Aplikacija za interaktivno programiranje u Pythonu

Rajić, Magdalena

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

APLIKACIJA ZA INTERAKTIVNO PROGRAMIRANJE U PYTHONU

Završni rad br. 01/RAČ/2022

Magdalena Rajić

Bjelovar, rujan 2022.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Magdalena Rajić** JMBAG: **0023135083**

Naslov rada (tema): **Aplikacija za interaktivno programiranje u Pythonu**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Tomislav Adamović, mag. ing. el.** zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Ivan Sekovanić, mag.ing.inf.et comm.techn., predsjednik**
2. **Tomislav Adamović, mag. ing. el., mentor**
3. **Krunoslav Husak, dipl. ing. rač., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 01/RAČ/2022

U sklopu završnog rada potrebno je:

1. Integrirati okruženje Judge0 u aplikaciju za izvršavanje Python koda
2. Izraditi Oracle bazu podataka za vođenje evidencije interaktivnog programiranja u Pythonu
3. Izraditi algoritam u PHP-u za procjenu kompleksnosti koda po Halsteadovoj i McCabeoj metrici
4. Integrirati algoritam za procjenu kompleksnosti koda u aplikaciju za interaktivno programiranje u Pythonu
5. Izraditi prezentacijski sloj aplikacije za interaktivno programiranje u Pythonu korištenjem HTML-a, CSS-a i JavaScripta

Datum: 24.06.2022. godine

Mentor: **Tomislav Adamović, mag. ing. el.**

REPUBLIKA HRVATSKA BJELOVAR VELEUČILIŠTE U BJELOVARU

Sadržaj

1. UVOD

Aplikacija za interaktivno programiranje u programskom jeziku Python je web aplikacija izrađena pomoću Oracle PL/SQL-a, PHP-a, JavaScript-a, CSS-a, HTML-a i Judge0 okruženja. Tehnologije koje su korištene u izradi ove aplikacije detaljnije su predstavljene u drugom poglavlju.

Cilj aplikacije je omogućiti korisniku progresivno učenje programskog jezika Python kroz rješavanje zadataka različitih kategorija i kompleksnosti. Zadaci se korisniku dodjeljuju ovisno o izračunu složenosti koda pomoću dvije vrste metodologije: Halstead kompleksnosti i ciklomatske ili McCabe kompleksnosti. U poglavlju 3, Kompleksnost koda, detaljnije se objašnjava način izračuna ovih metodologija.

U 4. poglavlju predstavlja se aplikacija i njen tijek. Korisnik ulaskom u aplikaciju ima opciju registracije, verifikacije putem elektroničke pošte te prijave na web stranicu. Nakon što korisnik uspješno izvrši prethodne korake, pristupa svom profilu gdje mu je omogućena izmjena imena, elektroničke adrese, lozinke i profilne fotografije. Budući da je naglasak na interaktivnoj aplikaciji, korisniku se u web pregledniku zadaje zadatak koji bi trebao riješiti te se zadatak onda testira pomoću Judge0 okruženja. Kada korisnik smatra da je zadatak točno riješen, predaje ga te se provjerava ispravnost zadatka. Osim navedenog, u 4. poglavlju je predstavljen i programski kod pomoću kojeg se izračunava Halstead i ciklomatska složenost te definira kojoj od kategorija zadatak pripada. Prikazani su i postupci kojima se provjerava ispravnost predanog zadatka. Ukoliko je predani kod korisnika ispravan, funkcija na temelju prethodno riješenog zadatka određuje koji se idući zadatak treba dodijeliti korisniku. Spomenuta funkcija prema kategoriji prethodnog zadatka bira iduću kategoriju te koliko bi se kompleksan kod trebao zadati korisniku.

Korisnik tako napreduje, odnosno svaki idući zadani zadatak mu je kompleksniji, tj. teži od prethodnog. Ukoliko korisnik ne zna riješiti zadatak na ispravan način, nudi mu se opcija prelaska na zadatak manje kompleksnosti.

2. OPIS RAZVOJNOG SUSTAVA ZA APLIKACIJU

2.1. Judge0

Judge0 je robustan i skalabilan online sustav otvorenog koda (eng. *open-source*) za izvršavanje koda. H. Z. Došilović započeo je razvoj Judge0 2016. godine u cilju stvaranja alata koji se može koristiti pri izgradnji različitih aplikacija koje bi zahtijevale online izvršavanje koda. Ideja je bila i stvaranje *Online Code Execution System-a* koji bi se mogao jednostavno integrirati u razne *web* aplikacije, što se vidi iz primjera gdje je izrađen Edgar – platforma za e-učenje (eng. *e-learning*) koju od akademske godine 2018./2019. koristi više od 3000 studenata Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu [1].

Judge0 je baziran na *Online Judge* ekosistemu (eng. *Online Judge Ecosystem*) koji uključuje komponente vidljive na tablici 2.1.

Tablica 2.2:1: Online Judge ecosystem arhitektura [1]

<table>
<tr><td colspan="2">Competitive Programming Platform, e-Learning Platform,
Recruitment Platform, Interview Preparation Platform…</td></tr>
<tr><td>Online Judge</td><td>Online Compiler, Online IDE</td></tr>
<tr><td colspan="2">Online Code Execution System</td></tr>
<tr><td colspan="2">Code Execution Engine</td></tr>
<tr><td colspan="2">Sandbox</td></tr>
</table>

Sandbox je sigurnosni mehanizam za razdvajanje pokrenutih programa i često se koristi za izvršavanje nepouzdanog koda. U kontekstu Online Judge-a, *sandbox* se koristi kao mjera sigurnosti i ograničava iskorištavanje resursa (memorija i/ili CPU). *Code execution engine* ili skraćeno CEE je sloj koji koristi *sandbox* za prevođenje/kompiliranje i pokretanje koda i za analizu podataka koji su generirani nakon kompilacije. *Online code execution system* (OCES) je sustav koji pruža web API koji koristi CEE za prevođenje i izvođenje koda. Online *compilers* ili Online code *editors* platforme su na kojima korisnici pišu, prevode i izvršavaju svoj kod u nekom od programskih jezika. *Online Judge* je online servis koji obavlja predaju, ocjenjivanje ili bodovanje

programskog koda. Zadnja kategorija predstavlja platforme i aplikacije koje mogu koristiti OJ na neki od načina predstavljenih u nastavku.

Osim spomenute platforme za e-učenje, postoje i drugi načini kako integrirati ovaj alat u druga rješenja. Moguće je koristiti kompajler poput Judge0 IDE ili ugraditi Judge0 u vlastiti IDE te samostalno učiti, vježbati i razvijati vještine programiranja. Također, porastao je broj tvrtki koje testiraju svoje kandidate pri zaposlenju i pomoću ovakvih tehnologija stvaraju alate kojima procjenjuju vještine kandidata. Sukladno tome, razvile su se i platforme koje pripremaju i uvježbavaju svoje korisnike kako bi ih uspješno pripremili za takvu vrstu intervjua. Osim toga, naglasak je i na općenitom unapređenju vještina kod programiranja, a neke od platformi njih koje koriste Judge0 su AlgoDaily, PrepForTech itd. [2]

2.2. Backend

2.2.1. PHP

PHP (eng. *Hypertext Preprocessor*) je serverski skriptni jezik široke upotrebe koji se koristi za web razvoj i gdje HTML može biti ugrađen (eng. *embedded*) u PHP.

Razvoj PHP-a je većinski usmjeren na skriptiranje na poslužiteljskoj strani. Dakle, kod PHP-a programski kod se izvršava na poslužitelju, tj. na serveru, a kasnije se generira HTML koji se šalje klijentu. Klijent prima rezultate pokrenute skripte, ali mu nije dostupan ishodišni programski kod.

Rad na PHP-u omogućava prikupljanje podataka preko obrazaca (eng. *form*), generiranje dinamičkog sadržaja stranice itd. Iako je skriptiranje na strani poslužitelja najzastupljeniji način uporabe PHP-a, ovaj skriptni jezik podržava i skriptiranje naredbenog retka, kao i stvaranje desktop aplikacija. Operativni sustavi kao što su Linux, neke verzije Unix-a, Microsoft Windows, macOS podržavaju rad PHP-a. Osim do sad spomenutih mogućnosti, PHP sadrži i opciju korištenja proceduralnog programiranja ili objektno orijentiranog programiranja, ali i njihove kombinacije.

Kako bi se PHP mogao pisati i izvršiti, potrebna je instalacija Apache web servera i uređivača koda (eng. *code editor*) po želji, npr. Visual Studio Code. PHP datoteke se spremaju s ekstenzijom .php unutar web direktorija, npr. ../xampp/htdocs mape. Uz PHP se koriste i baze podataka budući

da PHP može izvršavati različite operacije nad podacima u bazi, kao što su stvaranje, čitanje, izmjena, brisanje i slično.

Kada PHP parsira datoteku, traži otvaranje i zatvaranje tag-ova: <?php naredbe koje izvršavamo ?>. Oznake za otvaranje i za zatvaranje su granice koje pomoću PHP Zend Engine-ukazuju PHP-u kada treba započeti i zaustaviti interpretaciju koda. Zbog toga je PHP-u omogućeno ugrađivanje u različite dokumente jer parser zanemaruje sve izvan otvorenog i zatvorenog znaka (eng. *tag*). Način na koji se HTML kod ugrađuje u PHP datoteku, vidljiv je na programskom kodu 2.1.

Programski kod 2.1: HTML kod ugrađen u PHP datoteku

```
<html>

<head>
    <title>PHP skripta</title>
</head>

<body>
    <?php echo '<p>Pozdrav, svijete!</p>'; ?>
</body>

</html>
```

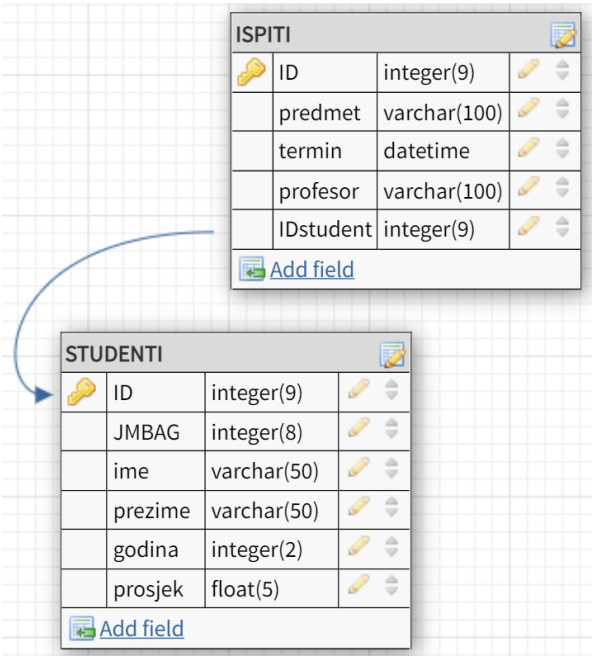
2.2.2. Oracle PL/SQL

Oracle SQL Developer je integrirano razvojno okruženje (eng. *Integrated development environment*), skraćeno IDE, koje se koristi za rad sa SQL-om u Oracle bazi podataka. SQL Developer je besplatan alat koji je napisan u programskom jeziku Java te pri radu koristi Java Development Kit. Oracle Coroporation razvio je distribuciju Java tehnologije, tj. *Java Development Kit* (JDK) koja se sastoji od virtualnog stroja (eng. *virtual machine*), kompajlera, alata za analizu performansi i programa za pronalaženje grešaka (eng. *debugger*) [3].

SQL je alat za organizaciju, upravljanje i dohvaćanje podataka pohranjenih u bazi podataka [4]. Naziv SQL skraćenica je engleske riječi *Structured Query Language*, a uloga ovog programskog jezika je komunikacija s bazom podataka. Postoji više različitih tipova baza podataka poput centralizirane, distribuirane i objektno orijentirane [4]. SQL je vezan uz relacijske baze podataka koje se temelje na relacijskom modelu i omogućavaju pristup međusobno povezanim podatkovnim točkama (eng. *data points*). Ovaj model pruža jednostavan način predstavljanja

podataka – tablicama gdje se stupci tablice nazivaju atributi: npr. ime, prezime, mjesto rođenja dok se svaki novi zapis u tablici naziva red i sadrži jedinstveni ID koji se naziva primarni ključ.

Uzet je jednostavan primjer od dvije tablice koje ostvaruju povezanost preko ključeva u modelu: studenti i prijava ispita, prikazano slikom 2.1.



Slika 2.1: Model jednostavne relacijske baze podataka

Jedna tablica predstavlja informacije o studentu – ime, prezime, godina, prosjek i JMBAG. Druga tablica služi za prijavljivanje predmeta na rokovima – naziv predmeta, termin, ime profesora i ID studenta koji prijavljuje predmet, ali ne i njegovo ime ili bilo koji podatak iz prve tablice. Zbog toga što ove dvije tablice imaju povezani ID, tj. ID studenta postaje dio tablice Ispiti, relacijska baza podataka uz strani ključ stvara odnos između njih. Kada je ostvarena relacija pomoću primarnog i stranog ključa, omogućeno je dohvaćanje podataka o studentu i izvršavati druge naredbe nad spojenim tablicama.

2.3. Frontend

2.3.1. HTML

HyperText Markup Language ili skraćeno HTML, temeljni je prezentacijski jezik World Wide Web-a (WWW). Izvorno, HTML je dizajniran kao jezik za semantičko opisivanje znanstvenih dokumenata, no njegov je opći dizajn omogućio da postane prilagodljiv i za prikaz niza drugih vrsta dokumenata i aplikacija.

HTML dokumenti se sastoje od dva bitna dijela: informacijski sadržaj i skup uputa koje nalažu kako prikazati taj sadržaj. Skup uputa – *markup* sačinjavaju HTML „jezik“. HTML nije programski jezik u klasičnom smislu nego skup uputa kako prikazati sadržaj u web pregledniku s ciljem da online sadržaj izgleda jednako bez obzira koji se operacijski sustav ili preglednik koristi. Svaki HTML dokument bi trebao sadržavati najmanje četiri elementa: <html>…</html>, <head>…</head>, <title>…</title> i <body>…</body> koji definiraju (istim redom): sam dokument, odjeljak zaglavlja, odjeljak naslova i tijelo. Programskim kodom 2.2, tj. jednostavnim HTML dokumentom prikazano je kako se svaki HTML element definira s jednom[1] ili dvije oznake (eng. *tag*) koje se nalaze u uglatim zagradama. U slučaju dvije oznake, element se upisuje u početnu oznaku: <element> i završnu oznaku: </element>.

Programski kod 2.2: HTML dokument

```
<!DOCTYPE html>

<html lang="en">

<head>
    <title>HTML primjer</title>
</head>

<body>
    <h1>Primjer</h1>
    <p>Ovo je <a href="primjer.html">jednostavan</a> primjer.</p>
    <!-- ovo je komentar -->
</body>

</html>
```

Budući da je HTML dokument samo tekstualni dokument, on se može stvarati pomoću bilo kojeg uređivača teksta s ekstenzijama .html ili .htm. Iako se HTML dokument može pisati u bilo kojem uređivaču teksta, to nije uvijek preporuka jer nakon korištenja programa za obradu teksta sa svim značajkama (eng. *full-featured word processor*) dokument više ne sadrži samo upisani tekst nego i druge informacije poput oblikovanja, rasporeda i sl. [5] Moguće je formatirati bilo koji dokument kao HTML dokument, ali takvi pretvoreni dokumenti mogu sadržavati prevelike količine dodatnih informacija te promijeniti datoteku u puno veći i složeniji oblik nego što bi trebao biti.

[1] HTML elementi bez završne oznake: br (prijelom retka), img (slika), col (svojstvo stupca) itd.

HTML nudi veliki broj opcija u pisanju, oblikovanju i prikazu zbog toga što su dokumenti, elementi, atributi i vrijednosti poredani u specifičnoj hijerarhiji: HTML dokument → elementi (eng. *elements*) → atributi (eng. *attributes*) → vrijednosti (eng. *values*) [5]. Elementi postoje kao dio dokumenta i mogu sadržavati atribute, a atributi najčešće imaju neku vrijednost. Kao što je već spomenuto, svi elementi su ugniježđeni unutar <html>…</html> elementa pa je, nadalje, element <title>…</title> smješten unutar nekog drugog elementa itd.

Stroga pravila u pogledu sintakse je nešto što obilježava druge programske jezike, ali HTML je drugačiji jer prema „zadanom standardu" pisanja korisnici mogu pristupiti onako kako njima odgovara. HTML ima definiranu sintaksu, ali ako se preglednik suoči s dijelom HTML koda kojeg ne razumije, obično ga ignorira. U suprotnosti, tako nešto bi srušilo (eng. *crash*) izvršavanje klasičnih programskih jezika.

Zbog postojanja atribut opcije, HTML nudi opciju uređenja jer se na taj način u dokument uključuju slike, opisi slika, visine i širine oblika, stil teksta, boja tablice i mnoge druga značajke. Web stranice su izvorno bile namijenjene za objavu istraživačkih radova i zbog toga dizajni i dekoracije nisu bile u prvom planu. Međutim, zbog ekspanzije uporabe HTML-a i sličnih alata, porasla je potreba za estetikom te se razvio jezik (eng. *style sheet language*) koji će biti predstavljen u sljedećem poglavlju.

2.3.2. CSS

Cascading Sytle Sheets (CSS) je stilski jezik (eng. *style sheet language*) W3C[2] standarda za definiranje načina prikaza *markup* jezika. Dakle, CSS datoteka sadrži niz pravila koji određuju kako će elementi HTML dokumenta biti stilski uređeni, a kasnije i prikazani pri otvaranju preglednika. Neki od svojstava na čiji se izgled može utjecati su veličina, boja, položaj i sl.

CSS je zaseban jezik s vlastitom sintaksom i postoji puno razloga zašto koristiti *style sheet languages*. CSS-om lako možemo postići preciznost i kontrolu izgleda. Promjenom nad samo jednim *style sheet*-om može se utjecati na izgled cijele stranice, npr. mijenjati boju i font teksta, boju pozadine i sl. Kada se CSS koristi u maksimalnom potencijalu, on postaje robustan i moćan alat za dizajniranje.

CSS pravila određuju kako se sadržaj elementa ili elementi s kojima je povezan prikazuju. Pravila se sastoje od dva dijela: selektor (eng. *selector*) i blok deklaracije (eng. *declaration block*)

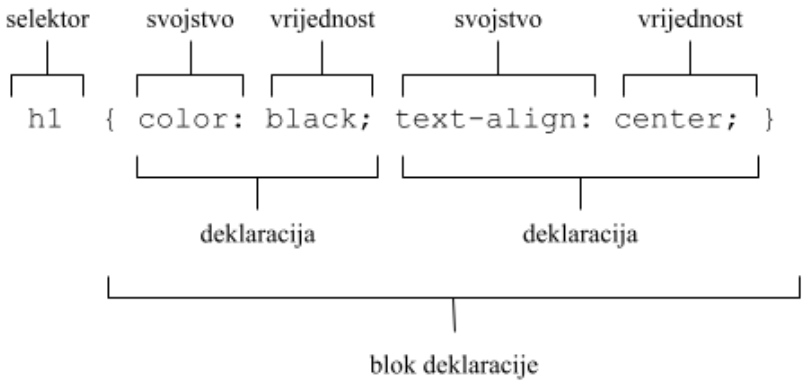
[2] World Wide Web Consortium (W3C) – organizacija koja se bavi standardizacijom tehnologija korištenih na webu

[6]. Uloga selektora je odabrati element (ili elemente) na koje se moraju primijeniti upute koje su navedene u bloku deklaracije. Blok deklaracije može sadržavati jednu i/ili više deklaracija zapisanih unutar vitičastih zagrada. Svaka deklaracija sadrži svojstvo i vrijednost. Svojstvo određuje koji element mijenja, a vrijednost označava odabranu postavku svojstva. Primjer koji prikazuje CSS sintaksu, selektor i deklaraciju vidljiv je u programskom kodu 2.3, a prikaz svakog elementa moguće je pronaći na slici 2.2.

Programski kod 2.3: CSS selektor i blok deklaracije

```
h1 {
    color: black;
    text-align: center;
    background-color: lightblue;
}
```

Slika 2.2: CSS sintaksa



Osim određivanja stila web stranice preko posebne datoteke (npr. styles.css), moguće je estetsku promjenu nad HTML elementima izvršiti i unutar .html dokumenta. Ako se na HTML elemente utječe iz html dokumenta, CSS pravila potrebno je upisati unutar elementa <style>…</style> na način kao što je prikazano programskim kodom 2.4.

Programski kod 2.4: CSS sintaksa unutar HTML dokumenta 1

```
…
<head>
    <title>HTML - CSS pimjer</title>
    <style>
```

```
        h1 {
            color: black;
            text-align: center;
            background-color: lightblue;
        }
    </style>
</head>

<body>
    <h1>Primjer s uređenim zaglavljem</h1>
</body>
…
```

U slučaju da se CSS pravila zadaju direktno nekom od HTML elemenata unutar oznaka tog elementa, to je moguće napraviti na način prikazan na Programskom kodu 2.5.

Programski kod 2.5: CSS sintaksa unutar HTML dokumenta 2

```
…
<head>
    <title>HTML – CSS primjer</title>
</head>

<body>
    <h1 style="color:black; text-align:center; background-
color:lightblue;">Primjer s uređenim zaglavljem</h1>
    <p>Ovo je <a href="primjer.html">jednostavan</a> primjer.</p>
</body>
…
```

Postoji više opcija kako primijeniti neku od CSS naredbi nad elementima u HTML-u. Neke od njih su već spomenute programskim kodom 2.3, 2.4 i 2.5 gdje je selektor h1 jedan od elemenata u HTML dokumentu. U slučaju da u HTML datoteci postoji više elemenata, npr. … i da se na svaki od tih div elemenata (eng. *division*) želi primijeniti neki univerzalni stil, koristi se opcija klase. U programskom kodu 2.6 vidljivo je kako se stvara klasa (eng. *class*) unutar oznake elementa div. Unutar elementa <head> uključena je i .css datoteka u kojoj se definiraju ta pravila. Sadržaj style.css datoteke vidljiv je programskim kodom 2.7.

Programski kod 1.6: Povezivanje oznake s klasom U HTML-u i CSS-u

```
…
<head>
    <title>HTML – CSS primjer</title>

    <link href="style.css" rel="stylesheet" />
</head>

<body>
    <div class="languages">
        <h2>Hrvatska</h2>
        <p>Hrvatski jezik je službeni jezik Hrvatske.</p>
    </div>

    <div class="languages">
        <h2>Španjolska</h2>
        <p>Španjolski jezik je službeni jezik Španjolske.</p>
    </div>
</body>
…
```

Programski kod 2.7: Sadržaj CSS datoteke s klasom

```
.languages {
    color: rgb(71, 71, 71);
    background-color: rgb(139, 183, 189);
    text-align: center;
}
```

Pomoću CSS-a i HTML-a moguće je koristiti i atribut id koji predstavlja jedinstveni identifikator za HTML element. Na taj je način omogućeno manipuliranje elementom s određenim ID-om, tj. moguće je utjecati na stil samo jednog elementa (koji može biti dio neke veće klase ili HTML elementa). Na sljedeća dva programska koda – programski kod 2.8 i programski kod 2.9, prikazano je definiranje id-a i način na koji se taj id koristi u CSS-u.

Programski kod 2.8: Definiranje id-a unutar HTML dokumenta

```
…
<body>
    <div class="languages" id="RH">
        <h2>Hrvatska</h2>
        <p>Hrvatski jezik je službeni jezik Hrvatske.</p>
    </div>

    <div class="languages" id="ES">
        <h2>Španjolska</h2>
        <p>Španjolski jezik je službeni jezik Španjolske.</p>
    </div>
</body>
…
```

Programski kod 2.8: CSS datoteka s definiranim id-em

```
.languages {
    color: rgb(71, 71, 71);
    background-color: rgb(139, 183, 189);
    text-align: center;
}

#HR {
    width: 100%;
}

#ES {
    width: 50%
}
```

Opcije i mogućnosti CSS-a su brojne te postoji puno metoda kojima možemo poboljšati izgled stranice u pregledniku. Uz opsežno znanje HTML-a, CSS-a, a i JavaScript-a moguće je osmisliti responzivne, upečatljive i prilagodljive web stranice.

2.3.3. Javascript

JavaScript je programski jezik web-a, jezik klijentske strane (eng. *client-side*) koji radi unutar web preglednika. Veliki broj web stranica i modernih web preglednika koriste JavaScript – na stolnim računalima, tabletima i telefonima interpretiraju (eng. *interprets*) JavaScript čineći ga najrasprostranjenijim programskim jezikom u povijesti [7]. JavaScript, skraćeno JS, dio je trijade već spomenutih tehnologija: HTML za određivanje sadržaja web stranica, CSS za određivanje prezentacije web stranica i JavaScript za određivanje ponašanja web stranica. JavaScript je omogućio nadogradnju jednostavnih HTML dokumenata korištenjem animacija, dinamičnih vizualnih efekata, interaktivnošću i sl. Zbog uporabe JS-a, web stranice postaju korisnije jer daju trenutne povratne informacije (eng. *feedback*), npr. kao kod unosa informacija u obrazac (eng. *form*) – ako nedostaje neki od obaveznih podataka, web stranica automatski vraća poruku o pogrešci.

JavaScript je dinamičan programski jezik visoke razine koji je interpretiran – što znači da se njegove upute izvršavaju izravno, bez prethodnog prevođenja koda (programa) u instrukcije poznate strojnom jeziku. Varijable u JavaScriptu nisu tipizirane (eng. *untyped*), dakle varijable se mogu stvarati bez definiranja tipa podatka i JavaScript varijable mogu sadržavati vrijednost bilo kojeg tipa. Iako naziv JS podsjeća na naziv programskog jezika Java, oni nisu povezani osim površno gledane sintaktičke sličnosti. [8]

Budući da web preglednik obično očekuje HTML, JavaScript uključujemo koristeći <script>…</script> oznake unutar <head>…</head> elementa[3]. Postoje bar dva načina kako koristiti JavaScript kod uz HTML, a jedan od njih je pisanje skripte unutar navedenih <script>…</script> oznaka. Unutar otvorene *script* oznake uključujemo i atribut *type* koji ukazuje na korišteni format i tip skripte – primjer vidljiv u programskom kodu 2.9.

[3] ispravno je navesti <script> oznake bilo gdje unutar HTML[7]

Programski kod 2.9: JS kod unutar <script> oznaka u HTML dokumentu

```
…
<head>
    <title>HTML primjer</title>
    <script type="text/javascript">
        alert('Pozdrav!');
    </script>
</head>
…
```

Kao drugi način uključenja JS-a u HTML, koriste se vanjske (eng. *external*) JavaScript datoteke. Slična je praksa kao i kod uključenja CSS datoteka gdje vanjska datoteka završava ekstenzijom .js. Primjer povezivanja JavaScript datoteke s web stranicom se nalazi u programskom kodu 2.10.

Programski kod 2.10: Uključenje JS datoteke unutar HTML dokumenta

```
…
<head>
    <title>HTML primjer</title>
    <script type="text/javascript" src="script.js"></script>
</head>
…
```

JavaScript je programski jezik koji nudi puno mogućnosti: od rješavanja matematičkih, logičkih i problemskih zadataka do rada s nizovima znakova (eng. *string*), nizovima, setovima, mapama i objektima. JavaScript djeluje kao spojnica s *backendom* jer se u JS skripti kroz funkcije i npr. Ajax mogu slati zahtjevi prema backendu. Ajax (eng. *Asynchronous JavaScript and XML*) je skup tehnika pomoću kojih web aplikacije mogu slati i dohvatiti podatke s poslužitelja. Nakon što se poziv izvrši, dohvaćeni se podaci pomoću metoda mogu prikazati u obliku JSON-a (npr. JSON.parse()). JSON omogućava jednostavan pristup određenom elementu vraćenog objekta jer je zapisan u obliku para: ključ – vrijednost. Primjer JSON formata je vidljiv u programskom kodu 2.11. Nakon što su potrebni podaci dohvaćeni i izmijenjeni u željeni oblik, prikazani su u web pregledniku.

Programski kod 2.11: JSON format[9]

```
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                { "value": "New", "onclick": "CreateDoc()" },
                { "value": "Open", "onclick": "OpenDoc()" },
                { "value": "Save", "onclick": "SaveDoc()" }
            ]
        }
    }
}
```

S obzirom na to što informacije navedene u ovom poglavlju ne predstavljaju cijeli JavaScript i njegove mogućnosti, više o ovom programskom jeziku predstavlja David Flanagan u knjizi JavaScript – The Definitve Guide [8].

3. KOMPLEKSNOST KODA

3.1. Uvod

Kompleksnost koda je značajno prisutna tema u području računarstva još od 1970-ih kada su M.H. Halstead i T.J. McCabe započeli s razvojem softverske metrike (eng. *software metrics*). Već se 1990. u radu *Predicting Source – Code Complexity at the Design Stage*, autora S. Henry i C. Selig spominje kako je računarstvo u tada posljednjem desetljeću doživio veliku revoluciju. Evolucijom razvoja softvera, nastala je potreba za mjerenjem jer je složenost koda postala suština unutarnje kvalitete koda. [10]

Postoje studije u kojima se istražuje postoji li poveznica između složenosti softvera i troškova njegovog održavanja. Prema studiji [11], na temelju analize održavanja projekta u komercijalnom okruženju, potvrđeno je da su troškovi održavanja softvera pod značajnim utjecajem kompleksnosti koda. U navedenom istraživanju, utvrđeno je da se troškovi održavanja softvera povećavaju s povećanjem složenosti koda**.** Zbog spomenutog rapidnog razvoja, većina modela softverske ekonomije[4] (eng. *Software Economics*) bila je usredotočena na razvoj novih tehnologija te su stoga zanemarili metriku kompleksnosti koda. Zaključeno je da zanemarivanje složenosti softvera potencijalno ozbiljan propust jer troškovi održavanja softvera visoke razine složenosti mogu iznositi i 25% svih troškova održavanja ili čak više od 17% ukupnih troškova životnog ciklusa (eng. *life – cycle*).

Složenost koda nije samo izračun koji pokazuje koliko je neki kod kompleksan za napisati nego predstavlja i metriku kojom se može izraziti koliki je napor (eng. *effort*) potreban za neki od problema. Osim spomenutih, moguće je izračunati i neke druge mjere poput broja pogrešaka, vremena potrebnog za programiranje i broja čvorova u grafu kontrolnog toka (eng. *control-flow graph*). U nastavku će biti predstavljene dvije najučestalije metrike kompleksnosti koda: Halstead kompleksnost i ciklomatska ili McCabe kompleksnost.

3.2. Halstead kompleksnost

Maurice Howard Halstead, profesor na Sveučilištu Purdue, začetnik je koncepta softverske znanosti (eng. *software science*) koji je doveo do razvoja softverske metrike. Umro je 1979. godine, 2 godine nakon što je objavljena njegova knjiga „Elementi softverske znanosti“ – *Elements*

[4] istraživačko područje koje se pitanjima vrednovanja softvera i određivanja ili procjene troškova koji su obično uključeni u njegovu proizvodnju

of Software Science (Operating and programming systems series) (1977.) u kojoj predstavlja Halstead kompleksnost (eng. *Halstead complexity*) - način mjerenja složenosti koda. Njegovo mjerenje postalo je široko korišteno kao alternativa brojanju linija koda (eng. *Lines of Code*). LOC se koristio kao mjera veličine i programske složenosti iako su linije koda isključivo ovisile o programskom jeziku. Poznato je da linije koda za isti programski kod mogu varirati od jezika do jezika što je dovelo do Halsteada – preciznog izračunavanja veličine i složenosti programa.

Halstead kompleknost predstavlja sedam mjera potpuno baziranih na broju operatora i operanda u programskom kodu. Operatori u programskom kodu su znakovi koji predstavlja određenu matematičku ili logičku radnju ili proces, a operandom se smatra element kojim se može upravljati. Da bi se izračunala kompleksnost koda, potrebno je klasificirati izvorni kod prema sljedećim varijablama prikazanima na tablici 3.1.

Tablica 3.1: Varijable u Halstead kompleksmosti

$$\eta_1 = broj\ jedinstvenih\ operatora \tag{3.1}$$

$$\eta_2 = broj\ jedinstvenih\ operanda \tag{3.2}$$

$$\mathrm{N}_1 = ukupan\ broj\ operatora \tag{3.3}$$

$$\mathrm{N}_2 = ukupan\ broj\ operanda \tag{3.4}$$

Na temelju prebrojavanja vrijednosti iz tablice 3.1., moguće je izračunati nekoliko mjera:

A. Veličina rječnika (eng. *Program Vocabulary*) – η

Veličina rječnika predstavlja sumu jedinstvenih operatora (3.1) i sumu jedinstvenih operanda (3.2) i računa se na sljedeći način:

$$\eta = \eta_1 + \eta_2 \tag{3.5}$$

B. Veličina/Dužina programa (eng. *Program Lenght*) – N

Veličina programa predstavlja sumu ukupnog broja operatora (3.3) i ukupnog broja operanda (3.4) i računa se na sljedeći način:

$$\mathrm{N} = \mathrm{N}_1 + \mathrm{N}_2 \tag{3.6}$$

C. Volumen/Obujam programa (eng. *Program Volume*) – V

Volumen programa je proporcionalan ukupnoj veličini programa (ovisi o A. Veličini rječnika i B. Veličini programa). Predstavlja memorijski prostor za pohranjivanje programa i predstavljen je u bitovima [12]. Volumen programa izračunavamo kao:

$$V = \mathrm{N} * \log_2 \eta \tag{3.7}$$

D. Procijenjena/Očekivana duljina/veličina programa (eng. *Calculated estimated program length*) – $\widehat{N}$

Procijenjena duljina programa predstavlja dobro strukturiran program i računa se prema broju jedinstvenih operatora (3.1) i broju jedinstvenih operanda (3.2) na sljedeći način:

$$\widehat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \tag{3.8}$$

E. Težina (eng. *Difficulty*) – D

Težina predstavlja razinu teškoće programa koja je proporcionalna broju jedinstvenih operatora u programu, a prikazuje se:

$$D = \frac{\eta_1}{2} * \frac{\mathrm{N}_2}{\eta_2} \tag{3.9}$$

F. Potreban trud (eng. *Programming Effort*) – E

Potreban trud za pisanje programskog koda je količina logičke i mentalne vježbe koja je potrebna kako bi se implementirao programski kod iz pseudokoda. Potreban trud se može izračunati:

$$E = D * V \tag{3.10}$$

G. Vrijeme potrebno za programiranje (eng. *Time required to program*) – T

Vrijeme potrebno za programiranje je predstavljeno kao razumno vrijeme za izradu programa, a izračunava se na sljedeće načine:

$$T = \frac{E}{\beta} \tag{3.11}$$

$$T = \frac{E}{18} \tag{3.12}$$

gdje je β broj koji se kreće između 5 i 20. Međutim, u najranijim Halsteadovim eksperimentima, broj 18 je davao najbolje rezultate te se za formulu uzima u obzir relacija (3.12).

H. Broj isporučenih grešaka (eng. *Number of delivered bugs*) – B

Broj isporučenih grešaka predstavlja broj pogrešaka u implementaciji. Smatra se da je broj isporučenih grešaka (eng. *bugs*) povezan s ukupnom složenošću softvera. Može se izračunati ovako:

$$B = \frac{E^{\frac{2}{3}}}{3000} \tag{3.13}$$

$$B = \frac{V}{3000} \tag{3.14}$$

U nastavku, na temelju programskog koda 3.1 u programskom jeziku C će se izračunati kompleksnosti koda prema Halsteadu.

Programski kod 2.1: Funkcija u programskom jeziku C

```
void funkcija() {
    int x, y, z, suma;
    scanf("%d %d %d", &x, &y, &z);
    suma = x + y + z;
    printf("suma = %d", suma);
}
```

Prvi korak pri određivanju cjelokupne kompleksnosti je izračun broja jedinstvenih operatora i broja jedinstvenih operanda te izračun sume svih operatora i operanda.

- Jedinstveni operatori (η_1) su sljedeći: void, (), {}, int, ,, ;, scanf, &, =, +, printf
- Jedinstveni operandi (η_2) su sljedeći: funkcija, x, y, z, suma, "%d %d %d", "suma = %d"
- Svi operatori (N_1) su sljedeći: void, (), {}, int, ,, ,, ,, ;, scanf, (), ,, &, ,, &, ,, &, ;, =, +, +, ;, printf, (), ,, ;

- Svi operandi (N_2) su sljedeći: funkcija, x, y, z, suma, "%d %d %d", x, y, z, suma, x, y, z, "suma = %d", suma

S obzirom na to što su operatori i operandi u programskom kodu prebrojeni, moguće je izračunati sve dijelove Halstead kompleksnosti. Prema relacijama 3.1 – 3.4 vrijedi:

$$\eta_1 = 11 \tag{3.15}$$

$$\eta_2 = 7 \tag{3.16}$$

$$\mathrm{N}_1 = 25 \tag{3.17}$$

$$\mathrm{N}_2 = 15 \tag{3.18}$$

Na osnovi izračunatih izraza iz relacija 3.1 – 3.4, relacije 3.5 – 3.14 poprimaju sljedeće vrijednosti:

A. Veličina rječnika

$$\eta = 11 + 7 \tag{3.19}$$
$$\eta = 18$$

B. Veličina/Dužina programa

$$\mathrm{N} = 25 + 15 \tag{3.20}$$
$$\mathrm{N} = 40$$

C. Volumen/Obujam programa

$$V = 40 * \log_2 18 \tag{3.21}$$
$$V = 166.797$$

D. Procijenjena/Očekivana duljina/veličina programa

$$\widehat{N} = 11 \log_2 11 + 7 \log_2 7 \tag{3.22}$$
$$\widehat{N} = 57.705$$

E. Težina

$$D = \frac{11}{2} * \frac{15}{7} \tag{3.23}$$
$$D = 11.786$$

F. Potreban trud

$$E = 11.786 * 166.797 \tag{3.24}$$
$$E = 1965.87$$

G. Vrijeme potrebno za programiranje

$$T = \frac{1965.87}{18} \tag{3.25}$$
$$T = 109.215\ s$$

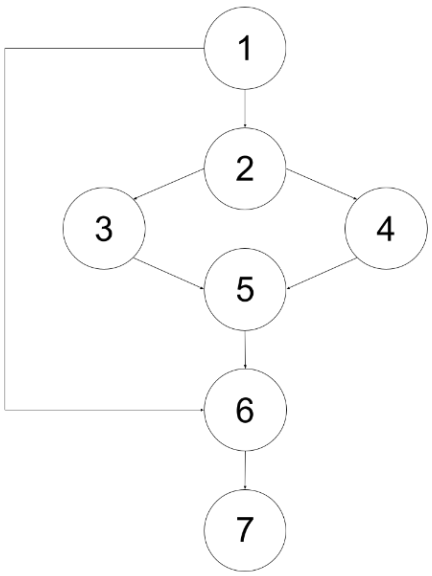
H. Broj isporučenih grešaka

$$B = \frac{166.797}{3000} \tag{3.26}$$
$$B = 0.056$$

Vidljivo je da Halstead kompleksnost može koristiti bilo koji programski jezik budući da je izračun jednostavan. Potrebno je prebrojiti operatore i operande, a iz toga proizlazi osam vrlo korisnih vrijednosti koje dalje omogućavaju detaljniju analizu programskog koda.

3.3. Ciklomatska složenost

Ciklomatska složenost je mjera koju je 1978. [13] prvi put predstavio Thomac McCabe. McCabeova ciklomatska složenost je metrika koja mjeri broj linearno nezavisnih putova kroz dio koda ili cijelog programa. Smatra se da se programe s nižom ciklomatskom složenošću može lakše razumjeti i manje ih je riskantno mijenjati nego one s visokim CCM (eng. *Cyclomatic Complexty Measures*) [14]. Ciklomatska složenost koristi graf upravljanja tokom (eng. *control-flow graph*) kako bi se kompleksnost koda mogla izračunati. Primjer kako izgleda graf upravljanja tokom, moguće je vidjeti na slici 3.1.



Slika 3.1: Primjer grafa upravljanja tokom

Pri izračunu, McCabeova kompleksnost predstavlja koliki broj „točaka odluke“ postoji u softveru. Sama CCM se može izračunati pomoću više metoda.

Definicija 1.

Ciklomatski broj v(G) grafa G s n čvorova, e rubova i p povezanih komponenti je:

$$v(g) = e - n + p \tag{3.27}$$

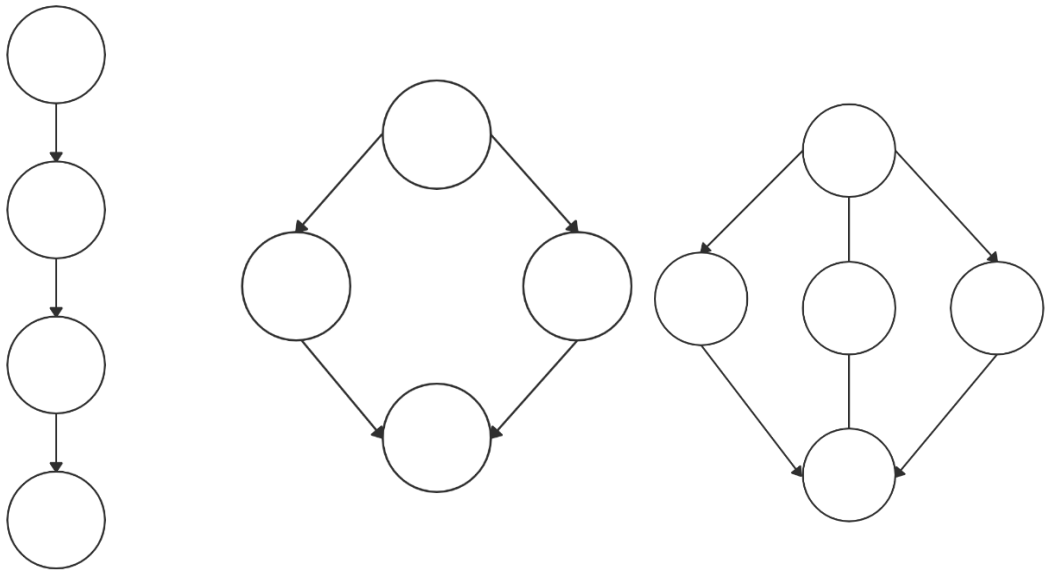
Teorem 1.

U jako povezanom grafu G, ciklomatski broj je jednak maksimalnom broju linearno neovisnih sklopova.

U Definiciji i Teoremu 1. naglasak je na grafu kojem je svaka izlazna točka povezana s ulaznom točkom, tj. pretpostavlja se da svaki čvor može dosegnuti ulazni čvor i svaki čvor može doći do izlaznog čvora.

Prema Definiciji 1., p označava broj povezanih komponenti (eng. *connected components*). Način na koji je definiran kontrolni tok grafa rezultira time da svi kontrolni grafovi imaju samo jednu povezanu komponentu. U slučaju da program MAIN ima dva potprograma X i Y, onda broj

povezanih komponenti nije 1, već je 3 (1 za glavni program i po 1 za svaki od potprograma), vidljivo na slici 3.2.



a) program MAIN *b) potprogram X* *c) potprogram Y*

Slika 3.2: Primjer programa MAIN koji se sastoji od dva potprograma

Ako se na temelju slike 3.2 graf prikaže kao $v(MAIN \cup X \cup Y)$, onda je ukupna kompleksnost koda 6 jer vrijedi [13]:

$$\begin{aligned} & v(MAIN \cup X \cup Y) \\ & = e - n + 2p \\ & = 13 - 13 + 2 * 3 \\ & = 6 \end{aligned} \tag{3.28}$$

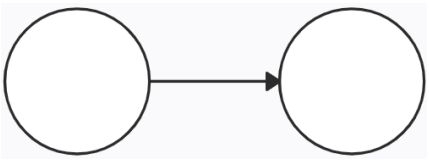
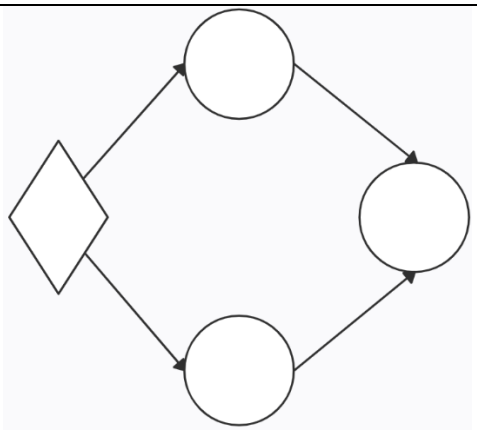
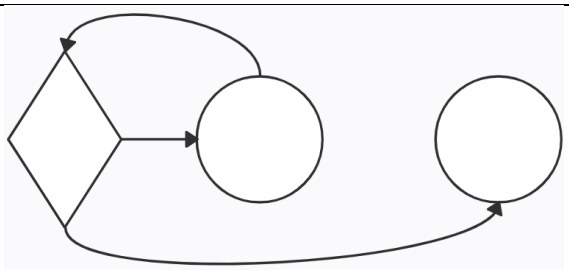
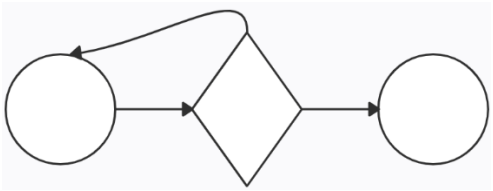
Ova metoda s $p \neq 1$ se može koristiti za izračunavanje složenosti zbirke programa, posebno kod hijerarhijskog ugniježđenja kojeg čine potprogrami slični kao oni prikazani na slici 3.2.

Stoga, ciklomatski se broj v(G) grafa G s n čvorova i e rubova može prikazati na idući način:

$$v(g) = e - n + 2p \tag{3.29}$$

Nekoliko primjera izračuna ciklomatske složenosti s obzirom na *control-flow graph* i relaciju (3.15) moguće je proučiti u tablici 3.2.

Tablica 3.2: Primjeri izračuna ciklomatske kompleksnosti koda

Sekvenca (eng. *sequence*)		$v(g) = e - n + 2p$ $v(g) = 1 - 2 + 2$ $v(g) = 1$
IF → THEN → ELSE		$v(g) = e - n + 2p$ $v(g) = 4 - 4 + 2$ $v(g) = 2$
WHILE		$v(g) = e - n + 2p$ $v(g) = 3 - 3 + 2$ $v(g) = 2$
UNTIL		$v(g) = e - n + 2p$ $v(g) = 3 - 3 + 2$ $v(g) = 2$

S obzirom na matematičku formulu iz relacije (3.29), za jedan program (ili potprogram), broj povezanih komponenti je uvijek jednak 1. Dakle, pojednostavljena formula u ovom slučaju bi glasila:

$$v(g) = e - n + 2 \tag{3.30}$$

Osim navedene formule te njenih varijacija postoji i pojednostavljen izračun složenosti za jedno-komponentne grafove. Osmišljena su dva rezultata: jedan omogućuje jednostavnije izračune

s pogleda programske sintakse dok drugi omogućava lakši izračun iz grafikona. Uz [13] i [15] dokazana je istinitost sljedeće formule:

$$v = \pi + 1 \tag{3.31}$$

Relacija 3.31 dokazuje da je ciklomatska složenost jednaka broju predikata odnosno *decision points*-a uvećan za jedan. Kasnije se pokazalo da je u izračunavanju ove vrste složenosti prikladnije brojati uvjete unutar programskog koda nego predikate te je dokazano [13] da je složenost bilo kojeg nestrukturiranog programa jednaka $\pi + 1$.

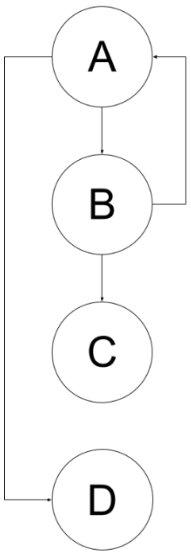
S obzirom na to što je matematička formula iz *Definicije 1.* pojednostavljena s ciljem da se programerima olakša korištenje iste [13], početnu formulu, tj. relaciju (3.27) možemo napisati na konačni način gdje je p čvor koji sadrži uvjet (eng. *condition*):

$$v = p + 1 \tag{3.32}$$

U računarstvu, uvjet je naredba programskog jezika koja služi za upravljanje odlukama. Postoji više vrsta uvjeta, a u programskom jeziku C najčešći su sljedeći: if, else, else if, switch, or, and, … Na tablici 3.3 moguće je vidjeti implementaciju ciklomatske složenosti na jednostavnom programskom kodu:

Tablica 3.3: Implementacija ciklomatske složenosti

```
void prosti_broj(int n) {
      for(int i = 2; i < n; i++) {
            if(n % i == 0) {
                  printf("Nije prosti
broj");
            }
      }
}
```



$v(g) = e - n + 2$

$v(g) = 4 - 4 + 2$

$v(g) = 2$

Određene su četiri razine kategorizacije ciklomatske složenosti koje je predstavio Tom McCabe. U svojoj prezentaciji „Mjerila kvalitete softvera za prepoznavanje rizika“ (eng. *Software Quality Metrics to Identify Risk*) za Ministarstvo domovinske sigurnosti SAD-a predstavio je sljedeće kategorije prikazane u tablici 3.3 i tablici 3.4 [16]:

Tablica 3.3: Kategorije rizika pouzdanosti

Ciklomatska složenost	Rizik pouzdanost
1 – 10	Jednostavan postupak, mali rizik
11 – 20	Složenije, umjereni rizik
21 – 50	Složeno, visok rizik
> 50	Kod koji se ne može testirati, vrlo visok rizik

Tablica 3.4: Kategorije vjerojatnosti lošeg popravka

Ciklomatska složenost	Vjerojatnost lošeg popravka
1 – 10	5%
20 – 30	20%
> 50	40%
Približno 100	60%

McCabe složenost, tj. ciklomatska složenost brzo se uči, jednostavno izračunava te je intuitivna za razumjeti. Iako je ovo lako shvatljiva metoda, previše ugniježđenih uvjeta može otežati razumijevanje koda. Općenito se smatra da metrike za računanje kompleksnosti koda samostalno daju ili nedovoljne ili nepotpune informacije, ali čak i gruba indikaciju koju ciklomatska složenost pruža, ima veliku vrijednost za voditelja projekta te programera[17].

4. PROGRAMSKA IZVEDBA APLIKACIJE

Aplikacija za interaktivno programiranje u Pythonu je web aplikacija izrađena pomoću tehnologija, okruženja, programskih jezika i metoda navedenih u prethodnim poglavljima. Ovo poglavlje bit će posvećeno implementaciji spomenutih tehnologija uz prikaz dijelova koda, sheme modela baze podataka te izgled same aplikacije.

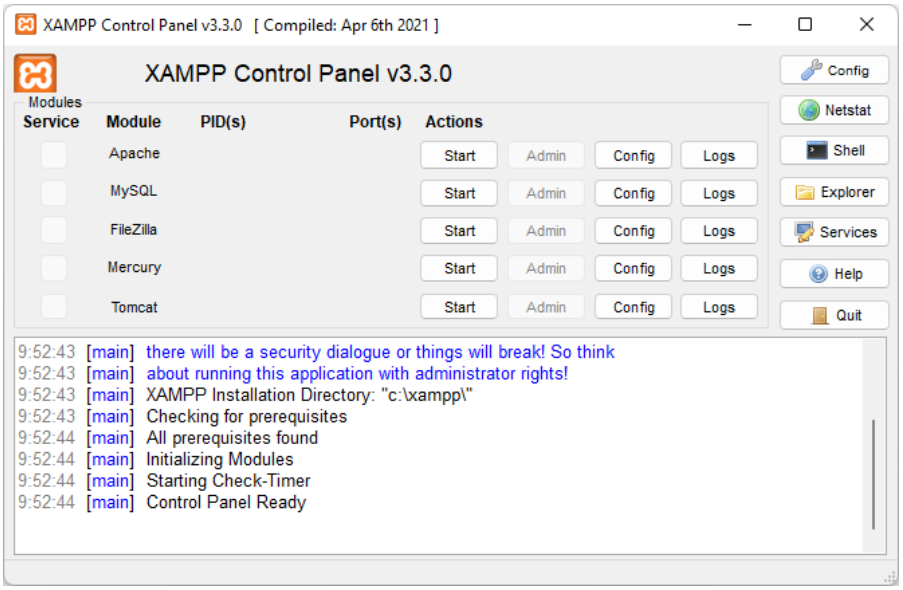
Tema Završnog Rada odnosno aplikacije bila je osmisliti platformu pomoću koje korisnici mogu učiti programski jezik Python. Korisniku je omogućena registracija i prijava te mogućnosti uređenja svog profila uz promjenu profilne fotografije, imena i/ili prezimena, e-mail adrese i lozinke. Nakon uspješno izvršene prijave, web preglednik prikazuje okruženje koje se sastoji od uređivača koda i konzole i od područja na kojem je prikazan zadatak koji bi korisnik trebao riješiti. Korisnik ima opciju pisati i uređivati programski kod, izvršavati ga (eng. *run*) i predati ako smatra da je kod ispravno napisan. Nakon što korisnik preda zadatak, program mu vraća poruku je li predani zadatak točno ili netočno riješen. U prvom slučaju, korisnik može klikom na gumb preći na kompleksniji zadatak, a u drugom korisniku je omogućeno korisniku ponovno pisanje i uređivanje, izvršavanje, a naposljetku i predaja koda. Korisnik, također, ima opciju i odustati od zadanog zadatka, tj. zatražiti rješenje ukoliko ne zna ili ne može riješiti trenutni zadatak. U tom slučaju korisniku je dana opcija prelaska na manje kompleksan zadatak.

Osim prostora za rješavanje zadataka, aplikacija sadrži i upravljačku ploču (eng. *admin panel*) u kojem ovlašteni korisnik – *admin* može unositi nove, uređivati stare i brisati postojeće zadatke. Admin prilikom unosa novog zadatka upisuje tri podatka u obrazac: naziv zadatka, opis zadatka, tj. objašnjenje kako riješiti zadatak i točan kod zadatka. Program na osnovi tih podataka izračunava kompleksnost zadatka i u koju kategoriju zadatak pripada.

4.1. XAMPP

Osim navedenih tehnologija, za rad ove aplikacije bilo je potrebno i konfigurirati *Cross-Platform, Apache, MySQL, PHP and Pearl* ili poznatiji kao XAMPP, široko korištena Apache distribucija koja sadrži najčešće tehnologije za web razvoj u jednom paketu. Paket sadrži Apache web poslužitelj te služi kao lokalni web poslužitelj i omogućava korisnicima lokalno testiranje programa.

Nakon instalacije XAMPP-a, moguće je otvoriti XAMPP kontrolnu ploču, vidljivo na slici 4.1:



Slika 4.1: XAMPP kontrolna ploča

Na XAMPP kontrolnoj ploči: Config → Apache (httpd.config) odrađene su sljedeće izmjene:

- Uklanjanje nastavaka .php i .html kod prikaza putanje web stranice izvršeno na način prikazan programskim kodom 4.1:

Programski kod 3.1: Uklanjanje ekstenzije iz URL-a

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^([^.]+)$ $1.php [NC,L]
RewriteRule ^([^.]+)$ $1.html [NC,L]
```

- Definirani podaci za konekciju na SMTP – protokol koji omogućava slanje e-mail poruke

Programski kod 4.2: Postavljanje SMTP-a

```
SetEnv SMTP_USERNAME your_email@gmail.com
SetEnv SMTP_PASSWORD your_password
```

- Definirani podaci za konekciju na bazu koji će se kasnije koristiti unutar PHP dokumenata

Programski kod 4.3: Postavljanje podataka za konekciju na bazu

a) httpd.config

```
SetEnv OCI_USERNAME db_username
SetEnv OCI_PASSWORD db_password
SetEnv OCI_URL db_url
```

b) .php

```
$conn = oci_connect(getenv("OCI_USERNAME"), getenv("OCI_PASSWORD"),
getenv("OCI_URL"), 'utf8');
```

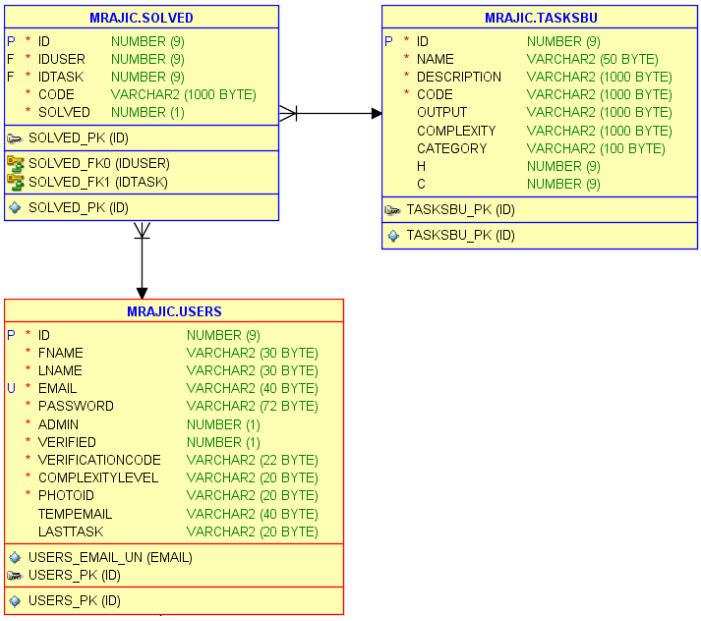
4.2. Baza podataka

Kako bi bilo moguće započeti s prikazom programskih kodova, potrebno je prvo predstaviti izrađeni model baze podataka i skriptu kako su baza i tablice u njoj stvorene.

Unutar baze podataka stvorene su tri tablice:

- Users,
- TasksBU i
- Solved

koje su povezane na način kao na slici 4.2:

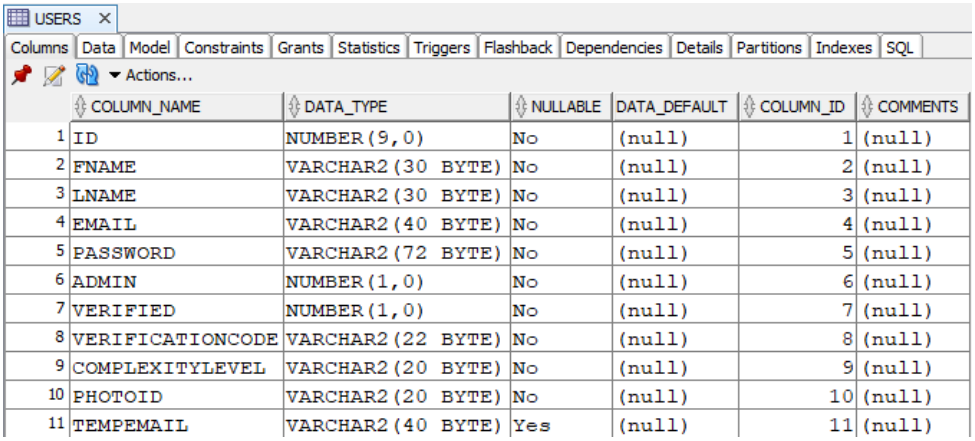


Slika 4.2: Model baze podataka

Kao što je na slici 4.2 moguće vidjeti, tablice Users, TasksBU i Solved su relacijski povezane preko stranih ključeva (eng. *foreign key*).

4.2.1. Tablica Users – Tablica korisnici

Tablica Users, tj. tablica korisnika je tablica koja sadrži podatke o svim korisnicima ove aplikacije. Tablica se sastoji od 11 atributa prikazanih na slici 4.3:



USERS

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(9,0)	No	(null)	1	(null)
2	FNAME	VARCHAR2(30 BYTE)	No	(null)	2	(null)
3	LNAME	VARCHAR2(30 BYTE)	No	(null)	3	(null)
4	EMAIL	VARCHAR2(40 BYTE)	No	(null)	4	(null)
5	PASSWORD	VARCHAR2(72 BYTE)	No	(null)	5	(null)
6	ADMIN	NUMBER(1,0)	No	(null)	6	(null)
7	VERIFIED	NUMBER(1,0)	No	(null)	7	(null)
8	VERIFICATIONCODE	VARCHAR2(22 BYTE)	No	(null)	8	(null)
9	COMPLEXITYLEVEL	VARCHAR2(20 BYTE)	No	(null)	9	(null)
10	PHOTOID	VARCHAR2(20 BYTE)	No	(null)	10	(null)
11	TEMPEMAIL	VARCHAR2(40 BYTE)	Yes	(null)	11	(null)

Slika 4.3: Atributi tablice Users

U tablici 4.1 moguće je proučiti svaki atribut i što on predstavlja:

Tablica 4:1.1: Opis atributa tablice Users

Atribut	Značenje
ID	Identifikacijski broj – primarni ključ
FNAME	Ime korisnika
LNAME	Prezime korisnika
EMAIL	E-mail adresa korisnika
PASSWORD	Lozinka korisnika
ADMIN	Ima li korisnik administratorske ovlasti ili ne
VERIFIED	Je li korisnik verificiran
VERIFICATIONCODE	Verifikacijski kod koji se korisniku šalje na e-mail adresu
COMPLEXITYLEVEL	Zadatak najveće kompleksnosti koji je korisnik uspio riješiti (do tada)
PHOTOID	Izabrana profilna fotografija

TEMPEMAIL	Služi za čuvanje nove email adrese kod promjene, sve do verifikacije iste adrese

Tablica Users je izrađena koristeći SQL naredbu CREATE na način prikazan na programskom kodu 4.1:

Programski kod 4.4: Stvaranje tablice Users

```
CREATE TABLE USERS (
      "ID" NUMBER(9,0),
      "FNAME" VARCHAR2(30 BYTE),
      "LNAME" VARCHAR2(30 BYTE),
      "EMAIL" VARCHAR2(40 BYTE),
      "PASSWORD" VARCHAR2(72 BYTE),
      "ADMIN" NUMBER(1,0),
      "VERIFIED" NUMBER(1,0),
      "VERIFICATIONCODE" VARCHAR2(22 BYTE),
      "COMPLEXITYLEVEL" VARCHAR2(20 BYTE),
      "PHOTOID" VARCHAR2(20 BYTE),
      "TEMPEMAIL" VARCHAR2(40 BYTE));
```

Osim kreiranja tablice, bilo je potrebno i stvoriti sekvencu za ID kako bi prilikom stvaranja svakog novog zapisa, ID automatski bio upisan kao broj za jedan viši od prethodnog. Programski kod koji je to omogućio vidljiv je u nastavku u programskom kodu 4.5:

Programski kod 4.5: Sekvenca za ID

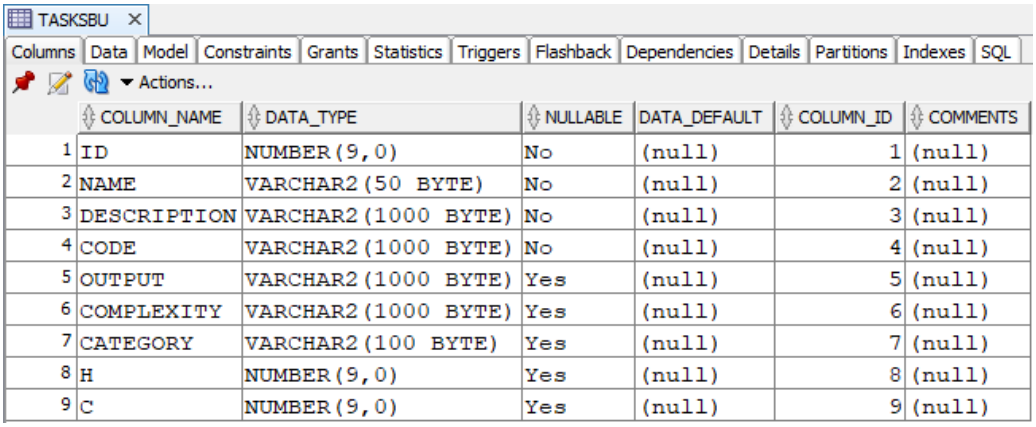
```
CREATE sequence USERS_ID_SEQ;

CREATE trigger BI_USERS_ID
    before insert on USERS
    for each row
begin
    select USERS_ID_SEQ.nextval into :NEW.ID from dual;
end;
```

Također, ID u tablici Users je primarni ključ (eng. *primary key*) i omogućava jedinstveno identificiranje pojedinog zapisa u tablici te kasnije stvaranje relacije s drugim tablicama preko stranog ključa.

4.2.2. Tablica TasksBU – Tablica zadaci

Tablica TasksBU predstavlja tablicu u kojoj se nalaze svi zadaci, njihovi opisi, rješenja, kompleksnosti prikazano slikom 4.4:



TASKSBU

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(9,0)	No	(null)	1	(null)
2	NAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	DESCRIPTION	VARCHAR2(1000 BYTE)	No	(null)	3	(null)
4	CODE	VARCHAR2(1000 BYTE)	No	(null)	4	(null)
5	OUTPUT	VARCHAR2(1000 BYTE)	Yes	(null)	5	(null)
6	COMPLEXITY	VARCHAR2(1000 BYTE)	Yes	(null)	6	(null)
7	CATEGORY	VARCHAR2(100 BYTE)	Yes	(null)	7	(null)
8	H	NUMBER(9,0)	Yes	(null)	8	(null)
9	C	NUMBER(9,0)	Yes	(null)	9	(null)

Slika 4.4: Atributi tablice TasksBU

U tablici 4.2 prikazani su atributi tablice TaksBU sa slike 4.4 uz kratko objašnjenje:

Tablica 4.2: Opis atributa tablice TasksBU

Atribut	Značenje
ID	Identifikacijski broj – primani ključ
NAME	Naziv zadatka
DESCRIPTION	Opis zadatka
CODE	Ispravan kod, tj. rješenje zadatka
OUTPUT	Očekivani ispis na osnovi koda iz atributa CODE
COMPLEXITY	Kompleksnost koda izračunata na osnovi koda iz atributa CODE
CATEGORY	Kategorija koda odabrana na osnovi koda iz atributa CODE
H	Halstead kompleksnost koda (preuzeta vrijednost iz COMPLEXITY atributa)
C	Ciklomatska kompleksnost koda (preuzeta vrijednost iz COMPLEXITY atributa)

Kao što je već prikazano programskim kodom 4.1 i 4.2, i ova tablica je izrađena na sličan način (samo s pripadajućim atributima i tipovima podataka). Isto tako je ID postao primarnim ključem tablice definiranjem *constraint*-a prikazanim na programskom kodu 4.3.

Programski kod 4.3: Definiranje primarnog ključa

```
constraint TASKSBU_PK PRIMARY KEY (ID)
```

4.2.3. Tablica Solved – Tablica riješenih zadataka

Tablica Solved je tablica koja je stranim ključem (eng. *foreign key*) povezana s dvjema gore spomenutim tablicama. Solved sadrži informaciju koji korisnik je riješio koji zadatak s kojim kodom te se pomoću izračuna određuje je li zadatak ispravno riješen. Tablicu je moguće vidjeti na slici 4.5, a objašnjenje atribute na tablici 4.3:

SOLVED

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(9,0)	No	(null)	1	(null)
2	IDUSER	NUMBER(9,0)	No	(null)	2	(null)
3	IDTASK	NUMBER(9,0)	No	(null)	3	(null)
4	CODE	VARCHAR2(1000 BYTE)	No	(null)	4	(null)
5	SOLVED	NUMBER(1,0)	No	(null)	5	(null)

Slika 4.5: Atributi tablice Solved

Tablica 4.3: Opis atributa tablice Solved

Atribut	Značenje
ID	Identifikacijski broj
IDUSER	Identifikacijski broj korisnika koji rješava zadatak
IDTASK	Identifikacijski broj zadatka koji se rješava
CODE	Kod koji je korisnik unio na predaju
SOLVED	Je li zadatak riješen ili ne

Također, način na koji je tablica kreirana i uvećanje ID-a sekvencom jednak je kao i prethodni primjeri, ali budući da se ova tablica sastoji od podataka iz drugih tablica tvoreći *parent*

– *child* relaciju preko stranog ključa, programskim kodom 4.4 je prikazana naredba za stvaranje relacije:

Programski kod 4.4: Spajanje primarnih i stranih ključeva

```
ALTER TABLE SOLVED ADD CONSTRAINT SOLVED_fk0 FOREIGN KEY (IDuser)
REFERENCES USERS(ID);
ALTER TABLE SOLVED ADD CONSTRAINT SOLVED_fk1 FOREIGN KEY (IDtask)
REFERENCES TASKSBU(ID);
```

Dakle, naredbom ALTER tablica Solved stvara *constraint* kojim definira dva strana ključa: SOLVED_fk0 i SOLVED_fk1. Strani ključ 1 (fk0) spaja atribut IDuser s atributom ID iz tablice Users. Na ovaj način, tablica Solved postaje dijete (eng. *child*) tablica dok tablica Users postaje roditelj (eng. *parent*) tablica, tj. tablica Solved je ovom relacijom postala ovisna o tablici Users. Isti princip se koristi i kod drugog stranog ključa (fk1) gdje su spojene tablice i atributi IDtask tablice Solved i atribut ID tablice TasksBU.

Unosi u bazu unutar ovog projekta izvode se upisom na frontend-u te zvanjem funkcija na backendu koje izvršavaju SQL naredbe. Također je moguće direktno preko aplikacije utjecati na podatke u tablicama i na same tablice pomoću ostalih SQL naredbi kao što su insert, alter, drop i delete.

4.3. Naslovna stranica

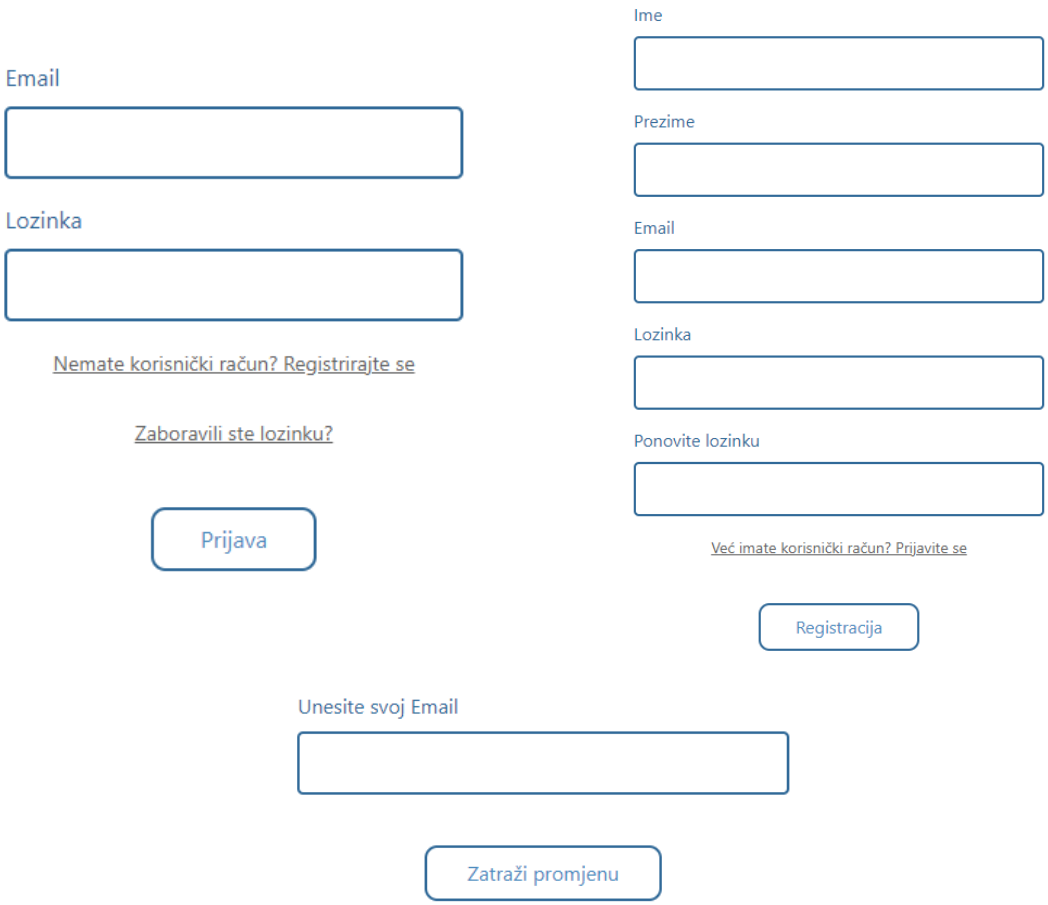
Početna stranicu InteractivePython aplikacije moguće je vidjeti na slici 4.6:

Slika 4.6: Naslovna stranica

Početna stranica sastoji se od obaveznih elemenata HTML-a, ali i dodatnih kao što su navigacijska traka, naslovna fotografija [18], nekoliko gumbova i pozdravne poruke. Gumb Login i KRENI usmjeravaju korisnika na login stranicu dok klik na InteractivePython ponovno usmjerava korisnika na index stranicu.

4.3.1. Registracija i prijava korisnika

Nakon što smo na početnoj stranici kliknuli na gumb KRENI ili Login, href atribut HTML-a otvara login stranicu na kojoj je vidljiv obrazac za prijavu. Ukoliko korisnik već ima izrađeni korisnički račun, izvršava prijavu, a ukoliko nema odabire opciju registracije. Obrazac prijave, registracije i promjene lozinke vidljivo je na slici 4.7:



Slika 4.7: Login, registracija i promjena e-mail adrese

Prvi korak je HTML forma pomoću koje sakupljamo potrebne podatke. U programskom kodu 4.5. prikazan je dio forme u koju korisnik upisuje svoje ime.

Programski kod 4.5: Registracijska forma

```
…
<form>
      <div>
            <label>Ime</label>
            <input type="text" id="fname">
            <p id="fnameMessage"></p>
     </div>
</form>
…
```

Ono što je iz programskog koda 4.5 vidljivo je da se id atribut nalazi unutar input oznake. Ono što je upisano u predviđenom prostoru je ono što će se preko id-a poslati i spremiti u bazu. Nakon što je korisnik upisao sve podatke koji su potrebni i ispravni, može kliknuti na gumb Registracija.

Potrebno je definirati ograničenja i kontrole u slučaju da korisnik želi predati formu s nepotpunim podacima, npr. ako korisnik ne navede svoje prezime. Isto tako, trebaju se spriječiti i neispravni podaci kao što je unos teksta koji zasigurno nije e-mail (npr. bez znaka @ i sl.) na mjestu gdje se e-mail očekuje. Ta se provjera odvija nakon što je gumb Registracija pritisnut. JavaScript preko id-a sprema upisane podatke u varijable na način prikazan na programskom kodu 4.6:

Programski kod 4.6: Spremanje podataka u varijable

```
let fname = $('#fname').val();
let lname = $('#lname').val();
let email = $('#email').val();
let password = $('#password').val();
let repeatPassword = $('#repeatPassword').val();
let pass1 = false;
let pass2 = false;
let check = true;
```

Nakon što je JS spremio unesene podatke, podaci se provjeravaju na sljedeći način, programski kod 4.7:

Programski kod 4.7: Provjera ispravnosti unesenih podataka

```
if (fname == '' || fname == null) {
      check = false;
}
…
if (password == '' || password == null) {
      check = false;
} else {
      pass1 = true;
}
if (repeatPassword == '' || repeatPassword == null) {
```

```
    check = false;
}
if (pass1 && pass2) {
     if (password != repeatPassword) {
        check = false;
    }
}
if (check == true) {
     register();
}
```

Ukratko, varijabla check (po defaultu *true*) će se promijeniti u *false* ako je bilo koje polje predano prazno, dakle ako je npr. fname == '' ili fname == null. Osim ove provjere, provjerava se i podudaranost zaporki. Ukoliko je ispunjeno sljedeće: lozinka je unesena, ponovno je upisana lozinka i ako su te lozinke jednake, check ostaje true (u protivnom postaje false i korisnik mora ponoviti obrazac) te zove se funkcija register.

Funkcija register je funkcija koja koristi *ajax* za slanje zahtjeva (eng. *request*) – spomenuto u poglavlju 2.3.3. JavaScript. U navedenoj funkciji se definiraju podaci kao što su tipovi zahtjeva[5] koji se šalju i url na koji zahtjev odlazi te podaci koji su potrebni da bi se zahtjev izvršio. Funkciju register i druge spomenute detalje moguće je vidjeti na programskom kodu 4.8:

Programski kod 4.8: Slanje POST zahtjeva za registraciju korisnika

```
function register() {
    $.ajax({
        type: 'POST',
        url: url,
        data: {
            procedura: 'p_register',
            fname: $('#fname').val(),
            lname: $('#lname').val(),
            email: $('#email').val(),
            password: $('#password').val()
        },
        success: (data) => {
            const jsonBody = JSON.parse(data);
            console.log(jsonBody);
        },
```

[5] Neki od zahtjeva: GET, POST, DELETE, …

```
        error: (xhr, textStatus, error) => {
            console.log(xhr.statusText);
            console.log(textStatus);
            console.log(error);
        },
        async: true,
    });
}
```

Tu je navedeno koja se procedura zove i koji se podaci prosljeđuju. *Router* je PHP datoteka koja poziva funkcije prema pozvanoj proceduri. Ako se procedura pozove na način kao u programskom kodu 4.8, *router* usmjerava poziv dalje, tj. izvršava funkciju koja je dio jedne od PHP datoteka. Vidljivo na programskom kodu 4.9, u slučaju odabira procedure p_register, poziva se funkcija f_register koja prosljeđuje podatke i konekciju na bazu.

Programski kod 4.9: router.php

```
switch ($in_obj->procedura) {
    case 'p_login':
        f_login($in_obj,$conn);
        break;
    case 'p_register':
        $sql = f_register($in_obj, $conn);
        break;
…
```

Nakon što nas je *router* preusmjerio na funkciju f_register, omogućeno je izvršavanje funkcije. Funkcija započinje provjerom jesu li podaci dostupni, proslijeđene podatke dohvaća i provjerava na idući način prikazano programskim kodom 4.10:

Programski kod 4.10: Provjera primljenih podataka

```
if($in_obj->fname == "" || $in_obj->lname == "" || $in_obj->email == ""
|| $in_obj->password == "") {
      echo $MESSAGES['insert_error'];
      exit;
}
```

Podacima je moguće pristupiti preko $in_obj – json objekta koji sadrži podatke proslijeđene s *frontenda* preko *routera* do *funkcije*. Pošto je izvršena provjera, PHP odrađuje još jednu. Budući da aplikacija može imati samo jednog korisnika s jednom e-mail adresom, PHP izvršava SQL naredbu pomoću koje broji koliko korisnika s unesenom e-mail adresom postoji. Ako je odgovor veći od 0, program vraća grešku da korisnik s navedenom e-mail adresom već postoji u bazi. Kako PHP izvršava SQL naredbe moguće je vidjeti na programskom kodu 4.11, a u tablici ispod, tablica 4.4, objašnjeno je što koja PHP metoda predstavlja.

Programski kod 4.11: Funkcija f_register – provjera e-mail adrese

```
$sql = "SELECT count(1) as COUNT FROM Users WHERE email = '{$in_obj-
>email}'";
$stid = oci_parse($conn, $sql);
oci_execute($stid);
oci_fetch($stid);

if(oci_result($stid, 'COUNT') > 0) {
      echo $MESSAGES['email_error'];
      exit;
}
```

Tablica 4.4: Elementi funkcije

`$sql`	Upisuje se SQL naredba
`oci_parse()`	Priprema $sql pomoću definirane veze ($conn) Vraća *false* pri grešci ili *identifier statement* pri uspjehu
`oci_execute()`	Izvršava naredbu prethodno vraćenu iz oci_parse
`oci_fecth()`	Dohvaća sljedeći redak iz upita u interni međuspremnik (eng. *internal buffers*) kojima se može pristupiti pomoću oci_result
`oci_result()`	Vraća vrijednost polja iz dohvaćenog retka

Kada PHP pomoću SQL-a izvrši provjeru e-mail-a, korisnička lozinka se sprema u obliku *hash*-a u novu varijablu koja će se kasnije spremiti u bazu. Atribut Verified postavlja se na vrijednost 0 što znači da korisnikov račun nije verificiran.

Programski kod 4.12: Funkcija f_register – Unos u tablicu Users

```
$hashed_password = password_hash($in_obj->password,
PASSWORD_DEFAULT);

$verification_code = substr(md5(rand()), 0, 20);

$sql = "INSERT INTO
           Users(fname, lname, email, password, admin, verified,
verificationcode, complexitylevel, photoid, lasttask)
        values
           ('{$in_obj->fname}','{$in_obj->lname}','{$in_obj-
>email}','{$hashed_password}', 0, 0, '{$verification_code}', 105, 1,
105)";
```

Za slanje e-maila na korisnikov račun, koristi se biblioteka (eng. *library*) PHPMailer. Nakon postavljanja postavki za slanje e-maila, stvara se e-mail u kojem se šalje ranije generirani verifikacijski kod na e-mail adresu korisnika.

Programski kod 4.13: Funkcija f_register – PHPMailer

```
$mail = new PHPMailer;
      $mail->isSMTP();
      $mail->Host = "your_host";
      $mail->SMTPAuth = true;
      $mail->Username = getenv("SMTP_USERNAME");
      $mail->Password =  getenv("SMTP_PASSWORD");
      $mail->SMTPSecure = "tls";
      $mail->Port = 587;
      ...
      $mail->From = 'no-reply@interactivepython.hr';
      $mail->FromName = 'InteractivePython - Verifikacija';
      $mail->addAddress($in_obj->email);
      $mail->isHTML(true);
      $mail->Subject = "Verificirajte svoj korisnički račun";
      $mail->Body = '
        Dobrodošli u aplikaciju Interactive Python!<br><br>
        Račun možete verificirati pritiskom na link:<br>
```

```
        <a href="../php/accounts/verifikacijaprofila.php?email=' .
$in_obj->email . '&code=' . $verification_code . '">CLICK</a>';

      $mail->send();
```

Nakon što korisnik otvori poveznicu poslanu na njegovu adresu, poziva se funkcija verifikacija profila koja provjerava podudaranost e-maila i verifikacijskog koda. Ukoliko se navedeno podudara, izvršava se SQL naredba kojom se atribut Verified mijenja u 1 čime je korisnik uspješno registriran i može se prijaviti u aplikaciju. Ovime je postupak registracije završen.

Prijavom korisnika u aplikaciju u $_SESSION se upisuju svi atributi iz tablice Users vezani uz specifičnog korisnika na način prikazan u kodu 4.14. *Session* je niz povezanih interakcija između klijenta i poslužitelja koje se odvijaju tijekom određenog vremenskog razdoblja [19].

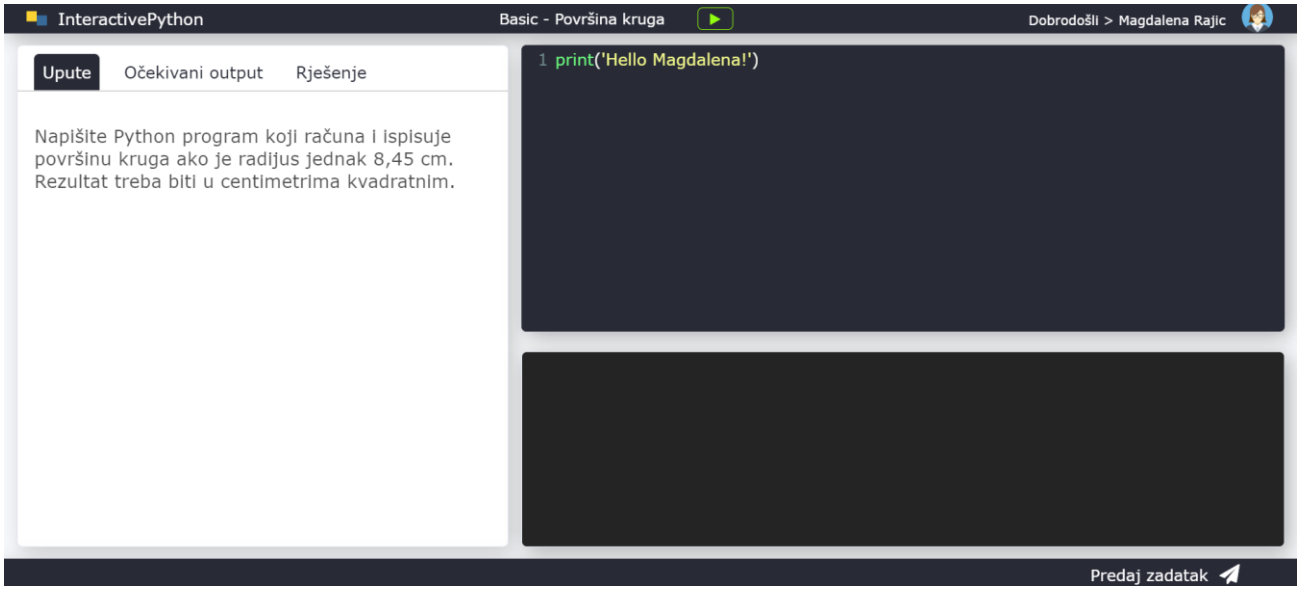
Programski kod 4.14: Funkcija f_login - session

```
$sql = "SELECT * FROM Users WHERE email = :email";
      $stid = oci_parse($conn,$sql);
      oci_bind_by_name($stid, ":email", $in_obj->email);
      oci_execute($stid);
      if ($row = oci_fetch_assoc($stid)) {
        if (password_verify($in_obj->password, $row['PASSWORD'])) {
          if($row['VERIFIED'] == 0) {
            echo $MESSAGES['verified_error'];
     exit;
          }
          foreach ($row as $key => $value) {
            $_SESSION[$key] = $value;
          }
      }
…
```

4.4. Integrirano razvojno okruženje

Uspješnom prijavom korisnika u pregledniku otvara se stranica task koja služi kao IDE (eng. *Integrated Development Environment*) korisniku u rješavanju zadataka u Python programskom jeziku. Kao što se može vidjeti na slici 4.8, s lijeve strane se nalazi područje koje služi kao smjernica korisnicima za rješavanje zadataka. Desna strana zaslona podijeljena je na

uređivač koda i „konzolu“, tj. prostor na kojem se ispisuje rješenje izvršenog zadatka. Također, zaslon se sastoji i od gornje i donje trake koje pružaju opciju testiranja zadatka, predaje zadatka i odlaska na stranicu kojom mijenjamo postavke profila.



Slika 4.8: Razvojno okruženje

Trenutni zadatak koji je zadan korisniku dohvaća se pri učitavanju prozora pozivom funkcije getTask. Funkcija poziva proceduru koja izvršava programski kod 4.15.

Programski kod 4.15: Dohvaćanje zadatka iz baze podataka

```
$sql = "SELECT * FROM Tasksbu WHERE ID =
'{$_SESSION["COMPLEXITYLEVEL"]}'";
```

Vraćeni podaci se parsiraju te se unutar web aplikacije prikazuju na način prikazan programskim kodom 4.16:

Programski kod 4.16: Pristup dohvaćenim podacima

```
$("#getTaskName").html(jsonBody['data'][0].CATEGORY + ' - ' +
jsonBody['data'][0].NAME);
$("#getTaskDescription").html(jsonBody['data'][0].DESCRIPTION.replace
(/\n/g, "<br />"));
```

```
$("#getTaskOutput").html(jsonBody['data'][0].OUTPUT.replace(/\n/g,
"<br />"));

$("#getTaskCode").html(jsonBody['data'][0].CODE.replace(/\n/g, "<br
/>"));
```

Funkcije getTaskName, getTaskDescription, getTaskOutput i getTaskCode su funkcije u JS dokumentu koje omogućuju otvaranje i zatvaranje prozora Upute, Očekivani output i Rješenje. Rad funkcija se zasniva na sakrivanju i prikazivanju dohvaćenog teksta.

Kao što je u poglavlju 4.3.1. navedeno, u $_SESSION je spremljen zadatak koji je korisnik zadnji uspješno riješio, tj. taj zadatak predstavlja napredak korisnika. Zadatak najmanje kompleksnosti u bazi je trenutno zadatak s ID-em 105, stoga prvi zadatak koji se korisniku zada je upravo taj 105. – najjednostavniji zadatak. Ako ga korisnik uspješno riješi, u bazi podataka, u tablici Users, vrijednost COMPLEXITYLEVEL više neće biti 105 nego ID zadatka idućeg po redu itd.

Korisniku je omogućeno rješavanje zadanog zadatka unutar CodeMirror uređivača koda u pregledniku. Klikom na ikonu *play* korisnik može testirati svoj kod pomoću Judge0 na sljedeći način prikazano programskim kodom 4.17:

Programski kod 4.17: Izvršavanje programskog koda

```
function runTask() {
    var succeed = false;
    var output = "";
    var prog = editor.getValue();
    var mypre = document.getElementById("output");
    mypre.innerHTML = '';
    $.ajax({
        type: "POST",
        url: judge0,
        data: {
            source_code: prog,
            language_id: 71
        },
…
```

Navedena funkcija dohvaća tekst, tj. kod koji je korisnik napisao unutar uređivača koda te šalje POST *request* prema Judge0. Podaci koji se u zahtjevu šalju su programski kod i oznaka

programskog jezika. Judge0 vraća odgovor kojeg kasnije ispisuje unutar prostora koji predstavlja „konzolu“. Konzolu u ovom kontekstu predstavlja prostor u kojem se prikazuje odgovor Judge0-a, tj. služi kao standardni izlaz (eng. *standard output*). Treba napomenuti da konzola u ovom kontekstu nema opciju standardnog ulaza.

4.4.1. Predaja zadatka

U slučaju da korisnik želi predati svoj kod, klikom na Predaj zadatak započinje poziv funkcije f_send_solution koja se može proučiti na programskom kodu 4.18:

Programski kod 4.18: Funkcija za predaju zadatka

```
function f_send_solution($in_obj, $conn) {
      $task_id = $_SESSION["COMPLEXITYLEVEL"];
    $user_code = $in_obj->code;
    $result = f_run_task($user_code);
    if($result == False) {
           echo $MESSAGES['request_error'];
        exit;
    }
    $user_output = $result;
    $complexity = f_compare_tasks($user_code, $user_output, $conn,
$task_id);
    if($complexity == False) {
           echo $MESSAGES['request_error'];
        exit;
    }
    f_save_solved($in_obj, $conn);
      f_higher_complexity_task($conn);
}
```

U ovoj se funkciji ID zadatka definira kao COMPLEXITYLEVEL spremljen u $_SESSION, a kod korisnika je onaj kod koji je korisnik pisao i uređivao te naposljetku i predao klikom na Predaj zadatak. Prvi korak je pokrenuti upisani zadatak na poslužiteljskoj strani te provjeriti je li Judge0 prihvatio napisani zadatak kao sintaktički točan – ako je, funkcija vraća izlaz (eng. *output*) navedenog koda. Naravno, ako programski kod nije napisan po pravilima programskog jezika Python, funkcija se završava.

Nakon sintaktičke provjere, poziva se funkcija f_compare_task koja prima predani kod korisnika, izlaz predanog koda, konekciju i ID zadatka koji se rješava.

4.4.2. Izračun kompleksnosti koda i određivanje kategorije

Unutar funkcije f_compare_task nalazi se funkcija calculate koja izračunava kompleksnost koda prema metrikama objašnjenima u 3. poglavlju, određuje u koju kategoriju zadataka zadatak pripada i od koliko se linija koda predani programski kod sastoji. Prvo će biti objašnjena funkcija calculate funkcija jer se na njenoj osnovi odrađuje usporedba zadatka i odluka je li zadatak točno riješen.

U funkciji calculate naglasak je na izračunu Halstead i ciklomatske složenosti. Kako bi se mogla izračunati Halsetad kompleksnost, skraćeno HCM, potrebno je prebrojiti broj jedinstvenih operatora i operanda kao i ukupni broj operatora i operanda. Poznati operatori u programskom jeziku Python su sljedeći: print, =, +, -, +, … te su definirane varijable vidljive na programskom kodu 4.19:

Programski kod 4.19: Definiranje varijabli za računanje HCM

```
$n1 = 0;     //number of distinct operators
$N1 = 0;     //total number of operators

$n2 = 0;     //number of distinct operands
$N2 = 0;     //total number of operands

$operators = array(
      "=" => 0,
      "+" => 0,
      "-" => 0,
      "*" => 0,
      "/" => 0,
      "%" => 0,
      "@" => 0,
      "&" => 0,
      "|" => 0,
...
);
$operands = array();
```

Svi elementi niza operators iznose nula te će se njihov broj povećavati za jedan svaki put kada se pronađe jedan od operatora u kodu. Budući da programski kod (najčešće) dolazi u više od jednog reda, potrebno je razložiti dobiveni kod na način da se omogući čitanje liniju po liniju što

je izvršeno pomoću *Regular Expressiona* i metode preg_split. U programskom kodu 4.20 moguće je i vidjeti uporabu dvije PHP metode koje uklanjanju (eng. *strip*) razmak s početka i kraja niza znakova (eng. *string*)

Programski kod 4.20: Razlaganje programskog koda na linije

```
foreach(preg_split("/((\r?\n)|(\r\n?))/", $code) as $line) {

      $line = ltrim($line);
      $line = rtrim($line);

      ...
}
```

Budući da zakomentirane linije ne utječu na kompleksnost koda, potrebno je detektirati komentare u programskom kodu. U Python-u komentari se mogu napisati na jedan od načina prikazanih programskim kodom 4.21:

Programski kod 4.21: Komentari u programskom jeziku Python

```
'''print('Pozdrav, svijete!')'''
#print('Hello, world!')
```

Stoga, kada linija koda započinje sa znakovima ''' i završava sa znakovima ''' te kada linija koda započinje sa znakom #, programski kod u toj liniji se ignorira i ne uračunava kasnije. U programskom kodu 4.22 vidljiva je upotreba metode str_starts_with koja provjerava započinje li *string* s danim *substringom*, tj. počinje li linija koda s ''' ili #. Ako programski kod ne započinje ni s jednim komentarom, ta linija koda postaje LoC (eng. *line of code*).

Programski kod 4.22: Provjera koda u slučaju komentara

```
if(str_starts_with($line, "'''")) {
      $CLoC += 1;
} else if(str_starts_with($line, "#")) {
      $CLoC += 1;
}
```

Prema Halstead pravilima, ono što se nalazi u zagradi unutar navodnika smatra se jednim operandom. Dakle, linija koda: `print('Pozdrav svijete')` se sastoji od samo jednog operanda, a to je 'Pozdrav svijete'. Zbog ovog pravila kreiran je *regular expression* kojim će

program, ukoliko naiđe na niz unutar linije koda koji počinje s „('“ ili „(“) i završava s „')“ ili „“)“, zamijeniti s nasumičnom vrijednošću (prikazano na programskom kodu 4.23).

Programski kod 4.23: Stvaranje operanda unutar print

```
$print_sub = "/(?:'|\")(.*?)(?:'|\")/";

$line = preg_replace($print_sub, "operand" . rand(1, 10000) , $line);
```

Ideja je bila spremiti svaki element koda u jedan niz te foreach petljom za svaki element provjeriti kojoj varijabli pripada – operatori ili operandi. Da bi bilo moguće ispravno razdvojiti svaku „riječ“ u kodu, tj. da dio koda print(a) postane niz: [0] => print [1] => ([2] => a [3] =>), korištene su metode u programskom kodu 4.24:

Programski kod 4.24: Dodavanje elemenata u words niz

```
$delimiters = array('(', ')', '[', ']', '{', '}', ',', '.', ':');
$words = array();

foreach($delimiters as $delimiter) {
      $line = str_replace($delimiter, ' ' . $delimiter . ' ', $line);
      $words = explode(" ", $line);
      $words = array_filter(array_map('trim', $words));
}
```

Delimiteri su definirani kao znakovi koji se u pisanju, što zbog pravila – što zbog urednosti, najčešće koriste uz druge dijelove koda, npr. print i (, pisanje neke sekvence: 1, 2, 3,… Stoga za svaku liniju koda, kod pojavljivanja delimitera se nadodavao razmak i s jedne i s druge strane kako bi svaki element zasigurno mogao biti procesuiran kao jedan. Također, metodom explode stringovi razdvojeni razmakom su postali dio niza words. Metodama array_filter i array_map se iz niza words uklonilo bilo koje pojavljivanje razmaka (eng. *whitespace*).

Nakon što je osigurano to da svaki element niza words ima značenje sam za sebe, programskim kodom 4.25 je pokazano kako se određuje kojoj varijabli element pripada.

Programski kod 4.25: Određivanje kojoj varijabli riječ pripada

```
foreach($words as $word) {
      if(in_array($word, array_keys($operators))) {
            if($operators[$word] == 0) {
                  $operators[$word] += 1;
                  $n1 += 1;
```

```
            } else {
                  $operators[$word] += 1;
            }
            $N1 += 1;
      } else {
            if($word == ")") {
                  continue;
            }
            if (!array_key_exists($word, $operands)) {
                  $operands[$word] = 0;
            }
            if($operands[$word] > 0) {
                  $operands[$word] += 1;
            } else {
                  $operands[$word] = 1;
                  $n2 += 1;
            }
            $N2 += 1;
      }
}
```

Pomoću foreach petlje, prolazi se kroz sve elemente niza words i odrađuje se nekoliko provjera kako bi se utvrdilo pripada li element operatorima ili operandima. Budući da smo definirali operatore u Python jeziku u operators nizu, provjeravamo nalazi li se iterator – $word unutar array-a. Ako je iterator dio niza, i ako je vrijednost uz taj iterator jednaka nuli, to znači da je pronađen jedinstveni operator (η_1 – $n1). Ukoliko se riječ nalazi unutar operators niza i ako je vrijednost uz ključ veća od 0, to znači da navedena riječ nije jedinstvena. Ukupni broj operatora se pribraja svakim pronalaskom elementa operators niza. Pravilo je da je operand sve ono što nije operator, stoga za svaku riječ koja nije pronađena unutar niza operatora, stvaramo novi element niza $operands. Na jednak način kao i kod operatora, provjeravamo je li već u nekoj prijašnjoj iteraciji riječ zabilježena kao operand – ako nije, pribraja se broju jedinstvenih operanda.

Nadalje, sve Halstead metrike se mogu izračunati ako je poznat broj $n1, $n2, $N1, $N2 na način spomenut u poglavlju 3.2.

Drugi izračun je ciklomatska složenost koja se prema relaciji (3.18) može izračunati na jednostavan način: potrebno je prebrojati sve uvjete u kodu (predikati) i nadodati jedan, što je implementirano na sljedeći način prikazano programskim kodom 4.26. Određeni su predikati, tj. uvjeti, koji se mogu pronaći u programskom jeziku Python te se provjerava nalazi li se jedan od uvjeta u liniji koda.

Programski kod 4.26: Izračun ciklomatske složenosti

```
$P = 0;
$cyclomatic = array('if', 'else', 'elif', 'and', 'or', 'while', 'for');

if(in_array($word, $cyclomatic)) {
      $P += 1;
}
```

Definiranje u koju će kategoriju koji programski jezik ući određeno je pomoću prepoznavanja ključnih riječi unutar linije koda te težinskim prosjekom. Pojedine kategorije definirane su pomoću Regular Expressiona na način prikazan programskim kodom 4.27:

Programski kod 4.27: Definiranje kategorija pomoću Regular Expressiona

```
$loop = array(
      "for" => "/^(for)\s(.*)\s(in)\s(range)(.*):/",
      "for2" => "/^(for)\s(.*)\s(in)\s(.*):/",
      "while" => "/^(while)\s(.*):/"
);

$condition = array(
      "if" => "/^(if)(.*):/",
      "elif" => "/^(elif)(.*):/",
      "else" => "/^(else):/",
      "ifelse" => "/(.*)(if)\s(.*)\s(else)/"
);

$data = array(
      "list" => "/^(.*)=\s[[](.*)[]]/",
      "list2" => "/^(.*)=\s(list)(.*)/",
      "tuple" => "/^(.*)=\s[(](.*)[)]/",
      "tuple2" => "/^(.*)=\s(tuple)(.*)/",
      "set" => "/^(.*)=\s[{](.*)[}]/",
      "set2" => "/^(.*)=\s(set)(.*)/",
      "dictionary" => "/^(.*)=\s[{](.*):(.*)[}]/"
);

$basic = array(
      "print" => "/^print((.*))/",
      "variable" => "/^(.*)\s(=)\s(.*)/"
);

$class = "/^(class)\s(.*):/";
$function = "/^(def)\s(.*):/";
```

Nakon što je definirano kojim će se izrazom pronaći koja kategorija, programskim kodom 4.28 je prikazano koja se metoda koristi za pronalazak Regular Expressiona unutar linije programskog koda.

Programski kod 4.28: Pretraga linije koda Regular Expressionom

```
$category = "";
$categoryStats = array(
      "Class" => 0,
      "Function" => 0,
      "Basic" => 0,
      "Dataset" => 0,
      "Loop" => 0,
      "Condition" => 0
);
…
if(preg_match_all($class, $line) != 0) {
      $categoryStats["Class"] += 25;
}
if(preg_match_all($function, $line) != 0) {
      $categoryStats["Function"] += 20;
}
if(preg_match_all($basic["print"], $line) != 0 ||
preg_match_all($basic["variable"], $line) != 0) {
      $categoryStats["Basic"] += 1;
}
…
$maxVal = max($categoryStats);
$maxKey = array_search($maxVal, $categoryStats);

$category = $maxKey;
```

Ako program uspije prepoznati neku od kategorija u liniji programskog koda, vrijednosti $categoryStats se mijenjaju prema kategoriji koja je otkrivena metodom preg_match_all. Budući da su „Basic“ isječci koda vrlo česti i subjektivno najjednostavniji elementi programskog jezika, oni imaju najmanju težinu u definiranju kategorije. U ovoj aplikaciji, vrijednosti pri definiranju kategorije su raspoređene na način prikazan u tablici 4.5 te se programski kod, na koncu, pridjeljuje onoj kategoriji koja ima najveću vrijednost.

Tablica 4.5: Težine pojedine kategorije

Basic	1
Condition	5

Loop	10
Dataset	15
Function	20
Class	25

4.4.3. Provjera točnosti predanog zadatka

U prvom dijelu koda, vidljivo na programskom kodu 4.4, izvršava se izračun kompleksnosti koda kojeg je korisnik napisao i predao. Izračun te rad funkcije calculate predstavljeno je u poglavlju 4.5.2.

Programski kod 4.29: Procesiranje predanog

```
$user_resp = calculate($user_code);
$user_result = json_decode($user_resp);

$user_process = array();
$user_process['H'] = $user_result->calculation->complexity->halstead-
>H;
$user_process['C'] = $user_result->calculation->complexity-
>cyclomatic;
```

Nakon što funkcija calculate izračuna i vrati vrijednost, program sprema Halstead i ciklomatsku složenost u dvije varijable: $user_process['H'] i $user_process['C'].

Podaci o kompleksnosti i kategoriji zadanog zadatka se već nalaze bazi podataka jer su mjerenja odrađena prilikom unosa zadatka u bazu preko *admin* panela. Na taj način, SQL naredbom se dohvaća očekivani output te H i C kompleksnost.

Usporedba započinje definiranjem dopuštenih granica pogreške na način prikazan programskim kodom 4.30:

Programski kod 4.30: Dopuštene granice pogreške

```
$allowed_min_H = $admin_process['H'] - $admin_process['H'] * 10;
$allowed_max_H = $admin_process['H'] + $admin_process['H'] * 10;
```

```
$allowed_min_C = $admin_process['C'] - $admin_process['C'] * 2;
$allowed_max_C = $admin_process['C'] + $admin_process['C'] * 2;
```

Uz provjeru nalazi li se kompleksnost korisničkog koda unutar granica kompleksnosti, provjerava se još je li *output* korisničkog koda jednak kao onaj u bazi, tj. *output* admina, vidljivo na programskom kodu 4.31:

Programski kod 4.31: Provjera ispravnosti korisničkog koda

```
if(strcmp($user_output, $admin_output) == 0)  {
      if(($user_process['H'] > $allowed_min_H) && ($user_process['H']
< $allowed_max_H)) {
            if(($user_process['C'] > $allowed_min_C) &&
($user_process['C'] < $allowed_max_C)) {
                  return True;
            } else {
                  return False;
            }
      }
} else {
      return False;
}
```

Ukoliko korisnik riješi zadatak točno, ali s odstupanjem manjim od 10% kod Halsteada i 2% prema ciklomatskoj složenosti, zadatak će biti prihvaćen. Tada se u tablicu Solved upisuju podaci o korisniku, koji je zadatak riješio te koji je programski kod unio.

U svrhu razvoja aplikacije za interaktivno programiranje, osmišljeno je i da se korisniku automatski zada idući zadatak koji će biti teži od prethodnog. To je omogućeno pozivom funkcije f_higher_complexity_task koja određuje koji bi zadatak i kategoriju korisnik trebao dobiti s obzirom na kompleksnost i kategoriju zadatka kojeg je zadnjeg riješio.

Funkcija određuje idući zadatak tako što prvo određuje kategoriju iz koje će zadatak biti izabran na način koji je prikazan u programskom kodu 4.32:

Programski kod 4.32: Određivanje iduće kategorije

```
$next_category = "";
switch ($category) {
      case "Basic":
            $next_category = "Dataset";
            break;
      case "Dataset":
            $next_category = "Condition";
            break;
      case "Condition":
            $next_category = "Loop";
            break;
      case "Loop":
            $next_category = "Function";
            break;
      case "Function":
            $next_category = "Class";
            break;
      case "Class":
            $next_category = "Basic";
            break;
      default:
            break;
}
```

Dakle, u slučaju da je kategorija zadnjeg zadatka bila *Loop*, iduća kategorija će biti *Function*, a onda ako korisnik uspješno riješi zadatak, nakon kategorije *Function*, zadat će mu se zadatak iz *Class* kategorije, itd. Svrha aplikacije je napredak korisnika stoga kompleksnost novog zadatka mora biti veća od prethodnog, a to se definira tako da je idući zadatak 1% – 10% kompleksniji.

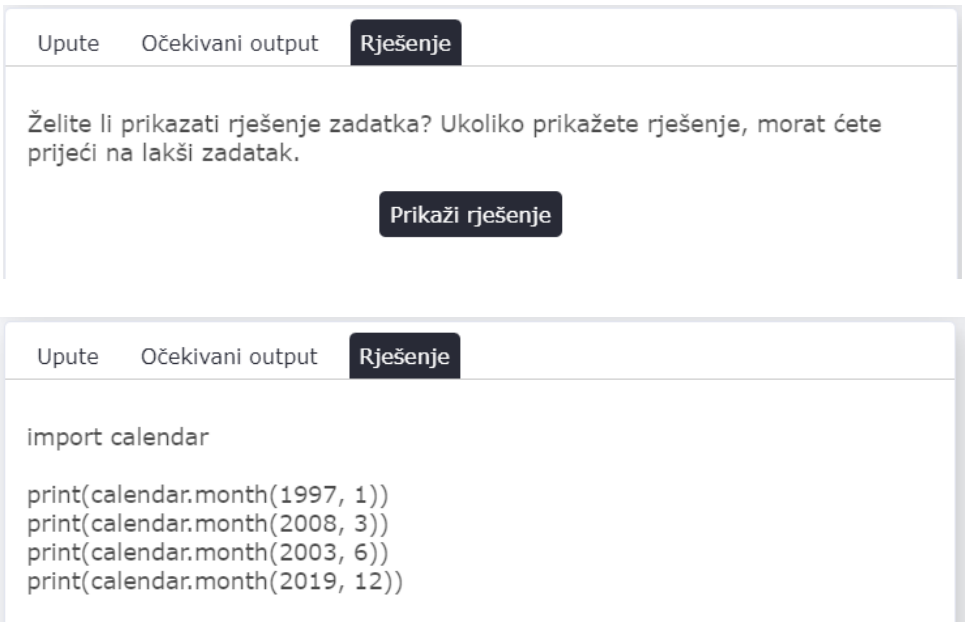
Kada se definiraju svi potrebni parametri, izvršava se SQL naredba koja kao odgovor vraća ID idućeg zadatka. SQL naredbu je moguće vidjeti u programskom kodu 4.33:

Programski kod 4.33: SQL naredba za odabir idućeg zadatka

```
$sql2 = "SELECT id
```

```
FROM    (
        SELECT *
        FROM tasksbu
        ORDER BY DBMS_RANDOM.RANDOM)
WHERE
        rownum < 2 and
        category = '{$next_category}' and
        H BETWEEN '{$min_H}' AND '{$max_H}'";
```

Također, potrebno je i uzeti u obzir i opciju da će se dogoditi da korisnik ne zna riješiti zadatak. U tom slučaju, korisnik jednostavno pristupa rješenju u trećem prozoru prikazano slikom 4.9.

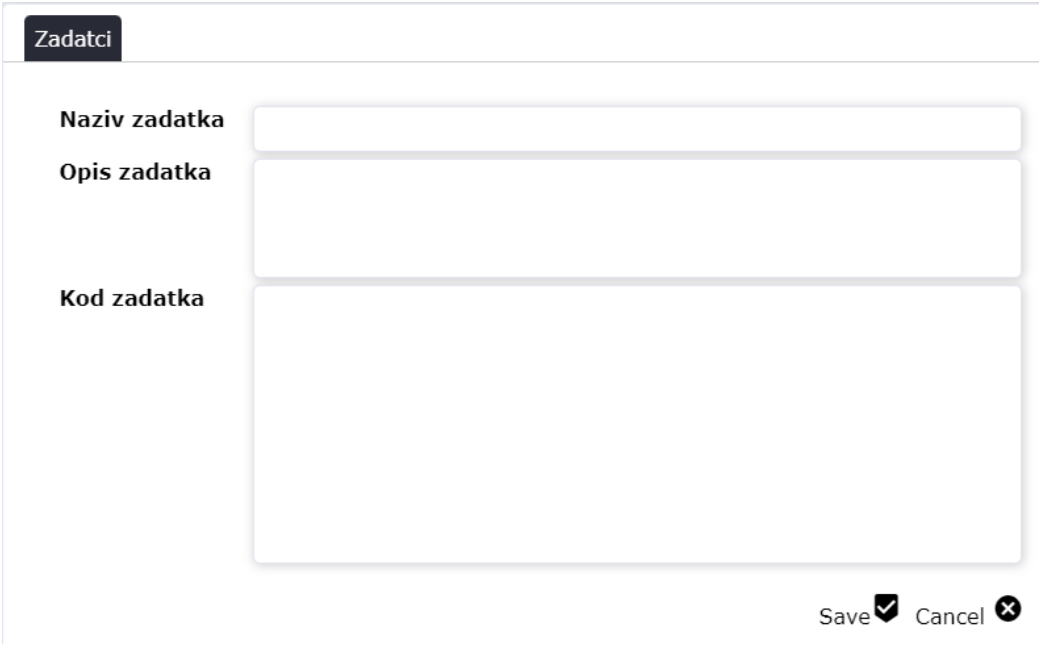


Slika 4.9: Otkrivanje rješenja zadatka

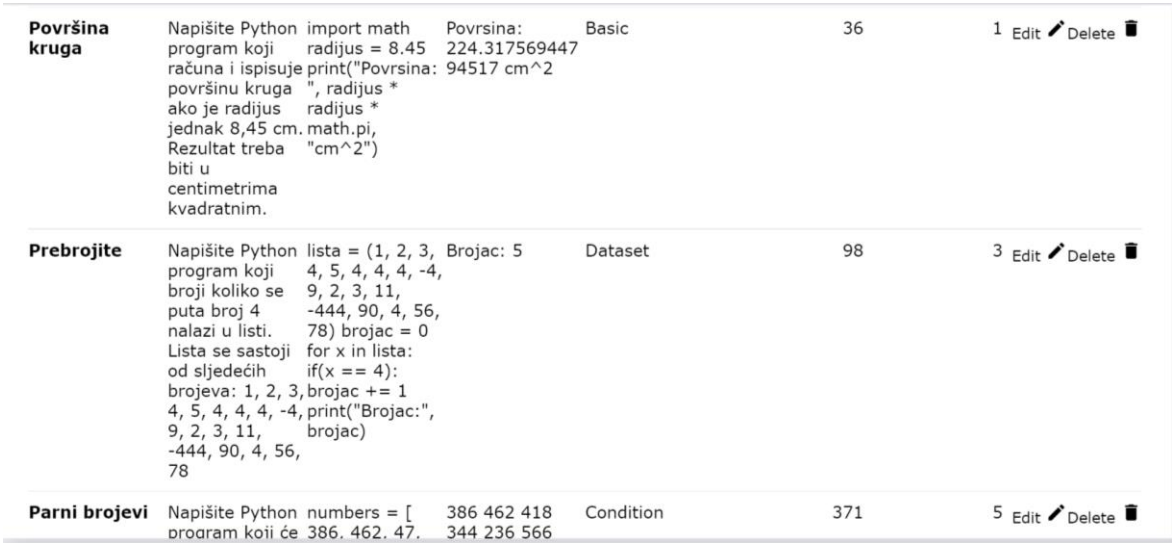
Nakon što korisnik otkrije rješenje, više nema opciju ponovne predaje zadatka, a ni prelaska na kompleksniji zadatak, već prelazi na jednostavniji, tj. manje kompleksan zadatak. Manje kompleksan zadatak se zadaje funkcijom f_lower_complexity_task koja radi po principu sličnom kao i f_higher_complexity_task, ali se traži zadatak manje kompleksnosti od prethodnog te da iduća kategorija bude kategorija „unazad“ od one kao u programskom kodu 4.33.

Osim ovog predstavljenog, postoji još nekoliko implementiranih opcija unutar ove aplikacije Neke od njih su kontrolna ploča u kojoj se mogu pregledati svi zadaci te unositi novi, a

naravno i izmjenjivati i brisati postojeći. Na slikama 4.10 i 4.11 moguće je vidjeti kontrolnu ploču te kako izgleda unos novog zadatka u bazu podataka.



Slika 4.10: Unos zadatka u bazu podataka

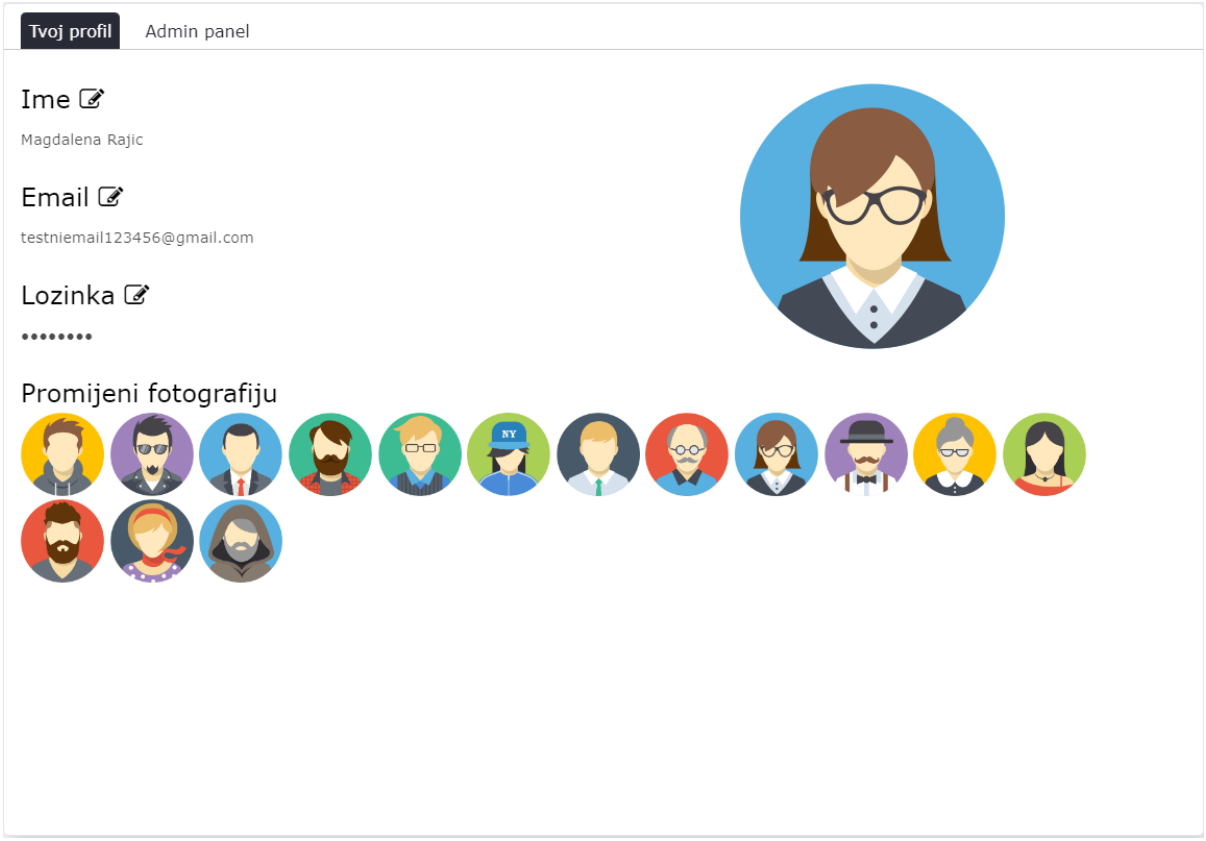


Površina kruga	Napišite Python program koji računa i ispisuje površinu kruga ako je radijus jednak 8,45 cm. Rezultat treba biti u centimetrima kvadratnim.	import math radijus = 8.45 print("Povrsina: ", radijus * radijus * math.pi, "cm^2")	Povrsina: 224.317569447 94517 cm^2	Basic	36	1 Edit Delete
Prebrojite	Napišite Python program koji broji koliko se puta broj 4 nalazi u listi. Lista se sastoji od sljedećih brojeva: 1, 2, 3, 4, 5, 4, 4, 4, -4, 9, 2, 3, 11, -444, 90, 4, 56, 78	lista = (1, 2, 3, 4, 5, 4, 4, 4, -4, 9, 2, 3, 11, -444, 90, 4, 56, 78) brojac = 0 for x in lista: if(x == 4): brojac += 1 print("Brojac:", brojac)	Brojac: 5	Dataset	98	3 Edit Delete
Parni brojevi	Napišite Python program koji će	numbers = [386, 462, 47,	386 462 418 344 236 566	Condition	371	5 Edit Delete

Slika 4.11: Prikaz svih zadataka

Izrađena je i profilna stranica na kojoj korisnik može mijenjati svoje podatke kao što su ime, prezime, e-mail, lozinku i profilnu fotografiju. Kako izgleda Tvoj profil moguće je vidjeti na slici 4.12. Unosom novog imena i lozinke ili odabirom nove fotografije, poziva se PHP funkcija koja izvršava SQL naredbu te mijenja tablicu Users. Ako je promjena u bazi uspješna, mijenja se

i prikaz kod korisnika pomoću JavaScript funkcije. U slučaju da korisnik želi promijeniti e-mail adresu, poziva se PHP funkcija koja šalje verifikacijski kod na novu e-mail adresu te sprema istu adresu pomoću SQL naredbe u tablicu Users u polje TEMPEMAIL. Nakon što korisnik otvori verifikacijsku poveznicu, izvršava se SQL naredba kojom se zamjenjuje atribut EMAIL i TEMPEMAIL u bazi podataka. Ovim korakom je e-mail adresa uspješno promijenjena.



Slika 4.12: Tvoj profil

5. ZAKLJUČAK

Aplikacija za interaktivno programiranje u Pythonu je kreirana s ciljem postupnog učenja programskog jezika Python kod početnika. Zbog velikog broja zadataka koji se nalaze u bazi, različitih kategorija i kompleksnosti, omogućen je razvoj vještina programiranja u Pythonu do visoke razine. Vidljivo je da je glavni naglasak Završnog Rada bila razrada teme o kompleksnosti koda. Budući da je dokazano da kompleksnost koda igra veliku ulogu u razvoju te održavanju softvera, fokus je bio na dvjema poznatim metrikama za analizu kompleksnosti koda: Halstead i ciklomatska (McCabe) složenost.

Ova aplikacija je trenutno u prvoj verziji jer opsežnost ove teme pruža mnogo preinaka i unapređenja. Kao što je predstavljeno kroz cijeli projekt, provjera točnosti riješenog zadatka se provjerava kroz dvije usporedbe. Prvi način je usporedba kompleksnosti koda korisnika i onog u bazi te provjera jesu li *outputi* korisnika i podatka u bazi jednaki. Naravno, postoji više načina za provjeru je li zadatak riješen prema zadanim smjernicama. Drugi način i daljnji cilj ove aplikacije je razvoj provjere koda kroz *test case*-ove. Na taj način će se način moći uzeti u obzir i kompleksniji zadaci s još većom interaktivnosti. Ukoliko se razvije druga verzija aplikacije, provjera koda će se odrađivati nizom provjera. Odrađivat će se pozivi prema Judge0, gdje će se provjeravati radi li zadatak prema očekivanjima za svaki dani primjer – u odnosu na input koji se unese.

6. LITERATURA

[1] H. Z. Došilović i I. Mekterović, Robust and Scalable Online Code Execution System, *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2020, pp. 1627-1632, doi: 10.23919/MIPRO48935.2020.9245310.

[2] https://github.com/judge0/judge0#references (28.7.2022.)

[3] https://www.oracle.com/database/sqldeveloper/ (26.7.2022.)

[4] Groff J.R., Weinberg P.N. SQL: The Complete Reference. Osborne/McGraw-Hill; 1999. str. 8 – 11, 26., 575., 608.

[5] Brooks D.R. Programming in HTML and PHP – Coding for Scientists and Engineers. Springer International Publishing; 2017.

[6] Macaulay M. Introduction to Web Interaction Design – With HTML and CSS. Taylor & Francis Group, LLC; 2018.

[7] Flanagan D. JavaScript: The Definitive Guide, Sixth Edition. O'Reilly Media, Inc.; 2011.

[8] Flanagan D. JavaScript: The Definitive Guide, Seventh Edition. O'Reilly Media, Inc.; 2020.

[9] https://www.javatpoint.com/json-example (3.8.2022.)

[10] Antinyan V., Staron M., Sandberg A. Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time, Empir Software Eng, 9 March 2017

[11] Banker R.D., Datar S.M., Kemerer C.F, Zweig D. Software Complexity and Maintenance Costs, Communications of the ACM, 1993

[12] Govil N. Applying Halstead Software Science on Different Programming Languages for Analyzing Software Complexity, IEEE Xplore Part Number: CFP20J32-ART; ISBN: 978-1-7281-5518-0, 2020

[13] McCabe T.J. A Complexity Measure, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-2, NO. 4, DECEMBER 1976

[14] https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity (1.7.2022.)

[15] Mills H.D. Mathematical foundations for structured programming, Federal System Division, IBM Corp., Gaithersburg, MD, FSC 72-6012, 1972.

[16] McCabe T.Jr Software Quality Metrics to Identify Risk, Department of Homeland Security – Software Assurance Working Group, 2008

[17] Antoniol G., Laplante P., Counsell S. Cyclomatic Complexity, IEEE SOFTWARE, November/December 2016

[18] Fotografija: http://www.freepik.com, autor: vectorjuice

[19] https://www.ibm.com/docs/en/sva/9.0?topic=overview-session-state-concepts (31.8.2022.)

7. OZNAKE I KRATICE

PL – proceduralni jezik (eng. *Procedural Language)*

SQL – strukturni upitni jezik (eng. *Structured Query Language*)

PHP – eng. *Hypertext Preprocessor*

JS – eng. *JavaScript*

HTML – eng. *Hypertext Markup Language*

CSS – eng. *Cascading Style Sheets*

OJ – eng. *Online Judge*

CPU – središnja procesorska jedinica (eng. *Central Processing Unit*)

CEE – alat za izvršavanje koda (eng. *Code execution engine*)

OCES – online sustav za izvršavanje koda (eng. *Online code execution system*)

API – aplikacijsko programsko sučelje (eng. *Application Programming Interface*)

IDE – integrirano razvojno okruženje (eng. *Integrated Development Environment*)

JDK – eng. *Java Development Kit*

ID – identitet (eng. *Identity*)

JMBAG – Jedinstveni matični broj akademskog građana

WWW – eng. *World Wide Web*

W3C – konzoricij World Wide Weba (eng. *The World Wide Web Consortium*)

Ajax – asinkroni JavaScript i XML (eng. *Asynchronous JavaScript and XML*)

JSON – zapis JavaScript objekta (eng. *JavaScript Object Notation*)

LoC – linije koda (eng. *Lines of Code*)

CCM – mjerenja ciklomatske složenosti (eng. *Cyclomatic Complexity Measures*)

HCM – mjerenja Halstead složenosti (eng. *Halsetad Complexity Measures*)

XAMPP – eng. *Cross-platform, Apache, MySQL, PHP and Perl*

8. SAŽETAK

Naslov: Aplikacija za interaktivno programiranje u Pythonu

Kroz ovaj završni rad opisano je korištenje različitih tehnologija često korištenih u području web razvoja. Cilj rada je izrada aplikacije za interaktivno učenje programiranja kojom se omogućava razvoj vještina programiranja od početničkih do naprednijih razina. Predstavljene su metode, tj. metrike za izračun kompleksnosti koda koje su zastupljene u svijetu računarstva od 1970-ih godina. Halstead i McCabe metrikom omogućena je provjera točnosti riješenog zadatka i odabir idućeg zadatka prema napretku korisnika. Također, prikazano je dizajnirano razvojno okruženje u kojem korisnici testiraju napisan kod te kontrolna ploča za admin korisnike i profilna stranica za korisnike.

Ključne riječi: interaktivno programiranje, web aplikacija, složenost koda

9. ABSTRACT

Title: Application for interactive programming in Python

Through this thesis, the use of various technologies often used in the field of web development is described. The goal of the thesis is to create an interactive application for learning that enables the development of programming skills from beginner to advanced levels. Methods, i.e. metrics for calculating code complexity, which have been present in the world of computing since the 1970s, are presented. Halstead and McCabe metrics enable checking the accuracy of the solved task and selecting the next task according to the user's progress. Also, a designed development environment in which users test written code is shown, as well as a control panel for admin users and a profile page for other users.

Ključne riječi: interactive programming, web application, code complexity

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 15. rujna 2022.	MAGDALENA RAJIĆ	Magdalena Rajić

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

MAGDALENA RAJIĆ

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 15. rujna 2022.

Magdalena Rajić

potpis studenta/ice