Izrada aplikacije Produjekt za naručivanje pića online

Dašić, Jelena

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Naziv preddiplomskog sveučilišnog studija

Jelena Dašić

Izrada aplikacije Produjekt za naručivanje pića online

Završni rad

Mentor: izv. prof. dr. sc. Marija Brkić Bakarić

Rijeka, rujan 2023.

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

Rijeka, 26.5.2023.

Zadatak za završni rad

Pristupnik: Jelena Dašić

Naziv završnog rada: Izrada aplikacije Produjekt za naručivanje pića online

Naziv završnog rada na engleskom jeziku: Development of the application Produjekt for online ordering of drinks

Sadržaj zadatka: Cilj rada je prikazati razvoj grafičke aplikacije za naručivanje online. Aplikacija je implementirana u C# jeziku pomoću .NET MAUI Hybrid i .NET Core okvira koji su dio .NET platforme. Aplikacije stvorene putem MAUI platforme namijenjene su za rad na Android, Windows i iOS uređajima. Frontend dio aplikacije implementiran je u HTML-u, CSS-u. Naziv aplikacije je Produjekt, aplikacija podržava prikaz sadržaja na dva jezika i CRUD operacije nad bazom podataka preko API-ja.

Mentor

Izv. prof. dr. sc. Marija Brkić Bakarić

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 26.5.2023.

Jelena Dašić

(potpis pristupnika)

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

Sadržaj

Sažetak

Tema ovog završnog rada je izrada višeplatformske aplikacije za webshop koja služi za naručivanje pića i koktela online. Aplikacija je razvijena u .NET okviru, koristeći ASP.NET Core i MAUI. U ASP.NET Core biti će razvijen API, dok je MAUI mobilni i desktop dio aplikacije. Za bazu podataka se koristi MongoDB a za dizajn sučelja aplikacije HTML, CSS i Bootstrap. Aplikacija ima različite mogućnosti ovisno o tome je li korisnik prijavljen te kontrolu pristupa za prijavljene korisnike. Za prijave i registraciju korisnika koristi se Microsoft Azure Active Directory B2C. Aplikacija je razvijena u Visual Studio 2022.

Ključne riječi

MAUI, MAUI Blazor Hybrid, ASP.NET Core, Minimal API, Azure, Azure Active Directory B2C, NoSQL, MongoDb, Webshop, C#, HTML, CSS, Bootstrap

Summary

The topic of this thesis is the creation of a cross-platform webshop application Produjekt for online ordering of drinks and cocktails. The application is developed in the .NET framework, using ASP.NET Core and MAUI. The API is developed in ASP.NET Core, while MAUI is the mobile and desktop part of the application. MongoDB is used for the database, and HTML, CSS and Bootstrap are used for user interface design. The application has different options depending on whether the user has registered, as well as access control for logged in users. Microsoft Azure Active Directory B2C is used for logins and user registrations. The application is developed in Visual Studio 2022.

Uvod

Svrha ovog završnog rada je izraditi i opisati razvoj aplikacije Produjekt. Produjekt je aplikacija za naručivanje pića i koktela, Za izradu aplikacije koristi se integrirano razvojno okruženje Visual Studio. Za autentifikaciju i autorizaciju korisnika koristi se Azure Active Directory B2C. Baza podataka je MongoDB dokumentna baza podataka.

U aplikaciji postoje tri vrste korisnika: neprijavljeni, prijavljeni i administratori. Neprijavljeni korisnici imaju mogućnost pregleda dostupnih pića i koktela, te kreiranja narudžbe. Prijavljeni korisnici također imaju mogućnost pregleda dostupnih pića i koktela ali osim toga vide i svoje prethodne narudžbe. Administratori imaju sve mogućnosti prijavljenih korisnika uz mogućnost pregleda svih narudžbi, te dodavanja novih pića i koktela.

Funkcionalnosti aplikacije su prijava korisnika, upload datoteka, dvojezičnost i ažuriranje podataka u bazi preko API sučelja. Putem API sučelja je implementiran upload datoteka i ažuriranje podataka, dok je dvojezičnost i prijava korisnika implementirana u MAUI dijelu aplikacije.

Korištene tehnologije

MAUI Blazor Hybrid

.NET Multi-platform App UI (MAUI), poznat i kao .NET MAUI, predstavlja višeplatformski razvojni okvir otvorenog koda stvoren od strane Microsofta za izradu mobilnih i desktop aplikacija pomoću programskih jezika C# i XAML. Ovaj okvir je evolucija prethodnog alata Xamarin.Forms, pri čemu donosi naprednije mogućnosti i veću fleksibilnost. Putem .NET MAUI-a, moguće je razvijati aplikacije koje se mogu izvoditi na Android uređajima, iOS uređajima, macOS računalima te Windows uređajima, sve to koristeći jedan zajednički kod.

Za razvoj aplikacije Produjekt koristiti će se .NET MAUI Blazor Hybrid. .NET MAUI Blazor Hybrid je spajanje .NET tehnologija MAUI i Blazor. Blazor aplikacije se sastoje od web komponenata izgrađenih putem C#, HTML-a i CSS-a, gdje se C# može koristiti i u frontend i u backend dijelu aplikacije. Za izradu stranica u Blazoru koriste se Razor komponente. .NET MAUI razvojni okvir u sebi uključuje BlazorWebView koji omogućava korištenje Razor komponenata u razvoju višeplatformskih aplikacija pomoću .NET MAUI. S time se omogućuje da mobilne, desktop i web aplikacije dijele isti kod. S .NET MAUI Blazor Hybrid razvijaju se mobilne i desktop aplikacije, ali se kod može dijeliti i sa web aplikacijama te se tako olakšava razvoj.

MongoDB

MongoDB je NoSQL sustav za upravljanje bazama podataka. MongoDB je dokumentna baza podataka koja omogućava organizaciju podataka u BSON formatu. BSON (engl. Binary JSON) je format koji dijeli puno toga sa JSON (engl. JavaScript Object Notation) formatom. BSON omogućava veliku brzinu i efikasnost u spremanju podataka, dok zadržava čitljivost JSON formata. Osim brzine i čitljivosti podataka, BSON podržava i više tipova podataka od JSON formata kao npr. datume i binarne podatke.

Microsoft Azure

Microsoft Azure je platforma izrađena za fleksibilan i skalabilan razvoj aplikacija. Azure nudi puno alata koji omogućavaju cloud hosting, nude skalabilnost i odlične performanse. Azure App Service omogućuje hosting aplikacija te dodavanje novih funkcionalnosti i mogućnosti.

Azure Active Directory B2C

Azure Active Directory B2C (Azure AD B2C) je servis koji omogućava autentifikaciju i autorizaciju korisnika u web ili mobilnim aplikacijama. Osim autentifikacije, Azure AD B2C može se koristiti i za definiranje privilegija koje korisnici imaju, odnosno upravljanje pristupom API-jima i API endpointovima.

Autentifikacija i Autorizacija

Azure AD B2C može se koristiti sa različitim providerima, te će se za aplikaciju Produjekt koristiti Google i email prijava. Korisnici prijavljeni preko Google računa imati će privilegije korisnika, dok korisnici prijavljeni preko Microsoft računa imaju privilegije administratora.

HTML

HTML(Hyper Text Markup Language) je jezik za izradu web stranica. Preglednicima omogućava interpretaciju i prikazivanje sadržaja na web stranicama. Koristi oznake kojima označava različite dijelove sadržaja i njihovu ulogu, kako bi preglednici mogli prikazivati tekst, veze, slike i druge elemente na web stranicama. HTML je razvio Tim Berners-Lee 1989. godine kao jezik koji će se koristiti uz World Wide Web. Od tad, HTML je prošao kroz više iteracija, a zadnja, HTML 5 je izdana 2008. godine.

CSS

CSS(Cascading Style Sheet) je jezik za stiliziranje web stranica. Povijest CSS-a kreće od 1994. godine s ciljem razvoja jezika koji će služiti za stiliziranje HTML-a. Koristi se uz HTML za kontrolu izgleda stranice. HTML određuje sadržaj stranice i strukturu iste, a CSS omogućuje da se taj sadržaj stilizira. CSS se koristi za upravljanje izgleda elemenata poput boje, fonta, veličine teksta, margina i slično.

Bootstrap

Bootstrap je biblioteka HTML, CSS i Java Script komponenata otvorenog koda koja se koristi za izradu responzivnih sučelja aplikacija. Bootstrap je razvio Twitter, te ga je za javnost objavio 2012. godine kao alat za olakšavanje razvoja sučelja web aplikacija. Bootstrap je od svoje druge verzije responzivan, te tako olakšava razvoj aplikacija koje se koriste na ekranima različitih veličina. Zadnja verzija Bootstrap 5 izašla je 2021. godine, te se ona koristi u ovom projektu za razvoj responzivnog sučelja.

Funkcionalnosti aplikacije

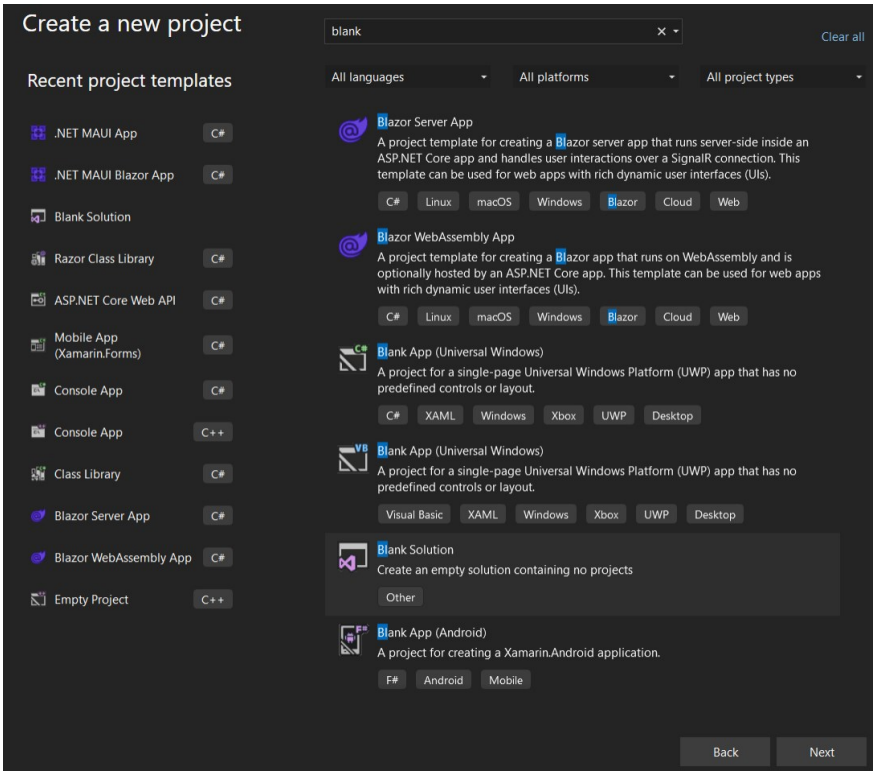
Implementirane funkcionalnosti u zavisnosti od statusa korisnika i razine pristupa prikazane su u Tablici 1.

Funkcionalnosti	Svi korisnici	Prijavljeni korisnici	Administratori
Prijava	Google ili Email	Google ili Email	Microsoft
Pregled dostupnih pića i koktela	Da	Da	Da
Stvaranje narudžbe	Da	Da	Da
Pregled prijašnjih narudžbi	Ne	Da	Da
Kreiranje novih pića i koktela	Ne	Ne	Da
Pregled narudžbi svih korisnika	Ne	Ne	Da
Slanje emaila	Potvrda narudžbe	Potvrda narudžbe	Potvrda narudžbe
Upload datoteka	Ne	Ne	Da
Dvojezičnost aplikacije	Hrvatski i Engleski	Hrvatski i Engleski	Hrvatski i Engleski
Komunikacija s bazom podataka preko API	Da	Da	Da

Tablica 1 Funkcionalnosti i razine pristupa po statusu korisnika

Kreiranje projekta

Za izradu projekta koristi se Visual Studio 2022. Prvo ćemo kreirati blank solution u Visual Studio 2022(slika 1) u koji ćemo naknadno dodati projekte za MAUI aplikaciju te API sučelje. Na ovaj način će jedan projekt sadržavati i MAUI i ASP.NET Core dijelove aplikacije.



1 Kreiranje praznog projekta u Visual Studio 2022.

Struktura projekta

Projekt se sastoji od dva glavna dijela: MAUI aplikacije te API sučelja

Mobilna aplikacija

Cilj izrade MAUI aplikacije je jednostavno sučelje koje omogućava sve funkcionalnosti kreiranja, brisanja i ažuriranja koje su dostupne na API endpointovima ovisno o privilegijama korisnika.

Application Programming Interface

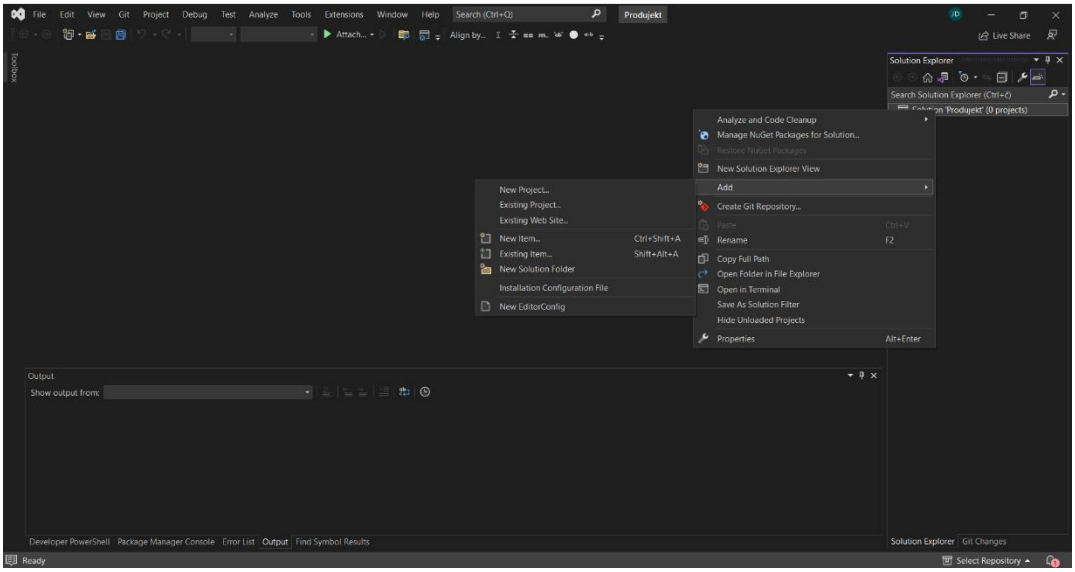
Cilj izgradnje API sučelja je osigurati pouzdanu i efikasnu komunikaciju između mobilne aplikacije i baze podataka. U tu svrhu, koristit će se minimalni API pristup, omogućen unutar ASP.NET Core okvira.

API će sadržavati niz endpointova koji će omogućiti sljedeće funkcionalnosti:

- kreiranje, ažuriranje i brisanje koktela i pićaće biti moguće samo za administratore,
- kreiranje, čitanje i brisanje narudžbi,
- slanje e-mail potvrde narudžbe nakon kreiranja,
- čitanje koktela i pića će biti dostupno svim korisnicima.

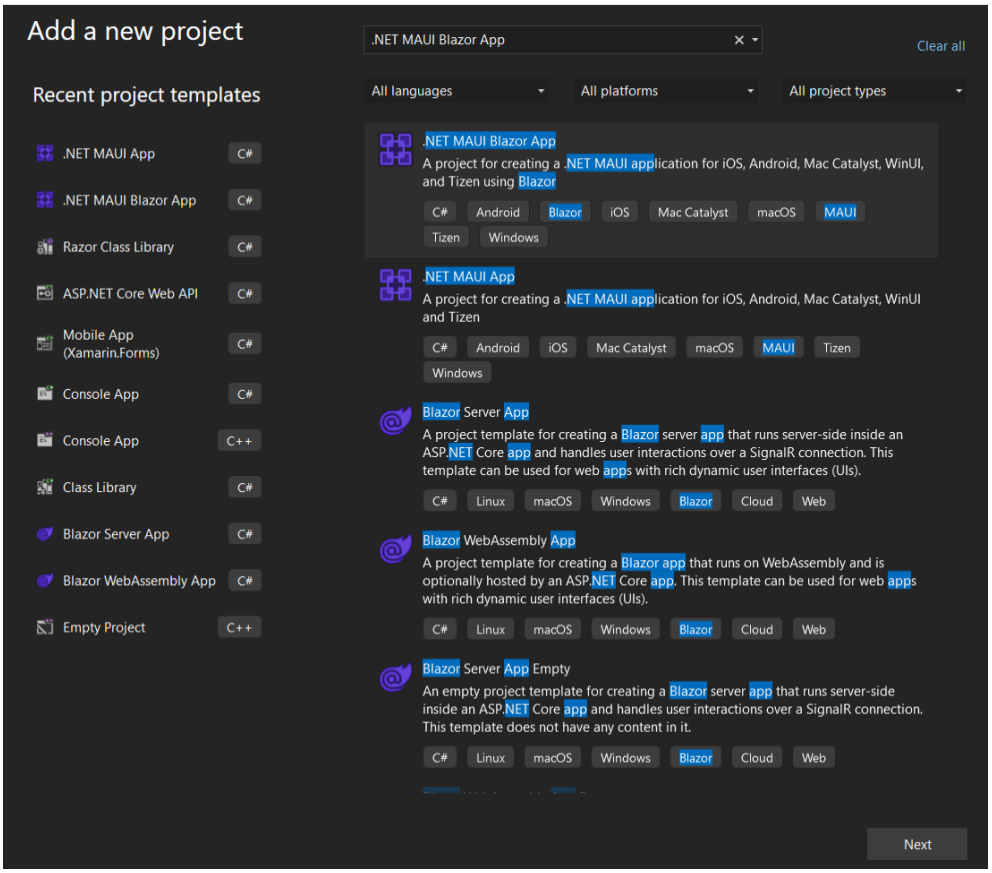
Izrada mobilne aplikacije

Projekt za mobilnu aplikaciju dodajemo u postojeći *solution* desnim klikom te na Add New Project.



2 Dodavanje MAUI aplikacije u postojeći solution

Izaberemo .NET MAUI Blazor App, te mu dajemo ime i biramo verziju.



3 Izbor MAUI Blazor Hybrid aplikacije

Struktura projekta mobilne aplikacije

Struktura .NET MAUI projekta organizirana je kako bi omogućila razvoj nativnih mobilnih i desktop aplikacija s maksimalnom efikasnošću. Svaka komponenta ima specifičnu ulogu u projektu, doprinoseći izgradnji korisničkog sučelja, logike i resursa. Evo kratkog objašnjenja za ključne komponente unutar strukture .NET MAUI projekta:

Dependencies: Ova mapa sadrži informacije o zavisnostima projekta. To mogu biti preuzeti paketi ili specifične zavisnosti za platformu.

Properties: U ovoj mapi se nalaze postavke i konfiguracije za projekt. Ovdje se mogu prilagoditi razne opcije za projekt, kao što su ikone aplikacije i drugi globalni aspekti.

Data: U ovom direktoriju se nalaze modeli koji se koriste u aplikaciji. Također je moguće definirati i servise koji će se koristiti u aplikaciji.

Pages: Stranice korisničkog sučelja aplikacije.

Platforms: U ovoj mapi se nalaze specifične implementacije za različite platforme (npr. Android, iOS, Windows). Ovdje se definiraju platformski specifični resursi, ponašanja i potrebne prilagodbe za nativne aplikacije.

Resources: Ova mapa sadrži resurse kao što su slike, ikone i ostali mediji koji se koriste u aplikaciji.

Shared: Ovdje se nalaze zajedničke komponente koje se koriste na više mjesta u aplikaciji. Ovo pomaže u smanjenju dupliciranog koda i olakšava održavanje. Primjer ovoga je navigacija.

_Imports.razor: Ovaj datoteka omogućava definiranje globalnih direktiva u projektu. Koristi se za uvoz različitih komponenti koje će biti dostupne kroz cijeli projekt.

App.xaml: Centralna datoteka koja definira glavni tok aplikacije, uključujući konfiguraciju navigacije i drugih osnovnih postavki. Ovdje možemo izabrati boju navigacije, fontove, boju pozadine i slično.

Main.razor: Ova datoteka definira glavnu strukturu korisničkog sučelja aplikacije. Ovdje se definira osnovni izgled i raspored elemenata aplikacije.

MainPage.xaml: Osnovna stranica koja će biti prikazana korisnicima prilikom pokretanja aplikacije. U MAUI Blazor Hybrid projektu, ovdje se nalazi BlazorWebView komponenta koja omogućava korištenje razor stranica za izradu aplikacije.

MauiProgram.cs: Ova datoteka sadrži "entry point" aplikacije, gdje se definira početna točka izvođenja aplikacije.

MVVM Arhitektura

Razvoj aplikacije u .NET MAUI Blazor Hybrid temeljit će se na Model-View-ViewModel (MVVM) arhitekturi. MVVM arhitektura sastoji se od tri glavna dijela: Model, Pogled (View) i Pogled Model (ViewModel). Prednosti MVVM arhitekture uključuju enkapsulaciju poslovne logike, sposobnost razvoja unit testova za Model i ViewModel bez korištenja Pogleda te mogućnost redizajniranja korisničkog sučelja aplikacije bez mijenjanja koda.

Pogled(View)

Pogled definira strukturu, raspored i izgled onog što korisnik vidi na ekranu. Pogled modeli (ViewModels) su odgovorni za definiranje logičkih stanja.

Pogled Model(ViewModel)

Pogled Model implementira svojstva i naredbe na koje se Pogled može vezati. Obavještava Pogled o bilo kakvim promjenama stanja putem događaja. Pogled Model treba koristiti asinkrone metode za IO operacije i asinkrone poruke koje obavještavaju Pogled o promjenama svojstava.

Model

Model je klasa koja enkapsulira podatke aplikacije i odgovorna je za čuvanje i upravljanje tim podacima. U ovoj aplikaciji to uključuje pohranu informacija o pićima, koktelima, narudžbama ili drugim entitetima. Također može sadržavati poslovnu logiku za obradu i manipulaciju tim podacima.

Constants.cs

U *Constants* datoteci (slika 4) nalaze se svi podaci koji se koriste između stranica. Na ovaj način u slučaju promjene nekog podatka, može se promijeniti na jednom mjestu umjesto traženja po datotekama gdje se koristi.

```
namespace ProdujektMobile;

public static class Constants
{
    public static string prodUrl = "https://produjektapi.azurewebsites.net/";
    // api access
    public static string DrinksUrl = prodUrl + "api/drinks";
    public static string CocktailsUrl = prodUrl + "api/cocktails";
    public static string EmailUrl = prodUrl + "api/emails";
    public static string FileUrl = prodUrl + "api/files";
    public static string OrderUrl = prodUrl + "api/orders";
    public static string LoginGoogle = prodUrl + "/.auth/login/google";
    public static string LoginMicrosoft = prodUrl + "/.auth/login/aad";
    public static string Logout = prodUrl + "/.auth/logout";
    public static string AccessAsUser = "";
    public static string AccessAsAdmin = "";
}
```

4 Datoteka Constants.cs

Dodavanje stranica

U mapi "Pages", dodane su različite mape koje služe za organiziranje funkcionalnosti vezanih uz modele Cocktails, Drinks i Orders. Svaka od ovih mapa sadrži stranice koje omogućuju različite operacije. Mape sadrže stranice za kreiranje (Create), čitanje svih zapisa u bazi podataka (Index), čitanje jednog zapisa iz baze podataka (Read) i ažuriranje (Update) podataka, osim mape "Orders" koja nema opciju ažuriranja.

Osim stranica grupiranih po entitetima, postoje i stranice koje su neovisne o njima te se nalaze u Pages mapi. To su na primjer stranica Index, na kojoj je prikaz svih pića i koktela te mogućnost dodavanja istih u narudžbu i stranica Login na kojoj su mogućnosti prijavljivanja u aplikaciju. Tu se također nalazi i OrderPage koja se koristi na stranici Indeks kao dijalog.

Ovakva struktura aplikacije olakšava organizaciju i preglednost projekta, te jasno odvaja moguće operacije nad entitetima.

Početna stranica

Na početnoj stranici aplikacije nalazi se prikaz svih pića i koktela u aplikaciji, te mogućnost dodavanja istih u narudžbu. Na svim stranicama koje dohvaćaju podatke sa API-ja, koristi se bool *IsLoading*, kako bi se za vrijeme dohvaćanja podataka iz baze prikazalo da se podaci još učitavaju. U slučaju da se dogodi greška kod dohvaćanja podataka sa API-ja, ona se ispiše na ekran, a ako su podaci učitani bez pogrešaka, prikazuju se dohvaćeni podaci i mogućnosti filtriranja po kategoriji. Vaijabla *isLoading* nalazi se u funkciji OnInitializedAsync (slika 5) koja se poziva kada se stranica otvori. U njoj se radi zahtjev na API endpointe Cocktail i Drink, te nakon što se ti zahtjevi izvrše uspješno ili neuspješno se vrijednost varijable *isLoading* postavlja na *false* kako bi se mogla prikazati ili greška kod dohvaćanja ili podaci koji su dohvaćeni sa API-ja.

```
protected override async Task OnInitializedAsync()
{
    try
    {
        availableCocktails = await Http.GetFromJsonAsync<Cocktail[]?>(Constants.CocktailsUrl);
        availableDrinks = await Http.GetFromJsonAsync<Drink[]?>(Constants.DrinksUrl);
    }
    catch(Exception ex)
    {
        exc = ex.Message;
    }
    isLoading = false;
}
```

5 Funkcija OnInitializedAsync na početnoj stranici

Kategorije po kojima je filtriranje moguće su pića, kokteli i sve. Za prikaz stavki narudžbe i mogućnost kreiranja iste koristi se RadzenButton. Radzen je skup komponenata koje se mogu dodati u MAUI i Blazor projekte. Potrebno je u projekt dodati Radzen pakete i ovisno o komponenti koja se koristi registrirati ju u MauiProgram.cs. U ovom projektu koristi se DialogService komponenta, koja omogućuje dodavanje dialoga u aplikaciju. Dialog se koristi za prikaz stavki narudžbe i kreiranje.

Filtriranje sadržaja na stranici napravljeno je preko 3 bool varijable: *showAll* za prikazivanje koktela i pića, *showDrinks* za prikazivanje pića i *showCocktails* za prikazivanje koktela. Klikom na gumb „Pića“ vrijednost *showDrinks* varijable se postavlja na true, dok vrijednosti *showAll* i *showCocktals* se stavljaju na false. Kada bool varijabla *showDrinks* ima vrijednost „true“ prikazuje se dio koda koji iterira kroz listu svih pića i prikazuje ih. Na isti način funckioniraju i *showCocktails* i *showAll*, uz razliku da *showCocktails* iterira kroz sve koktele, a *showAll* iterira i kroz listu koktela i kroz listu pića (slika 6).

```
private void ToggleItems()
{
    showAll = true;
    showDrinks = false;
    showCocktails = false;
}
private void ToggleDrinks()
{
    showAll = false;
    showDrinks = true;
    showCocktails = false;
}
private void ToggleCocktails()
{
    showAll = false;
    showDrinks = false;
    showCocktails = true;
}
```

6 Funkcije za filtriranje sadržaja

Primjer filtriranog sadržaja stranice za koktele nalazi se na slici 7.

7 Prikaz filtriranog sadržaja na početnoj stranici

Za dodavanje stavki u narudžbu koristi se *orderInMemory*. Razlog tome je jer ne želimo izgubiti podatke o narudžbi ako korisnik otiđe na drugu stranicu u aplikaciji. Na ovaj način, sve stranice gdje nam trebaju podaci o trenutnim stavkama narudžbe mogu koristiti *orderInMemory* za prikaz. Kako bi to bilo moguće, definiranje te mogućnosti radi se u MauiProgram.cs gdje dodajemo sve servise i klase za koje želimo da imaju iste podatke neovisno o stranici na kojoj se korisnik nalazi (slika 8).

```
builder.Services.AddScoped<OrderInMemory>();
builder.Services.AddScoped<AuthService>();
builder.Services.AddScoped<DialogService>();
```

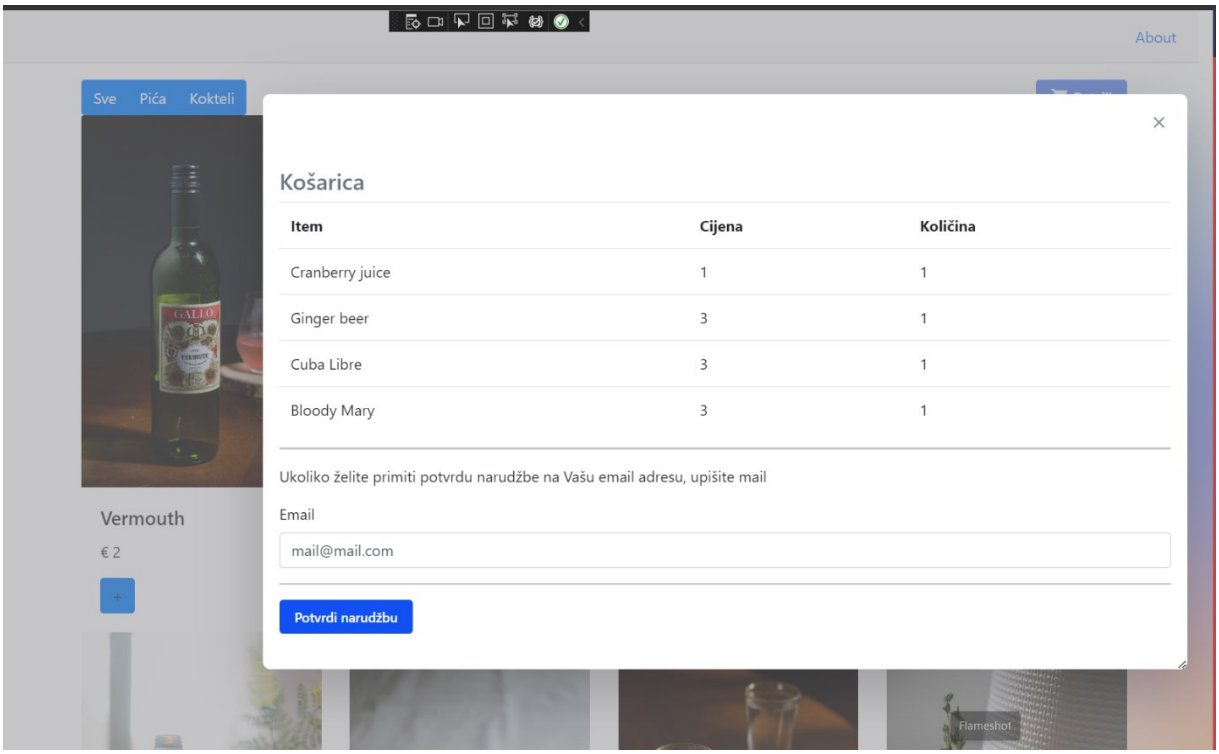
8 MauiProgram.cs i registriranje dijeljenih podataka u aplikaciji

Nakon što dodamo *orderInMemory* u MauiProgram.cs, jedino preostalo je da kod dodavanja stavke u narudžbu, istu dodamo ili u listu pića ili u listu koktela ovisno o tome što dodajemo. Za prikaz stavki koje su trenutno u narudžbi koristimo funkciju *OpenOrder* koja prikazuje *OrderPage* kao dijalog na početnoj stranici (slika 8).

```
private void AddDrinkToCart(Drink drink)
{
    orderInMemory.OrderedDrinks.Add(drink);
    StateHasChanged();
}

private void AddCocktailToCart(Cocktail cocktail)
{
    orderInMemory.OrderedCocktails.Add(cocktail);
    StateHasChanged();
}

public async Task OpenOrder()
{
    await DialogService.OpenAsync<OrderPage>("",
           new Dictionary<string, object>(),
           new DialogOptions() { Width = "700px", Height = "512px", Resizable = true, Draggable = false, CloseDialogOnOverlayClick = true, });
}
```

8 Funkcije za dodavanje stavki u narudžbu i prikaz narudžbe

Prikaz dijaloga i stavki košarice nalazi se na slici 9.



9 Prikaz dijaloga košarice(OrderPage)

Stranice za kreiranje

Dodavanje novih entiteta koktela, narudžbi i pića.

Pića

Stranica za dodavanje pića ima ulogu kreiranja novog pića te uploada datoteke vezane sa tim pićem. Za dodavanje pića potrebno je biti prijavljen korisnik pa se kod slanja zahtjeva za kreiranje na API također šalje i token korisnika koji radi taj zahtjev.

Forma za dodavanje novog pića sadrži podatke o imenu, količini, cijeni i o tome da li piće sadrži alkohol. Osim podataka vezanih za piće, također ima i odabir i prikaz slike pića. Kod za prikaz forme prikazan je na slici 10.

```
@page "/drinks/add"
@using System.Net.Http.Json;
@using System.Text.Json;
@using System.Net.Http.Headers;
@inject NavigationManager Navigation
@inject HttpClient Http
@inject AuthService authService
@inject IStringLocalizer<AppResources> localizer

<h3>@localizer["Create"]</h3>

<form enctype="multipart/form-data" @onsubmit="AddDrink" class="row g-3">
    <div class="mb-3">
        <img src="@_imageAvatar" @onclick="DisplayPhoto" height="150" width="auto" />
    </div>
    <div class ="mb-3">
        <label for="name" class="form-label">@localizer["Name"]</label>
        <input @bind="drink.DrinkItem.Name" placeholder="Name" class="form-control" />
    </div>
    <div class="mb-3">
        <label for="name" class="form-label">@localizer["Quantity"]</label>
        <input @bind="drink.DrinkItem.Quantity" placeholder="Quantity" class="form-control" />
    </div>
    <div class="mb-3">
        <label for="name" class="form-label">@localizer["Price"]</label>
        <input @bind="drink.DrinkItem.Price" placeholder="Price" class="form-control" />
    </div>
    <div class="mb-3">
        <div class="form-check">
            <input type="checkbox" @bind="drink.ContainsAlcohol" class="form-check-input" />
            <label class="form-check-label" for="containsAlcohol">@localizer["Contains Alcohol"]</label>
        </div>
    </div>
    <div class="mb-3">
        <button type="submit" class="btn btn-primary">@localizer["Save"]</button>
    </div>
</form>
```

10 Forma za dodavanje novog pića

Funkcija koja dodaje piće u bazu podataka zove se *AddDrink* (slika 11) i poziva se kada se klikne gumb „Spremi“. U toj funkciji dohvaća se token trenutno prijavljenog korisnika i stavlja se u Header dio HTTP zahtjeva. Ako je korisnik odabrao neku sliku, prije zahtjeva za kreiranje pića, šalje se zahtjev za kreiranje slike. Kada se dobije odgovor sa API-ja da je slika uspješno kreirana, šalje se novi zahtjev da se kreira piće uz to da je atribut pića *PhotoPath* onda ažuriran podacima o lokaciji gdje je slika uploadana. Ako korisnik ne priloži sliku, onda se za kreiranje pića ostavi prazan atribut *PhotoPath*. Nakon uspješnog kreiranja pića, ili slike i pića korisnika se preusmjeri na popis svih pića.

```
private async Task AddDrink()
{
    var token = authService.GetAccessToken();
    Http.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

    try
    {
        if (_imageBytes != null)
        {
            var formData = new MultipartFormDataContent();
            var imageContent = new ByteArrayContent(_imageBytes);
            formData.Add(imageContent, "receivedFile", photo.FileName);
            var responseFile = await Http.PostAsync(Constants.FileUrl, formData);

            if (responseFile.IsSuccessStatusCode)
            {
                var filePath = responseFile.Headers.Location.ToString();
                drink.DrinkItem.PhotoPath = filePath;

                var response = await Http.PostAsJsonAsync(Constants.DrinksUrl, drink);
                if (response.IsSuccessStatusCode)
                {
                    Navigation.NavigateTo("/drinks");
                }
            }
        }
        else
        {
            var response = await Http.PostAsJsonAsync(Constants.DrinksUrl, drink);
            if (response.IsSuccessStatusCode)
            {
                Navigation.NavigateTo("/drinks");
            }
        }
    }
    catch(Exception ex)
    {

    }
}
```

11 Funkcija za kreiranje novog pića i slike ili samo novog pića

Kokteli

Stranica za dodavanje koktela ima ulogu kreiranja novog koktela te uploada datoteke. Za dodavanje koktela potrebno je biti prijavljen korisnik pa se kod slanja zahtjeva za kreiranje na API šalje i token korisnika koji radi taj zahtjev, kao i kod kreiranja novog pića.

Kako se koktel sastoji od više pića, kod učitavanja stranice potrebno je prvo napraviti zahtjev da se dohvate sva pića (slika 12).

```
protected override async Task OnInitializedAsync()
{
    try
    {
        availableDrinks = await Http.GetFromJsonAsync<Drink[]>(Constants.DrinksUrl);
        isLoading = false;
    }
    catch(Exception ex)
    {

    }
}
```

12 Učitavanje svih pića

Forma za dodavanje novog koktela sadrži podatke o imenu, količini, cijeni i sastojcima koktela. Osim podataka vezanih uz koktel, također ima i odabir i prikaz slike koktela. Kod za prikaz forme nalazi se na slici 13.

```
<h3>@localizer["Create"]</h3>

    <form enctype="multipart/form-data" @onsubmit="AddCocktail">
        <div class="mb-3">
            <img src="@_imageAvatar" class="img-fluid" @onclick="DisplayPhoto" />
        </div>
        <div class="mb-3">
            <label for="name" class="form-label">@localizer["Name"]</label>
            <input id="name" class="form-control" @bind="cocktail.CocktailItem.Name" placeholder="Name" />
        </div>

        <div class="mb-3">
            <label for="quantity" class="form-label">@localizer["Quantity"]</label>
            <input id="quantity" class="form-control" disabled @bind="cocktail.CocktailItem.Quantity" placeholder="Quantity" />
        </div>

        <div class="mb-3">
            <label for="price" class="form-label">@localizer["Price"]</label>
            <input id="price" class="form-control" @bind="cocktail.CocktailItem.Price" placeholder="Price" />
        </div>

        <div class="card p-2" style="width: 80%;">
            <div class="mb-3">
                <select id="drink" class="form-select" @bind="selected" aria-label="Default select example">
                    <option selected>@localizer["Choose drink"]</option>
                    @foreach (var d in availableDrinks)
                    {
                        <option value="@d.Id">@d.DrinkItem.Name</option>
                    }
                </select>
            </div>
            <div>
                <label for="drinkQuantity" class="form-label">@localizer["Quantity"]</label>
                <input id="drinkQuantity" class="form-control" @bind="item.Quantity" placeholder="Drink Quantity" />
            </div>

            <div class="mb-3">
                <button type="button" class="btn btn-primary" @onclick="() => AddIngredient(selected)">+</button>
            </div>
        </div>
        <div class="card p-2" style="width: 80%;">
            @if (cocktail.Ingredients.Any())
            {
                <div>
                    <h5>@localizer["Ingredients picked"]</h5>
                    <ul>
                        @foreach (var i in cocktail.Ingredients)
                        {
                            <li>@availableDrinks.First(x => x.Id == i.DrinkId).DrinkItem.Name @i.Quantity ml</li>
                        }
                    </ul>
                </div>
            }
        </div>
        <div class="mb-3">
            <button type="submit" class="btn btn-primary">@localizer["Save"]</button>
        </div>
    </form>
```

13 Forma za dodavanje novog koktela

Funkcija koja dodaje koktel u bazu podataka zove se *AddCocktail* (slika 14), a poziva se kada se klikne gumb „Spremi“. Kreiranje novog koktela slijedi isti proces kao i kreiranje novog pića.

```
private async Task AddCocktail()
{
    try
    {
        foreach(var i in cocktail.Ingredients)
        {
            var drink = availableDrinks.FirstOrDefault(e => e.Id == i.DrinkId);
            if(drink.ContainsAlcohol)
            {
                cocktail.ContainsAlcohol = true;
                break;
            }
        }

        if(_imageBytes != null)
        {
            var formData = new MultipartFormDataContent();
            var imageContent = new ByteArrayContent(_imageBytes);
            formData.Add(imageContent, "receivedFile", photo.FileName);
            var responseFile = await Http.PostAsync(Constants.FileUrl, formData);

            if (responseFile.IsSuccessStatusCode)
            {
                var filePath = responseFile.Headers.Location.ToString();
                cocktail.CocktailItem.PhotoPath = filePath;

                var response = await Http.PostAsJsonAsync(Constants.CocktailsUrl, cocktail);
                if (response.IsSuccessStatusCode)
                {
                    Navigation.NavigateTo("/cocktails");
                }
            }
        }
        else
        {
            var response = await Http.PostAsJsonAsync(Constants.CocktailsUrl, cocktail);
            if (response.IsSuccessStatusCode)
            {
                Navigation.NavigateTo("/cocktails");
            }
        }
    }
    catch(Exception ex)
    {
    }
}
```

14 Funkcija za dodavanje novog koktela ili novog koktela i datoteke

Ono specifično za dodavanje novog koktela je izbor sastojaka i količine sastojaka koji idu u taj koktel. Dodavanje sastojaka radi se preko funkcije *AddIngredient* (slika 15) u koju se proslijedi *Id* pića. U listu *Ingredients* se zatim doda upisana količina i *Id* pića, te se atribut količina unutar koktela ažurira kao zbroj količina svih sastojaka. *DeleteItem* (slika 15) služi za brisanje sastojka iz koktela, kao parametar se proslijedi *Ingredient* te ako postoji u listi sastojaka koktela se izbriše iz nje.

```
private void AddIngredient(Guid id)
{
    item.DrinkId = id;
    cocktail.Ingredients.Add(item);
    cocktail.CocktailItem.Quantity = cocktail.Ingredients.Sum(i => i.Quantity);
    item = new Ingredient();
    StateHasChanged();
}

private void DeleteItem(Ingredient ingredient)
{
    var existingIngredient = cocktail.Ingredients.FirstOrDefault(i => i.DrinkId == ingredient.DrinkId && i.Quantity == ingredient.Quantity);
    if (existingIngredient != null)
    {
        cocktail.Ingredients.Remove(existingIngredient);
        StateHasChanged();
    }
}
```

15 Forma za dodavanje novog koktela

Narudžbe

Funkcija za dodavanje nove narudžbe nalazi se na *OrderPage* stranici koja se prikazuje na početnoj stranici kao dijalog. Kada se stranica otvori, iz *orderInMemory* se prikažu sva pića i

kokteli, a ukoliko postoji prijavljeni korisnik polje za unos emaila se popuni sa emailom prijavljenog korisnika.

Zbog preglednosti prikaza, ukoliko korisnik naruči više istih pića ili koktela njihovo ime se prikazuje jednom ali uz njih piše i količina koja se nalazi u košarici. To se odvija preko funkcija *CountDrinks* i *CountCocktails*(slika 16) koje spremaju Id pića ili koktela i količinu koliko puta se to piće ili koktel pojavljuje u listama *OrderedCocktails* ili *OrderedDrinks* u *orderInMemory*.

```
static Dictionary<Guid, int> CountDrinks(List<Drink> drinks)
{
    Dictionary<Guid, int> drinkCount = new Dictionary<Guid, int>();
    foreach (var drink in drinks)
    {
        if (drinkCount.ContainsKey(drink.Id))
            drinkCount[drink.Id]++;
        else
            drinkCount[drink.Id] = 1;
    }
    return drinkCount;
}

static Dictionary<Guid, int> CountCocktails(List<Cocktail> cocktails)
{
    Dictionary<Guid, int> cocktailCount = new Dictionary<Guid, int>();
    foreach (var cocktail in cocktails)
    {
        if (cocktailCount.ContainsKey(cocktail.Id))
            cocktailCount[cocktail.Id]++;
        else
            cocktailCount[cocktail.Id] = 1;
    }
    return cocktailCount;
}
```

16 Funkcije za zbrajanje količine stavki narudžbe

Za dodavanje narudžbe kreira se novi objekt narudžbe te se popunjava sa podacima iz *orderInMemory* i emailom korisnika ako je upisan (slika 17).

```
private async Task AddOrder()
{
    try
    {
        foreach (var o in orderInMemory.OrderedDrinks)
        {
            orderInMemory.Price += o.DrinkItem.Price;
        }
        foreach (var o in orderInMemory.OrderedCocktails)
        {
            orderInMemory.Price += o.CocktailItem.Price;
        }
        Order order = new Order()
        {
            Id = Guid.NewGuid(),
            DrinksOrdered = orderInMemory.OrderedDrinks,
            CocktailsOrdered = orderInMemory.OrderedCocktails,
            Price = orderInMemory.Price,
            UserEmail = userEmail,
            OrderId= $"#{Guid.NewGuid().ToString("N").Substring(0, 6)}"
        };
        var response = await Http.PostAsJsonAsync(Constants.OrderUrl, order);

        if(response.IsSuccessStatusCode && userEmail != string.Empty)
        {
            GenerateEmailText();
            email.Subject = $"Order {order.OrderId}";
            email.To = userEmail;
            email.Body = emailBody;
            var emailResponse = await Http.PostAsJsonAsync(Constants.EmailUrl, email);
        }
        orderInMemory = new OrderInMemory();
        StateHasChanged();
    }
    catch(Exception ex)
    {
    }
}
```

17 Funkcija za dodavanje narudžbe

Na uspješan zahtjev za kreiranje narudžbe, ako je email korisnika upisan generira se sadržaj maila preko funkcije *GenerateEmailText*, dodaje se naslov, sadržaj i primatelj maila te se šalje zahtjev na API kako bi se taj mail poslao primatelju. *GenerateEmailText* (slika 18) iterira kroz sve stavke narudžbe i prikazuje njihov naziv i cijenu te to dodaje u varijablu *emailBody* koja se kod slanja zahtjeva na API koristi kao sadržaj maila.

```
private void GenerateEmailText()
{
    var items = "<ul>";
    @foreach (var d in orderInMemory.OrderedDrinks)
    {
        items += $"<li>{d.DrinkItem.Name} {d.DrinkItem.Price}€</li>";
    }
    @foreach (var c in orderInMemory.OrderedCocktails)
    {
        items += $"<li>{c.CocktailItem.Name} {c.CocktailItem.Price}€</li>";
    }

    emailBody = "<h2>Stavke narudžbe</h2><hr/> " + items + "</ul><hr/><strong></br>Hvala na kupnji!";

}
```

18 Funkcija za generiranje teksta emaila

Stranice za ažuriranje

Kod učitavanja stranica za ažuriranje radi zahtjev na API (slika 19) kojim dobivamo podatke iz baze podataka za objekt koji ažuriramo.

```
protected override async Task OnInitializedAsync()
{
    try
    {
        var token = authService.GetAccessToken();
        Http.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
        drink = await Http.GetFromJsonAsync<Drink>($"{Constants.DrinksUrl}/{id}");
        isLoading = false;
    }
    catch (Exception ex)
    {
        exc = ex.Message;
    }
}
```

19 Funkcija koja se poziva kod učitavanja stranice za uređivanje pića

Stranice za ažuriranje dostupne su za pića i koktele, dijele istu logiku i kod kao i stranice za kreiranje uz razliku da se kod slanja zahtjeva za ažuriranjem objekta u bazi proslijedi i njegov *Id* (slika 20).

```
protected override async Task OnInitializedAsync()
{
    try
    {
        var token = authService.GetAccessToken();
        Http.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
        cocktail = await Http.GetFromJsonAsync<Cocktail?>($"{Constants.CocktailsUrl}/{id}");
        availableDrinks = await Http.GetFromJsonAsync<Drink[]>(Constants.DrinksUrl);
        isLoading = false;
    }
    catch(Exception ex)
    {
        exc = ex.Message;
    }
}
```

20 Funkcija koja se poziva kod učitavanja stranice za uređivanje koktela

Stranice za čitanje skupa podatka

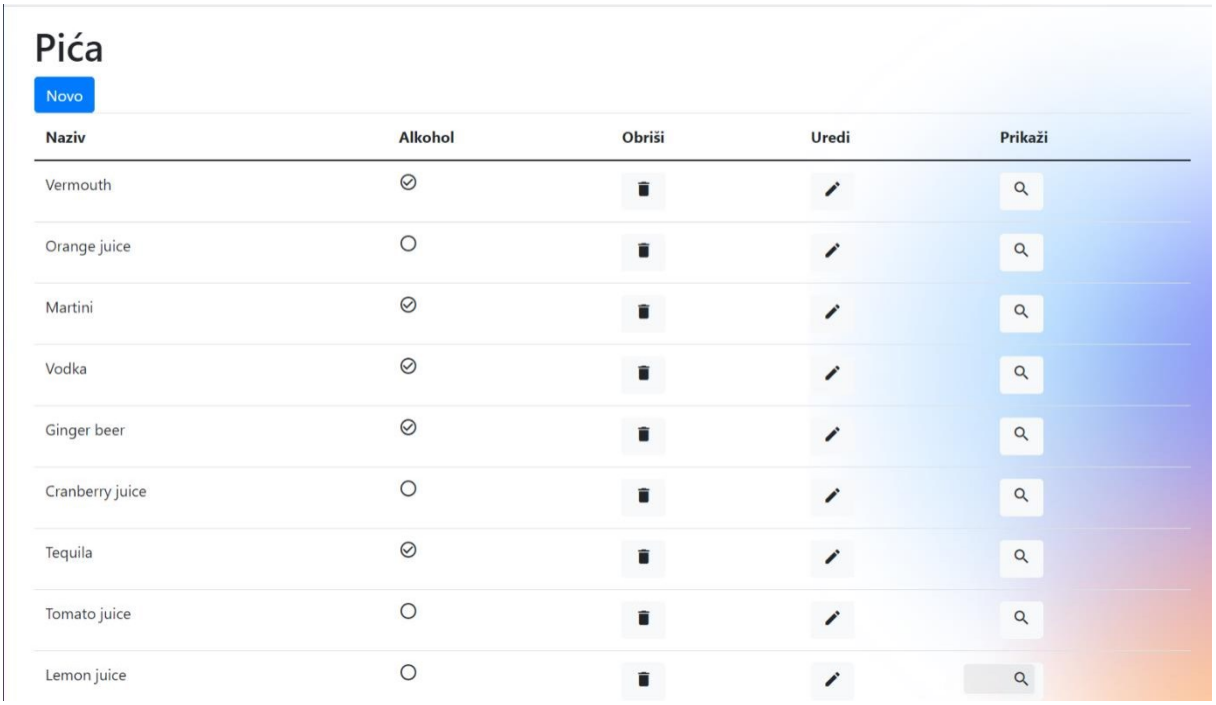
Kod prikaza svih podataka dostupnih u bazi podataka za pića, koktele ili narudžbe sve stranice slijede sličan proces. Kod učitavanja stranice radi se zahtjev na endpoint ovisno o entitetu koji želimo prikazati. Na primjer, za prikaz svih koktela radimo zahtjev na endpoint Cocktails (slika 21)

```
protected override async Task OnInitializedAsync()
{
    try
    {
        cocktails = await Http.GetFromJsonAsync<Cocktail?[]>(url);
    }
    catch (Exception ex)
    {
        exc = ex.Message;
    }

    isLoading = false;
}
```

21 Učitavanje stranice za prikaz svih koktela

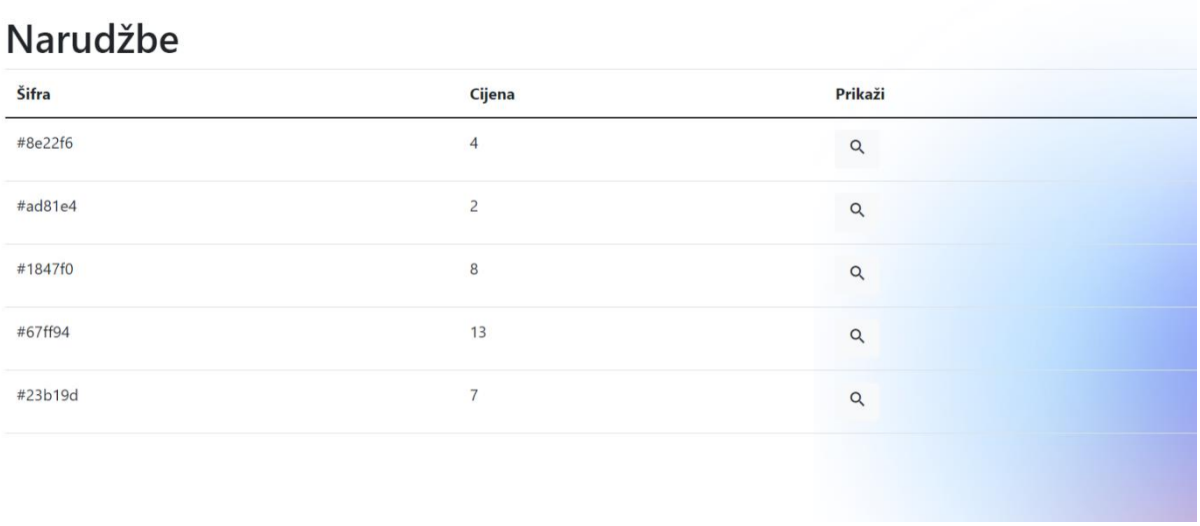
Za prikaz dobivenih informacija koristi se tablica. Ovisno o tome je li moguće uređivati ili brisati entitet, u tablici su prikazane i ikone koje omogućuju brisanje, uređivanje ili prikaz odabranog retka tablice. Kod pića i koktela moguće je brisanje, uređivanje i prikaz pojedinačnih , dok je za narudžbe moguć samo prikaz pojedinačne.

Primjer izgleda stranice za koktele i pića (slika 22).



22 Stranica za prikaz svih pića

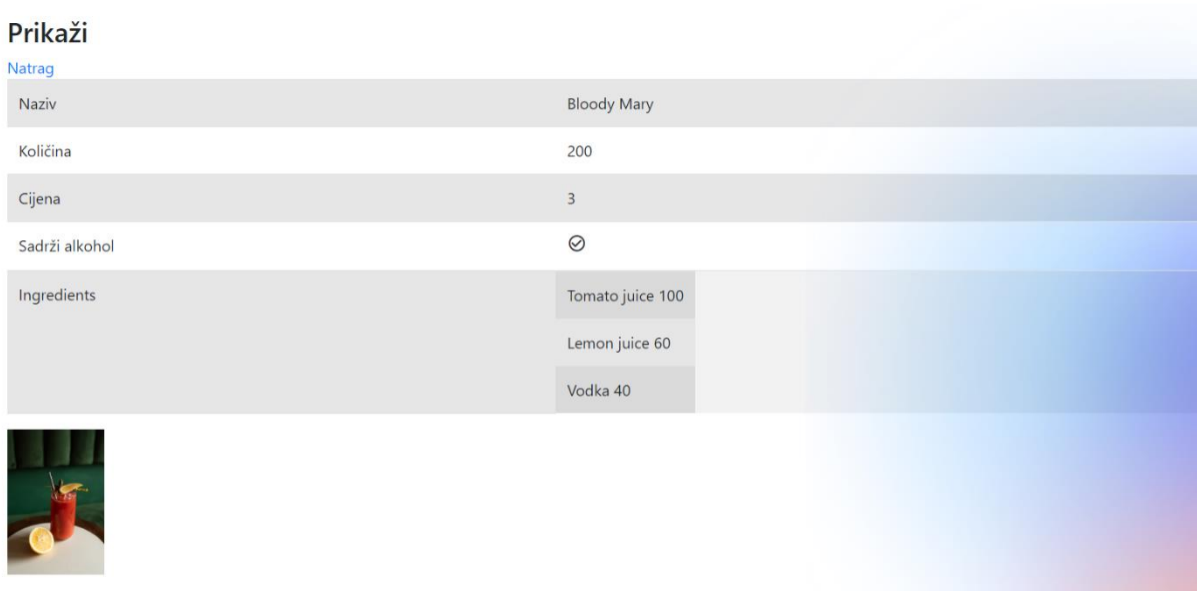
Prikaz izgleda narudžbi (slika 23)



23 Stranica za prikaz svih narudžbi ili narudžbi prijavljenog korisnika

Stranice za prikaz jednog entiteta

U mapama po entitetima *Read* stranice koriste se za prikaz jednog objekta. Prikaz pojedinog koktela, pića ili narudžbe slijedi logiku stranica za ažuriranje. Po proslijeđenom *Id*-u dobivamo zapis iz baze podataka preko API-ja, te ga onda prikazujemo. Kako na *Read* stranicama nema mogućnosti uređivanja, za prikaz se ne koristi forma već tablični prikaz (slika 24).



24 Prikaz jednog koktela

Dodavanje platformski specifičnog koda

MSAL Client

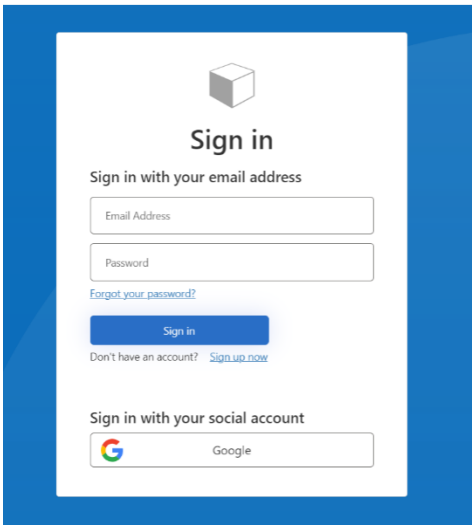
Kako bi se korisnici mogli prijaviti u aplikaciju, potrebno je dodati MSAL Client sa podacima koji se nalaze na Azure Active Directory B2C. U Azure AD B2C potrebno je napraviti registraciju aplikacije i izabrati postavke i providere koje želimo imati u aplikaciji.

U MAUI aplikaciji dodajemo potrebne podatke kako bi se spojili sa providerima koje definiramo u Identity Providers na Azure AD B2C

Mogućnost prijave se definira pod User Flows, u MAUI aplikaciji omogućavamo user flow B2C_1_social_susi, u kojem smo naveli providere koje želimo koristiti i podatke koje dobivamo nakon uspješne prijave korisnika kao što je email i ime i prezime.

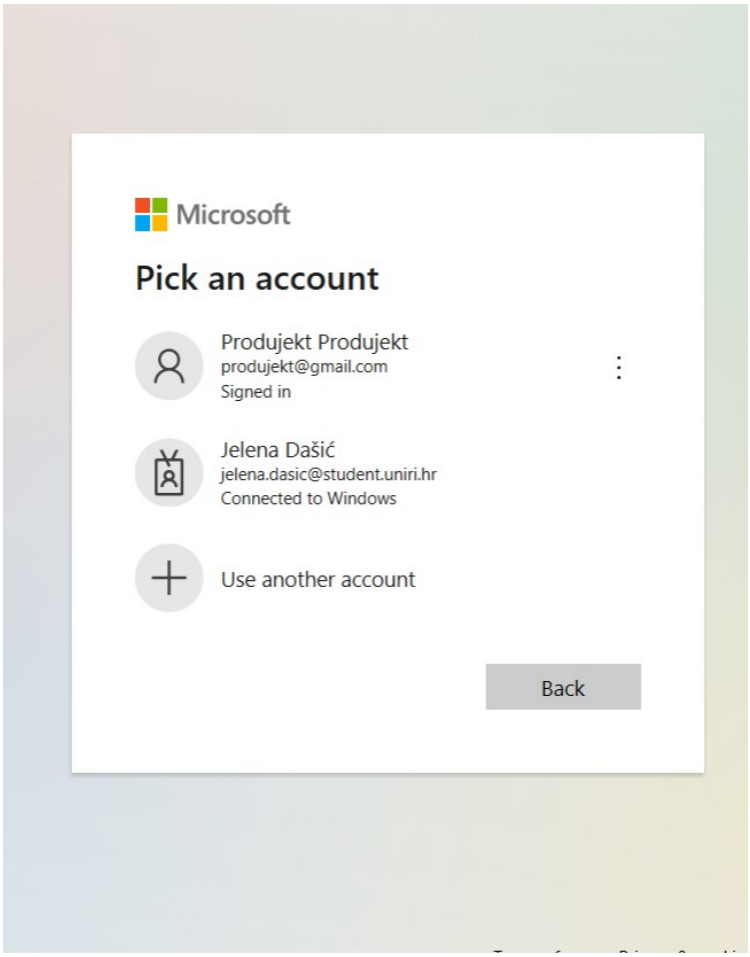
Kako bi u aplikaciji bilo moguće koristiti MSAL, potrebno je dodati nove klase u kojima su definirane postavke i šifre koje se nalaze na Azure AD B2C, te postavke specifične za platformu. Za mogućnost prijave i dobivanja potrebnih informacija iz korisničkog sučelja, dodana je AuthService.cs datoteka. U njoj su definirane metode za prijavu i odjavu korisnika. Kod za MSAL.Client, AuthService i platformsku implementaciju za Android je preuzet sa https://github.com/carlfranklin/MsalGoogleAuthInMaui/ .

Ekran prijave za korisnike nalazi se na slici 25.



25 Ekran prijave za korisnike

Ekran prijave za administratore nalazi se na slici 26.



26 Ekran prijave za administratore

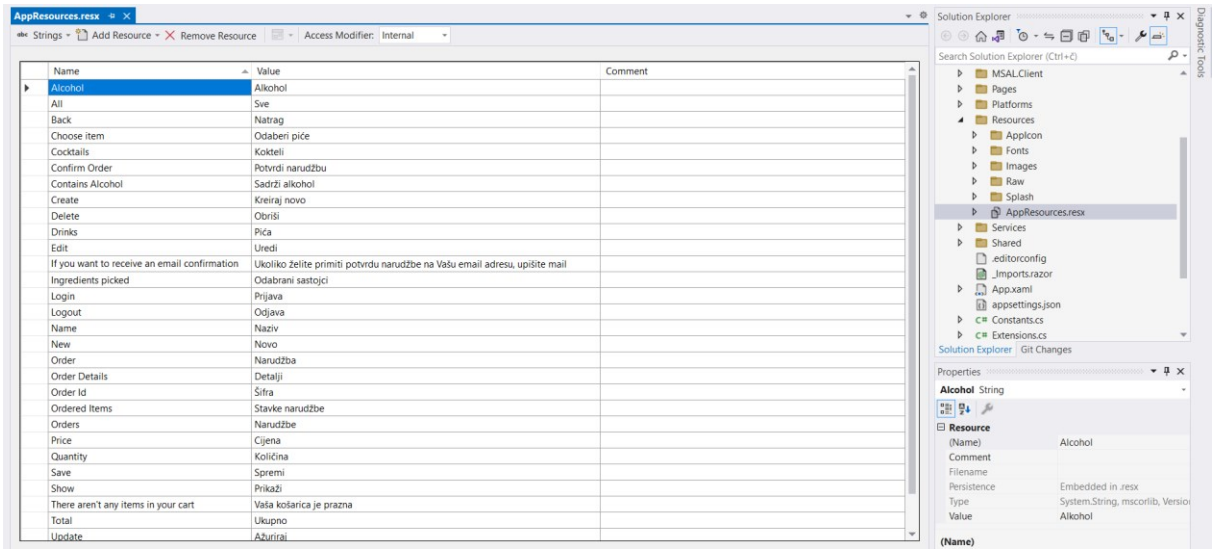
Dvojezičnost

Za implementaciju dvojezičnosti u aplikaciji koristi se paket *Microsoft.Exstensions.Localization*. U *MauiProgram.cs* registriramo preuzeti paket kao uslugu (slika 27).

```
builder.Services.AddLocalization();
```

27 Dodavanje paketa za prijevode kao usluge

Zatim prijevode definiramo u AppResources.resx (slika 28), datoteci koja je dodana u mapu Resources.

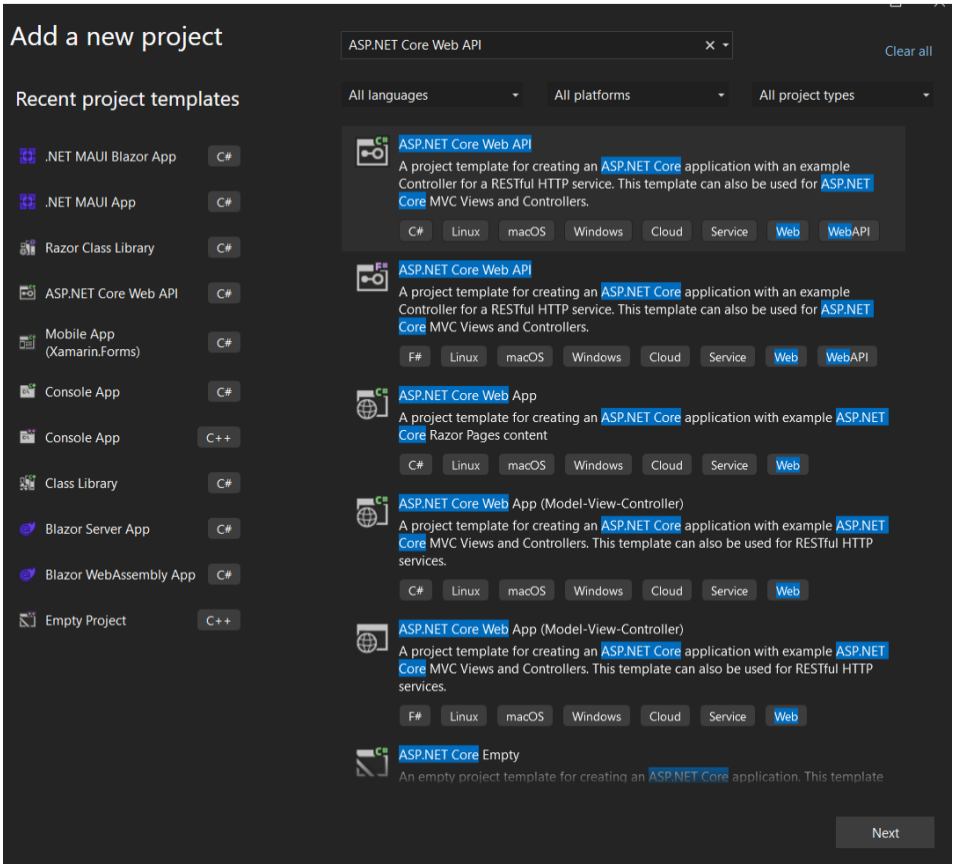


28 Datoteka AppResources.resx

Potom u svakoj stranici koju prevodimo dodajemo IStringLocalizer i prijevode koje želimo. U slučaju da neki prijevod ne postoji, pokazati će se tekst na engleskom jeziku.

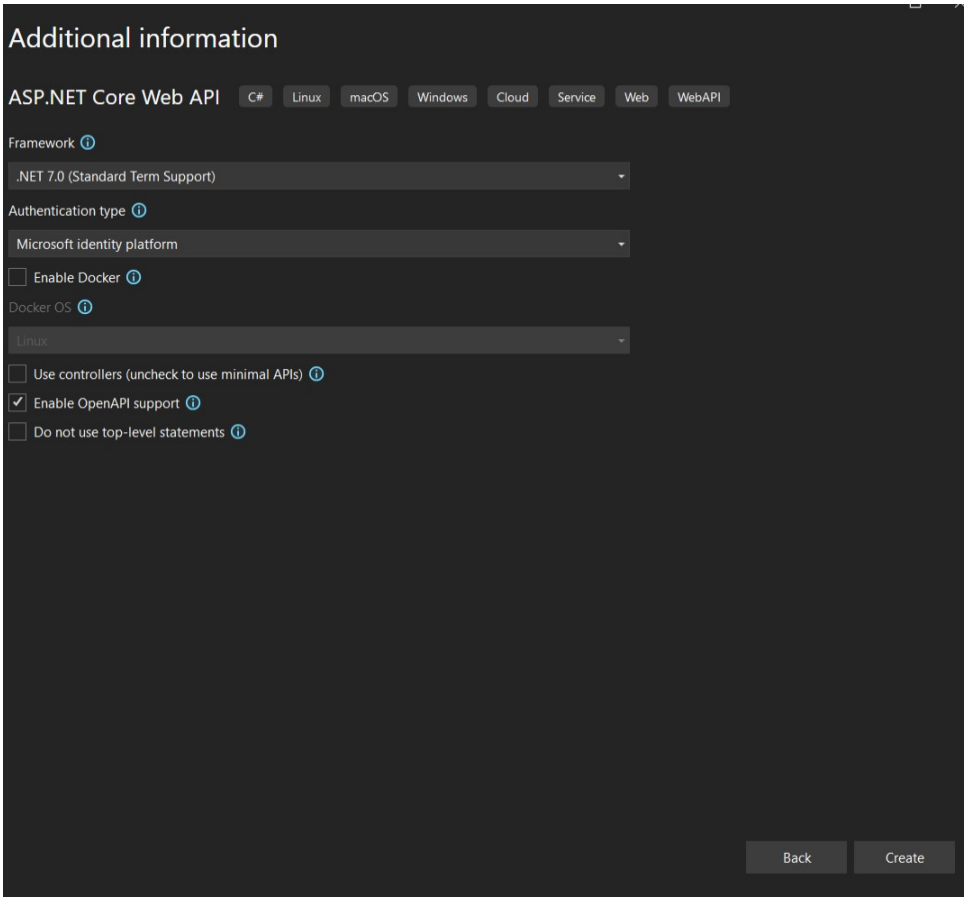
Izrada API

Kreiranje projekta za ASP.NET Core Web API



29 Odabir projekta

Kod kreiranja projekta, odabiremo opciju za kreiranje minimal API-ja (slika 30)



30 Opcija za kreiranje minimal API-ja

Kreiranje API Endpointa

Za kreiranje API Endpointa koristiti će se IEndpointRouteHandler i IEndpointRouteHandlerExtensions. Na ovaj način će se omogućiti da sve klase koje naslijeđuju IEndpointRouteHandler automatski imaju definirane endpointe registrirane u Program.cs datoteci. Tako se smanjuje količina koda koja bi bila u Program.cs (slika 31) datoteci kod korištenja Minimal API te umjesto definiranja svakog endpointa zasebno treba dodati samo jednu liniju koda u Program.cs datoteku.

```
app.MapEndpoints(Assembly.GetExecutingAssembly());
```

31 Dodavanje endpointa u Program.cs datoteku

Prednost ovog pristupa je da se smanjuje dužina Program.cs datoteke. Osim toga jednostavnije je dodati nove endpointe jer je potrebno samo naslijediti IEndpointRouteHandler umjesto dodavanja svake operacije posebno u Program.cs. Ovime poboljšavamo i čitljivost koda, jer su operacije i rute na API odvojene po klasama koje postoje.

IEndpointRouteHandler

Interface je deklaracija u kojoj se nalazi skup metoda koje implementira klasa koja koristi taj interface. U ovom slučaju interface IEndpointRouteHandler ima metodu MapEndpoints (slika 32). IEndpointRouteHandlerExtension (slika 33) prikazuje implementaciju metode MapeEndpoints.

```
namespace ProdujektAPI.Routing;

7 references
public interface IEndpointRouteHandler
{
    6 references
    public void MapEndpoints(IEndpointRouteBuilder app);
}
```

32 Datoteka IEndpointRouteHandler

IEndpointRouteHandlerExtensions

```
using System.Reflection;

namespace ProdujektAPI.Routing;

0 references
public static class IEndpointRouteBuilderExtensions
{
    1 reference
    public static void MapEndpoints(this IEndpointRouteBuilder app, Assembly assembly)
    {
        ArgumentNullException.ThrowIfNull(app);
        ArgumentNullException.ThrowIfNull(assembly);

        var endpointRouteHandlerInterfaceType = typeof(IEndpointRouteHandler);

        var endpointRouteHandlerTypes = assembly.GetTypes().Where(t =>
            t.IsClass && !t.IsAbstract && !t.IsGenericType
            && t.GetConstructor(Type.EmptyTypes) != null
            && endpointRouteHandlerInterfaceType.IsAssignableFrom(t));

        foreach (var endpointRouteHandlerType in endpointRouteHandlerTypes)
        {
            var instantiatedType = (IEndpointRouteHandler)Activator.CreateInstance(endpointRouteHandlerType)!;
            instantiatedType.MapEndpoints(app);
        }
    }
}
```

33 Datoteka IEndpointRouteBuilderExtensions

Drink

Endpoint Drink implementira GET, POST, PUT i DELETE metode. Za POST, PUT i DELETE potrebno je biti prijavljen korisnik. Svaka metoda koristi IDrinkService, što je interface u kojem su deklarirane metode iz DrinkService koji se koristi za komunikaciju s bazom podataka nad kolekcijom Drink (slika 34).

```
using ProdujektAPI.Models;
using ProdujektAPI.Routing;
using ProdujektAPI.Services;

namespace ProdujektAPI.Handlers;

0 references
public class DrinksHandler : IEndpointRouteHandler
{
    2 references
    public void MapEndpoints(IEndpointRouteBuilder app)
    {
        app.MapGet("/api/drinks", GetList);
        app.MapGet("/api/drinks/{id:guid}", Get);
        app.MapPost("/api/drinks", Insert)
            .RequireAuthorization();
        app.MapPut("/api/drinks/{id:guid}", Update)
            .RequireAuthorization();
        app.MapDelete("/api/drinks/{id:guid}", Delete)
            .RequireAuthorization();
    }
    1 reference
    private static Task<List<Drink>> GetList(IDrinkService drinkService) { return drinkService.GetAsync(); }
    1 reference
    private static Task<Drink?> Get(Guid id, IDrinkService drinkService) { return drinkService.GetAsync(id); }
    1 reference
    private static Task Insert(Drink drink, IDrinkService drinkService)
    {
        drink.Created = DateTime.Now;
        drink.LastUpdated = DateTime.Now;
        return drinkService.CreateAsync(drink);
    }
    1 reference
    private static Task Update(Guid id, Drink drink, IDrinkService drinkService)
    {
        drink.LastUpdated = DateTime.Now;
        return drinkService.UpdateAsync(id, drink);
    }
    1 reference
    private static Task Delete(Guid id, IDrinkService drinkService) { return drinkService.RemoveAsync(id); }
}
```

34 Datoteka DrinksHandler.cs

Cocktail

Endpoint Cocktail implementira GET, POST, PUT i DELETE metode. Isto kao i kod endpointa Drink, za POST, PUT i DELETE potrebno je biti prijavljen korisnik. Svaka metoda kao parametar ima ICocktailService, interface u kojem su deklarirane metode iz Cocktail koji se koristi za komunikaciju s bazom podataka nad kolekcijom Cocktail (slika 35).

```
using ProdujektAPI.Models;
using ProdujektAPI.Routing;
using ProdujektAPI.Services;

namespace ProdujektAPI.Handlers;

0 references
public class CocktailsHandler : IEndpointRouteHandler
{
    2 references
    public void MapEndpoints(IEndpointRouteBuilder app)
    {
        app.MapGet("/api/cocktails", GetList);
        app.MapGet("/api/cocktails/{id:guid}", Get);
        app.MapPost("/api/cocktails", Insert)
            .RequireAuthorization();
        app.MapPut("/api/cocktails/{id:guid}", Update)
            .RequireAuthorization();
        app.MapDelete("/api/cocktails/{id:guid}", Delete)
            .RequireAuthorization();
    }
    1 reference
    private static Task<List<Cocktail>> GetList(ICocktailService cocktailService) { return cocktailService.GetAsync(); }
    1 reference
    private static Task<Cocktail?> Get(Guid id, ICocktailService cocktailService) { return cocktailService.GetAsync(id); }
    1 reference
    private static Task Insert(Cocktail cocktail, ICocktailService cocktailService)
    {
        cocktail.Created = DateTime.Now;
        cocktail.LastUpdated = DateTime.Now;
        return cocktailService.CreateAsync(cocktail);
    }
    1 reference
    private static Task Update(Guid id, Cocktail cocktail, ICocktailService cocktailService)
    {
        cocktail.LastUpdated = DateTime.Now;
        return cocktailService.UpdateAsync(id, cocktail);
    }
    1 reference
    private static Task Delete(Guid id, ICocktailService cocktailService) { return cocktailService.RemoveAsync(id); }
}
```

35 Datoteka CocktailsHandler.cs

Order

Endpoint Order implementira metode GET i POST. Za ovaj endpoint potrebno je biti prijavljen korisnik samo za metodu GET. Svaka metoda kao parametar ima IOrderService, interface u kojem su deklarirane metode iz OrderService koji se koristi za komunikaciju s bazom podataka nad kolekcijom Order (slika 36).

```
using ProdujektAPI.Models;
using ProdujektAPI.Routing;
using ProdujektAPI.Services;

namespace ProdujektAPI.Handlers;

0 references
public class OrdersHandler : IEndpointRouteHandler
{
    2 references
    public void MapEndpoints(IEndpointRouteBuilder app)
    {
        app.MapGet("/api/orders", GetList);
        app.MapGet("/api/orders/{id:guid}", Get)
            .RequireAuthorization()
        app.MapPost("/api/orders", Insert);
        app.MapDelete("/api/orders/{id:guid}", Delete);
    }
    1 reference
    private static Task<List<Order>> GetList(IOrderService orderService, string Email = null) { return orderService.GetAsync(Email);
    1 reference
    private static Task<Order?> Get(Guid id, IOrderService orderService) { return orderService.GetAsync(id); }
    1 reference
    private static Task Insert(Order order, IOrderService orderService)
    {
        order.Created = DateTime.Now;
        order.LastUpdated = DateTime.Now;
        return orderService.CreateAsync(order);
    }
    1 reference
    private static Task Delete(Guid id, IOrderService orderService) { return orderService.RemoveAsync(id); }
}
```

36 Datoteka OrdersHandler.cs

Email

Endpoint Email (slika 37) ima metodu POST. Kako se ovaj endpoint koristi isključivo za slanje mailova, jedina potrebna funkcionalnost je kreiranje maila. Za razliku od prijašnjih endpointa, IEmailService se ne koristi za komunikaciju sa bazom podataka, već za kreiranje i slanje maila. Proces izrade i slanja maila će kasnije biti opisan u radu.

```
using ProdujektAPI.Dto;
using ProdujektAPI.Routing;
using ProdujektAPI.Services;

namespace ProdujektAPI.Handlers;


public class EmailHandler : IEndpointRouteHandler
{

    public void MapEndpoints(IEndpointRouteBuilder app)
    {
        app.MapPost("/api/emails", Insert);
    }
    1 reference
    private static void Insert(EmailDto email, IEmailService emailService) { emailService.SendEmail(email); }
}
```

37 Datoteka EmailHandler.cs

FileUpload

Kako endpoint *FileUpload* služi za stavljanje datoteka na server i komunikaciju s bazom podataka, *GetList* (slika 38) vraća putanje do svih datoteka koje su na serveru.

```
public class FileUploadHandler : IEndpointRouteHandler
{
    2 references
    public void MapEndpoints(IEndpointRouteBuilder app)
    {
        app.MapGet("/api/files", GetList);
        app.MapGet("/api/files/{id:guid}", Get);
        app.MapPost("/api/files", Insert);
        app.MapDelete("/api/files/{id:guid}", Delete);
    }
    1 reference
    private static async Task<List<string>> GetList(IFileUploadService fileUploadService, HttpContext ctx, IWebHostEnvironment _env)
    {
        List<string> files = new List<string>();

        try
        {
            DirectoryInfo dirInfo = new DirectoryInfo($"{_env.ContentRootPath}\\Images\\");
            var req = ctx.Request;
            var baseUrl = $"{req.Scheme}://{req.Host}";
            if (dirInfo.Exists)
            {
                foreach (FileInfo fInfo in dirInfo.GetFiles())
                {
                    files.Add(baseUrl + "/Images/" + fInfo.Name);
                }
            }
        }
        catch (Exception ex)
        {

        }
        return files.ToList();
    }
```

38 Datoteka FileUploadHandler.cs – funkcija GetList

Get (slika 39) vraća putanju do datoteke čiji smo *Id* proslijedili.

```
private static async Task<string> Get(Guid id, IFileUploadService fileUploadService, HttpContext ctx, IWebHostEnvironment _env)
{
    var files = string.Empty;

    try
    {
        DirectoryInfo dirInfo = new DirectoryInfo($"{_env.ContentRootPath}\\Images\\");
        var req = ctx.Request;
        var baseUrl = $"{req.Scheme}://{req.Host}";
        if (dirInfo.Exists)
        {
            foreach (FileInfo fInfo in dirInfo.GetFiles())
            {
                if (fInfo.Name.Contains(id.ToString()))
                    files = baseUrl + "/Images/" + fInfo.Name;
            }
        }
    }
    catch (Exception ex)
    {

    }
    return files;
}
```

39 Datoteka FileUploadHandler.cs – funkcija Get

Insert (slika 40) kreira novi zapis u bazi podataka i uploada datoteku u mapu *Images* na API-ju

```
private static async Task<IResult> Insert(IFormFile receivedFile, IFileUploadService fileUploadService,
    IWebHostEnvironment _env, HttpContext ctx)
{
    try
    {
        FileUpload file = new FileUpload
        {
            Id = Guid.NewGuid(),
            OriginalName = ContentDispositionHeaderValue
            .Parse(receivedFile.ContentDisposition)
            .FileName.Trim('"'),
            Created = DateTime.Now,
            LastUpdated = DateTime.Now,
        };

        var fName = file.Id.ToString() + Path.GetExtension(receivedFile.FileName);
        var root = $"{_env.ContentRootPath}\\Images";
        if (!Directory.Exists(root))
        {
            Directory.CreateDirectory(root);
        }

        var path = $"{root}\\{fName}";
        var fs = System.IO.File.Create(path);
        await receivedFile.CopyToAsync(fs);
        fs.Close();

        DirectoryInfo dirInfo = new DirectoryInfo($"{_env.ContentRootPath}\\Images\\");
        var req = ctx.Request;
        var baseUrl = $"{req.Scheme}://{req.Host}";

        return Results.Created($"{baseUrl}/Images/{fName}", file);
    }
    catch (Exception ex)
    {
        return Results.Problem(ex.ToString());
    }
}
```

40 Datoteka FileUploadHandler.cs – funkcija Insert

Delete (slika 41) služi za brisanje datoteke iz mape *Images* i iz baze podataka.

```
1 reference
private static async Task<IResult> Delete(Guid id, IFileUploadService fileUploadService, IWebHostEnvironment _webHostEnvironment)
{
    if (id.ToString() != string.Empty)
    {
        try
        {
            string file = Path.Combine(_webHostEnvironment.WebRootPath, "\\Images\\", id.ToString());
            System.IO.File.Delete(file);
            await fileUploadService.RemoveAsync(id);
            return Results.NotFound();
        }
        catch (Exception ex)
        {
            return Results.Problem(ex.ToString());
        }
    }
    return Results.Problem();
}
```

41 Datoteka FileUploadHandler.cs – funkcija Delete

Modeli baze podataka

U bazi podataka postoje kolekcije za koktele, pića, datoteke i narudžbe. Osim modela za entitete u bazi podataka, u kodu postoje *OrderItem*, *Ingredient* i *Email* modeli.

OrderItem

OrderItem (slika 42) ima podatke o atributima zajedničkim pićima i koktelima. To su podaci o imenu, cijeni, količini i slici pića ili koktela.

Name je naziv koktela ili pića.

Quantity količina pića ili koktela u mililitrima.

Price je cijena koktela ili pića.

PhotoPath je putanja do slike koktela ili pića na serveru.

```
namespace ProdujektAPI.Models;

4 references
public class OrderItem
{
    0 references
    public string Name { get; set; }
    0 references
    public int Quantity { get; set; }
    0 references
    public float Price { get; set; }
    0 references
    public string PhotoPath { get; set; }
}
```

42 Datoteka OrderItem.cs

Ingredient

Model *Ingredient* (slika 43) služi kako bi se u koktel moglo dodati piće i količina istog.

Quantity predstavlja količinu pića u mililitrima.

DrinkId predstavlja *Id* pića koje je sastojak koktela.

```
namespace ProdujektAPI.Models;

1 reference
public class Ingredient
{
    0 references
    public int Quantity { get; set; }
    0 references
    public Guid DrinkId { get; set; }
}
```

43 Datoteka Ingredients.cs

Drink

Model pića (slika 44) predstavlja podatke o pojedinačnim pićima. Osim podataka koji se nalaze u *OrderItem*, *Drink* sadrži podatke o tome da li piće sadrži alkohol.

Id je jedinstveni identifikator pića.

ContainsAlcohol označava ako piće ima alkohol ili ne,

Created označava vrijeme i datum kada je piće kreirano.

LastUpdated označava vrijeme i datum kada je piće zadnji put ažurirano.

```
namespace ProdujektAPI.Models;

public class Drink
{

    public Guid Id { get; set; }

    public bool ContainsAlcohol { get; set; }
    0 references
    public OrderItem DrinkItem { get; set; }
    1 reference
    public DateTime? Created { get; set; }
    2 references
    public DateTime? LastUpdated { get; set; }
}
```

44 Datoteka Drink.cs

Cocktail

Model koktela (slika 45) predstavlja podatke o pojedinačnim koktelima. Osim podataka koji se nalaze u OrderItem, Cocktail sadrži podatke da li koktel sadrži alkohol, te sastojke od kojih se sadrži.
Id je jedinstveni identifikator koktela.

Ingredients su sastojci i količine od kojih su u koktelu.

CocktailItem

ContainsAlcohol

Created je vrijeme kada je koktel kreiran.

LastUpdated je vrijeme kada je koktel zadnji put ažuriran.

```
namespace ProdujektAPI.Models;

15 references
public class Cocktail
{
    3 references
    public Guid Id { get; set; }
    0 references
    public ICollection<Ingredient> Ingredients { get; set; }
    0 references
    public OrderItem CocktailItem { get; set; }
    0 references
    public bool ContainsAlcohol { get; set; }
    1 reference
    public DateTime? Created { get; set; }
    2 references
    public DateTime? LastUpdated { get; set; }
}
```

45 Datoteka Cocktail.cs

Order

Klasa Order (slika 46) sadrži podatke o cijeni, šifri narudžbe, naručenim koktelima i naručenim pićima.

Id je jedinstveni identifikator narudžbe.

Price označava ukupnu cijenu narudžbe.

OrderId je skraćeni identifikator narudžbe, služi za prikaz *Id*-a korisniku.

DrinksOrdered je lista naručenih pića.

CocktailsOrdered je lista naručenih koktela.

UserEmail je identifikator osobe koja je napravila narudžbu.

Created je vrijeme kada je narudžba kreirana.

LastUpdated je vrijeme kada je narudžba zadnji put ažurirana, u aplikaciji ne postoji mogućnost uređivanja narudžbe.

```
namespace ProdujektAPI.Models;

13 references
public class Order
{
    3 references
    public Guid Id { get; set; }
    0 references
    public double Price { get; set; }
    0 references
    public string OrderId { get; set; }
    0 references
    public List<Drink> DrinksOrdered { get; set; }
    0 references
    public List<Cocktail> CocktailsOrdered { get; set; }
    1 reference
    public string UserEmail { get; set; }
    1 reference
    public DateTime? Created { get; set; }
    1 reference
    public DateTime? LastUpdated { get; set; }
}
```

46 Datoteka Order.cs

FileUpload

Klasa FileUpload sadrži podatke o datoteci, kao što su identifikator i originalno ime datoteke (slika 47).

Id je jedinstveni identifikator narudžbe.

OriginalName označava ukupnu cijenu narudžbe.

Created je vrijeme kada je datoteka uploadana.

LastUpdated je vrijeme kada je datoteka zadnji put ažurirana. U aplikaciji ne postoji mogućnost ažuriranja datoteke.

```
namespace ProdujektAPI.Models;

12 references
public class FileUpload
{
    5 references
    public Guid Id { get; set; }
    1 reference
    public string OriginalName { get; set; }
    1 reference
    public DateTime? Created { get; set; }
    1 reference
    public DateTime? LastUpdated { get; set; }
}
```

47 Datoteka FileUpload.cs

Services

Mapa *Services* sadrži klase i interface za svaki od entiteta koji se sprema u bazu podataka, te opciju slanja maila.

Za entitete *Drink*, *Order*, *Cocktail* i *FileUpload*, services služe za unos podataka u bazu, dok za email služe za slanje mailova preko paketa *MailKit* paketa.

Kod unosa u bazu koristi se paket *MongoDb.Driver* preko kojeg je moguće definirati funkcije *GetAsync*, *CreateAsync*, *UpdateAsync* i *RemoveAsync* u kojima se preko metoda *Find*, *InsertOneAsync*, *ReplaceOneAsync* i *DeleteOneAsync* mogu raditi zahtjevi nad kolekcijom u bazi podataka.

GetAsync je funkcija koja može vratiti ili listu podataka ili jedan podatak ovisno o tome jesmo li proslijedili identifikator traženog entiteta. Preko *Find* funkcije dobivamo sve zapise iz baze podataka, te vraćamo ili sve ili onaj čiji smo identifikator proslijedili.

U *CreateAsync* funkciji prosljeđujemo objekt tipa kojeg želimo kreirati. Zatim se preko *InsertOneAsync* metode ažuriraju podaci u bazi podataka.

U *UpdateAsync* funkciji prosljeđujemo objekt tipa kojeg želimo ažurirati i identifikator objekta u bazi podataka. Zatim se preko *ReplaceAsync* funkcije ažurira objekt u bazi podataka.

RemoveAsync funkciji prosljeđujemo *Id* objekta kojeg želimo izbrisati. Preko *DeleteOneAsync* funkcije objekt sa *Id*-om koji smo proslijedili se briše iz baze podataka.

OrderService

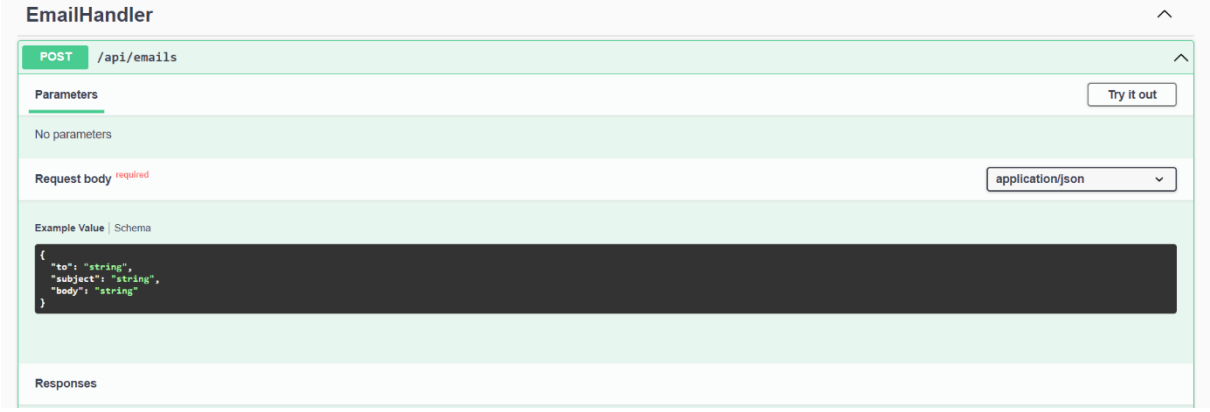
Primjer korištenja navedenih funkcija za operacije nad bazom podataka pokazati će se na *OrderService* (slika 48). Uz prije navedene mogućnosti, funkcija *GetAsync* može se modificirati tako da nalazi sve dokumente u bazi podataka po nekom parametru. U ovom slučaju, funkciji *GetAsync* prosljeđuje se parametar email, koji se koristi kako bi se našle sve narudžbe koje imaju proslijeđeni mail.

```
namespace ProdujektAPI.Services;

2 references
public class OrderService : IOrderService
{
    private readonly IMongoCollection<Order> _ordersCollection;

    0 references
    public OrderService(
        IOptions<ProdujektDatabaseSettings> produjektDatabaseSettings)
    {
        var mongoClient = new MongoClient(
            produjektDatabaseSettings.Value.ConnectionString);

        var mongoDatabase = mongoClient.GetDatabase(
            produjektDatabaseSettings.Value.DatabaseName);

        _ordersCollection = mongoDatabase.GetCollection<Order>(
            produjektDatabaseSettings.Value.OrdersCollectionName);
    }

    2 references
    public async Task<List<Order>> GetAsync(string Email)
    {
        if (Email == null)
            return await _ordersCollection.Find(_ => true).ToListAsync();

        return await _ordersCollection.Find(x => x.UserEmail == Email).ToListAsync();
    }

    2 references
    public async Task<Order?> GetAsync(Guid id) =>
        await _ordersCollection.Find(x => x.Id == id).FirstOrDefaultAsync();

    2 references
    public async Task CreateAsync(Order newOrder) =>
        await _ordersCollection.InsertOneAsync(newOrder);

    1 reference
    public async Task UpdateAsync(Guid id, Order updatedOrder) =>
        await _ordersCollection.ReplaceOneAsync(x => x.Id == id, updatedOrder);

    2 references
    public async Task RemoveAsync(Guid id) =>
        await _ordersCollection.DeleteOneAsync(x => x.Id == id);
}
```

48 Datoteka OrderService.cs

EmailService

Za slanje mailova instaliran je paket MailKit. EmailService iz POST zahtjeva (slika 49) na API-ju dobije email adresu osobe koja prima mail, naslov i sadržaj maila.



49 POST zahtjev na API sučelju za Email

Unutar EmailService, kreira se novi email sa podacima u POST zahtjevu, te iz konfiguracije čita pošiljatelja maila i šifru potrebnu za slanje (slika 50).

```
2 references
public class EmailService : IEmailService
{
    private readonly IConfiguration _config;
    0 references
    public EmailService(IConfiguration config)
    {
        _config = config;
    }
    2 references
    public void SendEmail(EmailDto request)
    {
        try
        {
            var email = new MimeMessage();
            email.From.Add(MailboxAddress.Parse(_config.GetSection("EmailUserName").Value));
            email.To.Add(MailboxAddress.Parse(request.To));
            email.Subject = request.Subject;
            email.Body = new TextPart(TextFormat.Html) { Text = request.Body };

            using var smtp = new SmtpClient();
            smtp.Connect(_config.GetSection("EmailHost").Value, 465, MailKit.Security.SecureSocketOptions.SslOnConnect);
            smtp.Authenticate(_config.GetSection("EmailUserName").Value, _config.GetSection("EmailPassword").Value);
            smtp.Send(email);
            smtp.Disconnect(true);
        }
        catch(Exception ex)
        {

        }
    }
}
```

50 Datoteka EmailService.cs

Zaključak

U ovom završnom radu implementirana je izrada web shop aplikacije „Produjekt“ uz korištenje tehnologija MAUI Blazor Hybrid, ASP.NET Core, MongoDb te Azure Active Directory B2C. Glavne funkcionalnosti aplikacije uključuju dvojezičnost, kontrolu pristupa, slanje email potvrde i mogućnost uploada datoteka.

U izradi aplikacije MAUI Blazor Hybrid je korišten za izradu korisničkog sučelja na iOS, Android i Windows platformama. Pokazao se kao brz način za izradu višeplatformskih aplikacija, uz jednostavnu mogućnost proširenja koda za izradu web aplikacije u Blazoru.

Korištenje ASP.NET Core za izradu API-ja pokazao se kao jednostavan način za implementiranje kreiranja, ažuriranja, brisanja i čitanja podataka iz baze. Osim toga, za autentifikaciju i autorizaciju korisnika koristili smo Azure Active Directory B2C, što je osiguralo siguran pristup aplikaciji i kontrolu pristupa za različite vrste korisnika.

Popis slika

Literatura

Alexander S. Gillis, Bridget Botelho. 2023.. MongoDB. *Tech Target.* [Mrežno] 22. Kolovoz 2023. https://www.techtarget.com/searchdatamanagement/definition/MongoDB.

Charbeneau, Edward. 2023. Blazor Hybrid Web Apps with .NET MAUI. *Code Magazine.* [Mrežno] 22. Kolovoz 2023. https://www.codemag.com/Article/2111092/Blazor-Hybrid-Web-Apps-with-.NET-MAUI.

2023.. HTML: HyperText Markup Language. *mdn web docs.* [Mrežno] 26. Kolovoz 2023. https://developer.mozilla.org/en-US/docs/Web/HTML.

2023.. HTML5. *Wikipedia.* [Mrežno] 26.. Kolovoz 2023. https://en.wikipedia.org/wiki/HTML5.

2023.. Introduction to ASP.NET Core Minimal APIs. *JetBrains.* [Mrežno] 15. Kolovoz 2023. https://blog.jetbrains.com/dotnet/2023/04/25/introduction-to-asp-net-core-minimal-apis/.

2023. Minimal APIs quick reference. *Microsoft Learn.* [Mrežno] 10. Kolovoz 2023. https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-7.0.

2023.. MongoDB: An introduction. *GeeksForGeeks.* [Mrežno] 21. Kolovoz 2023. https://www.geeksforgeeks.org/mongodb-an-introduction/.

Montemagno, James. 2023. Leveraging Existing Web Content to Build Hybrid Apps with .NET MAUI. *DEV.* [Mrežno] 22. Kolovoz 2023. https://dev.to/dotnet/leveraging-existing-web-content-to-build-hybrid-apps-with-net-maui-48me.

Rivero, Diego. 2023. Blazor And .NET MAUI: A Powerful Combination For Hybrid Apps. *Grial.* [Mrežno] 22. Kolovoz 2023. https://grialkit.com/blog/blazor-and-net-maui-a-powerful-combination-for-hybrid-apps.

2023. *The Developer's Guide to Azure.* s.l. : Microsoft Press, A division of Microsoft Corporation, 2023.

2023.. What are NoSQL databases? *IBM.* [Mrežno] 22. Kolovoz 2023. https://www.ibm.com/topics/nosql-databases.

2023. What is .NET MAUI? *Microsoft Learn.* [Mrežno] 29. Srpanj 2023. https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui.

2023.. What is MongoDB? *IBM.* [Mrežno] 22. Kolovoz 2023. https://www.ibm.com/topics/mongodb.

Wildermuth, Shawn. 2023. Minimal APIs in .NET 6. *CodeMag.* [Mrežno] 22. Kolovoz 2023. https://www.codemag.com/Article/2201081/Minimal-APIs-in-.NET-6.

Zang, Assis. 2023.. What's New in .NET 7 for Minimal APIs? *Telerik.* [Mrežno] 17. Kolovoz 2023. https://www.telerik.com/blogs/whats-new-dotnet-7-minimal-apis.