Inhalt

Geben Sie zum Beweis, dass Sie das Praktikum durchgeführt haben, einen Printscreen des MAUI-GUIs TodoDetails (wie auf Seite 17, mit eigenen Daten) im Dokument DT12.pdf ab. Abgabetermin ist eine Woche nach Praktikumsdurchführung.

Der Praktikumsteil mit den Themes ist fakultativ.

Infos

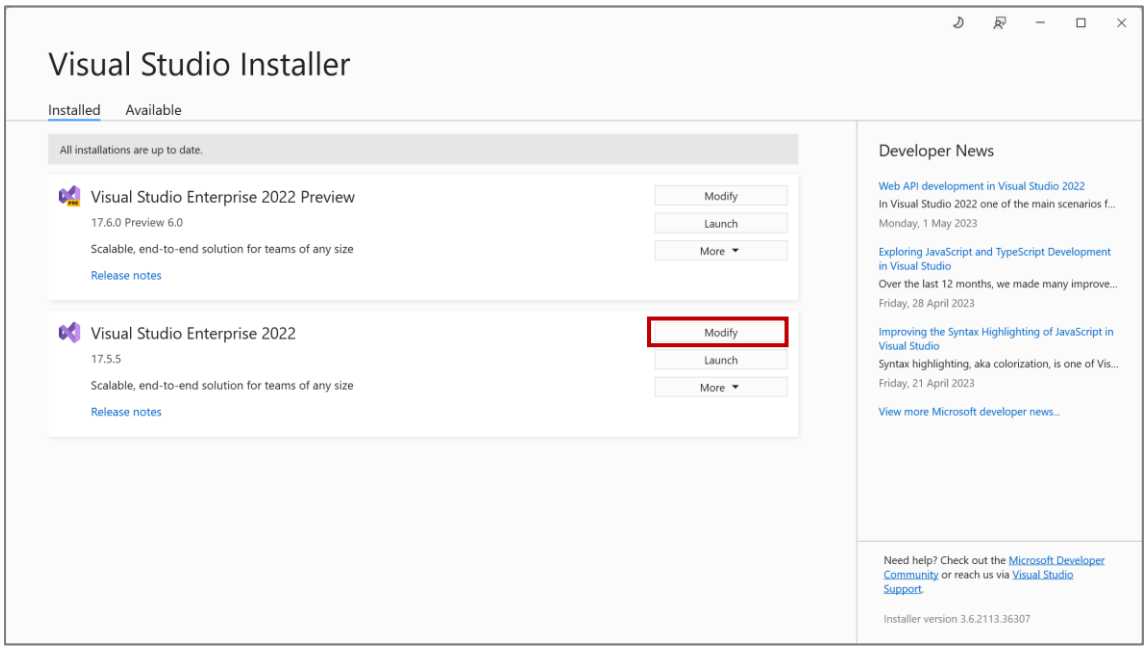
Dieses Praktikum basiert auf dem .NET MAUI Workshop Repo https://github.com/dotnet-presentations/dotnet-maui-workshop aber anstatt eine MonkeyFinder App zu bauen, werden wir eine Todo App erstellen.

Ziele

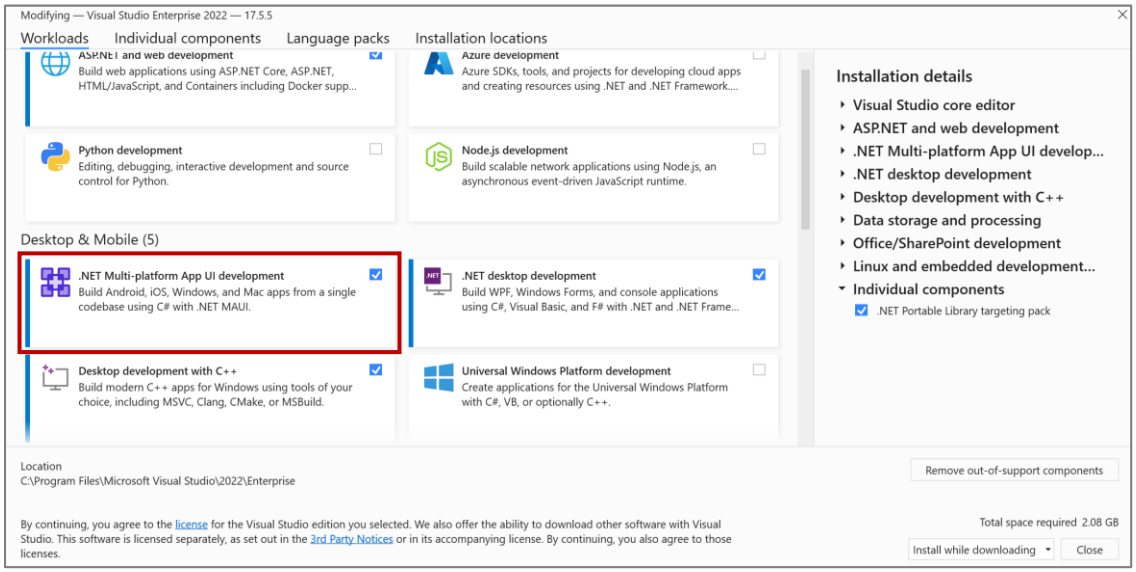
- .NET MAUI aktivieren
- DataTemplate
- Binding
- Navigation
- MVVM
- Theme

Vorbedingung

Damit für .NET MAUI programmiert werden kann, muss erst das Modul dafür in Visual Studio hinzugefügt werden. Öffnen Sie dafür den Visual Studio Installer und klicken Sie bei ihrer Visual Studio Variante auf “Modify”.

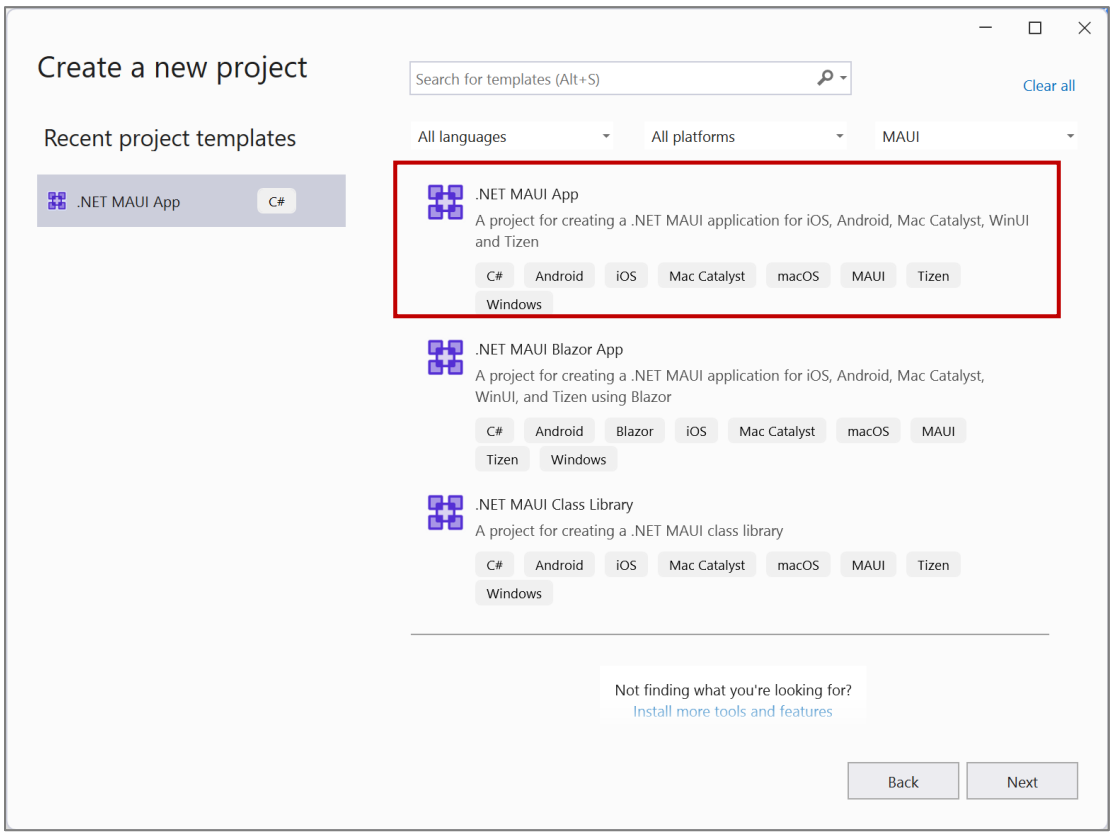


Wir benötigen das Modul .NET Multi-platform App UI development.



Neues .NET MAUI-Projekt

Erstellen Sie ein neues .NET MAUI App Projekt und wählen Sie als Framework .NET 7.0 (Standard Term Support). Nennen Sie die Anwendung nicht MauiApp sondern *MauiApp1* (sonst erhalten Sie einen Konflikt, da der generierte Namespace denselben Namen wie die Klasse erhält) und Sie können Sie den beigelegten Code 1:1 kopieren.



In der Vorlesung haben wir bereits gesehen, wie wir die Applikation starten können auf Android über den Emulator und auf Windows mit dem Entwicklermodus. Prüfen Sie, dass sie das neue Projekt ausführen können.

Daten anzeigen

Erstellen Sie einen neuen Ordner *Model* und darin eine neue Klasse *Todo.cs* mit folgendem Code:

```
public class Todo

    {
        public string Title { get; set; }
        public string Description { get; set; }
        public bool IsDone { get; set; }
        public string Category { get; set; } = "Default";
        public DateTime DueDate { get; set; }
    }
```

Zunächst möchten wir unsere *MainPage.xaml* anpassen um damit unser neues Todo Model benutzt wird. Dafür müssen wir im Tag der ContentPage den Namespace deklarieren, wo unser Model liegt. Dies geschieht mit:

```
xmlns:model="clr-namespace:<hier Name Ihrer Anwendung>.Model"
```

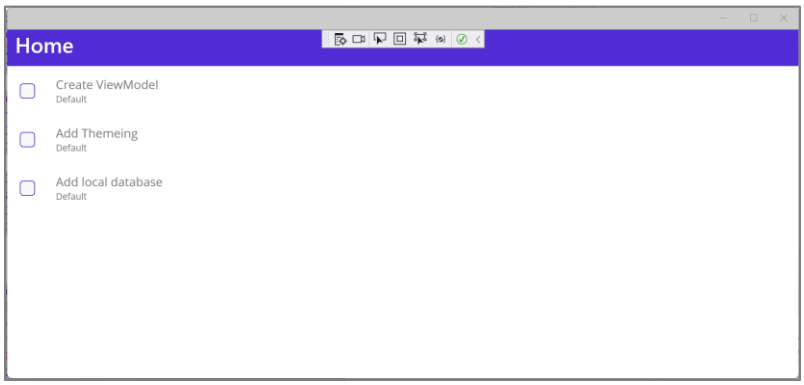
Fügen Sie zwischen den ContentPage-Tag folgendes XAML ein, um das Aussehen der MainPage anzupassen:

```
<ScrollView>
    <CollectionView>
        <CollectionView.ItemsSource>
            <x:Array Type="{x:Type model:Todo}">
                <model:Todo
                        Title="Create ViewModel"
                        Description="Create a ViewModel in the next step to learn
MVVM."
                        Category="Default"
                        IsDone="False"/>
                <model:Todo
                        Title="Add Theming"
                        Description="Integrate your own theme in the app."
                        Category="Default"
                        IsDone="False"/>
                <model:Todo
                        Title="Add local database"
                        Description="Learn how to add a local database to work with
(optional)."
                        Category="Default"
                        IsDone="False"/>
            </x:Array>
        </CollectionView.ItemsSource>
        <CollectionView.ItemTemplate>
            <DataTemplate x:DataType="model:Todo">
                <HorizontalStackLayout Padding="10"
HorizontalOptions="FillAndExpand" Spacing="10">
                    <CheckBox IsChecked="{Binding IsDone}"
VerticalOptions="Start"/>
                    <VerticalStackLayout VerticalOptions="Center">
                        <Label VerticalOptions="Center" FontSize="16"
TextColor="Gray" Text="{Binding Title}"/>
                        <Label Text="{Binding Category}" FontSize="12"
TextColor="Gray"/>
                    </VerticalStackLayout>
                </HorizontalStackLayout>
            </DataTemplate>
        </CollectionView.ItemTemplate>
    </CollectionView>
</ScrollView>
```

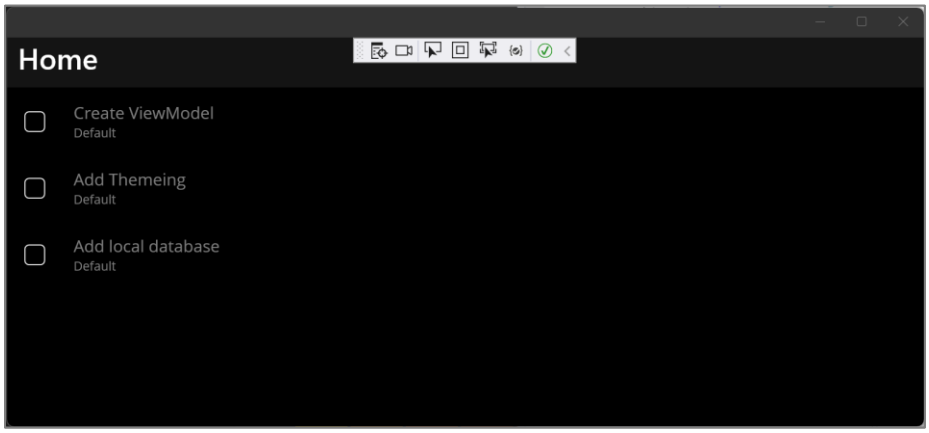
Wir erstellen vorerst ein lokales Array aus Todos (x:Array Type=”{x:Type model:Todo}”) und setzen dies direkt als ItemsSource der CollectionView ein, damit wir überhaupt Daten zum Anzeigen haben. Im *CollectionView.ItemTemplate* definieren wir, wie die einzelnen Elemente in der Liste aussehen sollen.
Das Ganze befindet sich in einer ScrollView damit wir runterscrollen können, wenn wir mehr Elemente haben als auf dem Bildschirm Platz haben.

Entfernen Sie von *MainPage.xaml.cs* die Methode `OnCounterClicked` und starten Sie die Applikation auf Windows und auf Android. Auf Windows sollte die Applikation nun folgendermassen

aussehen:



Oder falls Sie DarkMode von Windows benutzen, ändert die .NET MAUI-App automatisch zum integriertem DarkTheme:

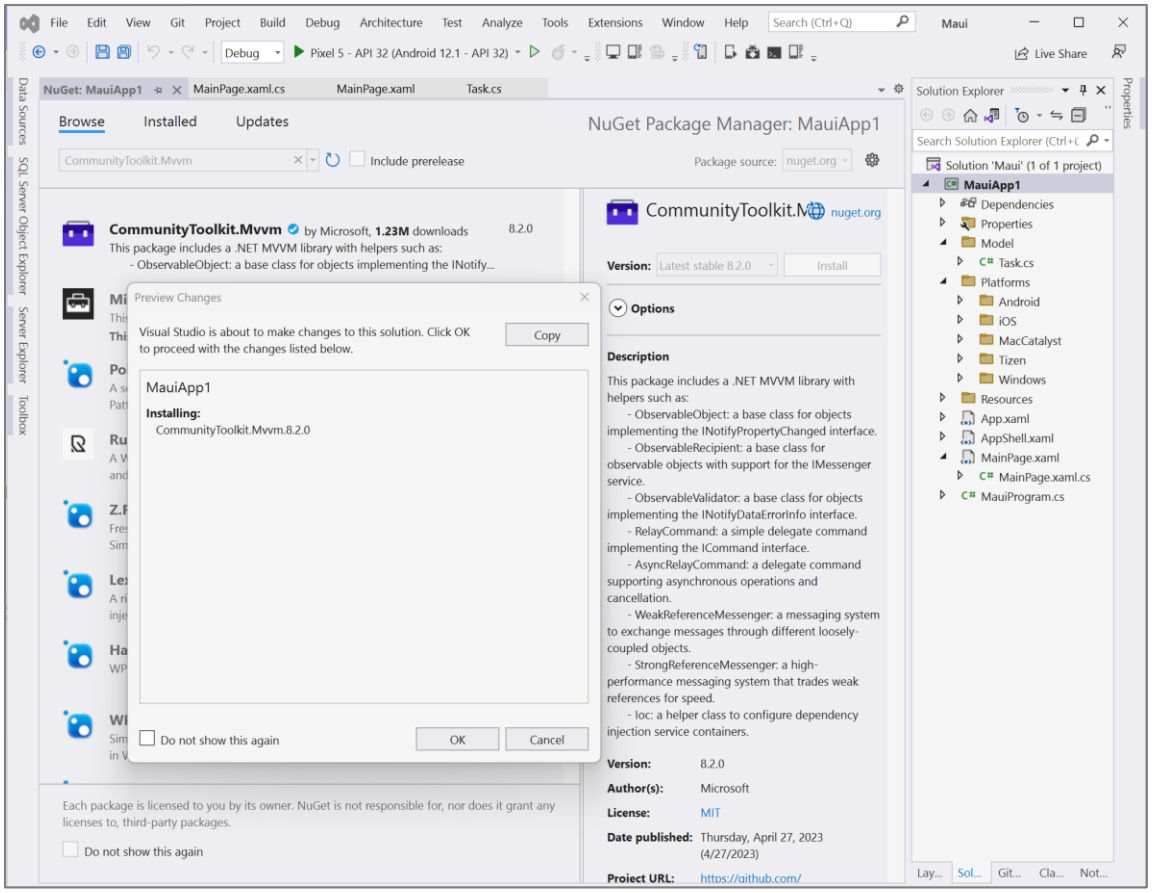


MVVM

Model-View-ViewModel ist die bevorzugte Architektur auf .NET MAUI. Wie in der Vorlesung erwähnt, nimmt uns das .NET Community Toolkit Arbeit ab, um MVVM nutzen zu können. Jede Klasse, die das CommunityToolkit verwenden will, muss als *partial* deklariert werden. Ansonsten kann der generierte Code vom CommunityToolkit nicht verwendet werden.

CommunityToolkit

Damit wir es aber überhaupt nutzen können, müssen wir erst das NuGet-Paket hinzufügen:



Um Codeduplikation zu vermeiden, erstellen wir eine neue Klasse *BaseViewModel.cs*, welche von *ObservableObject* erbt. ObservableObject wird durch das CommunityToolkit zur Verfügung gestellt. Ohne dem Toolkit müssen wir überall die Implementationen für INotifyPropertyChanged erstellen, was wir für MVVM und das DataBinding zwingend benötigen.

Erstellen Sie nun einen neuen Ordner *ViewModel* und darin die Klasse *BaseViewModel.cs* mit folgendem Code:

```
using CommunityToolkit.Mvvm.ComponentModel;

namespace MauiApp1.ViewModel
{
    public partial class BaseViewModel : ObservableObject
    {
        [ObservableProperty]
        [NotifyPropertyChangedFor(nameof(IsNotBusy))]
        bool isBusy;
```

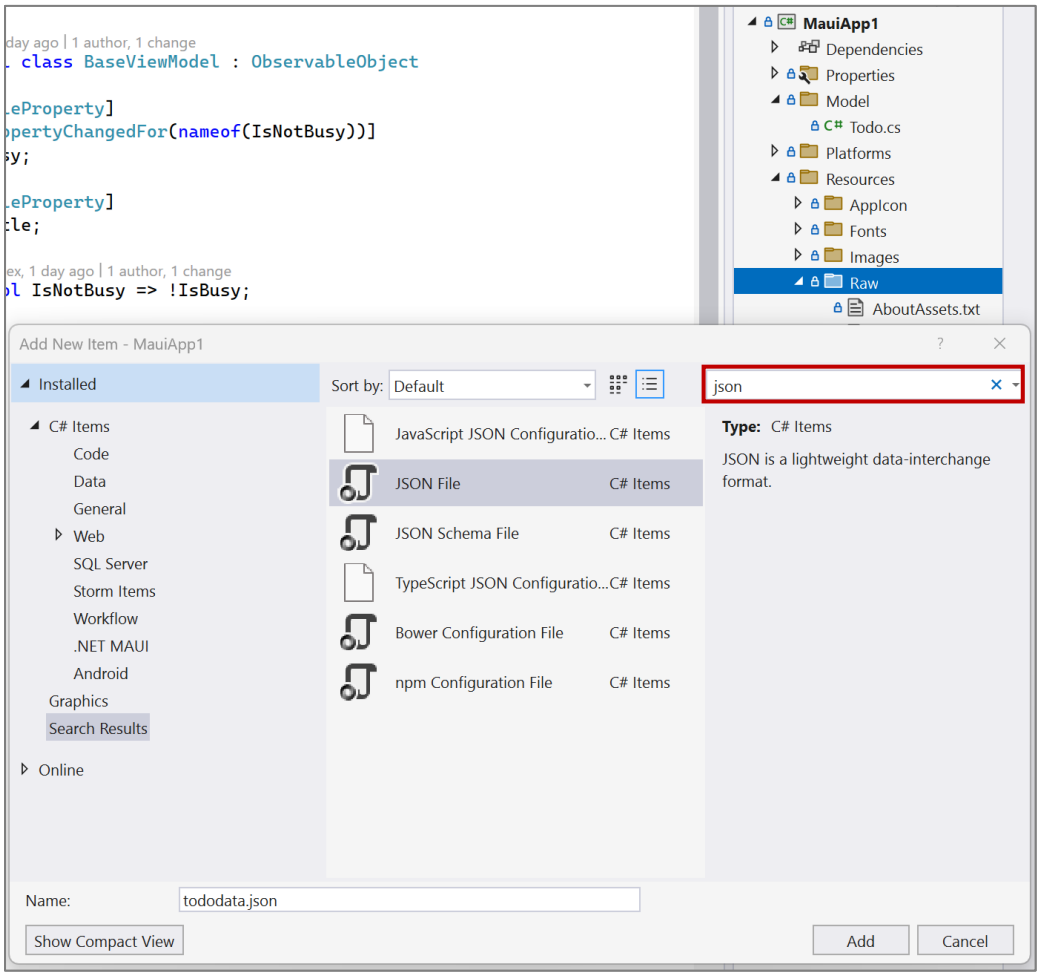
```
        [ObservableProperty]
        string title;

        public bool IsNotBusy => !IsBusy;
    }
}
```

TodoService

Damit Sie Todos laden und anzeigen können, benötigen Sie einen TodoService. In diesem Kapitel laden wir die Todos von einem Json File. Bereiten Sie zuerst das Json-File vor.

Gehen Sie in den Ordner *Resources -> Raw* und erstellen Sie dort ein Json-File mit dem Namen *tododata.json*. Das neue File muss sich zwingend im Order Raw unter Resources befinden damit es während der BuildAction als MauiAsset mitgeliefert wird und wir vereinfachten Zugriff darauf haben.



Kopieren Sie einfachheitshalber folgende Daten in das Json-File:

```
[
  {
    "Title": "Create ViewModel",
    "Description": "Create a ViewModel in the next step to learn MVVM.",
    "IsDone": true,
    "Category": "Default",
    "DueDate": "2023-05-31T00:00:00"
  },
  {
    "Title": "Add Themeing",
```

```
    "Description": "Integrate your own theme in the app.",
    "IsDone": false,
    "Category": "Default",
    "DueDate": "2023-05-31T00:00:00"
  },
  {
    "Title": "Add local database",
    "Description": "Learn how to add a local database.",
    "IsDone": false,
    "Category": "Default",
    "DueDate": "2023-05-31T00:00:00"
  }
]
```

Nun können wir den TodoService erstellen. Erstellen Sie im Projekt einen Ordner *Services* und darin die Klasse *TodoService.cs*. Kopieren Sie folgenden Text:

```
using MauiApp1.Model;
using System.Text.Json;

namespace MauiApp1.Services
{
    public class TodoService
    {
        List<Todo> todoList = new List<Todo>();
        public async Task<List<Todo>> GetTodos()
        {
            if (todoList?.Count > 0)
            {
                return todoList;
            }

            using var stream = await
FileSystem.OpenAppPackageFileAsync("tododata.json");
            using var reader = new StreamReader(stream);
            var contents = await reader.ReadToEndAsync();
            todoList = JsonSerializer.Deserialize<List<Todo>>(contents);

            return todoList;
        }
    }
}
```

Da sich das tododata.json-File im Raw Ordner befindet, können wir über *FileSystem.OpenAppPackageFileAsync* auf jeder Platform darauf zugreifen.

TodoViewModel

Im nächsten Schritt erstellen Sie im Ordner *ViewModel* eine neue Klasse *TodosViewModel.cs*. Dieses ViewModel werden wir verwenden, um die Daten aufzurufen und schlussendlich per Binding der MainPage zu übergeben. Dafür sind einige Schritte nötig.

Fügen Sie am Anfang der Klasse folgende Usings ein:

```
using MauiApp1.Services;
using CommunityToolkit.Mvvm.Input;
using MauiApp1.Model;
using System.Collections.ObjectModel;
```

Diese werden benötigt für den Zugriff auf unseren Service und für das vereinfachte Databinding.

Ersetzen Sie nun die Klassendefinition mit:

```
public partial class TodosViewModel : BaseViewModel
{
```

```
    public ObservableCollection<Todo> Todos { get;  } = new
ObservableCollection<Todo>();
    TodoService todoService;

    public TodosViewModel(TodoService todoService)
    {
        Title = "Todos";
        this.todoService = todoService;
    }
}
```

Wir injecten den TodoService über den Konstruktor zum TodosViewModel. Damit dies funktionieren kann müssen wir die Klasse *MauiProgram.cs* anpassen. Wechseln Sie zur Klasse *MauiProgram.cs* und fügen Sie das using für den TodoService und unser ViewModel hinzu mit:

```
using MauiApp1.Services;
using MauiApp1.ViewModel;
```

Und nach dem Block wo *builder.Logging.AddDebug()* verwendet wird, fügen Sie folgenden Code ein:

```
builder.Services.AddSingleton<TodoService>();
builder.Services.AddSingleton<TodosViewModel>();
builder.Services.AddSingleton<MainPage>();
```

Mit *AddSingleton<>* wird sichergestellt, dass die jeweilige Instanz nur einmal erstellt wird. Für den Fall, dass wir für jeden Aufruf eine neue Instanz haben wollen, müssten wir *AddTransient<>* verwenden, welches wir später in diesem Praktikum auch noch antreffen werden.

Bis jetzt verwendet aber unser ViewModel den Service noch gar nicht. Gehen Sie zurück zur Klasse *TodosViewModel.cs* und fügen Sie folgende Methode ein:

```
[RelayCommand]
async Task GetTodosAsync()
{
    if (IsBusy)
    {
        return;
    }

    try
    {
        IsBusy = true;
        var todos = await todoService.GetTodos();

        if (Todos.Count != 0)
        {
            Todos.Clear();
        }

        foreach (var todo in todos)
        {
            Todos.Add(todo);
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine($"Unable to get todos: {ex.Message}");
        await Shell.Current.DisplayAlert("Error!", ex.Message, "OK");
    }
    finally
    {
        IsBusy = false;
    }
```

```
}
```

In der neuen Methode verwenden wir das Property *IsBusy*, welches wir in der *BaseViewModel* Klasse definiert haben. Darüber können wir in der MainPage dem User mitteilen, dass die Klasse beschäftigt ist mit dem Laden der Daten.

Das Attribut *[RelayCommand]* kommt vom CommunityToolkit und ermöglicht es uns sehr einfach die Methode fürs DataBinding zur Verfügung zu stellen.

Damit ist die Arbeit im Hintergrund fast abgeschlossen und bald wir können das UI für die Todos erstellen.
Da die MainPage unsere erste Page ist, die aufgerufen wird, ändern wir dort das UI. Doch zuerst müssen wir den *BindingContext* von der MainPage auf unser TodosViewModel setzen. Aus diesem Grund haben wir auch im *MauiProgram.cs* die MainPage als Singleton zu den Services hinzugefügt. So kann unser TodosViewModel injected werden.

Öffnen Sie das File *MainPage.xaml.cs* ergänzen Sie den Code mit «`using MauiApp1.ViewModel;`» und ändern Sie den Konstruktor auf folgendes:

```
public MainPage(TodosViewModel viewModel)
{
    InitializeComponent();
    BindingContext = viewModel;
}
```

UI MVVM

Nun ist die Arbeit im Hintergrund abgeschlossen und wir können uns um das UI kümmern. Sie können das Aussehen vom UI beliebig verändern, beachten Sie jedoch das am Ende des Praktikums Themes angegangen wird. Dort wird erklärt, wie man das Aussehen der GUI-Elemente über die ganze App anpassen kann.

Zurück zum UI. Gehen Sie zur Klasse *MainPage.xaml*. Praktisch werden wir den ganzen bisherigen Code ersetzen. Sie können den gesamten XAML-code entfernen.

Wir ersetzen den bisherigen ContentPage-Tag mit folgendem Code:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:model="clr-namespace:MauiApp1.Model"
             xmlns:viewmodel="clr-namespace:MauiApp1.ViewModel"
             x:DataType="viewmodel:TodosViewModel"
             x:Class="MauiApp1.MainPage"
             Title="{Binding Title}">
</ContentPage>
```

Neu ist die Definition des Namespace *viewmodel*. Dies benötigen wir, um den *DataType* auf unser TodosViewModel zu setzen. Weiter nutzen wir hier für den Title bereits ein Binding auf den Title von TodosViewModel.

Wir möchten nun auf unsere MainPage verschiedene GUI-Elemente platzieren. Aus diesem Grund werden wir ein Grid verwenden, was uns hilft eine genauere Struktur aufzubauen.

Fügen Sie zwischen den ContentPage-Tag die Grid-Definition hinzu:

```
<Grid
    ColumnSpacing="5"
    RowSpacing="0">
    <Grid.ColumnDefinitions>
```

```
        <ColumnDefinition  Width="*"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
</Grid>
```

Unser Grid wird zwei Spalten und zwei Zeilen enthalten. Der Wert *Auto* in der *RowDefinition* für Height bedeutet, dass sich die Höhe der zweiten Zeile in unserem Grid sich dem Inhalt anpassen soll. * bedeutet das die Höhe/Breite sich proportional zum verfügbaren Platz anpasst.

Als Nächstes fügen wir wieder eine CollectionView ein. Fügen Sie innerhalb unseres Grids folgenden Code ein:

```
<CollectionView ItemsSource="{Binding Todos}"
    SelectionMode="None"
    Grid.ColumnSpan="2">
</CollectionView>
```

Nun müssen wir das DataTemplate wieder hinzufügen, um zu definieren wie unsere Todos auf dieser Übersichtsseite aussehen sollten. Fügen Sie folgenden Block innerhalb der CollectionView ein:

```
<CollectionView.ItemTemplate>
    <DataTemplate x:DataType="model:Todo">
        <Grid Padding="10">
            <Border HeightRequest="125" >
                <Grid Padding="0">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*"/>
                        <ColumnDefinition Width="*"/>
                        <ColumnDefinition Width="*"/>
                    </Grid.ColumnDefinitions>
                    <CheckBox IsChecked="{Binding IsDone}"/>
                    <VerticalStackLayout
                        Grid.Column="1"
                        VerticalOptions="Center"
                        Padding="10">
                        <Label FontSize="16" Text="{Binding Title}"/>
                        <Label FontSize="12" Text="{Binding Category}"/>
                    </VerticalStackLayout>
                    <Label Text="{Binding DueDate}" Grid.Column="2"
VerticalOptions="Center"/>
                </Grid>
            </Border>
        </Grid>
    </DataTemplate>
</CollectionView.ItemTemplate>
```

Damit haben wir das Aussehen für unsere Todos definiert. Jetzt fehlt uns aber noch die Möglichkeit das Laden der Daten zu forcieren. Daher fügen wir unterhalb der CollectionView einen Button hinzu auf den wir dann das Command von GetTodosAsync ausführen können.

Fügen Sie unterhalb des CollectionView-Tags den Button ein:

```
<Button Text="Get Todos"
        Command="{Binding GetTodosCommand}"
        IsEnabled="{Binding IsNotBusy}"
        Grid.Row="1"
        Grid.Column="0"
        Margin="8"/>
```

Dieser Button wird nur zur Verfügung stehen, wenn unser ViewModel nicht am Daten laden ist. Das Binding auf die Methode geschieht über GetTodosCommand. Wir selbst haben GetTodosCommand nie definiert, aber da wir das Attribut *RelayCommand* für GetTodosAsync angefügt haben, generiert das CommunityToolkit für uns dieses Command.

Weiter möchten wir gerne den User informieren, sobald unser ViewModel im Hintergrund irgendeine Aktivität am Ausführen ist. Fügen Sie nach dem Button folgenden ActivityIndicator ein:

```
<ActivityIndicator IsVisible="{Binding IsBusy}"
   IsRunning="{Binding IsBusy}"
   HorizontalOptions="Fill"
   VerticalOptions="Center"
   Color="{DynamicResource Primary}"
   Grid.RowSpan="2"
   Grid.ColumnSpan="2"/>
```

Nach dem ActivityIndicator sollten nur noch die Tags für das erste Grid und die ContentPage geschlossen werden.

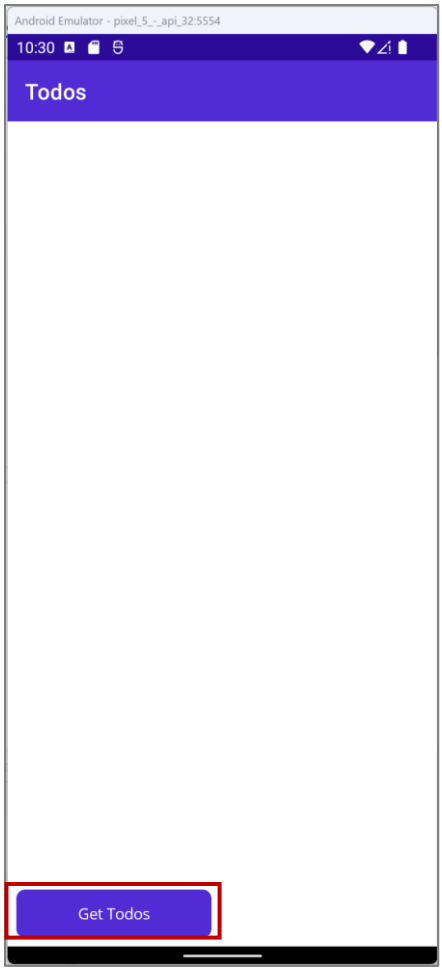
Hier noch der gesamte Code der MainPage.xaml auf einmal:

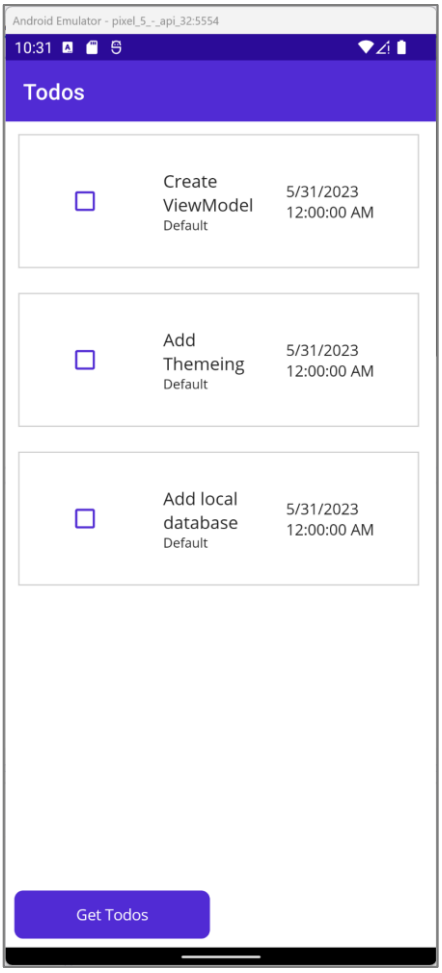
```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:model="clr-namespace:MauiApp1.Model"
             xmlns:viewmodel="clr-namespace:MauiApp1.ViewModel"
             x:DataType="viewmodel:TodosViewModel"
             x:Class="MauiApp1.MainPage"
             Title="{Binding Title}">
    <Grid
        ColumnSpacing="5"
        RowSpacing="0">
        <Grid.ColumnDefinitions>
            <ColumnDefinition  Width="*"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <CollectionView ItemsSource="{Binding Todos}"
                        SelectionMode="None"
                        Grid.ColumnSpan="2">
            <CollectionView.ItemTemplate>
                <DataTemplate x:DataType="model:Todo">
                    <Grid Padding="10">
                        <Border HeightRequest="125" >
                            <Grid Padding="0">
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition Width="*"/>
                                    <ColumnDefinition Width="*"/>
                                    <ColumnDefinition Width="*"/>
                                </Grid.ColumnDefinitions>
                                <CheckBox IsChecked="{Binding IsDone}"/>
                                <VerticalStackLayout
                                    Grid.Column="1"
                                    VerticalOptions="Center"
                                    Padding="10">
                                    <Label FontSize="16" Text="{Binding Title}"/>
                                    <Label FontSize="12" Text="{Binding
Category}"/>
                                </VerticalStackLayout>
                                <Label Text="{Binding DueDate}" Grid.Column="2"
VerticalOptions="Center"/>
                            </Grid>
                        </Border>
```

```
                    </Grid>
                </DataTemplate>
            </CollectionView.ItemTemplate>
        </CollectionView>
        <Button Text="Get Todos"
                Command="{Binding GetTodosCommand}"
                IsEnabled="{Binding IsNotBusy}"
                Grid.Row="1"
                Grid.Column="0"
                Margin="8"/>

        <ActivityIndicator IsVisible="{Binding IsBusy}"
                           IsRunning="{Binding IsBusy}"
                           HorizontalOptions="Fill"
                           VerticalOptions="Center"
                Color="{StaticResource Primary}"
                           Grid.RowSpan="2"
                           Grid.ColumnSpan="2"/>
    </Grid>
</ContentPage>
```

Starten Sie jetzt die App neu und schauen Sie sich das Ergebnis an.





Navigation

Damit wir die Details unserer Todos anschauen können, müssen wir neue Klassen erstellen und die Navigation definieren.

Fügen wir erst in der Klasse *TodosViewModel.cs* vom vorherigen Kapitel die Möglichkeit hinzu auf eine DetailsPage zu wechseln. Fügen Sie folgende Methode hinzu:

```
[RelayCommand]
async Task GoToDetails(Todo todo)
{
    if (todo == null)
    {
        return;
    }

    await Shell.Current.GoToAsync(nameof(TodoDetailsPage), true, new
Dictionary<string, object>
    {
        { "Todo", todo }
    });
}
```

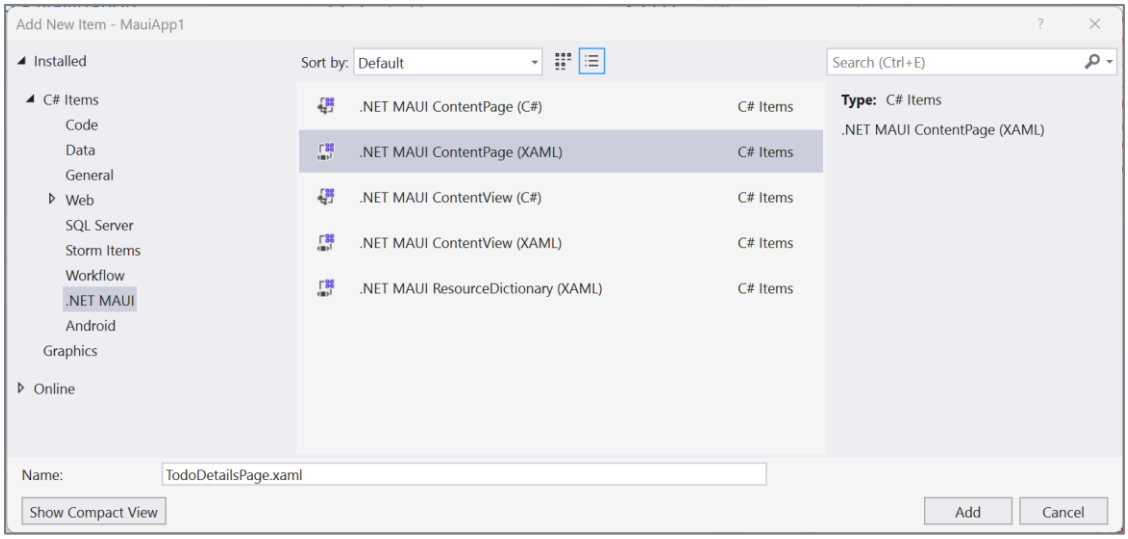
Es gibt verschiedene Möglichkeiten auf Userinput zu reagieren. Wir verwenden für diesen Fall die Tapgesture, um unsere Methode aufrufen zu können. Wir fügen auf der *MainPage.xaml* innerhalb des Border-Tags einen TapGestureRecognizer hinzu.

```
<Border.GestureRecognizers>
       <TapGestureRecognizer
    Command="{Binding Source={RelativeSource AncestorType={x:Type
    viewmodel:TodosViewModel}}, Path=GoToDetailsCommand}"
    CommandParameter="{Binding .}"/>
</Border.GestureRecognizers>
```

Mit RelativeSource geben wir an, dass wir ein anderes Binding nutzen wollen. Der AncestorType spezifiziert, dass wir gerne das Binding von TodosViewModel nutzen möchten.

TodoDetails

Es wird Zeit die neuen Klassen für die TodoDetails zu erstellen. Erstellen Sie im Ordner *ViewModel* die Klasse *TodoDetailsViewModel.cs*. Erstellen Sie zusätzlich einen neuen Ordner *View* und darin die Klasse *TodoDetailsPage.xaml* als .NET MAUI ContentPage (XAML).



Wir erweitern das neue TodoDetailsViewModel mit folgendem Code, damit das ViewModel ein Todo-Objekt über die Navigation erhalten kann:

```
using CommunityToolkit.Mvvm.ComponentModel;
using MauiApp1.Model;

namespace MauiApp1.ViewModel
{
    [QueryProperty(nameof(Todo), "Todo")]
    public partial class TodoDetailsViewModel : BaseViewModel
    {
        [ObservableProperty]
        Todo todo;

        public TodoDetailsViewModel(){}
    }
}
```

Damit die Navigation von der MainPage zur TodoDetailsPage funktionieren kann, müssen wir die Route erst zur Shell hinzufügen. Fügen Sie in *AppShell.xaml.cs* in den Konstruktor nach *InitializeComponent();* folgende Zeile ein:

```
Routing.RegisterRoute(nameof(TodoDetailsPage), typeof(TodoDetailsPage));
```

Im File *MauiProgram.cs* müssen wir das neue ViewModel und die neue Page als Transient hinzufügen, damit jedesmal die DetailsPage bei jedem Aufruf neu erstellt wird.

```
builder.Services.AddTransient<TodoDetailsViewModel>();
builder.Services.AddTransient<TodoDetailsPage>();
```

Zuletzt müssen wir das BindingContext von TodoDetailsPage auf TodoDetailsViewModel setzen. Dafür ersetzen wir den Konstruktor von *TodoDetailsPage.xaml.cs* mit folgendem Code («`using MauiApp1.ViewModel;`» nicht vergessen):

```
public TodoDetailsPage(TodoDetailsViewModel viewModel)
{
    InitializeComponent();
    BindingContext = viewModel;
}
```

Ab diesem Zeitpunkt können wir die App wieder kompilieren und starten. Wenn Sie die App nun öffnen und auf ein Todo klicken, wird unsere TodoDetailsPage angezeigt. Da wir das UI aber noch nicht angepasst haben, ist nichts von unserem Todo ersichtlich.

Ändern Sie die *TodoDetailsPage.xaml* um zu:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.View.TodoDetailsPage"
             xmlns:viewmodel="clr-namespace:MauiApp1.ViewModel"
             x:DataType="viewmodel:TodoDetailsViewModel"
             Title="{Binding Todo.Title}">
    <ScrollView>
        <Grid Padding="10">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
```

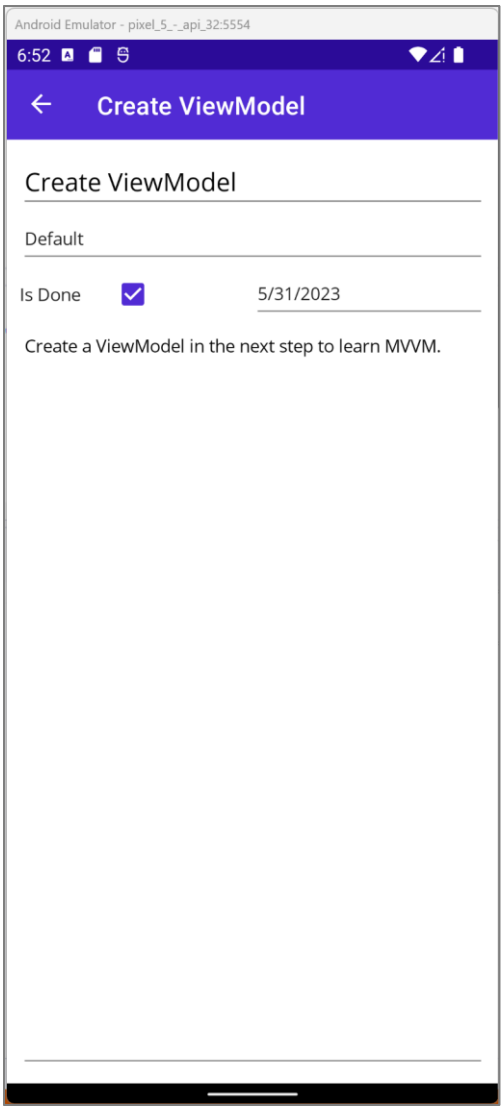
```
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Entry Text="{Binding Todo.Title}" FontSize="20" Grid.Row="0"
Grid.ColumnSpan="2"/>
            <Picker Grid.Row="1" Grid.ColumnSpan="2" >
                <Picker.ItemsSource>
                    <x:Array Type="{x:Type x:String}">
                        <x:String>Default</x:String>
                        <x:String>XAML</x:String>
                        <x:String>MVVM</x:String>
                        <x:String>Database</x:String>
                    </x:Array>
                </Picker.ItemsSource>
                <Picker.SelectedIndex>0</Picker.SelectedIndex>

            </Picker>
            <HorizontalStackLayout Grid.Row="2" Grid.Column="0"
VerticalOptions="Center" Spacing="20">
                <Label Text="Is Done" VerticalTextAlignment="Center"/>
                <CheckBox IsChecked="{Binding Todo.IsDone}" />
            </HorizontalStackLayout>

            <DatePicker Date="{Binding Todo.DueDate}" Grid.Row="2"
Grid.Column="1"/>
            <Editor Text="{Binding Todo.Description}" Grid.Row="3"
Grid.ColumnSpan="2"/>

        </Grid>
    </ScrollView>
</ContentPage>
```

Sie können nun die App wieder starten. Klicken Sie auf der Übersicht auf ein Todo und die TodoDetailsPage sollte erscheinen. Auf Android sieht die TodoDetailsPage etwa so aus:


Android Emulator - pixel_5_-_api_32:5554
6:52
Create ViewModel
Create ViewModel
Default
Is Done
5/31/2023
Create a ViewModel in the next step to learn MVVM.

Themes

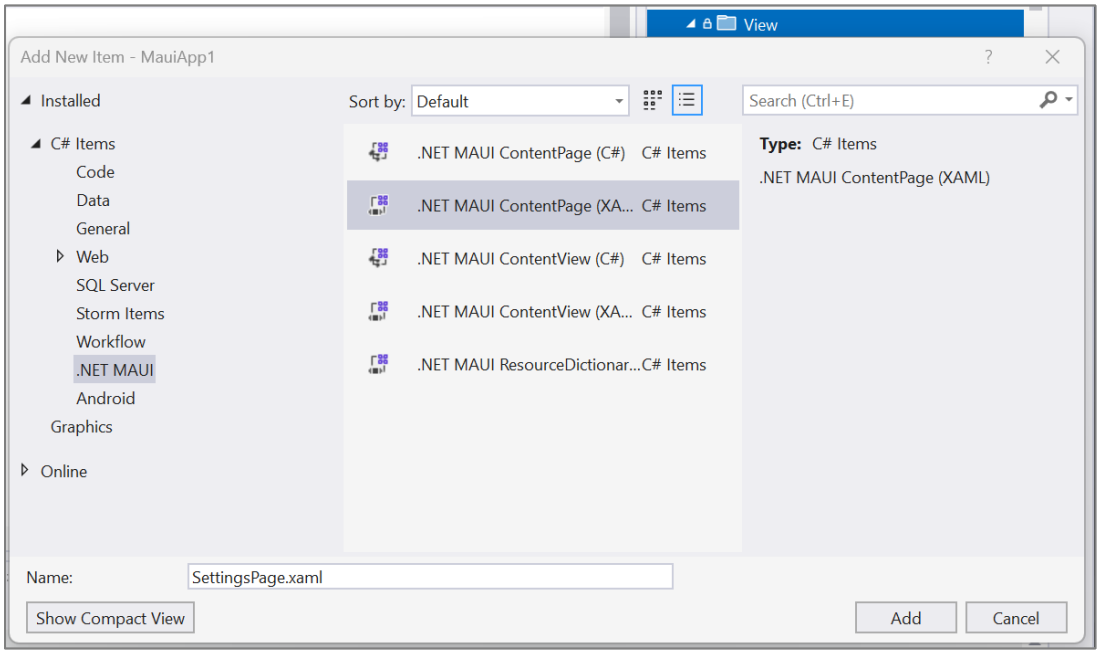
Dieses Kapitel über Themes basiert in Teilen auf diesem Video https://www.youtube.com/watch?v=0cY8iCz50fI, dem dazugehörigen Github Repo von Daniel Hindrikes und der Dokumentation von Microsoft.

In diesem Kapitel erstellen wir die Möglichkeit dynamisch das Theme der App zu ändern. Wir werden eine neue Page und ein neues ViewModel für Settings erstellen. Weiter fügen wir die Settingspage zur Navigation in der AppShell hinzu, aber anders als die TodoDetailsPage wird die Settingspage in der Navigation selbst angezeigt. Wir werden auch neue ResourceDictionaries erstellen, um das Aussehen unserer Apps für die verschiedenen Themes zu definieren.

Erstellen Sie zuerst unter *ViewModel* die Klasse *SettingsViewModel.cs*. Wir schreiben den Code dafür später. Die Klasse muss auf public gesetzt werden.

SettingsPage

Erstellen Sie nun im Order *View* eine neue .NET MAUI ContentPage *SettingsPage.xaml*:



Fügen Sie in *SettingsPage.xaml* den Namespace von ViewModel zur Contentpage hinzu und setzen Sie den *DataType* auf *SettingsViewModel*.

```
xmlns:viewmodel="clr-namespace:MauiApp1.ViewModel"
x:DataType="viewmodel:SettingsViewModel"
```

Fügen Sie als Inhalt der ContentPage einen *Border* und darin ein *VerticalStackLayout* mit *Padding ="5"* ein*:* `<VerticalStackLayout Padding="5">`

Um das Theme auszuwählen nutzen wir das Element *Picker*. Jedoch besitzt Picker nicht die Eigenschaft Command. Daher müssen wir den Event *SelectedIndexChanged* als Umweg zu unserer Methode vom ViewModel nutzen. Fügen Sie innerhalb des VerticalStackLayouts folgenden Picker hinzu:

```
<Picker x:Name="ThemePicker"
    Title="Pick a Theme"
```

```
    SelectedIndexChanged="ThemePicker_SelectedIndexChanged"
    ItemsSource="{Binding Themes}"/>
```

Der Inhalt von SettingsPage.xaml sollte so aussehen:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.View.SettingsPage"
             xmlns:viewmodel="clr-namespace:MauiApp1.ViewModel"
             x:DataType="viewmodel:SettingsViewModel"
             Title="Settings">
    <Border>
        <VerticalStackLayout Padding="5">
            <Picker x:Name="ThemePicker"
                            Title="Pick a Theme"
                            SelectedIndexChanged="ThemePicker_SelectedIndexChanged"
                            ItemsSource="{Binding Themes}"/>
        </VerticalStackLayout>
    </Border>
</ContentPage>
```

Übergeben Sie dem Konstruktor in *SettingsPage.xaml.cs* das SettingsViewModel und setzen Sie es als *BindingContext*.

```
public SettingsPage(SettingsViewModel viewModel)
{
    InitializeComponent();
    BindingContext = viewModel;
}
```

Als nächsten benötigen wir in der Klasse SettingsPage die *ThemePicker_SelectedIndexChanged* Methode für unseren Event.

```
private void ThemePicker_SelectedIndexChanged(object sender, EventArgs e)
{
    var picker = (Picker)sender;
    var selectedItem = (string)picker.SelectedItem;

    if (selectedItem != null)
    {
        (BindingContext as ViewModel.SettingsViewModel).LoadNewTheme(selectedItem);
    }
}
```

SettingsViewModel

Wechseln wir nun zum *SettingsViewModel.cs.* Damit wir verschiedene Themes laden können, fügen wir folgende Liste und Dictionary zu unsere Klasse SettingsViewModel hinzu:

```
Dictionary<string, ResourceDictionary> defaultThemes = new Dictionary<string,
ResourceDictionary>()
{
    { "Light", new MauiApp1.Resources.Themes.Light() },
    { "Dark", new MauiApp1.Resources.Themes.Dark() },
    { "Earth", new MauiApp1.Resources.Themes.Earth() },
    { "Pink", new MauiApp1.Resources.Themes.Pink() }
};

public List<string> Themes { get; set; }
```

Diese vordefinierten Themes existieren noch nicht. Wir erstellen diese, nachdem der Code für das SettingsViewModel fertig ist.
Die Liste *Themes* vereinfacht für uns das Zuweisen der Werte für den Picker auf der SettingsPage.

Den Konstruktor setzen wir auf folgendes:

```
public SettingsViewModel()
{
    Themes = defaultThemes.Select(x => x.Key).ToList();
}
```

Nun geht es darum das Theme dynamisch zu setzen. Fügen Sie folgende Methode ein:

```
public void LoadNewTheme(string themeName)
{
    if (!MainThread.IsMainThread)
    {
        MainThread.BeginInvokeOnMainThread(() => LoadNewTheme(themeName));
        return;
    }

    ResourceDictionary dictionary = defaultThemes[themeName];

    if (dictionary != null)
    {
        Application.Current.Resources.MergedDictionaries.Clear();
        Application.Current.Resources.MergedDictionaries.Add(new
MauiApp1.Resources.Styles.ThemeStyles());
        Application.Current.Resources.MergedDictionaries.Add(new
MauiApp1.Resources.Themes.BasicTheme());
        Application.Current.Resources.MergedDictionaries.Add(dictionary);
    }
}
```

Themes und Styles müssen jeweils in der Applikation zu den MergedDictionaries hinzugefügt werden. Da wir unsere eigenen Themes laden und dynamisch ändern wollen, entfernen wir jegliche Einträge von der MergedDictionaries der Applikation.
Da wir aber immer alle entfernen, müssen wir unseres *BasicTheme* und *ThemeStyles* wieder hinzufügen. Schlussendlich laden wir unser gewähltes Theme in die MergedDictionaries.

Fügen Sie nun in der Klasse *MauiProgram.cs* gleich wie im vorherigen Kapitel die neu generierten *SettingsPage* und *SettingsViewModel* als Singleton hinzu.

```
builder.Services.AddSingleton<SettingsViewModel>();
builder.Services.AddSingleton<SettingsPage>();
```

AppShell Navigation

Wir möchten nun unsere neue SettingsPage zur Navigation hinzufügen. Ersetzen Sie in der *AppShell.xaml* den Inhalt mit:

```
<Shell
    x:Class="MauiApp1.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:MauiApp1"
    xmlns:view="clr-namespace:MauiApp1.View"
    Shell.FlyoutBehavior="Flyout">
    <ShellContent
        Title="Home"
        ContentTemplate="{DataTemplate local:MainPage}"
        Route="MainPage" />
    <ShellContent
        Title="Settings"
        ContentTemplate="{DataTemplate view:SettingsPage}"
        Route="SettingsPage" />
```

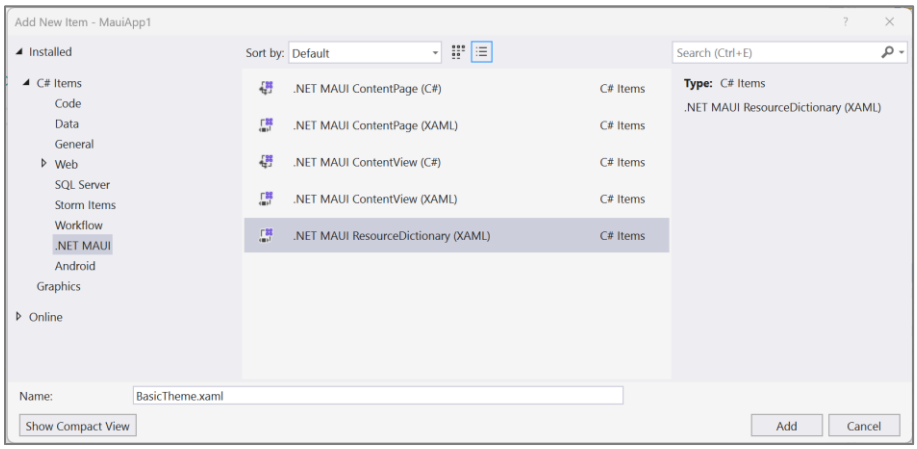
```
</Shell>
```

ResourceDictionaries

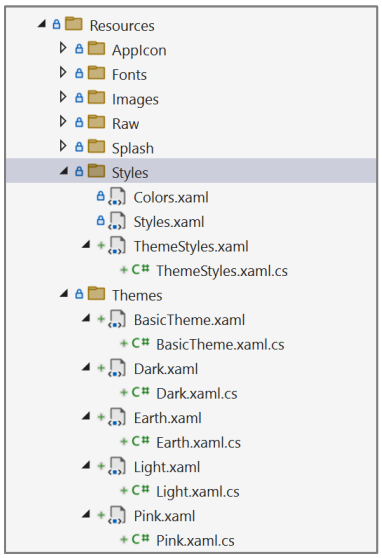
Noch können Sie die App nicht starten, da unsere Resources Files fehlen für die Themes.

Erstellen Sie im Ordner *Resources* einen neuen Ordner *Themes*. Erstellen Sie im Ordner *Themes* folgende ResourceDictionaries:

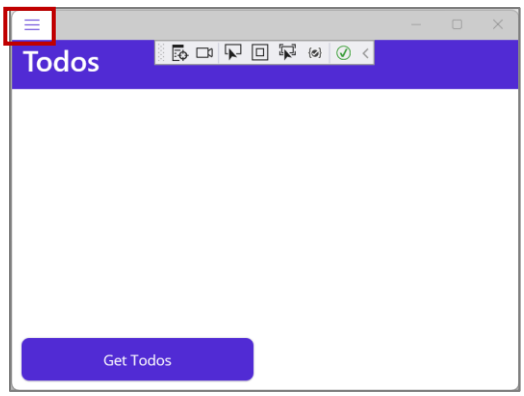
- BasicTheme.xaml
- Dark.xaml
- Earth.xaml
- Light.xaml
- Pink.xaml

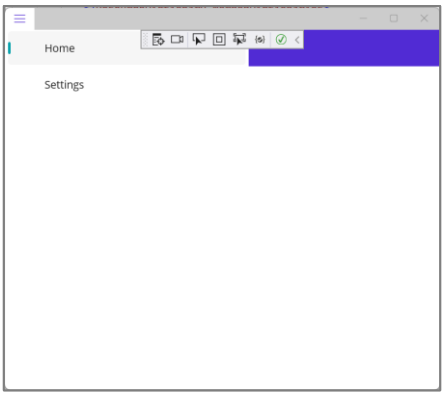


Jetzt fehlt uns noch eine Resource. Erstellen Sie im Ordner *Styles* nochmals eine ResourceDictionary und nennen Sie dieses *ThemeStyles.xaml.* Ihre Styles und Themes Ordner sollten etwa so aussehen:



Starten Sie die Applikation. Wir haben in der App nun Zugriff auf die Navigation erhalten.





Unter Settings haben Sie den Picker zur Verfügung, um das Theme zu ändern. Jedoch wird Ihnen auffallen, dass sich noch nichts ändert. Der Grund dafür ist, dass das File *ThemeStyles* noch leer ist. Am einfachsten wäre es den Inhalt aus dem File *Styles.xaml* in *ThemeStyles* zu kopieren. Das File *Styles.xaml* definiert das Aussehen für jedes GUI-Element in der ganzen App, solange das Aussehen auf einer Page nicht verändert wurde. Sie finden in Styles.xaml ganz oben folgenden Eintrag:

```
<Setter Property="Color" Value="{AppThemeBinding Light={StaticResource Primary},
Dark={StaticResource White}}" />
```

Wenn wir uns diese Zeile genauer anschauen, sehen wir, dass je nach *AppThemeBinding* für Light und Dark verschiedene Farbwerte verwendet werden.
Das *StaticResource* setzt diese Farben beim Start der App statisch, was bedeutet wir können sie dynamisch während der Runtime nicht ändern. Um dynamisch die Werte ändern zu können, müssen wir *DynamicResource* verwenden.

Aktuell ist aber *AppThemeBinding* nicht mit *DynamicResource* kompatibel (voraussichtlich verfügbar ab .NET 8). Aus diesem Grund finden Sie im Anhang eine von mir überarbeitete ResourceDictionary, die Sie als ThemeStyles nutzen können. Kopieren Sie den Inhalt in ihr *ThemeStyles.xaml*.

Als Nächstes definieren wir unsere Themes. Kopieren Sie für *BasicTheme.xaml* folgenden Inhalt:

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
                    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                    x:Class="MauiApp1.Resources.Themes.BasicTheme">
    <Color x:Key="Primary">#512BD4</Color>
    <Color x:Key="Secondary">#DFD8F7</Color>
    <Color x:Key="Tertiary">#2B0B98</Color>
    <Color x:Key="White">White</Color>
    <Color x:Key="Black">Black</Color>
    <Color x:Key="Gray100">#E1E1E1</Color>
    <Color x:Key="Gray200">#C8C8C8</Color>
    <Color x:Key="Gray300">#ACACAC</Color>
    <Color x:Key="Gray400">#919191</Color>
    <Color x:Key="Gray500">#6E6E6E</Color>
    <Color x:Key="Gray600">#404040</Color>
    <Color x:Key="Gray900">#212121</Color>
    <Color x:Key="Gray950">#141414</Color>
    <SolidColorBrush x:Key="PrimaryBrush" Color="{StaticResource Primary}"/>
    <SolidColorBrush x:Key="SecondaryBrush" Color="{StaticResource Secondary}"/>
    <SolidColorBrush x:Key="TertiaryBrush" Color="{StaticResource Tertiary}"/>
    <SolidColorBrush x:Key="WhiteBrush" Color="{StaticResource White}"/>
    <SolidColorBrush x:Key="BlackBrush" Color="{StaticResource Black}"/>
    <SolidColorBrush x:Key="Gray100Brush" Color="{StaticResource Gray100}"/>
    <SolidColorBrush x:Key="Gray200Brush" Color="{StaticResource Gray200}"/>
    <SolidColorBrush x:Key="Gray300Brush" Color="{StaticResource Gray300}"/>
```

```
    <SolidColorBrush x:Key="Gray400Brush" Color="{StaticResource Gray400}"/>
    <SolidColorBrush x:Key="Gray500Brush" Color="{StaticResource Gray500}"/>
    <SolidColorBrush x:Key="Gray600Brush" Color="{StaticResource Gray600}"/>
    <SolidColorBrush x:Key="Gray900Brush" Color="{StaticResource Gray900}"/>
    <SolidColorBrush x:Key="Gray950Brush" Color="{StaticResource Gray950}"/>

    <Color x:Key="Yellow100Accent">#F7B548</Color>
    <Color x:Key="Yellow200Accent">#FFD590</Color>
    <Color x:Key="Yellow300Accent">#FFE5B9</Color>
    <Color x:Key="Cyan100Accent">#28C2D1</Color>
    <Color x:Key="Cyan200Accent">#7BDDEF</Color>
    <Color x:Key="Cyan300Accent">#C3F2F4</Color>
    <Color x:Key="Blue100Accent">#3E8EED</Color>
    <Color x:Key="Blue200Accent">#72ACF1</Color>
    <Color x:Key="Blue300Accent">#A7CBF6</Color>
</ResourceDictionary>
```

Der neue Inhalt gleicht dem vom *Colors.xaml* und wir verwenden dieses BasicTheme, damit wir bei den spezifischen Themes wie Light und Dark nicht alle Farben neu definieren müssen.

Fügen Sie für *Dark.xaml* folgenden Inhalt ein:

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.Resources.Themes.Dark">
    <Color x:Key="Primary">#df3ff9</Color>
    <Color x:Key="Secondary">#ff78ff</Color>
    <Color x:Key="Tertiary">#a900c5</Color>
    <Color x:Key="White">#121212</Color>
    <Color x:Key="Black">White</Color>
    <SolidColorBrush x:Key="PrimaryBrush" Color="{StaticResource Primary}"/>
    <SolidColorBrush x:Key="SecondaryBrush" Color="{StaticResource Secondary}"/>
    <SolidColorBrush x:Key="TertiaryBrush" Color="{StaticResource Tertiary}"/>
    <SolidColorBrush x:Key="WhiteBrush" Color="{StaticResource White}"/>
    <SolidColorBrush x:Key="BlackBrush" Color="{StaticResource Black}"/>
</ResourceDictionary>
```

Fügen Sie für *Earth.xaml* folgenden Inhalt ein:

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.Resources.Themes.Earth">
    <Color x:Key="Primary">Green</Color>
    <Color x:Key="Secondary">LightGreen</Color>
    <Color x:Key="Tertiary">DarkGreen</Color>
    <SolidColorBrush x:Key="PrimaryBrush" Color="{StaticResource Primary}"/>
    <SolidColorBrush x:Key="SecondaryBrush" Color="{StaticResource Secondary}"/>
    <SolidColorBrush x:Key="TertiaryBrush" Color="{StaticResource Tertiary}"/>
</ResourceDictionary>
```

Fügen Sie für *Light.xaml* folgenden Inhalt ein:

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.Resources.Themes.Light">
    <Color x:Key="Primary">#00797e</Color>
    <Color x:Key="Secondary">#aee8e9</Color>
    <Color x:Key="Tertiary">#00b1c0</Color>
    <SolidColorBrush x:Key="PrimaryBrush" Color="{StaticResource Primary}"/>
    <SolidColorBrush x:Key="SecondaryBrush" Color="{StaticResource Secondary}"/>
    <SolidColorBrush x:Key="TertiaryBrush" Color="{StaticResource Tertiary}"/>
</ResourceDictionary>
```

Fügen Sie für *Pink.xaml* folgenden Inhalt ein:

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
                    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                    x:Class="MauiApp1.Resources.Themes.Pink">
    <Color x:Key="Primary">#df3ff9</Color>
    <Color x:Key="Secondary">#ff78ff</Color>
    <Color x:Key="Tertiary">#a900c5</Color>

    <SolidColorBrush x:Key="PrimaryBrush" Color="{StaticResource Primary}"/>
    <SolidColorBrush x:Key="SecondaryBrush" Color="{StaticResource Secondary}"/>
    <SolidColorBrush x:Key="TertiaryBrush" Color="{StaticResource Tertiary}"/>
</ResourceDictionary>
```

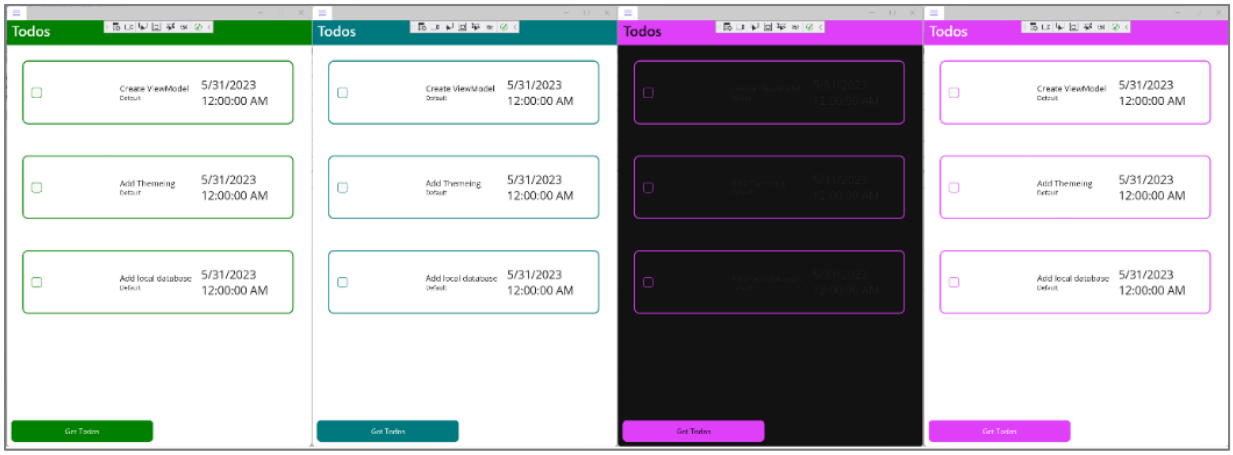
Wir benötigen noch zwei letzte Änderungen im Code. Dies ist notwendig damit unsere App unser definiertes Thema bereits beim Aufstarten verwendet. Ersetzen Sie die zwei Zeilen innerhalb von *ResourceDictionary.MergedDictionaries* von *App.xaml* mit folgender Zeile:

```
<ResourceDictionary Source="Resources/Themes/BasicTheme.xaml"/>
```

Fügen Sie in *App.xaml.cs* nach *InitializeComponent();* diese Zeile ein:

```
Resources.MergedDictionaries.Add(new MauiApp1.Resources.Styles.ThemeStyles());
```

Starten Sie die App nun neu. Wenn Sie das Theme nun ändern, sollte sich die ganze Applikation farblich auch ändern.



Anhang

ThemeStyles.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ResourceDictionary xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.Resources.Styles.ThemeStyles">
    <Style TargetType="ActivityIndicator">
        <Setter Property="Color" Value="{DynamicResource Primary}" />
    </Style>

    <Style TargetType="IndicatorView">
        <Setter Property="IndicatorColor" Value="{DynamicResource Gray200}"/>
        <Setter Property="SelectedIndicatorColor" Value="{DynamicResource
Gray950}"/>
    </Style>

    <Style TargetType="Border">
        <Setter Property="Stroke" Value="{DynamicResource Primary}" />
        <Setter Property="StrokeShape" Value="RoundRectangle 10,10,10,10"/>
        <Setter Property="StrokeThickness" Value="2"/>
        <Setter Property="Padding" Value="10"/>
        <Setter Property="Margin" Value="20"/>
    </Style>

    <Style TargetType="BoxView">
        <Setter Property="Color" Value="{DynamicResource Gray950}" />
    </Style>

    <Style TargetType="Button">
        <Setter Property="TextColor" Value="{DynamicResource White}" />
        <Setter Property="BackgroundColor" Value="{DynamicResource Primary}" />
        <Setter Property="FontFamily" Value="OpenSansRegular"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="CornerRadius" Value="8"/>
        <Setter Property="Padding" Value="14,10"/>
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray950}" />
                            <Setter Property="BackgroundColor"
Value="{DynamicResource Gray200}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="CheckBox">
        <Setter Property="Color" Value="{DynamicResource Primary}" />
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="Color" Value="{DynamicResource
Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
```

```
        </Setter>
    </Style>

    <Style TargetType="DatePicker">
        <Setter Property="TextColor" Value="{DynamicResource Gray900}" />
        <Setter Property="BackgroundColor" Value="Transparent" />
        <Setter Property="FontFamily" Value="OpenSansRegular"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray200}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="Editor">
        <Setter Property="TextColor" Value="{DynamicResource Black}" />
        <Setter Property="BackgroundColor" Value="Transparent" />
        <Setter Property="FontFamily" Value="OpenSansRegular"/>
        <Setter Property="FontSize" Value="14" />
        <Setter Property="PlaceholderColor" Value="{DynamicResource Gray200}" />
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="Entry">
        <Setter Property="TextColor" Value="{DynamicResource Black}" />
        <Setter Property="BackgroundColor" Value="Transparent" />
        <Setter Property="FontFamily" Value="OpenSansRegular"/>
        <Setter Property="FontSize" Value="14" />
        <Setter Property="PlaceholderColor" Value="{DynamicResource Gray200}" />
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="Frame">
        <Setter Property="HasShadow" Value="False" />
        <Setter Property="BorderColor" Value="{DynamicResource Gray200}" />
```

```
        <Setter Property="CornerRadius" Value="8" />
    </Style>

    <Style TargetType="ImageButton">
        <Setter Property="Opacity" Value="1" />
        <Setter Property="BorderColor" Value="Transparent"/>
        <Setter Property="BorderWidth" Value="0"/>
        <Setter Property="CornerRadius" Value="0"/>
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="Opacity" Value="0.5" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="Label">
        <Setter Property="TextColor" Value="{DynamicResource Gray900}" />
        <Setter Property="FontFamily" Value="OpenSansRegular" />
        <Setter Property="FontSize" Value="Medium" />
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="ListView">
        <Setter Property="SeparatorColor" Value="{DynamicResource Gray200}" />
        <Setter Property="RefreshControlColor" Value="{DynamicResource Gray900}" />
    </Style>

    <Style TargetType="Picker">
        <Setter Property="TextColor" Value="{DynamicResource Gray900}" />
        <Setter Property="TitleColor" Value="{DynamicResource Gray900}" />
        <Setter Property="BackgroundColor" Value="Transparent" />
        <Setter Property="FontFamily" Value="OpenSansRegular"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray300}" />
                            <Setter Property="TitleColor" Value="{DynamicResource
Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>
```

```
    <Style TargetType="ProgressBar">
        <Setter Property="ProgressColor" Value="{DynamicResource Primary}" />
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="ProgressColor"
Value="{DynamicResource Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="RadioButton">
        <Setter Property="Background" Value="Transparent"/>
        <Setter Property="TextColor" Value="{DynamicResource Black}" />
        <Setter Property="FontFamily" Value="OpenSansRegular"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="RefreshView">
        <Setter Property="RefreshColor" Value="{DynamicResource Gray900}" />
    </Style>

    <Style TargetType="SearchBar">
        <Setter Property="TextColor" Value="{DynamicResource Gray900}" />
        <Setter Property="PlaceholderColor" Value="{DynamicResource Gray500}" />
        <Setter Property="CancelButtonColor" Value="{DynamicResource Gray500}" />
        <Setter Property="BackgroundColor" Value="Transparent" />
        <Setter Property="FontFamily" Value="OpenSansRegular" />
        <Setter Property="FontSize" Value="14" />
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray300}" />
                            <Setter Property="PlaceholderColor"
Value="{DynamicResource Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="SearchHandler">
        <Setter Property="TextColor" Value="{DynamicResource Gray900}" />
```

```
        <Setter Property="PlaceholderColor" Value="{DynamicResource Gray500}" />
        <Setter Property="BackgroundColor" Value="Transparent" />
        <Setter Property="FontFamily" Value="OpenSansRegular" />
        <Setter Property="FontSize" Value="14" />
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray300}" />
                            <Setter Property="PlaceholderColor"
Value="{DynamicResource Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="Shadow">
        <Setter Property="Radius" Value="15" />
        <Setter Property="Opacity" Value="0.5" />
        <Setter Property="Brush" Value="{DynamicResource White}" />
        <Setter Property="Offset" Value="10,10" />
    </Style>

    <Style TargetType="Slider">
        <Setter Property="MinimumTrackColor" Value="{DynamicResource Primary}" />
        <Setter Property="MaximumTrackColor" Value="{DynamicResource Gray200}" />
        <Setter Property="ThumbColor" Value="{DynamicResource Primary}" />
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="MinimumTrackColor"
Value="{DynamicResource Gray300}"/>
                            <Setter Property="MaximumTrackColor"
Value="{DynamicResource Gray300}"/>
                            <Setter Property="ThumbColor" Value="{DynamicResource
Gray300}"/>
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="SwipeItem">
        <Setter Property="BackgroundColor" Value="{DynamicResource White}" />
    </Style>

    <Style TargetType="Switch">
        <Setter Property="OnColor" Value="{DynamicResource Primary}" />
        <Setter Property="ThumbColor" Value="{DynamicResource White}" />
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="OnColor" Value="{DynamicResource
Gray300}" />
                            <Setter Property="ThumbColor" Value="{DynamicResource
Gray300}" />
```

```
                        </VisualState.Setters>
                    </VisualState>
                    <VisualState x:Name="On">
                        <VisualState.Setters>
                            <Setter Property="OnColor" Value="{DynamicResource
Secondary}" />
                            <Setter Property="ThumbColor" Value="{DynamicResource
Primary}" />
                        </VisualState.Setters>
                    </VisualState>
                    <VisualState x:Name="Off">
                        <VisualState.Setters>
                            <Setter Property="ThumbColor" Value="{DynamicResource
Gray400}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="TimePicker">
        <Setter Property="TextColor" Value="{DynamicResource Gray900}" />
        <Setter Property="Background" Value="Transparent"/>
        <Setter Property="FontFamily" Value="OpenSansRegular"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="TextColor" Value="{DynamicResource
Gray300}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>

    <Style TargetType="Page" ApplyToDerivedTypes="True">
        <Setter Property="Padding" Value="0"/>
        <Setter Property="BackgroundColor" Value="{DynamicResource White}" />
    </Style>

    <Style TargetType="Shell" ApplyToDerivedTypes="True">
        <Setter Property="Shell.BackgroundColor" Value="{DynamicResource Primary}"
/>
        <Setter Property="Shell.ForegroundColor" Value="{OnPlatform
WinUI={StaticResource Primary}, Default={StaticResource White}}" />
        <Setter Property="Shell.TitleColor" Value="{DynamicResource White}" />
        <Setter Property="Shell.DisabledColor" Value="{DynamicResource Gray200}" />
        <Setter Property="Shell.UnselectedColor" Value="{DynamicResource Gray200}"
/>
        <Setter Property="Shell.NavBarHasShadow" Value="False" />
        <Setter Property="Shell.TabBarBackgroundColor" Value="{DynamicResource
White}" />
        <Setter Property="Shell.TabBarForegroundColor" Value="{DynamicResource
Primary}" />
        <Setter Property="Shell.TabBarTitleColor" Value="{DynamicResource Primary}"
/>
        <Setter Property="Shell.TabBarUnselectedColor" Value="{DynamicResource
Gray900}" />
    </Style>

    <Style TargetType="NavigationPage">
        <Setter Property="BarBackgroundColor" Value="{DynamicResource Primary}" />
```

```
        <Setter Property="BarTextColor" Value="{DynamicResource Gray200}" />
        <Setter Property="IconColor" Value="{DynamicResource Gray200}" />
    </Style>

    <Style TargetType="TabbedPage">
        <Setter Property="BarBackgroundColor" Value="{DynamicResource White}" />
        <Setter Property="BarTextColor" Value="{DynamicResource Primary}" />
        <Setter Property="UnselectedTabColor" Value="{DynamicResource Gray200}" />
        <Setter Property="SelectedTabColor" Value="{DynamicResource Gray950}" />
    </Style>

    <Style TargetType="Grid">
        <Setter Property="VisualStateManager.VisualStateGroups">
            <VisualStateGroupList>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Selected">
                        <VisualState.Setters>
                            <Setter Property="BackgroundColor"
                                    Value="{DynamicResource Secondary}" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateGroupList>
        </Setter>
    </Style>
</ResourceDictionary>
```