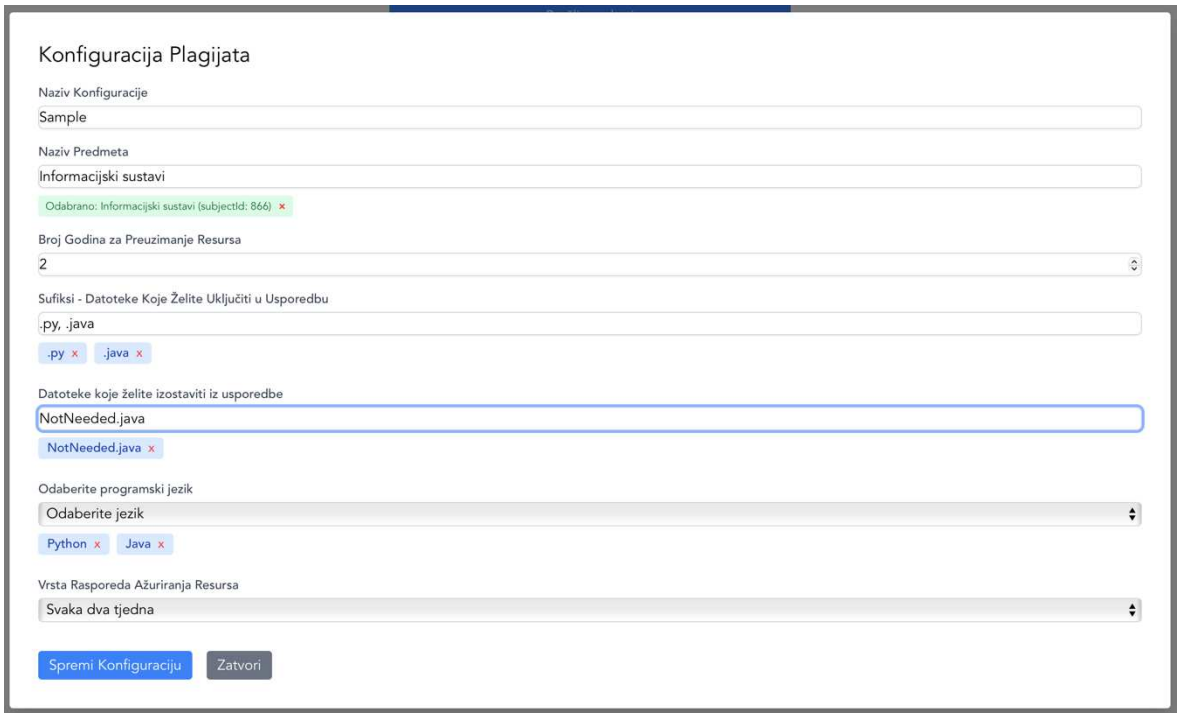
Za implementaciju poslužiteljske logike odabran je razvojni okvir Spring Boot [21] u kombinaciji s programskim jezikom Kotlin [22]. Spring Boot je razvojno okruženje pogodno za implementaciju poslužiteljskih aplikacija i mikro servisa. Većina stvari nužnih za implementaciju REST API-ja dostupna je bez dodatnih konfiguracija, a ima i dobru integraciju s bazama Mongo i Redis. Kotlin je programski jezik koji pruža odličnu integraciju sa Springom, a izvršava se na Javinom virtualnom stroju (engl. *Java Virtual Machine* - JVM). Prednosti Kotlina u odnosu na Javu su: razne sintaksne pokrate, manje šablonskog koda za pisanje i *null* sigurnost. Kotlin je u potpunosti interoperabilan s Javom tako da može pozivati metode implementirane u Javi. Ova mogućnost je ključna za pozivanje JPlag metoda koje su implementirane u Javi.

Generiranje rješenja s pomoću velikih jezičnih modela napravljeno je koristeći Python servis i FastAPI [24] biblioteku koja pruža mogućnost jednostavnog stvaranja REST API pristupnih točki. Općenito, za sve potrebe rada s velikim jezičnim modelima Python je u većini slučajeva najbolji odabir zbog velike podrške i količine biblioteka koje olakšaju

posao. Za potrebe ovog sustava korištena je OpenAI biblioteka [25] pomoću koje se ostvaruje generiranje rješenja.

Za izgradnju web aplikacije korišten je JavaScript razvojni okvir Vue.js [23]. Vue.js se koristi za izgradnju jednostraničnih aplikacija. Omogućava laku implementaciju navigacije u aplikaciji pomoću Vue Routera, reaktivan je te omogućava dizajniranje atraktivnih korisničkih sučelja bez puno truda. Osim navedenih kvaliteta, vizualizacija rezultata detekcije plagijata u JPlagu, koja je djelomice preuzeta, implementirana je u Vue.js okruženju.

Središnji servis, u daljnjem tekstu upravljački servis, ovog sustava zadužen je za upravljanje konfiguracijama za detekciju plagijata te pokretanje poslova detekcije plagijata. Stvaranje nove konfiguracije detekcije plagijata korisniku je dostupno u web aplikaciji putem sučelja prikazanog na slici 35. Prilikom stvaranja konfiguracije korisnik mora unijeti sve potrebne podatke vidljive u formi, a opisane u tablici 1.



Slika 35. Sučelje za stvaranje nove konfiguracije za detekciju plagijata

Čim se stvori konfiguracija detekcije plagijata, upravljački servis kreće preuzimati sve resurse studentskih rješenja sa sustava Edgar. Komunikacija s Edgarom odvija se u dva koraka. Prvi korak je dohvat popisa svih projekata na predmetu za trenutačnu akademsku godinu. Projekti uključuju laboratorijske vježbe, provjere znanja i domaće zadaće. Na

osnovu ovog popisa stvara se lista projekata kojom se konfiguracija nadopunjuje. Edgar servis osim imena projekata vraća i *hash* s pomoću kojeg je moguće preuzeti sva studentska rješenja kao ZIP arhivu za pojedinačni projekt i akademsku godinu. Dohvat popisa projekata odvija se na temelju identifikatora predmeta i akademske godine. Servis spaja resurse studentskih rješenja na osnovu naziva projekta, odvajajući pritom rješenja trenutačne godine i prethodnih akademskih godina. Upravljački servis zbog vizualizacije rezultata mora imati preuzeta studentska rješenja iako ne obavlja detekciju plagijata. Rješenja se pohranjuju u datotečni sustav organizirana po konfiguracijama i akademskim godinama. Nakon što je konfiguracija stvorena, moguće ju je dodatno urediti ručnim prijenosom novih studentskih rješenja, podešavanjem detekcije plagijata uz pomoć umjetne inteligencije ili brisanjem studentskih rješenja koja su se prenijela pogreškom. Prikaz detalja o konfiguraciji plagijata dostupan je na slici 36. Prozor koji se otvara kada korisnik želi prenijeti dodatne izvore studentskih rješenja osim onih već preuzetih s Edgara vidljiv je na slici 37. Korisnik ima mogućnost prenijeti nova rješenja tako što prenese ZIP arhivu ili unese poveznicu na GitHub ili GitLab repozitorij. Oba načina prijenosa zahtijevaju odabir akademske godine tijekom koje su rješenja nastala te organizaciju rješenja na način da svakom studentu pripada jedan direktorij unutar vršnog direktorija. Korisnik također može i generirati rješenje koristeći OpenAI GPT-4o mini veliki jezični model. U tom slučaju upravljački servis poziva Python servis koristeći REST API pristupnu točku. Rješenje koje je generirala umjetna inteligencija pohranjuje se kao regularno rješenje studenta koji pohađa trenutačnu akademsku godinu. Usporedbom ovog rješenja s rješenjima ostalih studenata, otkrivaju se studenti koji su za rješavanje zadaće potencijalno koristili velike jezične modele. Ponekad, u slučaju predane ZIP arhive krivog formata, predajom poveznice na repozitorij koji je privatan ili nepostojeći te u slučaju nedostupnosti Edgar servisa, resurse za detekciju plagijata neće biti moguće ažurirati. Korisnik je o tome obaviješten ikonom žutog uskličnika iznad kojeg kad se prijeđe mišem piše detaljnija poruka o pogrešci. Primjer ovog slučaja vidljiv je na slici 36. U unaprijed zakazanom terminu ažuriranja resursa za neki projekt, sustav će za sve resurse, osim naravno ZIP arhiva, pokušati ponoviti preuzimanje nadajući se da je potencijalni kvar drugih servisa u međuvremenu otklonjen.

Slika 36. Prikaz detalja konfiguracije plagijata

Slika 37. Prikaz prozora za prijenos novih resursa za detekciju plagijata

Svrha sustava je detekciju plagijata u potpunosti automatizirati. Posao detekcije plagijata je resursno zahtjevan te se zbog toga detekcija pokreće svaki dan u ponoć i prestaje u pet sati ujutro. Imati rezultate detekcije plagijata što prije nije od kritične važnosti, a pokretanjem

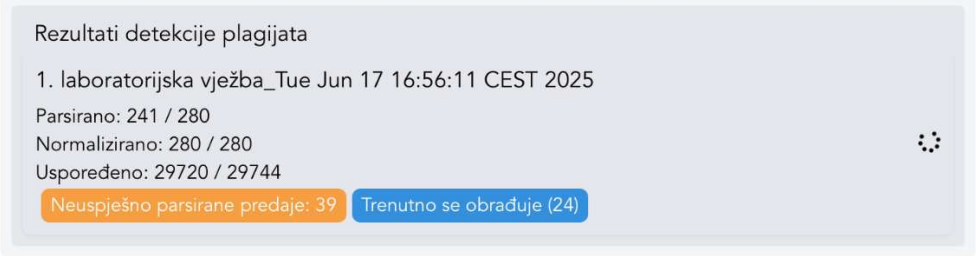
noću omogućava se instalacija sustava na poslužitelj koji preko dana ima i drugu svrhu te bi ga posao detekcije plagijata ometao.

Detekcija plagijata se pokreće u sljedećim situacijama:

a) u ponoć, onog dana kada je konfiguracija stvorena
b) u ponoć, nakon preuzimanja studentskih rješenja za novi projekt (u slučaju provedene nove provjere znanja, laboratorijske vježbe i sličnog)
c) ručno, po želji korisnika pritiskom na odgovarajući gumb na korisničkom sučelju, ali samo u slučaju kada se posao detekcije plagijata za taj projekt već ne obavlja.

Osvježavanje studentskih rješenja za pojedine projekte ostvareno je koristeći Springovu CronJob funkcionalnost. Na ovaj način definiran je posao koji se pokreće svakog dana u ponoć te provjerava zadnje vrijeme ažuriranja rješenja za pojedine konfiguracije. Ako je vrijeme proteklo od zadnjeg ažuriranja rješenja, veće ili jednako od vremena kojeg je korisnik definirao kao učestalost ažuriranja rješenja, pokreće se novo ažuriranje rješenja za taj projekt.

Za vrijeme trajanja detekcije plagijata korisniku je dostupno uživo praćenje napretka posla. JPlag proces sastoji se od tri glavna koraka: prevođenje, normalizacija i usporedba parova studentskih rješenja. Korisnik u svakom trenutku vidi napredak pojedinog koraka, kao i nazive studentskih rješenja koja se trenutačno obrađuju u obliku liste. Popis imena rješenja koja se trenutačno obrađuju upravljački servis dohvaća iz Redis baze, a brojke o napretku procesa iz Mongo baze. Praćenje napretka JPlag procesa vidljivo je na slici 38. Mnogi studenti predaju rješenja u krivom formatu, rješenja koja se ne mogu prevesti ili pak prazna rješenja. Sustav je dovoljno robustan da takve iznimke ignorira, a korisnik će znati o kojim se rješenjima radi pomoću liste vidljive na slici 39. Ova rješenja korisnik bi trebao ručno pregledati te pronaći uzrok neuspješnog prevođenja.



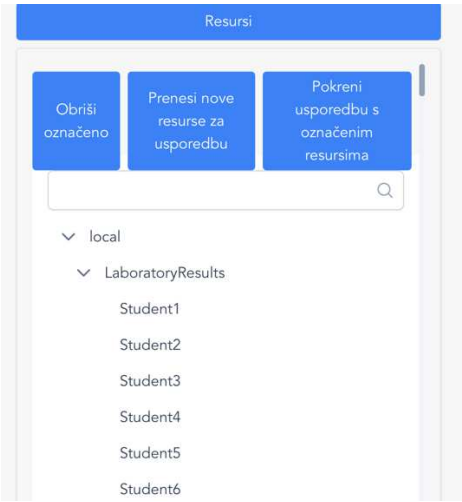
Slika 38. Uživo praćenje napretka posla detekcije plagijata

Slika 39. Lista neuspješno prevedenih studentskih rješenja

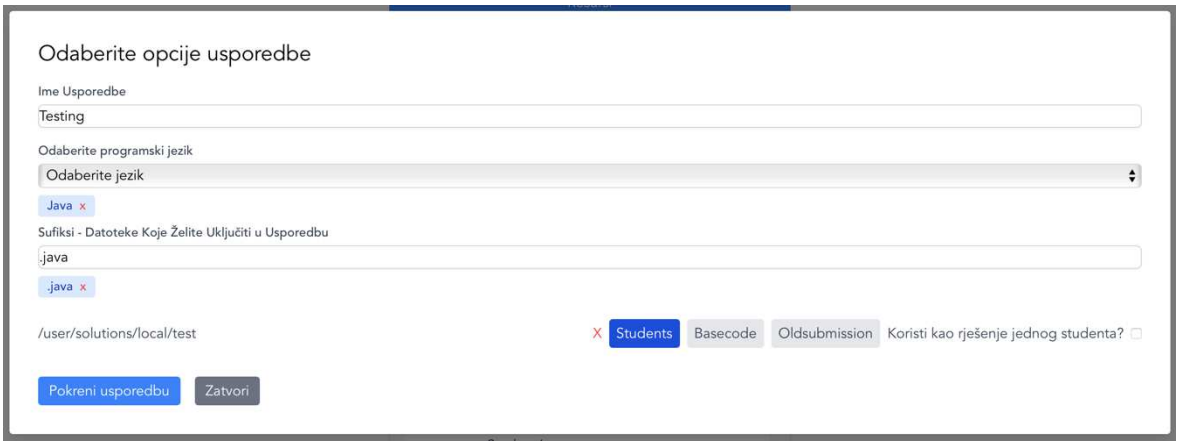
Pritiskom na rezultat detekcije plagijata korisniku se otvara prikaz sa slike 3. Nastavnicima se na raspolaganje stavlja mogućnost filtriranja rezultata po imenima studenata, ali i koristeći kombinacije različitih imena povezujući ih s operatorima AND i OR. Većina studentskih rješenja će imati barem malo sličnosti s ostalim rješenjima što ne znači da je riječ o plagijatu. Kako bi se skratilo vrijeme potrebno za analizu rezultata detekcije plagijata, nastavnici mogu filtrirati rezultate po postotnoj sličnosti. Tako je na primjer moguće pregledavati samo one rezultate kojima je sličnost veća od 80 %. Za potrebe prikaza rezultata detekcije iskorištena je postojeća JPlag web aplikacija uz određene promjene. JPlag aplikacija očekuje prijenos ZIP arhive koja u sebi ima već prethodno opisane JPlag JSON objekte. Za potrebe ovog sustava, logika web aplikacije je izmijenjena na način da web aplikacija sve rezultate dohvaća preko upravljačkog servisa iz Mongo baze. Rezultati se interno referenciraju koristeći *resultId* spomenut u dijelu 2.3.3. Podatci o sličnostima i broju tokena dohvaćaju se iz Mongo baze, a datoteke studentskih rješenja koja se koriste za prikaz sa slike 4, upravljački servis vraća u obliku ZIP arhive. Dohvat datoteka odvija se po potrebi, tek nakon što korisnik pritisne na detaljniji pregled rezultata detekcije između dva studenta. Ova funkcionalnost razlog je zašto upravljački servis pohranjuje studentska rješenja iako ne obavlja detekciju plagijata.

Za potrebe testiranja detekcije plagijata, korisniku je na raspolaganju pokretanje detekcije plagijata bez povezivanja s Edgarom i stvaranja konfiguracija za predmete. Na slici 40 vidi se korisničko sučelje preko kojeg korisnik definira skup studentskih rješenja za provjeru plagijata. U ovom slučaju korisnik može prenijeti rješenja u obliku ZIP arhive ili predati poveznicu na GitHub ili GitLab repozitorij. Jednom kad korisnik definira skup rješenja,

provjeru pokreće preko sučelja vidljivog na slici 41 gdje se odabire konfiguracija s kojom se JPlag pokreće.



Slika 40. Sučelje za prijenos studentskih rješenja bez povezivanja s Edgarom



Slika 41. Sučelje za pokretanje JPlag detekcije plagijata bez povezivanja s Edgarom

Pregled rezultata detekcije funkcionira na isti način kao i kod pregleda rezultata za neku konfiguraciju predmeta.

2.4.1. Posao detekcije plagijata

Upravljački servis implementira cijelu logiku vezanu uz konfiguracije i pokretanje detekcije plagijata u točno određenom vremenu. Posao detekcije plagijata upravljački servis prepušta drugom servisu, u daljnjem tekstu JPlag servis, vidljivom na slici 34. Nakon što upravljački servis utvrdi da je potrebno pokrenuti detekciju plagijata, na temelju uvjeta opisanih u prethodnom poglavlju, JPlag servisu se pomoću REST API-ja šalju sve potrebne informacije da obavi detekciju plagijata. Objekt kojim se definira posao detekcije plagijata opisan je u tablicama 8 i 9.

Ime atributa	Podatkovni tip atributa	Opis atributa
studentSolutions	List<JPlagResource>	Definicija resursa koji će se koristiti prilikom detekcije plagijata, opisano u tablici 9.
languages	List<String>	Programski jezici za koje će se obaviti detekcija plagijata, uključujući i prirodni jezik.
resultId	String	Identifikator rezultata detekcije plagijata.
fileSuffixes	List<String>	Nastavci datoteka koje će se uzeti u obzir prilikom detekcije plagijata.

Tablica 8. Objekt kojim se definira posao detekcije plagijata

Ime atributa	Podatkovni tip atributa	Opis atributa
edgarHash	String	Sažetak koji se koristi za preuzimanje resursa s Edgara.
solutionPath	String	Parametar koji definira putanju s pomoću koje se resurs dohvaća ako nije riječ o Edgar resursu.
typeOfResource	TypeOfResource	Enumeracija koja označava pripadnost rješenja trenutnoj ili prethodnim akademskim godinama te je li riječ o resursu koji predstavlja kod koji se ignorira prilikom detekcije plagijata.

Tablica 9. Objekt kojim se definira resurs korišten za detekciju plagijata

Prije pokretanja detekcije plagijata, upravljački servis stvara nasumično generirani *resultId* koji se kasnije koristi za dohvat rješenja iz Mongo i Redis baze. Na temelju podataka o

konfiguraciji i projektu, upravljački servis stvara objekt definiran u tablici 8 te ga šalje JPlag servisu pomoću REST API-ja. Posao detekcija plagijata je asinkron, što znači da upravljački servis ne čeka JPlag servis da završi s poslom detekcije, a stupanj gotovosti upravljački servis provjerava na temelju parametra *resultId* koristeći baze Mongo i Redis. JPlag servis je jednostavan u smislu da ne poznaje detalje o konfiguracijama predmeta te služi isključivo kao radni servis za pokretanje JPlaga. Prije nego što krene s poslom detekcije, JPlag servis preuzima sve potrebne resurse. Resursi koji se nalaze na Edgar servisu preuzima pomoću parametra *edgarHash*, a resurse koje je korisnik ručno prenio kao GitHub ili GitLab repozitorije i ZIP arhive preuzima sa upravljačkog servisa. Sve rezultate detekcije plagijata JPlag servis zapisuje u Mongo i Redis baze. Nakon što je posao detekcije gotov, preuzeti resursi brišu se sa JPlag servisa. Ovakvom organizacijom JPlag servisa, postiglo se to da je moguće instalirati više neovisnih JPlag servisa na različitim računalima te na taj način skalirati sustav vrlo lako.

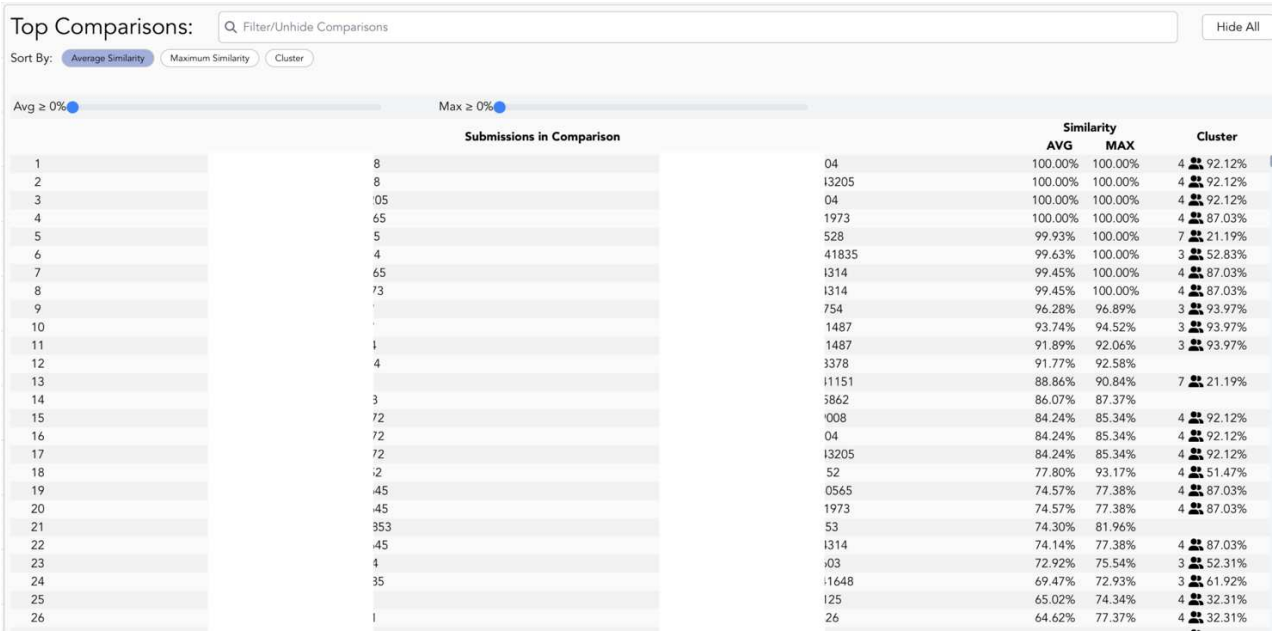
JPlag servis detekciju plagijata pokreće isključivo noću te radi do pet sati ujutro. Nakon pet sati ujutro, poseban proces, ostvaren Springovim CronJobom, koji se obavlja na JPlag servisu šalje signal za prekidom svih poslova detekcije plagijata. Prekid poslova nije nasilan već kontroliran što znači da ako se u trenutku primanja signala neka studentska rješenja prevode, normaliziraju ili međusobno uspoređuju, taj korak će se završiti, ali svi novi koraci neće biti pokretani. Sljedećeg dana u ponoć, svi prekinuti poslovi detekcije plagijata bit će ponovno pokrenuti. Ovo je omogućeno tako što upravljački servis ima poseban proces koji prati ako je neka detekcija plagijata prekinuta te šalje zahtjev za nastavkom posla. Kako bi bilo moguće prekidati i nastavljati procese usporedbe, JPlag biblioteka je modificirana što je opisano u dijelu 2.4.3. Svako studentsko rješenje u JPlag biblioteci predstavljeno je apstrakcijom *Submission.* Svaki *Submission* može se nalaziti u dva stanja: prevedenom ili normaliziranom. JPlag ova stanja pohranjuje kao enumeraciju unutar *Submission* objekta te mijenja ovisno o napretku procesa. U svaki korak procesa detekcije plagijata pojedinačni *Submission* može ući tek ako je njegovo prethodno stanje odgovarajuće. Ovo znači da se normalizacija mora obaviti nakon prevođenja te da se neko rješenje može usporediti s drugim tek ako je normalizirano. JPlag servis za svaki *Submission* pamti njegovo stanje te njemu pripadajuće generirane tokene i sve dosadašnje rezultate međusobnih usporedbi studentskih rješenja. Kada se proces ponovno pokrene JPlag servis učitava spremljeno stanje svih *Submission* objekta spremljenih u Mongo bazi te nastavlja izvršavati posao detekcije. Za svaki *Submission* provodi se odgovarajući sljedeći korak te ako je na primjer posao za neki

Submission prekinut nakon prevođenja, njegovo interno stanje označava da je sljedeći korak za taj *Submission* normalizacija. Isto tako prilikom međusobne usporedbe rješenja, preskaču se svi oni dosad uspoređeni parovi. Ovakvom implementacijom JPlag servis pravovremeno oslobađa zauzete procesorske i memorijske resurse te ih stavlja na raspolaganje ostalim procesima koji se odvijaju tijekom dana.

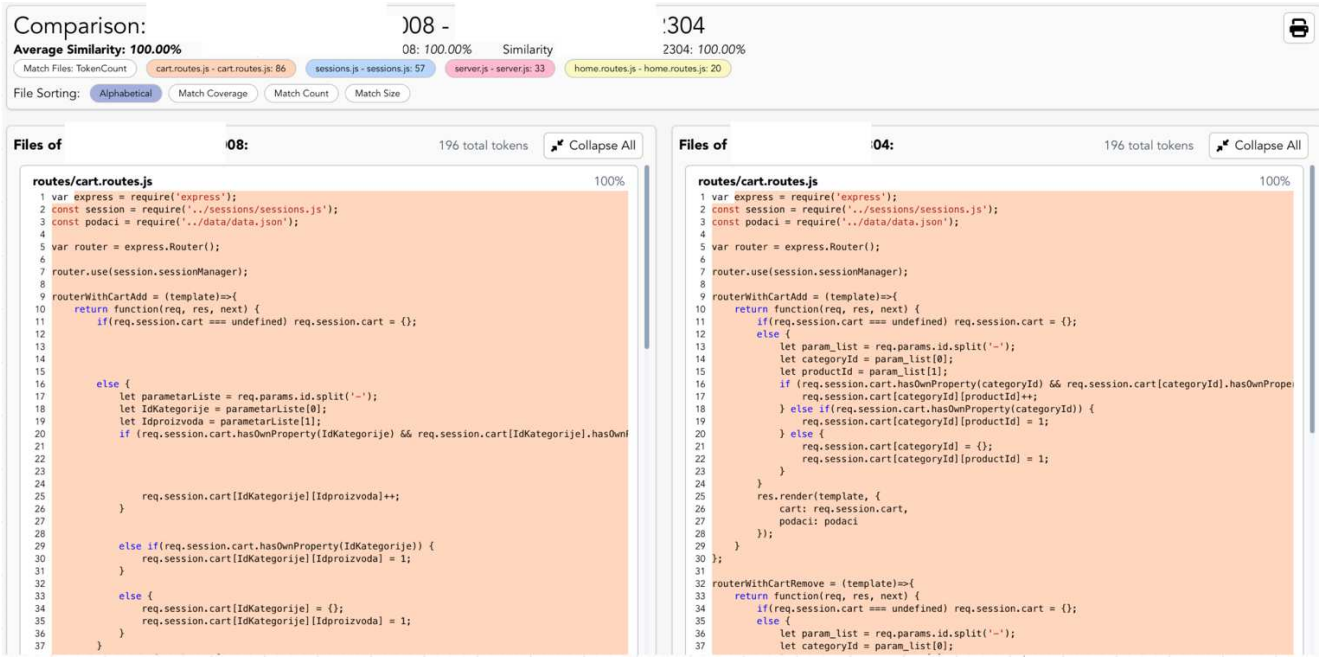
2.4.2. Primjer rada sustava na stvarnim skupu studentskih rješenja

U svrhu testiranja sustava koriste se stvarna studentska rješenja laboratorijske vježbe na predmetu Razvoj programske potpore za web. Predmet se predaje na drugoj godini preddiplomskog studija na Fakultetu elektrotehnike i računarstva. Svrha laboratorijske vježbe je studente upoznati s osnovnim tehnologijama za razvoj web aplikacija uključujući razvoj sučelja, servisa te modeliranje baze podataka. Odabrana tehnologija za razvoj servisa je Express.js [28] uz korištenje EJS [29] mehanizma za kombiniranje HTML i JavaScript koda. Studentska rješenja podijeljena su u dvije akademske godine tako da jedna akademska godina ima 113 rješenja, a druga 306. Rješenja koja su studenti predali organizirana su u obliku projekta s više datoteka organiziranih u direktorije bez ikakvog pravila imenovanja ili strukture direktorija. JPlag algoritam je s ovim primjerom bio gotov za nešto manje od 7 minuta na računalu s 16 GB radne memorije te procesoru s 10 jezgri frekvencije 3.2 GHz. Od ukupno 419 rješenja, njih 19 nije bilo moguće prevesti. Razlozi neuspjelog prevođenja u ovom slučaju su bila rješenja bez ijedne predane JavaScript datoteke, u potpunosti zakomentiran programski kod unutar datoteke te datoteke koje su u potpunosti prazne. Osim navedenih razloga, neuspjelo prevođenje se može dogoditi i kada student preda datoteku koja ima programski kod napisan tako da ne zadovoljava sintaksu programskog jezika. Pregled rezultata detekcije plagijata vidljiv je na slici 42. Sustav je detektirao 16 parova studenata s prosječnom sličnosti većom od 80 %, a njih 4 označio je kao potpuni plagijat sa sličnošću jednakom 100 %. Pritiskom na takav jedan par studentskih rješenja otvara se detaljniji prikaz koda za kojeg JPlag smatra da se podudara, a vidljiv je na slici 43. Na slici 44 vidljiva su dva studentska rješenja za koje JPlag tvrdi da imaju sličnost od 74.14 %. Iz slike se jasno vidi da su ova dva rješenja gotovo pa identična, ali na dva mjesta u kodu koja nisu obojena došlo je do prekida niza istih tokena te su na kraju detektirana tri podudarna bloka koda umjesto jednog velikog. Iz navedenih primjera može se zaključiti da bi nastavnici trebali pregledati što više parova studentskih rješenja, počevši od onih s najvećom sličnosti.

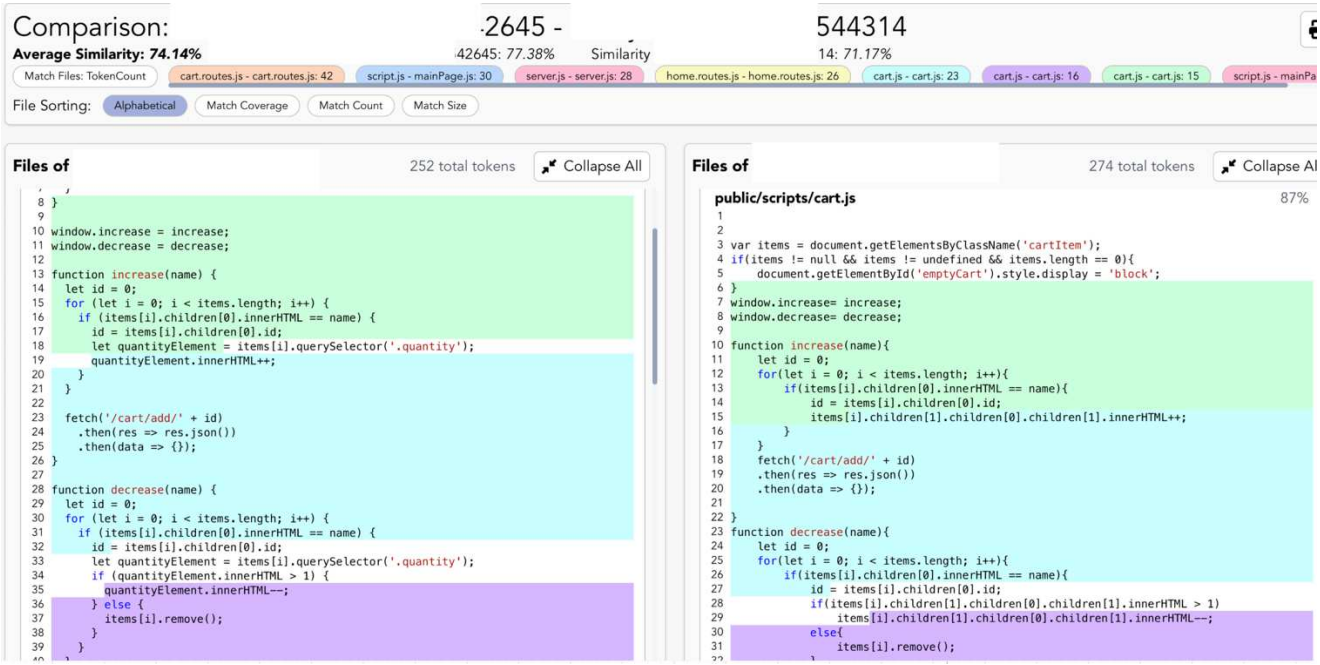
Iako parovi rješenja imaju sličnost manju od sličnosti koja bi odgovarala sigurno plagiranim rješenjima, moguće je da JPlag nije uspio nastaviti sekvencu tokena jer je student namjerno obfuscirao kod i izmijenio manje dijelove u cilju pokušaja izbjegavanja detektiranja.



Slika 42 Pregled rezultata detekcije plagijata na stvarnom programskom kodu



Slika 43 Pregled usporedbe dvaju studentskih rješenja sa sličnosti od 100 %



Slika 44 Pregled usporedbe dvaju studentskih rješenja sa sličnosti od 74.14 %

2.4.3. Modifikacije JPlag biblioteke za potrebe sustava

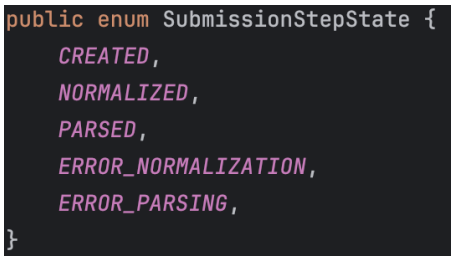
Funkcionalnost prekidanja i ponovnog pokretanja poslova detekcije plagijata ostvarena je uz minimalne modifikacije JPlag biblioteke. JPlag biblioteka je zamišljena da se koristi tako da se predaju svi konfiguracijski parametri i putanja do direktorija gdje se nalaze studentska rješenja nakon čega JPlag vrati rezultat na osnovu kojeg se može generirati vizualni pregled rezultata detekcije. Studentskih rješenja i datoteka može biti mnogo što za posljedicu ima veliko trajanje posla detekcije plagijata. Prilikom testiranja JPlaga na kompleksnim studentskim projektima s oko 1000 studentskih rješenja, nerijetko se dešava da JPlag traje i do pola sata. Servis koji pokreće JPlag u tom trenutku na raspolaganju mora imati što više računalnih resursa, a obavljanje nekih drugih kompleksnih zadataka, osim naravno u slučaju jako moćnog računala, nije moguće obavljati na efikasan način. Za potrebe ovog diplomskog rada odlučeno je da se posao detekcije obavlja isključivo tijekom noćnih sati kada su serveri na kojima je instaliran sustav pod najmanjim opterećenjem. Kako bi ovaj zahtjev mogao biti ostvaren, svi poslovi detekcije moraju se moći prekinuti i ponovno pokrenuti. JPlagova implementacija nije u skladu s ovime jer JPlag posao detekcije obavlja u komadu, a za potrebe ovog sustava nužno je ostvariti veću granularnost posla. Interno JPlagova implementacija sastoji se od tri glavna koraka: prevođenje, normalizacija i usporedba rješenja. Metode prevođenja i normalizacije označene su kao privatne metode razreda

Submission koji enkapsulira sve informacije jednog predanog studentskog rješenja. Ove metode promijenjene su u javne kako bi ih se moglo pozivati iz drugih paketa. Razred *Submission* pohranjuje i listu svih tokena koji čine studentsko rješenje kao i oznaku internog stanja *SubmissionState. SubmissionState* je enumeracija koja ima vrijednosti vidljive na slici 45. Ova enumeracija koristi se u originalnoj verziji JPlaga kako bi označila stanje u kojem se studentsko rješenje nalazi. S ovom enumeracijom moguće je označiti stanje rješenja koje može biti prevedeno, nije prevedeno ili prevođenje nije bilo moguće. U slučaju pokretanja JPlaga bez zaustavljanja ovo je sasvim dovoljno jer sve ostale korake moguće je obaviti nakon prevođenja. S obzirom da se prekid posla detekcije može dogoditi i između koraka normalizacije i međusobne usporedbe, bilo je potrebno osigurati dodatna stanja. Zbog ovog zahtjeva stvorena je nova enumeracija *SubmissionStepState* s vrijednostima na slici 46. koja je pridružena razredu *Submission,* a originalna enumeracija se obrisala iz upotrebe. Stanje se sada mijenja direktno iz JPlag servisa, a ne unutar JPlag biblioteke čime je ostvarena potpuna kontrola nad koracima detekcije. Svaki puta kada posao detekcije plagijata bude prekinut, studentska rješenja pohranjuju se u Mongo bazu s listom svih dosad međusobno uspoređenih rješenja. Pohranjena studentska rješenja imaju informacije o tome jesu li prevedena ili normalizirana kao i listu svih generiranih tokena. Prilikom ponovnog pokretanja posla odradi se mapiranje spremljenog objekta iz Mongo baze u onaj korišten u JPlagu te se za svako rješenje pokreće sljedeći korak ovisno o spremljenom stanju koje može biti takvo da je rješenje prevedeno te je sljedeći korak normalizacija ili je rješenje normalizirano pa je sljedeći korak usporedba ovog rješenja sa svim ostalim normaliziranim rješenjima. Osim navedenih glavnih stanja, moguće je i da se dogodila neka pogreška prilikom prevođenja ili normalizacije pa se to rješenje preskače u svim daljnjim koracima. Modifikacije JPlag biblioteke su minimalne te u slučaju prelaženja na novu verziju ne bi trebale predstavljati veliki problem prilikom migracije.

```
public enum SubmissionState {
    VALID,
    NOTHING_TO_PARSE,
    CANNOT_PARSE,
    TOO_SMALL,
    UNPARSED;

}
```

Slika 45. Originalna JPlag *SubmissionState* enumeracija



```
public enum SubmissionStepState {
    CREATED,
    NORMALIZED,
    PARSED,
    ERROR_NORMALIZATION,
    ERROR_PARSING,
}
```

Slika 46 *SubmissionStepState* enumeracija

3. Ocjena ostvarenog pristupa i smjernice za daljnji razvoj

Sustav koji je razvijen u sklopu ovog diplomskog rada nudi jednostavan način detekcije plagijata za one predmete gdje studenti rješenja svojih zadataka predaju putem sustava Edgar, ali moguće ga je koristiti tako da se poveže s nekim drugim sustavom sličnim Edgaru pod uvjetom da su rješenja koja se sa sustava preuzimaju organizirani na način opisan u dijelu 2. Osim toga, za potrebe testiranja sustav se može koristiti i bez povezivanja s drugim sustavima predajom rješenja u obliku ZIP arhiva ili *online* repozitorija. Kratkim unosom informacija o predmetu, nastavnici dobivaju rezultate detekcije plagijata bez ikakvih lokalnih instaliranja alata ili organiziranja studentskih rješenja. Odabrani alat, JPlag, za vrijeme pisanja ovog rada najbolji je alat otvorenog koda za detekciju plagijata. U periodu od prošle godine spominjan je u čak 143 znanstvena rada. Trenutačno se u slučaju pogreške prilikom prevođenja nekog rješenja korisniku prikaže poruka o pogrešci. Dobra nadogradnja sustava uključivala bi interaktivni preglednik koda studenta kako bi nastavnici mogli detaljno vidjeti problematično mjesto u projektu. Sustav također prati i napredak umjetne inteligencije uspoređujući studentska rješenja s onim kojeg je generirao veliki jezični model. Sve većim napretkom umjetne inteligencije i alata za obfuskaciju koda, pronalaženje plagijata postat će teže jer studenti neće morati prepisivati radove jedni od drugih, a raznolikost radova koje generiraju ovakvi modeli mogla bi biti velika do te mjere da alat smatra da između studenata uopće nije bilo prepisivanja. Područje koje se bori s detektiranjem koda napisanog s umjetnom inteligencijom u velikom je razvoju te zasad ovaj sustav se može unaprijediti dodavanjem što većeg broja različitih jezičnih modela za generiranje koda u nadi da će jedan od njih biti upravo onaj koji studenti koriste. Osim JPlaga, sustavu se zbog modularne organizacije lako može dodati i neki drugi alat za detekciju plagijata što bi dodatno poboljšalo sigurnost nastavnika prilikom donošenja odluke o plagijatima. Studenti rješenja zadataka često pronalaze na forumima i *online* grupama. Sustav bi se u budućnosti mogao nadograditi da resurse za usporedbu osim iz dosad navedenih izvora, traži i na internetu. Ovaj posao zahtijevao bi prikupljanje programskih kodova u raznim oblicima te njihovu pravilnu organizaciju. Iako je sustav trenutno visoko skalabilan, tako da je moguće imati više instanci JPlag servisa, i dalje se cijeli posao vezan za detekciju plagijata jednog projekta mora obaviti u cijelosti na jednoj instanci, s iznimkom u slučaju pauziranja detekcije gdje je moguće da se detekcija nastavi na drugoj instanci. Ovo

je napravljeno uzimajući u obzir da detekcija plagijata predanih studentskih zadaća nema kritičnu vremensku važnost te zbog mogućnosti stvaranja grupa sličnih rješenja. Ubrzanje je moguće tako da se JPlag ne pokreće sa svim studentskim rješenjima već uvijek samo s dva. U ovom slučaju gubi se mogućnost detektiranja grupa sličnih rješenja te bi parove za usporedbu bilo potrebno ručno generirati. Ubrzanje bi omogućilo pokretanje malih poslova detekcije plagijata od svega dva rješenja na puno različitih instanci JPlag servisa, a rezultate bi na kraju bilo potrebno ručno organizirati na isti način kao što to radi JPlag. Ovo ubrzanje moglo bi biti ugrađeno kao još jedna dodatna opcija prilikom pokretanja detekcije uz već postojeću implementaciju.

4. Zaključak

Detekcija plagijata u studentskim radovima od velike je važnosti za kvalitetu studijskog programa. Pravovremena detekcija plagijata nužna je nastavnicima sveučilišta kako bi dobili uvid u mogućnosti grupe studenata kojima predaju, njihov napredak, kvalitetu predavanja i pisanih provjera znanja. Čak i kad su sve navedene stvari na visokoj razini, nažalost uvijek se pronađu studenti koji odluče varati, bilo to zbog nedostatka vremena, nezainteresiranosti ili neznanja. Korištenjem pouzdanih alata za otkrivanje sličnosti među studentskim radovima, omogućava se transparentnije i pravednije ocjenjivanje, potiče se samostalan rad studenata te dugoročno doprinosi stvaranju kvalitetnijeg obrazovnog okruženja. Zahvaljujući sustavu Edgar koji studentska rješenja organizira na strukturiran način, ali i alatu JPlag koji implementira algoritam detekcije plagijata, izgradnja sustava automatske detekcije plagijata ostvarena je na funkcionalan i pouzdan način. Ostvareni sustav je robustan i visoko skalabilan. Odabrane tehnologije i stil arhitekture pokazao se prikladan s obzirom na iskustvo rada u njima. Iznimka je Mongo baza koja trenutno pohranjuje i podatke o konfiguracijama za detekciju plagijata. Ovi podatci odgovaraju organiziranim podatcima stabilne strukture te bi ispravnije bilo takve podatke pohranjivati u relacijsku bazu podataka. Funkcionalnost privremenog zaustavljanja detekcije plagijata uzrokovala je modificiranje JPlag izvornog koda. Iako minimalne, ove promjene predstavljaju problem kada se sustav treba nadograditi na noviju verziju JPlaga. Promjene JPlag biblioteke uključuju promjenu modifikatora vidljivosti metoda i razreda kako bi se postigla veća granularnost nad poslom detekcije plagijata kao i mogućnost praćenja internog napretka posla detekcije. Sustav trenutno detekciju plagijata provodi tako da se pojedini koraci kao što su prevođenje, normalizacija i usporedba obavljaju odvojeno, a ne pod kontrolom JPlaga što bi značilo da ako JPlag u proces detekcije uvede još jedan korak, sustav bi trebalo modificirati tako da se i metoda koja obavlja taj korak označi kao javna te pozove na odgovarajućem mjestu u kodu. Moguće je također ove promjene modifikatora vidljivosti predložiti autorima JPlaga kako bi postali dio standardne biblioteke.

Literatura

[1] H. Cheers, Y. Lin, S. P. Smith, *Academic Source Code Plagiarism Detection by Measuring Program Behavioral Similarity,* IEEE Access 9. izdanje, (2021), str. 50391-50412.

[2] Breanna Devore-McDonald, Emery D. Berger, *Mossad: Defeating Software Plagiarism Detection,* Proc. ACM Program. Lang. 4, OOPSLA, Članak 138, (2020).

[3] M. Novak, Review of source-code plagiarism detection in academia, 39. međunarodna konferencija o informacijskoj i komunikacijskoj tehnologiji, Opatija, Hrvatska, (2016), str. 796-801.

[4] Mehsen, Raddam, *Review of Source Code Plagiarism Detection Techniques,* (2022).

[5] Halstead, M. Aurice oward, *Elements of Software Science*, Elsevier (1977)

[6] Ottenstein, Karl J., *An Algorithmic Approach to the Detection and Prevention of Plagiarism*, SIGCSE Bulletin, (1977), str. 30-41.

[7] G. Cosma, M. Joy, *An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis*, IEEE Transactions on Computers, 61. izdanje, broj 3, (2012), str. 379-394.

[8] L. Prechelt, G. Malpohl, and M. Philippsen (2002). *Finding plagiarisms among a set of programs with JPlag*, Journal of Universal Computer Science, 8. izdanje, broj 11, (2002), str. 1016-1038.

[9] M. Mozgovoy, *Enhancing Computer-Aided Plagiarism Detection*. Dizertacija, Sveučilište u Joensuu, Finska, (2007).

[10] M. Wise, *YAP3: improved detection of similarities in computer program and other texts*, SIGCSE Bulletin, 28. izdanje, broj 1, (1996), str. 130–134.

[11] M. Mozgovoy, Fredriksson, K., White, D., Joy, M., Sutinen, E., *Fast Plagiarism Detection System*, (2005).

[12] B. Baker, *Parameterized pattern matching by Boyer-Moore type algorithms*, Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, (1995), str. 541-550.

[13] Jurriaan Hage, Brian Vermeer, Gerben Verburg, *Plagiarism Detection for Haskell with Holmes*. Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research (CSERC '13). Open Universiteit, Heerlen, Heerlen, NLD, (2013), str. 19–30.

[14] Chao Liu, Chen Chen, Jiawei Han, Philip S. Yu., *GPLAG: detection of software plagiarism by program dependence graph analysis*. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '06). Association for Computing Machinery, New York, NY, USA, (2006), str. 872–881.

[15] G. Cosma, M. Joy, *An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis*, IEEE Transactions on Computers, izdanje 61, broj 3, (2012), str. 379-394.

[16] Alex Aiken, *MOSS, a System for Detecting Software Similarity*, Poveznica: https://theory.stanford.edu/~aiken/moss/; pristupljeno 14. lipnja 2025.

[17] Timur Sağlam, Moritz Brödel, Larissa Schmid, Sebastian Hahner, *Detecting Automatic Software Plagiarism via Token Sequence Normalization,* Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), članak 113, (2024), str. 1–13.

[18] Timur Sağlam, Nils Niehue, Larissa Schmid, Sebastian Hahner, *Mitigating Obfuscation Attacks on Software Plagiarism Detectors via Subsequence Merging,* Karlsruher Institut für Technologie (KIT), Engineering (ICSE '24), (2025)

[19] Baza podataka MongoDB, poveznica: https://www.mongodb.com; pristupljeno 17. lipnja 2025.

[20] Baza podataka Redis, poveznica: https://redis.io; pristupljeno 17. lipnja 2025.

[21] Spring okruženje za razvoj poslužiteljskih aplikacija, Poveznica: https://spring.io; pristupljeno 17. lipnja 2025.

[22] Programski jezik Kotlin, poveznica: https://kotlinlang.org; pristupljeno 17. lipnja 2025.

[23] JavaScript razvojno okruženje za web aplikacije, poveznica: https://vuejs.org; pristupljeno: 17. lipnja 2025.

[24] Python FastAPI, poveznica: https://fastapi.tiangolo.com; pristupljeno 17. lipnja 2025.

[25] OpenAI Python biblioteka, poveznica: https://platform.openai.com/docs/libraries?language=python; pristupljeno 17. lipnja 2025.

[26] Grupiranje algoritmom K-sredina, poveznica: https://stanford.edu/~cpiech/cs221/img/kmeansViz.png; pristupljeno 24. lipnja 2025.

[27] Hijerarhijsko aglomerativno grupiranje i prikaz dendrograma, poveznica: https://support.minitab.com/en-us/minitab/help-and-how-to/statistical-modeling/multivariate/how-to/cluster-observations/interpret-the-results/all-statistics-and-graphs/dendrogram/; pristupljeno 24. lipnja 2025.

[28] Radni okvir za razvoj web aplikacija Express.js, poveznica: https://expressjs.com; pristupljeno 25. lipnja 2025.

[29] EJS mehanizam za kombiniranje HTML i JavaScript koda, poveznica: https://ejs.co; pristupljeno 25. lipnja 2025.

Sažetak

Općenamjenski servis za detekciju plagijata

Diplomski rad nudi pregled dostupnih tehnika i alata za detekciju plagijata u studentskim radovima. Naglasak se stavlja na detekciju plagijata u programskom kodu, ali razmotrene su i opcije detekcije plagijata radova napisanih prirodnim jezikom. U drugom dijelu rada demonstriran je implementirani sustav za automatsku detekciju plagijata. Sustav je organiziran u stilu mikroservisne arhitekture za čiju su implementaciju odabrane tehnologije Spring, Kotlin, Python, Vue.js, Mongo i Redis. Sustav je integriran s Edgar sustavom za pisanje provjera znanja i predaju rezultata koji je razvijen na Fakultetu elektrotehnike i računarstva. Detekcija plagijata ostvarena je korištenjem alata otvorenog koda, JPlag, koji implementira detekciju plagijata na temelju tokena koji čine programski kod. Implementirani sustav visoko je skalabilan te korisniku nudi i web aplikaciju za pregled i konfiguriranje poslova detekcije plagijata.

Ključne riječi: detekcija plagijata, JPlag, MOSS, Edgar, izračun sličnosti programskog koda, usporedba programskog koda na temelju tokena, detekcija korištenja umjetne inteligencije za pisanje programskog koda

Summary

General-purpose plagiarism detection service

The thesis provides an overview of available techniques and tools for plagiarism detection in student assignments. The focus is placed on detecting plagiarism in source code, although options for detecting plagiarism in natural language texts are also considered. In the second part of the thesis, an implemented system for automatic plagiarism detection is demonstrated. The system follows a microservice architecture, implemented using technologies such as Spring, Kotlin, Python, Vue.js, MongoDB, and Redis. It is integrated with the Edgar system for administering exams and submitting results, developed at the Faculty of Electrical Engineering and Computing. Plagiarism detection is achieved using the open-source tool JPlag, which detects plagiarism based on the token sequences that make up the source code. The implemented system is highly scalable and offers a web application for configuring and reviewing plagiarism detection jobs.

Keywords: plagiarism detection, JPlag, MOSS, Edgar, source code similarity calculation, source code comparison based on tokens, detection of artificial intelligence usage in source code generation