Izrada web aplikacije za online vrednovanje koristeći .NET Core i Blazor

Jorgić, Dina

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Sveučilišni diplomski studij informatike

Dina Jorgić

Izrada web aplikacije za *online* vrednovanje koristeći .NET Core i Blazor

Diplomski rad

Mentorica: izv. prof. dr. sc. Martina Holenko Dlab

Rijeka, 23. 2. 2024.

Sveučilište u Rijeci
FIDIT Fakultet informatike i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 27. travnja 2023.

Zadatak za diplomski rad

Pristupnik: Dina Jorgić

Naziv diplomskog rada: Izrada web aplikacije za *online* vrednovanje koristeći .NET Core i Blazor

Naziv diplomskog rada na eng. jeziku: Creating a web application for online assessment using .NET Core and Blazor

Sadržaj zadatka:

Zadatak diplomskog rada je opisati .NET Core i Blazor tehnologiju za izradu web aplikacija. Kao praktični dio rada potrebno je napraviti aplikaciju koja će omogućiti nastavnicima *online* vrednovanje. Aplikacija treba omogućavati izradu zadataka za STEM područje s naglaskom na matematičke zadatke koji se sastoje od niza podzadataka.

Mentorica:

Izv. prof. dr. sc. Martina Holenko Dlab

Voditeljica za diplomske radove:

Prof. dr. sc. Ana Meštrović

Zadatak preuzet: 27. travnja 2023.

(potpis pristupnika)

Jorgić Dina

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: **+385(0)51 584 700**
E-mail: **ured@inf.uniri.hr**
OIB: **64218323816**
IBAN: **HR1524020061400006966**

uniri

Sažetak

U uvodnom dijelu ovog diplomskog rada objašnjena je tehnologija .NET Core i okvir Blazor za izradu web aplikacija. Dana je definicija .NET Core tehnologije, ukratko je opisana povijest razvoja i najznačajnije karakteristike. Nadalje, objašnjen je razvojni okvir Blazor i uspoređen s Razorom te korišteni WebAssembly. Glavni dio rada sadrži opis izrađene aplikacije za *online* vrednovanje. Sučelje aplikacije se sastoji od dva dijela, za nastavnike i studente. Dio za nastavnike je složeniji i opsežniji te omogućuje izradu strukture kolegije, izradu baze matematičkih zadataka te definiranje provjera znanja. Dio za studente omogućuje studentima pristupanje provjerama znanja koje sadrže matematičke zadatke.

Ključne riječi: .NET Core, Blazor, WebAssembly, C#, ASP.NET, Visual Studio, ELARS aplikacija

Sadržaj

1. Uvod

Svakim danom dolazi do sve veće potrebe za bržim dobivanjem informacija i međusobnim povezivanjem korisnika. Kako rastu potrebe korisnika tako istovremeno raste potreba za bržom, jednostavnijom i visokoučinkovitom izradom web aplikacija. Tehnologija vrlo brzo napreduje i postaje jednostavnija za primjenu i korištenje.

Web aplikacija je program koji se pokreće u internetskom pregledniku, pohranjen je na nekom udaljenom poslužitelju, te omogućava interakciju između klijenta i poslužitelja.

Web tehnologija se u posljednih 30-ak godina razvila od statičnih HTML stranica do vrlo interaktivnih aplikacija koje se mogu ažurirati u stvarnom vremenu. Na početku su se web aplikacije mogle praviti samo za određenu platformu, Windows, macOS ili Linux. Različite tehnologije su bile za izradu aplikacije koja se može pokretati samo na jednoj platforme. Kako je sve više korisnika krenulo koristiti različite platforme, bilo je potrebe za razvojem tehnologije koja omogućava izradu jedne aplikacije koja se može pokretati na različitim platformama. .NET upravo to omogućava i pruža mogućnost razvoja na različitim programskim jezicima, koristeći brojne biblioteke i alate. Međutim, tradicionalne web tehnologije kao što je JavaScript duže vrijeme su služile kao okosnice web razvoja te dolaze s određenim ograničenjima. Iz tog razloga uz napredovanje tehnologije pojavljuje se ASP.NET Core u kombinaciji s .NET sustavom i Blazor-ovim značajkama u kojima izvedba i snaga poslužitelja pruža mogućnost izradu novijih web aplikacija.

Motivacija za odabir teme diplomskog rada je nastavak teme završnog rada 'Hibridne web aplikacije-pregled i usporedba suvremenih tehnologija', gdje sam razradila temu hibridnih web aplikacija, odnosno aplikacije koje se mogu pokretati na različitim platformama. Vrlo je zanimljivo koliko je .NET Core tehnologija napredna i pojednostavila je izradu i implementaciju aplikacije.

Navedene tehnologije su u praktičnom dijelu rada primijenjene u izradi web aplikacije za *online* vrednovanje studenata ELARS (E-Learning Activities Recommender System) koja je tema ovog diplomskog rada. U nastavku rada, opisane su korištene tehnologije i postupak implementacije.

2. Web aplikacije

Sve više korisnika i tvrtki svakim danom koriste internet kao najbrži i najbolji način komunikacije, koji im omogućava prikaz i razmjenu informacija s ciljem sigurne transakcije. Web aplikacija je računalni program za upravljanje, pohranjivanje i dohvaćanje informacija.

Kako bi se svaka web aplikacija mogla izvršavati, postoje elementi koji to omogućavaju. Strana klijenta koristi JavaScript i HTML skripte za pokretanje aplikacije i predstavljanje informacija korisnicima. Dok strana poslužitelja koristi kombinaciju skripti PHP i ASP.NET koji su potrebni za obradu zahtjeva klijenta i podataka. Na taj način omogućena je interakcija korisnika i web aplikacije, te dijeljenje informacija, stvaranje dokumenata, suradnju na projektima, ispunjavanje obrazaca, uređivanje videa i slika, proračunske tablice i još mogo toga [1].

Web aplikacija radi na način da korisnik kreira zahtjev putem korisničkog sučelja prema web poslužitelju. Web poslužitelj taj zahtjev šalje poslužitelju web aplikacija, koji izvršava dobiveni zadatak, generira rezultate i šalje natrag odgovor web poslužitelju. Web poslužitelj zatim prikazuje tražene informacije klijentu [1].

Web aplikacija je dobar odabir softvera, jer omogućava kompatibilnost s više platformi. Web aplikaciju može koristiti više korisnika istovremeno, ne zauzima memoriju tijekom rada te je dostupna s bilo kojeg uređaja. Svi korisnici pristupaju istoj verziji, tako da nema problema s kompatibilnošću. Potrebno je manja podrška i održavanje, tako da ima minimalne softverske troškove.

3. .NET Core tehnologija

ASP.NET Core je besplatan i moderan okvir otvorenog kôda koji je kompatibilan sa širokim rasponom operativnih sustava. Razvijen je preko .NET okvira koji pruža značajke za izgradnju modernih, fleksibilnih, skalabilnih i robusnih aplikacija. Radi na različitim sustavima što je korisno za programere koji rade u različitim okruženjima. Podržava mnoge jezike kao na primjer C#, F# i Visual Basic, što ga čini odličnim za Windows, web, mobilne, IoT, Cloud i AI aplikacije [1]. .NET Core platforma je modularna i agilna zato što je dostavljena kao skup NuGet[1] paketa, odnosno svaka komponenta je set manjih NuGet paketa. Modularni dizajn .NET Core-a omogućava da svaka aplikacija razvije ono što treba te se na taj način smanjuje veličina aplikacije na serveru. Svaka aplikacija ovisi iskjučivo o svojim paketima.

Koristi dosljedan API[2] model napisan u .NET Standardu koji je zajednički svim .NET aplikacijama te tako osigurava zaštitu podataka. Isti API ili biblioteka može se koristiti s više platformi na više jezika. Napravljen je tako da omogućava brzi razvoj komponenti izvođenja, API-ja, kompajlera i jezika istovremeno pružajući stabilnu i podržanu platformu za održavanje aplikacije u radu [2].

U usporedbi s tradicionalnim .NET razvojnim okvirom omogućava optimizaciju performansi, kao što je bolja podrška za asinkrono i smanjeno vrijeme pokretanja. Ima snažnu podršku za kontejnerizaciju, kao što je Docker, omogućavajući programerima da 'spakiraju' aplikaciju u prijenosne spremnike [3]. Više verzija ASP.NET Core-a može istovremeno biti na istom poslužitelju, tako da jedna aplikacija može prihvatiti najnoviju verziju, dok druge aplikacije nastavljaju raditi na verziji na kojoj su testirane.

Ono što je najbitnije da Microsoft redovno izdaje ažuriranja i poboljšanja za .NET Core, uvodeći nove značajke, poboljšavajući performanse i rješavajući greške. Pružaju pravovremene sigurnosna ažuriranja za rješavanje potencijalnih ranjivosti. Tako se omogućava programerima da stalno imaju pristup najnovijim alatima i tehnologijama za uspješnu izradu sigurnih i vrhunskih aplikacija. Microsoftovo integrirano razvojno okruženje, Visual Studio pruža podršku za .NET Core razvoj, uređivanje koda, otklanjanje pogrešaka, testiranja i implementacije.

3.1. Povijest .NET Core-a

Microsoft je početkom 2000.-ih pokrenuo .NET okvir za izradu Windows aplikacija. Osim Windows sustava korisnici su počeli sve više koristiti i druge sustave, s toga je za izradu aplikacija bilo potrebno razviti fleksibilan okvir koji radi na različitim platformama. ASP.NET Core je prvi put objavljen 2016. godine, predstavljen kao moderni okvir dizajniran da bude skalabilan i lagan te može raditi na različitim platformama, kao što su Windows, macOS i Linux. ASP.NET Core je potpuno redizajniran od ASP.NET-a, izgrađen je od temelja kako bi bio lagan, modularan i višeplatformski. Predstavljen je kao novi model usluge poslužitelja, ima

[1] NuGet je bitan alat putem kojeg programeri mogu stvarati, dijeliti I koristiti koristan kôd
[2] API (eng. Application Programming Interface) sučelje aplikacijskog programa je softverski posrednik koji omogućuje da dvije aplikacije mogu međusobno komunicirati.

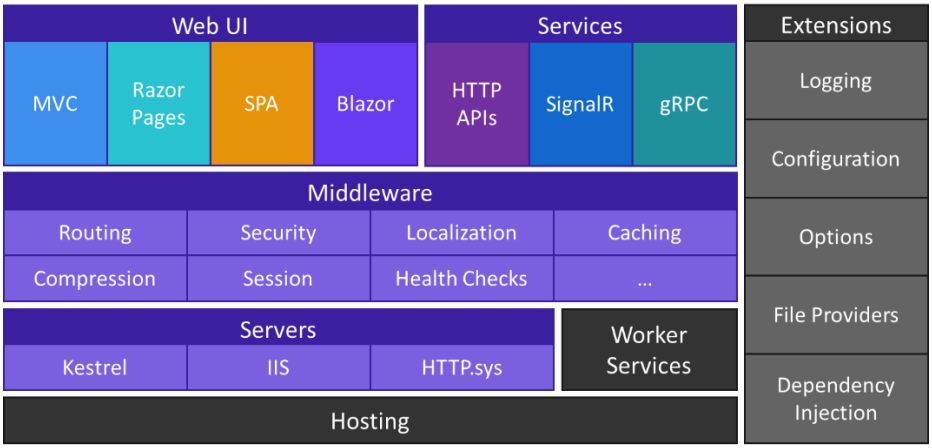
poboljšane performanse te moderni web razvoj. Kao prvo službeno izdanje okvira bio je .NET Core 1.0, koji je omogućavao korištenje biblioteka za izgradnju modernih višeplatformskih aplikacija. Međutim, imao je nedostatke, nedostajale su mu neke značajke i puna konpatibilnost nad .NET razvojnim okvirom. Značajna prekratnica u razvoju dogodila se 2020. godine kada je Microsoft izdao .NET 5 verziju kao jedinstvenu platformu za sav budući .NET razvoj. Omogućio je veću kompatibilnost s postojećim biblioteka, poboljšane performanse te pojednostavio tijek razvoja [3].

3.2. Karakteristike .NET Core-a

Okvir otvorenog kôda omogućava razvoj aplikacije bez ograničenja platforme i nemogućnosti kompatibilnosti. .NET Core okvir je iznova napisan okvir te tako ima moćne prednosti i značajke. Poboljšan je sigurnošću, samostalnim *hosting-om,* opremljen inovacijom koje omogućavaju izgradnju robusnih web aplikacija. Omogućena višestruka okruženja i način razvoja kao jedna od najcjenjenijih značajki .NET Core-a. Omogućava jednostavno razlikovanje softverskih kodova u smislu postavljanja i proizvodnje. Značajna karakteristika .NET Core-a je podrška za lokalizaciju i globalizaciju aplikacija koje služe različitim jezičnim i kulturnim preferencijama. Omogućuje lokaliziranje teksta, brojeva i datuma unutar same aplikacije [4].

Kod izvođenja tijeka operacija web aplikacija, postoji sinkrona i asinkrona komunikacija. Sinkrona komunikacija se izvršava, tako da nakon klijentovog slanja zahtjeva poslužitelju i dobivanja odgovora od poslužitelja se osvježava preglednik, te na taj način opada kvaliteta korisničkog iskustva. Dok kod asinkrone komunikacije, ne izvršava se osvježavanje preglednika, nego slanje zahtjeva i primanje odgovora se izvršava u pozadini aplikacije te tako istovremeno pruža interaktivnost aplikacije.

U .NET Core-u se koristi asinkrono programiranje, olakšavajući kontinuirani tijeka operacija te osnažuje performanse aplikacije s većom osjetljivošću. Na taj način je moguća istovremena obrada višestrukih zahtjeva, osiguravajući brz tijek rada.



Slika 1 ASP.NET Core - kompletna integrirana rješenja [5]

4. Blazor

Blazor je novi okvir za razvoj aplikacija korisničkog sučelja te je vrlo učinkovit i produktivan model programiranja. Dolazi od kombinacije riječi 'Browser' i 'Razor'. Njegova velika prednost je mogućnost pisanja bogatih web sučelja koristeći HTML, CSS i C# umjesto JavaScript-a. Kako bi se HTML prikazao Blazor poglede izvršava na klijentu. Blazor u potpunosti podržava Razor sintaksu te koristi njegovi puni skup značajki, kao korištenje petlji, uvjeta te komponente, kao što su elementi stranice, gumbi i sl. Odnosno možemo reći da Blazor gradi fleksibilna, interaktivna korisnička sučelja jedne ili više komponenti napisane Razor-ovom sintaksom [5][6].

Prvo izdanje Blazor-a uključivalo je interaktivnost poslužitelja i klijenta. Kasnije je dodano renderiranje na strani poslužitelja kako bi se poboljšala izvedba klijentskih aplikacija te pruža bolje korisničko iskustvo, brže učitavanje i optimizaciju za tražilice [7][8].

Aplikacije napravljene s Blazor-om mogu se izvršavati na poslužiteljskoj i klijentskoj strani. Blazor Server aplikacije se interaktivno izvršavaju na strani poslužitelja unutar ASP.NET Core aplikacije. Blazor Server komponente moraju imati interaktivni način prikaza, bilo to u datoteci definicije komponente ili naslijeđen od određene komponente. Pomoću protokola WebSockets i preko SignalR veze, izvršavaju se sva ažuriranja korisničkog sučelja, JavaScript pozivi i rukovanje događaja. SignalR veza u realnom vremenu pruža API za komunikaciju i implementaciju funckionalnosti između klijenta i poslužitelja. Stanje poslužitelja i klijenta povezano je krugovima, koji mogu tolerirati privremene prekide mreže i pokušaj da se klijent ponovno poveže s poslužiteljem kada se veza izgubi. Na primjer, otvaranje iste aplikacije na više zaslona preglednika ne zahtijeva dodatne resurse na poslužitelju, nego svaki zaslon zahtijeva poseban krug i instance stanja komponenti. Prilikom svakog zatvaranja preglednika, krug i pridruženi resursi se oslobađaju [9][10].

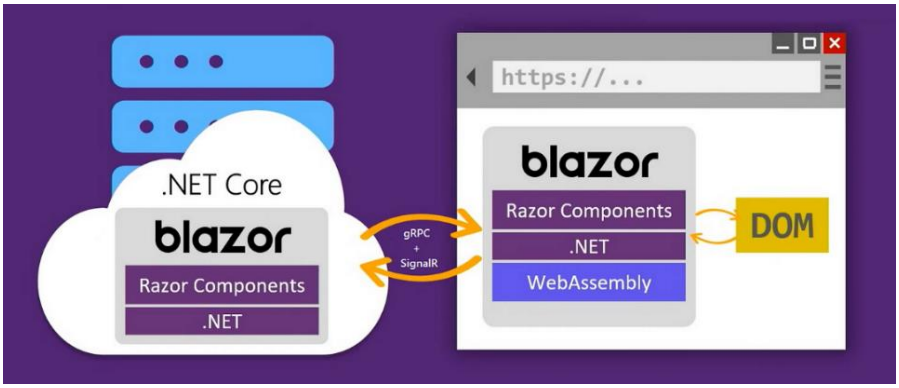
Dok na klijentskoj strani komponente funkcioniraju kako su i prikazane, odnosno rade u interakciji s Blazor WebAssembly-jem te se kôd komponente šalje klijetu koji se može dekompilirati i pregledati. Iz tog razloga nije dobar način osjetljive informacije stavljati u komponente koje prikazuje klijent [9]. Više o klijentskoj strani u poglavlju 3.3. WebAssembly.

Blazor se svakim danom nadograđuje i proširuje iskorištavanje prednosti poslužitelja te uključuje najnovije trendove web okvira. Blazor je potpuno integriran s okvirom poslužitelja te koristi jedan sustav i jezik izrade. Vrlo je fleksibilan iz razloga što postoji razlika između načina na koji izračunava promjene korisničkog sučelja i načina na koje se te promjene primjenjuju te tako stvara i izvorna sučelja za mobilne uređaje [8].

Neke osnovne značajke Blazor okvira su:

- snažan model komponenti
- dvosmjerno povezivanje podataka
- prikaz velikih skupa podataka s virtualizacijom
- izrada web aplikacije koja podržava offline rad
- jednostavno rukovanje UI događajima.

.NET kôd se izvršava izravno u pregledniku koristeći WebAssembly te tako pruža dinamički web sadržaj što ga čini bržim i manje troši mrežnu propusnost.



Slika 2 ASP.NET Core Blazor [9]

4.1. Blazor sintaksa

Blazor se temelji na Razor komponentama koje su napisane u obliku Razor stranice s ekstenzijom datoteke .razor. Razor je sintaksa koja omogućava kombiniranje HTML oznaka s C# kôdom te olakšava produktivnost programera. Blazor koristi HTML oznake za sastav korisničkog sučelja. U nastavku je dan primjer dijela kôda iz aplikacije ELARS.

```
@page "/mycourses"
@inject UserCookieService UserCookieService;
@using ElarsWebAss.Shared.Components;
@using SharedModels.Models.Domena;
@using ElarsWebAss.Shared.Components.Students;
@inject NavigationManager _navManager
@using SharedServices;
@inject ICourses course;
@inject IUser login;
@using SharedModels.Models;

<GenericPage>
    <div class="container">
        <h2>Moji kolegiji</h2>
            @if (showTable)
                {
                    <TableComponentMyCourses ObjectListParam="Table" IgnoreProperties="ignore"
                    NameHandlers="NameHandlers"
                    CourseModel="Kolegij"></TableComponentMyCourses>
                }
                <br />
   </div>
</GenericPage>
@code
{
```

```
    bool showTable = false;
    private static List<CourseModel> Kolegij = new List<CourseModel>();
    List<Object> Table = new List<object>();

    string IdUser;
    private Users User = new Users();

    List<string> ignore = new List<string> { "IDCourse", "Description", "Shema", "CourseObjective",
"CourseShortName", "UserId" };

    public Dictionary<string, string> NameHandlers = new Dictionary<string, string>
    {
        {"CourseName", "Kolegij"},
        {"BeginDate", "Datum početka"}
    };

    protected override async Task OnInitializedAsync()
    {
        User = await UserCookieService.GetUserFromCookie("cookie");

        if(User != null)
        {
            IdUser = User.IDUser;

            var getKolegij = await course.CoursesGet(IdUser);
            Kolegij = getKolegij.res;
            Kolegij.ForEach(item => Table.Add(item));

            showTable = true;
        }
        else
        {
            _navManager.NavigateTo("login");
        }

        StateHasChanged();
    }
}
```

Prikazani kôd odnosi se na stranicu Moji kolegiji, u sučelju za studente. Za korištenje usluga u komponenti koristi se *@inject* direktiva. Kako bi se dopustio pristup uvoza tipova definiranih unutar tih imenskih prostora se koristi *@using.*

<GenericPage> se odnosi na komponentu koju je napravio programer, jer se isti kôd koristi na više stranica, pa je komponenta najbolji način izbjegavanja pisanja duplog kôda. Kako bi se postavio neki uvjet za o kojemu ovisi da li će neka HTML oznaka biti prikazana ili ne, koristi se *@if*, u ovom primjeru konkretno *@if(showTable)*, gdje označava ako je *showTable* postavljen na *true* onda će biti prikazano.

@code direktiva koristi se za definiranje blokova C# kôda unutar Razor komponente u kojoj se implementira logika. Konkretno, u navedenom primjeru se odnosti prvo na dohvat podataka iz 'kolačića', odnosno provjere da li je korisnik prijavljen u aplikaciju. U slučaju da je, dohvaćaaju se kolegiji iz baze podataka, stavljaju u tablicu te se tablica postavlja na *true* kako bi se mogla prikazati. U slučaju da korisnik nije prijavljen, usmjerava ga na stranicu prijave.

Kolegij	Datum početka
Primjena hipermedije u obrazovanju 1	22.02.2024. 00:00
Izvannastavne informatičke i tehničke aktivnosti	01.10.2024. 00:00

Slika 3 Prikaz tablice na stranici 'Moji kolegiji'

4.2. Usporedba Blazor-a i Razor-a

Razor je sintaksa na poslužiteljskoj strani koji omogućava izradu dinamičkih web stranica pomoću napisanih pomoću HTML-a i .NET kôda te se koristi za povezivanje podataka, logiku obrade, implementaciju naredbi i sl. Razor generira web stranice iz Razor stranica napisane C# i VB kôdom. Stranice napisane VB kôdom imaju datotečni nastavak .vbhtml, a stranice napisane C# kôdom imaju datotečni nastavak .cshtml. Kada korisnik klikne na stranicu, preglednik šalje zahtjev poslužitelju, ulazi u bazu podataka, dohvaća .cshtml Razor stranicu, spaja podatke i oznake te sve vraća u preglednik [6].

Postoji odnos između Razor i Blazor komponenti. Blazor-ove komponente predstavljene su datotečnim nastavkom .razor te se koristi Razor sintaksa za logiku korisničkog sučelja.

Razor se izvršava na strani poslužitelja, što je bitno za učinkovit rad web aplikacija. Implementacija Razor-a omogućava robusnu sigurnost te tako smanjuje pogrešku tijekom izvođenja. Razor je prvenstveno prikladan za izradu statičnih web aplikacija te iz tog razloga nije prikladan za izradu interaktivnih web aplikacija.

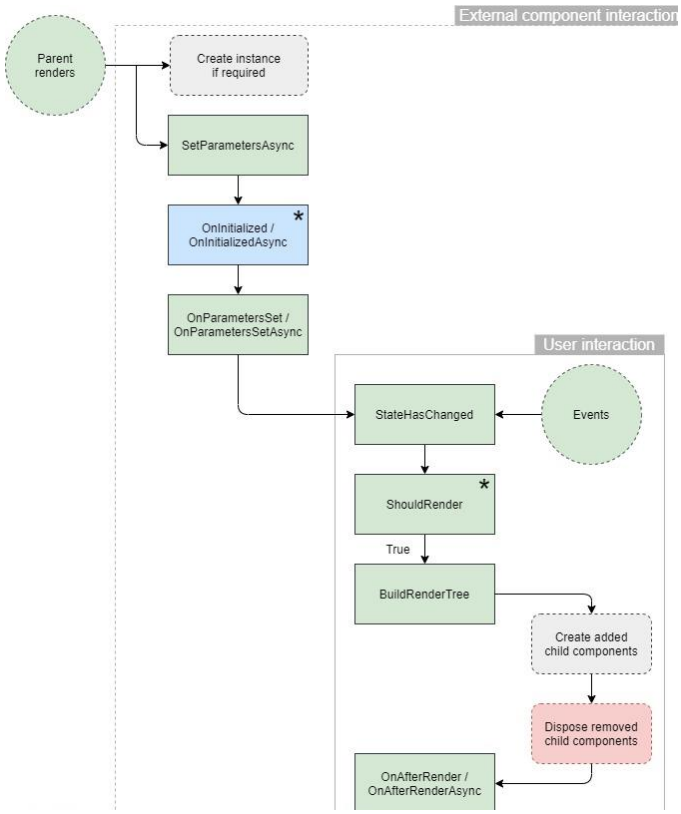
Razor ima prednost u tome što je vrlo fleksibilan, struktura mu je dobro organizirana i omogućuje umetanje C# kôda na web stranice. Njegov najveći nedostatak je taj što je potreban JavaScript za implementaciju dinamičkih interakcija na strani klijenta te je iz tog razloga teško upravljati više samostalnim stranicama [7].

4.3. Životni ciklus Blazor-a

Tijekom pozivanja izvođenja različitih operacija u kôdu, vrlo je bitan tijek izvođenja istih, kako se ne bi dogodilo da se pozove druga operacija, prije nego je prva završila. To omogućava životni ciklus Razor komponenti unutar Blazor-a u skupu sinkronih i asinkronih metoda koje

se mogu nadjačati u izvođenju. Postoje više grupa metoda u životnom ciklusu komponenti, koje dolaze od *ComponentBase* klase [11].

U nastavku je prikazan dijagram te objašnjeni događaji životnog ciklusa komponente.



Slika 4 Životni ciklus Blazor-a [12]

U koliko se komponenta prikazuje prvi put na zahtjev stvara se instanca komponente, uključuje se izvršavanje svojstava koje prosljeđuje roditeljska komponenta *SetParametersAsync*. Prije nego se komponenta prikaže, upućuje se poziv servisu kako bi se ispunili svi objekti koji su potrebni za prikaz na zaslonu, pozivom *OnInitialized* za sinkronu metodu ili *OnInitializedAsync* za asinkronu metodu, gdje zadatak čeka u koliko se vrati nedovršeni, a zatim se komponenta ponovno prikazuje. Nakon inicijalizacije tih parametara slijedi grupa metoda *OnParametersSet* i *OnParametersSetAsync* koje se pozivaju svaki put kada se prime novi parametri od roditelja [11][12].

Kako bi komponenta obavijestila korisničko sučelje da su se dogodile neke promjene koje bi rezultirale drugačijim prikazom i ponovno pokrenulo prikazivanje komponente, zaslužena je metoda *StateHasChanged*. Zatim slijedi metoda *ShouldRender* koja se koristi za sprječavanje *RenderTree* komponente, u slučaju da znamo da će stanje biti nepromijenjeno, tada onda niti nećemo proći kroz proces *BuildRenderTree* kako bismo uštedjeli vrijeme obrade. U koliko stavimo vrijednost na *true* korisničko sučelje će prisilno biti osvježeno, kako bismo spriječili da stanje objekta ostane u predmemoriji. Posljednje dvije metode *OnAfterRender* i *OnAfterRenderAsync* se izvršavaju svaki put kada Blazor ponovno generira *RenderTree,* u slučaju kada je potrebna ponovna interakcija korisnika s komponentom [11][12].

4.4. WebAssembly

Blazor se izvršava na poslužitelju ili se pokreće u samom pregledniku koristeći WebAssembly. WebAssembly (WASM) je binarni format skupa instrukcija dizajniran da bude kompatibilan s postojećim web standardima i tehnologijama, omogućujući integraciju s JavaScript-om, API-jima, DOM-om i značajkama preglednika [13].

WebAssembly pokreće Razor komponente i pripadajuće ovisnosti u pregledniku na strani klijenta. Rukovanje događajima i ažuriranje korisničkog sučelja odvija se unutar istog procesa. Prilikom izrade aplikacije koja radi isključivo na WebAssembly-ju dobiva se potpuno iskustvo web razvoja s .NET Core-om, omogućujući dijeljenje kôda između klijentskih i poslužiteljskih aplikacija, integraciju s Razor i MVC stranicama. Kako bi klijentska aplikacija mogla komunicirati sa svojom poslužiteljskom, omogućeno je korišrenjem različitih okvira za razmjenu protokola i poruka, kao na primjer SignalR i web API-ja [10].

Kolaborativni pristup osigurava da se WebAssembly razvija potičući inovacije i interoperabilnost na web platformi koji koristi programerima, preglednicima i krajnjim korisnicima. Prikladan je za širok raspon upotrebe izvan tradicionalnih web aplikacija, kao što su multimedijske aplikacije, virtualna i proširena stvarnost i sl. [11].

4.5. Arhitektura WebAssembly-ja

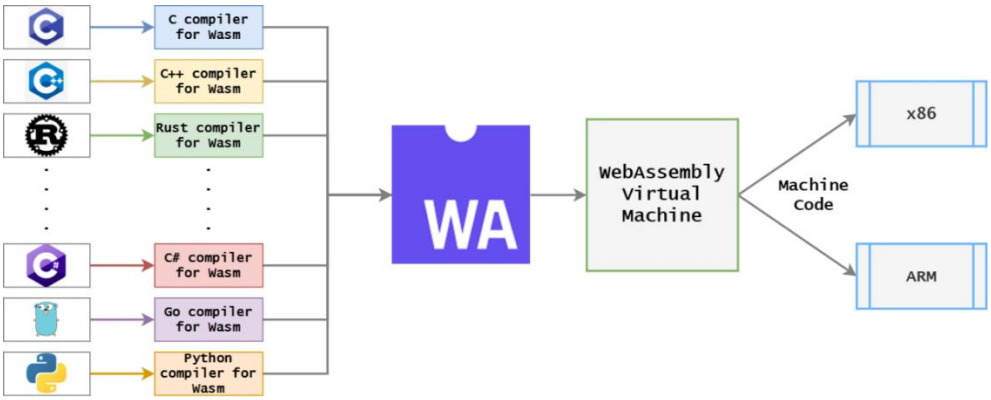
WebAssembly ima jednostavnu i učinkovitu athitekturu koja omogućava brz i siguran rad u web preglednicima. Vrti se oko virtualnog stroja, skupa učinkovitih instrukcija niske razine, sigurnog okruženja i interoperabilnosti s JavaScript-om omogućavajući visokoučinkovito i višeplatformsko izvršavanje koda u web preglednicima uz zadržavanje sigurnosti i prenosivosti [10].

U središtu WebAssembly-ja je virtualni stroj, neka vrsta „simuliranog računala“ koji je dizajniran za učinkovito izvršavanje WebAssembly koda. Taj specijaliziran mehanizam ugrađen je u preglednik koji omogućava pokretanje WebAssembly-ja. Ima vlastiti skup instrukcija koje su niske razine i dizajnirane za brzo izvršavanje. Takve upute uključuju pristup memoriji, operacije za aritmetiku, kontrolu toka i sl. WebAssembly kôd se pohranjuje u binarnom formatu koji je manji i brži za učitavanje te kompaktan i učinkovit način predstavljanja programa. Radi u zaštićenom okruženju, odnosno izoliran je od računala i web stranice na kojoj se nalazi, te tako sprječava zlonamjerni kôd da ošteti sustav [14].

Komunikacija s JavaScript-om je i dalje moguća, odnosno WebAssembly može pozivati JavaScript funkcije. Takav način omogućava interoperabilnost, gdje se JavaScript može koristiti za druge potrebne aspekte web aplikacije.

Kontrolirani model memorije programerima omogućava učinkovito dodjeljivanje i upravljanje memorijom. Takav memorijski model je potreban za izvođenje koda niske razine te povećava sigurnost, osiguravajući da programi ne pristupaju memoriji kojoj ne bi trebali.

WebAssembly je dizajniran da bude neovisan o platformi, odnosno može se pokrenuti u bilo kojem modernom pregledniku na različitim arhitekturama i operativnim sustavima.



Slika 5 Postavljanje kompilacije WebAssembly-ja [15]

5. Praktični rad

Najveći naglasak ovog diplomskog rada je na praktičnom dijelu koji uključuje izradu web aplikacije ELARS za *online* vrednovanje studenata koristeći .NET Core i Blazor. U sljedećim poglavljima je objašnjena struktura aplikacije, kretanje po stranicama te implementacija tehnologije.

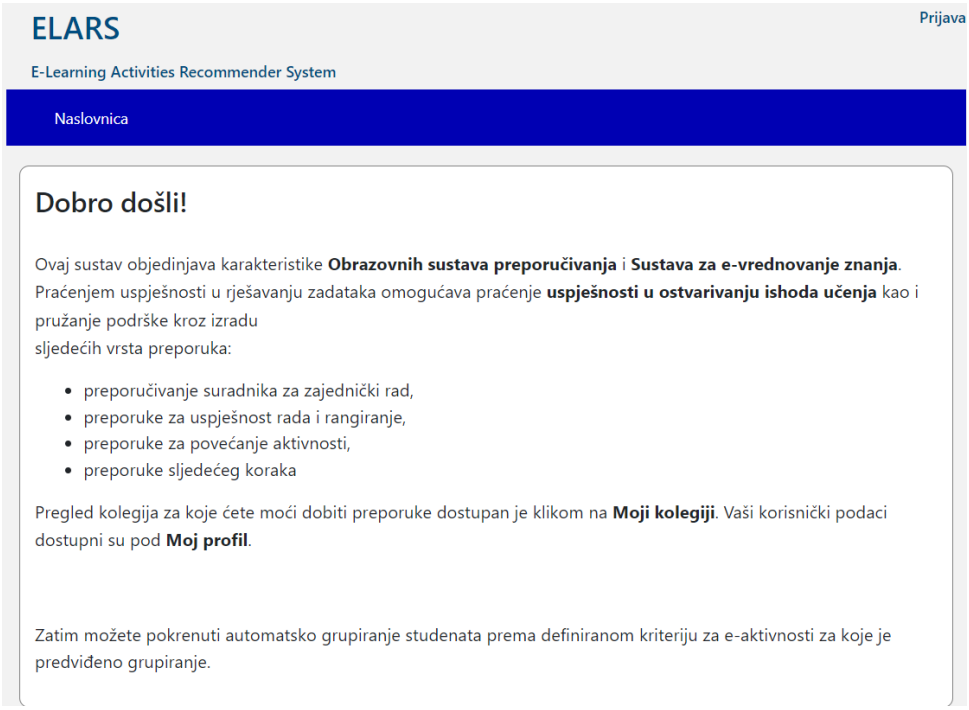
5.1. Elars aplikacija

ELARS - E-Learning Activities Recommender System je obrazovni sustav preporučivanja koji se koristi za preporučivanje aktivnosti u e-učenju kako bi se poboljšalo iskustvo učenja i potaknulo studente na veću aktivnost prilikom učenja te korištenje digitalnih alata u suradničkom učenju. Rad sustava se temelji na preporučivanju u skladu s karakteristikama studenata poput razine znanja, razine aktivnosti i preferencija kako bi se studentima pružila podrška u pronalaženju relevantnih resursa i aktivnosti. U posljednjoj verziji razvoja modela sustava, sustav je priagođen vrednovanju STEM području kako bi se omogućilo uvođenje kontinuiranog *online* vrednovanja znanja i praćenje uspješnosti studenata u usvajanju koncepata i ishoda učenja predmeta. U okviru ovog diplomskog rada, na temelju osmišljenog modela sustava za online vrednovanje, izrađena je web aplikacija.

Aplikacija ima dio za profesore i dio za studente. Nakon što korisnik unese podatke za prijavu, provjerava se da li je korisnik profesor ili student te shodno tome ulazi u aplikaciju. Profesori mogu vidjeti svoje kolegije ili kreirati nove. Unutar svakog kolegija su definirani moduli učenja i te se mogu vidjeti studenti koji idu na taj kolegij. Na svakoj stranici je omogućeno kreiranje novog unosa, izmjena, pregled detalja i brisanje. Studenti imaju vrlo jednostavnu kretnju kroz aplikaciju. Ponuđen im je popis kolegija te unutar svakog imaju popis modula učenja, a unutar modula, popis aktivnosti. Svakoj aktivnosti su pridruženi određeni zadaci koje student može rješavati. U sljedećem poglavlju je prikazana kretnja kroz stranica.

5.2. Opis sučelja aplikacije

Prilikom dolaska na početnu stranicu aplikacije prikazuje se Naslovnica na kojoj su navedene kratke informacije.



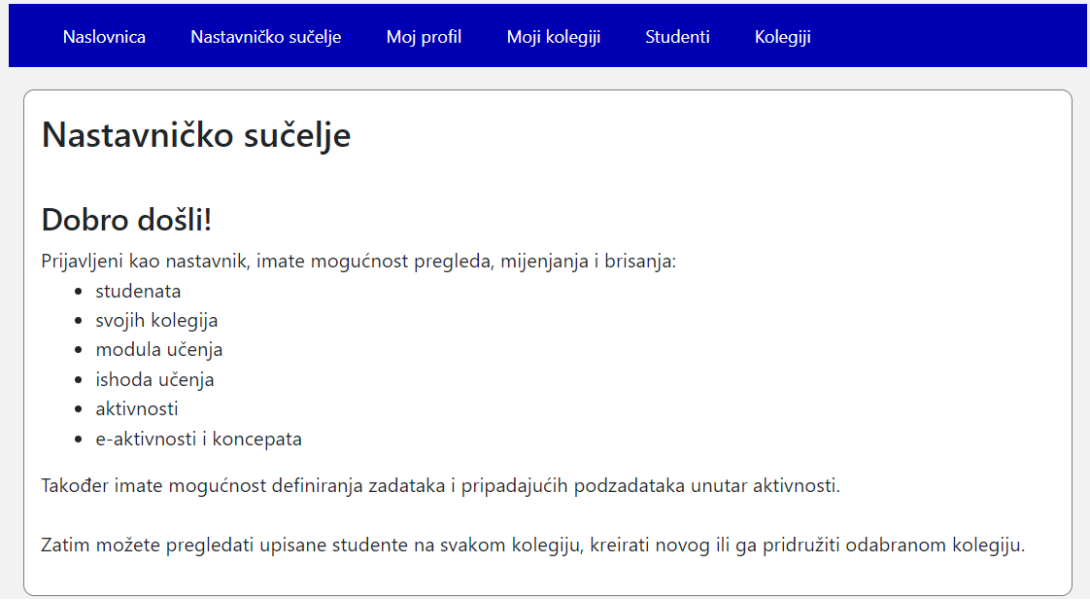
Slika 6 Prikaz stranice 'Naslovnica'

Svaki korisnik prilikom ulaska u aplikaciju upisuje svoje korisničko ime i lozinku. U koliko je lozinka točna, provjerava se da li je korisnik nastavnik ili student te shodno tome ulazi u sučelje u kojemu može izvršavati pojedine radnje aplikacije. U nastavniku je prvo prikazana kretnja kroz sučelje za nastavnike, zatim za studente.

5.2.1. Sučelje za nastavnike

Nastavnik u svom sučelju ima mogućnost kreirati kolegij koji predaje i zatim mu kreirati sve potrebne stranice kako bi kolegij bio funkcionalan. Prvo kreira module učenja te unutar svakog modula definira koncepte, ishode učenja i aktivnosti. U dijelu Uredi koncepte nastavnik mora unijeti novi koncept kako bi ga onda mogao pridružiti pojedinom ishodu učenja. Najvažniji dio su aktivnosti gdje je za svaku određuje tip. U koliko je tip aktivnosti STEM provjera unutar nje se mogu definirati zadaci koje će student moći rješavati te pripadajuće podzadatke koji se odnose na rješenja zadataka. Svaki međurezultat i rezultat se može povezati s odgovarajućim konceptima.

Ulaskom u aplikaciju, prva stranica Nastavničko sučelje daje nastavniku osnovne upute što mu je dostupno.



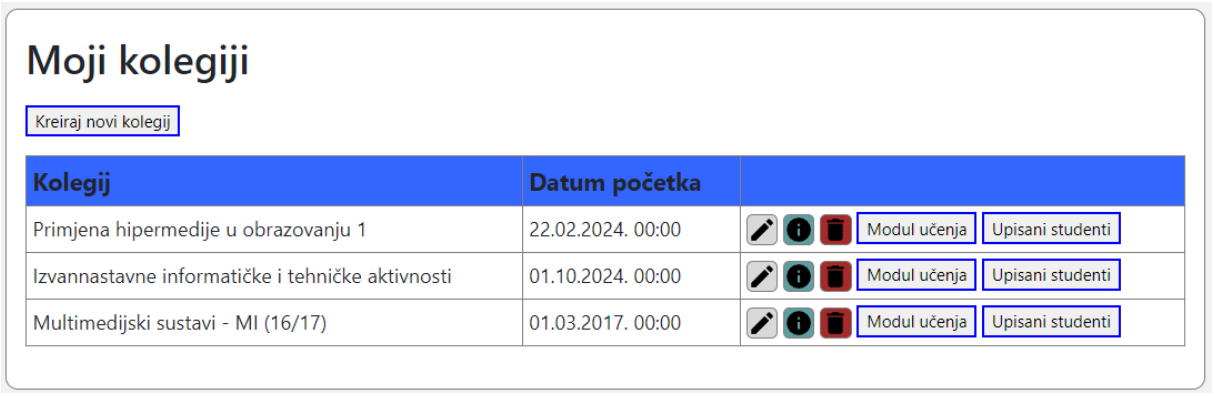
Slika 7 Prikaz stranice 'Nastavničko sučelje'

Nastavnik u aplikaciji odabirom stranice Moj profil može promijeniti svoje ime, prezime ili lozinku. Nadalje, pod Moji kolegiji vidi isto sučelje kao i studenti, kako bi mogao provjeriti da li se studentima ispravno prikazuju sve potrebne informacije. Taj dio će biti objašnjen u sučelju za studente.

Na stranici Studenti nastavnik ima popis svih studenata, te može pretražiti pojedinog studenta, kreirati novog ili izmijeniti podatke odabranog studenta.

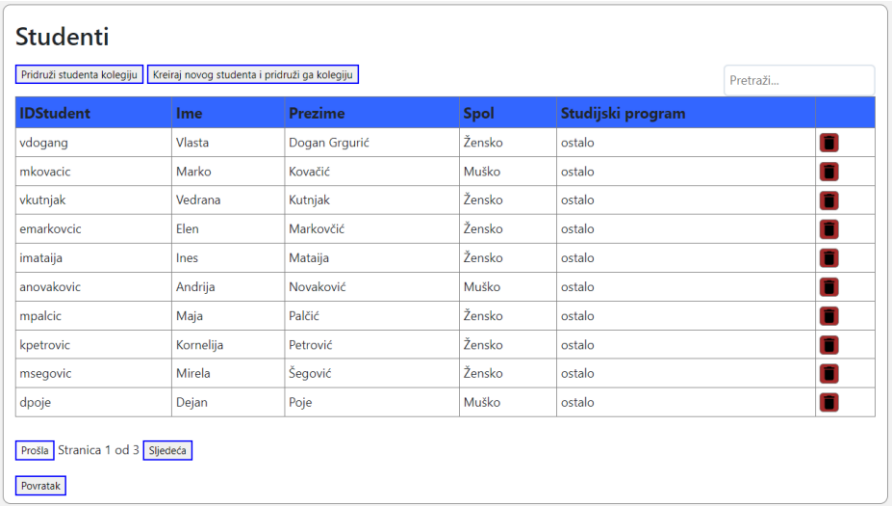
Slika 8 Prikaz stranice 'Studenti'

Odabirom Kolegiji nastavnici imaju uvid u svoje kolegije. Mogu kreirati novi kolegij te izmijeniti ili obrisati postojeći kolegij. Za svaki kolegij mogu vidjeti module učenja i upisane studente.



Slika 9 Prikaz stranice 'Moji kolegiji'

Na stranici Upisani studenti, koja je prikaza na slici 10 prikaže se popis studenata za odabrani kolegij. Nastavnik može pridružiti postojeć studenta trenutnom kolegiju ili kreirati novi i pridružiti ga. Osim toga, svakog studenta može maknuti iz kolegija.

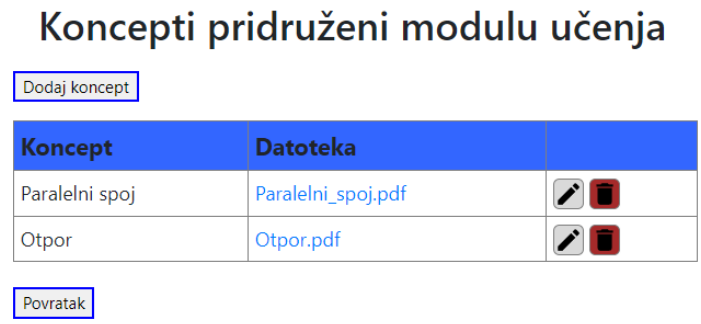


Slika 10 Prikaz stranice 'Upisani studenti'

Dolaskom na stranicu Moduli učenja omogućeno je kreiranje novog modula, pridruživanje postojećeg te izmjena, pregled detalja i brisanje modula iz kolegija. Za svaki modul, potrebno je definirati redni broj modula, naziv i označiti da li je modul aktivan. Kao što se vidi na slici 11, svakom modulu su pridruženi ishodi učenja, aktivnosti i koncepti.

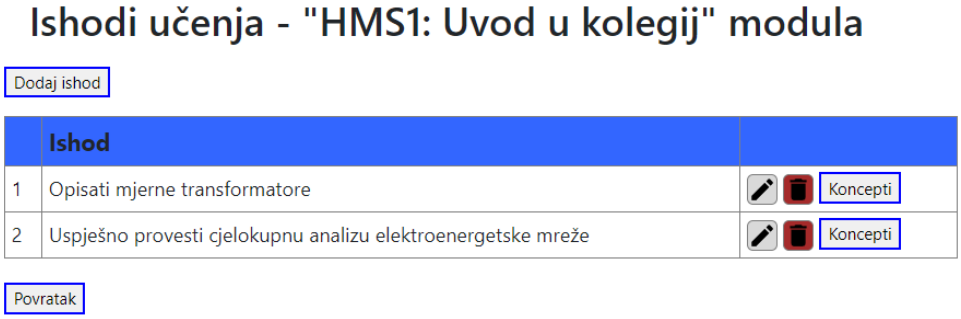
Slika 11 Prikaz stranice 'Moduli učenja'

Potrebno je prvo unijeti koncept na stranici Uredi koncepte, kako bi se poslije mogli pridružiti pojedinom ishodu učenja. Za svaki koncept se definira ime i učitava potrebni dokument.



Slika 12 Prikaz stranice 'Uredi koncepti'

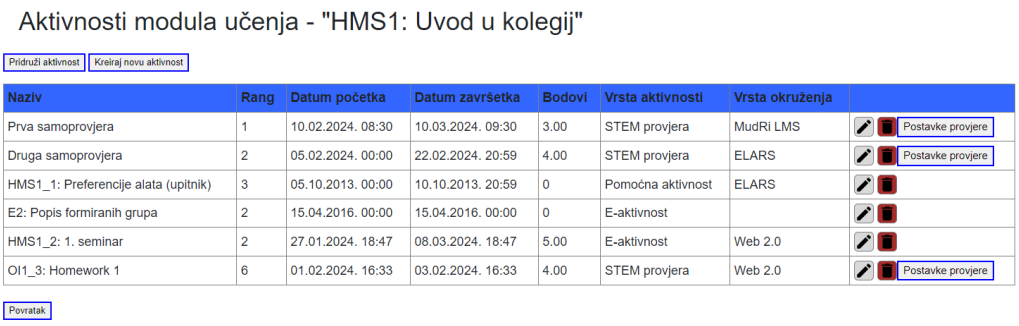
Odabirom Ishodi učenja, nastavniku se prikazuju ishodi za svaki modul. Može se kreirati novi ishod, izmijeniti, obrisati iz modula ili pridružiti koncept postojeći koncept.



Slika 13 Prikaz stranice 'Ishodi učenja'

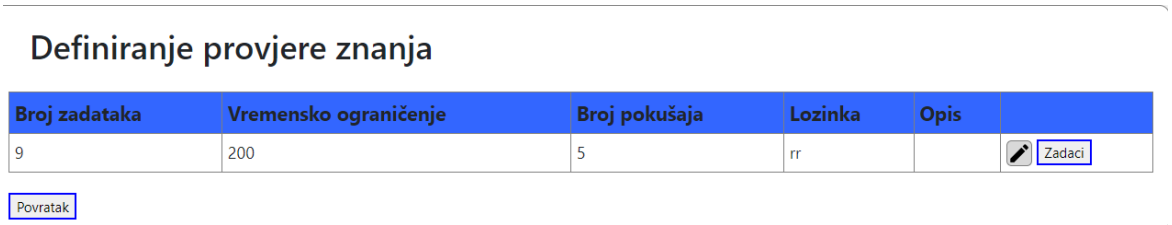
Slika 14 Prikaz stranice 'Koncepti'

Na stranici Aktivnosti može se kreirati nova ili pridružiti postojeća aktivnost. Za svaku aktivnost definiran je naziv, datum početka i završetka, rang aktivnosti, bodovi te vrsta aktivnosti i okruženja. U koliko je za vrstu aktivnosti odabrana STEM provjera, omogućen je gumb za odlazak na stranicu Postavke provjere, dio koji se odnosi na definiranje zadataka. U suprotnom aktivnost se može samo promijeniti ili izbrisati.



Slika 15 Prikaz stranice 'Aktivnosti'

Postavke provjere odnosi se na dio gdje se definira broj zadataka za pojedinu aktivnost, vremensko ograničenje i broj pokušaja rješavanja zadatka te lozinka.

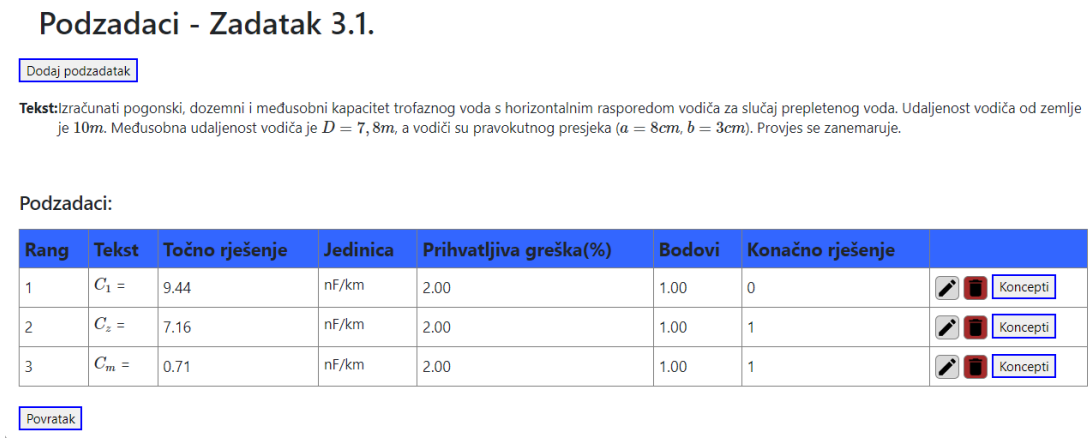


Slika 16 Prikaz stranice 'Postavke provjere'

Na stranici Zadaci može se dodati novi zadatak, izmijeniti ga ili obrisati. U koliko je neki zadatak već dodan prikazati će se u tablici. Ako je zadatak aktivan prikazati će se studentu, a ako nije neće.

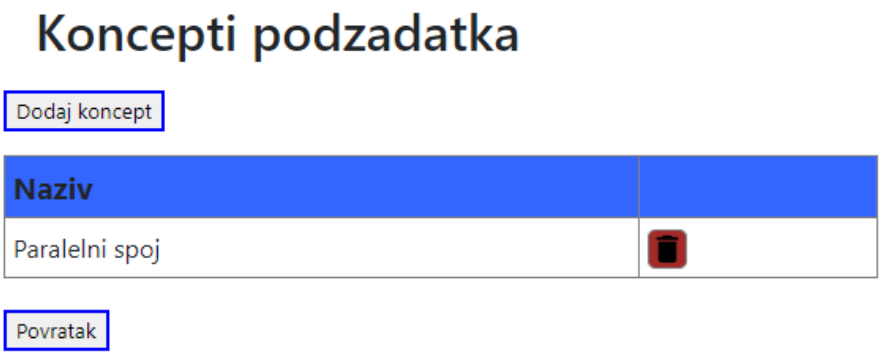
Slika 17 Prikaz stranice 'Zadaci'

Na stranici Podzadaci unose se rezultati. Za svaki podzadatak potrebno je unijeti tekst, točno rješenje, mjernu jedinicu, prihvatljivu grešku, bodove te da li je podzadatak konačno rješenje ili nije. U koliko nije konačno rješenje biti će prikazan kao međurezultat, a u koliko je onda kao rezultat.



Slika 18 Prikaz stranice 'Podzadaci'

Svakom podzadatku se može pridružiti koncept koji se definira na stranici Uredi koncepte.



Slika 19 Prikaz stranice 'Koncepti'

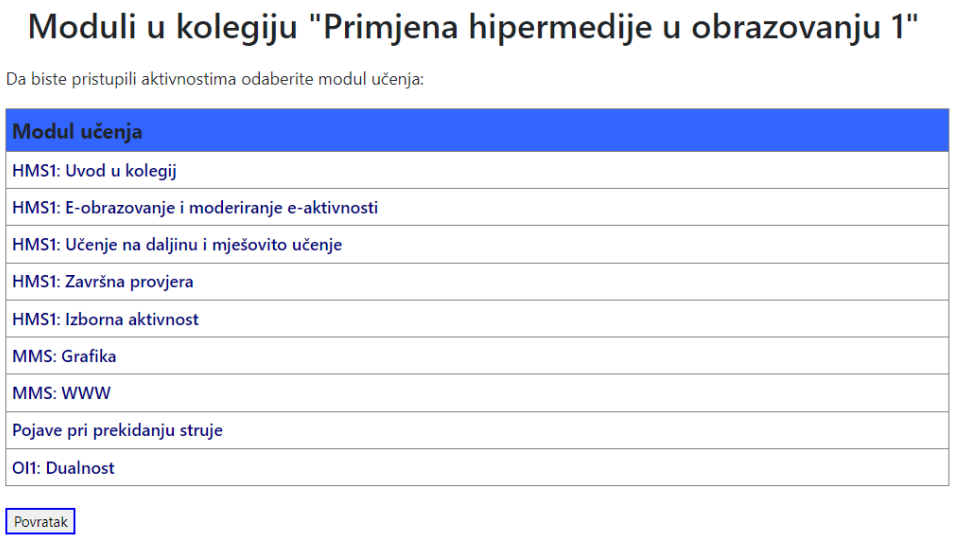
5.2.2. Sučelje za studente

Ulaskom u aplikaciju student može vidjeti svoj profil i kolegije na koje je upisan. Unutar profila može izmijeniti samo lozinku. Sučelje za studenta prvenstveno je namijenjen za rješavanje zadataka.

Na stranici Moji kolegiji svaki student vidi kolegije na koje je upisan i datum početka.

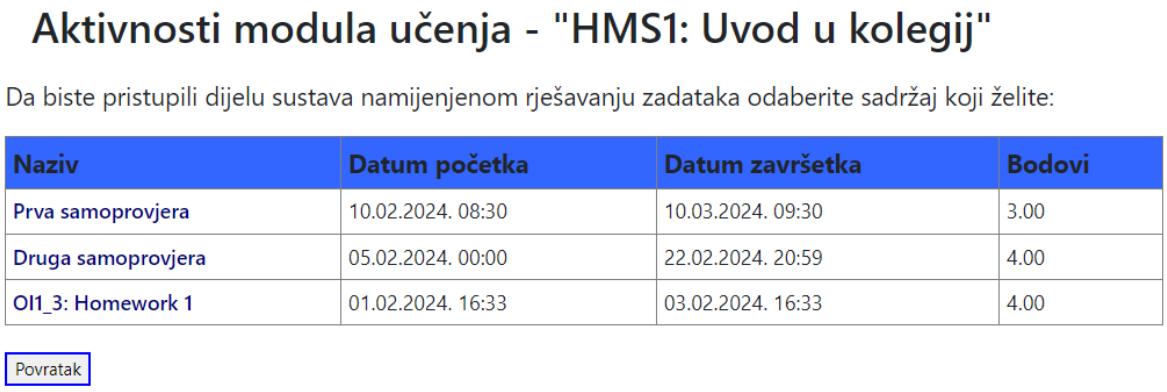
Slika 20 Prikaz stranice 'Moji kolegiji'

Za svaki kolegij student može vidjeti aktivne module učenja.

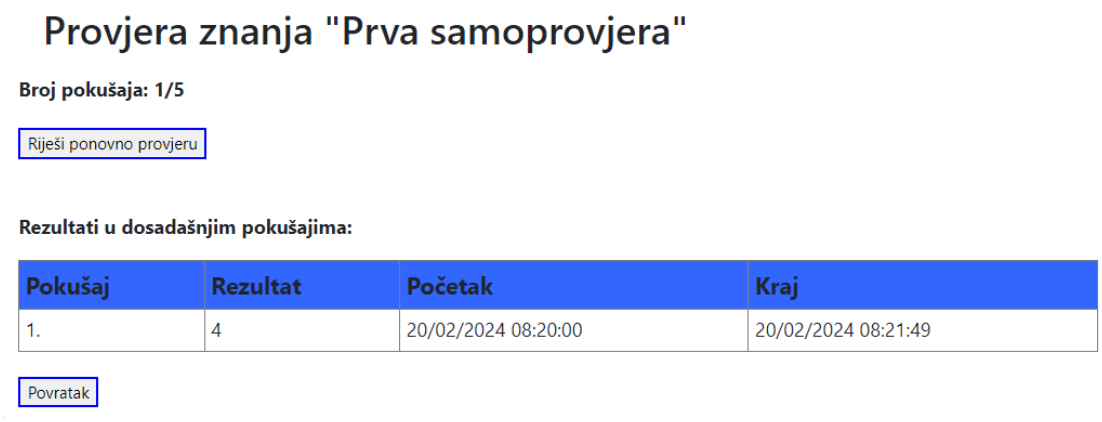


Slika 21 Prikaz stranice 'Moduli učenja'

Nakon odabira željenog modula učenja prikazuju se aktivnosti, datum početka i završetka te bodovi.

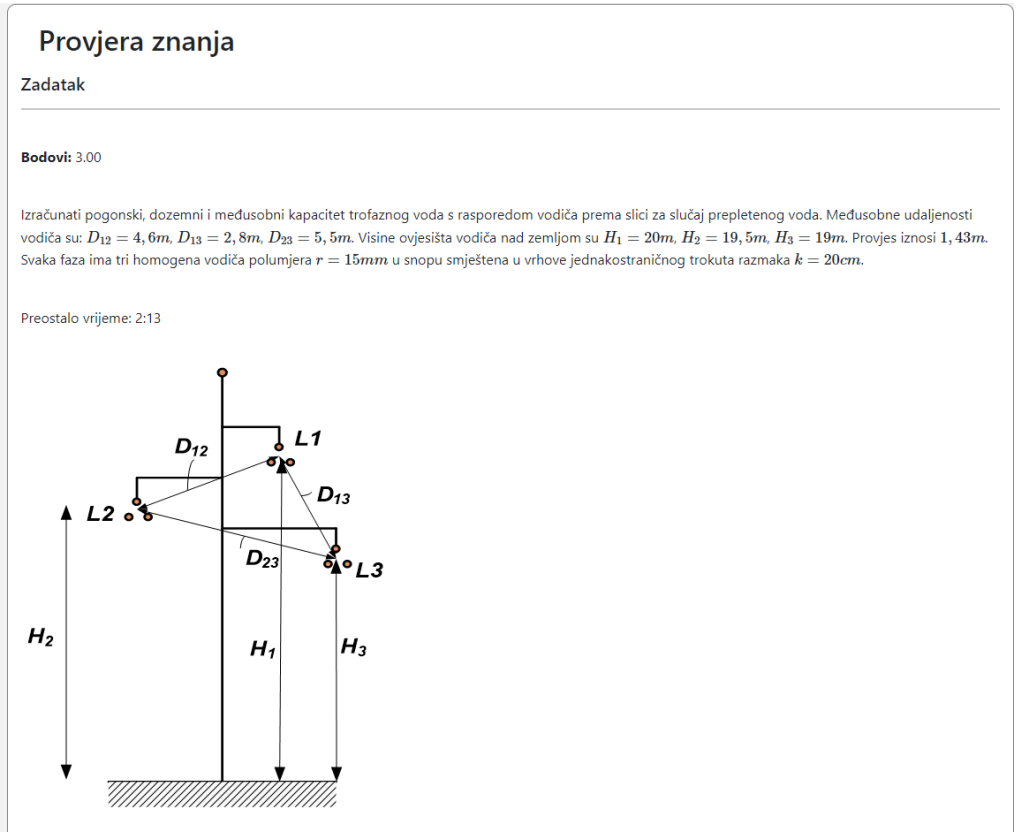


Kako bi student pristupio dijelu za rješavanja zadataka mora odabrati aktivnost. Zatim se dolazi na stranicu Provjere znanja, gdje se prvo provjera da li je već pokrenut neki zadatak koji još nije završio. U koliko je, biti će prikazano dugme 'Povratak na zadatak' te se rješeva taj tekući zadatak. A u koliko nije, prikazuje se dugme 'Riješi ponovno provjeru' gdje se onda odabire neki slučajan zadatak. Kada student riješi neki zadatak u tablicu se prikaže rezultat, datum početka i završetka rješavanja. Broj pokušaja se odnosi na to koliko puta student može rješavati zadatke za neku aktivnost. Kada iskoristi sve pokušaje, više ih nije u mogućnosti rješavati, nego samo pregledavati riješene zadatke.

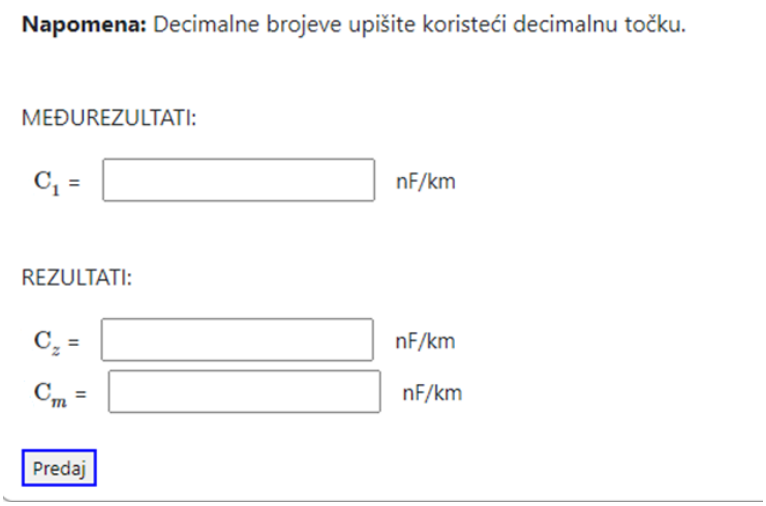


Slika 22 Prikaz stranice 'Provjera znanja'

Pritiskom na 'Riješi ponovno provjeru', provjerava se da li je definirana lozinka. U koliko je, student je mora unijeti kako bi pristupio zadatku. Ulaskom u zadatak, pokreće se odbrojavač, unutar koje vremena student može riješavati zadatak. Kada vrijeme istekne, zadatak završava. Za svaki zadatak prikazani su bodovi, tekst, slika zadatka te međurezultati i rezultati koje je potrebno izračunati i unijeti.



Slika 23 Prikaz zadatka na stranici 'Provjera znanja'



Slika 24 Prikaz unosa rješenja za pripadajući zadatak

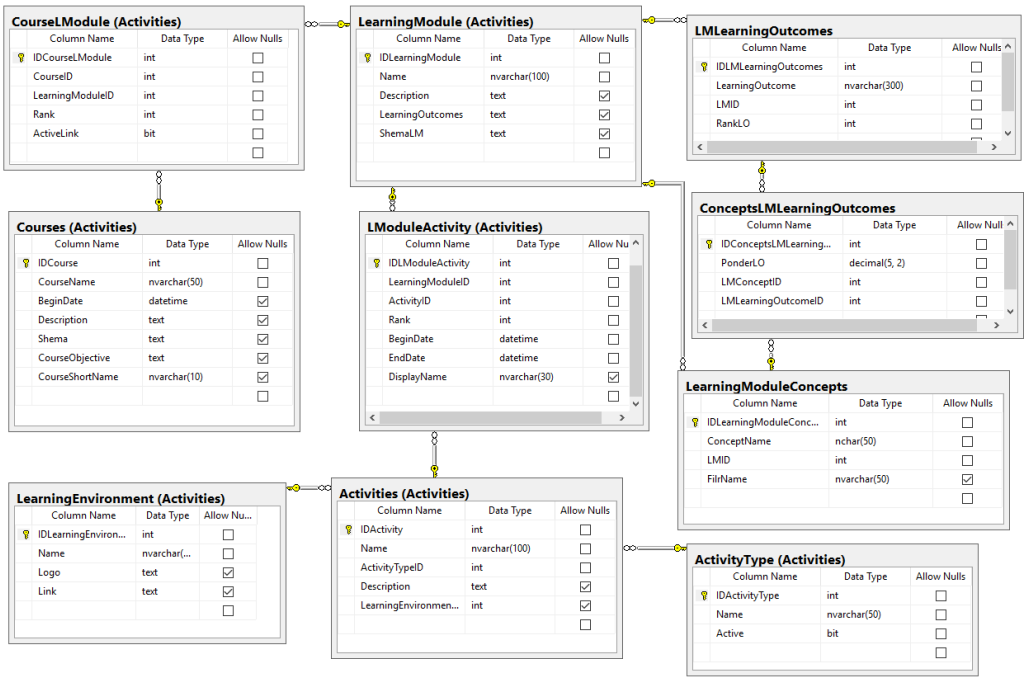
Nakon što korisnik klikne na 'Predaj' ili po isteku vremena, provjeravaju se uneseni rezultati s točnima te se na stranici 'Analiza' prikazuju se da li su unesena rješenja točna i koliko bodova je student skupio.

Slika 25 Prikaz rezultata na stranici 'Analiza'

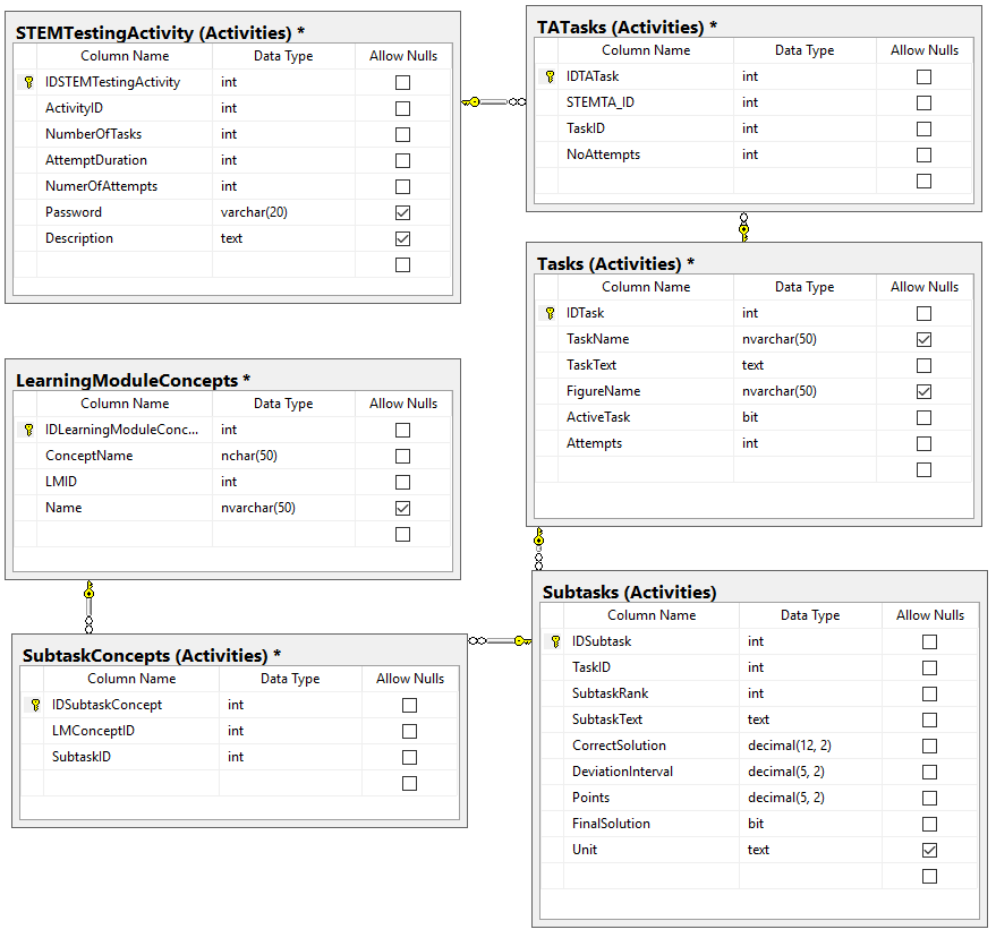
5.3. Implementacija

U ovom dijelu rada prikazana je korištena shema podataka za izradu nastavničkog sučelja, objašnjena je implementacija korištene tehnologije te su navedeni i objašnjeni neki primjeri kôda koji su specifični u izradi aplikacije. Na kraju je prikazano rješenje responzivnosti aplikacije

Za aplikaciju ELARS bilo je potrebno napraviti bazu podataka. Pomoću sheme baze podataka prikazane na slikama 21 i 22 koristile su se relacije i atributi za realizaciju ove aplikacije.



Slika 26 Shema dijela modela baze podataka predmeta, ishoda i koncepta



Slika 27 Shema dijela modela baze podataka aktivnosti, zadataka i podzadataka

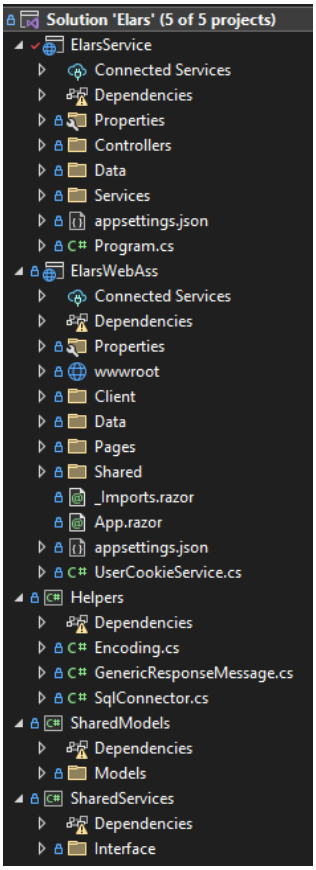
Za svaki podatak definiran je tip i da li može biti bez vrijednosti. Kada podatak ima ograničenje da ne smije biti prazan, gdje god se unosi ili ažurira, stavljena je provjera da li je unesen. U koliko nije, korisniku se prikaže obavijest da je obavezno unijeti to polje. To osigurava da se u bazu pravilno upišu podaci. Za dohvat podataka iz tablica, izmjenu i ažuriranje istih koristile su se pohranjene procedure. Procedurama se proslijeđuju parametri, tako da pohranjena procedura može dijelovati na temelju vrijednosti parametra. Korištenje pohranjenih procedura smanjuje mrežni promet i poboljšava vrijeme odziva. Primjer kako izgleda proslijeđivanje parametra i pozivanje procedure, prikazano je slijedećim kôdom.

```
using (SqlConnection conn = sqlConnector.Clone())
{
   conn.Open();

   SqlCommand cmd = new SqlCommand("Activities.CoursesGet", conn);
   cmd.CommandType = CommandType.StoredProcedure;

   cmd.Parameters.Add("@UserId", System.Data.SqlDbType.NVarChar).Value = IdUser;

   using (var reader = cmd.ExecuteReader())
   {
      while (reader.Read())
      {
         CourseModel m = new CourseModel();

         m.IDCourse = Convert.ToInt32(reader["IDCourse"]);
         m.CourseName = reader["CourseName"].ToString();
         if (reader["BeginDate"] != DBNull.Value)  m.BeginDate = Convert.ToDateTime(reader["BeginDate"]);
         if (reader["Description"] != DBNull.Value)  m.Description = reader["Description"].ToString();
         if (reader["Shema"] != DBNull.Value)  m.Shema = reader["Shema"].ToString();
         if (reader["CourseObjective"] != DBNull.Value)  m.CourseObjective = reader["CourseObjective"].ToString();
         if (reader["CourseShortName"] != DBNull.Value)  m.CourseShortName = reader["CourseShortName"].ToString();

         rez.res.Add(m);
      }
   }
   conn.Close();
}
```

Pomoću klase *SqlConnection* uspostavlja se veza s bazom podataka te *conn.Open()* otvara vezu s bazom podataka. *CommandType.StoredProcedure* navodi da je tip naredbe *Activities.CoursesGet* pohranjena procedura. Zatim se pomoću *cmd.Parameters.Add()* prosljeđuje odgovarajući parameter. U Koliko pohranjena procedura vraća neku vrijednost, potrebno je te vrijednosti primiti i pohraniti ih u model, pozivom *cmd.ExecuteReader()*. Prilikom dohvata podataka, ako je za neki definiran da može biti prazan, potrebno je provjeriti da li procedura vraća vrijednost za isti te u slučaju da vraća, može se upisati u model. Procedurama koje ne vraćaju nikakve vrijednosti, potrebno je dati naredbu *cmd.ExecuteNonQuery()*.

Kod izrade web aplikacije koristi se softverski dizajn Model-View-Controller (MVC) koji dijeli povezanu programsku logiku na tri međusobno povezana elementa. Ti elementi su interni prikazi informacija (model), sučelje (view) koji predstavlja informacije korisniku i iste dohvaća te upravljački softver (controller) koji povezuje prikaz informacija i sučelje.

Za izradu aplikacije korišteni su Blazor WebAssembly i ASP.NET Core Web API predlošci. Prilikom zahtjeva korisnika za neku izvršavanje neke aktivnosti klasa unutar WebAssembly predloška enkapsulira logiku za dohvaćanje podataka te komunicira s udaljenom uslugom preko HTTP zahtjeva. Definirana su sučelja unutar SharedService-s predloška koja se odnose kao ugovori. Svaka klasa koja implementira sučelje mora sadržavati implementaciju svih članova definiranih u sučelju. Kontroler unutar predloška ASP.NET Web API odgovoran je za rukovanje HTTP zahtjeva koje dobiva putem HTTP metode i URL zahtjeva. Kontroler poziva potrebnu logiku, nazvanom *Data*, koja se nalazi u istom predlošku kao i kontroler. Uključuje interakciju s pohranjenom procedurom, kao na primjer dohvaćanje, mijenjanje ili brisanje podataka. Nakon uspješne obrade podataka, kontroler vraća HTTP odgovor klijentu, Web Assembly prima odgovor i ažurira korisničko sučelje vežući vraćene podatke za komponente korisničkog sučelja. Ovakva struktura slijedi tipičan obrazac u kojima se sučelja ubacuju u klase promičući labavu spregu i lakše testiranje jedinice. Navedeni pristup koristi se u web aplikacijama na strani klijenta gdje treba komunicirati s API-jem na strani poslužitelja kako bi se podaci ispravno dohvatili.



Slika 28 Koriteni predlošci u aplikaciji

Korištene metode životnog ciklusa u Blazor komponentama su *OnInitializedAsync* i *OnParametersSet*. *OnInitializedAsync* se poziva asinkrono, pri inicijalizaciji komponente te je nadjačan u nadređenim komponentama. Koristi se za zadatke inicijalizacije, koje uključuju asinkrone operacije, kao na primjer dohvaćanje podataka s udaljenog poslužitelja. Na svako pozivanje servisa za dobivanje, unos, izmjenu ili brisanje podataka koristilo se asinkrono programiranje, kako bi se omogučilo da prelazak na sljedeću aktivnost bude moguće tek kada se prva izvrši do kraja.

U sljedećem primjeru, prikazana je metoda *OnInitializedAsync*.

```
protected override async Task OnInitializedAsync()
{
   User = await UserCookieService.GetUserFromCookie("cookie");

   if(User != null)
   {
      IdUser = User?.IDUser;


      var getKolegij = await course.CoursesGet(IdUser);
      Kolegij = getKolegij.res;
      Kolegij.ForEach(item => Table.Add(item));

      showTable = true;

      StateHasChanged();
   }
   else
   {
      _navManager.NavigateTo("login");
   }
}
```

Prvo, pomoću asinkrone operacije dohvaća korisničke podatke iz 'kolačića'. Ako su korisničke informacije uspješno dobivene, dohvaćaju se kolegiji. Nakon izvršavanja asinkronih operacija, poziva se *StateHasChanged()* kako bi se obavijestilo korisničko sučelje da se možda stanje komponente promijenilo te da ga treba ponovno prikazati.

Metoda *OnParametersSet* koristi se za reagiranje na postavljanje ili promjene ulaznih parametara, izvođenje inicijalizacije tih promjena te za ažuriranje tanja na temelju tih promjena. U sljedećem primjeru navedena metoda je nadjačana u podređenoj komponenti.

```
protected override void OnParametersSet()
{
   ObjectList = ObjectListParam;

   if(ObjectListParam.Count > 0)
   {
      myType = ObjectListParam[0].GetType();
      ObjectList = new List<object>();
      ObjectListParam.ForEach(obj => ObjectList.Add(obj));
      props = new List<PropertyInfo>(myType.GetProperties());
   }
```

```
    StateHasChanged();
}
```

Dodjeljuje se vrijednost parametra u *ObjectList,* te ako ima elemente, inicijaliziraju se lokalne varijable, na temelju tipa i svojstava objekata u *ObjectListParam.* Poziva se *StateHasChanged()* kako bi se zatražilo ponovno prikazivanje komponente.

Na svakoj stranici za prikaz tablice korištene su komponente radi jednostavnosti i bolje preglednosti nad kôdom. Osim tablica, komponente su korištene i kod prikaza dugmeta, gdje se jedna komponenta poziva na više stranica.

Za prikaz ikona izmjene, detalja i brisanja pojedinog podatka koristio se MudBlazor, biblioteka komponenti temeljena na materijalnom dizajnu koja je jednostavna za korištenje.

Slika 29 Prikaz ikona

Nakon instalacije NuGet paketa za korištenje MudBlazor biblioteke, u kôd se jednostavno implementira, prikazano primjerom:

```
<MudIcon Icon="@Icons.Material.Filled.Edit" Class="svg" @onclick="buttonActions"></MudIcon>
```

Omogućavanje prijave u aplikaciju korišten je JavaScript, te tako prikazano da se u Blazor aplikaciju može koristiti JavaScript jezik te će biti potpuno funkcionalan. Definirane su tri JavaScript funkcije: *WriteCookie*, *ReadCookie* i *DeleteCookie* koje pružaju osnovno sučelje za upravljanje 'kolačićima' u web pregledniku.

```
window.WriteCookie = {
    WriteCookie: function (name, value, days) {

        var expires;
        if (days) {
            var date = new Date();
            date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
            expires = "; expires=" + date.toUTCString();
        }
        else {
            expires = "";
        }
        document.cookie = name + "=" + value + expires + "; path=/";
    }
}
```

U prikazanom primjeru, funkcija je odgovorna za pisanje 'kolačića' s danim imenom, vrijednošću izbornim danom isteka. Provjerava se da li je primljen parametar dana naveden, ako je, onda se izračunava datum isteka na temelju trenutnog datuma i vremena s nadodanim parametrom. U koliko parametar dana nije naveden, 'kolačić' će isteći kada sesija preglednika

završi. Postavljeni dani su definirani prilikom prijave korisnika, gdje se i poziva ova JavaScript funkcija.

Nastavnik ima mogućnost učitavanja datoteka i slika u aplikaciju, koje omogućava komponenta *<InputFile>* te se pokreće metoda za učitavanje datoteke. Metoda poziva objekt *MultipartFormDataContent* za pripremu datoteke na učitavanje i stvara objekt *StreamContent* iz odabrane datoteke te ga dodjeljuje sadržaju. Zatim se sa sadržajem datoteke šalje zahtjev navedenoj API krajnjoj točki i poziva se kontroler u kojem je definirana metoda UploadFile, prikazana sljedećim kôdom:

```
public async Task<ActionResult<UploadResult>> UploadFile(IFormFile files)
{
    UploadResult uploadResults = new UploadResult();
    string result;
    string extension;
    string sharedPathDocuments = _configuration["SharedPath_Documents"];

    var uploadResult = new UploadResult();
    string trustedFileNameForFileStorage;
    var untrastedFileName = files.FileName;
    uploadResult.FileName = untrastedFileName;
    var trustedFileNameForDisplay = WebUtility.HtmlEncode(untrastedFileName);

    result = Path.GetFileNameWithoutExtension(trustedFileNameForDisplay);
    extension = Path.GetExtension(trustedFileNameForDisplay);

    trustedFileNameForFileStorage = string.Format("{0}", Path.GetRandomFileName().Replace(".", string.Empty));
    trustedFileNameForFileStorage = string.Format($"{trustedFileNameForFileStorage}{extension}");

    var path = Path.Combine(sharedPathDocuments, trustedFileNameForFileStorage);
    result = Path.GetFileName(path);

    await using FileStream fs = new(path, FileMode.Create);
    await files.CopyToAsync(fs);

    uploadResult.StoredFileName = trustedFileNameForFileStorage;
    uploadResults = uploadResult;

    return Ok(uploadResults);
}
```

Metoda ekstrahira podatke o datoteci, a to su izvorni naziv datoteke i ekstenzija. Zatim se generira jedinstveni naziv datoteke za pohranu, što osigurava sigurnost i nemogućnost spremanja datoteka s istim nazivom. Putanja pohrane stvorena je pomoću *Path.Combine()* u kojem je definirana putanja spremanja i naziv datoteke za pohranu. Na kraju je konstruiran objekt *UploadResult* koji sadrži potrebne informacije te se vraćaju klijentu kako bi ih mogao upisati u bazu, te dohvatiti u potrebnom dijelu.

Važan dio aplikacije je unos i prikaz matematičkih zadataka koje student rješava. Kako bi se omogućilo ispravno prikazivanje matematičkih izraza, potrebno je uključiti referencu na biblioteku MathJax:

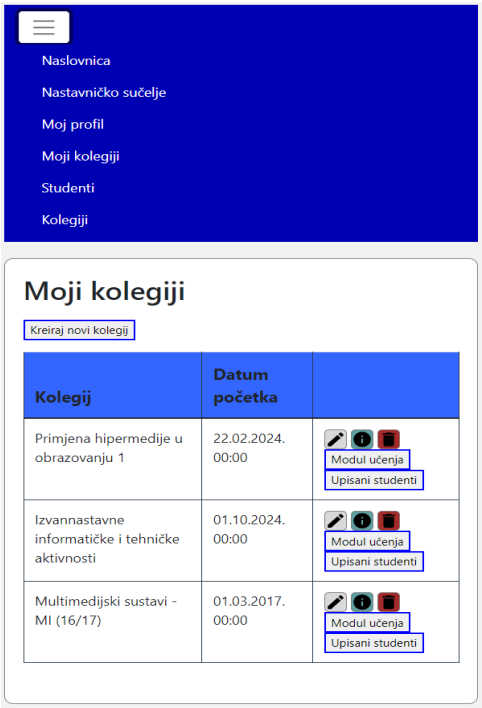
```
<script id="MathJax-script" async src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-chtml.js"></script>
```

Koristi se *async* atribut za asinkrono učitavanje skripte, omogućavajući tako pregledniku da nastavi analizirati i prikazivati stranicu dok se skripta dohvaća. S time je poboljšana performansa učitavanja stranica, posebno ako je datoteka skripte prevelika ili se nalazi na sporom poslužitelju.

Prilikom svake izrade aplikacije, potrebno je implementirati upravljanje pogreškama. U dijelu dohvata podataka iz baze, implementiran je blok *try-catch.* Unutar try bloka, kôd pokušava izvršiti upit nad bazom. U koliko upit nije uspješno izvršen, catch blok ispisuje poruku o iznimci u konzoli. Na taj način programeri mogu vidjeti poruku o pogrešci kako bi je mogli otkloniti.

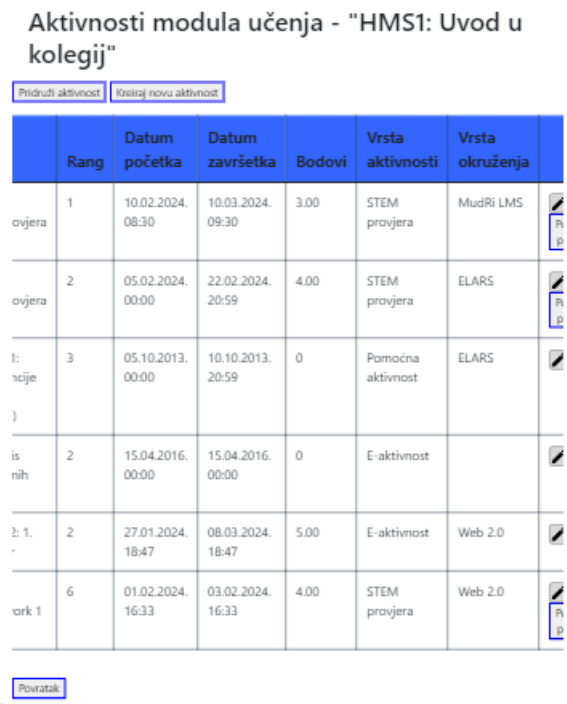
Kako sve više korisnika, web aplikacije koristi na mobilnim uređajima, dizajn ELARS aplikacije je napravljen tako da bude responzivan, odnosno prilagodljiv na manjim ekranima. Za to se koristio besplatni CSS okvir otvorenog kôda, Bootstrap koji sadrži predloške dizajna za navigaciju, gumbe, tablice i ostale komponente sučelja.

Bootstrap kod navigacije omogućava da se linkovi sakriju, a pritiskom na bijeli prozorćić označen s tri crtama otvara se padajući izbornik s pripadajućim linkovima na stranice, kao što je prikazano na slici 29.



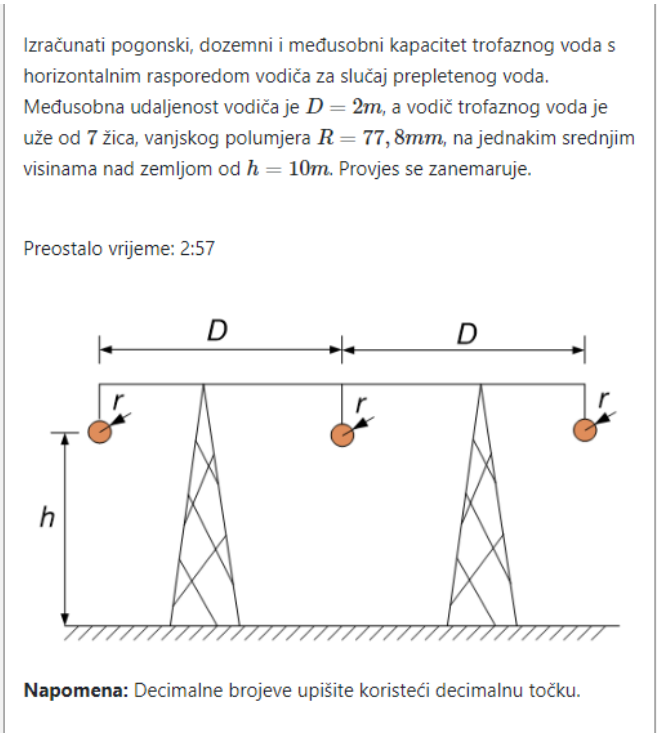
Slika 30 Prikaz responzivne navigacijske trake

U slučaju prikaza više podataka u tablici, spriječava se 'prelijavanja' tablica izvan okvira, tako što je omogućeno horizontalno skrolanje po tablici, kako bi korisnik mogao vidjeti sve podatke.



Slika 31 Prikaz responzivne tablice

Za prikaz slike korištena je HTML oznaka *figure* unutar koje slike nemaju eksplicitnu veličinu, s toga je za sliku dodana klasa *img-fluid* koja omogućava da se slika responzivno prikaže.



Slika 32 Responzivni način prikaza slike u zadatku

6. Zaključak

Svakim danom raste sve veća potreba za korištenje web aplikacija, pogotovo kod mlađih generacija koji sve rješavaju najviše putem mobilnog uređaja. S time raste i potreba za što bržim i lakim načinom izrade aplikacije. Tehnologije se stalno razvijaju i obnavljaju kako bi programerima omogućilo što bolju implementaciju kôda i testiranje same aplikacije. Upravo je Blazor primjer kako je pored uspješnog Razor-a, napredovao te olakšao i omogućio još bolji način izrade aplikacije. Nije potrebno pisati dugačak JavaScript kôd, nego ima gotove funkcionalnosti i komponente koje se mogu iskoristiti.

Prilikom kretanja u izradu aplikacije potrebno je specificirati i definirati korisničke zahtjeve, osmisliti dizajn, odabrati potrebnu tehnologiju u skladu sa svim navedenim zahtjevima, napraviti shemu podataka i stranica.

U ovom diplomskog radu objašnjenje su popularne tehnologije korištene u izradi same aplikacije, kako su se razvijale u skladu s potrebama i način na koji olakšavaju programerima oko same izrade. U praktičnom dijelu rada kreiran je početni dizajn aplikacije, omogućeno da aplikacija bude responzivna kako bi se mogla koristiti i na mobilnim uređajima. Zatim su napravljene potrebne tablice u bazi podataka za dohvaćanje, izmjenu, ubacivanje i brisanje podataka. Koristeći pripadne tehnologije napravljena je funkcionalnost aplikacije koju mogu koristiti nastavnici u studenti s dodijeljenim korisničkim imenom i lozinkom.

Naglasak ovog rada bilo je omogućiti potrebnu interakciju i funkcionalnost s korisnicima, dok se u sljedećoj fazi razvoja, aplikacija može poboljšati boljim dizajnom i dijelom koji bi omogućavao izračun i prikaz statistike ostvarenih rezultata svakog studenta.

7. Literatura

[1] What is a Web Application?, indeed. https://www.indeed.com/career-advice/career-development/what-is-web-application

[2] What is .NET Core?, C# Corner. https://www.c-sharpcorner.com/article/what-is-dot-net-core/

[3] What is .NET Core and everything you need to know about it, Hidden Brains. https://www.hiddenbrains.com/blog/what-is-net-core-and-everything-you-need-to-know-about-it.html

[4] List of top .NET Core features you need to discover right now, integrative systems. https://www.integrativesystems.com/features-of-dot-net-core/

[5] What's new in ASP.NET Core in .NET 6, Code magazine. https://www.codemag.com/Article/2111062/What%E2%80%99s-New-in-ASP.NET-Core-in-.NET-6

[6] What is Blazor?, Blazor University. https://blazor-university.com/overview/what-is-blazor/

[7] Razor vs Blazor, ironpdf. https://ironpdf.com/blog/net-help/razor-vs-blazor/

[8] Is Blazor the future of everything web?, Telerik. https://www.telerik.com/blogs/is-blazor-future-everything-web

[9] Blazor:switching Server and WebAssembly at runtime, Medium. https://itnext.io/blazor-switching-server-and-webassembly-at-runtime-d65c25fd4d8

[10] ASP.NET Core Blazor hosting models, Microsoft. https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-8.0

[11] Blazor Life Cycle Events, C# Corner. https://www.c-sharpcorner.com/article/blazor-life-cycle-events-oversimplified/

[12] Component lifecycles, Blazor University. https://blazor-university.com/components/component-lifecycles/

[13] What is Blazor, Pragim technologies. https://www.pragimtech.com/blog/blazor/what-is-blazor/

[14] Future of Web development: Constructing the Future Web, simlilearn. https://www.simplilearn.com/future-of-web-development-article

[15] When Wasm meets eBPF, eunomia. https://eunomia.dev/blogs/ebpf-wasm/

[16] ASP.NET Core Blazor layouts, Microsoft. https://learn.microsoft.com/en-us/aspnet/core/blazor/components/layouts?view=aspnetcore-8.0

8. Popis slika