Razvoj .NET MAUI aplikacije

Hrsto, Blagica

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET

ZAVRŠNI RAD

RAZVOJ .NET MAUI APLIKACIJE

Blagica Hrsto

Split, rujan 2024.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

RAZVOJ .NET MAUI APLIKACIJE

Blagica Hrsto

SAŽETAK

Rad se bavi razvojem cross-platformskih aplikacija u .NET MAUI-u (*Multi-platform App UI*), posebno aplikacija za Android koristeći MVVM ili čistu arhitekturu. .NET MAUI omogućava razvoj unificiranih aplikacija za različite platforme, uključujući Android, koristeći zajedničku bazu koda i XAML za definiranje korisničkog sučelja. MVVM (Model-Pogled-Pogled modela) arhitektura odvaja poslovnu logiku od prikaza, dok povezivanje podataka i promatrani podaci (npr. *ObservableCollection*) osiguravaju automatsko ažuriranje UI-a pri promjenama u podacima. Ubrizgavanje ovisnosti olakšava upravljanje ovisnostima i organizaciju koda, dok *SQLite* služi kao lagana baza podataka za pohranu podataka unutar aplikacije. Korištenje čiste arhitekture dodatno poboljšava održivost i modularnost aplikacije, omogućujući jasnu razdiobu između slojeva sustava.

Ključne riječi: .NET MAUI, XAML, povezivanje podataka, ubrizgavanje ovisnosti, MVVM arhitektura, Čista arhitektura, Android aplikacija, promatrani podaci, SQLite.

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 39 stranica, 18 grafičkih prikaza i 16 literaturnih navoda.

Izvornik je na hrvatskom jeziku.

Mentor: **Doc.dr.sc. Divna Krpan**, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Ocjenjivači: **Doc.dr.sc. Divna Krpan**, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Doc.dr.sc. Goran Zaharija, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Mag. educ. math. et inf., Dino Nejašmić, predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad prihvaćen: Rujan, 2024.

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Computer Science
Ruđera Boškovića 33, 21000 Split, Croatia

DEVELOPMENT OF .NET MAUI APPLICATION

Blagica Hrsto

ABSTRACT

This paper is concentrated on the development of cross-platform applications in .NET MAUI (Multi-platform App UI), especially Android applications using MVVM or Clean architecture. .NET MAUI enables the development of unified applications for different platforms, including Android, using a common code base and XAML to define the user interface. The MVVM (Model-View-ViewModel) architecture separates business logic from the view, while data binding and observable data (eg ObservableCollection) ensure that the UI automatically updates when the data changes. Dependency injection facilitates dependency management and code organization, while SQLite serves as a lightweight database to store data within the application. The use of a clean architecture further improves the maintainability and modularity of the application, allowing a clear separation between the layers of the system.

Key Words: .NET MAUI, XAML, data binding, dependency injection, MVVM architecture , Clean architecture , Android application, observable data, SQLite.

Thesis deposited in library of Faculty of science, University of Split.

Thesis consist of: 39 pages, 18 figures and 16 references

Original language: Croatian

Supervisor: **Divna Krpan**, **Ph.D.** Professor of Faculty of Science, University of Split

Reviewers: **Divna Krpan**, **Ph.D.** Professor of Faculty of Science, University of Split,

Goran Zaharija, Ph.D. Professor of Faculty of Science, University of Split,

Dino Nejašmić, mag. educ. math. et inf., Lecturer of Faculty of Science, University of Split

Thesis accepted: September, 2024.

Sadržaj

UVOD

Tema ovog završnog rada je izrada mobilnih aplikacija u .NET MAUI okruženju s ciljem proučavanja i razumijevanja novih tehnologija te različitih pristupa izradi i arhitekturi softvera. U današnje vrijeme, mobilne aplikacije postaju neizostavan dio svakodnevnog života, omogućavajući korisnicima da obavljaju razne zadatke, od praćenja vremenske prognoze do upravljanja osobnim financijama. Kako bi se ispunili zahtjevi modernog tržišta, od programera se očekuje da budu u toku s najnovijim tehnologijama i metodologijama razvoja softvera. U tom kontekstu, .NET MAUI predstavlja moderni okvir za razvoj unificiranih aplikacija koje mogu raditi na različitim platformama, uključujući Android, iOS, Windows i macOS.

U ovom završnom radu, kroz izradu tri različite mobilne aplikacije za Andorid, istražit će se mogućnosti .NET MAUI okruženja, kao i prednosti i izazovi korištenja različitih arhitektonskih pristupa u razvoju mobilnih aplikacija.

Aplikacije su:

- **Aplikacija za prikaz vremenske prognoze**, koja koristi *Geocoding* API za dohvaćanje i prikaz vremenskih podataka.
- **Aplikacija za popis gostiju**, izrađena prema čistoj arhitekturi za jednostavnije održavanje i proširivost.
- **Aplikacija za vođenje osobnih troškova**, bazirana na MVVM arhitekturi, koja omogućuje jasno razdvajanje logike od korisničkog sučelja.

1 .NET OKRUŽENJE

.NET okruženje je snažan i prilagodljiv okvir za razvoj softvera, koji programerima omogućava kreiranje raznih vrsta aplikacija. Microsoft je razvio .NET kako bi pružio podršku za više programskih jezika, alata i biblioteka, omogućavajući razvoj aplikacija za različite platforme kao što su web, mobilni uređaji, desktop računala, igre i IoT (engl. *Internet of Things*).

Ključne značajke .NET okruženja:

- **Podrška za više platformi**:
  - **.NET Core** omogućava razvoj aplikacija koje mogu raditi na različitim operativnim sustavima, poput Windowsa, macOS-a i Linuxa. Ovo čini .NET pogodnim za programere koji žele da njihove aplikacije budu dostupne na raznim platformama.
- **Programski jezici i sintaksa**:
  - **C#**: Glavni programski jezik za .NET, poznat po svojoj jednostavnosti i modernom objektno orijentiranom pristupu.
  - **F#**: Funkcionalni programski jezik koji također podržava objektno orijentirano i imperativno programiranje.
  - **Visual Basic**: Jezik s jednostavnom sintaksom, osmišljen za brzo i efikasno razvijanje aplikacija.
- **Ponovna upotreba koda**:
  - Jedna od ključnih prednosti .NET-a je mogućnost ponovne upotrebe koda na različitim uređajima i platformama, što znatno smanjuje vrijeme razvoja i povećava produktivnost.
- **Jedinstveno razvojno okruženje**:
  - **Visual Studio**: Primarni alat za razvoj u .NET-u, koji pruža sveobuhvatnu podršku za pisanje, testiranje i implementaciju aplikacija. **Visual Studio Code** je lakša alternativa koja podržava više jezika i može se koristiti na svim platformama.

- **Sigurnost i performanse**:
  - .NET okruženje nudi snažne sigurnosne mogućnosti i optimizirane performanse, što ga čini idealnim za razvoj složenih poslovnih aplikacija. (Microsoft, .NET Framework, 2024)

2 .NET MAUI

2.1 Općenito

.NET MAUI (engl. .NET Multi-platform App UI) je okvir za razvoj nativnih mobilnih i desktop aplikacija koji koristi C# i XAML. Omogućuje stvaranje aplikacija koje mogu raditi na Andorid, IOS, macOS i Windows sustavima sa jedinstvenom dijeljenom „bazom“ koda.

Glavne značajke .NET MAUI:

- **Cross-platformska podrška:** Mogućnost razvijanja aplikacije za sve četiri glavne platforme iz jednog zajedničkog koda, uz mogućnost dodavanja specifičnog koda i resursa za svaku platformu.
- **Evolucija Xamarin.Forms:** .NET MAUI je nadogradnja Xamarin.Forms, proširujući podršku s mobilnih na desktop aplikacije (naknadno dodana mogućnost) s poboljšanim kontrolama za bolju izvedbu.
- **Jedinstveni okvir:** Ujedinjuje API-je za Android, iOS, macOS i Windows u jedan API, omogućujući razvoj aplikacija koje se mogu pokrenuti bilo gdje.
- **Kod i Kontrole:** Pisanje koda u .NET MAUI uglavnom uključuje rad s .NET MAUI kontrolama i API slojem. Ove kontrole pružaju osnovne funkcionalnosti, dok API sloj omogućuje interakciju s nativnim API-jima svake platforme. Pruža bogat set kontrola i komponenti za izgradnju korisničkog sučelja, uključujući kontrole za prikaz podataka, navigaciju i unos korisničkih podataka. Također pruža podršku za prilagodbu kontrola i UI elemenata pomoću vlastitih događaja, omogućujući bolje usklađivanje s specifičnim zahtjevima.

- **Pristup uređajima:** Omogućuje pristup funkcijama uređaja kao što su GPS, akcelerometar, kamera i stanje baterije kroz zajedničke API-je. Također pruža funkcionalnosti za crtanje i prikazivanje grafike, uključujući operacije sastavljanja i transformaciju grafičkih objekata.

Slika 2.1: .NET MAUI (Mohanan, 2024)

2.2 Kompiliranje i izvršavanje

Android:

- Kompilacija: Aplikacije razvijene u .NET MAUI kompiliraju se iz C# koda u međukod (engl. *Intermediate Language*, skraćeno IL).
- Izvršenje: Kada se aplikacija pokrene, IL se dinamički JIT (engl. *Just-In-Time*) kompilira u nativne instrukcije specifične za Android uređaj. Ovo omogućava aplikaciji da koristi punu snagu i mogućnosti uređaja u trenutku izvršenja.

iOS:

- Kompilacija: Aplikacije za iOS se AOT (engl. Ahead-Of-Time) kompiliraju iz C# koda u nativni ARM kod prije nego što se aplikacija pokrene.

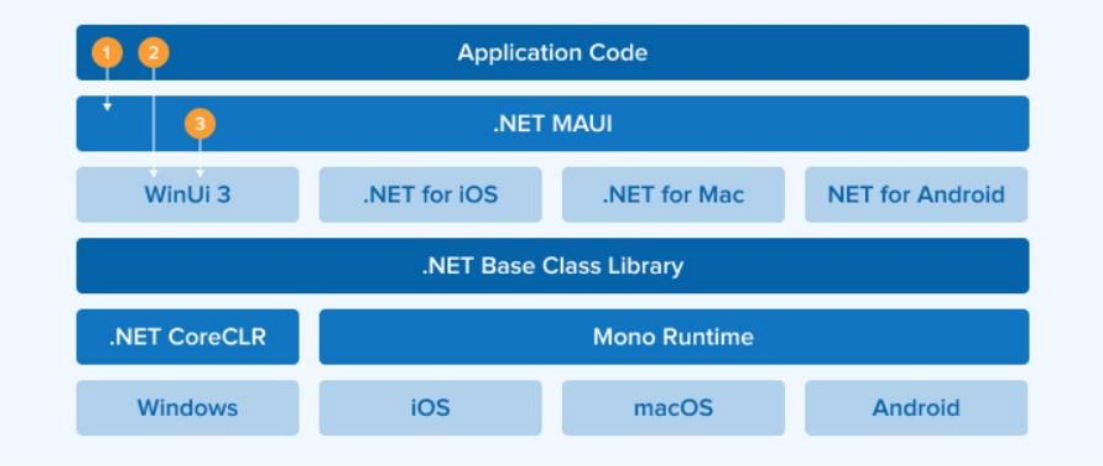
- Izvršenje: Ovaj pristup omogućava brže pokretanje aplikacije jer je već kompilirana u izvršni kod specifičan za iOS uređaje.

macOS:

- Kompilacija i Izvršenje: Na macOS-u, .NET MAUI koristi Mac Catalyst za prilagodbu iOS aplikacija za desktop okruženje. *Mac Catalyst* omogućava da se iOS aplikacije napravljene uz *UIKit* prilagode za macOS, dodajući dodatne funkcionalnosti i API-je specifične za desktop računala, poput *AppKit-a*.

Windows:

- Kompilacija i Izvršenje: Za Windows aplikacije, .NET MAUI koristi *WinUI 3* biblioteku. *WinUI 3* pruža moderne kontrole i API-je za stvaranje nativnih Windows aplikacija koje se mogu pokretati na različitim verzijama Windowsa, omogućujući bogato korisničko sučelje i visokokvalitetne performanse.

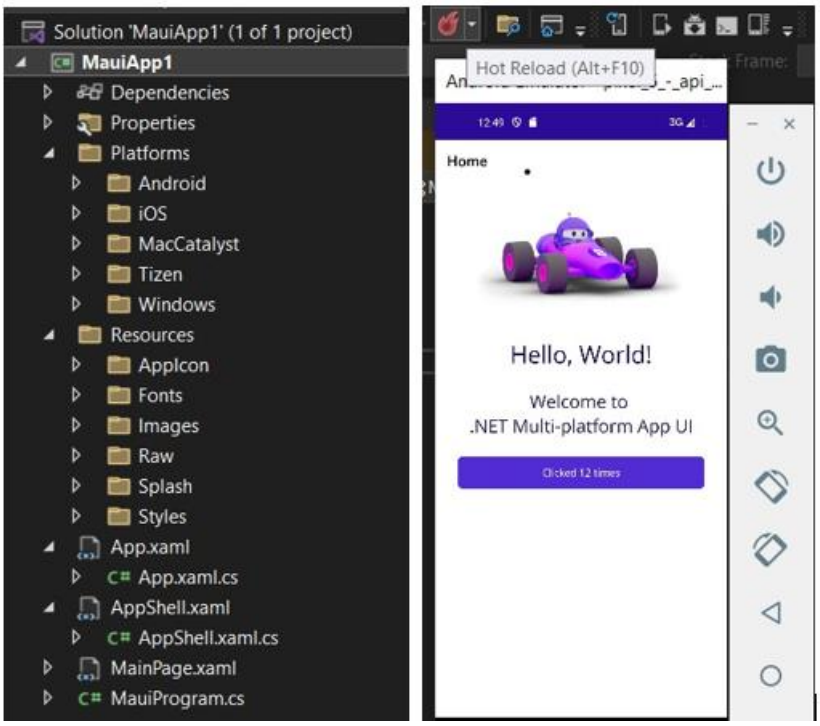


Slika 2.2: Izvršavanje (Microsoft, .NET 8 MAUI, 2024)

2.3 Razvoj i učitavanje

.NET MAUI omogućuje razvoj aplikacija na PC-ju ili Mac-u. Programeri mogu koristiti Visual Studio za kreiranje aplikacija i raditi na njima bez obzira na platformu na kojoj se nalaze. .NET MAUI pruža značajke kao što su **.NET Hot Reload** i **XAML Hot Reload**. Ove funkcionalnosti omogućuju programerima da:

- .NET Hot Reload: Izmjenjuju kod dok je aplikacija u funkciji, omogućujući brzu provjeru promjena bez potrebe za ponovnim pokretanjem aplikacije.
- XAML Hot Reload**:** Ugrađuju promjene u XAML datotekama i odmah možemo vidjeti rezultate u aplikaciji bez ponovnog kompiliranja, što olakšava izmjenu dizajna korisničkog sučelja i održavanje trenutnog stanja navigacije i podataka. (Microsoft, .NET 8 MAUI, 2024)



Slika 2.3: Prikaz platformi s lijeve strane i mogućnost Hot Reloda s desne

2.4 Evolucija i druge platforme

2.4.1 Xamarin

.NET MAUI možemo shvatiti kao nadogradnju Xamarin.Forms. Xamarin.Forms je bio okvir za razvoj aplikacija koji omogućuje izgradnju korisničkog sučelja za više platformi (Android, iOS) koristeći .NET i C#. Omogućio je programerima da stvaraju mobilne aplikacije s jednim dijelom koda, uz osnovne kontrole i API-je specifične za svaku platformu. Microsoftova podrška za Xamarin je prestala vrijediti 1. svibnja 2024. Nakon tog datuma, Microsoft više ne nudi popravke grešaka, sigurnosna ažuriranja niti nove značajke za Xamarin. Ako bi pokušali

koristiti Xamarin nakon tog datuma, aplikacije vjerojatno neće raditi na najnovijim verzijama iOS-a i Androida.

Neke od razlika koje su dodane u .NET MAUI su: .NET MAUI uključuje jedinstvenu strukturu projekta koja objedinjuje više platformskih projekata (iOS, Android, macOS, Windows) u jedan projekt, što pojednostavljuje organizaciju projekata i upravljanje zajedničkim resursima. *Hot Reload* omogućuje trenutne povratne informacije programerima, što ubrzava razvojni proces. MAUI također podržava arhitekturu Model-Pogled-Ažuriraj (engl. *Model-View-Update*, skraćeno MVU), olakšavajući izradu korisničkih sučelja i potičući jasnu podjelu odgovornosti.

Ukratko .NET MAUI predstavlja kasniju inačicu Microsoftovog cross-platformskog okvira za razvoj aplikacija koji uključuje prednosti Xamarina uz pružanje poboljšanih funkcionalnosti, pojednostavljenu organizaciju projekta i jednostavniju interakciju sa širom .NET mrežom.

2.4.2 Blazor

Blazor je Microsoftov proizvod s kojim možemo razvijati aplikacije sa SPA (engl. *Single Page Application*) pristupom s *WebAssembly* ili Server verzijama.

MAUI Blazor Hybrid također predstavlja evoluciju Xamarin.Forms (okvira za izradu nativnih aplikacija pomoću C#). S MAUI-em, programeri mogu kreirati nativna korisnička sučelja koristeći jedinstveni dijeljeni kod koji radi na više platformi, poput iOS-a, Androida, macOS-a i Windowsa, a dodatno uz Blazor Hybrid, mogu se koristiti poznati web razvojni okviri, čime se razvoj dodatno pojednostavljuje i ubrzava. Dakle, Blazor MAUI Hybrid kombinira Balzor i .NET MAUI omogućujući razvoj web aplikacija i koristeći web tehnologije (HTML i CSS) za razvoj korisničkog sučelja unutar MAUI okvira. (Kathiresan, Xamarin Versus .NET MAUI, 2024) (Nazarevich, 2023)

Usporedba

.NET MAUI koristi nativne komponente za cijeli UI, dok .NET MAUI Blazor Hybrid omogućuje integraciju Blazor komponenata unutar nativnih MAUI aplikacija. .NET MAUI može pružiti bolje performanse za aplikacije koje se u potpunosti oslanjaju na nativne komponente, dok Blazor Hybrid može biti prikladan za scenarije gdje je potreban spoj web i nativnih elemenata.

3 XAML

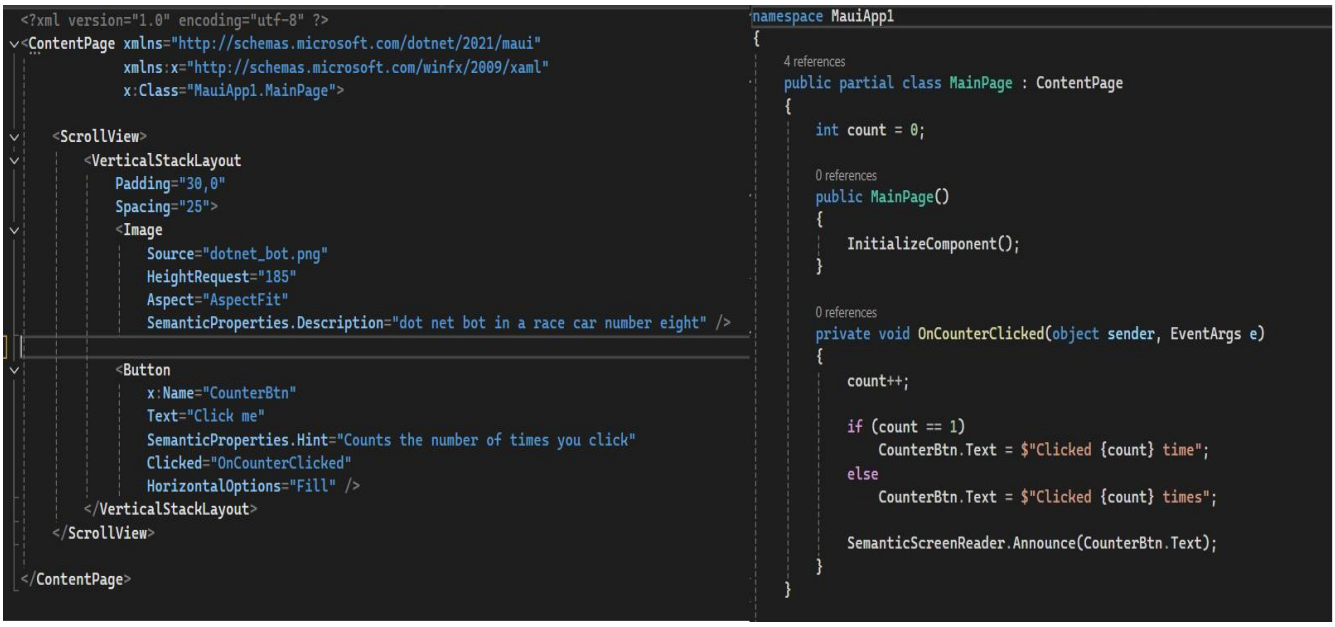
XAML (engl. *Extensible Application Markup Language*) je deklarativni jezik koji se koristi za definiranje korisničkih sučelja (UI) u aplikacijama, posebno unutar Microsoftovih tehnologija za razvoj desktop i mobilnih aplikacija. Omogućuje razdvajanje prezentacijskog sloja od logike aplikacije, što pojednostavljuje proces razvoja i olakšava održavanje koda.

XAML je razvio Microsoft kao dio svoje šire strategije za razvoj bogatih korisničkih sučelja u svojim aplikacijama. Njegov razvoj započeo je s uvođenjem WPF (engl. *Windows Presentation Foundation*) u .NET Framework 3.0, koji je izdan 2006. godine. (Britch, 2023)

Razvoj XAML-a od strane Microsofta potaknut je željom za omogućavanjem stvaranja složenih, grafički bogatih korisničkih sučelja koja su prilagodljiva. Korištenjem deklarativnog jezika poput XAML-a, razvoj korisničkih sučelja postao je jednostavniji i intuitivniji, posebno u kombinaciji s alatima kao što je Visual Studio. Ovaj pristup također je omogućio bolju podjelu rada između dizajnera i programera, pri čemu se dizajneri mogu fokusirati na dizajn korisničkog sučelja bez potrebe za dubokim poznavanjem programskih jezika, dok su se programeri mogli usredotočiti na implementaciju poslovne logike.

3.1 Izgled stranice

Kada dodajemo XAML stranicu u projektu (*Content Page*), primjerice *MauiApp.xaml*, automatski će nam se stvoriti i stranica sa kodom iza naziva *MauiApp.xaml.cs* gdje pišemo kod u C# koji služi za upravljanje elementima korisničkog sučelja koje stvorimo na XAML stranici. Razlog razdvajanja u ove dvije datoteke je jasna podjela između UI-a i logike aplikacije čiji je cilj bio olakšati razvoj, održavanje i testiranje same aplikacije. (Britch, 2023)



```xaml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.MainPage">

    <ScrollView>
        <VerticalStackLayout
            Padding="30,0"
            Spacing="25">
            <Image
                Source="dotnet_bot.png"
                HeightRequest="185"
                Aspect="AspectFit"
                SemanticProperties.Description="dot net bot in a race car number eight" />

            <Button
                x:Name="CounterBtn"
                Text="Click me"
                SemanticProperties.Hint="Counts the number of times you click"
                Clicked="OnCounterClicked"
                HorizontalOptions="Fill" />
        </VerticalStackLayout>
    </ScrollView>

</ContentPage>
```

```xaml
namespace MauiApp1
{
    4 references
    public partial class MainPage : ContentPage
    {
        int count = 0;

        0 references
        public MainPage()
        {
            InitializeComponent();
        }

        0 references
        private void OnCounterClicked(object sender, EventArgs e)
        {
            count++;

            if (count == 1)
                CounterBtn.Text = $"Clicked {count} time";
            else
                CounterBtn.Text = $"Clicked {count} times";

            SemanticScreenReader.Announce(CounterBtn.Text);
        }
    }
```

Slika 3.1: Lijevo predložak xaml stranice i desno kod iza u xaml.cs-u

Objašnjenje XAML stranice:

XAML stranica obično se sastoji od elemenata koji predstavljaju vizualne komponente, poput gumbova, oznaka, slika i rasporeda, kao i njihovih atributa, stilova i događaja. Gornji primjer pokazuje kako može izgledati osnovna XAML stranica u .NET MAUI.

ContentPage predstavlja glavnu stranicu koja sadrži sve UI elemente. U samom vrhu stranice nalaze se zadani imenski prostori. U XAML-u, imenski prostori se koriste za organizaciju i identifikaciju različitih elemenata i tipova koji se koriste unutar XAML datoteke. Oni omogućuju povezivanje XAML elemenata s odgovarajućim klasama i objektima definiranima u .NET-u. U nastavku je objašnjenje važnosti imenskih prostora koji se često koriste u .NET MAUI aplikacijama.

- `xmlns="http://schemas.microsoft.com/dotnet/2021/maui"`

Ovaj imenski prostor definira osnovni imenski prostor za .NET MAUI. To znači da su svi elementi koji su deklarirani unutar ovog imenskog prostora standardni MAUI elementi kao što su *Button, Label, Grid, StackLayout* itd. Ovaj imenski prostor omogućava korištenje svih MAUI kontrola i funkcionalnosti bez potrebe za dodatnim prefiksima.

- `xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"`

Ovo je standardni XAML imenski prostor koji pruža podršku za nekoliko ključnih XAML funkcionalnosti, kao što su:

- Koristi se za definiranje povezivanja XAML datoteke s odgovarajućom klasom u C#. Na primjer, `x:Class="MauiApp1.MainPage"` povezuje ovu XAML datoteku s *MainPage* klasom u MauiApp1 imenskom prostoru.
- Omogućava imenovanje XAML elemenata tako da im se može pristupiti iz C# koda.
- Koristi se unutar resursa za definiranje jedinstvenog ključa kojim se može pristupiti resursu.
- Omogućava pristup statičnim članovima iz XAML-a.

Također kada instalirate neke biblioteke koje želite koristiti u projektu potrebno ih je navesti unutar novog imenskog prostora na vrhu stranice koju koristimo.

ContentPage

Nastavimo dalje sa *ContentPageom*. Na stranici se može nalaziti samo jedan element ContentPage, a on može sadržavati samo jedno dijete, odnosno može imati samo jednog direktnog potomka. Ovaj element može biti bilo koja vizualna kontrola, poput *Label*, *Button, Image*, ili kompleksnija struktura poput *StackLayout, Grid,* ili *ScrollView*. Dakle, iako ContentPage direktno podržava samo jedno dijete, to dijete može biti složeni kontejner koji sadrži mnogo drugih elemenata.

ContentPage ima svojstvo *BindingContext*, koje se koristi za vezivanje podataka (više o tome u odlomku 3.3). Ovo svojstvo omogućava vezivanje korisničkog sučelja sa podacima, omogućujući dinamičko ažuriranje prikaza kad se podaci promijene.

ContentPage dolazi s ugrađenim metodama koje odgovaraju različitim fazama životnog ciklusa stranice, kao što su *OnAppearing* (koja se poziva kada stranica postane vidljiva) i *OnDisappearing* (koja se poziva kada stranica više nije vidljiva). Ove metode se pišu u kodu iza stranice, odnosno u *xaml.cs* datoteci.

ContentPage se često koristi u kombinaciji s *NavigationPage*(*ContentPage* također ima svojstvo Title, koje se koristi za postavljanje naslova stranice, često prikazanog u navigacijskoj traci kada se koristi unutar *NavigationPage*.) ili *TabbedPage* za implementaciju navigacije unutar aplikacije. Ovo omogućava korisnicima da prelaze između različitih stranica u aplikaciji.

Objašnjenje koda u XAML.cs:

- `public partial class MainPage : ContentPage`
  Ovdje se definira klasa *MainPage*, koja nasljeđuje *ContentPage*. Ova klasa je djelomična (engl. *partial*), što znači da je njen drugi dio definiran u XAML-u.
- `InitializeComponent()`:
  Ova metoda poziva se u konstruktoru i generirana je na osnovu XAML datoteke. Njena uloga je inicijalizirati elemente sučelja definirane u XAML-u.
- `OnButtonClick(object sender, EventArgs e)`
  Ovo je metoda koja upravlja događajem klika gumba. Kada korisnik klikne na gumb, tekst u labeli se mijenja u *„Button Clicked“.*

3.2 Kontrole

U .NET MAUI postoje razne kontrole ili elementi, slično kao i u HTML-u. U ovom tekstu su nabrojane neke od ključnih kontrola koje se koriste. Kontrole koje spadaju u kategorije pogleda i prikaza (engl. *View, Layout*) su posebno bitne jer obično sadrže sve ostale elemente stranice i definiraju daljnji tok dizajna. Da ponovimo *ContentPage* je osnovna stranica koja služi kao glavni kontejner za sve UI elemente. Kontrole prikaza kao što su *StackLayout, Grid, AbsoluteLayout i FlexLayout* pomažu u organiziranju drugih kontrola na stranici. Tekstualne kontrole uključuju *Label* za prikaz zadanog teksta, *Entry* za unos jednostrukog reda teksta, i Editor za višeredni tekst. Botun omogućava korisničku interakciju, dok *Image* prikazuje slike. *ListView, CollectionView, ScrollView* prikazuju popis stavki s različitim mogućnostima prikaza. *Picker* i *DatePicker* koriste se za odabir vrijednosti i datuma. Switch omogućava uključivanje ili isključivanje opcija, dok *Slider* i *Stepper* služe za odabir i prilagodbu vrijednosti. *ProgressBar* pokazuje napredak procesa, *SearchBar* omogućava pretraživanje, a *WebView* omogućava prikaz web stranica unutar aplikacije. *TabBar* i *FlyoutPage* pomažu u navigaciji između različitih sekcija aplikacije.

3.2.1 Prikazi

Najkorišteniji prikazi (engl. *Layouts*) su:

StackLayout, HorizontalStackLayout, VerticalStackLayout, Grid i *FlexLayout*.

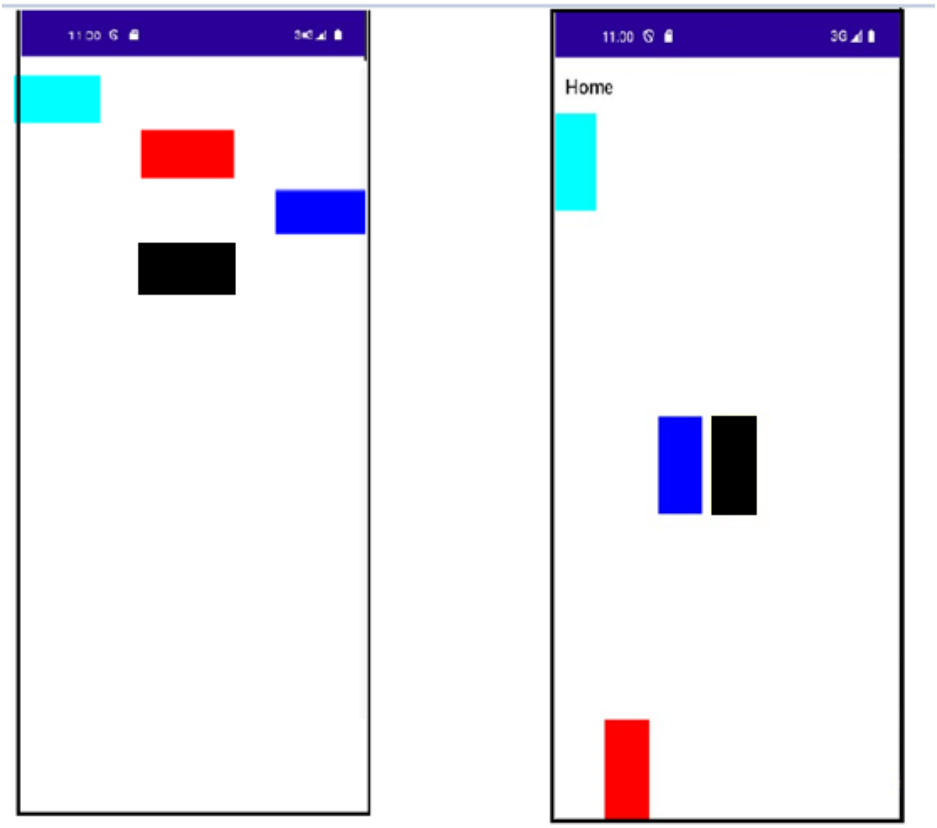
StackLayout je kontrola u .NET MAUI koja raspoređuje svoju djecu elemente u nizu, bilo horizontalno ili vertikalno. Evo ključnih razlika i svojstava između navedena tri načina rasporeda.

StackLayout se može koristiti za horizontalno ili vertikalno postavljanje elemenata, ovisno o vrijednosti atributa *Orientation*. Ključni atributi uključuju *Orientation* (koji određuje usmjerenje elemenata), *Spacing* (razmak između elemenata) i *Padding* (razmak između ruba *layouta* i elemenata). Fleksibilnost u rasporedu omogućava različite načine dizajniranja.

HorizontalStackLayout je specijalni oblik *StackLayout-a* unaprijed postavljen za horizontalno raspoređivanje elemenata. Ima iste ključne atribute kao *StackLayout, Spacing i Padding*, ali ne zahtijeva dodatnu konfiguraciju orijentacije.

VerticalStackLayout je specijalizirani oblik *StackLayout*-a unaprijed postavljen za vertikalno raspoređivanje elemenata. Kao i kod *HorizontalStackLayout*-a, koristi *Spacing* i *Padding*, ali se automatski postavlja za vertikalni raspored bez potrebe za dodatnim postavkama.

Sažeto, *StackLayout* je univerzalni prikaz za oba smjera, dok su *HorizontalStackLayout i VerticalStackLayout* specijalizirani za horizontalni i vertikalni raspored elemenata, što omogućuje jednostavnije i čitljivije postavljanje u kodu.



Slika 3.2: VerticalStackLayout i HorizontalStackLayout

Grid i ***FlexLayout*** su također dva različita prikaza (engl. *Layout*) u .NET MAUI koja se koriste za raspoređivanje UI elemenata, svaki s jedinstvenim karakteristikama i primjenama:

- *Grid* omogućuje postavljanje elemenata u mrežu sa definiranim redovima i stupcima. Pruža preciznu kontrolu nad rasporedom elemenata kroz definirane ćelije. Idealno je za rasporede gdje je potrebno postaviti elemente u točno određene pozicije ili za kreiranje složenih struktura. Ključni atributi uključuju *RowDefinitions* i *ColumnDefinitions* za definiranje broja redova i stupaca.
- *FlexLayout* omogućuje fleksibilno i dinamično raspoređivanje elemenata u horizontalnom ili vertikalnom smjeru, slično CSS *Flexbox-u*. Koristi se za prilagodljive i responzivne dizajne gdje elementi mogu automatski promijeniti veličinu i poziciju prema raspoloživom prostoru. Ključni atributi uključuju *Direction* (usmjeravanje elemenata) i *JustifyContent* (poravnanje elemenata unutar prostora).

Ukratko, *Grid* je prikladan za statičke i precizne rasporede, dok *FlexLayout* nudi veću fleksibilnost i responzivnost za dinamične i adaptivne dizajne.

3.2.2 Pogledi

U .NET MAUI aplikacijama, postoje različiti elementi koji se često koriste za prikazivanje i manipulaciju podacima te organizaciju korisničkog sučelja. Tri najčešće korištena pogleda (engl. *View*) su:

- ***CollectionView*** je „kolekcijski" pogled koji se koristi za prikazivanje popisa stavki s visokom prilagodljivošću i dobrom izvedbom. Idealan je za prikazivanje velikih skupova podataka u obliku popisa ili mreže. **Atribut *ItemSource*** postavlja izvor podataka za *CollectionView*, obično vezan na kolekciju objekata (npr. *ObservableCollection*). Svaka stavka u kolekciji se prikazuje kao element u popisu ili mreži. Uz ovaj element se jako često koristi i povezivanje podatka za dodavanje novih elemenata na neki popis.

- **ScrollView** je pogled koji omogućuje pomicanje sadržaja unutar okvira, što je korisno kada sadržaj premašuje raspoloživi prostor na ekranu. Koristi se za prikazivanje sadržaja koji je veći od prikazanog područja, poput velikih slika, dugih tekstova ili kompleksnih formi.

- **ListView** je tradicionalni pogled za prikazivanje popisa stavki. Iako je sličan *CollectionView-u, ListView* je stariji i manje fleksibilan, ali se još uvijek koristi u mnogim aplikacijama. Služi za prikazivanje jednostavnih popisa podataka, kao što su popisi kontakata ili zadataka.

3.3 Povezivanje podataka

Povezivanje podataka (engl. *Data Binding*) u .NET MAUI predstavlja osnovni koncept koji služi za povezivanje podataka s korisničkim sučeljem, čime se pojednostavljuje interakcija između poslovne logike aplikacije i njenog vizualnog prikaza. Korištenjem povezivanja podatka, može se osigurati da korisničko sučelje automatski reflektira promjene u podacima,

dok istovremeno svaka promjena koju korisnik napravi u sučelju može automatski ažurirati podatke u pozadini. Ova dvosmjerna komunikacija ne samo da pojednostavljuje razvoj aplikacija već i omogućava bolje održavanje i proširivost, jer odvaja logiku aplikacije od njenog prikaza osiguravajući čitljiviji kod. Povezivanje podataka je stoga ključan alat u modernom razvoju aplikacija, posebno u kontekstu složenih aplikacija koje zahtijevaju dinamičko korisničko sučelje.

Ključni koncepti povezivanja podataka u .NET MAUI uključuju:

- **Kontekst povezivanja** je svojstvo koje određuje izvor podataka za element korisničkog sučelja. Svaki element može imati svoj vlastiti *BindingContext*, ili može naslijediti *BindingContext* od roditeljskog elementa. To omogućava fleksibilno povezivanje podataka s različitim dijelovima sučelja.

- **Način povezivanja** (engl. *Binding Mode*): Postoje različiti načini povezivanja podataka koji određuju smjer ažuriranja podataka:

  Jednosmjerno: Podaci se ažuriraju samo iz izvora podataka prema korisničkom sučelju.

  Dvosmjerno: Promjene u korisničkom sučelju ažuriraju izvor podataka i obrnuto.

  Jednosmjerno prema izvoru (engl. *OneWayToSource*): Podaci se ažuriraju samo iz korisničkog sučelja prema izvoru podataka.

  Jednokratno (engl. *OneTime*): Podaci se postavljaju prilikom inicijalizacije i ne mijenjaju se kasnije.

- **Predlošci podataka** (engl. *Data Templates*): omogućava prilagođavanje prikaza elemenata kada se koriste kontrole poput *ListView, CollectionView, ili Picker*. Ovaj koncept omogućava definiranje kako će podaci biti vizualno predstavljeni u korisničkom sučelju.

- **Konverteri vrijednosti** (engl. *Value Converters*): *IValueConverter* se koristi za pretvaranje podataka između izvora podataka i korisničkog sučelja. Na primjer, ako izvor podataka sadrži vrijednost koja treba biti prikazana u različitom formatu ili vrsti podataka u UI-u, konverteri mogu obaviti tu transformaciju.

- **Komande** (engl. *Commands*): omogućuju povezivanje radnji s korisničkim sučeljem putem komandi, umjesto direktnog rada sa događajima. To je često korišteno u MVVM obrascu, gdje komande u modelu pogleda mogu biti povezane s botunima ili drugim elementima korisničkog sučelja.

- ***INotifyPropertyChanged***: Ovaj interfejs omogućava objektima da obavijeste korisničko sučelje o promjenama u njihovim svojstvima. Implementacija ovog interfejsa je ključna za dvosmjerni *data binding*, jer osigurava da se sučelje ažurira kad god dođe do promjene podataka.

- **Povezivanje u XAML-u**: U .NET MAUI povezivanje podataka (engl. *data binding*) se često definira u XAML-u, što omogućava deklarativno povezivanje podataka sa sučeljem. To pomaže u čuvanju čistoće koda, jer su vizualni elementi i njihovi podaci povezani direktno unutar XAML datoteka.

U napravljenim aplikacijama najčešće korišteni koncepti su *INotfiyPropertyChanged, Binding Context,* posebno *IValueConverter* za konverzije oznaka i animacija za prikazivanje trenutnog vremena.

3.4 Promatrani podaci

U .NET MAUI, koncept promatranih podataka (engl. *Observable Data*) se odnosi na podatke koji mogu obavijestiti korisničko sučelje (UI) kada dođe do promjena. Ova funkcionalnost je ključna za omogućavanje dinamičkog i reaktivnog korisničkog iskustva, gdje se korisničko sučelje automatski ažurira u skladu s promjenama podataka. Na primjer, kada se koristi `ObservableCollection<Type>` (generička kolekcija u .NET MAUI koja implementira sučelje `INotifyCollectionChanged`, što omogućuje obavještavanje sučelja kada se promijeni kolekcija) za prikazivanje popisa stavki u *CollectionView* ili *ListView,* dodavanje, uklanjanje ili zamjenu stavki, u popisu se automatski ažurira prikaz bez potrebe za dodatnim kodiranjem. Ovo značajno pojednostavljuje razvoj i omogućava nam da se fokusiramo na poslovnu logiku, dok se .NET MAUI okruženje brine o ažuriranju sučelja. (Britch, 2023)

Posebno u kombinaciji s MVVM arhitekturom, promatrani podaci omogućavaju čistu razdiobu između prezentacijskog sloja i logike aplikacije, što dovodi do lakšeg održavanja i testiranja koda.

Kada bi u nekim slučajevima za vrijeme kodiranja koristili običnu listu umjesto *ObservableCollection* ne bi došlo do ažuriranja podataka u .NET MAUI aplikaciji te je to također jedna od specifičnosti za ovo okruženje.

Primjer:

```
public ObservableCollcetion<Guest> popis=new ObservableCollection<Guest>();
```

3.5 Navigacija

Navigacija u .NET MAUI okruženju omogućuje korisnicima da se kreću između različitih stranica unutar aplikacije. Ona je ključni aspekt dizajna korisničkog iskustva, osiguravajući jednostavan i intuitivan način prelaska između različitih dijelova aplikacije.

Osnovni Koncepti Navigacije u .NET MAUI

1. ***NavigationPage*** pruža osnovni model za navigaciju između stranica u obliku stoga. Stranice se dodaju na stog kada se korisnik kreće naprijed, a uklanjaju se sa stoga kada se korisnik vraća na prethodne stranice.
   `PushAsync(Page Naziv)` metoda dodaje stranicu na vrh stoga (odnosno, korisnik se „pomiče“ naprijed), dok `PopAsync()` metoda uklanja trenutnu stranicu sa stoga (odnosno, korisnik se „vraća“ unatrag).

   C#

   ```
   await Navigation.PushAsync(new GuestsPage());

   await Navigation.PopAsync();
   ```

2. ***Shell*** je moćnija i fleksibilnija opcija za organiziranje i upravljanje navigacijom u složenijim aplikacijama. Omogućava hijerarhiju i navigaciju karticama unutar jedne stranice.
   Shell koristi URI navigaciju, što omogućava lako definiranje i rukovanje rute unutar aplikacije.

   C#:

   ```
   await Shell.Current.GoToAsync("Transakcije");

   await  Shell.Current.GoToAsync("..");  // ako se želite vratiti na roditeljsku stranicu
   ```

3. ***TabbedPage*** omogućava navigaciju između različitih sadržaja kroz kartice. Svaka kartica može predstavljati različitu stranicu u aplikaciji.
   Ova vrsta navigacije je korisna kada aplikacija ima nekoliko različitih kategorija sadržaja ili funkcionalnosti koje korisnik treba često mijenjati.

4. ***FlyoutPage*** omogućava navigaciju pomoću bočnog izbornika koji se može proširiti i sakriti. Ovaj tip navigacije je popularan u aplikacijama gdje se izbornik često koristi za pristup različitim dijelovima aplikacije.

Navigacija u .NET MAUI može biti jednostavna ili složena, ovisno o potrebama aplikacije. Od osnovne navigacije putem *NavigationPage-a*, preko moćne i fleksibilne Shell strukture, do posebnih stranica kao što su *TabbedPage i FlyoutPage*. .NET MAUI pruža sveobuhvatan skup alata za organiziranje i upravljanje korisničkim iskustvom unutar aplikacije. Korištenjem MVVM arhitekture, navigacija postaje još čišća i održivija, što je ključ za razvoj profesionalnih aplikacija. (Microsoft, .NET 8 MAUI, 2024)

3.6 Dijeljene komponente

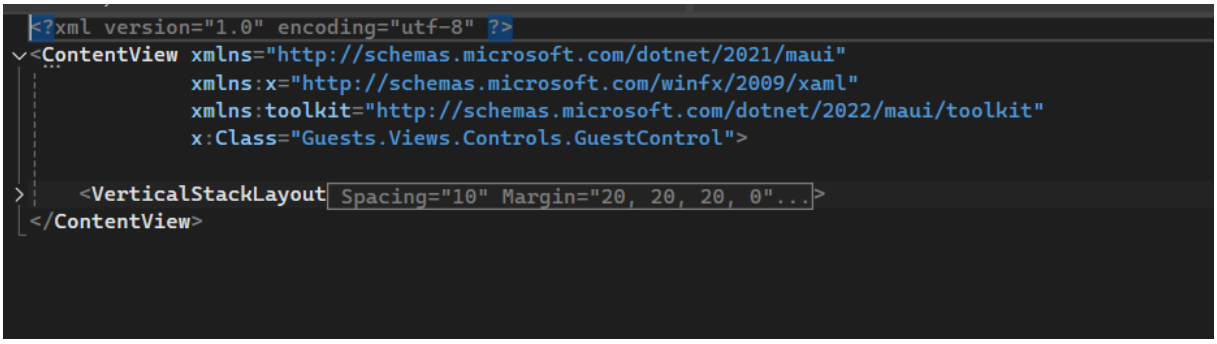
U .NET MAUI, dijeljenje komponenata ili stvaranje višestruko upotrebljivih kontrola omogućuje nam da izradimo upotrebljive dijelove korisničkog sučelja koji se mogu koristiti na različitim mjestima u aplikaciji. Ove komponente pomažu u održavanju koda, poboljšavaju konzistentnost UI-a i ubrzavaju razvoj.

ContentView

ContentView je osnovna klasa za stvaranje višestruko upotrebljivih komponenti u .NET MAUI-u. Pomoću *ContentView-a* možemo definirati korisničko sučelje u XAML-u ili C#-u, a zatim ga koristiti na različitim mjestima u aplikaciji.

Prilagođene kontrole (engl. *Custom Controls*)

Možemo izraditi i potpuno prilagođene kontrole koje nasljeđuju od osnovnih kontrola kao što su *Button, Entry, Label* itd., te dodati dodatne funkcionalnosti ili prilagoditi postojeće ponašanje.



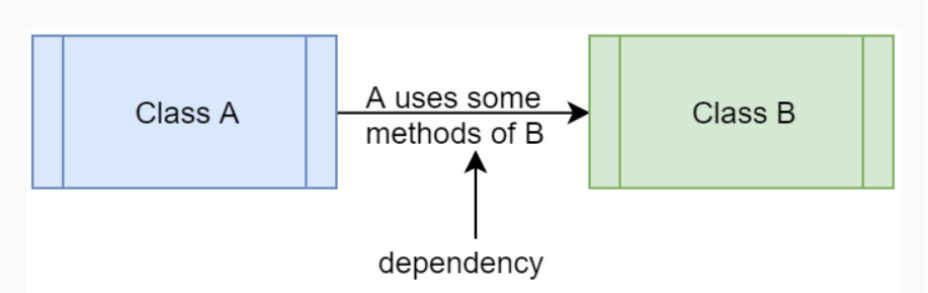
Slika 3.3: ContentView

4 UBRIZGAVANJE OVISNOSTI

Ubrizgavanje ovisnosti (engl. *Dependency Injection*, skraćeno DI) je tehnika u softverskom inženjerstvu koja omogućuje minimalne ovisnosti između objekata kroz jasno definirane odnose. Ova metoda doprinosi pisanju koda s niskim stupnjem povezivanja što znači da kod ovisi samo o neophodnim dijelovima drugih klasa. Time se smanjuje količina tvrdo kodiranih ovisnosti i omogućuje čišći i fleksibilniji kod.

Da bi ovo objasnili, zamislimo da ste na redu za naplatu u supermarketu. Hoćete li predati blagajniku svoj novčanik da bi uzeo traženi novac? Naravno da nećete. Umjesto toga, ili ćete mu dati gotovinu ili proći karticom. Održavate odnos tj. „sučelje“ između vas i blagajnika što jednostavnijim. To je srž ubrizgavanja ovisnosti. Ako vam treba nešto u kodu, tražite samo to što vam je potrebno, i ništa više.

Ubrizgavanje ovisnosti se fokusira na injektiranje ovisnosti umjesto njenoga stvaranja. To je jednostavan, ali učinkovit koncept koji čini kod bolje organiziranim. Radi se o traženju samo onoga što vam je potrebno i implementiranju slobodnije povezanog koda. Zahtjeva pažljivo planiranje i dizajn, ali čini održavanje koda znatno lakšim i učinkovitijim u jezicima poput Jave, C#, C++ i PHP-a. (Mohanan, 2024)



Slika 4.1:Koncept ubrizgavanja ovisnosti (BrightDevelopers, 2023)

4.1 Ubrizgavanje ovisnosti u .NET MAUI-u

Ubrizgavanje ovisnosti u .NET MAUI tehnika je koja omogućava da objekti u aplikaciji dobivaju svoje ovisnosti izvana, umjesto da ih sami stvaraju. Time se smanjuje čvrsta povezanost između objekata i olakšava testiranje i održavanje koda. U .NET MAUI, DI je integriran kroz paket *Microsoft.Extensions. DependencyInjection*, koji omogućuje registraciju i upravljanje ovisnostima u kontejneru usluga.

Servisi se registriraju prilikom pokretanja aplikacije, obično u datoteci *MauiProgram.cs*, a zatim se mogu injektirati u komponente kao što su stranice ili modeli pogleda (engl. *ViewModels*). Injektiranje ovisnosti može se obaviti putem konstruktora, metoda ili svojstava. DI također omogućava definiranje životnog ciklusa servisa (*Singleton, Transient, Scoped*), što određuje koliko dugo servis treba biti aktivan unutar aplikacije. (Microsoft, .NET 8 MAUI, 2024)

Singleton je servis koji se stvara samo jednom tijekom cijelog vijeka trajanja aplikacije. Nakon inicijalnog stvaranja, ista instanca servisa koristi se gdje god je potrebno. Ovaj životni ciklus je idealan za servise koji trebaju zadržati stanje ili koristiti resurse koje je efikasnije dijeliti, poput konfiguracijskih postavki, pristupa bazi podataka ili prijave.

Transient servisi stvaraju se svaki put kad se zatraže, što znači da će svaka komponenta koja zatraži ovaj servis dobiti novu instancu. Ovaj model je prikladan za servise koji ne čuvaju stanje ili kada je potrebno da svaka upotreba ima vlastitu instancu, kao što su usluge za obradu podataka ili kratkotrajne operacije. Prednost ovog pristupa je izbjegavanje dijeljenja stanja između različitih dijelova aplikacije, čime se smanjuje rizik od neočekivanih nuspojava (Slika *4.2*).

Scoped servisi stvaraju se jednom po svakoj specifičnoj domeni zahtjeva, primjerice, za svaki HTTP zahtjev u web aplikacijama. Unutar istog zahtjeva, svi dijelovi koda koji zatraže ovaj servis dobit će istu instancu, što omogućuje dijeljenje servisa unutar tog konteksta bez dijeljenja između različitih zahtjeva. Ovo je korisno kada radimo s podacima specifičnim za jedan korisnički zahtjev, čime se osigurava konzistentnost i izolacija podataka.

```
// DEPENDENCY INJECTION
//builder.Services.AddSingleton<IGuestRepository,GuestInMemoryRepository>();
builder.Services.AddSingleton<IGuestRepository, GuestSQLiteRepository>();
builder.Services.AddSingleton<IViewGuestsUseCase, ViewGuestsUseCase>();

builder.Services.AddSingleton<IViewGuestUseCase, ViewGuestUseCase>();

builder.Services.AddTransient<IEditGuestUseCase, EditGuestUseCase>();

builder.Services.AddTransient<IAddGuestUseCase, AddGuestUseCase>();

builder.Services.AddTransient<IDeleteGuestUseCase, DeleteGuestUseCase>();

// inejct pages
builder.Services.AddSingleton<GuestsPage>();
builder.Services.AddSingleton<EditGuestPage>();
builder.Services.AddSingleton<AddGuestPage>();

return builder.Build();
```

Slika 4.2: Implementacija DI-a

Korištenje DI-a u .NET MAUI povećava mogućnost testiranja, poboljšava organizaciju koda i smanjuje dupliciranje koda, čime se postiže fleksibilnija aplikacija. (Kathiresan, Learn How to Use Dependency Injection in .NET MAUI, 2024)

5 MVVM ARHITEKTURA

Model-Pogled-Model Pogleda (engl. *Model-View-ViewModel*, skraćeno MVVM) je softverski arhitektonski obrazac koji se najčešće koristi za razvoj Andorid aplikacija s korisničkim sučeljem koje se temelji na podacima. MVVM se koristi kako bi se postiglo jasno razdvajanje između poslovne logike, prikaza korisničkog sučelja i međusobne interakcije tih komponenti.

5.1 MVVM prikaz

- **Model**:

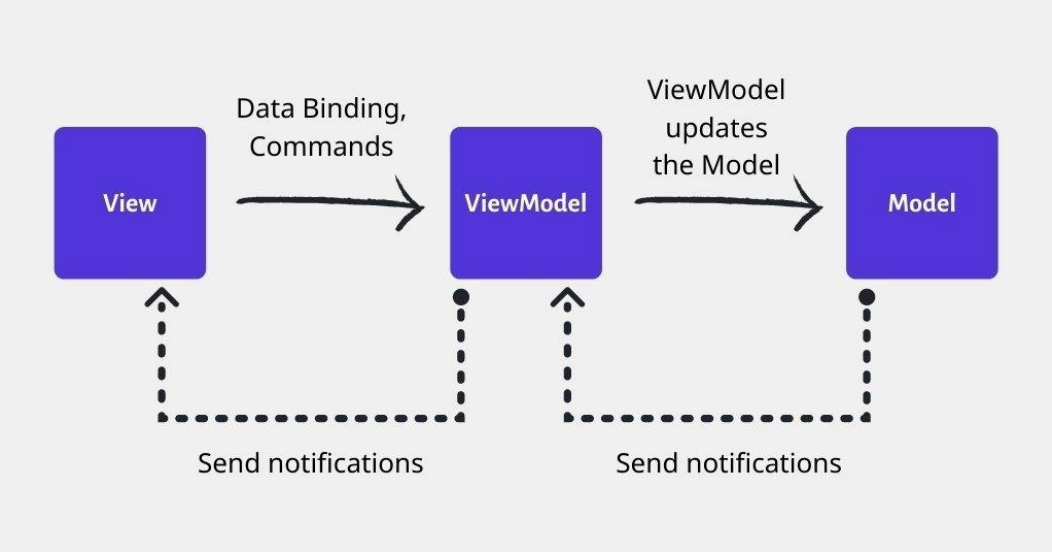
  Predstavlja podatke i poslovnu logiku aplikacije. U ovoj komponenti definiraju se klase koje sadržavaju podatke i metode za rad s tim podacima. Model je odgovoran za rukovanje podacima, validaciju, te poslovne operacije unutar aplikacije.

- **Pogled**:

  Predstavlja korisničko sučelje aplikacije. U .NET MAUI-u to su obično XAML datoteke koje definiraju izgled i kontrole na ekranu. Pogled je odgovoran za prikazivanje podataka i omogućavanje korisničke interakcije, ali ne sadrži logiku aplikacije.

- **Model-pogleda**:

  Služi kao spojnica između modela i pogleda. Model pogleda dohvaća podatke iz modela, obrađuje ih ako je potrebno, i priprema ih u obliku koji je jednostavan za prikaz u pogledu. Također, model pogleda upravlja korisničkim interakcijama s pogledom, te prema potrebi ažurira model. Model pogleda održava stanje pogleda i omogućava dvosmjerno vezivanje podataka, što olakšava održavanje koda i testiranje aplikacije.



Slika 5.1: Koncept DI-a (Serkan, 2021)

5.2 MVVM i Povezivanje podatka

Jedan od ključnih aspekata MVVM-a je povezivanje podataka (engl. *data binding*), koji omogućava povezivanje podataka iz modela pogleda s UI elementima u pogledu. U .NET MAUI-u, povezivanje podatka omogućava da UI automatski reagira na promjene podataka u modelu pogleda. Kada se svojstvo u modelu pogleda promijeni, povezivanje podatka osigurava da se te promjene automatski reflektiraju u UI-u, bez potrebe za dodatnim kodiranjem u pogledu.

U MVVM prikazu, pogled je obično XAML datoteka koja referencira svojstva iz modela pogleda putem povezivanja. Model pogleda je struktura (obično klasa) koja implementira `INotifyPropertyChanged` sučelje (engl. *interface*), omogućavajući pogledu da prima obavijesti kada se svojstva promijene. Na taj način, kada se podaci u Modelu pogleda ažuriraju, UI se automatski osvježava, održavajući sinkronizaciju između prikaza i podataka.

Ovaj pristup omogućava da se poslovna logika izolira od korisničkog sučelja, čineći aplikaciju modularnijom i lakšom za testiranje i održavanje. Uz to, zahvaljujući dvosmjernom povezivanju podatka, korisnici mogu interaktivno ažurirati podatke u modelu pogleda putem UI-a, a sve promjene će biti odmah vidljive u sučelju.

Ovo objašnjava kako se MVVM i povezivanje podatka koriste u .NET MAUI aplikacijama za postizanje efektivnog i održivog razvoja aplikacija. (Microsoft, 2023)

Prednosti MVVM-a:

Korištenje MVVM obrasca omogućava jasnu razdvojenost odgovornosti između različitih komponenti aplikacije, što olakšava testiranje i omogućava promjene u korisničkom sučelju bez utjecaja na poslovnu logiku. MVVM je također posebno prikladan za Android razvoj, jer omogućava jednostavno povezivanje pogleda i modela pogleda putem povezivanja podataka, olakšavajući deklarativni pristup korisničkom sučelju i potencijalno poboljšavajući performanse aplikacije.

Nedostaci MVVM-a:

MVVM može biti složeniji za implementaciju u usporedbi s drugim arhitekturama, poput MVP-a (engl. *Model-View-Presenter*, skraćeno MVP), što može povećati trud u razvoju aplikacije. Također, MVVM je u razvoju Android aplikacija manje popularan od MVP-a, što može otežati pronalaženje online resursa, vodiča i iskusnih developera s iskustvom u MVVM-u.

6 ARHITEKTURA VOĐENA DOGAĐAJIMA

Ovo je arhitektonski pristup izradi aplikacija koji bi na neki način predstavljao najjednostavniji način, ali ne mnogo praktičan za stvarnu životnu upotrebu. Više bi se mogao koristit za neka početna UI testiranja.

Arhitektura vođena događajima (engl. *Event-Driven Architecture*, skraćeno EDA) temelji se na asinkronoj komunikaciji između različitih dijelova sustava putem događaja. Komponente sustava reagiraju na određene događaje, što omogućuje visok stupanj fleksibilnosti, osobito u sustavima s velikim volumenom podataka. Svaka komponenta sustava može emitirati događaje i pratiti događaje koji generiraju druge komponente. Na primjer, korisničko sučelje može emitirati događaj kada korisnik izvrši određenu akciju, dok drugi dijelovi aplikacije mogu reagirati na taj događaj.

Arhitektura vođena događajima u .NET MAUI omogućuje dinamično ažuriranje korisničkog sučelja temeljem događaja i promjena u podacima. Koristeći *ObservableCollection* i *INotifyPropertyChanged*, aplikacija automatski ažurira UI kad se podaci promijene. Metode za događaje omogućuju povezivanje UI akcija, poput klika na gumb, dok centar za poruke olakšava komunikaciju između različitih dijelova aplikacije putem poruka. Ova arhitektura omogućuje stvaranje UI-ja koji u stvarnom vremenu reagira na promjene podataka i vanjske događaje. (Microsoft, .NET 8 MAUI, 2024)

7 ČISTA ARHITEKTURA

Tijekom zadnjih dvadesetak godina razvilo je se dosta različitih arhitektura kojima je cilj razdvajanje odgovornosti unutar sustava, neke od njih su Onion arhitektura, Heksagonalna arhitektura, Slojna arhitektura… Sve one postižu ovo razdvajanje organiziranjem softvera u slojeve, gdje svaki sloj ima specifičnu ulogu, obično uključujući poslovni sloj i sloj za sučelja.

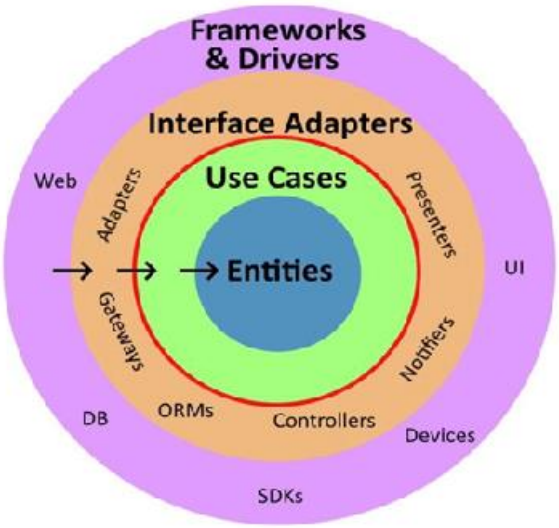
Čista arhitektura (engl. *Clean architecture*) je arhitektonski obrazac za razvoj softvera koji je osmišljen kako bi stvorio održive, fleksibilne aplikacije. Osnovna ideja iza čiste arhitekture je jasna podjela odgovornosti unutar aplikacije, tako da svaka komponenta ima specifičnu ulogu i može se mijenjati neovisno od ostalih komponenti. Ona omogućuje izgradnju sustava koji ne ovisi o okvirima, odnosno nije ovisna o postojanju specifičnih softverskih biblioteka, poslovni sloj se može testirati bez potrebe za UI i baze podataka. Neovisni su o korisničkom sučelju i

bazi podataka, primjerice možemo zamijeniti SQL Server s drugim bazama. Također neovisni su o bilo kojim drugim vanjskim entitetima.

„Kao što smo prethodno rekli, dobra arhitektura mora podržavati:

- Scenarije upotrebe i rad sustava.
- Održavanje sustava.
- Razvoj sustava.
- Implementaciju sustava.“ (Martin, Granning, & Brown, 2017)

7.1 Slojevi čiste arhitekture



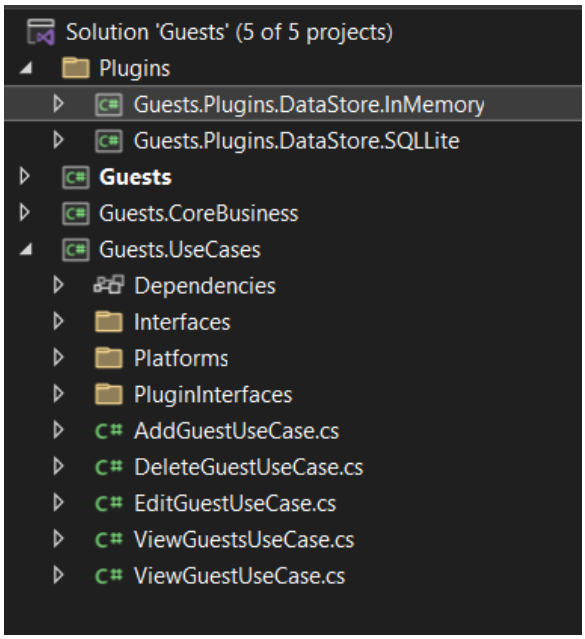
Slika 7.1: Čista arhitektura (Martin, Granning, & Brown, 2017)

Odnos slojeva je prikazan u koncentričnim krugovima i sugerira da se ovisnosti u krugu usmjeravaju prema unutarnjim krugovima. Kod ne smije pozivati funkcije, klase ili varijable iz vanjskih krugova u unutarnje, kako bi se osigurala neovisnost različitih slojeva. Prikazana strelica unutar krugova upravo simbolizira taj smjer ovisnosti unutar arhitekture. Kao što sam već navela pravilo je da kod u vanjskim slojevima smije ovisiti o kodu u unutarnjim slojevima, ali ne i obrnuto. Time se osigurava neovisnost poslovne logike, što omogućuje laku zamjenjivost tehnologija ili sučelja bez utjecaja na osnovnu funkcionalnost sustava.

Slojevi čiste arhitekture (Slika *7.1*):

1. **Entiteti** (engl. *Entities*) Sloj u središtu prikazuje poslovna pravila ili logiku koja je neovisna o specifičnim aplikacijama. Ovi entiteti predstavljaju osnovne poslovne objekte i pravila koja mogu biti primjenjiva u različitim sustavima i kontekstima unutar organizacije. Ovaj sloj je najapstraktniji i ne ovisi ni o čemu vanjskom.
2. **Scenariji upotrebe** (engl. *Use Cases*) – Sljedeći sloj prikazuje aplikacijska poslovna pravila specifična za određene slučajeve upotrebe. Ovaj sloj definira interakcije između korisnika i sustava te sadrži poslovnu logiku koja je specifična za određene aplikacije. Ovdje se također implementira logika koja povezuje entitete sa specifičnim funkcionalnostima. „Prva stavka—use cases (scenariji uporabe)—znači da arhitektura sustava mora podržavati namjeru sustava. Ako je sustav aplikacija za kupovinu, tada arhitektura mora podržavati scenarije uporabe povezane s košaricom za kupovinu. Zapravo, ovo je primarna briga arhitekta i prvi prioritet arhitekture. Arhitektura mora podržavati scenarije uporabe.“ (Martin, Granning, & Brown, 2017)
3. **Adapteri sučelja** (engl. *Interface Adapters*) - Predzadnji sloj obuhvaća prilagodbe koje omogućuju komunikaciju između unutarnjih slojeva (entiteta i slučajeva upotrebe) i vanjskih sustava kao što su baze podataka, korisnička sučelja (UI) ili web servisi. Adapteri su odgovorni za prevođenje podataka i njihovu prilagodbu potrebama unutarnjih slojeva, bez narušavanja njihove neovisnosti.
4. **Okviri i pogoni** (engl. *Frameworks & Drivers*) - Zadnji vanjski sloj prikazuje vanjske sustave i alate koji se koriste za izvršavanje aplikacije, poput baza podataka, web sučelja, uređaja ili vanjskih API-ja. Ovaj sloj sadrži konkretne implementacije koje omogućuju sustavu da komunicira s vanjskim svijetom, ali bez utjecaja na unutarnju poslovnu logiku.

7.2 Upotreba u aplikaciji



Slika 7.2: Podjela koda u slojeve

Čista arhitektura je korištena u aplikaciji koja sadrži popis gostiju (Slika *7.2*). Entiteti tj. poslovna logika se nalaze u direktoriju *Guests.CoreBusiness*(koja sadrži ključne potrebne podatke vezane uz gosta kao što su ime, email, broj telefona i adresa). Scenariji upotrebe se nalaze u direktoriju *Guests.UseCases* koji sadrži logiku specifičnu za određene slučajeve upotrebe, kao što su dodavanje, uređivanje i brisanje gostiju.

Guests.Plugins.DataStore.InMemory i *Guests.Plugins.DataStore.SQLite*

direktoriji predstavljaju konkretne implementacije za pohranu podataka, što su adapteri u čistoj arhitekturi. Svaka implementacija implementira *interface* definiran u nekom od sučelja, što omogućuje jednostavnu zamjenu različitih vrsta pohrane podataka.

U mapi *PlugInInterfaces* su smješteni interfejsi koje koriste adapteri. Interfejsi definiraju kako će sustav komunicirati s vanjskim slojevima, a adapteri implementiraju ove interfejse kako bi omogućili povezivanje s bazama podataka ili drugim servisima.

Guests.Plugins.DataStore je dio koji se odnosi na konkretne baze podataka (SQLite) te spada u vanjski sloj koji ovisi o poslovnoj logici definiranoj u unutarnjim slojevima. XAML datoteke unutar *Guests.Views* direktorija u projektu predstavljaju korisničko sučelje (UI) aplikacije. U kontekstu čiste arhitekture, XAML datoteke također pripadaju vanjskom sloju.

8 SQLite

SQLite je lagana, ugrađena baza podataka otvorenog koda. To je relacijska baza podataka koja podržava SQL jezik, omogućujući korisnicima rad s podacima putem standardnih SQL upita. Jedna od ključnih značajki SQLite-a je da pohranjuje cijelu bazu podataka u jednu datoteku na disku, što ga čini idealnim za aplikacije koje trebaju integriranu bazu podataka bez kompleksnosti povezane s tradicionalnim sustavima upravljanja bazama podataka (DBMS).

SQLite je široko korišten u raznim područjima, uključujući mobilne aplikacije (npr. Android i iOS), web preglednike, softverske alate i ugrađene sustave. Jednostavnost upotrebe, mala veličina i minimalni zahtjevi za konfiguraciju čine ga popularnim izborom za mnoge razvojne projekte. Njegova učinkovitost i brzina čine ga odličnim za manje aplikacije ili one koje se pokreću na uređajima s ograničenim resursima. (Kreibich, 2010)

Korištenje u .NET MAUI:

Prvo je potrebno dodati paket kao što je *sqlite-net-pcl*. Ovaj paket pruža API za rad s SQLite bazama podataka.

Primjer deklaracije u aplikaciji:

```
public class Constants
{
    public const string DatabaseFileName = "GuestsSQLite.db3";

    1 reference
    public static string DatabasePath=>
        Path.Combine(FileSystem.AppDataDirectory, DatabaseFileName);
}
```

Slika 8.1; Definiranje baze u aplikaciji

Inicijalizacija i kreiranje tablice na asinkroni način kako bi se izbjeglo blokiranje glavnog korisničkog sučelja:

```
this.database = new SQLiteAsyncConnection(Constants.DatabasePath);
this.database.CreateTableAsync<Guest>();
```

Slika 8.2: Inicijalizacija baze u aplikaciji

U aplikacijama za popis gostiju i praćenje osobnih financija je implementirana bazu podataka.

Prednosti korištenja SQLite-a u .NET MAUI:

- Izvanmrežni pristup: Aplikacije mogu raditi bez internetske veze, što je odlično za mobilne korisnike.
- Brzina i učinkovitost: SQLite je brz, što je važno za aplikacije koje rade na uređajima s ograničenim resursima.
- Jednostavna implementacija: Implementacija SQLite-a u .NET MAUI je jednostavna, bez potrebe za složenim procesima.

9 APLIKACIJE

Korištenje REST API-ja u .NET MAUI

Korištenje REST (engl. *Representational State Transfer*) API-ja u .NET MAUI aplikacijama omogućava interakciju s web servisima kako bismo dohvatili, poslali, ažurirali ili obrisali podatke putem HTTP zahtjeva. Potrebno je koristiti *HttpClient* za slanje različitih vrsta HTTP zahtjeva (GET, POST, PUT, DELETE) prema REST API-ju. To je klasa koja se koristi za slanje HTTP zahtjeva i primanje HTTP odgovora iz REST API-ja. (Microsoft, .NET 8 MAUI, 2024)

Logika korištenja zahtjeva je slična kao i u drugim alatima koje smo dosad koristili, ali razlika je u implementaciji u kodu.

GET ZAHTJEV: Koristi *GetAsync* metodu za slanje zahtjeva na određeni URL i dohvaća listu objekata. Ako je zahtjev uspješan (*IsSuccessStatusCode*), odgovarajuća JSON struktura se raspakira u listu objekata koje definiramo u kodu.

PUT ZAHTJEV: Ažurira postojeći korisnički zapis na temelju ID-a. Serijalizirani JSON s novim podacima šalje se na URL pomoću *PutAsync* metode.

POST ZAHTJEV: Ovaj zahtjev kreira novog korisnika slanjem JSON podataka na određeni URL. JSON se serijalizira iz objekta User i šalje kao *StringContent*.

DELETE ZAHTJEV: Briše objekt s određenim parametrom koristeći *DeleteAsync* metodu.

9.1 Aplikacija za vremensku prognozu

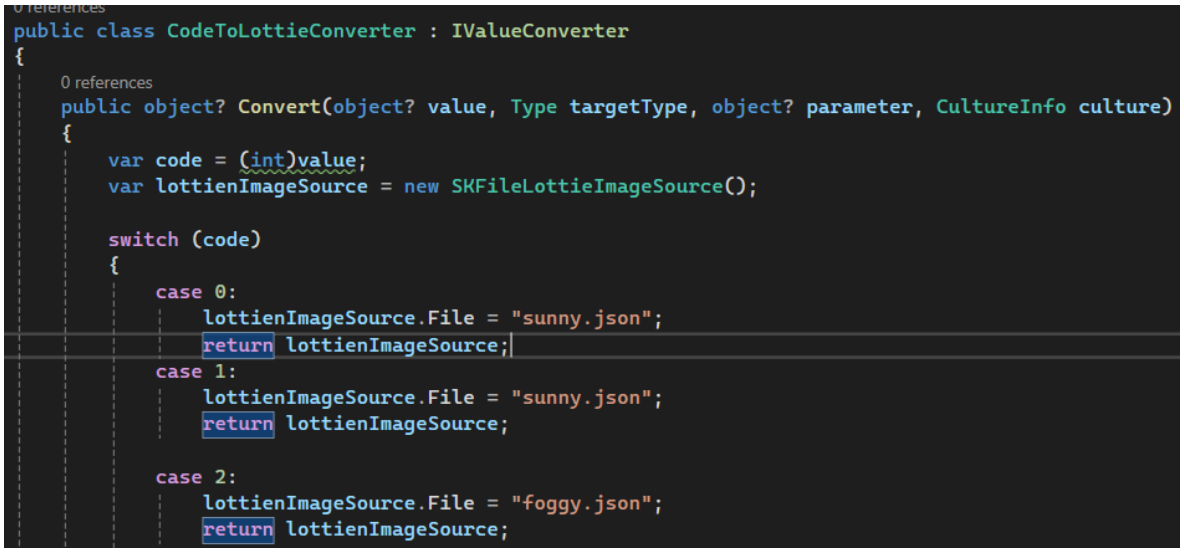
Korištenje geografske lokacije u .NET MAUI aplikacijama omogućava pristup geografskim koordinatama (zemljopisna širina i zemljopisna dužina) korisnika ili za određenu adresu, što može biti korisno za funkcionalnosti poput praćenja lokacije, prikazivanja korisnikovog položaja na mapi ili dobivanja informacija o točnoj lokaciji na temelju unesenih podataka. Odnosno u ovome slučaju za lokacije gradova u kojima želite vidjeti vremensku prognozu za taj dan i sljedeći tjedan. U projektu je korišten *Geocoding* API, koji je ugrađen u .NET MAUI, za pretvaranje adrese u geografske koordinate. Geokodiranje je proces koji omogućava dohvaćanje koordinata na temelju unesene adrese.

Besplatne informacije o vremenskoj prognozi mogu se dobiti sa *Free Weather Api*. (Open-meteo, n.d.)

U **.**NET MAUI, **konverteri** su klasa koja se koristi za pretvaranje vrijednosti iz jednog tipa podataka u drugi unutar podataka za vezivanje (*data binding*). Najčešće se koriste u vezivanju podataka kako bi prilagodili podatke koji se prikazuju u korisničkom sučelju. U ovom slučaju to je pretvaranje brojeva koji predstavljaju kod za vrijeme u tekst (npr. 0-sunčano, 45-maglovito itd.) i u animacije formata *.json*.

Konverteri u .NET MAUI implementiraju sučelje *IValueConverter*, koje se sastoji od dvije metode:

- ***Convert*** – pretvara vrijednost iz izvornog tipa u ciljani tip, najčešće za prikaz u korisničkom sučelju.
- ***ConvertBack*** – pretvara vrijednost natrag iz ciljanog tipa u izvorni tip (ako je potrebno, npr. kod dvosmjernog vezivanja podataka).



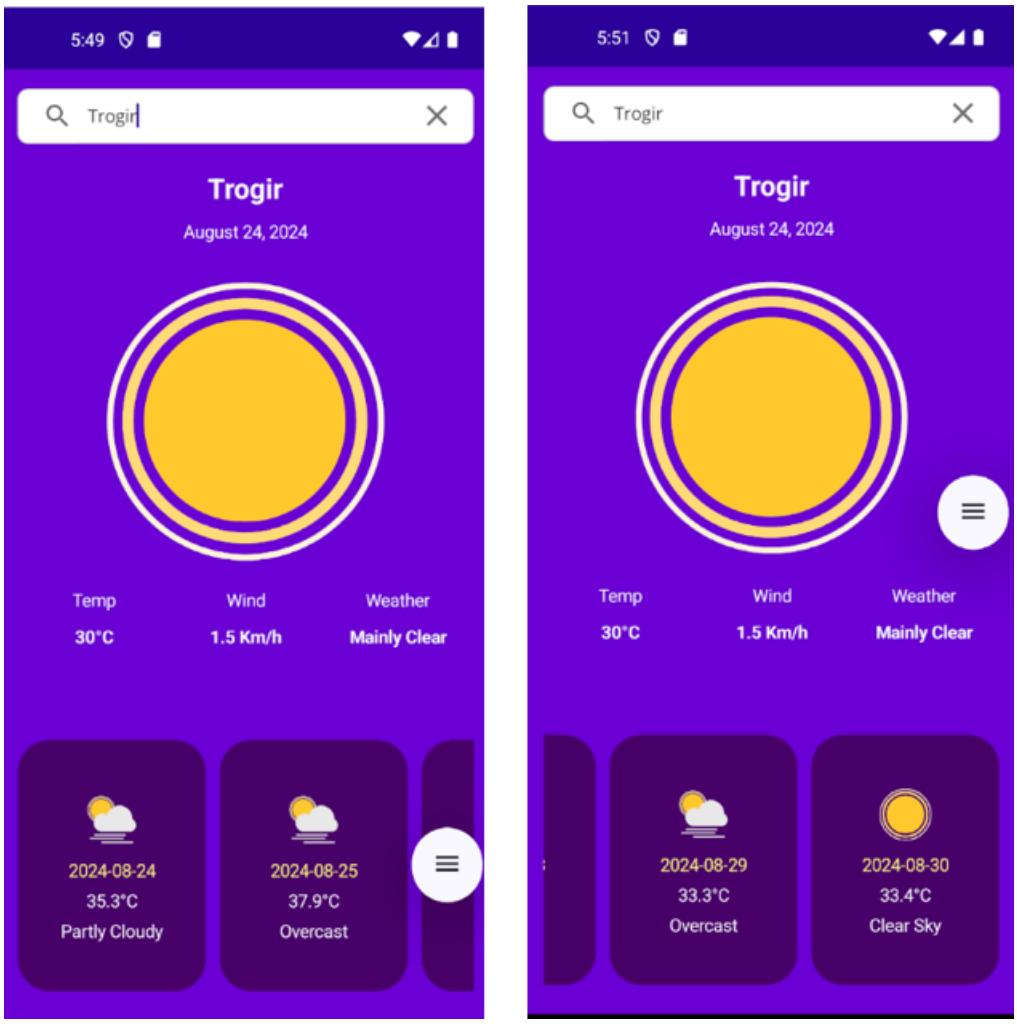
```
0 references
public class CodeToLottieConverter : IValueConverter
{
    0 references
    public object? Convert(object? value, Type targetType, object? parameter, CultureInfo culture)
    {
        var code = (int)value;
        var lottienImageSource = new SKFileLottieImageSource();

        switch (code)
        {
            case 0:
                lottienImageSource.File = "sunny.json";
                return lottienImageSource;
            case 1:
                lottienImageSource.File = "sunny.json";
                return lottienImageSource;

            case 2:
                lottienImageSource.File = "foggy.json";
                return lottienImageSource;
```

Slika 9.1: Konverter za animacije

U ovoj aplikaciji konverter za animacije koristi klasu *SKFileLottieImageSource* iz biblioteke *SkiaSharp*, točnije njezine *Lottie* ekstenzije *SkiaSharp.Extended.UI.Lottie*, koja omogućuje renderiranje *Lottie* animacija u aplikacijama temeljenim na SkiaSharp grafici. Ključna komponenta, *SKFileLottieImageSource* učitava animaciju iz .json datoteke i prikazuje je u aplikaciji. U ovom slučaju, *switch-case* naredba omogućuje odabir različitih *Lottie* animacija na temelju ulazne brojčane vrijednosti (Slika *9.1*).



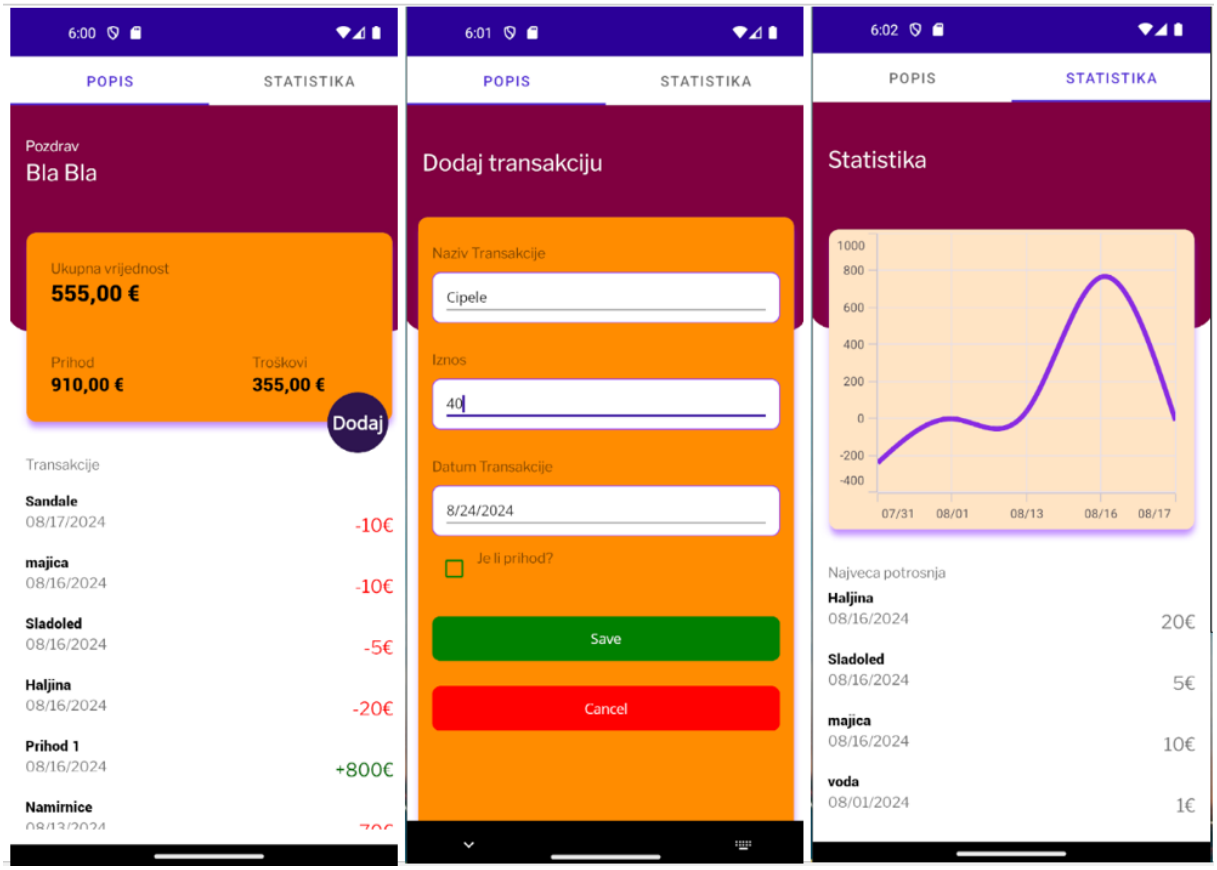
Slika 9.2: Prikaz vremenske prognoze u Trogiru kroz tjedan

9.2 Aplikacija Tokovi Novca

Aplikacija za praćenje vlastitih financija. Sastoji se od 3 stranice, jedna sadrži popis svih prihoda i troškova, druga služi za dodavanje novih transakcija, a treća prati statistiku zarade i potrošnje po datumima. Ovdje sam koristila MVVM obrazac za izradu aplikacije što je objašnjeno u poglavlju 5.

Za prikaz statistike koristi se ***SfCartesianChart*** komponenta iz *Syncfusion* biblioteke u .NET MAUI aplikaciji, koja služi za prikazivanje kartezijanskih grafova. U ovom primjeru koristi se *SplineSeries*, vrsta linijskog grafikona s glatkim zakrivljenim linijama koje povezuju točke na temelju podataka.

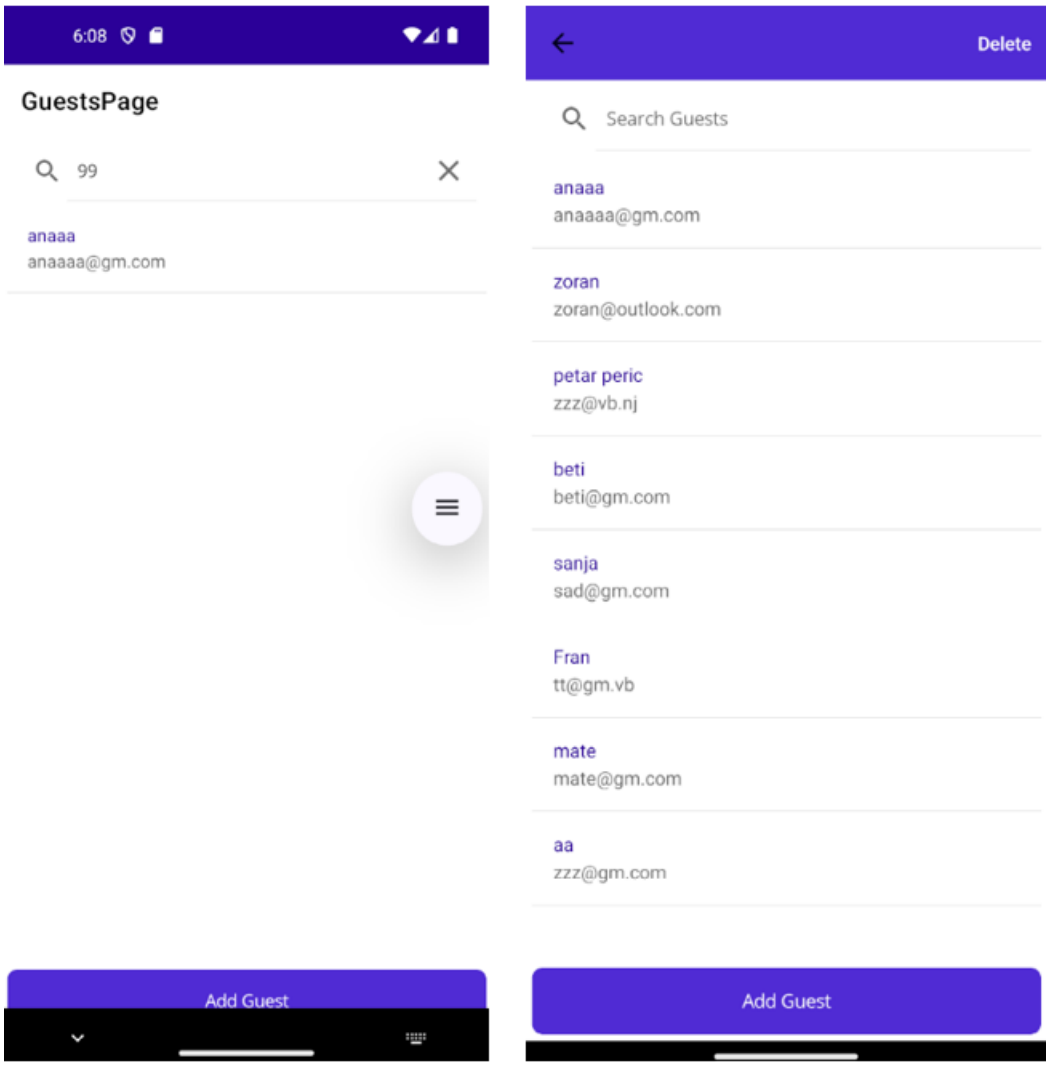
SfCartesianChart je glavna komponenta za prikaz kartezijanskog grafikona. Na *XAxes* se koristi *CategoryAxis*, koja prikazuje kategorije podataka na X-osi, u ovom slučaju vrijednosti vezane uz *ShownDate*. *YAxes* koristi *NumericalAxis*, koji prikazuje numeričke vrijednosti na Y-osi, a ovdje se koristi za prikazivanje vrijednosti atributa *TransactionsTotal*. *SplineSeries* prikazuje seriju podataka s glatkim zakrivljenim linijama koje povezuju pojedinačne podatke na grafikonu, stvarajući prikaz promjene u podacima.



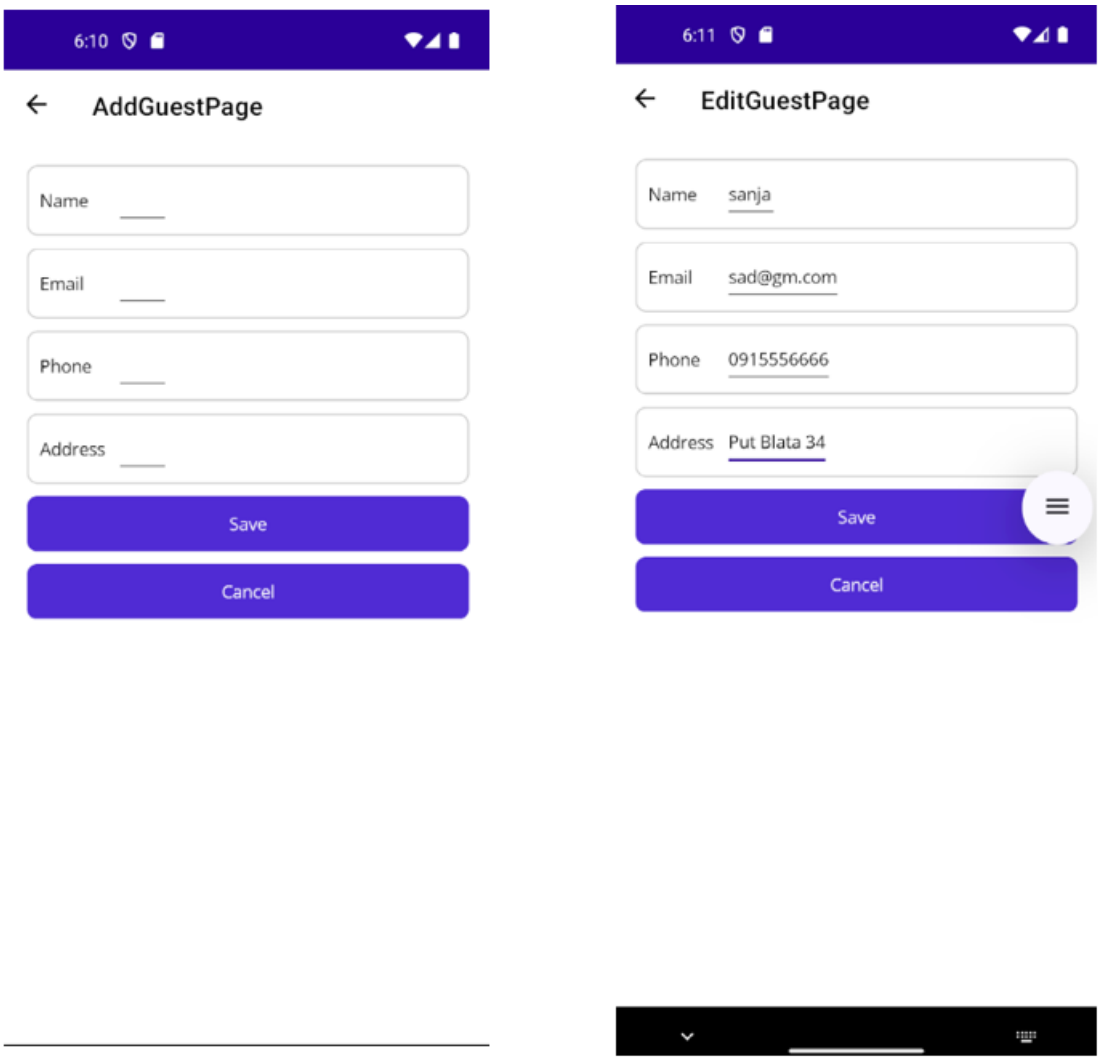
Slika 9.3: Stranice sa prikazom financijskog stanja, novom transakcijom i statistikom trošenja

9.3 Aplikacija za popis gostiju

Ova aplikacija prati popis gostiju. Kod u aplikaciji je organiziran pomoću čiste arhitekture (opisano u poglavlju 7.2). Gosti se mogu dodavati, mijenjati, pretraživati po svojim podacima te brisati.



Slika 9.4: Mogućnost pretrage liste gostiju



Slika 9.5:Dodaj ili uredi gosta

ZAKLJUČAK

.NET MAUI je zanimljivo razvojno okruženje, posebno zbog mogućnosti stvaranja cross-platformskih aplikacija s jednom bazom koda, što značajno smanjuje vrijeme razvoja i kompleksnost projekta. Posjeduje jako dobru integraciju s .NET ekosustavom i podršku za različite platforme kao što su iOS, Android, Windows i macOS, što čini MAUI opširnim alatom za tvrtke koje žele pokriti širok spektar uređaja s minimalnim naporom.
Korištenje XAML u .NET i njegova deklarativna priroda i vrlo dobra podrška za MVVM obrazac omogućuju efikasno upravljanje UI-em i podatkovnim povezivanjem. Međutim, u usporedbi s HTML-om i CSS-om, XAML može biti složeniji za početnike, s manje intuitivnim alatima za dizajn i zahtjevnijom sintaksom. HTML i CSS pružaju fleksibilniji i moderniji način dizajna.

Čista arhitektura je vrlo slojevita, s jasnim razgraničenjem između slojeva. Iako ova struktura pomaže u organizaciji i održavanju velikih aplikacija, može biti previše složena i nepotrebno opterećujuća za manje .NET MAUI aplikacije. S druge strane, MVVM arhitektura je jednostavnija i prirodno se uklapa u .NET MAUI, jer se fokusira na vezu između korisničkog sučelja i poslovne logike. S manje slojeva i komponenti za upravljanje, MVVM je lakši za implementaciju i održavanje, što ga čini pogodnijim za manje projekte. Također za testne primjere je puno bolje koristiti MVVM obrazac ili arhitekturu vođenu događajima sa statičkim repozitorijem za testiranje rada sučelja.

Literatura

BrightDevelopers. (2023). *Dependency Injection*.
Dohvaćeno iz https://www.brightdevelopers.com/simple-introduction-to-what-is-dependency-injection/a-dependencies-b/

Britch, D. (2023). *XAML*.
Dohvaćeno iz https://learn.microsoft.com/en-us/dotnet/maui/xaml/?view=net-maui-8.0

dotnet, M. (2024). *What is .NET Framework?*
Dohvaćeno iz https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework

Geeks for Geeks. (2023). *Introduction to Model View View Model (MVVM)*.
Dohvaćeno iz https://www.geeksforgeeks.org/introduction-to-model-view-view-model-mvvm/

Kathiresan, S. G. (2024). *Learn How to Use Dependency Injection in .NET MAUI*.
Dohvaćeno iz https://www.syncfusion.com/blogs/post/learn-how-to-use-dependency-injection-in-net-maui

Kathiresan, S. G. (2024). *Xamarin Versus .NET MAUI*.
Dohvaćeno iz https://www.syncfusion.com/blogs/post/xamarin-versus-net-maui

Kreibich, J. (2010). *Using SQLite*. O'Reilly Media, Inc.

Martin, R. C., Granning, J., & Brown, S. (2017). Clean Architecture:. U R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software*.

Microsoft. (2023). *Data binding and MVVM*.
Dohvaćeno iz https://learn.microsoft.com/en-us/dotnet/maui/xaml/fundamentals/mvvm?view=net-maui-8.0

Microsoft. (2024). *.NET 8 MAUI*.
Dohvaćeno iz https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0

Microsoft. (2024). *.NET Framework*.
Dohvaćeno iz https://learn.microsoft.com/en-us/training/modules/dotnet-introduction/2-what-is-dotnet

Microsoft. (2024). *Dependency injection*.
Dohvaćeno iz https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/dependency-injection?view=net-maui-8.0

Mohanan, R. (2024).
Dohvaćeno iz https://www.spiceworks.com/tech/devops/articles/what-is-dependency-injection/

Nazarevich, D. (2023). *.NET MAUI and the future of Xamarin*.
Dohvaćeno iz https://innowise.com/blog/net-maui-vs-xamarin/

Open-meteo. (n.d.). *Free Weather Api*.
Dohvaćeno iz Open-Meteo: https://open-meteo.com/

Serkan. (2021).
Dohvaćeno iz https://www.serkanseker.com/xamarin-forms-mvvm-pattern/

Skraćenice

API	*Application Programming Interface*	sučelje za programiranje aplikacija
DI	*Dependency Injection*	tehnika definiranja odnosa objekata
EDA	*Event-driven architecture*	arhitektonski stil komunikacijom događajima
JSON	*JavaScript Object Notation*	notacija JavaScript objekta
IL	*Intermediate Language*	međukod
MAUI	*Muli-platform App UI*	.NET okvir za izradu aplikacija
MVVM	*Model View-ViewModel*	arhitektonski stil razdvajanja podataka
REST	*Representational State Transfer*	reprezentacijski prijenos podataka
SQL	*Structured Query Language*	strukturirani upitni jezik
XAML	*Extensible Application Markup Language*	deklarativni jezik
UI	*User Interface*	korisničko sučelje