|  |  |  |
| --- | --- | --- |
| http://www.tfzr.uns.ac.rs/Content/files/0/ZnakUniverziteta.bmp | Univerzitet u Novom Sadu  Tehnički fakultet “Mihajlo Pupin”  Zrenjanin | http://www.tfzr.uns.ac.rs/Content/files/0/Tehnicki_Fakultet_Mihajlo_pupin_Zrenjanin%281%29s.bmp |

**Seminarski rad**

**Evidencija nezaposlenih**

Predmet: **Internet programiranje**

|  |  |  |
| --- | --- | --- |
| Profesor: prof. dr Ljubica Kazi  Asistent: Vuk Amidzic |  | Student: Ognjen Grgur  Broj indeksa: SI 10/20 |

**Zrenjanin. 2024.**

[1. Analiza poslovnog sistema, specifikacija zahteva i planiranje razvoja 4](#_Toc169093771)

[1.1. Analiza poslovnog Sistema 4](#_Toc169093772)

[1.1.1. Svrha postojanja organizacionog sistema i kategorizacija poslova 4](#_Toc169093773)

[1.1.2. Osnovni objekti obrade i opis to ka poslovnog procesa 4](#_Toc169093774)

[1.1.3. Organizaciona struktura sistema i pripadnost drugim organizacionim strukturama 5](#_Toc169093775)

[1.1.4. Radna mesta i opis zaduženja 5](#_Toc169093776)

[1.1.5. Spisak korišćene dokumentacije 5](#_Toc169093777)

[1.1.6. Zakonska regulativa, standardi i interni propisi organizacionog sistema 5](#_Toc169093778)

[1.2. Specifikacija zahteva korisnika 6](#_Toc169093779)

[1.2.1. Profili korisnika i očekivane funkcionalne mogućnosti softvera 6](#_Toc169093780)

[1.2.2. Spisak potrebnih ulaznih ekranskih formi 6](#_Toc169093781)

[1.2.3. Spisak potrebnih tabelarnih prikaza sa kriterijumima filtriranja 6](#_Toc169093782)

[1.2.4. Spisak potrebnih štampi dokumenata (štampi, reporta), statističkih izveštaja za podršku odlučivanju i eksporta podataka 6](#_Toc169093783)

[1.2.5. Spisak poslovnih pravila, automatizama i ograničenja koje je potrebno ugaditi u softver 6](#_Toc169093784)

[1.2.6. Potrebne karakteristike korisničkog interfejsa i ostali nefunkcionalni zahtevi 7](#_Toc169093785)

[1.3. Elementi projektnog planiranja 8](#_Toc169093786)

[1.3.1. Granice (scope) projekta 8](#_Toc169093787)

[1.3.2. Kratak opis tehnologija, programskih jezika i razvojnih alata koji će biti korišćeni u radu 9](#_Toc169093788)

[2. Poslovna sistemska analiza i dizajn softverskog rеšenja 10](#_Toc169093789)

[2.1. Model poslovnih procesa 10](#_Toc169093790)

[2.1.1. Algoritamski model poslovnih procesa 10](#_Toc169093791)

[2.1.2. Spisak skladišta podataka 10](#_Toc169093792)

[2.1.3. Analiza jednog poslovnog dokumenta 11](#_Toc169093793)

[2.2. Dzajn softverskih funkcija 12](#_Toc169093794)

[2.2.1. Tabela preslikavanja primitivnih procesa u softverske funkcije 12](#_Toc169093795)

[2.2.2. USE CASE dijagram softverskih funkcija 13](#_Toc169093796)

[Specifikacija slučaja korišćenja za unos i tabelarni prikaz podataka 14](#_Toc169093797)

[2.3. Dizajn modela podataka 17](#_Toc169093798)

[2.3.1. Konceptualni model podataka 17](#_Toc169093799)

[2.3.2. Fizički (relacioni) model podataka 17](#_Toc169093800)

[3. Implementacija, testiranje i dokumentovanje realizovanog softvera 18](#_Toc169093801)

[3.1. Korisničko i tehničko uputstvo 18](#_Toc169093802)

[3.1.1. Korisničko uputstvo 18](#_Toc169093803)

[3.1.2. Tehnicko uputstvo 18](#_Toc169093804)

[3.2. Izveštaj o testiranju softvera 18](#_Toc169093805)

[3.2.1. Test slučajevi ispravnog rada korisnika i prikaz rezultata testiranja 18](#_Toc169093806)

[3.2.2. Test slučajevi neispravnog rada korisnika i prikaz rezultata testiranja 18](#_Toc169093807)

[3.3. Dokumentovanje implementacije softverskog rešenja 18](#_Toc169093808)

[3.3.1. Modeli i opisna implementacija 18](#_Toc169093809)

[3.3.1.1. Dijagram komponenti 18](#_Toc169093810)

[3.3.1.2. Dijagram klasa 19](#_Toc169093811)

[3.3.1.3. Dijagram sekvenci za osnovni scenario unosa podataka i tabelarnog prikaza sa filterom 20](#_Toc169093812)

[3.3.2. Tabelarni prikaz komponenti 21](#_Toc169093813)

[3.3.3. SQL skripta 22](#_Toc169093814)

[3.3.4. Delovi koda sa objašnjenjima po slojevima 23](#_Toc169093815)

[3.3.4.1 Sloj za rad sa podacima 23](#_Toc169093816)

[3.3.4.2. Sloj servisa ili DTO 25](#_Toc169093817)

[3.3.4.4. Kod za osnovnu validaciju podataka sloja prezentacione logike 29](#_Toc169093818)

[3.3.4.5. Korisnički interfejs 31](#_Toc169093819)

[3.3.5. Delovi koda i opis implementacije podrške personalizaciji aplikacije i bezbednosti korišćenja 34](#_Toc169093820)

[3.3.6. Delovi koda i opis implementacije osnovnih principa OOP 36](#_Toc169093821)

[3.3.7. Delovi koda i opis implementacije SOLID principa 37](#_Toc169093822)

[3.3.8. Delopvi koda, objašnjenja realizacije clean code pristupa 39](#_Toc169093823)

[4. Literatura 42](#_Toc169093824)

## Analiza poslovnog sistema, specifikacija zahteva i planiranje razvoja

## Analiza poslovnog Sistema

### Svrha postojanja organizacionog sistema i kategorizacija poslova

Svrha ovog organizacionog sistema je da obezbedi efikasno i precizno evidentiranje, praćenje i izveštavanje o statusu nezaposlenih lica. Sistem omogućava državnim institucijama, kao što su zavodi za zapošljavanje, da lako upravljaju podacima o nezaposlenima, prate njihove aktivnosti i statističke podatke o tržištu rada.

Kategorizacija poslova:

* Evidentiranje podataka: Unos i ažuriranje podataka o nezaposlenim licima, uključujući njihove lične informacije, radnu istoriju i trenutni status.
* Praćenje statusa: Praćenje promena u statusu nezaposlenih, kao što su prelazak sa nezaposlenosti u zaposlenost, obuka ili drugi programi.
* Izveštavanje: Generisanje različitih izveštaja za potrebe analize i statistike.
* Administracija: Upravljanje korisničkim nalozima, podešavanje sistema i obezbeđivanje bezbednosti podataka.

### Osnovni objekti obrade i opis to ka poslovnog procesa

* Osnovni objekti obrade:
  + Korisnici: Registrovani korisnici sistema (zaposleni u zavodima za zapošljavanje).
  + Nezaposleni: Osobe koje su prijavljene kao nezaposlene.
  + Poslodavci: Organizacije koje traže radnu snagu i objavljuju oglase za posao.
* Opis toka poslovnog procesa:
* Prijava korisnika: Korisnici se prijavljuju u sistem koristeći svoje korisničke naloge.
* Evidentiranje nezaposlenih: Unos i ažuriranje podataka o nezaposlenima.
* Pretraga nezaposlenih: poslodavci pretražuju dostupne nezaposlene.
* Pretraga nezaposlenih - Iskustvo: Poslodavac pretrazuje zaposlenie na osnovu njihovog zanimanja, kao i na osnovu toga da li su imali posla u struci.

### Organizaciona struktura sistema i pripadnost drugim organizacionim strukturama

Organizaciona struktura sistema:

* Administrator sistema: Odgovoran za upravljanje celokupnim sistemom, korisničkim nalozima i sigurnosnim postavkama.
* Korisnik: Zaposleni u zavodima za zapošljavanje koji evidentiraju i ažuriraju podatke o nezaposlenima.
* Nezaposleni: Korisnici koji traže posao.

Sistem za evidenciju nezaposlenih je deo državnog sistema za zapošljavanje i direktno je povezan sa zavodima za zapošljavanje, ministarstvom rada i socijalne politike, kao i drugim relevantnim državnim institucijama.

### Radna mesta i opis zaduženja

* Administrator sistema:
* Upravljanje korisničkim nalozima.
* Podešavanje sigurnosnih postavki.
* Generisanje globalnih izveštaja.
* Dodavanje, izmena i brisanje nezaposlenih
* Dodavanje poslodavaca
* FIltriranje po iskustvu nezaposlenog
* Operater:
  + Prikaz nezaposlenih
  + Praćenje statusa nezaposlenih.

### Spisak korišćene dokumentacije

* Tehnička dokumentacija za C# i Entity Framework.
* Dokumentacija za Razor Pages.
* Pravilnici i smernice zavoda za zapošljavanje.
* Statistički izveštaji o nezaposlenosti.
* Zakonski okviri i propisi u vezi sa zapošljavanjem.

### Zakonska regulativa, standardi i interni propisi organizacionog sistema

* Zakonska regulativa:
* Zakon o zapošljavanju i osiguranju za slučaj nezaposlenosti.
* Zakon o zaštiti podataka o ličnosti.
* Pravilnici o radu zavoda za zapošljavanje.
* Standardi:
* ISO/IEC 27001: Standard za upravljanje bezbednošću informacija.
* ISO 9001: Standard za upravljanje kvalitetom.

## Specifikacija zahteva korisnika

### Profili korisnika i očekivane funkcionalne mogućnosti softvera

* Administrator sistema:
* Upravljanje korisničkim nalozima.
* Podešavanje sigurnosnih postavki.
* Generisanje globalnih izveštaja.
* Dodavanje, izmena i brisanje podataka o nezaposlenima.
* Dodavanje i ažuriranje podataka o poslodavcima.
* Filtriranje nezaposlenih po iskustvu.
* Operater:
* Prikaz podataka o nezaposlenima.
* Praćenje statusa nezaposlenih.

### Spisak potrebnih ulaznih ekranskih formi

1. Forma za prijavu korisnika: Omogućava korisnicima prijavu u sistem.
2. Forma za unos podataka o nezaposlenima: Unos i ažuriranje ličnih podataka, radne istorije, i statusa.
3. Forma za unos podataka o poslodavcima: Unos podataka o poslodavcima.
4. Forma za pretragu nezaposlenih: Filtriranje i pretraga nezaposlenih na osnovu imena I prezimena.
5. Forma za pretragu nezaposlenih - Iskustvo: Filtriranje i pretraga nezaposlenih na osnovu toga da li imaju iskustvo I na kojoj pozicciju su radili .
6. Forma za pretragu poslodavaca: Filtriranje i pretraga poslodavaca na osnovu naziva.

### Spisak potrebnih tabelarnih prikaza sa kriterijumima filtriranja

* Tabelarni prikaz nezaposleni
  + Prikaz svih nezaposlenih.
  + Prikaz nezaposlenih po imenu i prezimenu.
  + Prikaz nezaposlenih po tome da li su radili u struci.
* Tabelarni prikaz poslodavaca
  + Prikaz svih poslodavaca.
  + Prikaz poslodavaca po nazivu.

### Spisak potrebnih štampi dokumenata (štampi, reporta), statističkih izveštaja za podršku odlučivanju i eksporta podataka

1. Stampa dokumentata o nezaposlenom (njegovi podatci i prethodna radna iskustva).
2. Export podataka u .XML fajl.

### Spisak poslovnih pravila, automatizama i ograničenja koje je potrebno ugaditi u softver

1. JMBG mora biti odgovarajuc datumu rodjenja.
2. Datum zavrsetka rada mora bit veci od datuma pocetka radnog odnosa
3. ID korisnika se random generise
   1. . Mora imati 13 cifara
   2. format mora biti: XXX-XXXXXXXX-XX
   3. prva tri su random kao i centralni broj dok su poslednje dve cifre kontrolni broj
4. PIB mora imati 8 cifara
5. Prilikom unosa nezaposlenog upisuje se njegovo zanimanje (iz diplome) kao i pozicija na kojoj je radio.
   1. Iz .JSON fajla se ucitavaqju podaci o zanimanjima.
   2. Svako zanimanje ima spisak pozicija koje moze obavljati unutar struke.
   3. Na osnovu toga se odlucuje da li je neazaposleni obavljao posao u struci ili ne.

### Potrebne karakteristike korisničkog interfejsa i ostali nefunkcionalni zahtevi

1. Padajuci meni koji se prilagodjava u odnosu na aktivnog korisnika.
   1. Admin – sve
   2. Korisnik – sami prikazi
   3. Ne registrovani – nista
2. Lista nezaposlenih
3. Lista radnih odnosa unutar nezaposlenih
4. Lista poslodavaca
5. Izmena nezaposlenih
6. Dodavanje nezaposlenih kao i dinamicko dodavanje poslodavaca u listu radnih odnosa.
7. Radni odnos predstavlja period rada nezaposlenog u odredjenom preduzecu.

## Elementi projektnog planiranja

### Granice (scope) projekta

1. Prijava korisnika: Korisnici se prijavljuju u sistem koristeći svoje korisničke naloge.
2. Unos podataka o nezaposlenima: Unos i ažuriranje ličnih podataka, radne istorije i statusa nezaposlenih.
3. Unos podataka o poslodavcima: Unos i ažuriranje podataka o poslodavcima.
4. Pretraga nezaposlenih: Filtriranje i pretraga nezaposlenih na osnovu imena i prezimena.
5. Pretraga nezaposlenih - Iskustvo: Filtriranje i pretraga nezaposlenih na osnovu iskustva i pozicija na kojima su radili.
6. Pretraga poslodavaca: Filtriranje i pretraga poslodavaca na osnovu naziva.
7. Tabelarni prikazi:
   1. Prikaz svih nezaposlenih.
   2. Prikaz nezaposlenih po imenu i prezimenu.
   3. Prikaz nezaposlenih po tome da li su radili u struci.
   4. Prikaz svih poslodavaca.
   5. Prikaz poslodavaca po nazivu.
8. Štampa dokumenata: Štampanje dokumenata o nezaposlenom (podaci i prethodna radna iskustva).
9. Export podataka: Export podataka u .XML fajl.
10. Automatizacija poslovnih pravila:
    1. JMBG mora odgovarati datumu rođenja.
    2. Datum završetka rada mora biti veći od datuma početka radnog odnosa.
    3. ID korisnika se random generiše i ima 13 cifara sa specifičnim formatom.
    4. PIB mora imati 8 cifara.
    5. Prilikom unosa nezaposlenog upisuje se njegovo zanimanje (iz diplome) i pozicija na kojoj je radio. Podaci o zanimanjima se učitavaju iz .JSON fajla.
11. Korisnički interfejs:
    1. Padajući meni koji se prilagođava aktivnom korisniku (Admin – sve, Korisnik – samo prikazi, Ne registrovani – ništa).
    2. Lista nezaposlenih.
    3. Lista radnih odnosa unutar nezaposlenih.
    4. Lista poslodavaca.
    5. Izmena nezaposlenih.
    6. Dodavanje nezaposlenih i dinamičko dodavanje poslodavaca u listu radnih odnosa.
    7. Radni odnos predstavlja period rada nezaposlenog u određenom preduzeću.

Funkcije koje su izvan obuhvata projekta:

* 1. Napredne analize tržišta rada: Detaljne analize i predikcije tržišta rada koje uključuju eksterne faktore i prediktivne modele.
  2. Integracija sa eksternim sistemima: Automatizovana integracija sa drugim državnim ili privatnim sistemima za razmenu podataka.
  3. Mobilna aplikacija: Razvoj mobilne verzije aplikacije za pristup sa pametnih telefona.
  4. Podrška za više jezika: Implementacija višejezične podrške za korisnički interfejs.
  5. Detaljna statistička izveštavanja: Napredni izveštaji koji uključuju složene statističke analize i grafičke prikaze.

### Kratak opis tehnologija, programskih jezika i razvojnih alata koji će biti korišćeni u radu

1. **Programski jezik:** C#
2. **Razvojni okvir:** ASP.NET Core sa Razor Pages
3. **ORM alat:** Entity Framework Core
4. **Baza podataka:** Microsoft SQL Server
5. **Frontend:** HTML, CSS, JavaScript
6. **IDE:** Visual Studio
7. **Version Control System:** Git (sa GitHub/GitLab/Bitbucket)
8. **JSON fajlovi:** Korišćeni za učitavanje podataka o zanimanjima.
9. **Alati za dokumentaciju:** Microsoft Word, PDF generatori

**Opis:**

* **C# i ASP.NET Core** će se koristiti za razvoj serverske logike i API-a.
* **Entity Framework Core** će omogućiti rad sa bazom podataka putem ORM pristupa.
* **Razor Pages** će omogućiti razvoj dinamičkih web stranica sa čistim razdvajanjem logike i prikaza.
* **HTML, CSS, JavaScript** će se koristiti za frontend razvoj, omogućavajući interaktivne i responzivne web stranice.
* **Visual Studio** kao glavni razvojni alat pruža robustan skup alata za pisanje, testiranje i debugovanje koda.
* **Git** će se koristiti za kontrolu verzija i kolaboraciju među članovima tima.
* **Microsoft SQL Server** će služiti kao glavna baza podataka za čuvanje svih podataka o korisnicima, nezaposlenima i poslodavcima.
* **JSON** će se koristiti za lako učitavanje i parsiranje podataka o zanimanjima unutar aplikacije.

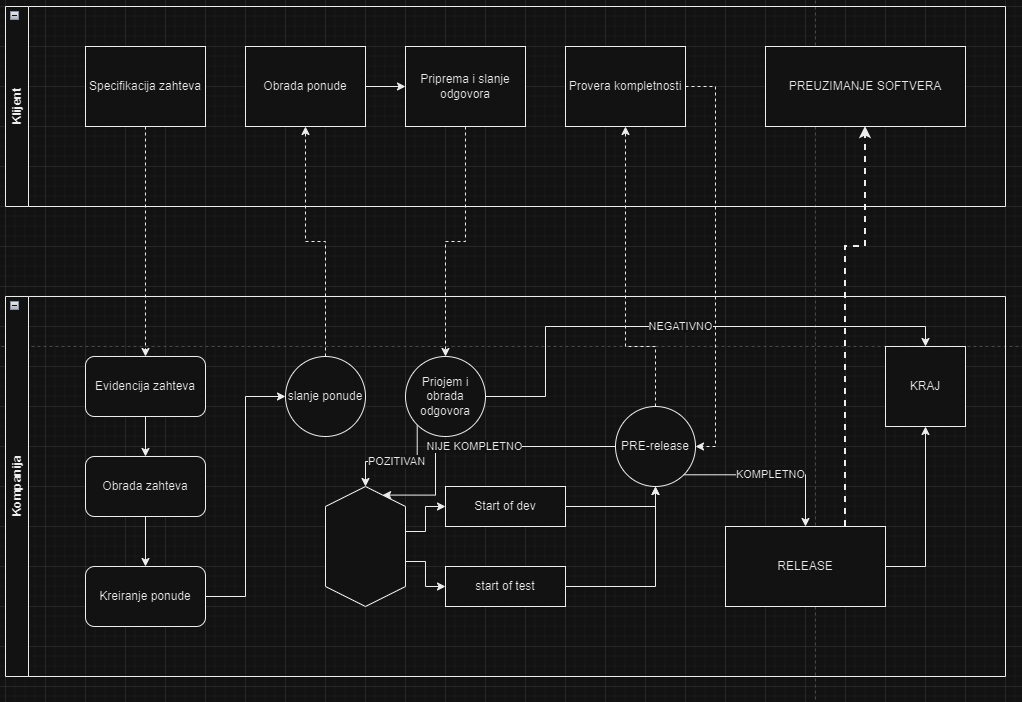
Ove tehnologije i alati obezbediće stabilan i efikasan razvoj sistema za evidenciju nezaposlenih, omogućavajući lako skaliranje i održavanje aplikacije.

## Poslovna sistemska analiza i dizajn softverskog rеšenja

## Model poslovnih procesa

### Algoritamski model poslovnih procesa

Algoritamski model poslovnih procesa predstavlja sekvencijalni prikaz koraka koji se izvršavaju u okviru sistema za evidenciju nezaposlenih. Sledeći koraci predstavljaju tok poslovnih procesa:



*Diagram 1 Model poslovanjan*

### Spisak skladišta podataka

Skladišta podataka predstavljaju entitete u bazi podataka koji služe za čuvanje informacija. U ovom softverskom rešenju, skladišta podataka obuhvataju:

1. **Tabela Nezaposleni:** Čuva informacije o nezaposlenima kao što su ID, JMBG, ime, prezime, datum rođenja, kontakt informacije i adresa.
2. **Tabela Poslodavac:** Čuva informacije o poslodavcima kao što su ID, PIB, naziv, grad i adresa.
3. **Tabela RadniOdnos:** Čuva informacije o radnom odnosu između nezaposlenih i poslodavaca, uključujući datume početka i završetka radnog odnosa.

Ova skladišta podataka su ključni za funkcionisanje sistema za evidenciju nezaposlenih

### Analiza jednog poslovnog dokumenta

Za analizu jednog poslovnog dokumenta u okviru sistema za evidenciju nezaposlenih, posmatraćemo primer formulara za prijavu nezaposlenog.

* **Elementarni podaci:**
  + Ime i prezime nezaposlenog
  + JMBG
  + Datum rođenja
  + Broj telefona
  + Adresa
  + Radna istorija (datumi zaposlenja, pozicije)
  + Informacije o poslodavcima (naziv, adresa)
* **Domeni kao uži skupovi dozvoljenih vrednosti:**
  + JMBG mora biti validan i jedinstven.
  + Datum rođenja mora biti u prošlosti.
  + Broj telefona mora biti u odgovarajućem formatu.
  + Adresa može sadržati samo određene tipove informacija (ulica, broj, grad, poštanski broj).
  + Naziv poslodavca ne sme biti prazan.
  + Datumi zaposlenja moraju biti validni.
* **Sintaksni prikaz strukture dokumenta:**
  + Formular za prijavu nezaposlenog može se prikazati pomoću specijalnih vrsta zagrada, gde svaki elementarni podatak ima svoje polje za unos.

## Dzajn softverskih funkcija

### Tabela preslikavanja primitivnih procesa u softverske funkcije

| **Primitivni proces** | **Softverska funkcija** | **Prioritet** |
| --- | --- | --- |
| Prijava korisnika | Identity/Login | 1 |
| Unos podataka o nezaposlenima | KreirajNezaposlenog | 1 |
| Unos podataka o poslodavcima | KreirajPoslodavca | 1 |
| Pretraga nezaposlenih | DajSvePoimenuIPrezimenu | 1 |
| Pretraga nezaposlenih - Iskustvo | U script delu HTML stranice | 1 |
| Pretraga poslodavaca | GetFromJsonAsync<List<PoslodavacPrikaz>>  ("https://localhost:7240/api/FirmaKontroler") | 1 |
| Validacija podataka (JMBG, PIB, datumi) | ValidateData | 1 |

Tabela 1 Primitivni procesi

### USE CASE dijagram softverskih funkcija

A diagram of a person with blue circles and text

Description automatically generated

Diagram 2 Use case diagram zaposlenog(Administrator)

### Specifikacija slučaja korišćenja za unos i tabelarni prikaz podataka

**Use Case: Unos podataka o nezaposlenima**

A screenshot of a computer

Description automatically generated

Slika 1 Preduslov za kreiranje nezaposlenog

A screenshot of a computer

Description automatically generated

Slika 2Procedura za kreiranje nezaposlenog

A screenshot of a computer

Description automatically generated

Slika 3Procedura za Izmenu poslodavca

A screenshot of a computer

Description automatically generated

Slika 4 Preduslov za izmenu poslodavca

A screenshot of a computer

Description automatically generated

Slika 5 Rezultat izmene poslodavca

## Dizajn modela podataka

### Konceptualni model podataka

**A computer screen shot of a diagram

Description automatically generated**

Diagram 3 Konceptualni klasni model

### Fizički (relacioni) model podataka

A computer screen shot of a diagram

Description automatically generated

Diagram 4 Fizicki model podataka(generisan)

## Implementacija, testiranje i dokumentovanje realizovanog softvera

## 3.1. Korisničko i tehničko uputstvo

### 3.1.1. Korisničko uputstvo

Nakon registracije korisnik plikacije ima mogucnost samo pregledanja postojecih podataka o neazposlenom i poslodavcima, ovo je namenjeno za izlistavanje nezaposlenih I tome moze imati pristup svako.

Za mogucnost dodavanja potrebno je da se dodeli rola administrator, koju obavlja admin sistemam. Sa tom pozicijom moze se menjatoi dodavati i brisati nezaposleni, kao i radni odnosi nezaposlenog. Takodje poslodavac moze isto da se doda I obrise.

1. Registracija

A screenshot of a computer

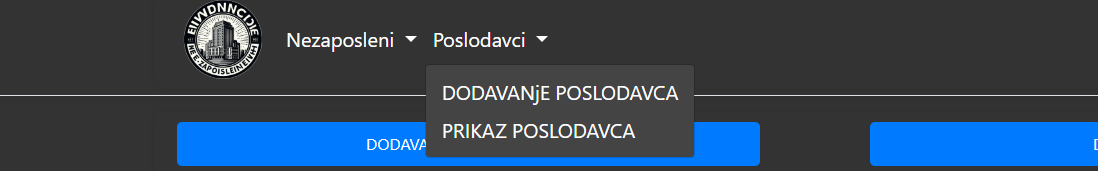
Description automatically generated

1. Login

A screenshot of a login form

Description automatically generated

1. Prikaz





A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Filtriranje

A screenshot of a computer

Description automatically generated



1. Dodavanje Nezaposlenih

A screenshot of a computer

Description automatically generated

## Izmena

A screenshot of a computer

Description automatically generated

1. Brisanje nezaposlenih

Izmeni nezaoslenog -> obrisi

A screenshot of a computer

Description automatically generated



1. Dodavanje firem

A screenshot of a computer

Description automatically generated

### 3.1.2. Tehnicko uputstvo

Ovo tehničko uputstvo pruža korak po korak smernice za instalaciju, konfiguraciju i korišćenje aplikacije za evidenciju nezaposlenih. Aplikacija je razvijena koristeći Entity Framework Core, Dependency Injection, Identity Scaffolding, Power Designer, objektno orijentisano programiranje (OOP), migracije u bazi podataka i Microsoft Visual Studio 2022.

**Instalacija i konfiguracija**

1. **Preuzimanje i instalacija alata**:
   * Preuzmite i instalirajte [Microsoft Visual Studio 2022](https://visualstudio.microsoft.com/sr/vs/).
   * Instalirajte .NET Core SDK sa [Microsoft-ovog zvaničnog sajta](https://dotnet.microsoft.com/download).
2. **Klonsiranje repozitorijuma**:
   * Otvorite terminal ili komandnu liniju.
   * Klonirajte repozitorijum aplikacije koristeći git:

bash

Copy code

git clone <URL vašeg repozitorijuma>

* + Uđite u direktorijum projekta:
  + Pokrenuti .sln fajl

1. **Postavljanje baze podataka**:
   * Konfigurišite konekcioni string za vašu bazu podataka u appsettings.json fajlu.
   * Pokrenite migracije za kreiranje baza podataka: Update-Database

**Pokretanje aplikacije**

1. **Pokretanje aplikacije iz Visual Studia**:
   * Otvorite Visual Studio 2022.
   * Učitajte projektni fajl (.sln) aplikacije.
   * Kliknite na dugme "IIS Express" ili "Play" za pokretanje aplikacije.
2. **Prijava i registracija korisnika**:
   * Aplikacija koristi Identity Scaffolding za autentifikaciju korisnika.
   * Otvorite pregledač i idite na https://localhost:7240/Home.
   * Registrujte novog korisnika ili se prijavite sa postojećim nalogom.

**Korišćenje aplikacije**

1. **Navigacija kroz korisnički interfejs**:
   * **Administrator sistema**:
     + Upravljanje korisničkim nalozima.
     + Podešavanje sigurnosnih postavki.
     + Generisanje globalnih izveštaja.
     + Dodavanje, izmena i brisanje podataka o nezaposlenima.
     + Dodavanje i ažuriranje podataka o poslodavcima.
     + Filtriranje nezaposlenih po iskustvu.
   * **Operater**:
     + Prikaz podataka o nezaposlenima.
     + Praćenje statusa nezaposlenih.
2. **CRUD operacije**:
   * Koristite formu za unos novih podataka o nezaposlenima i poslodavcima.
   * Pretražujte i filtrirajte postojeće podatke koristeći odgovarajuće forme.
   * Ažurirajte ili brišite postojeće zapise.

## 3.2. Izveštaj o testiranju softvera

### 3.2.1. Test slučajevi ispravnog rada korisnika i prikaz rezultata testiranja

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### 3.2.2. Test slučajevi neispravnog rada korisnika i prikaz rezultata testiranja

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## 3.3. Dokumentovanje implementacije softverskog rešenja

### 3.3.1. Modeli i opisna implementacija

#### 3.3.1.1. Dijagram komponenti

A black screen with white text

Description automatically generated

Diagram 5 Diagram komponenti softvera

#### 3.3.1.2. Dijagram klasa

A diagram of a computer

Description automatically generated

Diagram 6 Diagram klasa

#### 3.3.1.3. Dijagram sekvenci za osnovni scenario unosa podataka i tabelarnog prikaza sa filterom

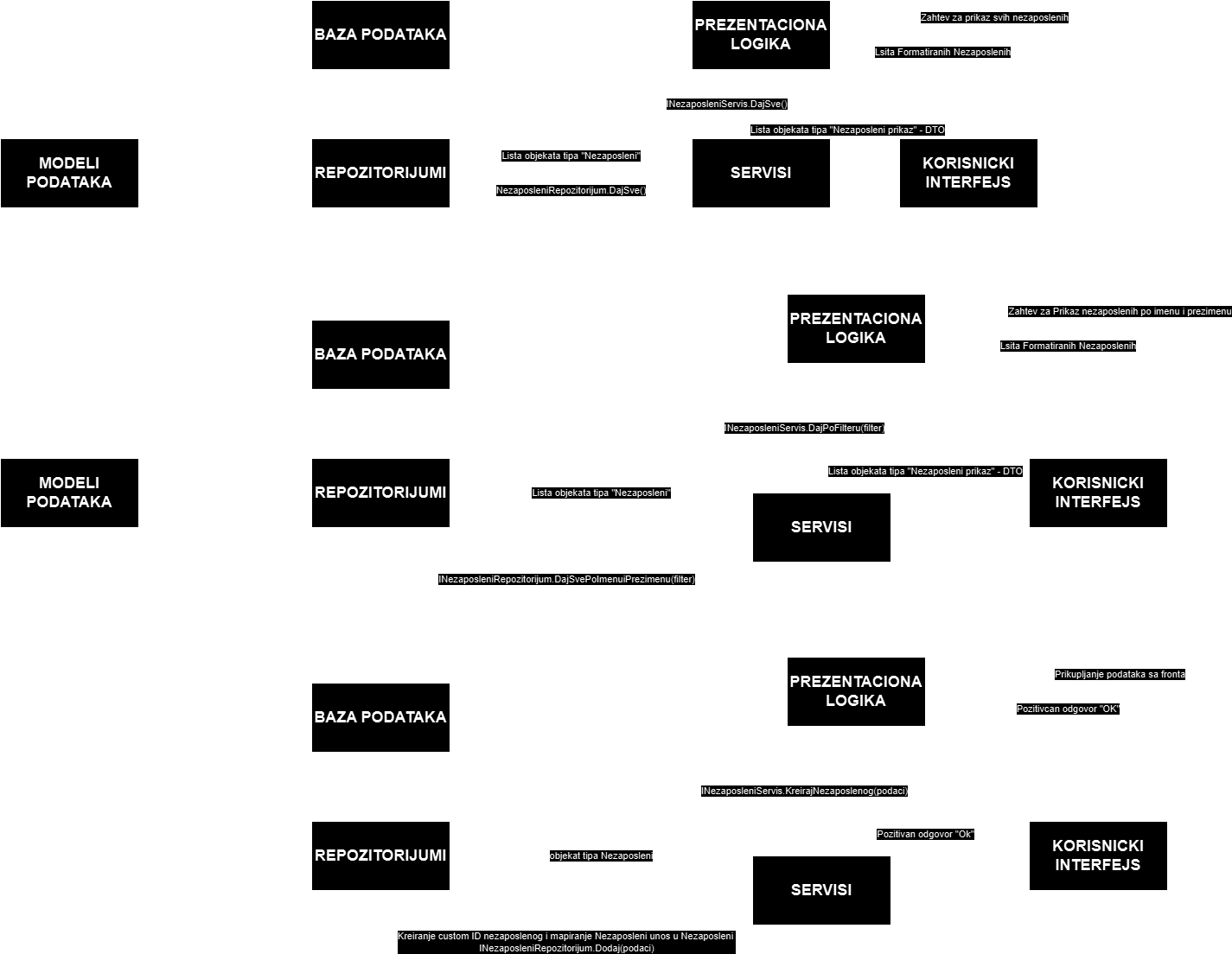


Diagram 7 Onovni procesi aplikacije. Dodavanje Nezapozaposlenog, prikaz svih nezaposlenih i filtriranje

### 3.3.2. Tabelarni prikaz komponenti

| **SLOJ** | **PODSLOJ** | **TEHNOLOŠKA IMPLEMENTACIJA** |
| --- | --- | --- |
| **PREZENTACIONI SLOJ** | Korisnički interfejs | Razor Pages, cshtml, ASP.NET Core, C# |
|  | Prezentaciona logika | Class Library, DLL, C# |
| **SLOJ SERVISA** | API kontroler | Web API, ASP.NET Core, C# |
| **SLOJ POSLOVNE LOGIKE** | Biblioteka klasa poslovne logike | Class Library, DLL, C# |
|  |  | (Za potrebe implementacije svojih metoda poziva servise i klase za rad sa podacima iz baze podataka) |
| **SLOJ ZA RAD SA PODACIMA** | Klase podataka | Class Library, DLL, C# |
|  |  | - Nasleđuju baznu klasu tabela i prosleđuju upit |
|  |  | - Realizuju CRUD operacije kroz svoje metode |
|  |  | - Imaju iste nazive kao tabele iz baze podataka |
|  |  | - Za svaku tabelu u bazi podataka postoje 3 klase (pojedinac (get-set), lista pojedinaca, DB klasa (repository, realizuje CRUD operacije)) |
|  | Tehnološke klase | Class Library, DLL, C# |
|  |  | - Enkapsuliraju konkretnu tehnologiju konekcije i izvršavanja upita nad konkretnim DBMS, ponašaju se kao bazne klase |
| **EKSTERNO** | Relaciona baza podataka | SQL Server, Entity Framework Core |

Tabela 2 Prikaz komponenti

### 3.3.3. SQL skripta

SQL skripta se sastoji samo iz procedure i pogleda.

|  |
| --- |
| CREATE VIEW [dbo].PoslodavacPrikaz AS SELECT PIB, Naziv, Grad, Adresa FROM Poslodavci |
| CREATE PROCEDURE [dbo].AddPoslodavac  @PIB INT, @Naziv NVARCHAR(255), @Grad NVARCHAR(255), @Adresa NVARCHAR(255)  AS BEGIN  INSERT INTO Poslodavci (PIB, Naziv, Grad, Adresa)  VALUES (@PIB, @Naziv, @Grad, @Adresa);  END; |

Tabela 3 SQL kod, Procedura i pogled

Ostatak baze podataka se kreira koriscenjem Entity Frameworka i migracija koricenjem sledecih komandi.

1. Izmeniti konekcioni string u appsettings.json fajlu

2. Otvoriti NPM konzolu

3. Update-database

Nakon tih komandi baza podatak ce biti kreirana sa svim potrebnim tabelama.

Takvo kreiranje baze podataka je postignuto koriscenjem DbContext klase koja se nalazi u projektu “Repozitorijumi”. U toj klasi se definisu primarni kljucevi kaso i koje ce se tabele kreirati. Takodje u kontekstu je mapiran I View.

### 3.3.4. Delovi koda sa objašnjenjima po slojevima

#### 3.3.4.1 Sloj za rad sa podacima

|  |
| --- |
| public class Nezaposleni  {  [Key]  public string ID { get; set; }  public string JMBG { get; set; }  public string Ime { get; set; }  public string Prezime { get; set; }  public DateTime DatumRodjenja { get; set; }  public string BrojTelefona { get; set; }  public string Adresa { get; set; }  public string Zanimanje { get; set; }  public List<RadniOdnos> RadniOdnos { get; set; }  } |
| public class EvidencijaNezaposlenihDBContext : DbContext  {  public EvidencijaNezaposlenihDBContext(DbContextOptions<EvidencijaNezaposlenihDBContext> options) : base(options)  {  }  public DbSet<Nezaposleni> Nezaposleni { get; set; }  public DbSet<Poslodavac> Poslodavci { get; set; }  public DbSet<RadniOdnos> RadniOdnosi { get; set; }  public DbSet<PoslodavacPrikaz> PoslodavacPrikaz { get; set; }  protected override void OnModelCreating(ModelBuilder modelBuilder)  {  modelBuilder.Entity<PoslodavacPrikaz>().HasNoKey().ToView("PoslodavacPrikaz");  modelBuilder.Entity<RadniOdnos>().HasKey(x => new { x.ID, x.NezaposleniID });  }  } |
| public interface IRepozitorijum<T> where T : class  {  Task<T?> Obrisi(object PK);  Task<T?> DajSvePoPrimarnomKljucu(object PK);  Task<List<T>> DajSvePoFilteru(object filter);  Task<IEnumerable<T>> DajSve();  T Dodaj(T obj);  T? Izmeni(T obj);  void Snimi();  }  public interface INezaposleniRepozitorijum : IRepozitorijum<Nezaposleni>  {  Task<Nezaposleni?> DajSvePoJMBG(object filter);  } |
| using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  public class NezaposleniRepozitorujum : INezaposleniRepozitorijum  {  private readonly EvidencijaNezaposlenihDBContext \_context;  public NezaposleniRepozitorujum(EvidencijaNezaposlenihDBContext context)  {  \_context = context ?? throw new ArgumentNullException(nameof(context));  }  public async Task<IEnumerable<Nezaposleni>?> DajSve()  {  return await \_context.Nezaposleni  .Include(x => x.RadniOdnos)  .ThenInclude(x => x.Poslodavac)  .ToListAsync();  }  .  .  .  } |

Tabela 4 Prikaz Dela koda koji radi sa podacima, Model, DbContext, Interfejs i repozitorijum

#### 3.3.4.2. Sloj servisa ili DTO

|  |
| --- |
| namespace EvidencijaNezaposlenih.ModeliPodataka.DTO  {  public class NezaposleniUnos  {  public required string Ime { get; set; }  public required string Prezime { get; set; }  public DateTime DatumRodjenja { get; set; }  public required string BrojTelefona { get; set; }  public required string Adresa { get; set; }  public required string JMBG { get; set; }  public string Zanimanje { get; set; }  public required List<RadniOdnosPrikaz> RadniOdnosPrikaz { get; set; }  }  } |
| public interface INezaposleniServis  {  Task<IEnumerable<NezaposleniPrikaz>> DajSve();  Task<IEnumerable<NezaposleniPrikaz>> DajSvePoimenuIPrezimenu(object filter);  Task<NezaposleniPrikaz> DajSvePoJMBGU(object JMBG);  Task<NezaposleniPrikaz> DajSvePoID(object PK);  Task KreirajNezaposlenog(NezaposleniUnos obj);  Task<object> Obrisi(object PK);  Task Azuriraj(NezaposleniUnos obj);  } |
| public class NenzaposleniServis : INezaposleniServis  {  private readonly INezaposleniRepozitorijum \_nezaposleniRepozitorijum;  private readonly IPoslodavacRepozitorijum \_poslodavacRepozitorijum;  private readonly IRadniOdnosRepozitorijum \_radniOdnosRepozitorijum;  private readonly IPoslovnaLogika \_poslovnaLogika;  private readonly IRadUStruci \_radUStruci;  public NenzaposleniServis (INezaposleniRepozitorijum nezaposleniRepozitorijum, IPoslodavacRepozitorijum poslodavacRepozitorijum, IRadniOdnosRepozitorijum radniOdnosRepozitorijum, IPoslovnaLogika poslovnaLogika, IRadUStruci radUStruci)  {  \_nezaposleniRepozitorijum = nezaposleniRepozitorijum;  \_poslodavacRepozitorijum = poslodavacRepozitorijum;  \_radniOdnosRepozitorijum = radniOdnosRepozitorijum;  \_poslovnaLogika = poslovnaLogika;  \_radUStruci = radUStruci;  }  public async Task Azuriraj(NezaposleniUnos obj)  {  var data = await \_nezaposleniRepozitorijum.DajSvePoJMBG(obj.JMBG);  if (data == null)  throw new ArgumentException("Pogresan ID");  obj = \_radUStruci.DaLiJeRedioUStruci(obj);  foreach (var item in obj.RadniOdnosPrikaz)  {  var radniOdnos = await \_poslodavacRepozitorijum.PronadjiPoNazivu(FormatirajFirmu(item.NazivFirme));  RadniOdnos radniOdnosi = new()  {  DatumPocetka = item.DatumPocetka,  DatumZavrsetka = item.DatumZavrsetka,  Pozicija = item.Pozicija,  Struka = item.Struka,  ID = radniOdnos.ID,  NezaposleniID = data.ID,  };  \_radniOdnosRepozitorijum.Izmeni(radniOdnosi);  \_radniOdnosRepozitorijum.Snimi();  }  .  .  .  } |

Tabela 5 Prikaz dela koda koji ima ulogu servisa i koji koristi DTO modele podataka

3.3.4.3. Sloj poslovne logike

Opis:

Poslovna logika se bazira na proveri da li je nezaposleni radio u struci.

Sto znaci da svaki nezaposleni mora imati svoje zanimanje pri unosu u bazu podataka. Takodje tabela “RadniOdnos” posecuje polje “string pozicija” kao i “bool struka”. Pozicija se unosi priliklom dodavanja radnog odnosa za odredjenog nezaposlenog, dok se polje struka samo popunjava u odnosu na to da li je dato zanimanje opisano strukom zaposlenog. U sifarniku ispod vidimo da svako zanimanje ima vise pozicija koje moze obavljati iz slicnog sektora. Kao na primer Menadzer, on ne mora striktno biti samo menadzer necega (npr. Licni menadzer nekog sportiste), on moze biti i Scrum master ili Product owner. Jer njegova struka obuhvata sve to.

|  |
| --- |
| public interface IRadUStruci  {  NezaposleniUnos DaLiJeRedioUStruci(NezaposleniUnos obj);  } |
| public class RadUStruci : IRadUStruci  {  public NezaposleniUnos DaLiJeRedioUStruci(NezaposleniUnos obj)  {  string json = File.ReadAllText("zanimanja.json");  var data = JsonConvert.DeserializeObject<Dictionary<string, List<Zanimanje>>>(json);  // Pozicija koju želimo da pretražujemo  string trazenaPozicija = "";  List<RadniOdnosPrikaz> odnosi = new();  foreach (var kvp in obj.RadniOdnosPrikaz)  {  trazenaPozicija = kvp.Pozicija.ToString().ToUpper();  List<string> zanimanjaKojaSadrzePoziciju = new List<string>();  foreach (var zanimanje in data["zanimanja"])  {  if (zanimanje.Pozicije.Contains(trazenaPozicija))  {  zanimanjaKojaSadrzePoziciju.Add(zanimanje.Naziv);  }  }  foreach (var poz in zanimanjaKojaSadrzePoziciju)  {  if (poz == obj.Zanimanje.ToUpper())  {  kvp.Struka = true;  }  }  odnosi.Add(kvp);  };  obj.RadniOdnosPrikaz = null;  obj.RadniOdnosPrikaz = odnosi;  return obj;  }  } |
| {  "zanimanja": [  {  "naziv": "PROGRAMER",  "pozicije": [  "JUNIOR DEVELOPER",  "MEDIOR DEVELOPER",  "SENIOR DEVELOPER",  "BACKEND DEVELOPER",  "FRONTEND DEVELOPER",  "FULL-STACK DEVELOPER",  "EMBEDDED DEVELOPER"  ]  },  {  "naziv": "MENADZER",  "pozicije": [  "SCRUM MASTER",  "PROJECT OWNER",  "PROJECT MANAGER",  "PRODUCT MANAGER",  "OPERATIONS MANAGER"  ]  },  {  "naziv": "DIZAJNER",  "pozicije": [  "GRAPHIC DESIGNER",  "UI/UX DESIGNER",  "WEB DESIGNER"  ]  },  {  "naziv": "Marketar",  "pozicije": [  "DIGITAL MARKETING SPECIALIST",  "SEO SPECIALIST",  "CONTENT MARKETING SPECIALIST"  ]  }  ]  } |

Tabela 6 Prikaz dela koda poslovne logike

#### 3.3.4.4. Kod za osnovnu validaciju podataka sloja prezentacione logike

Pored svakog input polja postoji required sto znaci da to polje mora biti popunjeno. To je osnovna validacija potpunosti, dok se ostale metode za validaciju nalaze u sloju poslovne logike. Tu se proverava na primer da li je JBMG kompatibilan sa datumom rodjenja, da li je datum pocetka radnog odnosa manji od datuma zavrsetka, …

|  |
| --- |
| <form id="data-entry-form" method="post">  <div class="form-group">  <label for="name">Ime:</label>  <input type="text" id="name" name="name" required>  </div>  <div class="form-group">  <label for="surname">Prezime:</label>  <input type="text" id="surname" name="surname" required>  </div>  <div class="form-group">  <label for="dateOfBirth">Datum Rodjenja:</label>  <input type="date" id="dateOfBirth" name="dateOfBirth" required>  </div>  <div class="form-group">  <label for="jmbg">JMBG:</label>  <input type="text" id="jmbg" name="jmbg" required>  </div>  <div class="form-group">  <label for="phoneNumber">Broj telefona:</label>  <input type="text" id="phoneNumber" name="phoneNumber" required>  </div>  <div class="form-group">  <label for="adresa">Adresa:</label>  <input type="text" id="adresa" name="adresa" required>  </div>  <div class="form-group">  <label for="adresa">Zanimanje:</label>  <input type="text" id="zanimanje" name="zanimanje" required>  </div> |

Tabela 7 Frontend validacija podataka

Jednostavnim prosledjivanjem kompletnog DTO modela izdvajaju se potrebni podaci i validiraju, takodje sve metode su bool sto znaci da se vraca true ili false kao povratna vrednost, i na osnovu toga se vrsi validacija tih podataka.

|  |
| --- |
| public bool ValidirajJMBG(NezaposleniUnos obj)  {  var JMBG = obj.JMBG;  if (JMBG.Length != 13)  {  return false;  }  var datumRodjenja = obj.DatumRodjenja;  var dan = datumRodjenja.Day.ToString("D2");  var mesec = datumRodjenja.Month.ToString("D2");  var godina = datumRodjenja.Year.ToString().Substring(1,3);  var jmbgDan = JMBG.Substring(0, 2);  var jmbgMesec = JMBG.Substring(2, 2);  var jmbgGodina = JMBG.Substring(4, 3);  return jmbgDan == dan && jmbgMesec == mesec && jmbgGodina == godina;  }  public bool ValidirajTrajanjeRadnogOdnosa(NezaposleniUnos obj)  {  foreach(var item in obj.RadniOdnosPrikaz)  {  if (item.DatumZavrsetka < item.DatumPocetka) return false;  }  return true;  } |

Tabela 8 Backend validacija podataka

#### 3.3.4.5. Korisnički interfejs

A blue rectangles on a black background

Description automatically generated

Slika 6 Admin pocetna stranica

A screenshot of a computer

Description automatically generated

Slika 7 Prikaz svih nezaposlenih

A screenshot of a computer

Description automatically generated

Slika 8 Prikaz nezaposlenih po filteru (ime ili prezime ili oba)

A screenshot of a computer

Description automatically generated

Slika 9 Prikaz filtriranja po tome da li neko raadi u struci

A screenshot of a computer

Description automatically generated

Slika 10 Dodavanje nezaposlenih

A screenshot of a computer

Description automatically generated

Slika 11 Dodavanje nezaposlenih

Pritiskom na dodaj iskustvo lista se dinamiski siri, isto tako se I dinamicki smanjuje pritiskom na kanticu za brisanje.

A screenshot of a computer

Description automatically generated

Slika 12 Dodavanje nezaposlenih dinamicki prikaz i combobox

### 3.3.5. Delovi koda i opis implementacije podrške personalizaciji aplikacije i bezbednosti korišćenja

**Prijava korisnika**

Prijava korisnika je realizovana pomoću ASP.NET Core Identity scaffolding sistema. ASP.NET Core Identity je moćan sistem za upravljanje korisnicima koji omogućava funkcionalnosti kao što su registracija, prijava, resetovanje lozinke i druge operacije vezane za korisničke naloge.

Identity koristi kolačiće za upravljanje autentifikacijom i autorizacijom korisnika. Kada se korisnik prijavi, kreira se sesija koja traje sve dok korisnik ne odluči da se odjavi ili dok sesija ne istekne.

**Rad sa sesijama**

Sesije korisnika se održavaju pomoću kolačića koji se koriste za autentifikaciju. Kada se korisnik prijavi, ASP.NET Core Identity generiše autentifikacioni kolačić koji sadrži potrebne informacije o korisniku. Ove informacije se koriste tokom trajanja sesije za proveru identiteta korisnika pri svakom zahtevu ka serveru.

**Razmena podataka putem POST, PUT, DELETE i GET metoda**

API kontroler koristi POST i GET metode za upravljanje podacima o poslodavcima. Ove metode omogućavaju dodavanje, ažuriranje, pretragu i brisanje podataka o poslodavcima. Primeri POST i GET metoda prikazani su u nastavku.

| **Metoda** | **Opis** | **URL** | **Parametri** | **Povratna vrednost** |
| --- | --- | --- | --- | --- |
| POST | Kreiranje novog poslodavca | /api/FirmaKontroler | PoslodavacUnos (body) | Status kreiranja i URL novog poslodavca |
| GET | Dobijanje poslodavca po PIB-u | /api/FirmaKontroler/{pib} | pib (path parametar) | Podaci o poslodavcu sa datim PIB-om |
| DELETE | Brisanje poslodavca po PIB-u | /api/FirmaKontroler/{pib} | pib (path parametar) | noContent |
| PUT | Izmena poslodavca | /api/FirmaKontroler/{pib} | PoslodavacUnos (body) | noContent |

Tabela 9 Spisak osnovnih metoda koriscenih u radu

Primeri:

|  |
| --- |
| GET:  [HttpGet("{pib}")]  public async Task<ActionResult<PoslodavacPrikaz>> GetByPIB(string pib)  {  var result = await \_poslodavacServis.DajSvePoPIB(pib);  if (result == null)  {  return NotFound();  }  return Ok(result);  }: |
| DELETE:  [HttpDelete("{pib}")]  public async Task<ActionResult> Delete(string pib)  {  await \_poslodavacServis.Obrisi(pib);  return NoContent();  } |
| POST:  [HttpPost]  public async Task<ActionResult> Create(PoslodavacUnos poslodavacUnos)  {  await \_poslodavacServis.KreirajPoslodavca(poslodavacUnos);  return CreatedAtAction(nameof(GetByPIB), new { pib = poslodavacUnos.PIB }, poslodavacUnos);  } |
| PUT:  [HttpPut]  public async Task<ActionResult> Update(NezaposleniIzmena nezaposleniIzmena)  {  await \_poslodavacServis.Azuriraj(nezaposleniIzmena);  return NoContent();  } |

Tabela 10 Primer koriscenih metoda u radu

### 3.3.6. Delovi koda i opis implementacije osnovnih principa OOP

U radu je korisceno nasledjivanje nasledjivanjem interfejsa klasa, kao i implementacije interfejsa.

|  |
| --- |
| public class PoslovnaLogika : IPoslovnaLogika |

Tabela 11 Primer nasledjivanja

Sto se polimorfizma tice postoji zajednicki interfejs koji se koristi u repozitorijumima i on ima ulogu da se koriscenjem njega sve metode naslede I u druge klase repozitorijuma.

Takodje taj interfejs se nasledjuje u Interfejse svakog repozitorijuma.

Svaki repozitorijum ima svoj interfejs koji nasledjuje, a taj interfejs nasleduje Irepozitorijum interfejs koji sadrzi sve osnovne operacije.

|  |
| --- |
| public interface IRepozitorijum<T> where T : class  {  Task<T?> Obrisi(object PK);  Task<T?> DajSvePoPrimarnomKljucu(object PK);  Task<List<T>> DajSvePoFilteru(object filter);  Task<IEnumerable<T>> DajSve();  T Dodaj(T obj);  T? Izmeni(T obj);  void Snimi();  } |
| public interface IPoslodavacRepozitorijum : IRepozitorijum<Poslodavac>  {  Task<Poslodavac> PronadjiPoNazivu(object filter);  Task<List<Poslodavac>> DajSvePoFilteru(object naziv);  Task<IEnumerable<PoslodavacPrikaz>> DajSvePogled(object pogled);  Task DodajStorred(PoslodavacUnos obj);  } |
| public class PoslodavacRepozitorijum : IPoslodavacRepozitorijum |

Tabela 12 Primer polimorfizma

### 3.3.7. Delovi koda i opis implementacije SOLID principa

Svaka klasa ima svoju odgovornost, tacnije zasluzena je za samo jedan model, sto ne znaci da ne moze da komunicira I sa drugim modelima, ali njena osnovna namena je da osnovne operacije vrsi samo sa maticnim modelom. Sve polazi od repozitorijuma gde se za svaki repozitorijum pravi interfejs koji u sebi ima klasu za koju je namenjen pa tako imam situaciju koja je prikazana u sledecoj tabeli:

|  |
| --- |
| public interface IRadniOdnosRepozitorijum : IRepozitorijum<RadniOdnos> |

Tabela 13 Primer klase koja ima single responsability

Radni odnos repozitorijum je namenjen samo za klasu radni odnos i samo ona se koristi pa je stoga ona samo I nasledjena.

Dependency injection se koristi u svim delovima Softvera. Svaka klasa se instancira na pocetku u konstruktoru, opet preko interfejsa:

|  |
| --- |
| public class NenzaposleniServis : INezaposleniServis  {  private readonly INezaposleniRepozitorijum \_nezaposleniRepozitorijum;  private readonly IPoslodavacRepozitorijum \_poslodavacRepozitorijum;  private readonly IRadniOdnosRepozitorijum \_radniOdnosRepozitorijum;  private readonly IPoslovnaLogika \_poslovnaLogika;  private readonly IRadUStruci \_radUStruci;  **public NenzaposleniServis (INezaposleniRepozitorijum nezaposleniRepozitorijum, IPoslodavacRepozitorijum poslodavacRepozitorijum, IRadniOdnosRepozitorijum radniOdnosRepozitorijum, IPoslovnaLogika poslovnaLogika, IRadUStruci radUStruci)**  **{**  **\_nezaposleniRepozitorijum = nezaposleniRepozitorijum;**  **\_poslodavacRepozitorijum = poslodavacRepozitorijum;**  **\_radniOdnosRepozitorijum = radniOdnosRepozitorijum;**  **\_poslovnaLogika = poslovnaLogika;**  **\_radUStruci = radUStruci;**  **}** |

Tabela 14 Primer dependency injection-a

Daljje u toj klasi se koriste instance klasa kao na prmer ovde :

|  |
| --- |
| await \_nezaposleniRepozitorijum.DajSvePoJMBG(obj.JMBG); |

Tabela 15 Poziv metode iz instancirane klase

Sve klase koje se koriste i sa kojima se vrsi dependency injection se definisu u Program.cs fajlu koji je i osnova za pokretanje aplikacije. Takodje i kontekst je definisan u ovom fajlu da bi se znalo koja se baza podataka koristi.

|  |
| --- |
| builder.Services.AddDbContext<EvidencijaNezaposlenihDBContext>(options =>  options.UseSqlServer(builder.Configuration.GetConnectionString("EvidencijaNezaposlenihDBContext"))  .UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking));  builder.Services.AddScoped<INezaposleniRepozitorijum, NezaposleniRepozitorujum>();  builder.Services.AddScoped<IPoslodavacRepozitorijum, PoslodavacRepozitorijum>();  builder.Services.AddScoped<INezaposleniServis, NenzaposleniServis>();  builder.Services.AddScoped<IPoslodavacServis, PoslodavacServis>();  builder.Services.AddScoped<IRadniOdnosServis, RadniOdnosServis>();  builder.Services.AddScoped<IRadniOdnosRepozitorijum, RadniOdnosRepozitorijum>();  builder.Services.AddScoped<IPoslovnaLogika, PoslovnaLogika>();  builder.Services.AddScoped<IRadUStruci, RadUStruci>(); |

Tabela 16 Konfiguracija za dependency injections u Program.cs fajlu

### 3.3.8. Delopvi koda, objašnjenja realizacije clean code pristupa

Clean code pristup u razvoju softvera podrazumeva pisanje koda koji je čitljiv, razumljiv i lako održiv.

1. **Smislena imena**

**Klase i metode:**

Imena klasa treba da budu imenice koje jasno opisuju njihov sadržaj.

Nezaposleni, Poslodavac, RadniOdnos.

Imena metoda treba da budu glagoli koji jasno opisuju radnju koju metoda vrši. KreirajPoslodavca, AzurirajNezaposlenog.

**Promenljive:**

Promenljive treba da imaju smislena imena koja jasno opisuju njihovu svrhu. datumRodjenja, brojTelefona, adresa.

2. **Kratke i jasne metode**

Metode treba da budu kratke i da vrše samo jednu radnju. Ako metoda postane predugačka ili kompleksna, treba je podeliti na manje metode.

|  |
| --- |
| public async Task<IEnumerable<Nezaposleni>?> DajSve()  {  return await \_context.Nezaposleni  .Include(x => x.RadniOdnos)  .ThenInclude(x => x.Poslodavac)  .ToListAsync();  } |

Tabela 17 Primer Jasne metode

3. **Komentari**

Komentari treba da se koriste štedljivo i samo kada je zaista potrebno. Kod treba da bude samodokumentovan kroz smislena imena klasa, metoda i promenljivih.

|  |
| --- |
| public NezaposleniUnos DaLiJeRedioUStruci(NezaposleniUnos obj)  {  string json = File.ReadAllText("zanimanja.json");  //Ucitavanje podataka iz JSON fajla  var data = JsonConvert.DeserializeObject<Dictionary<string, List<Zanimanje>>>(json);  // Pozicija koju želimo da pretražujemo  string trazenaPozicija = "";  List<RadniOdnosPrikaz> odnosi = new();  //Prolazak kroz listu radnih odnosa  foreach (var kvp in obj.RadniOdnosPrikaz)  {  trazenaPozicija = kvp.Pozicija.ToString().ToUpper();  List<string> zanimanjaKojaSadrzePoziciju = new List<string>();  foreach (var zanimanje in data["zanimanja"])  {  if (zanimanje.Pozicije.Contains(trazenaPozicija))  {  zanimanjaKojaSadrzePoziciju.Add(zanimanje.Naziv);  }  }  foreach (var poz in zanimanjaKojaSadrzePoziciju)  {  //Radi lakse pretrage slova se pretvaraju u velika  if (poz == obj.Zanimanje.ToUpper())  {  kvp.Struka = true;  }  }  odnosi.Add(kvp);  };  obj.RadniOdnosPrikaz = null;  obj.RadniOdnosPrikaz = odnosi;  //Vracanje izmenjenog objekta  return obj;  } |

Tabela 18 Pravilno koriscenje komentara

4. **Konzistentno formatiranje**

Korišćenje konzistentnog formatiranja koda olakšava čitanje i razumevanje koda. Ovo uključuje tabove, razmake i stil zagrada.

|  |
| --- |
| public class Nezaposleni  {  [Key]  public string ID { get; set; }  public string JMBG { get; set; }  public string Ime { get; set; }  public string Prezime { get; set; }  public DateTime DatumRodjenja { get; set; }  public string BrojTelefona { get; set; }  public string Adresa { get; set; }  public List<RadniOdnos> RadniOdnos { get; set; }  } |

Tabela 19 Formatiranje koda

**5.Jednostavnost**

Kod treba da bude što jednostavniji.

|  |
| --- |
| public bool ValidirajTrajanjeRadnogOdnosa(NezaposleniUnos obj)  {  foreach(var item in obj.RadniOdnosPrikaz)  {  if (item.DatumZavrsetka < item.DatumPocetka) return false;  }  return true;  } |

Tabela 20 Primer upotrebe jednostavnog koda, bez nepotrebnoh funkcija

Ovim pristupom, nas projekat će imati kvalitetan i održiv kod koji je lak za razumevanje i proširenje.

Takodje u buducnosti lako se mogu izvrsiti izmene i ispraviti greske koje su neizbezne u razvoju softvera. Takodje kod se vrlo laok moze i prepravljati za neke druge namene.

## Literatura

| **Tehnologija/Alat** | **Link** |
| --- | --- |
| Entity Framework Core | [Entity Framework documentation hub | Microsoft Learn](https://learn.microsoft.com/en-us/ef/) |
| Dependency Injection | [Dependency injection - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection) |
| Identity Scaffolding | [ASP.NET Core Identity](https://docs.microsoft.com/sr-latn-rs/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=netcore-cli) |
| Power Designer | [PowerDesigner 16.6 - Documentation](https://www.powerdesigner.biz/EN/powerdesigner-resources/powerdesigner-documentation.html) |
| OOP (Objektno Orijentisano Programiranje) | [Object-Oriented Programming - C# | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop) |
| Migracije u bazi podataka | [Entity Framework Core Migracije](https://docs.microsoft.com/sr-latn-rs/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli) |
| MS Visual Studio 2022 | [Visual Studio 2022](https://visualstudio.microsoft.com/sr/vs/) |

Tabela 21 Spisak koriscenih literatura