Upravljanje podacima iz paketomata

Hlevnjak, Juraj

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 379

UPRAVLJANJE PODACIMA IZ PAKETOMATA

Juraj Hlevnjak

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 379

UPRAVLJANJE PODACIMA IZ PAKETOMATA

Juraj Hlevnjak

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 379

Pristupnik:	**Juraj Hlevnjak (0036524365)**
Studij:	Računarstvo
Profil:	Znanost o mrežama
Mentor:	izv. prof. dr. sc. Dario Bojanjac

Zadatak: **Upravljanje podacima iz paketomata**

Opis zadatka:

Osnovna funkcija paketomata je preuzimanje i dostava paketa u pretince. Oni ne prate vremenske parametre iz okoline koji bi mogli koristiti u svrhu dodatnog praćenja onečišćenja zraka, temperature, vlage te ovise o stalnom napajanju električne energije. Zbog toga, tvrtka Piklean razvija paketomate koji mogu izdržati godinu dana bez struje (mobilni su) i prikupljaju podatke iz okoline. Važno je naglasiti da ih svaka firma može iznajmiti za svoje potrebe zbog B2B modela. Njihova infrastruktura će biti korištena za dohvat i prikaz podataka. U sklopu diplomskog rada razvit će se mobilna aplikacija koja će dohvaćati podatke o vremenskim uvjetima iz paketomata najnovije generacije. Dodatno, treba dohvatiti prvi slobodan termin i prikazati QR kod koji služi za otključavanje pretinca. Cilj rada je poštivati MVVM standard u .Net MAUI okruženju. U sklopu rada bit će prikazani razni primjeri dobrih i loših praksi koje treba koristiti ili izbjegavati prilikom izrade projekta.

Rok za predaju rada: 28. lipnja 2024.

SADRŽAJ

1. Uvod

Danas na tržištu postoje paketomati koji moraju biti priključeni na stalni izvor napajanja, uz to, infrastrukture se ne može dijeliti, npr. GLS-ov paketomat koristi samo GLS. Cilj je postaviti paketomat koji svi mogu koristiti (iznajmiti pretince na određen period) i spremati u njih stvari.

Ovaj rad će biti fokusiran na aplikaciju koja će prikazivati stanje okoline i upravljati rezervacijama na već postojeću infrastrukturu tvrtke koja je investirala u njezinu izgradnju. Mobilna aplikacija će se spajati na backend koji se povezuje na IoT. Prvo će se proučiti .Net MAUI tehnologija a zatim će se proći pokazati primjeri implementacije. Važno je naglasiti cijela infrastruktura (backend+IoT) će biti korištena ali njen način rada je poslovna tajna nad kojom se ne može tražiti intelektualno vlasništvo koje faks sadržava nad ovim radom.

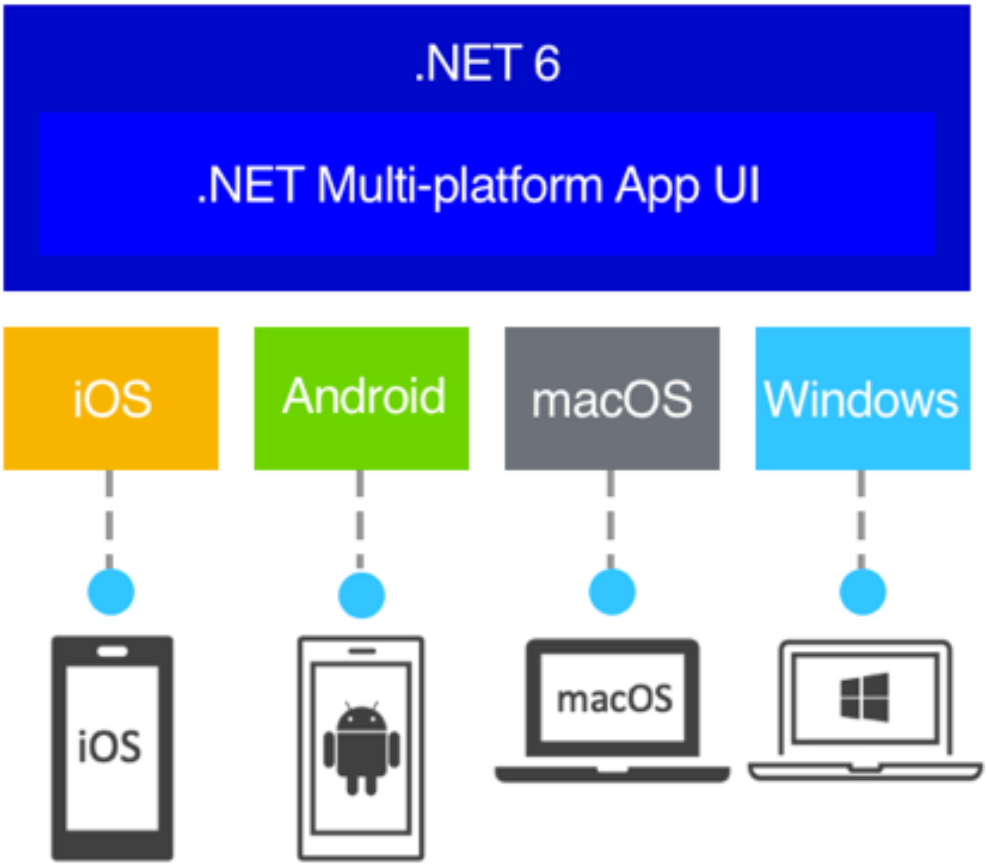
Slika 1.1: Primjer paketomata

.Net MAUI je *cross-platform framework* za razvoj aplikacija. Početna zamisao je da se s pomoću *c#-a* piše kod koji će biti zamjena za Javu i Swift. Java se koristi za pisanje Android aplikacija, a Swift iOS-ovih. Prije je za razvoj iste aplikacije bilo potrebno minimalno dva developera. Jedan koji će razvijati u Swiftu koji je potisnuo Objective-C a drugi u Javi. Pojavila se ideja kojom se pokušalo stvoriti apstraktni sloj iznad ovih jezika kojima bi se objedinile njihove funkcionalnosti. Tako je nastao Xamarin. Ideja je da se stvori apstraktni sloj iznad te dvije platforme čije će se sposobnosti programirati s pomoću zajedničkog jezika c#-a. Najveća prednost je što će jedna osoba razviti aplikaciju za dvije odvojene platforme što bi inače bio posao za dvoje iskusnijih developera. Najveća prednost je isplativost jer s pomoću jedne platforme razvijamo dva izgledno ista ali različita programska rješenja. Početno ime je bilo Ximian koje se kasnije promijenilo. Uz Android i iOS, podržani su još i tvOS, watchOS i MacOS. Xamarin Forms se razvio naknadno sa sobom donio XAML. On omogućava konfiguriranje izgleda aplikacije i automatsko osvježavanje (hot reload) bez potrebe za ponovnim pokretanjem kompajliranja koda. Za XAML možemo reći da je to XML na steroidima. Xamarin Studio je postojao kao neovisni IDE koji danas nije podržan nego dolazi kao za ekstenzija za Visual Studio. Xamarin Test Cloud ima mogućnost testiranja aplikacija. Cijelu povijest nastanka i razvoja moguće je pronaći u literaturi Wikipedia (2021).

Slika 1.2: Xamarin logo

2. .NET MAUI

Konačno, možemo proučiti MAUI zasnovanom na .NET-u6 koji je opisan u prethodnom poglavlju. Najveća prednost mu je podrška za desktop verzije aplikacija i open-source kod. Napredne funkcije i pravilo pisanje bit će praćeni po materijalima Rosas (2021) i Montemagno (2022).



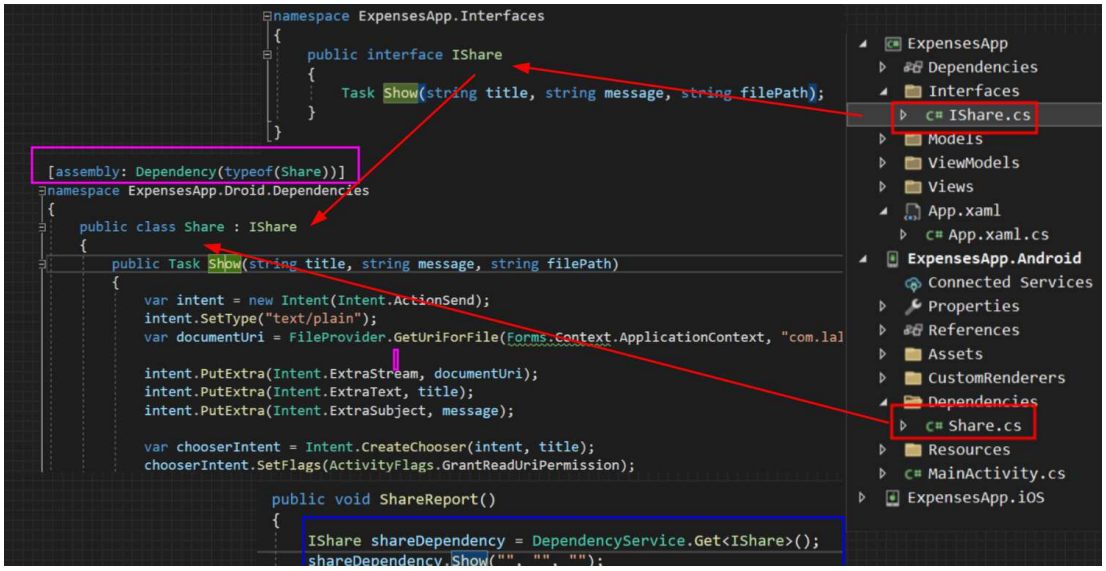
Slika 2.1: Logo

2.1. Način rada

Prema slici 2.4, prvo pišemo kod u XAML-u ili c#-u, ovisno o arhitekturi i potrebi te brzini razvoja. On se prevodi u .NET MAUI kojeg koristi .NET Android. Bitno je napomenuti da .NET Android je običan kod u c#-u koji se onda prevodi u nativni Android preko Mono Runtimea.

2.2. Pristupanje nativnim funkcijama

Neke implementacije funkcionalnosti se razlikuju na Android i iOS platformama. Stoga, sada ćemo proći detaljne korake kako se približiti svakoj arhitekturi posebno. Za vježbu, pogledat ćemo kako se dijeli datoteka. Ona ne može biti univerzalna nego se na svakoj platformi mora posebno postaviti.



Slika 2.2: Pristup nativnim funkcijama

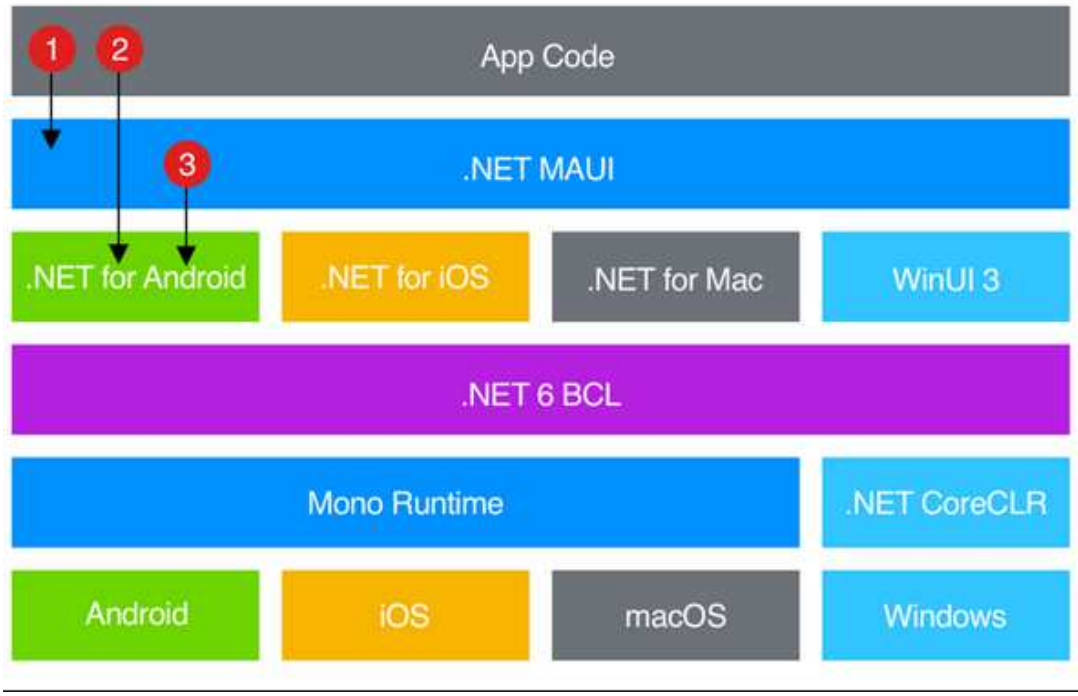
Prvo smo napisali sučelje **IShare.cs** koje sadrži metodu *Show*. Tako možemo ubaciti tu metodu gdje god želimo unutar dijeljenog koda dostupnog za sve platforme. Taj postupak se zove Denepndecy Injection i vidljiv je na slici 2.3. Nije to jedini način, općenita praksa je da se unutar **App.xaml.cs** povezuju sva sučelja i klase dostupna za injektiranje. Taj pristup pridonosi boljoj organizaciji i čitljivosti.

Nadalje, prema slici 2.2 vidimo implementaciju unutar Xamarin.Androida gdje klasa **Share.cs** realizira **IShare.cs** sučelje. Točna realizacija ovisi o platformi i nije bitna.

Važno je napomenuti da linija *[assembly : Dependency(typeOf(Share))]* šalje ovisnost s Android platforme i omogućuje dijeljenje te dohvat metode (slika 2.3).

```
IShare shareDependency = DependencyService.Get<IShare>();
shareDependency.Show("", "", "");
```

Slika 2.3: Dependency injection



Slika 2.4: Primjer prevođenja

3. .NET6

Nakon Xamarin Formsa, dolazi MAUI kao najnovija nadogradnja koja koristi .Net 6. U svakom potpoglavlju bit će pojašnjene značaje zbog kojih se poboljšava čitljivost i performanse koda.

3.1. Globalne direktive

Umjesto da se unutar svake klase navode direktive koji se paketi koriste unutar datoteke, dovoljno je unutar jedne navesti sve tako da se ne moraju ponavljati i refaktorirati ako se više ne koriste.

```
global using Model;  // Global usings apply to entire project
// using System;     // Implicit usings for each project type
```

Slika 3.1: Primjer globalnih direktiva

3.2. Zagrade imenskog prostora (namespace-a)

Prije ovog poboljšanja, bilo je potrebno obuhvatiti klasu s pomoću zagrada imenskog prostora. Tako se cijeli kod morao nepotrebno uvlačiti u desno što povećava mogućnost pogreške dodavanja ili brisanja zagrade te smanjuje čitljivost.

```
namespace Model;     // File-scope namespace, no indentation needed
```

Slika 3.2: Deklariranje imenskog prostora kao varijable

3.3. Tip varijable u lambdi

Lambde u jednoj liniji mijenjaju desetke linija koda. Često je izvršavanje u konstantnom vremenu pa su performanse maksimalne. Na njih se treba naviknuti i često nije jasno kojeg su tipa objekti po kojima lista iterira. Stoga je dodan tip kako bi se povećala čitljivost.

```
// Lambdas have types
var parse = (string s) => int.Parse(s); // Func<string, int>
```

Slika 3.3: Primjer globalnih direktiva

3.4. Recordi

Strukture za razliku od klasa ne mogu podržavati nasljeđivanje, prenose se po vrijednosti, ne mogu imati referencu na prazan pokazivač i nemaju memorije po svakoj novoj instanci, Zajedno obje sadrže metode i događaje te mogu podržavati sučelja. U c#-u 10, moguće je strukturu i klasu deklarirati kao record što prije nije bilo moguće. Njih koristimo kada želimo thread-safe opciju, nepromjenjivost i usporedbe vrijednosti umjesto referenci.

```
// Records can be structs as well as classes
public record struct Order(Customer Customer, string Kind, int Items);
public record class Customer(string Name, string Address);
```

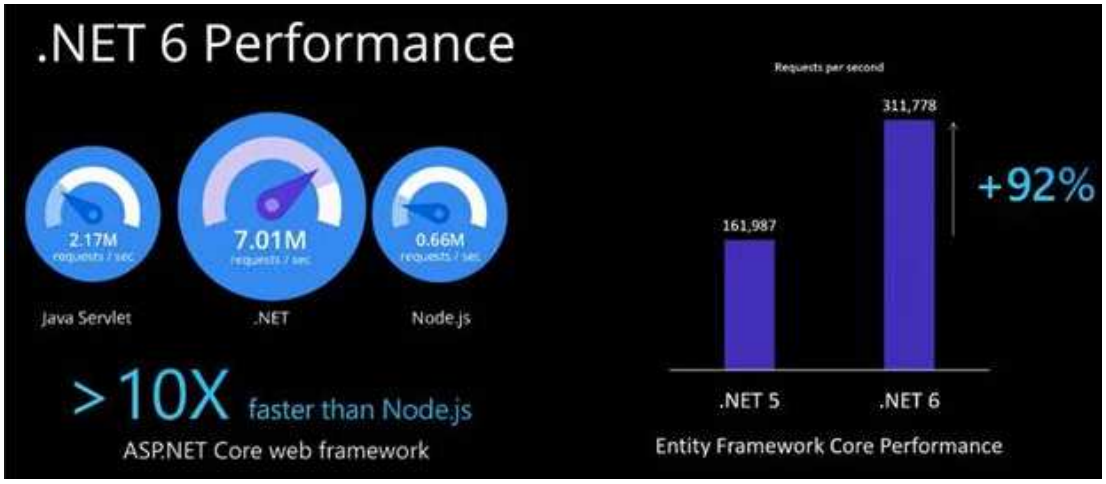
Slika 3.4: Primjer globalnih direktiva

3.5. Optimizacije performansi

Cijeli FileStream koji služi za otvaranje i čitanje datoteka je napisan ponovno tako da je povećana brzina i pouzdanost. Crosgen2 omogućava generiranje jedinstvenog nativnog koda u modu za pokretanje. To znači da možemo podići mašinu za generiranje koda za sve ciljne skupine.

3.6. .NET5 vs .NET6

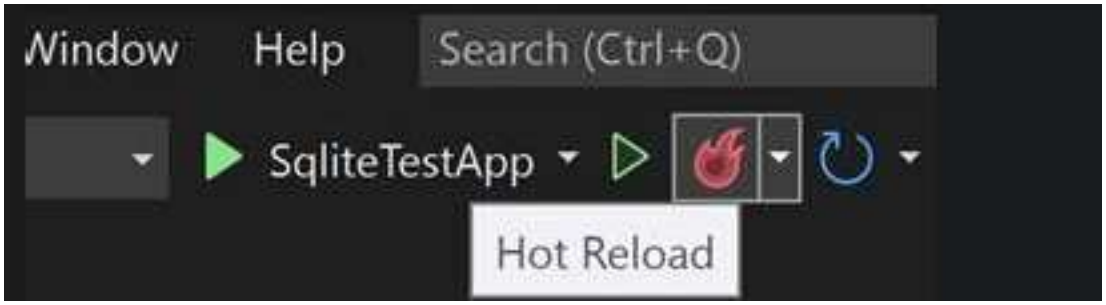
Prvo postoje razlike u podržanim programskim jezicima. U .NET-u 5, podržan je najviše c#9 dok je u .NET-u6 c#10. On podržava dodatno Windows Arms65, MacOS i Apple Silicon u odnosu na .NET5 i ima poboljšanu brzinu izvršavanja.



Slika 3.5: Karakteristike verzije 6

3.7. Hot reload

Najgore je stalno pokretati ponovno kod koji smo napisali, zato se sve više teži prema *vrućem pokretanju* kojim su nakon spremanja već vidljive izmjene. Naravno, to je samo moguće preko skriptnih konfiguracijskih jezika kao što je XAML jer c# treba kompajlirati po standardnom postupku.



Slika 3.6: Izgled ikone

4. XAML

Prije je bilo navedeno da je XAML skriptni jezik koji služi za lakše prikazivanje kontrola i automatske izmjene (potpoglavlje 3.7). Sada ćemo proučiti njegovu srž koja će nam kasnije omogućiti razumijevanje svih prednosti koje donose biblioteke. Isto tako, dokumentacija Microsoft (2022) ne prati MVVM [1] obrazac oblikovanja. Sada ćemo na ovim primjerima vidjeti što se sve ne preporučuje raditi i kasnije ćemo pokazati ispravan način.

4.1. Obično povezivanje podataka

Cilj nam je na jednostavnom primjeru klizača rotirati tekst i pokazati kako radi XAML te kako ne treba povezivati podatke. Nije taj način loš, samo se ne poštuje glavna pretpostavka, odvajanje podataka (*eng. decoupling*).

Slika 4.1: Poželjna funkcionalnost

[1] https://learn.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

4.2. Izgled i funkcionalnost

Na slici ispod na lijevoj strani su prikazani tagovi koji grade izgled aplikacije. Unutar taga *ContentPage* vidimo da se direktno spajamo na klasu s pomoću naredbe *x:Class=* kako bi pristupili svojstvima unutar nje. Taj pristup ćemo kasnije promijeniti u svrhu zadovoljavanja MVVM arhitekture. Nadalje, unutar taga *StackLayout* samo dodajemo elemente koje želimo prikazati. Postoje još *Grid, Relative, Absolute* ali oni samo definiraju različite načine slaganja elemenata. Primjer još jednog kršenja arhitekture vidimo po atributima *x:Name*. Na istoj slici desno primjećujemo kako je BindingContext postavljen na cijeli objekt klizača, a njegova vrijednost *"Value"* upravlja okretanjem *Label.RotationProperty*. Važno je napomenuti da slična naredba u XAML-u (*BindingContext="slider"*) neće raditi.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DataBindingDemos.BasicCodeBindingPage"
             Title="Basic Code Binding">
    <StackLayout Padding="10, 0">
        <Label x:Name="label"
               Text="TEXT"
               FontSize="48"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand" />

        <Slider x:Name="slider"
                Maximum="360"
                VerticalOptions="CenterAndExpand" />
    </StackLayout>
</ContentPage>
```

```
public partial class BasicCodeBindingPage : ContentPage
{
    public BasicCodeBindingPage()
    {
        InitializeComponent();

        label.BindingContext = slider;
        label.SetBinding(Label.RotationProperty, "Value");
    }
}
```

Slika 4.2: XAML i c#

Istu funkcionalnost moguće je prikazati samo u XAML-u.

```
<Label Text="TEXT"
       FontSize="80"
       HorizontalOptions="Center"
       VerticalOptions="CenterAndExpand"
       BindingContext="{x:Reference slider}"
       Rotation="{Binding Value}" />
```

Slika 4.3: Korištenje samo XAML-a

Ono što XAML čini nečitljivim je to što postoji puno kombinacija i argumenata koji se mogu zanemariti. Po slici 4.4 vidimo kako *BindingContext* možemo mijenjati sa *Source* i *Path* argumentima, ako im zamijenimo mjesta, onda *Path* možemo izostaviti.

Možemo još dodatno stvarati kompleksne objekte (*Label.Scale, Binding.Source*), ali nije preporučljivo. Isto tako, vrijedi pravilo naslijeđivanja[2] unutar tagova.

```
 <Label Text="TEXT"
        FontSize="40"
        HorizontalOptions="Center"
        VerticalOptions="CenterAndExpand"
        Scale="{Binding Source={x:Reference slider},
                        Path=Value}" />
```

```
Scale="{Binding Value, Source={x:Reference slider}}" />
```

```
<Label Text="TEXT"
       FontSize="40"
       HorizontalOptions="Center"
       VerticalOptions="CenterAndExpand">
    <Label.Scale>
        <Binding Source="{x:Reference slider}"
                 Path="Value" />
    </Label.Scale>
</Label>
```

```
<Label Text="TEXT"
       FontSize="40"
       HorizontalOptions="Center"
       VerticalOptions="CenterAndExpand">
    <Label.Scale>
        <Binding Path="Value">
            <Binding.Source>
                <x:Reference Name="slider" />
            </Binding.Source>
        </Binding>
    </Label.Scale>
</Label>
```

Slika 4.4: Različiti zapisi iste funkcionalnosti

4.3. Value converters

Kako bi dobili automatsku verziju vrijednosti koja je integrirana unutar XAML-a, potrebno je implementirati sučelje (slika 4.5). U metodi *Convert* moramo prvo niz

[2] https://learn.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/data-binding/basic-bindings#binding-context-inheritance

znakova pretvoriti u tip *double*. Tada množimo dobivenu vrijednost i zaokružujemo je kako bi realni izraz prebacili u cjelobrojni. Metodu *Convert* koristimo da vrijednost iz XAML-a pošaljemo metodi *ConvertBack* iz c#-a. Slanje parametara moguće je provesti iz skripte ili koda iza. (slika 4.6).

```
public class DoubleToIntConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (int)Math.Round((double)value * GetParameter(parameter));
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (int)value / GetParameter(parameter);
    }

    double GetParameter(object parameter)
    {
        if (parameter is double)
            return (double)parameter;

        else if (parameter is int)
            return (int)parameter;

        else if (parameter is string)
            return double.Parse((string)parameter);

        return 1;
    }
}
```

Slika 4.5: Klasa za pretvaranje vrijednosti

```
<Label Text="{Binding Red,
                      Converter={StaticResource doubleToInt},
                      ConverterParameter=255,
                      StringFormat='Red = {0:X2}'}" />
```

```
binding.ConverterParameter = 255;
```

Slika 4.6: Postavljanje parametara

4.4. Compiled Bindings

Prednosti ovog pristupa su poboljšanje brzine u odnosu na klasično povezivanje čak do osam puta koje koristi notifikacije za promjenu vrijednosti, dvadeset puta za one koje ne koristi. Postavljanje *BindingContexta* je pet puta brže ako se koristi i sedam puta ako ne. Osim poboljšanja odvijanja aplikacije, javljaju se greške.

4.4.1. Omogućavanje postavki za kompajliranje

Važna je samo linija u kojoj piše *[assembly: ...]* u kojoj omogućavamo tu postavku za cijeli projekt. Ona može biti bilo gdje, ali poželjno je staviti je u **Assembly.cs** zbog standarda koji će pomoći ostalima kada će čitati kod.

```
using Xamarin.Forms.Xaml;
...
[assembly: XamlCompilation (XamlCompilationOptions.Compile)]
namespace PhotoApp
{
  ...
}
```

Slika 4.7: Postavka za bolje performanse

4.4.2. Upotreba kompajliranih postavki

Prije pokazivanja kako se koristi, moramo uočiti da ako koristimo *Source* svojstvo koje se postavlja s pomoću *x:Reference* ekstenzije, nije moguće koristiti ovakva poboljšanja jer ne mogu biti izvršena u vremenu kompajliranja. Na slici 4.8 vidimo kako smo unutar statičkih elemenata označenih crveno, stavili i *x:DataType...*, koji je podcrtan zelenom bojom i poželjno ga je koristiti.

```
<Grid>
    ...
    <ListView x:Name="colorListView"
              ItemsSource="{x:Static local:NamedColor.All}"
              ... >
        <ListView.ItemTemplate>
            <DataTemplate x:DataType="local:NamedColor">
                <ViewCell>
                    <StackLayout Orientation="Horizontal">
                        <BoxView Color="{Binding Color}"
                                 ... />
                        <Label Text="{Binding FriendlyName}"
                               ... />
                    </StackLayout>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
    <!-- The BoxView doesn't use compiled bindings -->
    <BoxView Color="{Binding Source={x:Reference colorListView}, Path=SelectedItem.Color}"
             ... />
</Grid>
```

Slika 4.8: Primjer korištenja *x:DataType* atributa

4.4.3. *x:DataType =''{x:Null}'' atribut*

Postoji mogućnost isključenja opcije kompajliranja, korištenjem null svojstva. Katkad ne želimo da nam jedan izvor podataka obuhvati sve unutarnje objekte pa zato nam ih je korisno isključiti po potrebi (slika ispod, ljubičasta boja).

```
<StackLayout x:DataType="local:HslColorViewModel">
    <StackLayout.BindingContext>
        <local:HslColorViewModel Color="Sienna" />
    </StackLayout.BindingContext>
    <BoxView Color="{Binding Color}"
             VerticalOptions="FillAndExpand" />
    <StackLayout x:DataType="{x:Null}"
                 Margin="10, 0">
        <Label Text="{Binding Name}" />
        <Slider Value="{Binding Hue}" />
        <Label Text="{Binding Hue, StringFormat='Hue = {0:F2}'}" />
        <Slider Value="{Binding Saturation}" />
        <Label Text="{Binding Saturation, StringFormat='Saturation = {0:F2}'}" />
        <Slider Value="{Binding Luminosity}" />
        <Label Text="{Binding Luminosity, StringFormat='Luminosity = {0:F2}'}" />
    </StackLayout>
</StackLayout>
```

Slika 4.9: Isključivanje kompajliranja

5. Azure DevOps

Rješenje za upravljanje softverskim projektima, koje uključuje sistem za editiranje koda, upravljanje zadacima i praćenje pogrešaka. Omogućava timovima da uspostave suradnju na projektima i automatizaciju procesa razvoja softvera. Korištena je prilikom razvoja pa će zato biti objašnjena u narednim poglavljima.

Sastoji se od nekoliko ključnih komponenti koje zajedno čine integrirani sistem.

Slika 5.1: Funkcionlanosti Azure DevOps-a

5.1. Azure Boards

Komponenta Azure DevOps platforme koja se koristi za upravljanje zadacima i projektnim planiranjem. Omogućava timovima da kreiraju liste zadataka, definiraju prioritete, dodjele zadatke članovima i prate napredak projekta. Nudi različite vrste tablica za upravljanje zadacima, uključujući kanban i Scrum.

U usporedbi s Confluenceom, nudi bolje mogućnosti za upravljanje zadacima i projektnim planiranjem, dok je on više usredotočen na upravljanje dokumentacijom i komunikaciju unutar tima.

Kada se uspoređuje s Jirom, nudi slične mogućnosti različite u načinu organiziranja informacija za upravljanje zadacima. Ima bolju integraciju s drugim Azure DevOps alatima, nudi fleksibilne tablice za upravljanje zadacima, a Jira backlog i sprint tablice.

Izbor između Azure Boards, Confluencea i Jire ovisi o specifičnim potrebama tima i projekta. Ako je tim fokusiran na upravljanje zadacima i projektnim planiranjem, Azure Boards je bolji izbor. U slučaju da je fokus tima na suradnji i upravljanju dokumentacijom, bolji izbor je Confluence.

Slika 5.2: DevOps vs Jira

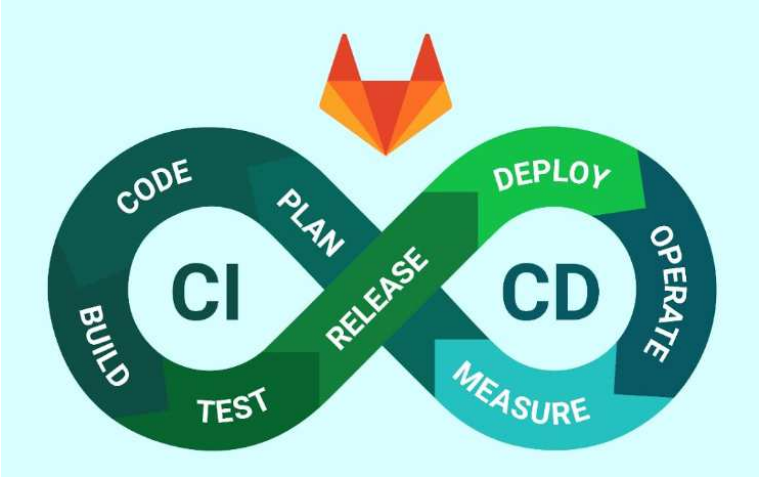
5.2. Azure Repos

Popularna platforma Azure DevOps-a za upravljanje kodom bazirane na Git-u s GitLab-om, obje nude slične mogućnosti.

Nudi osnovne mogućnosti upravljanja kodom kao što su kloniranje, kreiranje grana, spajanje i pregled. Nudi izmjene i povlačenja zahtjeva te integraciju s drugim Azure DevOps alatima.

GitLab je potpuna platforma koja nudi alate za kolaborativni rad na kodu, upravljanje verzijama, procese CI/CD (kontinuiranu integraciju i dostavu), planiranje projekata i druge značajke. Nudi ugrađene alate za upravljanje kvalitetom i pregled koda, izmjenu i povlačenje zahtjeva te integracije s drugim servisima.

Nudi više naprednih značajki poput procesa CI/CD i podrške za planiranje projekata, što je čini odličnim izborom za velike timove koji traže sveobuhvatnu platformu za upravljanje projektima. Azure Repos, ima prednost za manje timove koji

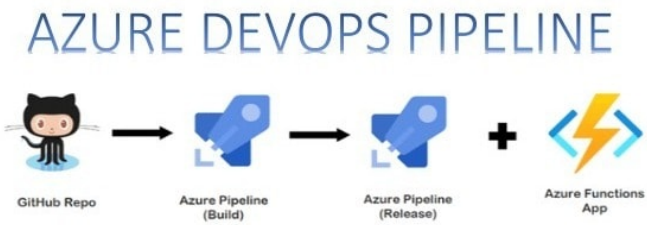


Slika 5.3: CI/CD Pipeline Setup with GitLab

traže jednostavnu i laganu platformu za upravljanje kodom, posebno ako već koriste druge Azure DevOps alate.

5.3. Azure Pipelines

Azure Pipelines i GitLab Pipelines su alati za automatizaciju procesa CI/CD (kontinuirane integracije i dostave) koji se često koriste u razvoju softvera. Koriste se za automatizaciju procesa testiranja, gradnje, dostave i ubrzanja procesa isporuke softvera i smanjenja rizika od ručnih grešaka. Koriste se za više platformi, uključujući Windows, Linux i macOS.



Slika 5.4: Azure Pipelines

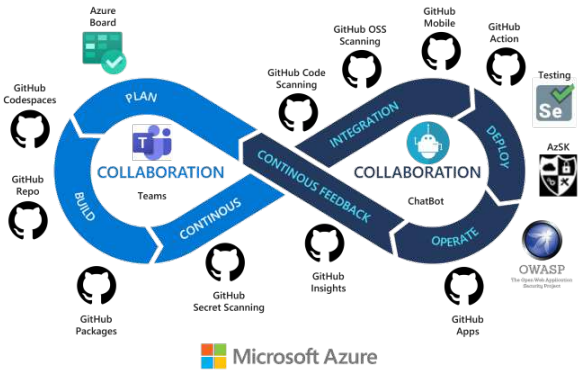
Podržava integraciju s drugim Azure DevOps alatima, poput Azure Repos i Azure Artifacts, olakšavajući upravljanje cijelim procesom razvoja softvera.

GitLab Pipelines je dio GitLab-a za upravljanje kodom. Integrira se s drugim alatima

platforme i uslugama, poput GitLab Repos i GitLab Container Registry.

Obe platforme nude alate za definiranje, izgradnju i testiranje softvera, kao i za implementaciju u proizvodnu okolinu. Koriste skripte za definiranje procesa CI/CD, koje programeri mogu koristiti za konfiguriranje i automatizaciju procesa. To im omogućava da definiraju korake gradnje, testiranja i implementacije, kao i integraciju s drugim uslugama i alatima.

Izbor između dva alata ovisi o potrebama vašeg tima i prirodi vašeg projekta. Ako se koristi Azure DevOps alate, Azure Pipelines bi mogao biti bolji izbor. Ako su u pitanju GitLab alati, bolji izbor je GitLab Pipelines. Obe platforme nude alate za automatizaciju procesa CI/CD koji će ubrzanje razvoja softvera.



Slika 5.5: Azure Pipelines workflow

5.4. Azure Test Plans

Alat je za upravljanje testiranja softvera koji se nudi kao dio Azure DevOps platforme. Daje mogućnost timovima za testiranje da organiziraju, planiraju, prate i izvještavaju o testovima, te da automatiziraju procese testiranja softvera.

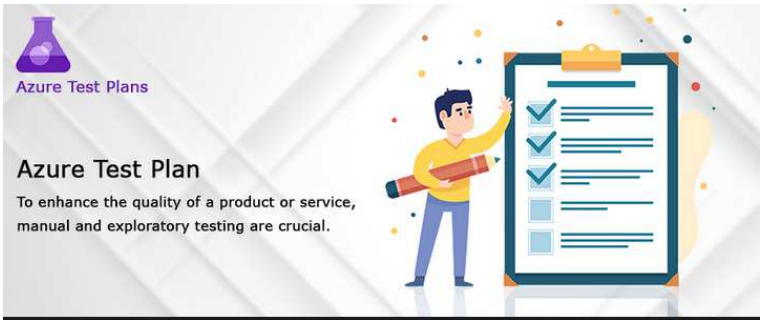
Omogućava timovima da izrađuju i izvršavaju testove na više platformi, uključujući web aplikacije, desktop aplikacije, mobilne aplikacije i servise u oblaku. Podržava više vrsta testova, uključujući jedinične, funckcionalne, za performance i sigurnost.

Daje alate za upravljanje testovima, koji olakšavaju organiziranje i planiranje

testova. Timovi za testiranje mogu ga koristiti za izradu testnih planova, testnih skupova i testnih slučajeva, te za definiranje prioriteta testiranja i rasporeda testiranja. Ovi testni planovi i skupovi testova mogu se organizirati na različite načine, ovisno o potrebama projekta.

Timovi za testiranje mogu ga koristiti da bi pratili napredak testiranja, ažurirali status testova i izvještavali o rezultatima testova. Ovi podaci se mogu prikazati u različitim oblicima, kao što su grafikoni, izvještaji i sl.

Pomaže da se osigura kvaliteta softvera, poboljša produktivnost i smanji rizik od grešaka u softveru.



Slika 5.6: Azure Test Plan preview

6. Pojašnjenja parametara očitanih iz okoline

```
1  public class EnvironmentStatus
2      {
3          public int? SerialNumber { get; set; }
4
5          public DateTime? DeviceEpoch { get; set; }
6
7          public int? ConfigurationNumber { get; set; }
8
9          public double? AbsoluteAirPressure { get; set; }
10
11         public double? AirHumidity { get; set; }
12
13         public double? AirTemperature { get; set; }
14
15         public string? IndoorAirQuality { get; set; }
16
17         public double? BatteryLevel { get; set; }
18
19         public DateTime? DateCreated { get; set; }
20
21         public DateTime? DateModified { get; set; }
22     }
23 }
```

SerialNumber: Broj serije uređaja koji prati ovu statusnu informaciju.
DeviceEpoch: Datum i vrijeme kada je uređaj zabilježio ovu informaciju.
ConfigurationNumber: Broj konfiguracije ili postavki koje se primjenjuju na uređaj.
AbsoluteAirPressure: Mjera apsolutnog tlaka zraka, obično izražena u hektopaskalima (hPa) ili sličnim jedinicama.
AirHumidity: Postotak relativne vlažnosti zraka u okolini.
AirTemperature: Temperatura zraka u okolini, obično izražena u Celzijevim stupnjevima.
IndoorAirQuality: Informacija o kvaliteti unutarnjeg zraka, koja može sadržavati podatke o prisutnosti štetnih tvari, alergenima ili drugim relevantnim faktorima.
BatteryLevel: Postotak preostale baterijske energije ili napon baterije uređaja.
DateCreated: Datum i vrijeme kada je ovaj statusni zapis prvotno stvoren.
DateModified: Datum i vrijeme kada je ovaj statusni zapis posljednji put izmijenjen ili ažuriran.

7. Opis statusa

```
<Grid Margin="14">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />

        <!-- Additional rows if necessary -->
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="Auto" />
        <!-- Additional columns if necessary -->
    </Grid.ColumnDefinitions>
```

Ovdje počinje definicija korisničkog sučelja (UI). Kreiramo **Grid** element koji će se koristiti za razmještaj drugih elemenata.

```
        <Label Text="Sort by Humidity:"
           HorizontalOptions="Start" FontSize="17"
           TextColor="WhiteSmoke" Grid.Row="0"
           Grid.Column="0" />
```

```
2
3           <!-- Switch in the first row, third column -->
4           <Switch x:Name="sortToggle" Toggled="OnSortToggled"
               ThumbColor="WhiteSmoke" Grid.Row="0"
               Grid.Column="2"
5                   HorizontalOptions="Start" Margin="0,-10,0,0"/>
6
7           <Label Text="Serial number" Grid.Row="2"
               Grid.Column="0" FontSize="17"
               TextColor="WhiteSmoke"/>
8           <Entry x:Name="serial" Grid.Row="2" Grid.Column="2"
               Placeholder="Insert module number"
               Keyboard="Numeric"
9               TextColor="White"/>
10
11          <Label Text="Locker  number" Grid.Row="3"
               Grid.Column="0" FontSize="17"
               TextColor="WhiteSmoke"/>
12          <Entry x:Name="locker" Grid.Row="3" Grid.Column="2"
               Placeholder="Insert locker number"
               Keyboard="Numeric"
13              TextColor="White"/>
14
15          <!-- Label for "Start Date/Time" in the second row,
               first column -->
16          <Label Text="Reservation Start" Grid.Row="4"
               Grid.Column="0" FontSize="17"
               TextColor="WhiteSmoke"/>
17
18          <!-- DatePicker for Start Date/Time in the second
               row, second column -->
19          <DatePicker x:Name="startDatePicker" Grid.Row="4"
               Grid.Column="2" FontSize="17"
               TextColor="WhiteSmoke"/>
20          <TimePicker x:Name="startTimePicker" Grid.Row="5"
               Grid.Column="2" FontSize="17"
               TextColor="WhiteSmoke"/>
21
```

```
22          <!-- Label for "End Date/Time" in the third row,
                first column -->
23          <Label Text="Reservation End" Grid.Row="6"
                Grid.Column="0" FontSize="17"
                TextColor="WhiteSmoke"/>
24
25          <!-- DatePicker for End Date/Time in the third row,
                second column -->
26          <DatePicker x:Name="endDatePicker" Grid.Row="6"
                Grid.Column="2" FontSize="17"
                TextColor="WhiteSmoke"/>
27          <TimePicker x:Name="endTimePicker" Grid.Row="7"
                Grid.Column="2" FontSize="17"
                TextColor="WhiteSmoke"/>
28
29          <!-- Button for applying filters in the fourth row,
                second column -->
30          <Button Text="Send Reservation"
                Clicked="OnApplyFiltersClicked" Grid.Row="8"
                Grid.Column="2" Margin="0,10,0,0"/>
31
32          <Image x:Name="qr" Grid.Row="9" Grid.Column="0"
                Grid.ColumnSpan="1" HeightRequest="220"
                Margin="0,80,0,0"/>
```

Prikazan je razmještaj elemenata u okviru **Grid** na stranici. Ovaj kod se koristi za postavljanje rasporeda elemenata na stranici, što omogućava korisnicima da unesu informacije i primjene filtere u okviru **.NET MAUI** aplikacije. **Label**, **Switch**, **DatePicker**, **TimePicker**, i **Button** elementi su postavljeni unutar određenih redova i stupaca s pomoću atributa **Grid.Row** i **Grid.Column**. Ovi elementi su organizirani tako da predstavljaju različite komponente za unos podataka kao što su tekstualne etikete, prekidači, izbornici datuma (**DatePicker**) i vremena (**TimePicker**) te gumb za primjenu filtera.

```
1  public partial class Settings : ContentPage
2  {
3      public Settings()
4      {
```

```
5           InitializeComponent();
6       }
7
8       private void OnSortToggled(object sender,
            ToggledEventArgs e)
9       {
10          bool sortValue = e.Value;
11          Temp.SortOnHumidity = sortValue;
12      }
13
14      private void OnApplyFiltersClicked(object sender,
            EventArgs e)
15      {
16          Temp.Start = startDatePicker.Date;
17          Temp.End = endDatePicker.Date;
18
19      }
20  }
```

Klasa **Settings** upravlja događajima i akcijama koji se odvijaju na stranici **Settings**. Na primjer, omogućava promjenu sortiranja prema vlažnosti i primjenu filtera na osnovu odabranih datuma.

8. Renderiranje detalja

```
1   <ContentPage.ToolbarItems>
2       <ToolbarItem Text="Additional Info" Priority="0"
           Order="Secondary" />
3       <ToolbarItem Text="------------------------"
           Priority="0" Order="Secondary" />
4       <ToolbarItem Text="{Binding
           status.SerialNumber,StringFormat='Serial Number:
           {0}'}" Priority="0" Order="Secondary"/>
5       <ToolbarItem Text="{Binding status.ConfigurationNumber,
           StringFormat='Configuration: {0}'}" Priority="0"
           Order="Secondary"/>
6       <ToolbarItem Text="{Binding status.BatteryLevel,
           StringFormat='Battery Level: {0}[V]'}" Priority="0"
           Order="Secondary"/>
7   </ContentPage.ToolbarItems>
8
9   <Grid HorizontalOptions="FillAndExpand"
       VerticalOptions="FillAndExpand" RowSpacing="0">
10      <Grid.RowDefinitions>
11          <RowDefinition Height="Auto" />
12          <RowDefinition Height="*" />
13      </Grid.RowDefinitions>
14      <Grid.ColumnDefinitions>
15          <ColumnDefinition Width="*" />
16          <ColumnDefinition Width="Auto" />
17      </Grid.ColumnDefinitions>
18      <Image x:Name="bgImg" Aspect="AspectFill"
           Grid.RowSpan="2"  Source="off.jpg"
           HorizontalOptions="FillAndExpand"
```

```
            VerticalOptions="FillAndExpand"/>
19      <StackLayout>
20          <StackLayout Orientation="Horizontal"
               HorizontalOptions="Center" Spacing="0">
21              <Label x:Name="temperatureTxt" Text="{Binding
                   status.AirTemperature}" TextColor="White"
                   FontSize="110" HorizontalOptions="Center"/>
22              <Label Text="" TextColor="White" FontSize="120"
                   HorizontalOptions="Center"/>
23          </StackLayout>
24          <Label Text="CELSIUS" Margin="40,-30,0,0"
               TextColor="White" FontSize="23"
               HorizontalOptions="Center"/>
25      </StackLayout>
26      <Grid Grid.Row="3" Margin="0,350,0,0">
27          <Grid WidthRequest="320" ColumnSpacing="10"
               RowSpacing="20"
               HorizontalOptions="CenterAndExpand"
               VerticalOptions="CenterAndExpand">
28              <Grid.ColumnDefinitions>
29                  <ColumnDefinition Width="*"/>
30                  <ColumnDefinition Width="*"/>
31              </Grid.ColumnDefinitions>
32              <Grid.RowDefinitions>
33                  <RowDefinition Height="Auto" />
34                  <RowDefinition Height="*" />
35              </Grid.RowDefinitions>
36              <StackLayout Orientation="Vertical"
                   Spacing="10">
37                  <Image Source="humidity.png"
                       HeightRequest="35"
                       HorizontalOptions="Center"/>
38                  <StackLayout Spacing="7"
                       HorizontalOptions="CenterAndExpand">
39                      <Label Text="{Binding
                           status.AirHumidity,
                           StringFormat='{0}%'}"
                           TextColor="White" FontSize="19"
```

```
                    FontAttributes="Bold"
                    HorizontalOptions="Center"/>
40                <Label Text="Humidity"
                    TextColor="White" Margin="0,-5,0,0"
                    FontSize="12"
                    HorizontalOptions="Center"/>
41            </StackLayout>
42        </StackLayout>
43        <StackLayout Grid.Row="0" Grid.Column="1"
              Orientation="Vertical">
44            <Image Source="pollution.png"
                HeightRequest="50"
                HorizontalOptions="Center"/>
45            <StackLayout Spacing="7"
                HorizontalOptions="CenterAndExpand">
46                <Label Text="{Binding
                    FormattedIndoorAirQuality}"
                    TextColor="White" Margin="0,0,0,0"
                    FontSize="16" FontAttributes="Bold"
                    HorizontalOptions="Center"/>
47                <Label Text="Air Pollution"
                    TextColor="White" Margin="0,-5,0,0"
                    FontSize="12"
                    HorizontalOptions="Center"/>
48            </StackLayout>
49        </StackLayout>
50        <StackLayout Grid.Row="1" Grid.Column="0"
              Orientation="Vertical" Spacing="10">
51            <Image Source="gauge.png"
                HeightRequest="35"
                HorizontalOptions="Center"/>
52            <StackLayout Spacing="7"
                HorizontalOptions="CenterAndExpand">
53                <Label Text="{Binding
                    status.AbsoluteAirPressure,
                    StringFormat='{0} hPa'}"
                    TextColor="White" FontSize="17"
                    FontAttributes="Bold"
```

```
                        HorizontalOptions="Center"/>
54                      <Label Text="Pressure"
                        TextColor="White" Margin="0,-5,0,0"
                        FontSize="12"
                        HorizontalOptions="Center"/>
55                  </StackLayout>
56              </StackLayout>
57              <StackLayout Grid.Row="1" Grid.Column="1"
                Orientation="Vertical" Spacing="10"
                HorizontalOptions="CenterAndExpand">
58                  <Image Source="cloudiness.png"
                    HeightRequest="30"
                    HorizontalOptions="Center"/>
59                  <StackLayout Spacing="7"
                    HorizontalOptions="CenterAndExpand">
60                      <Label x:Name="cloudinessTxt"
                        Text="24%" TextColor="White"
                        FontSize="17" FontAttributes="Bold"
                        HorizontalOptions="Center"/>
61                      <Label Text="Cloudiness"
                        TextColor="White" Margin="0,-3,0,0"
                        FontSize="12"
                        HorizontalOptions="Center"/>
62                  </StackLayout>
63              </StackLayout>
64          </Grid>
65      </Grid>
66  </Grid>
```

Kod definira složeni raspored elemenata na stranici za prikaz informacija o temperaturi, vlažnosti, kvaliteti zraka i drugim meteorološkim podacima. Također uključuje alatne trake na vrhu stranice za dodatne opcije. *ToolbarItems* definira alatne trake koje će biti prikazane na vrhu stranice. To uključuje tekstualne alatne trake kao što su **Additional Info**, **Serial Number**, **Configuration**, i **Battery Level**. Svaka alatna traka ima svoje tekstualno obilježje (Text), prioritet (Priority) i redoslijed (Order). *Image* element *bgImg)* prikazuje sliku (u ovom slučaju off.jpg) koja će popuniti cijelu rešetku. 'Grid.RowSpan="2"' označava da će zauzeti dva reda.

```
1  public partial class StatusDetails : ContentPage,
       IQueryAttributable, INotifyPropertyChanged
2  {
3      public EnvironmentStatus status { get; set; }
4
5      public StatusDetails()
6      {
7          InitializeComponent();
8          BindingContext = this;
9
10         Shell.SetTabBarIsVisible(this, false);
11
12         Random random = new Random();
13         int randomNumber = random.Next(20, 26); // Generates
               numbers from 17 to 25 (inclusive)
14
15         // Set the generated number as the text of the Label
16         cloudinessTxt.Text = randomNumber.ToString() + "%";
17     }
18
19     public void ApplyQueryAttributes(IDictionary query)
20     {
21         status = query[nameof(status)] as EnvironmentStatus;
22         FormattedIndoorAirQuality =
               AddSpacesToCamelCase(status.IndoorAirQuality);
23         OnPropertyChanged(nameof(status));
24     }
25
26     public string AddSpacesToCamelCase(string input)
27     {
28         if (string.IsNullOrWhiteSpace(input))
29         {
30             return input;
31         }
32
33         // Use regular expression to insert spaces before
               capital letters
34         string result = Regex.Replace(input, "(\\B[A-Z])", "
```

```
            $1");
35
36          // Capitalize the first letter
37          result = char.ToUpper(result[0]) +
               result.Substring(1);
38
39          return result;
40      }
41
42      private string _formattedIndoorAirQuality;
43
44      public string FormattedIndoorAirQuality
45      {
46          get => _formattedIndoorAirQuality;
47          set
48          {
49              if (_formattedIndoorAirQuality != value)
50              {
51                  _formattedIndoorAirQuality = value;
52                  OnPropertyChanged(nameof(FormattedIndoorAirQuality));
53              }
54          }
55      }
56  }
```

Ova klasa se koristi za prikaz detalja o statusu okoline ili životne sredine na stranici. Ona obrađuje i formatira podatke koji se primaju putem atributa i obavještava korisničko sučelje o promjenama u prikazanim podacima. *Public EnvironmentStatus status get; set;* je svojstvo koje predstavlja informacije o statusu okoline ili životne sredine. Koristi se za prikazivanje podataka na stranici. *Public StatusDetails()* inicijalizira korisničko sučelje. *Public void ApplyQueryAttributes(IDictionary query)* metoda se koristi za primjenu atributa koji se prenose sa zahtjeva ili kroz URL. *Public string FormattedIndoorAirQuality* je svojstvo koje čuva formatiranu vrijednost *IndoorAirQualit*. Koristi se za prikaz formatiranog teksta na stranici. Također implementira **INotifyPropertyChanged** kako bi obavijestio korisničko sučelje o promjenama vrijednosti.

9. Zaključak

Prošli smo usporedbe arhitektura i tehnologija da vidimo prednosti i mane. Krenuli smo prvo s upoznavanjem svih blagodati koje .NET-a 6 donosi kako bi shvatili poboljšanja u izgledu i organizaciji. Nisu velika, ali čine razliku. Globalne direktive, tipovi varijabli u lambdama, optimizacije performansi i ostale pogodnosti daju puno bolje mogućnosti u snalaženju i razvoju sustava. Zatim smo upoznali najnoviju arhitekturu .NET MAUI. To je framework koji uz mobilne platforme nudi i podršku za desktop verzije. Pitanje je, zašto se pojašnjavao XAML koji nije po MVVM obrascu i ne sadrži najnovije pogodnosti .NET-a 6. Isti je razlog zašto se prvo uči množenje u školama pa tek onda diferencijalne jednadžbe, kako bi imali osjećaj što je od čega nastalo i kako se koristi. Problem je što dokumentacija[1] ne prati standard razdvajanja objekata pa je teško pravovremeno reagirati na logičke greške. One postaju uočljive tek kasnije i nemoguće ih je otkloniti. Stoga je važno učiti na greškama drugih. Često developeri sva svoja loša iskustva žele podijeliti s drugima i popraviti sve nepravilnosti koje su napravili. U poglavlju 4 obrađene su samo najvažnije teme koje će sigurno kasnije napraviti razliku. Postoje biblioteke koje rješavaju te probleme, ali nisu dovoljno dobro dokumentirane pa razvoj postaje izazov i potrebno je uložiti vrijeme dok se sve ne posloži na svoje mjesto.

[1] `https://learn.microsoft.com/en-us/xamarin/xamarin-forms/xaml/`

LITERATURA

Microsoft. Xamarin.forms basic bindings, 2022. URL https://github.com/marcbruederlin/particles.js.

James Montemagno. Learn .net maui, 2022. URL https://www.youtube.com/watch?v=DuNLR_NJv8U.

Eduardo Rosas. The advanced xamarin developer masterclass, 2021. URL https://www.udemy.com/course/the-advanced-xamarin-developer-masterclass.

Wikipedia. Xamarin, 2021. URL https://en.wikipedia.org/wiki/Xamarin#Founding_Xamarin.

10. Sažetak

.NET je jako popularno Microsftov-o okruženje za razvoje fronted-a, backend-a, mobilnih te desktop verzija. Verzija 6 je uvela malena poboljšanja s c#-om 10. MAUI ga automatski podržava uz desktop verzije. Omogućava pristup nativnim funkcijama kao i njegov prethodnik Xamarin Forms, ali uz očita poboljšana brzine i čitljivosti. XAML je prvi nastao i kasnije se razvijao u sklopu cross-platform arhitektura. Verzija 6 je uvela malena poboljšanja s c#-om 10. MAUI ga automatski podržava uz desktop verzije. Omogućava pristup nativnim funkcijama kao i njegov prethodnik Xamarin Forms, ali uz očita poboljšana brzine i čitljivosti. XAML je prvi nastao i kasnije se razvijao u sklopu cross-platform arhitektura. Glavna prednost je hot reload koja jednostavnim spremanjem prikazuje promjene. Uz to, pojašnjene su još neke osnovne funkcionalnosti ali ne detaljno. Što je veća kompleksnost to on postaje nečitljiviji pa je dobra praksa držati samo osnovne funkcionalnosti a sve dodatne zahtjeve pisati u c#-u.