Univerzitet u Beogradu

Fakultet organizacionih nauka

Primena savremenih alata u procesu automatizacije razvoja softvera

Katedra za softversko inženjerstvo | Automatizacija razvoja softvera

Mentor:

dr Miloš Milić

Student:

Dušan Tavić 2023/3801

Beograd,

2024.

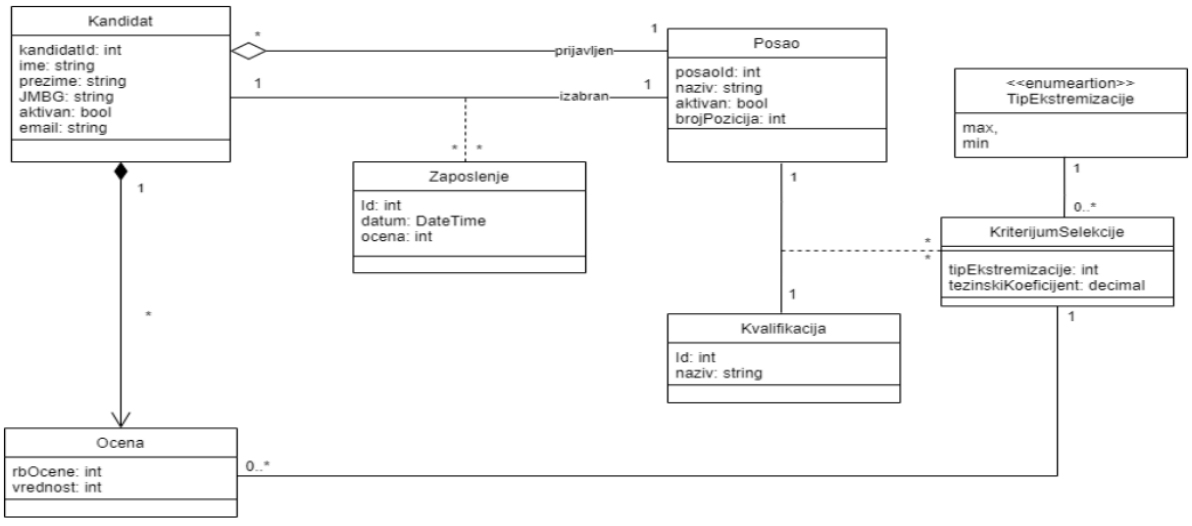
Sadržaj

Konceptualni (domenski) model sistema

Na slici 1 dat je prikaz konceptualnog (domenskog) modela sistema. Svaki od prikazanih domenskih obejkata implementiran je u kasnijim fazama u vidu konkretnih klasa u .NET okruženju i C# programskom jeziku. Na slici 1 prikazani su osnovni domenski entiteti, kao što su kandidati, poslovi, zaposlenja ili kvalifikacije.

Kandidati apliciraju za rad na nekoj poziciji i time stupaju u proces selekcije. Za svakog kandidata čuva se informacija o poslu za koji je aplicirao i o poslu za koji je kasnije izabran, ukoliko uspešno prođe sve krugove selekcije. Asocijativna klasa koja predstavlja zaposlenje povezana je sa jednim kandidatom i jednim poslom i može sadržati ocenu zaposlenja koja je prilikom kasnijeg vrednovanja dodeljena konkretnom kandidatu. Ova ocena zaposlenja može se koristiti za analizu uspešnosti procesa selekcije u cilju preduzimanja korektivnih mera ukoliko se za tim javi potreba. Zaposlenje sadrži i datum zaposlenja, koji omogućava da se implementira ograničenje kojim se sprečava da zaposleni bude vrednovan u prvih šest meseci svog rada.

Za svaki posao definišu se kriterijumi koji će biti razmatrani prilikom selekcije kandidata. Kriterijum selekcije predstavlja asocijativnu klasu koja nastaje iz veze između posla i kvalifikacije i sadrži dodatne informacije o tipu ekstremizacije i težinskom koeficijentu. Tip ekstremizacije predstavlja osnovni cilj razmatranog kriterijuma i može uzeti dve vrednosti: maksimizacija vrednosti kriterijuma ili njegova minimizacija. Nije nam za sve kriterijume cilj da njihova vrednost bude što veća, te je potrebno ostvariti jasan mehanizam kojim će se praviti razlika između ciljeva.



Slika 1. Konceptualni (domenski) model

Svaki kandidat se ocenjuje prema kriterijumima selekcije posla za koji je aplicirao. Za svakog kandidata u sistemu beleže se ocene koje je dobio od strane menadžera u sektoru za upravljanje ljudskim resursima. Jedna ocena odnosi se na tačno jednog kandidata i tačno jedan kriterijum selekcije. Važno je primetiti da je ocena kandidata egzistencijalno i identifikaciono zavisna od samog kandidata, te da bez njega ne bi trebalo da postoji u sistemu. Na osnovu zabeleženih ocena sistem u kasnijim fazama daje predlog kandidata koji najviše odgovaraju datoj poziciji, uz implementaciju principa iz teorije odlučivanja koji se odnose na višekriterijumsku analizu odlučivanja.

Relacioni model

Konceptualni model prikazuje strukturu sistema. U svojoj osnovi on opisuje konceptualne klase domena problema i prikazuje domenske objekte definisane u okviru sistema i asocijacije između njih. Konceptualni model predstavlja model objektne analiize. Sa druge strane, relacioni model je podržan SQL upitnim jezikom i jednostavnim preslikavanjem može biti definisan na osnovu konceptualnog modela. Na osnovu relacionog modela vrši se konkretna implementacija sistema na nivou relacionih baza podataka. Na slici 2 dat je prikaz relacionog modela definisanog na osnovu konceptualnog modela prikazanog na slici 1.

Kandidat (kandidatId, ime, prezime, JMBG, aktivan, email, *posaoId*)

Posao (posaoId, naziv, aktivan, brojPozicija)

Zaposlenje *(kandidatId, posaoId*, Id, datum, ocena)

Ocena (*kandidatId* , rbOcene, vrednost, *kriterijumSelekcijeId*)

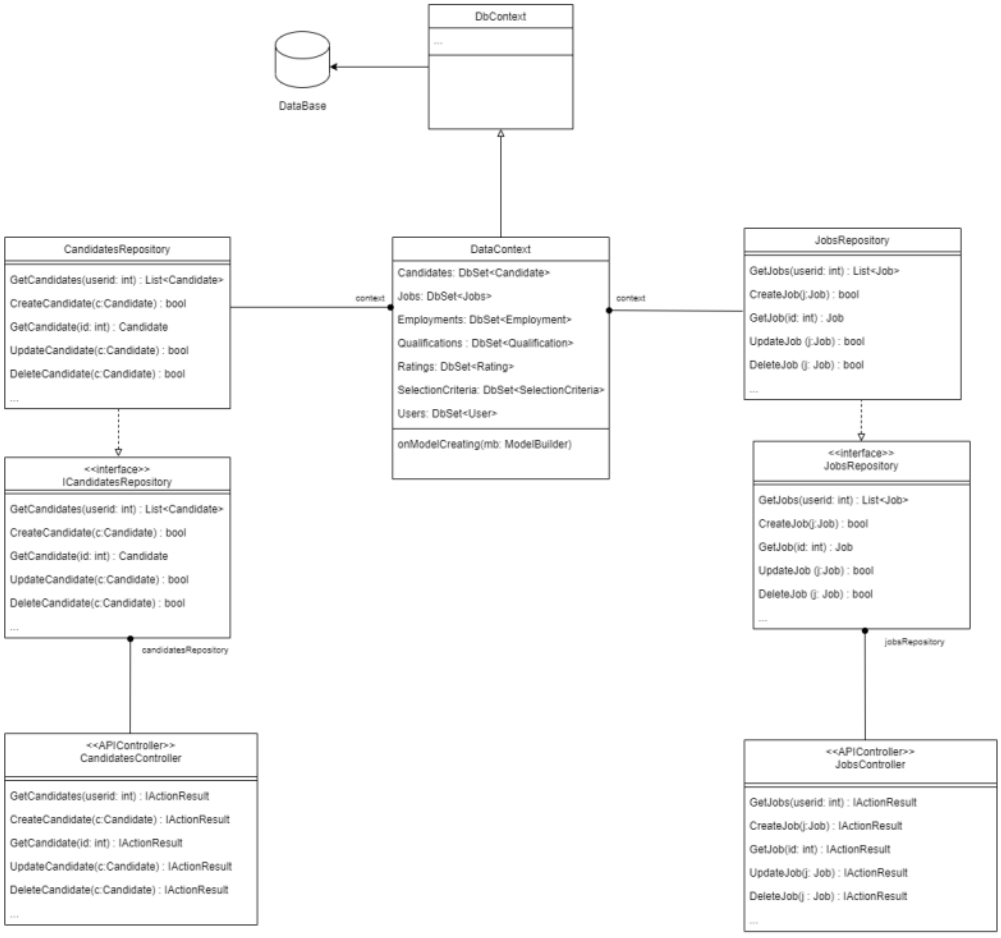
Kvalifikacija (kvalifikacijaId, naziv)

KriterijumSelekcije(*posaoId, kvalifikacijaId,* tipEkstremizacije, tezinskiKoeficijent)

Slika 2. Relacioni model

Prikaz arhitekture sistema

Kao što je prikazano na slici 3, celokupna komunikacija sa ASP.NET Web API-em započinje sa klasom EntityController, koja nasleđuje klasu Controller, i predstavlja ulaznu tačku u komunikaciji klijenta sa serverom. Ova klasa implementirana je za svaki entitet u sistemu, ali ćemo se ovde radi preglednosti dijagrama fokusirati na dva osnovna entiteta, a to su Candidate i Job. Klasa CandidateController sadrži niz metoda za manipulaciju podacima o kandidatima koji su smešteni u relacionoj bazi podataka, od kojih su na dijagramu prikazane osnovne CRUD metode. Važno je napomenuti da klasa CandidateController ne implementira direktno ove metode, već u tu svrhu poziva metode klase CandidatesRepository koja sadrži polje klase tipa DataContext za direktnu komunikaciju sa relacionom bazom podataka. Klasa DbContext smatra se osnovnom gradivnom klasom u projektovnaju i impementaciji arhitekture objektnog programa koji koristi Entity Framework za komunikaciju sa bazom podataka. Ova klasa reprezentuje sesiju sa bazom podataka i omogućava nam izvršavanje osnovnih operacija za manipulaciju podacima u bazi.



Slika 3. Prikaz arhitekture sistema

Jedinično testiranje i xUnit alat

Testiranje softvera danas predstavlja jednu od ključnih aktivnosti za postizanje i održavanje kvaliteta softvera. Osnovni zadatak aplikacionih testova je postizanje kvaliteta funkcija i pouzdanosti rezultata koji se isporučuju prezentacionom sloju direktno do krajnjeg korisnika. Testovima na aplikacionom nivou treba obezbediti integritet rezultata sistemskih operacija i zaštitu prodiranja neispravnih ili neželjenih podataka do baze podataka.

Na aplikacionom sloju kao implementaciona tehnologija korišćen je ASP.NET Web API, programski jezik C# i Entity Framework kao objektno relacioni maper. Za testiranje aplikacionog sloja korišćen je xUnit okvir.

Okvir xUnit predstavlja jedan od danas najpopularnijih okvira za testiranje softvera, koji se koristi za pisanje i izvršavanje automatskih testova u programskom jeziku C#. Alat xUnit razvijen je kako bi omogućio programerima da jednostavno izvršavaju svoje testove u razvojnom okruženju. Automatizacija testiranja treba da pruži okruženje koje istovremeno podržava jednostavno i ekspresivno pisanje testova.

U savremenim softverima složenost implementiranog sistema može prevazilaziti mogućnosti ručnog testiranja softverskih komponenti, pogotovo uzimajući u obzir brzinu kojom se softverski sistem menja i nadograđuje. U uslovima dinamike tržišta, virtualizacije timova i globalizacije u saradnji prilikom razvoja softverskih sistema, gotovo je nemoguće obebzediti ručno testiranje kojim bi stabilnost sistema bila zagarantovana na dužem vremenskom horizontu.

```
[Fact]
public void CandidatesController_GetCandidates_ReturnsOk()
{
  //Arrange
  int userId = 1;
  var candidatesFromDb = A.Fake<List<Candidate>>();
  var candidates = A.Fake<List<CandidateDto>>();
  A.CallTo(() => candidatesRepository.GetCandidates(userId)).Returns(candidatesFromDb);
  A.CallTo(() => mapper.Map<List<CandidateDto>>(candidatesFromDb)).Returns(candidates);
  var controller = new CandidatesController(candidatesRepository, jobsRepository,
                                            usersRepository, mapper);

  //Action
  var result = controller.GetCandidates(userId);

  //Assert
  result.Should().NotBeNull();
  result.Should().BeOfType(typeof(OkObjectResult));
}
```

Slika 4. Osnovni prikaz xUnit testa

Na slici 4 dat je osnovni prikaz xUnit testa definisanog u C# programskom jeziku. Prema konvenciji, svaki xUnit test sastoji se od tri gradivna elementa i to: priprema, radnja i provera (eng. arrange, act and assert). Priprema (eng. Arrange) je deo testa koji podrazumeva pripremu svih potrebnih pretpostavki ili uslova koji su neophodni za izvršavanje testa, a što može uključitavi inicijalizaciju objekata, postavljanje početnih vrednosti, podešavanje ulaznih parametara i druge aktivnosti za pripremu izvršavanja testa. U fazi radnje (eng. Act) definišu se akcije koje se izvršavaju i koje se testiraju. Ova faza predstavlja suštinu testa i treba biti definisana na nedvosmislen i jasan način. Deo za akcije, odnosno radnje, najčešće uključuje pozivanje metoda ili operacija koju želimo da testiramo, čime je fokus premešten na testiranje izvršavanja konkretne funkcionalnosti. Deo koji se odnosi na proveru (eng. Assert) sadrži definisane provere očekivanih i dobijenih rezultata izvršavanja koda. U ovom delu testa koriste se tvrdnje (eng. assertions) da bi povratna vrednost funkcije bila proverena i upoređena sa očekivanom.

Na slici 5 dat je prikaz metode definisane u okviru kontrolera za koju je prikazani test razvijen. Ključni koncept prilikom jediničnog testiranja jeste usaglašenost jediničnih testova sa jedinicama za koje su razvijane.

```
[HttpGet("user/{userId}")]
public IActionResult GetCandidates(int userId)
{
 var candidates = mapper.Map<List<CandidateDto>>(candidatesRepository.GetCandidates
                                                                    (userId));
  if (candidates == null)
    return NotFound();

  if (!ModelState.IsValid)
    return BadRequest(ModelState);

  return Ok(candidates);
}
```

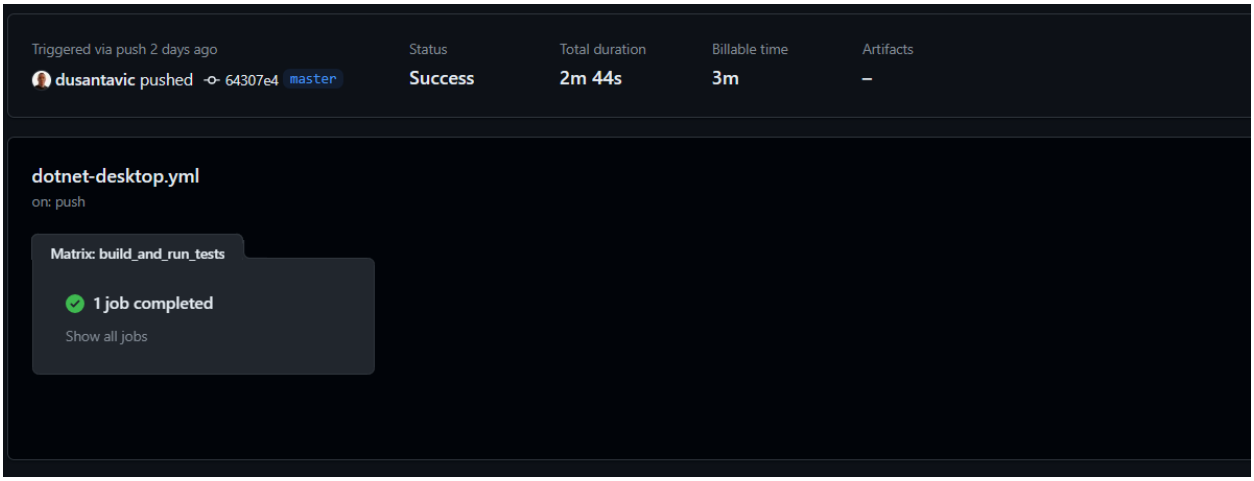
Slika 5. Prikaz kontrolerske metode za koju je razvijen test

CI/CD Pipeline i Github akcije

CI/CD Pipeline (Continuous Integration, Continuous Delivery) predstavlja metodologiju razvoja softvera koja omogućava timovima da automatski testiraju i isporučuju kod u produkcijsko okruženje. CI/CD metodologija obezbeđuje pouzdaniji razvoj softvera kroz automatizaciju procesa. Ovako definisana praksa omogućava da razvojni tim vrši redovnu integraciju svojih promena u kodu u centralni repozitorjum, što omogućava brzu povratnu informaciju o potencijalnim problemima ili nedostacima u kodu, i efikasnu saradnju timova koji mogu biti fizički udaljeni.

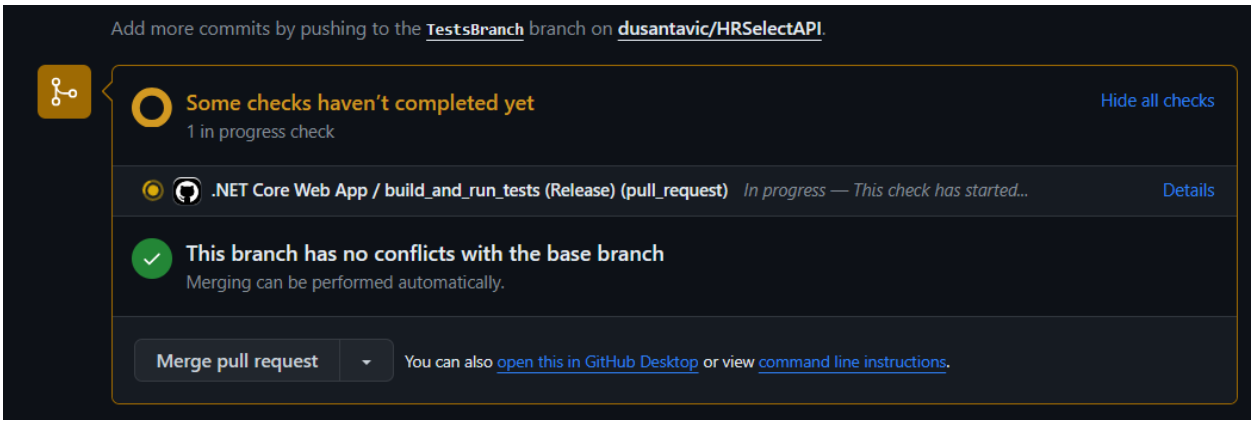
Za obezbeđivanje CI/CD Pipeline-a na GitHub-u koriste se GitHub akcije. GitHub akcije predstavljaju integrisani alat za automatizaciju koji omogućava programerima da definišu, izvršavaju i dele automatizovane procesne tokove (eng. workflows) direktno u okviru GitHub repozitorijuma. GitHub akcija definiše se YAML sintaksom, koja omogućava jasno definisanje poslove i korake u okviru poslova koji treba da se izvršavaju automatski prilikom promena u repozitorijumu. Kada se kreira GitHub repozitorijum, za svaki događaj nad tim repozitorijumom kao što je na primer push ili pull-request, procesni tok definisan YAML sintaksom pokreće se automatski.

Na slici 6 dat je prikaz automatizovane akcije koja se pokreće kada dođe do novog pull-requesta u okviru repozitorijuma. Svaki put kada se dogodi pull-request, GitHub akcija treba da se pokrene samostalno i da proveri da li su svi defisani testovi uspešno izvršeni. Ukoliko to nije slučaj pull-request bi trebalo da se uzme u razmatranje, te da se identifikovane greške isprave pre nego što se funkcionalna grana spoji sa glavnom granom u okviru repozitorijuma.



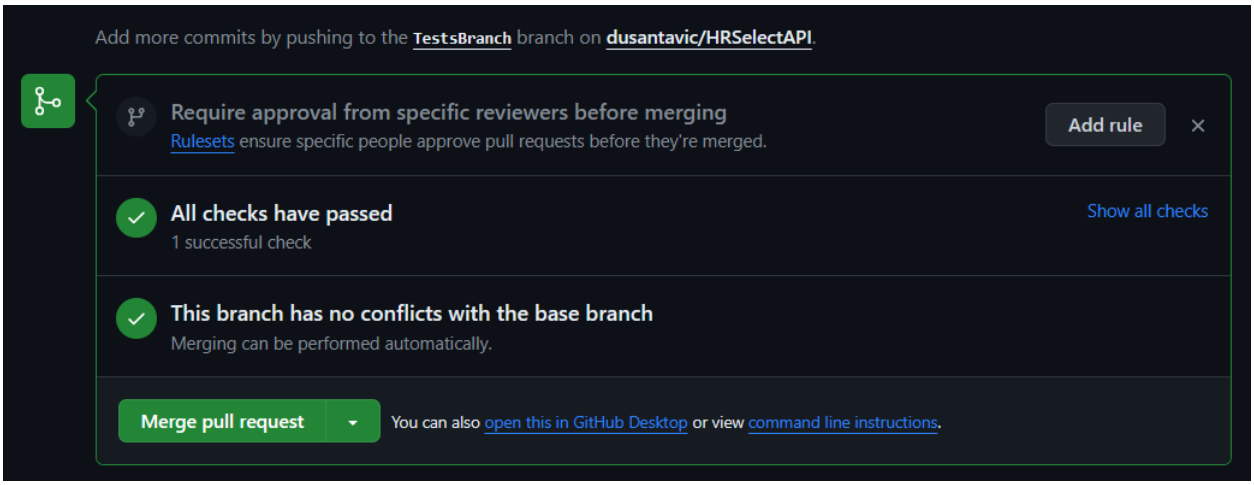
Slika 6. Prikaz GitHub akcije

Na slici 7 dat je prikaz situacije u kojoj se dogodio novi pull-request. Pre nego što je omogućeno spajanje grana, potrebno je sačekati da se GitHub akcija automatizovano izvrši i prikaže rezultat uspešnosti testova definisanih u okviru sistema.



Slika 7. Čekanje na akciju pre spajanja grana

Kao što je prikazano na slici 8, prilikom nastanka novog pull-requesta pokreću se automatizovane akcije za testiranje softverskog sistema. Tek onda kada je osigurano da su svi testovi uspešno izvršeni, omogućeno je spajanje grana (eng. Merge pull request).



Slika 8. Uspešno izvršena akcija i omogućeno spajanje

Na slici 9 dat je prikaz dela YAML fajla kojim se definiše GitHub akcija za automatsko pokretanje testova. YAML predstavlja lako čitljiv format podataka koji se često koristi za konfiguraciju i definisanje podataka koji imaju jasnu strukturu. U našem slučaju struktura je jasna i unapred poznata, jer svaka GitHub akcija definisana je on konfiguracijom, kojom se definišu događaji koji mogu da uzrokuju pokretanje akcije. Pored toga, svaka akcija sastoji se od poslova (eng. jobs), a svaki posao sastoji se od koraka (eng. steps). Ovako jasno definisana struktura omogućava efikasno korišćenje čitljjivog YAML fajla za njen opis. U okviru YAML fajla u delu env koji se odnosi na konfiguraciju okruženja, navedeni su delovi projekta koji se odnose na Solution, putanju do testnog projekta, direktorijum u kome se nalazi Web aplikacija i putanju kojom se toj Web aplikaciji pristupa.

```yaml
name: .NET Core Web App

on:
  push:
    branches: [ "master", "TestBranch" ]
  pull_request:
    branches: [ "master", "TestBranch"]

jobs:
  build_and_run_tests:
    strategy:
      matrix:
        configuration: [Release]
    runs-on: windows-latest

    env:
      Solution_Name: HRSelectAPI.sln
      Test_Project_Path: HRSelectAPI.Tests.csproj
      Wap_Project_Directory: HRSelectAPI.HRSelectAPI
      Wap_Project_Path: HRSelectAPI\HRSelectAPI.csproj.user

    steps:
    - name: Checkout
      uses: actions/checkout@v4
      with:
        fetch-depth: 0
```

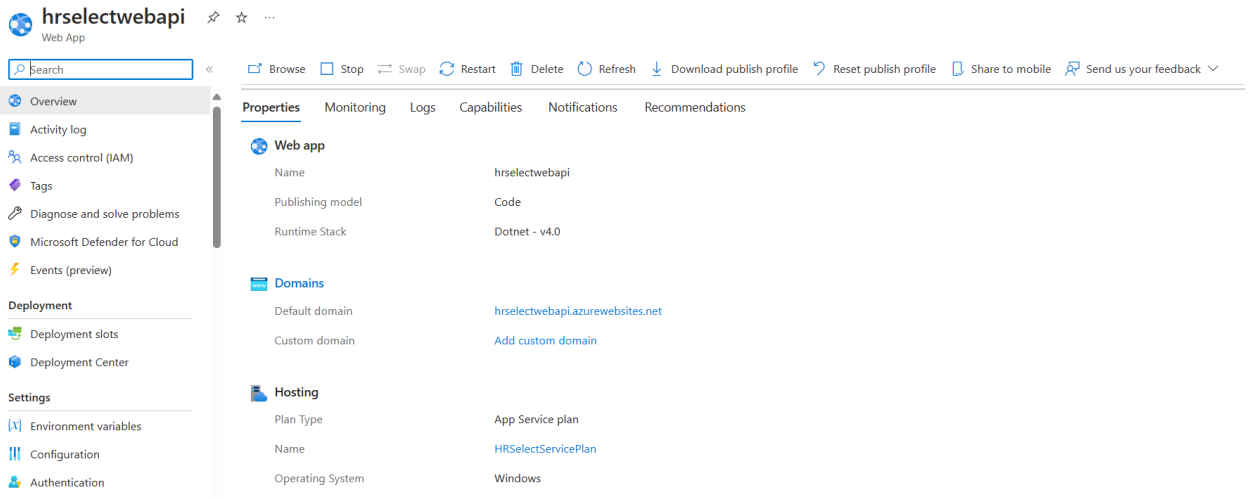
Slika 9. Prikaz dela YAML fajla

Cloud computing i Azure platforma

Cloud Computing predstavlja model koji obezbeđuje korišćenje različitih računarskih resursa putem interneta. Serveri na kojima je pokrenuta aplikacija treba da budu dovoljno moćni da podrže skalabilnost koda i pristupačnost softverskog sistema i u uslovima velikih opterećenja. Pored toga, serveri moraju biti dostupni dvadeset četiri časa dnevno, što najčešće nije moguće ili je teško izvodljivo u lokalnim uslovima. Iz tog razloga, cloud servisi kao što je Azure treba da omoguće korišćenje servera i skladištenje različitih aplikacija na serverima koje će biti pokrenute i dostupne u svakom slučajnom vremenskom trenutku, uz mogućnost istovremenog rada sa velikim brojem klijentskih aplikacija.

Azure predstavlja cloud platformu koja pruža različite vrste usluga i resursa za razvoj, testiranje, implementaciju i upravljanje aplikacaijama i servisima putem interneta. U našem primeru, Azure platforma korišćena je za deployment aplikacije. Ovim je omogućeno da pristup aplikaciji bude omogućen ne samo lokalno već i sa bilo kog udaljenog računara na kome je pokrenuta klijentska aplikacija. Umesto localhost-a, sada se koristi konkretna web adresa na kojoj je aplikacija uskladištena i pokrenuta. Korišćenjem Azure platforme aplikacija je realizovana u svom ne-lokalnom obliku i omogućena je interakcija sa različitim korisnicima širom sveta.

Na slici 10 dat je osnovni prikaz Azure platforme koji se odnosi na konkretnu aplikaciju. Važno je napomenuti da pored korišćenja računarskih resursa, Azure platforma omogućava niz drugih funkcionalnosti za upravljanje softverskim sistemom, kao što je AzureDevops alat za kontinuiranu integraciju i dostavu, upravljanje projektima i kodom, kao i niz drugih alata za praćenje performansi softverskog sistema kroz vreme.



Slika 10. Prikaz Azure platforme za web aplikaciju