WEB APLIKACIJA ZA STUDENTE FERIT-A

Dudjak, Kristina

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:207315

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-10**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

WEB APLIKACIJA ZA STUDENTE FERIT-A

Završni rad

Kristina Dudjak

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek,28.08.2021.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Kristina Dudjak
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4192, 24.07.2018.
OIB studenta:	70141670418
Mentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Web aplikacija za studente FERIT-a
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	28.08.2021.
Datum potvrde ocjene Odbora:	08.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTIRADA

Osijek, 27.09.2021.

Ime i prezime studenta:	Kristina Dudjak
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4192, 24.07.2018.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za studente FERIT-a**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Alfonzo Baumgartner

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Prosječni student Fakulteta elektrotehnike, računarstva i informacijskih tehnologija (FERIT) ima užurban i gust raspored kojeg je potrebno unaprijed isplanirati. Za planiranje dana obično se koristi neka vrsta dnevnika u kojeg se upisuju određene obaveze. Uz nastavu, ispite i ostale obaveze na fakultetu, studenti također mogu biti zaposleni preko studentskog centra i prehranjivati se u studentskim menzama. Kako bi se taj proces planiranja olakšao i digitalizirao, zadatak ovog završnog rada je izraditi aplikaciju koja prikazuje raspored, dnevne menije i raspoložive poslove preko Studentskog centra u Osijeku. FERITOrganizator je progresivna web aplikacija u kojoj je moguće unositi bilješke te putem push poruka dobivati obavijesti na mobilnom ili desktop uređaju. Korišteni programski okvir je Blazor koji kombinira uporabu programskih jezika C# i HTML. Blazor WebAssembly podržava izradu progresivne aplikacije koju korisnik može instalirati na računalu ili mobilnom uređaju.

U prvom sljedećem poglavlju upoznaje se sama Blazor platforma te njene značajke kao što su progresivna web aplikacija i push obavijesti te druge tehnologije koje su doprinijele izradi aplikacije.

U drugom poglavlju glavnog dijela rada opisan je postupak izrade aplikacije te je sadržaj unutar njega podijeljen u potpoglavlja za pregled izvoda svake stranice aplikacije.

Treće poglavlje omogućava uvid u sami izgled aplikacije i njene funkcionalnosti uz pružena kratka objašnjenja za svaku priloženu sliku.

Posljednje poglavlje predstavlja zaključak u kojem su opisani najvažniji postupci izrade aplikacije, postignuti postavljeni ciljevi, nedostatci te mogućnosti poboljšanja i unapređenja aplikacije.

1.1. Zadatak završnog rada

Zadatak završnog rada je napraviti progresivnu web aplikaciju koja će prikupljati studentski raspored sa službenog sustava Mrkve te dnevne menije i oglase za studentske poslove preko Studentskog centra Osijek. Aplikacija se također sastoji od unosa bilješki koje se spremaju u bazu podataka. Korištenjem Blazor WebAssembly programskog okruženja omogućeno je slanje push poruka na mobilni ili desktop uređaj.

2. KORIŠTENE TEHNOLOGIJE

U sljedećim poglavljima opisana je Blazor tehnologija u kojoj je izrađena aplikacija te različiti poslužiteljski modeli Blazor aplikacije. Uvodi se pojam progresivne aplikacije te njene značajke poput push obavijesti. Za bazu podataka korišten je SQL Server Management Studio. Syncfusion komponente korisne su za olakšani i ubrzani prikaz podataka. Selenium omogućuje korištenje WebDriver-a i web struganje za prikupljanje podataka.

2.1. Blazor

Microsoft je 2018. predstavio prvo izdanje Blazor-a, besplatne značajke otvorenog koda ASP.NET-a. ASP.NET je proširenje .NET platforme orijentirano na izradu dinamičkih web aplikacija [1]. Ime Blazor nastalo je kombinacijom riječi Razor i Browser, jer se Blazor sastoji od Razor stranica koje se mogu prikazati na browser-u (pregledniku). Web korisničko sučelje (UI - *User Interface*) kombinira programske jezike HTML, C# i CSS umjesto JavaScript jezika. Komponente se spremaju u reprezentaciju preglednikovog DOM (*Document Object Model*) elementa koji se koristi za ažuriranje korisničkog sučelja. Blazor je funkcionalan na svim modernim web i mobilnim preglednicima kao što su Google Chrome, Mozilla Firefox, Apple Safari te Microsoft Edge [2]. Blazor implementira .NET Standard koji omogućava korištenje raznih .NET biblioteka te dijeljenje logike pisane unutar klijenta i servera [1]. Postoje dva poslužiteljska modela, WebAssembly i Server koja će biti detaljnije objašnjena u sljedećim potpoglavljima.

2.1.1. Blazor Server

Blazor Server aplikacija se nalazi unutar ASP.NET aplikacije te se izvršava na serveru. On komunicira s klijentom šaljući DOM elemente putem SignalR veze [3]. SignalR je biblioteka koja pruža sučelje za programiranje aplikacija (API - *Application Programming Interface*) kojim se pojednostavljuje komunikacija između servera i klijenta korištenjem poziva udaljenih procedura (*RPC - Remote Procedure Calls*) poziva. Među njegovim najvažnijim značajkama su automatsko podržavanje veze, prilagodba povećanju prometa te istovremeno slanje poruka svim klijentima ili samo određenim grupama. Zbog toga je pogodan kod aplikacija koje zahtijevaju česta ažuriranja ili notifikacije (društvene mreže, GPS aplikacije, računalne igre i slično) [4]. Prije odlučivanja o određenom modelu, bitno je proučiti njegove prednosti i mane. Neke od prednosti Blazor Servera su:

- Veličina preuzimanja manja je nego kod WebAssembly aplikacije
- Brže učitavanje aplikacije
- Aplikacija iskorištava sve raspoložive kompatibilne API-jeve
- Radi s web preglednicima koji ne podržavaju WebAssembly.

S druge strane, među manama se izdvajaju sljedeće činjenice:

- Ne podržava izvanmrežni (*offline*) način rada, bez internetske veze dolazi do pada aplikacije
- Svakom korisnikovom interakcijom te povezivanjem s mrežom dolazi do usporavanja
- Otežana skalabilnost s velikim brojem korisnika
- Obavezan ASP.NET Core server za rad aplikacije [3].

2.1.2. Blazor WebAssembly

Blazor WebAssembly pokreće klijentov kod u internet pregledniku te on preuzima aplikaciju i .NET vrijeme izvođenja. Aplikacija, ažuriranja te rukovanje događajima se izvršavaju unutar preglednikove UI komponente. Dijelovi aplikacije preneseni su kao statičke datoteke do web servera ili usluge koji poslužuju statički sadržaj klijentima. Neke od korisnih funkcionalnosti su:

- Ne ovisi o .NET server strani, aplikacija radi nakon preuzimanja
- Klijentovi resursi i sposobnosti su posve iskorišteni
- Rad se odvaja iz klijentovog dijela u server
- Moguć rad aplikacije i bez ASP.NET Core servera.

Prisutna ograničenja Blazor WebAssembly aplikacija su:

- Aplikacija je ograničena sposobnostima internet preglednika
- Potrebno sposobno klijentovo sklopovlje (*hardware*) i programska podrška (*software*)
- Veličina preuzimanja je veća od Blazor Server aplikacije
- Duže učitavanje aplikacije [3].

Blazor WebAssembly aplikacija bez *backend* aplikacije za predavanje dokumenata se zove samostalna (*standalone*) WebAssembly aplikacija. Ako se koristi *backend* aplikacija za držanje podataka, radi se o *hosted* WebAssembly aplikaciji.

2.1.3. Blazor ASP.NET Core hosted

Za ovu aplikaciju izabran je ovaj model Blazor aplikacije koji se sastoji od tri projekta: Klijent projekt, Server projekt i Dijeljeni projekt. Unutar klijentovog projekta nalazi se logika vezana uz klijenta i razor stranice, unutar server projekta se obavlja glavna logika aplikacije, a u dijeljenom projektu se mogu nalaziti klase korištene u oba projekta. Klijentov dio aplikacije šalje se pregledniku te se povezuje sa serverom putem SignalR veze ili web API poziva. Prednosti ASP.NET Core *hosted* modela:

- .NET Core omogućava iskorištavanje ponuđenih mogućnosti za usmjeravanje i kompresiju
- Ista tehnologija koristi se za *backend* i *frontend*
- Pogodno za korištenje baze podataka jer web API obavlja operacije unutar baze

Mane spomenutog modela su:

- Ne podržava izvanmrežni (*offline*) način rada
- DOM ažuriranja obavljaju se na ASP.NET Core serveru [5].

2.2. Progresivna web aplikacija

Progresivna web aplikacija obično je aplikacija koja ima jednu stranicu te koristi mogućnosti i API-jeve modernih web preglednika kako bi se ponašala kao desktop aplikacija. Riječ progresivna označava razvoj od klijentovog otkrića aplikacije na web pregledniku do kasnije instalacije na desktop. Podržava izvanmrežni (*offline*) način rada te se može pokrenuti neovisno o brzini interneta. Nakon instalacije, može se pokrenuti u svom prozoru aplikacije pri čemu više ne ovisi o pokretanju unutar preglednikovog prozora [6]. Velika prednost progresivnih aplikacija je mogućnost slanja push obavijesti pri čemu povećavaju klijentovu angažiranost pri korištenju aplikacije.

2.2.1. Progresivna vs. nativna aplikacija

Uvođenjem nove tehnologije progresivnih web aplikacija dolazimo do dvojbe između korištenja nativnih ili progresivnih aplikacija. Nativne aplikacije obično su pisane u programskim jezicima Java (Android) ili Swift (iOS), dok PWA (Progressive Web Application) koristi HTML, CSS i JavaScript. PWA aplikacije znaju biti jeftinije i efikasnije za izradu od nativnih, jer ne zahtijevaju pisanje i ažuriranje različitih verzija aplikacije za različite operacijske sustave i App trgovine. Potrebna je samo jedna verzija aplikacije koja zbog svoje responzivnosti omogućava

kvalitetan prikaz na svim uređajima. Progresivne aplikacije imaju veću razinu sigurnosti jer moraju biti razvijene preko protokola za prijenos hiperteksta (HTTPS - *Hypertext Transfer Protocol Secure)* koji omogućava da ne dođe do razmjene povjerljivih podataka između klijenta i servera. Iako se progresivne aplikacije čine kao sigurniji izbor, korisnici nerijetko više vjeruju aplikacijama nego jedinstvenim lokatorima resursa (URL - *Universal Resource Locator*) jer prije objave na App trgovini moraju proći sigurnosne mjere. Nativne aplikacije također mogu ugraditi i dodatne sigurnosne mjere kao što je višestruka provjera autentičnosti. Zbog većeg broja potrebnih koraka i dužeg vremena, korisnici se rijetko odlučuju na instaliranje aplikacija. Uz samo par klikova, korisnici progresivnu aplikaciju mogu dodati na početni zaslon bez čekanja instalacije i velike količine zauzete memorije. Progresivne aplikacije pokreću se iz preglednika što rezultira većom potrošnjom baterije uređaja i mogućim usporavanjem aplikacije. Iza svake progresivne aplikacije nalaze se *Service workers*, skripte odvojene od aplikacije koje upravljaju izvanmrežnim (*offline*) načinom rada, dohvaćanjem podataka, spremanjem resursa i slično. Nativne aplikacije nalaze se u operacijskom sustavu uređaja te mogu izvoditi kalkulacije za pružanje optimalnog, boljeg iskustva korištenja aplikacije. Između ostalog, imaju i veći izbor dodatnih mogućnosti, kao što su interakcija s ostalim aplikacijama, lakše mobilno plaćanje te korištenje pametnog ključa, senzora i drugih alata mobilnog uređaja.

Obje vrste aplikacija imaju svoje prednosti i nedostatke, pa izbor među njima ovisi o samoj vrsti i biti aplikacije. Nativne aplikacije su dobar izbor za razvoj aplikacija koje koriste razne značajke uređaja, ali i za izgradnju kredibiliteta te pouzdanja mogućih klijenata zbog sigurnosti objavljenih aplikacija. Progresivne aplikacije su jednostavniji, brži i jeftiniji način izrade aplikacija te vode brigu o interakciji s korisnicima slanjem push obavijesti [7].

2.2.2. Push obavijesti

Push notifikacije su *pop-up* klikabilne poruke koje se mogu pojaviti na mobilnim uređajima i dok korisnik ne koristi aplikaciju. Prvim korištenjem aplikacije obično iskoči mali prozorčić kojim korisnik može prihvatiti ili odbiti primanje push notifikacija. Prihvaćanjem opcije, korisnici se dodaju u poseban popis pretplatnika. Samim time omogućeno im je da dobivaju posebne poruke različitih popusta, novog sadržaja i ostalih promjena u aplikaciji. Također, ako korisnici nisu dugo otvorili ili koristili aplikaciju, push notifikacijom aplikacija ih može podsjetiti i pozvati natrag. Međutim, pri razvoju notifikacija važno je zapamtiti da prečestim slanjem poruka mogu dosaditi korisniku te smanjiti njihov interes te time uzrokovati deinstalaciju aplikacije.

2.3. SQL Server Management Studio (SSMS)

Za bazu podataka korišten je SQL Server Management Studio (SSMS). SSMS je integrirano okružje za upravljanje SQL infrastrukturama, od SQL Server do Azure SQL baze podataka. SSMS pruža alate za konfiguraciju, praćenje i upravljanje instancama SQL baze podataka. Koristi se za upite, dizajn i upravljanje baze podataka s lokalnog računala ili oblaka [8]. Microsoft SQL Server (MS SQL Server) je relacijski sustav baze podataka kojeg mogu koristiti razni aplikacijski sustavi. U ovoj aplikaciji korištena je najnovija verzija Microsoft SQL Server 2019, izdana u studenom 2019. godine.

2.4. Syncfusion

Syncfusion je poznati pružatelj bogatog izbora UI komponenti i biblioteka za razvoj web, mobilnih i desktop aplikacija. Syncfusion pruža besplatne licence i tehničku podršku malim poslovnim poduzećima i individualnim programerima. Postoji preko 1600 UI komponenti za .NET, JavaScript, Windows Forms, Angular, Vue, React, Blazor, itd [9]. Uz dokumentaciju, postoje video objašnjenja korištenja svake podržane Blazor komponente. U ovoj aplikaciji korištena je *DataGrid* komponenta za prikaz i unos bilješki. Uz izvođenje potrebnih operacija za stvaranje, čitanje, ažuriranje i brisanje (CRUD - *Create, Read, Update, Delete*), omogućava učitavanje velikog broja podataka u tabularnom formatu u kratkom vremenu. Ima responzivni dizajn koji se prikladno prilagođava desktop ili mobilnom uređaju [10]. Sljedeća korištena značajka je *Scheduler* koja prikazuje raspored nastave i ispita studenata. *Scheduler* je također definirana responzivna komponenta te omogućava lak uvid u dnevni, tjedni, mjesečni raspored te lako dodavanje i upravljanje elementima rasporeda. Kako bi se određeni raspored prikazao, studenti prvo biraju određeni smjer i trenutni polazeći semestar preko *Dropdown List* elementa. *Dropdown* je brža varijanta HTML *select tag* elementima [11]. Radi bolje preglednosti, korišteno je i grupiranje elemenata u kategorije. Zadnji korišteni element iz Syncfusion biblioteke je *Button* koji korisnikovim klikom okida web API poziv prikupljanja rasporeda i njegovog prikaza u *Scheduler*.

2.5. Selenium

Selenium je neprofitna organizacija koja pruža paket alata za automatiziranje web preglednika. Selenium Project formirana je zajednicom korisnika i suradnika otvorenog koda koji razvijaju, koriste i promoviraju Selenium projekte kao što su IDE, Grid i WebDriver. WebDriver je sučelje

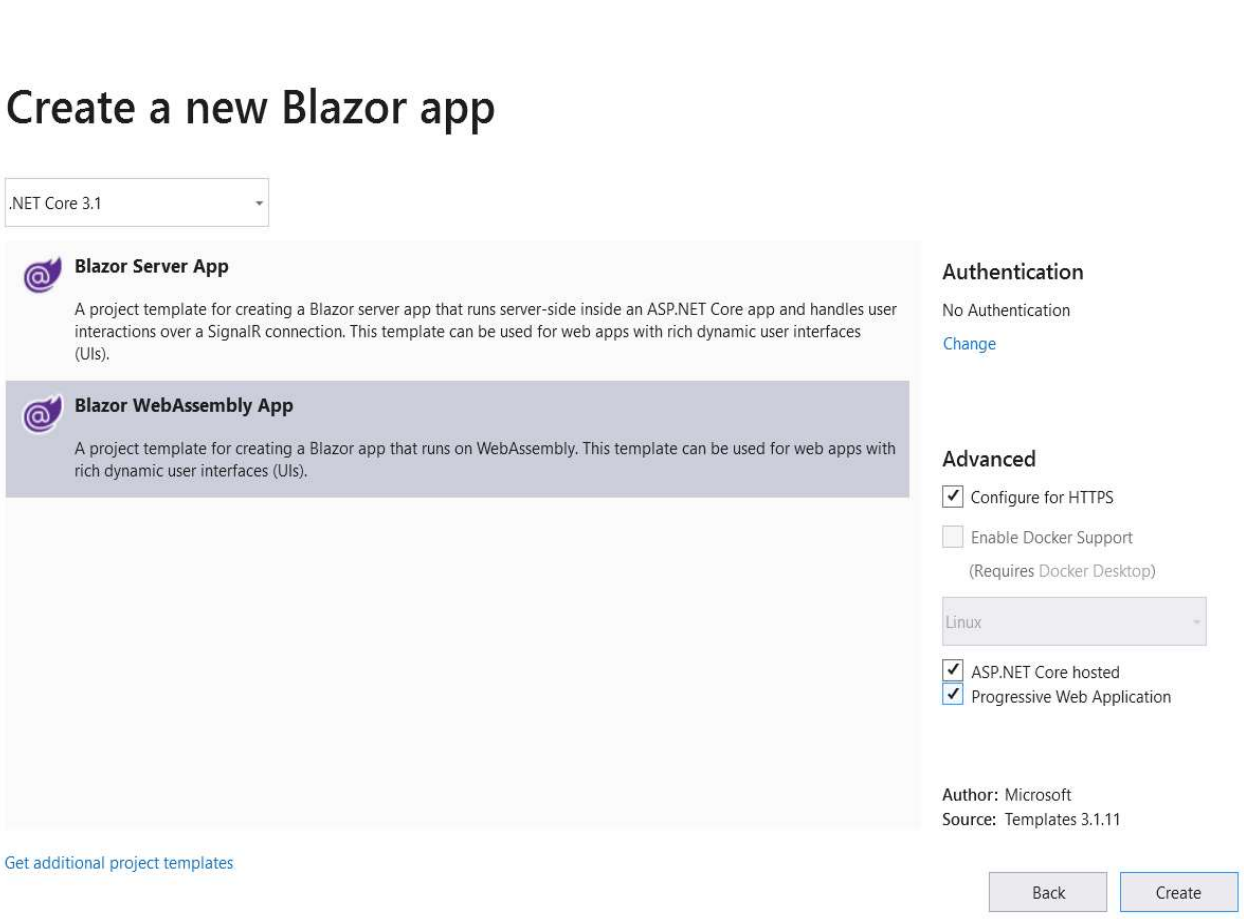
koje otkriva i manipulira DOM elementima unutar web dokumenata. Primarno se koristi kao dozvola pisanja testova za automatizaciju, ali također se može koristiti za dozvolu skriptama web preglednika kontroliranje odvojenog web preglednika [12]. Razlog korištenja Seleniuma u ovoj aplikaciji je jer podržava web struganje uz pomoć ChromeDriver-a.

2.5.1. Web struganje

Web struganje (*web scraping*) je automatizirana tehnika izvlačenja podataka iz obrađenog HTML-a web stranica, za razliku od screen struganja koje kopira samo piksele prikazane na ekranu. Prikladna je kod web stranica koje nemaju ponuđeni API za prikupljanje podataka. Ključan dio web struganja su selektori koji pronalaze podatke preko HTML putanje, CSS selektora, imena klase, oznake ili ID-a. Velika prednost Selenium web strugala u odnosu na ostale je njegova mogućnost pokretanja JavaScript koda web stranica. Neke stranice, poput web stranice studentskog centra u Osijeku, koriste JavaScript za odrađivanje određenih poslova unutar stranice, čime nije odmah vidljiv sav HTML kod. Međutim, pokretanje skripte usporava proces prikupljanja podataka u usporedbi s običnim HTTP (*Hypertext Transfer Protocol*) zahtjevom web pregledniku [13].

3. IZRADA RADA

Kako je ideja ovog rada napraviti progresivnu aplikaciju, Blazor WebAssembly je najbolji izbor. Prilikom kreiranja projekta, u desnom uglu prozora može se označiti kutijica "*Progressive Web Application*". Prema slici 3.1., radi preglednije strukture i korištenja usluga server dijela aplikacije, također je označena i kutijica ASP.NET Core *hosted*.



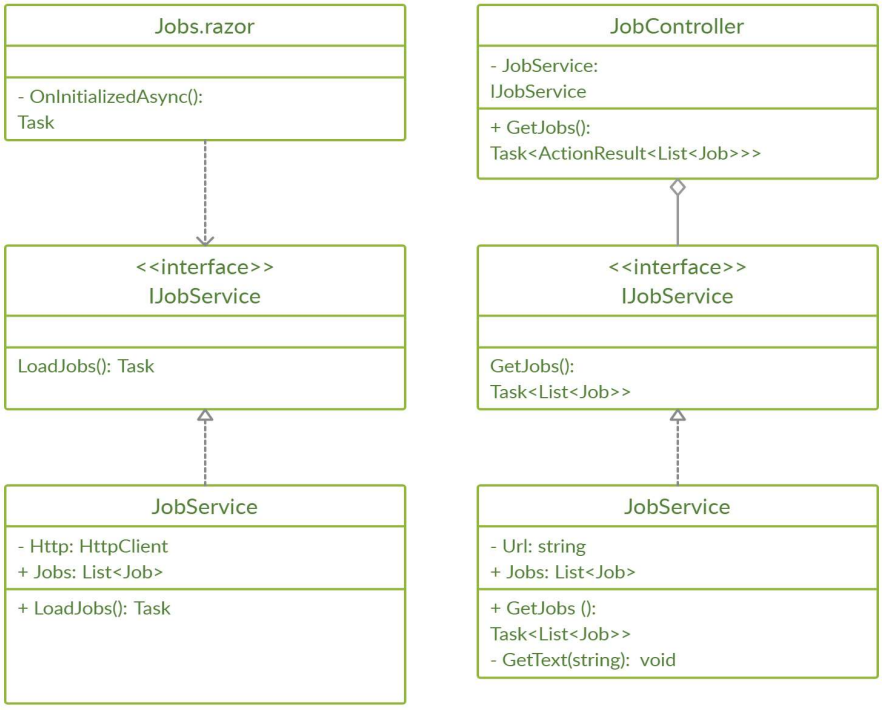
Slika 3.1. Prikaz kreiranja Blazor WebAssembly aplikacije

3.1. Struktura aplikacije

Nakon stvaranja aplikacije, možemo primijetiti da se *solution* sastoji od tri projekta. Unutar *.Client* projekta nalazi se logika stvaranja UI komponenti koje će biti prikazane u aplikaciji. Sve prikazane stranice su Razor komponente smještene u *Pages* mapi, dok se u *Shared* mapi nalazi *MainLayout.razor* u kojoj je definiran glavni izgled aplikacije te *NavMenu.razor* koji predstavlja navigacijsku traku s lijeve strane unutar aplikacije. Pošto je Blazor *Single Page Application*

(SPA), uvijek je prikazana i aktivna samo jedna stranica. Klikom svake stavke na navigacijskoj traci otvara se druga stranica aplikacije. U *.Server* projektu obavlja se server dio koda, kao što su operacije vezane uz bazu podataka i Web API pozivi. Krucijalna stavka *.Server* projekta je C*ontroller* mapa unutar koje se nalaze svi korišteni kontroleri. Visual Studio u ovom slučaju nudi stvaranje API ili MVC kontrolera koji mogu biti prazni, s CRUD operacijama te s CRUD operacijama korištenjem podataka iz Entity Framework-a. Kontroler iz MVC (M*odel-View-Controller*) je klasa koja obrađuje dolazeće URL zahtjeve te je izvedena iz *System.Web.Mvc.Controller* klase. Web API kontroler klasa obrađuje nadolazeće HTTP zahtjeve te je izvedena iz *System.Web.Http.ApiController* klase [14]. Zadnji projekt je *.Shared* projekt u kojem se nalazi kod koji se dijeli i koristi i u server i klijent projektu, kao što su modeli klasa.

Jedan od načina strukturiranja Blazor projekta je korištenje usluga, tj. *services*. Unutar mape *Services* dodaju se ostale mape, gdje svaka predstavlja jednu uslugu. Klijent projekt koristi mapu usluga kako bi iz Razor komponente izdvojio pozivanje kontrolera iz server projekta. Server projekt koristi drugu mapu usluga za obavljanje glavnog posla te vraćanje rezultata kontroleru. Svaka pojedina mapa usluga sastoji se od sučelja i klasa koje ga implementiraju. Tako je moguće lako dodavanje načina obavljanja određenog posla i njihovo izmjenjivanje. Način povezivanja klijent projekta i server projekta je pozivanjem API kontrolera iz *Service* klase u klijent projektu. Na slici 3.2. prikazan je primjer korištenja *JobService* klasa.



Slika 3.2. Primjer strukture klasa

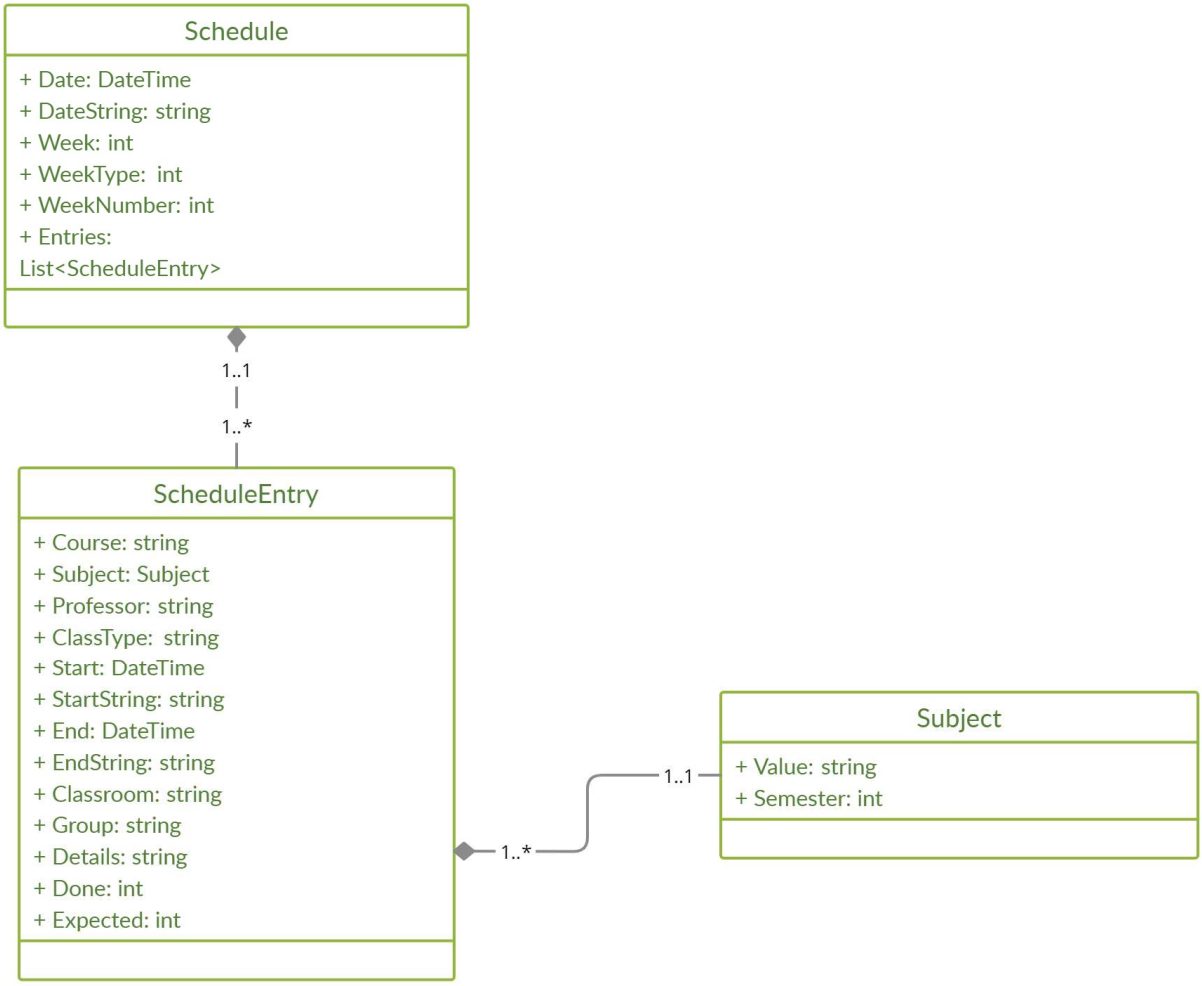
3.2. Raspored

Jedno od svojstava aplikacije je mogućnost odabira i prikaza studentovog rasporeda i ispita. Službena stranica FERIT-a koristi sustav Mrkve za planiranje, izradu i evidenciju rasporeda nastave i ispita. Također sadrži i API s potrebnim informacijama kao što su predmeti, prostorije, raspored te studenti. Postupak dohvaćanja odrađuje se u metodi *GetScheduleAsync* koja iz primljenog URL-a sastavlja i vraća objekt *Schedule* klase. Instancom *HttpClient* klase šalje se HTTP zahtjev stranici za pristup podataka. Nazad se vraća odgovor u XML (Extensible Markup Language) formatu. XML je dinamičan programski jezik te se koristi za prijenos podataka, za razliku od HTML-a koji se koristi za prikaz podataka. Njegov jednostavan dizajn i podrška Unicode-a (*Universal character encoding standard*) omogućava uporabu različitih jezika [15]. Nakon toga, koristi se međuspremnik (*buffer*) koji odgovor pisan u Windows-1250 formatu pretvara u niz bajtova. Koristeći *MemoryStream* klasu, stvaramo novi *XmlSerializer* koji serijalizira objekt tipa *Schedule*. Serijalizacija je postupak pretvaranja objekta u niz bajtova i spremanje u datoteku. Konačno, kreirani *deserializer* čita iz datoteke, rekreira te vraća *Schedule* objekt. Opisana metoda prikazana je na slici 3.3.

```
private async Task<Schedule> GetScheduleAsync(string url)
{
    Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
    HttpResponseMessage response = http.GetAsync(url).Result;
    var xmlString = response.Content.ReadAsStringAsync().Result;
    var buffer = Encoding.GetEncoding("Windows-1250").GetBytes(xmlString);

    using (var stream = new MemoryStream(buffer))
    {
        var serializer = new XmlSerializer(typeof(Schedule));
        return (Schedule)serializer.Deserialize(stream);
    }
}
```

Slika 3.3. Metoda *GetScheduleAsync*

Kako bi se podaci s API-ja pravilno preuzeli, potrebno je napraviti klasu sa svojstvima koji predstavljaju željene resurse.



Slika 3.4. UML model klasa *Schedule*, *ScheduleEntry* i *Subject*

Prema slici 3.4., klasa *Schedule* predstavlja model rasporeda u koju će se unositi potrebni podaci iz sustava Mrkve. Unutar njega je XML atribut imena *Date* koji predstavlja datum rasporeda te je namješten da poštuje određeni DateTime format. Nadalje, XML atribut *Week* je broj tjedna u godini, a *WeekNumber* je redni broj nastavnog (mogući raspon od 1 do 15) ili ispitnog tjedna (mogući raspon od 1 do 5) koji ima nedodijeljenu vrijednost preko praznika. Element *WeekType* poprima vrijednost 1 u slučaju nastavnog tjedna, vrijednost 2 u slučaju ispitnog tjedna i nedodijeljenu vrijednost u slučaju praznika. Raspored se sastoji od brojnih stavki rasporeda tijekom tog dana zbog čega je napravljena lista svih stavki koje su izdvojene u novi model klase.

Klasa *ScheduleEntry* sadrži elemente *Course* za smjer studenta, *Professor* za profesora koji održava predmet, *ClassType* koja može predstavljati predavanja, auditorne vježbe, laboratorijske vježbe, ispitni rok i nepoznato za ništa od navedenog. Elementi *Start* i *End* su namješteni kako

bi poštivali odabrani format koji predstavlja početak i kraj stavke rasporeda. Također se može saznati prostorija (*Classroom*) u kojoj se nalazi, grupa studenata (*Group*) kojoj se održava nastava, koliko nastave je odrađeno (*Done*) od planiranog (*Expected*) te dodatni opis (*Details*) stavke.

Subject drži podatke o predmetu (*Value*) i polazećem semestru (*Semester*) studenta.

Schedule.razor komponenta sastoji se od dijela u kojem se piše HTML kod za prikaz te od *@code* dijela u kojem je C# kod s logikom iza prikaza. Na sljedećoj slici prikazan je HTML dio stranice.

```
@page "/"
@page "/schedule"
@using FERITOrganizator.Shared.Models
@using Syncfusion.Blazor.DropDowns
@inject IScheduleService ScheduleService
@inject IJSRuntime JsRuntime


<SfDropDownList TItem="CourseData" TValue="string" DataSource="Courses" Placeholder="Odaberi smjer" @bind-Value="@smjer">
    <DropDownListFieldSettings GroupBy="Category" Text="CourseId" Value="Name"></DropDownListFieldSettings>
</SfDropDownList>
<br />
<br />

<SfDropDownList TItem="SemesterData" TValue="int" DataSource="Semesters" Placeholder="Odaberi semestar" @bind-Value="@semestar">
    <DropDownListFieldSettings GroupBy="Category" Text="Number" Value="Number"></DropDownListFieldSettings>
</SfDropDownList>
<br />
<br />
<SfButton @onclick="GetSchedule" CssClass="e-primary">Prikaži</SfButton>
<br />
<br />


<SfSchedule TValue="ScheduleForm" SelectedDate="@(new DateTime(2021, 4, 12))" CurrentView="View.Week" FirstDayOfWeek="1">
    <ScheduleViews>
        @if (Width > 1024)
        {
            <ScheduleView Option="View.Day" StartHour="08:00" EndHour="22:00"></ScheduleView>
            <ScheduleView Option="View.Week" StartHour="08:00" EndHour="22:00"></ScheduleView>
        }
        else
        {
            <ScheduleView Option="View.Agenda" StartHour="07:00" EndHour="22:00"></ScheduleView>
        }
    </ScheduleViews>
    <ScheduleResources>
        <ScheduleResource TItem="ResourceDataModel" TValue="int" DataSource="@ResourceData"
                          Field="ResourceId" TextField="Type" IdField="ResourceId" ColorField="Color">
        </ScheduleResource>
    </ScheduleResources>
    <ScheduleEventSettings DataSource="@schedules"></ScheduleEventSettings>
</SfSchedule>
```

Slika 3.5. HTML kod *Schedule.razor* komponente

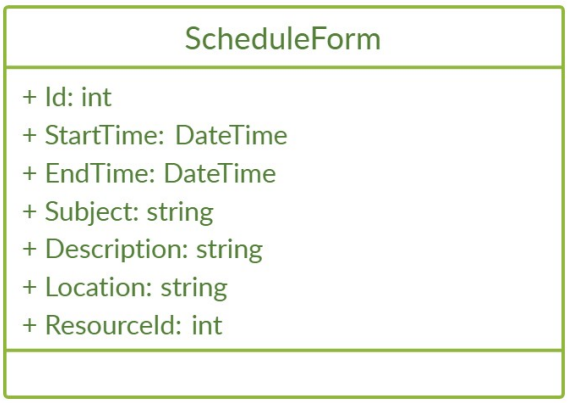
U gornjem dijelu nalazi se *using* dio stranice u kojem uključujemo potrebne biblioteke ili klase. Za prikaz svih elemenata korištena je Syncfusion biblioteka. *SfDropDownList* koristimo za *Dropdown* listu kojom se bira smjer te semestar studenta. Potrebno je definirati *TItem* parametar koji predstavlja objekt klase koji sadrži željeni parametar. *Text* prima parametar koji će biti prikazan, dok je odgovarajući *Value* parametar odabran ovisno o kliku korisnika. *DataSource* je lista *TItem* elemenata, a *bind-Value* metoda sprema *Value* vrijednost u dani parametar. To je potrebno kako bi se odabrani smjer i semestar studenta mogli spremiti u varijablu koju koristi metoda kako bi znala koje stavke rasporeda prikazati. Slika 3.6. prikazuje klasu *SemesterData* korištenu kao *TItem dropdown* komponente, ali i primjer potrebnog popunjavanja liste objektima spomenute klase. Uvođenjem *Category* parametra grupiramo semestre ovisno kojoj akademskoj godini pripadaju. Time je prikaz *dropdown* liste uređeniji i pregledniji.

```
public class SemesterData
{
    public string Category { get; set; }
    public int Number { get; set; }
}
List<SemesterData> Semesters = new List<SemesterData>
{
    new SemesterData(){ Category = "1. godina", Number = 1},
    new SemesterData(){ Category = "1. godina", Number = 2},
    new SemesterData(){ Category = "2. godina", Number = 3},
    new SemesterData(){ Category = "2. godina", Number = 4},
    new SemesterData(){ Category = "3. godina", Number = 5},
    new SemesterData(){ Category = "3. godina", Number = 6}
};
```

Slika 3.6. Klasa *SemesterData* i popunjena lista *Semesters*

Nakon izbora smjera i semestra, student stišće gumb koji okida *onclick* metodu *GetSchedule*. *GetSchedule* metoda najprije dohvaća dimenzije korisnikovog ekrana pa uz *await* naredbu čeka da se učita i kreira raspored, što se događa u server dijelu projekta. Zatim slijedi filtriranje rasporeda ovisno o smjeru i semestru, te se ispravne stavke rasporeda spremaju u listu *ScheduleForm* objekata koji su kreirani kako bi odgovarali i mogli biti prikazani u *SfScheduler*-u.

SfSchedule ima definirane nazive parametara objekta kojeg moraju poštivati, zbog čega je potrebno stvarati objekte pripadajućih imena i slagati im vrijednosti uz pomoć *Schedule* objekta. Model *ScheduleForm* klase prikazan je na slici 3.7.



Slika 3.7. UML klasa *ScheduleForm*

Omogućena je laka personalizacija rasporeda u smislu odabira prikaza mjeseca, tjedna i dana, postavljanje početnog dana u tjednu, početnog i završnog sata rasporeda. Ovisno o širini korisnikovog ekrana, odabrane su opcije prikaza rasporeda. Ako je širina ekrana veća od 1024, raspored će biti prikazan u normalnom obliku tjednog rasporeda u kojem su dani poredani s lijeva na desno, dok su kod manje širine ekrana dani poredani odozgo prema dolje. Kako bi se elementi rasporeda lako razlikovali jedni od drugih, definiran je *ScheduleResources* element koji koristi *ResourceData* listu *ResorceDataModel* objekata, vidljivo na slici 3.8. U njoj se nalaze parametri *Type* koji predstavlja različite oblike nastave (predavanja, auditorne vježbe, laboratorijske vježbe i sl.) te *Color* za pripadajuću boju koja ga predstavlja.

```
public class ResourceDataModel
{
    public int ResourceId { get; set; }
    public string Type { get; set; }
    public string Color { get; set; }
}
List<ResourceDataModel> ResourceData = new List<ResourceDataModel>
{
    new ResourceDataModel { ResourceId = 1, Type= "AV - auditorne vježbe", Color = "#ea7a57"},
    new ResourceDataModel{ResourceId = 2, Type="PR - predavanja", Color = "#865fcf"},
    new ResourceDataModel { ResourceId = 3, Type = "LV - laboratorijske vježbe", Color = "#df5286"},
    new ResourceDataModel { ResourceId = 4, Type = "Nepoznato", Color = "#c9c926"},
    new ResourceDataModel { ResourceId = 5, Type = "ISP - ispitni rok", Color="#c23725"}
};
```

Slika 3.8. Klasa *ResourceDataModel* i popunjena lista *ResourceData*

3.3. Bilješke

Kako bi studenti mogli upisivati svoje bilješke i kako bi mogle ostati zapamćene, potrebno ih je povezati s bazom podataka. Korištena je MS SQL Server baza podataka koja koristi Entity Framework Core. Entity Framework (EF) Core je proširiva verzija EF tehnologije za pristup podacima te omogućava rad s bazom podataka koristeći .NET objekte. Pristup podacima odvija se kroz model kojeg čine *Entity* klasa i *Context* objekt [16]. Na slici 3.9. prikazani su elementi tablice *Note* koja sadrži *Id* za razlikovanje bilješki, *Name* za kontekst bilješke te *Due* koji predstavlja datum i vrijeme potrebnog obavljanja bilješke.

	NoteId	Name	Due
1	10217	Provjeriti e-mail poruke	2021-07-30 08:00:00.000
2	10218	Sastanak	2021-07-30 14:00:00.000
3	10219	Završiti seminar	2021-08-06 00:00:00.000

Slika 3.9. Popunjeni stupci *Note* klase u bazi podataka

Prije povezivanja tablice s našim projektom, potrebno je instalirati određene biblioteke, *Microsoft.EntityFrameworkCore.Tools* koji stvara kontekst i modele klase iz baze te *Microsoft.EntityFrameworkCore.SqlServer* koji dozvoljava rad sa SQL serverom. Nakon toga se unutar *Package Manager Console* upisuje naredba koja povezuje bazu s projektom. Unutar njega stvara se *Note* klasa, ali i *OrganizatorContext* klasa koja implementira *DbContext* klasu. Prema slici 3.10. vidimo da klasa *OrganizatorContext* implementira *OnModelCreating* metodu koja prima *ModelBuilder* parametar. On definira izgled *entity*-ja te veze među njima.

```
using FERITOrganizator.Shared.Models;
using Microsoft.EntityFrameworkCore;

namespace FERITOrganizator.Server.Models
{
    6 references
    public partial class OrganizatorContext : DbContext
    {
        8 references | 0 exceptions
        public virtual DbSet<Note> Notes { get; set; }

        0 references | 0 exceptions
        public OrganizatorContext(){}

        0 references | 0 exceptions
        public OrganizatorContext(DbContextOptions<OrganizatorContext> options)
            : base(options){}

        0 references | 0 exceptions
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Note>(entity =>
            {
                entity.Property(e => e.Due).HasColumnType("datetime");

                entity.Property(e => e.Name)
                    .IsRequired()
                    .HasMaxLength(80)
                    .IsUnicode(false);

            });
            OnModelCreatingPartial(modelBuilder);
        }

        1 reference | 0 exceptions
        partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
    }
}
```

Slika 3.10. Klasa *OrganizatorContext*

Kako bi se podaci iz baze mogli upotrebljavati, potreban je Web API kontroler koji predaje podatke iz *DbContext*-a u Blazor aplikaciju pošto direktna veza s bazom u njoj nije moguća. U kontroleru sa slike 3.11. definiran je kod za upravljanje CRUD operacijama. Iznad svake metode definiran je odgovarajući Http atribut (*HttpGet*, *HttpPost*, *HttpPut* i *HttpDelete*). U *Get* metodi dohvaćamo bilješke, u *Post* metodi ih dodajemo, u *Put* metodi bilješki pridajemo određene vrijednosti te u *Delete* metodi uklanjamo bilješku iz liste.

```
// GET api/<NotesController>/5
[HttpGet("{id}")]
0 references | 0 requests | 0 exceptions
public object Get(long id)
{
    return new { Items = db.Notes.Where(x => x.NoteId.Equals(id)).FirstOrDefault(), Count = db.Notes.Count() };
}

// POST api/<NotesController>
[HttpPost]
0 references | 0 requests | 0 exceptions
public async Task PostAsync([FromBody] Note note)
{
    db.Notes.Add(note);
    db.SaveChanges();
    SendNotificationAsync(note, "Bilješka uspješno dodana!");
}

// PUT api/<NotesController>/5
[HttpPut("{id}")]
0 references | 0 requests | 0 exceptions
public void Put(long id, [FromBody] Note note)
{
    Note note1 = db.Notes.Where(x => x.NoteId.Equals(id)).FirstOrDefault();
    note1.Name = note.Name;
    note1.Due = note.Due;
    db.SaveChanges();
}
// DELETE api/<NotesController>/5
[HttpDelete("{id}")]
0 references | 0 requests | 0 exceptions
public void Delete(long id)
{
    Note note1 = db.Notes.Where(x => x.NoteId.Equals(id)).FirstOrDefault();
    db.Notes.Remove(note1);
    db.SaveChanges();
}
```

Slika 3.11. CRUD operacije *NotesController*-a

Jedino je *POST* api metoda asinkrona te nakon spremanja promjena unutar baze poziva metodu unutar koje se definiraju potrebni parametri za slanje push poruke, a to su privatni ključ, javni ključ te pretplata. Prihvaćanjem primanja obavijesti korisnik se dodaje u bazu pod tablicom *NotificationSubscriptions*, koja sprema detalje korisnika poput njegovog *Url*-a, *P256dh* ključa te *Auth* ključa. Za samo slanje push poruke koristi se *WebPush* biblioteka koja omogućava stvaranja klijenta koji prima pretplatu, poruku te detalje poput privatnog i javnog ključa.

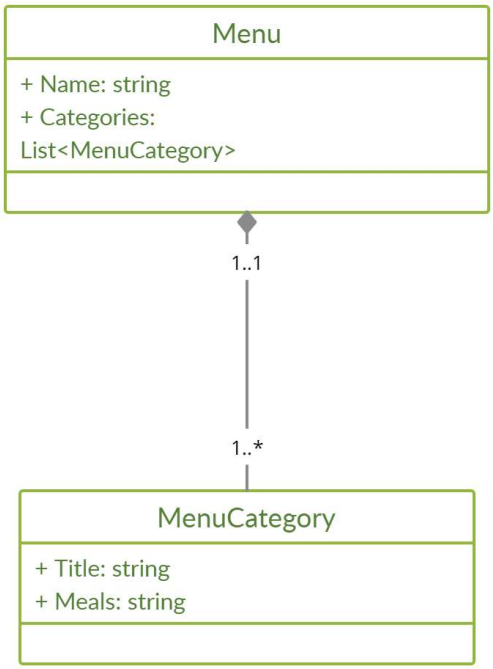
Za prikaz bilješki koristi se Syncfusion Grid (*SfGrid*) sa slike 3.12., element kojem je *TValue Note* objekt. Definiran je *Toolbar* u kojem je moguće dodavanje bilješki i odustajanje. Koristi se *SfDataManager* u kojem je definiran Url do kontrolera bilješki te *GridEditSettings* u kojem je omogućeno dodavanje i brisanje stavki. Svaki stupac definiran je *GridColumn* naredbom uz koju se predaje *Field* element koji predstavlja *Id* bilješke, ime ili datum. *Id* se ne prikazuje u tablici, dok je ime obavezno polje koje je potrebno ispuniti. Datum je uređen da se prikazuje u određenom formatu. Unutar *GridCommandColumn* stavki definirani su gumbi za dodavanje, brisanje i odustajanje, njihov stil te ikone. Dodavanjem nove bilješke korisniku iskače poruka o uspješnom dodavanju bilješke.

```
@page "/notes"
@using FERITOrganizator.Shared.Models
@using Syncfusion.Blazor.Grids
<meta charset=utf-8 />

<SfGrid TValue="Note" Toolbar="@(new List<string>() { "Add", "Cancel" })">
    <SfDataManager Url="/api/Notes" Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
    <GridEditSettings AllowAdding="true" AllowDeleting="true"></GridEditSettings>
    <GridColumns>
        <GridColumn Field="@nameof(Note.NoteId)" IsPrimaryKey="true" Visible="false"></GridColumn>
        <GridColumn Field="@nameof(Note.Name)" HeaderText="Naslov" TextAlign="TextAlign.Left"
                    ValidationRules="@(new Syncfusion.Blazor.Grids.ValidationRules(){Required=true})"></GridColumn>
        <GridColumn Field="@nameof(Note.Due)" HeaderText="Datum i vrijeme"
                    EditType="EditType.DateTimePickerEdit" Format="dd.MM.yyyy HH:mm"></GridColumn>
        <GridColumn>
            <GridCommandColumns>
                <GridCommandColumn Type="CommandButtonType.Delete"
                                   ButtonOption="@(new CommandButtonOptions()
                                                   { IconCss = "e-icons e-delete", CssClass = "e-flat" })"></GridCommandColumn>
                <GridCommandColumn Type="CommandButtonType.Save"
                                   ButtonOption="@(new CommandButtonOptions()
                                                   { IconCss = "e-icons e-save", CssClass = "e-flat" })"></GridCommandColumn>
                <GridCommandColumn Type="CommandButtonType.Cancel"
                                   ButtonOption="@(new CommandButtonOptions()
                                                   {IconCss="e-icons e-cancel-icon", CssClass="e-flat" })"></GridCommandColumn>
            </GridCommandColumns>
        </GridColumn>
    </GridColumns>
</SfGrid>
```

Slika 3.12. HTML kod *Notes.razor* komponente

3.4. Meniji

Studentski centar u Osijeku na svojoj službenoj stranici dnevno objavljuje trenutne ponude jelovnika u restoranima Campus i Istarska. S obzirom na to da stranica koristi zaštićeni API, podatke je moguće prikupiti samo web struganjem. To je moguće korištenjem Selenium ChromeDriver-a. Pri njegovom kreiranju dodana je opcija *headless* drivera kako se prilikom otvaranja menija u aplikaciji ne bi otvarao poseban chrome prozor s danim URL-om. S obzirom na to da se na službenoj stranici koriste JavaScript skripte za popunjavanje tablica menija, potrebno je narediti kratko čekanje učitavanja skripti. Konačno, slijedi pretraživanje željenih elemenata preko XPath (XML Path Language) putanja. XPath je query jezik koji se koristi za odabiranje čvorova unutar XML ili HTML dokumenta [17]. Nakon dohvaćanja, stvorene varijable spremaju se unutar objekta klasa *Menu* i *MenuEntry* čiji je UML model prikazan na slici 3.13.



Slika 3.13. UML model klasa *Menu* i *MenuCategory*

Klasa *Menu* sastoji se od *Name* svojstva koje predstavlja ime restorana te *Categories*, tj. liste *MenuCategory* objekata. Klasa *MenuCategory* sadržava svojstvo *Title* kao naziv kategorije u koju spadaju određena jela, te *Meals* koji predstavlja spomenutu listu jela.

Prošli opisani postupak dohvaćanja napisan je u Server dijelu aplikacije, unutar *MenuServices* mape. Nju poziva *MenuController*, *ApiController* koji služi za komunikaciju klijent i server projekta. Sastoji se od *IMenuService* parametra kojeg postavlja unutar konstruktora. Njegova glavna metoda je tipa *Task*, asinkrona operacija koja može vratiti vrijednost i *Status 200 OK* odgovor ako je zahtjev uspio. Unutar nje poziva se usluga koja obavlja zadatak dohvaćanja menija. *MenuController* prikazan je na slici 3.14.

```
using FERITOrganizator.Server.Services.MenuService;
using FERITOrganizator.Shared.Models;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace FERITOrganizator.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference | 0 requests
    public class MenuController : ControllerBase
    {
        private readonly IMenuService menuService;

        0 references | 0 exceptions
        public MenuController(IMenuService menuService)
        {
            this.menuService = menuService;
        }

        [HttpGet]
        0 references | 0 requests | 0 exceptions
        public async Task<ActionResult<List<Menu>>> GetMenus()
        {
            return Ok(await menuService.GetMenus());
        }
    }
}
```

Slika 3.14. Klasa *MenuController*

Time dolazimo do *Client* dijela projekta, točnije do *MenuService* klase unutar klijenta koji je obavio poziv kontrolera. Navedeno je postignuto korištenjem objekta *HttpClient* klase. Prikazano svojstvo sa slike 3.15. popunjava se u *LoadMenus*, asinkronoj metodi tipa *Task* koja poziva kontroler iz server dijela i čeka njegovo daljnje usmjeravanje i popunjavanje *Menu* objekata.

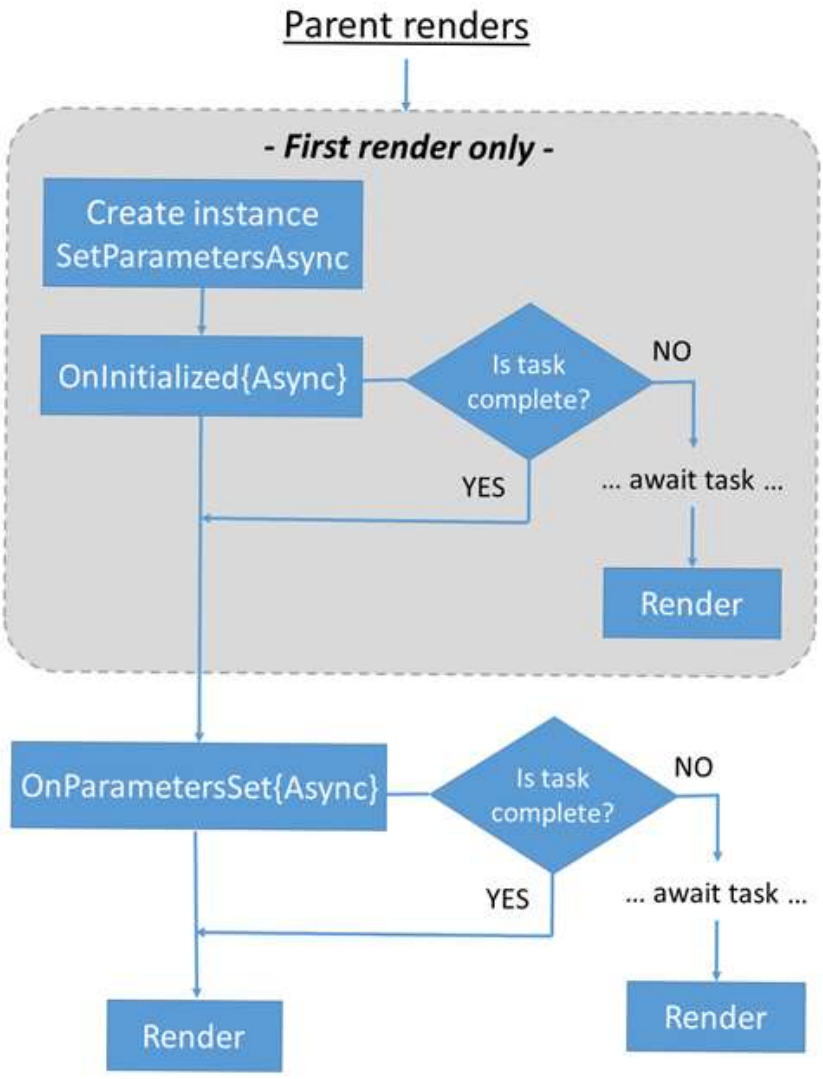
```
using FERITOrganizator.Shared.Models;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;

namespace FERITOrganizator.Client.Services.MenuService
{
    2 references
    public class MenuService:IMenuService
    {
        private readonly HttpClient http;
        3 references | 0 exceptions
        public List<Menu> Menus { get; set; } = new List<Menu>();

        0 references | 0 exceptions
        public MenuService(HttpClient http)
        {
            this.http = http;
        }

        2 references | 0 exceptions
        public async Task LoadMenus()
        {
            Menus = await http.GetFromJsonAsync<List<Menu>>("api/Menu");
        }
    }
}
```

Slika 3.15. Klasa *MenuService* (*Client*)

Menus.razor stranica u *code* dijelu unutar *OnInitializedAsync* metode poziva service za učitavanje menija, a u HTML dijelu formira i prikazuje tablicu s elementima *Menu* objekta.

3.5. Poslovi

Zadnja stranica aplikacije omogućava uvid u najnovije objave i oglase sa službene Facebook stranice Studentskog servisa Osijek. Dohvaćanje objava obavlja se na po uzoru na dohvaćanje menija, korištenjem Selenium ChromeDriver-a koji struga elemente web stranice. Nakon učitavanje poslova unutar *OnInitialized* metode, prikazuju se tablicom. *OnInitialized* metoda je primjer komponenti koje je moguće definirati unutar Razor stranice. Razor komponenta procesira događaje u setu njihovih sinkronih i asinkronih životnih ciklusa. Njih se može implementirati ako se žele dodati dodatne operacije. Dijagram 3.1. prikazuje životni ciklus metoda unutar Razor komponente.

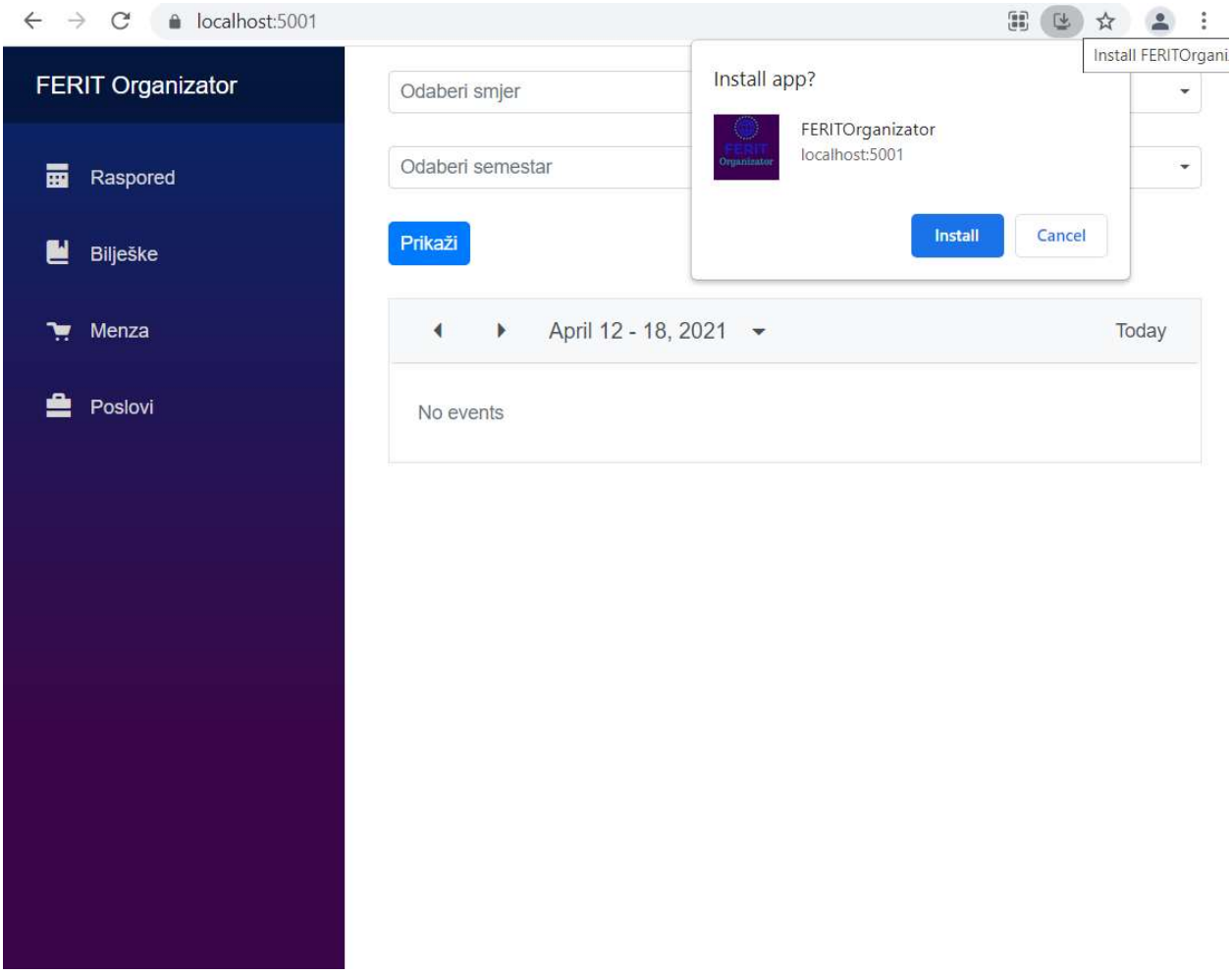


Dijagram 3.1. Životni ciklus Razor komponenti [18]

Ako se zahtjev za prikaz komponente pojavljuje prvi put, stvara se instanca komponente te pokreće *SetParametersAsync* metoda koja postavlja parametre predane preko roditelja komponente. Povratni tip joj je *Task* koji je izvršen ažuriranjem elementa. Zatim se zove *OnInitializedAsync* asinkrona metoda pozvana kad je komponenta spremna započeti te je primila početne parametre. Ako zadatak nije izvršen, čeka se izvršavanje *Task* objekta te se komponenta ponovno prikazuje. Zatim se poziva *OnParametersSetAsync* metoda koja se poziva kad je komponenta primila parametre te su prikladne vrijednosti dodijeljene svojstvima komponente. Kao i u prošlom slučaju, ako zadatak nije izvršen, čeka se izvršavanje *Task* objekta te se komponenta ponovno prikazuje.

4. PRIKAZ RADA

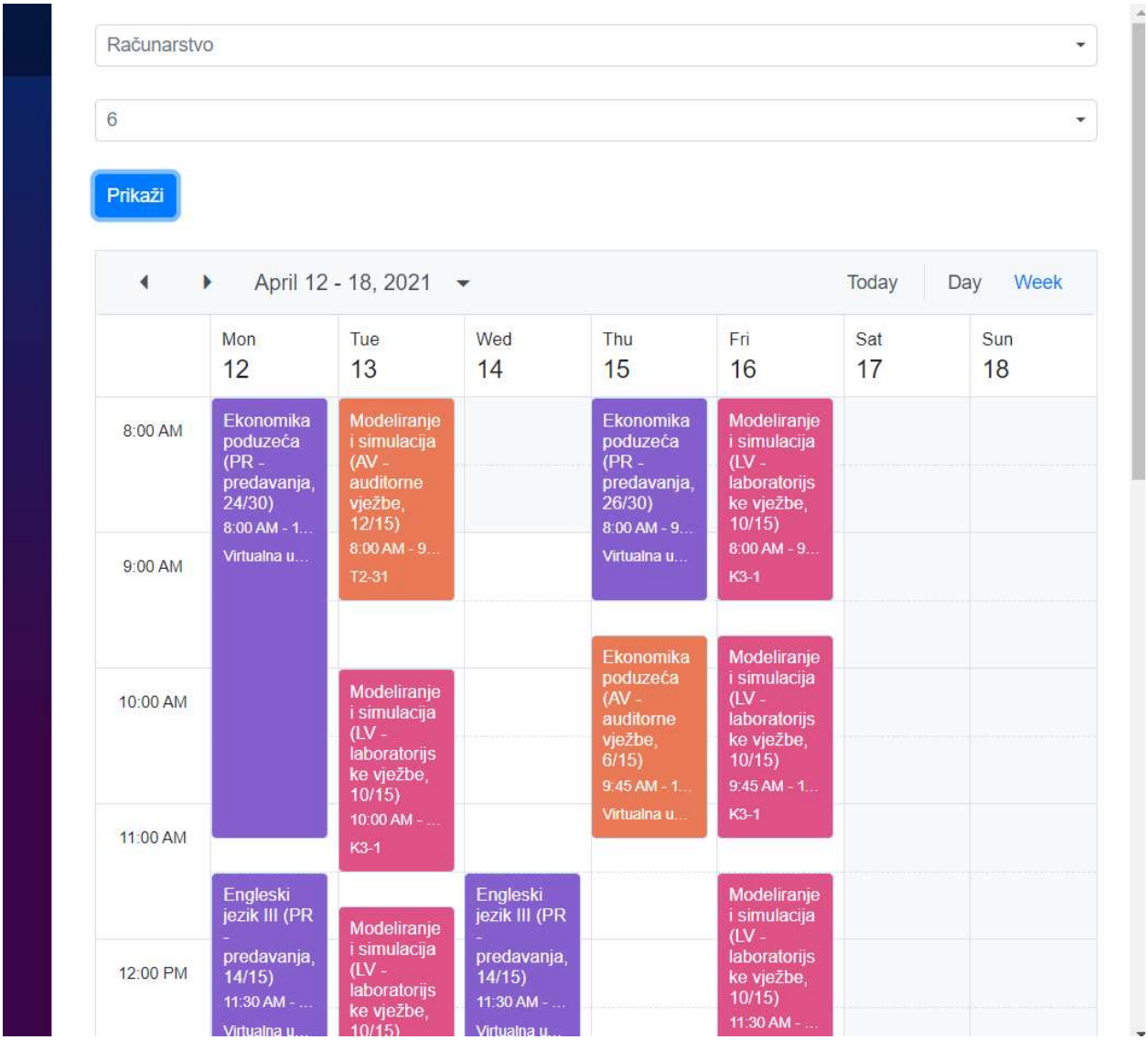
Aplikacija FERIT Organizator je progresivna web aplikacija, što znači ju je moguće otvoriti i instalirati na desktop ali i na mobilni uređaj. Zbog toga je vrlo bitno da sama aplikacija i njene komponente budu responzivni, tj. da budu čitljive i lake za korištenje neovisno o širini ekrana na kojem se nalaze. Na slici 4.1. prikazana je početna stranica web stranice. S desne trake u kojoj se nalazi putanja stranice nalazi se ikona za instalaciju aplikacije. Ovakav način instalacije moguć je jedino kod progresivnih web aplikacija koje zaobilaze dugotrajni, tradicionalni način instalacije preko App Store aplikacija.



Slika 4.1. Početna stranica u web pregledniku

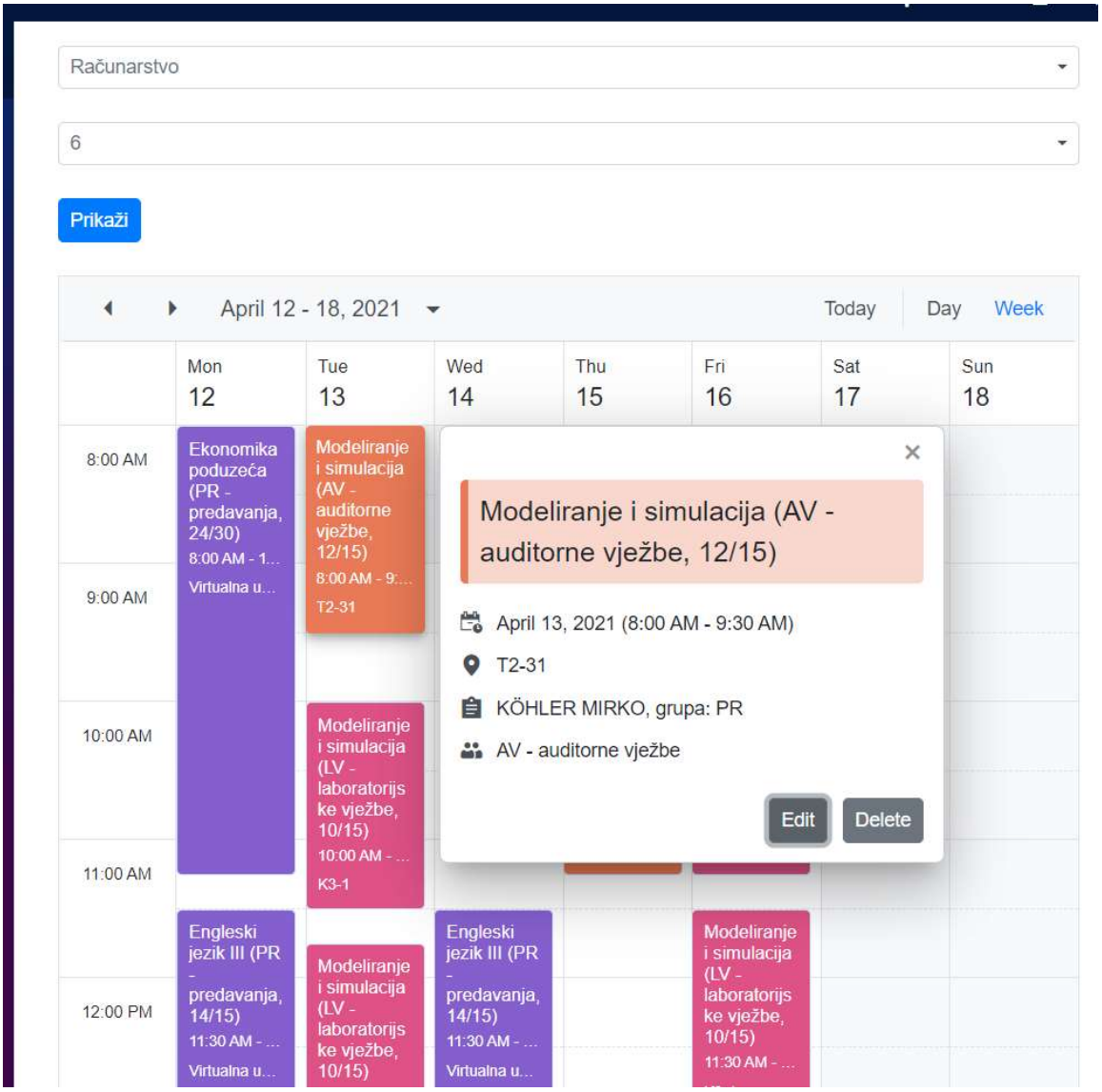
S gornje lijeve strane nalazi se ime aplikacije, a ispod nje je navigacijska traka s pojedinim stranicama aplikacije. Kao početna stranica izabrana je stranica u kojoj se prikazuje raspored. Na njoj student bira svoj smjer i trenutni polazeći semestar te se klikom na gumb generira njegov

raspored za tjedan, vidljivo na slici 4.2. Osim tjednog prikaza, korisnik može pregledati i dnevni raspored.



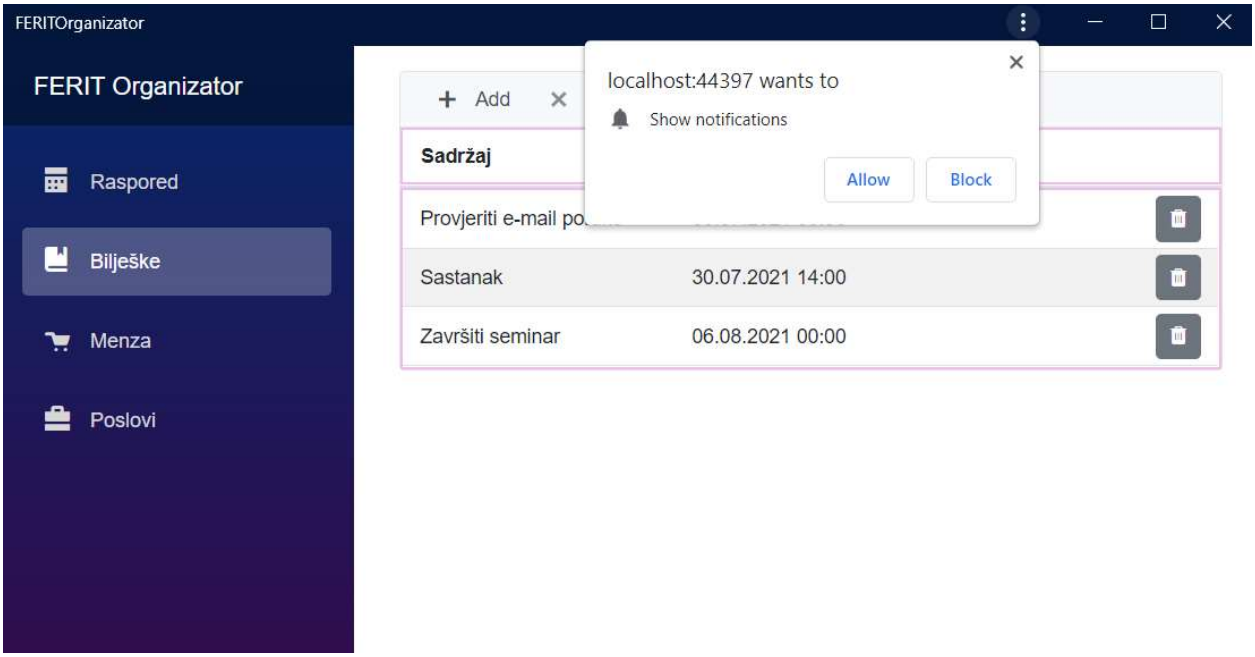
Slika 4.2. Prikazani tjedni raspored studenta

Syncfusion nudi veliki izbor prigodnih, interaktivnih komponenti, pa tako i *Schedule* komponenta omogućava detaljniji prikaz u stavke rasporeda što je vidljivo na slici 4.3.



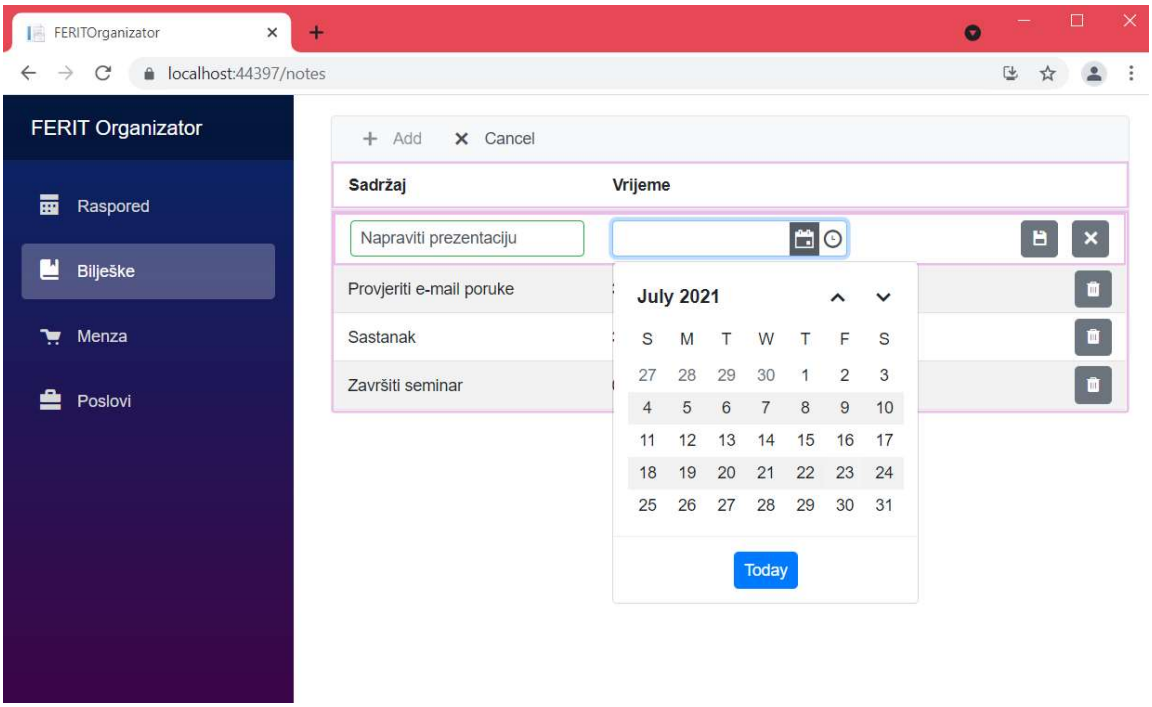
Slika 4.3. Detaljni uvid u stavku rasporeda

Odabirom sljedeće stranice aplikacije dolazimo do korisnikovih bilješki. Prvim korisnikovim ulazom u aplikaciju iskače prozor koji traži njegovu dozvolu za slanje obavijesti vidljivo na slici 4.4.



Slika 4.4. Upit aplikacije za prihvaćanje obavijesti

Na slici 4.5. prikazan je primjer unošenja nove bilješke, njenog naslova te vremena predviđenog za njeno izvršavanje.

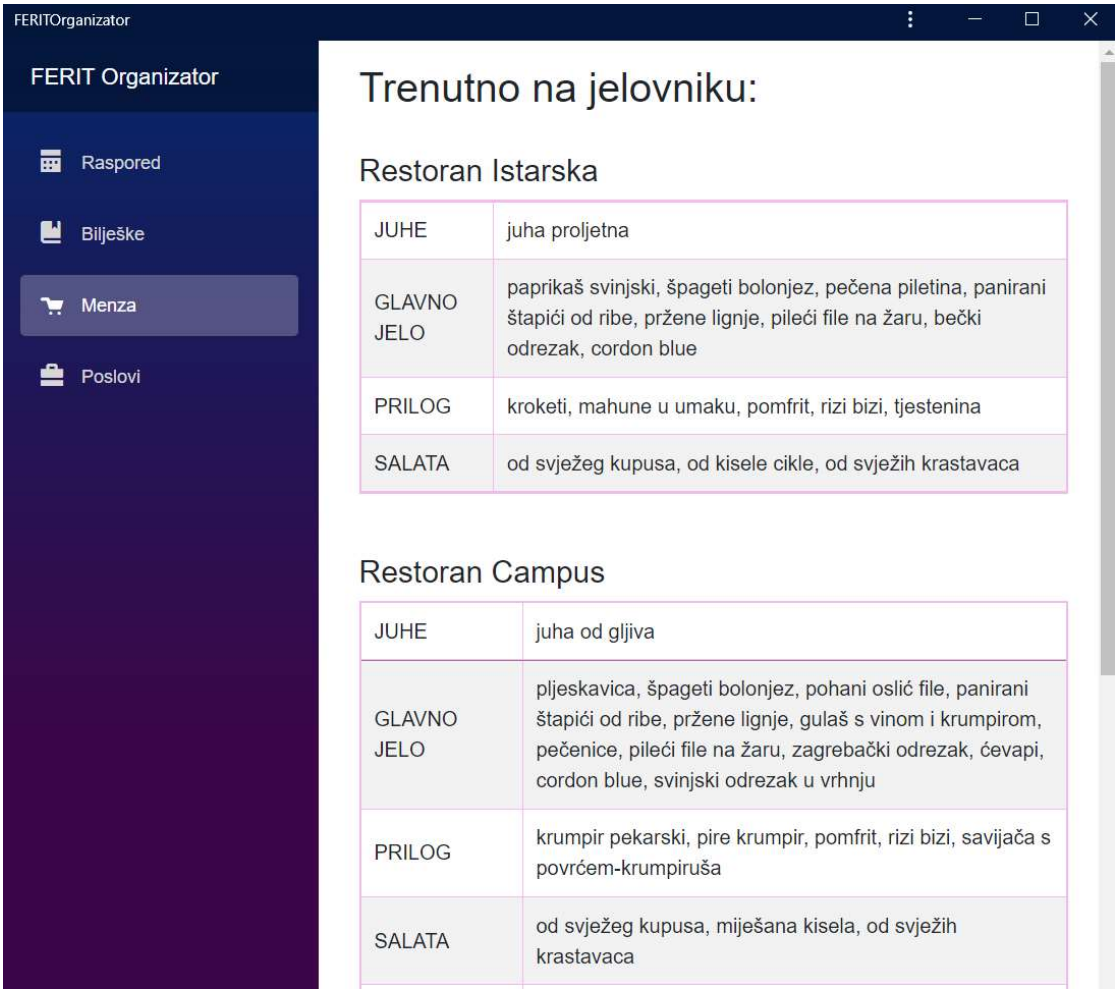


Slika 4.5. Primjer unošenja nove bilješke

Kada se nova bilješka doda u postojeći popis, korisnik je o tome pravovremeno obaviješten dobivanjem push poruke. Samim time može se uvjeriti o uspješnosti izvršavanja radnje. Na slici 4.6. prikazan je Windows prozor u kojem se nalaze obavijesti raznih aplikacija korisnika. Pošto se radi o progresivnoj aplikaciji iza koje stoji web preglednik, obavijest će na isti način iskočiti i kad je pokrenuta aplikacija sa mobilnog uređaja.

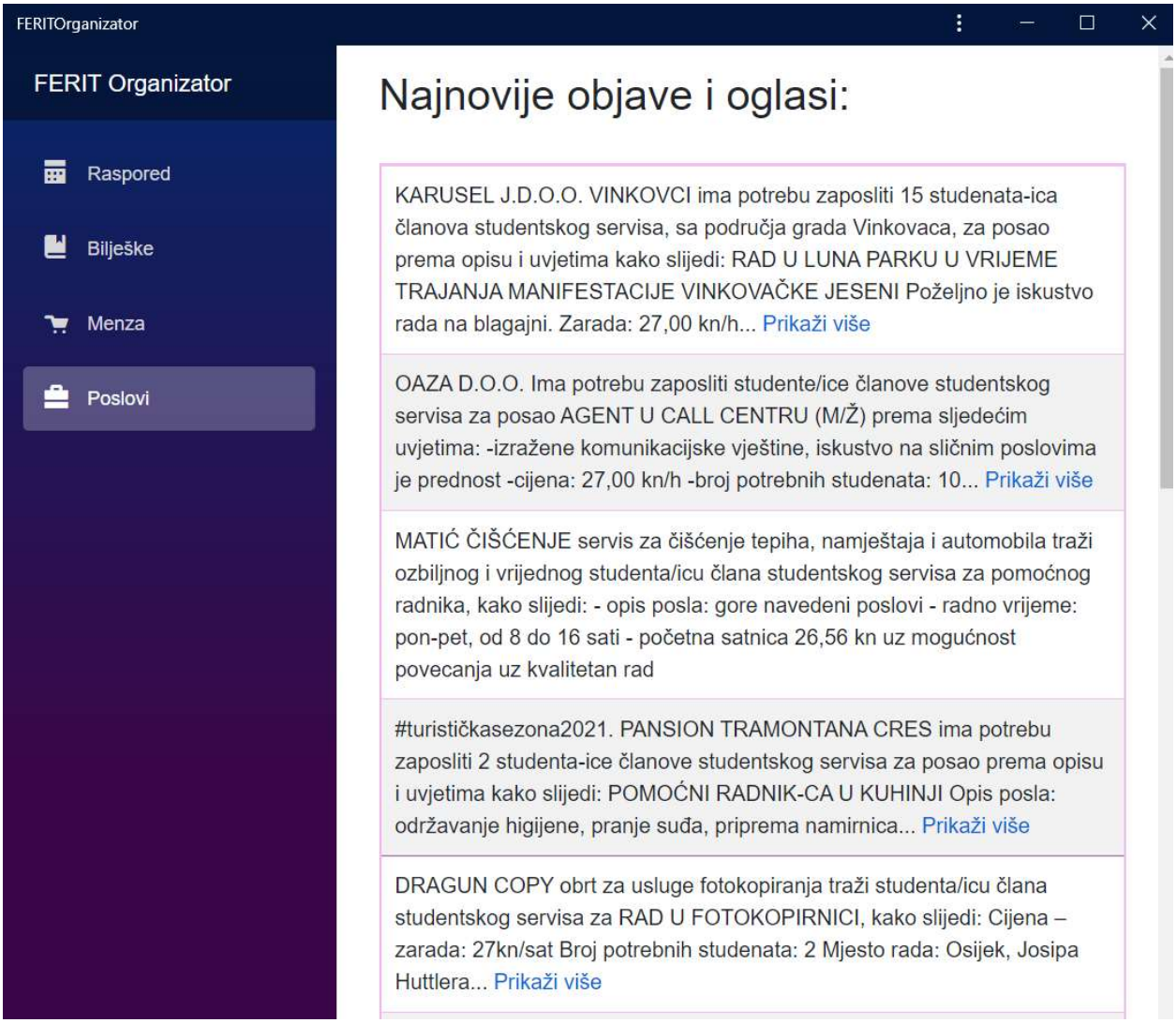
Slika 4.6. Primljena push obavijest

Nakon bilješki slijedi stranica označena u navigacijskoj traci pod nazivom Menza, a prikazuje trenutne dnevne jelovnike osječkih menzi, restoran Campus te restoran Istarska. Unutar tablice sa slike 4.7. prikazane su postojeće kategorije i ponuda jela raspoloživa za kupnju tog dana.



Slika 4.7. Primjer dnevnog jelovnika menzi Campus i Istarska

Zadnja stranica aplikacije sa slike 4.8. nudi prikaz najnovijih objava s Facebook stranice Studentskog centra Osijek među kojima većinom prevladavaju ponude za posao raznih firmi. Korisnik može proučiti dio objave te ako je zainteresiran, klikom na Prikaži više odveden je na originalnu Facebook stranicu gdje može pronaći detalje objave.



Slika 4.8. Primjer prikaza aktualnih objava Studentskog centra Osijek

5. ZAKLJUČAK

Zadatak ovog završnog rada je izraditi progresivnu web aplikaciju koju lako mogu koristiti studenti FERIT-a. Ona omogućuje odabir i pregled odgovarajućeg rasporeda nastave i ispita studenta, unos korisnikovih bilješki, prikaz trenutnog jelovnika osječkih menzi te najnovijih objava Studentskog servisa Osijek. Korištena je Blazor WebAssembly aplikacija te je odabrana opcija ASP.NET Core *hosted* modela koji se sastoji od tri projekta: *Client*, *Server* i *Shared*.

Navedene korištene tehnologije te njihove značajke, prednosti i mane opisane su u među prvim potpoglavljima rada. Opisana je razlika između Blazor Server i Blazor WebAssembly aplikacije. Uveden je pojam progresivne aplikacije, njeno uspoređivanje s nativnim aplikacijama te mogućnost slanja push obavijesti korisnicima. Za rukovanje s bazom podataka, korišten je Microsoft SQL Server 2019 u kojem su unesene korisnikove bilješke i podaci o pretplaćenim korisnicima za slanje push poruka. Za prikaz mnogih elemenata aplikacija, osim običnog HTML koda, korišten je Syncfusion i njegove UI komponente. Selenium i njegov ChromeDriver omogućili su lako web struganje podataka potrebnih za menije i poslove.

Nakon teorijskog pregleda, opisana je sama izrada aplikacije. Za raspored korišten je API sustava Mrkve u kojem se nalaze svi bitni podaci kao što su predmeti, prostorije, raspored te studenti. Za prikaz rasporeda definirane su klase koje odgovaraju *SfSchedule* komponenti koja je različito prikazana ovisno o korisnikovoj širini ekrana, omogućavajući pregled na desktop i mobilnim uređajima. Korisnik može pregledati i unijeti nove bilješke koje se spremaju u bazu podataka te ako dopuste primanje obavijesti, primaju push obavijest o svakom uspješnom dodavanju nove bilješke. Selenium ChromeDriver osigurava pokretanje JavaScript skripti zbog čega je idealan za web struganje elemenata web stranica koje su korištene za prikupljanje i prikaz dnevnih jelovnika restorana Campusa i Istarska te aktualnih objava i oglasa za posao sa Studentskog centra Osijek. Nakon opisa izvedbe aplikacije, prikazane su sve njene stranice i funkcionalnosti.

Navedena aplikacija mogla bi se poboljšati uvođenjem logiranja čime bi se omogućio privatniji pregled studentovih informacija te ograničavanje pristupa osjetljivim stranicama kao što su raspored i bilješke. Zbog samog sustava Mrkve koji omogućava pregled rasporeda samo dan po dan, obrađivanje i dohvaćanje mora se izvoditi šest puta umjesto jednom za cijeli tjedan. Ovo rezultira znatno usporenim postupkom prikaza rasporeda, zbog čega se u aplikaciji prikazuje samo trenutni tjedan, a ne i raspored za idući ili prošli tjedan. Također, stranice za prikaz dnevnih menija i trenutnih poslova otvorene su za uvođenje interaktivnih mogućnosti za

korisnike, kao što su izbor i spremanje jela iz navedenih kategorija te spremanje željenih objava sa Studentskog centra Osijek.

LITERATURA

[1] Microsoft, Blazor: Interactive web UI with C#, 2020., dostupno na:

https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor [21.8.2021.]

[2] Microsoft, Blazor: ASP.NET Core Blazor supported platforms, 2020., dostupno na:

https://docs.microsoft.com/en-us/aspnet/core/blazor/supported-platforms?view=aspnetcore-5.0 [21.8.2021.]

[3] Microsoft, ASP.NET Core Blazor hosting models: Blazor Server, 2020., dostupno na:

https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-5.0#blazor-server-1 [21.8.2021.]

[4] Microsoft, Introduction to ASP.NET Core SignalR: What is SignalR?, 2019., dostupno na:

https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-5.0 [21.8.2021.]

[5] V. G. Guzman Lucio, 3 Different Hosting Models in Blazor: ASP.NET Core, Syncfusion, 2021., dostupno na:

https://www.syncfusion.com/blogs/post/3-blazor-hosting-models.aspx#asp-net-core [21.8.2021.]

[6] Microsoft, Progressive Web Applications: Build Progressive Web Applications with ASP.NET Core Blazor WebAssembly, 2021., dostupno na:

https://docs.microsoft.com/en-us/aspnet/core/blazor/progressive-web-app?view=aspnetcore-5.0&tabs=visual-studio [21.8.2021.]

[7] I. Luong, PWA vs Native App and how to choose between them, Magestore, 2021., dostupno na:

https://www.magestore.com/blog/pwa-vs-native-app-and-how-to-choose-between-them/ [21.8.2021.]

[8] Microsoft, SQL Server Management Studio (SSMS): Download SSMS, 2021., dostupno na:

https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15 [21.8.2021.]

[9] Syncfusion, Why you need Essential Studio, dostupno na: https://www.syncfusion.com/ [21.8.2021.]

[10] Syncfusion, Blazor Components: DataGrid, dostupno na:

https://www.syncfusion.com/blazor-components/blazor-datagrid [21.8.2021.]

[11] Syncfusion, Blazor Components: Dropdown List, dostupno na:

https://www.syncfusion.com/blazor-components/blazor-dropdown-list [21.8.2021.]

[12] W3C, WebDriver: Abstract, 2020., dostupno na: https://www.w3.org/TR/webdriver/ [21.8.2021.]

[13] C. Kenny, What web scraping tools are available?: Selenium, Zyte, dostupno na: https://www.zyte.com/learn/what-python-web-scraping-tools-are-available/ [21.8.2021.]

[14] TutorialsTeacher, Web API Controllers, dostupno na:

https://www.tutorialsteacher.com/webapi/web-api-controller [21.8.2021.]

[15] MKS075, HTML vs XML, GeeksforGeeks, 2021., dostupno na:

https://www.geeksforgeeks.org/html-vs-xml/ [21.8.2021.]

[16] Microsoft, Entity Framework Core: Welcome!, 2020., dostupno na:

https://docs.microsoft.com/en-us/ef/core/ [21.8.2021.]

[17] Import.io, XPath Tutorial: Using XPath: What is XPath?, 2018., dostupno na:

https://www.import.io/post/xpath-tutorial-intro-to-xpath/ [21.8.2021.]

[18] Microsoft, ASP.NET Core Razor component lifecycle: Lifecycle events, 2020., dostupno na: https://docs.microsoft.com/en-us/aspnet/core/blazor/components/lifecycle?view=aspnetcore-5.0 [21.8.2021.]

SAŽETAK

Ova progresivna aplikacija namijenjena je studentima FERIT-a kako bi objedinili sve informacije bitne tijekom njihovog prosječnog dana. Tehnologija koja podržava izradu progresivne aplikacije je Blazor WebAssembly, a njene glavne karakteristike i ostale korištene funkcionalnosti opisane su u samom radu. Zatim je opisana izrada aplikacije koja je podijeljena u četiri stranice. Prikupljanje dnevnog rasporeda obavlja se koristeći API sustava Mrkve, a Syncfusion *Scheduler* komponentom prikazuje se dnevni i tjedni raspored. Omogućen je unos i pregled bilješki, njihovo spremanje u bazu podataka te dobivanje push obavijesti o njihovom uspješnom spremanju. Korištenjem Selenium ChromeDriver elementom obavlja se struganje web stranica, čiji se elementi spremaju u varijable i koriste za prikaz dnevnog jelovnika osječkih menzi te prikaz aktualnih objava i oglasa Studentskog centra Osijek. Konačno, opisane stranice aplikacije i njihove funkcionalnosti prikazane su u radu uz dodatno objašnjenje.

Ključne riječi: Blazor WebAssembly, Microsoft SQL Server, progresivna aplikacija, Syncfusion, web struganje

ABSTRACT

Web application for FERIT students

This progressive application is intended for FERIT students to unite all of their important information from their average day. Technology which supports progressive applications is Blazor WebAssembly. Its main characteristics and functionalities are further described in the paper. After theoretical overview comes description of production of an application which is divided into four web pages. Getting a daily schedule is done by using an API from Mrkve system. Syncfusion Scheduler component enables daily and weekly schedule view. Users can also enter and review notes which are saved in the database. If the note is successfully saved, they get a push notification. One of Selenium ChromeDrivers functionalities is web scraping web pages whose elements are saved in variables and used for displaying daily menus from student restaurants in Osijek and current posts and job ads from Studentski centar Osijek. Finally, described application pages and their performances are shown in the paper with proper explanation.

Keywords: Blazor WebAssembly, Microsoft SQL Server, progressive application, Syncfusion, web scraping

PRILOZI

[1] Web_aplikacija_za_studente_FERIT-a.docx

[2] Web_aplikacija_za_studente_FERIT-a.pdf

[3] Programsko rješenje aplikacije