# Wetter App Hands On Lab

Ziel der Anwendung ist eine Kiosk App, die dauerhaft auf Großbildschirmen läuft. Es wird das Wetter aller 12 ppedv Standorte angezeigt. Jeweils die 5 Tage Vorschau mit rotierender Anzeige. Betriebssystem ist Windows 10 Anniversary Update

Voraussetzungen:

Visual Studio 2017

Internet Zugang

Email Account

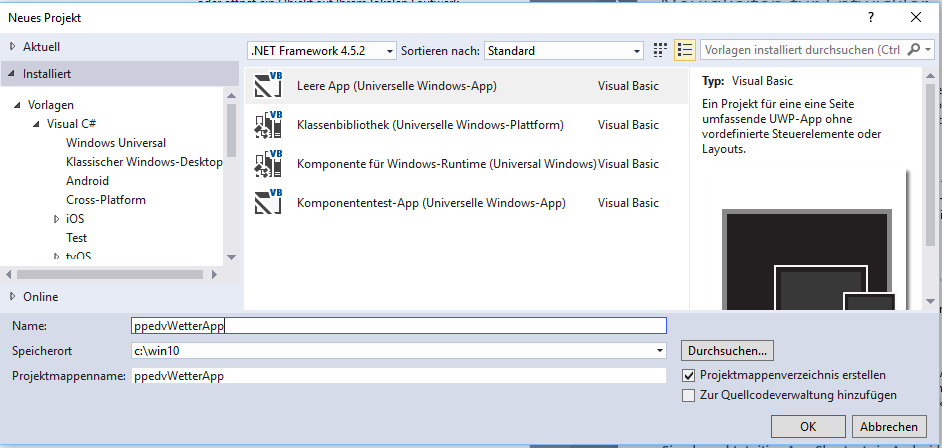
Microsoft ID

Kreditkarte mit 14€ belastend

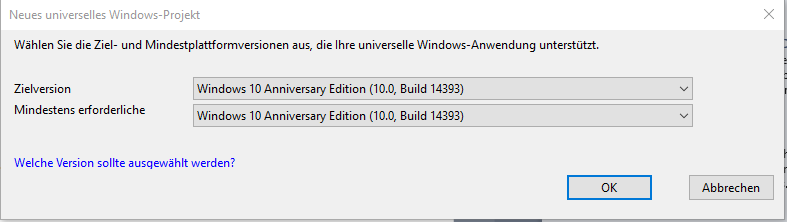
# Lab 1 App erzeugen und UWP Toolkit einbinden

Dauer: 20 Min

Visual Studio 2017 starten und Universal Windows App Projekt anlegen



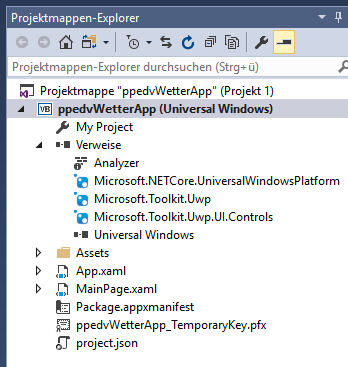
Min OS Version Build 14393

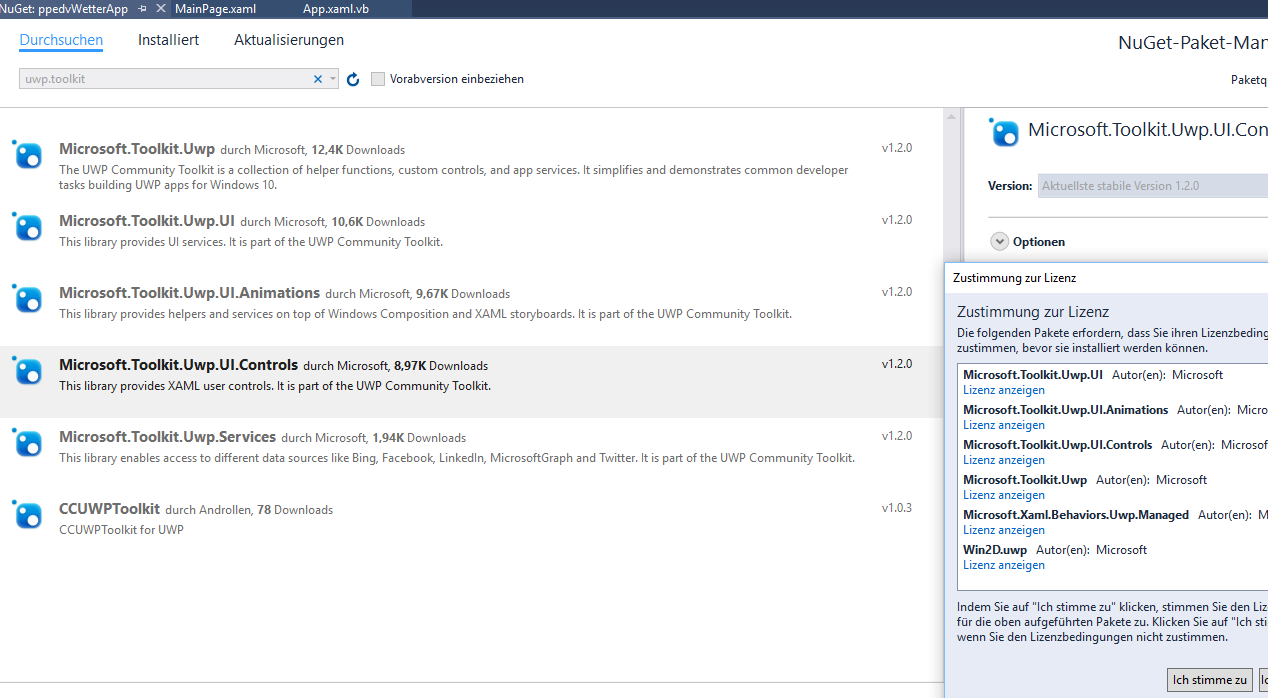


Nuget

Microsoft.Toolkit.Ui.UWP

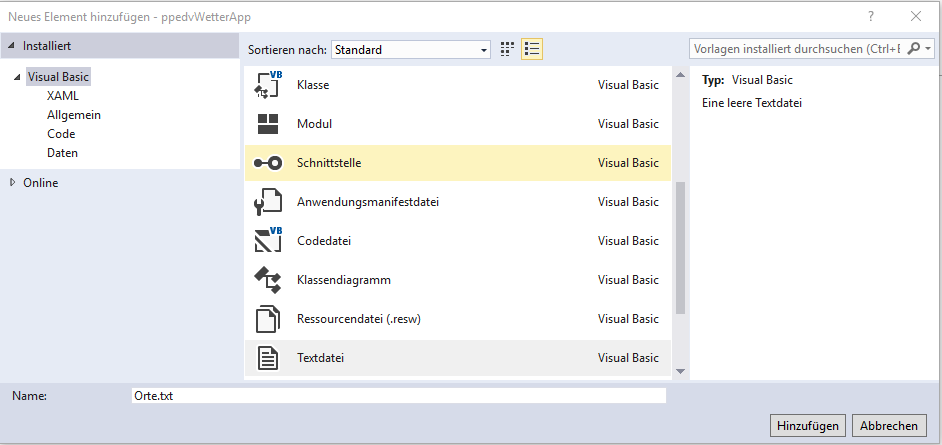
Microsoft.Toolkit.Ui.UWP.Controls





Binding Basics mit Orte Datei

Projekt Element Hinzufügen TEXTDATEI



Orte.txt . Diese mit den Orten füllen. Nach jeder Zeile ein Return außer bei der Letzen.

Burghausen

Berlin

München

Nürnberg

Stuttgart

Leipzig

Wien

Frankfurt

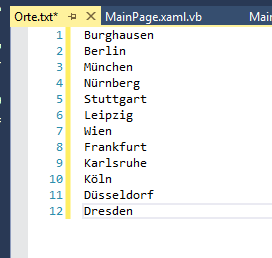
Karlsruhe

Köln

Düsseldorf

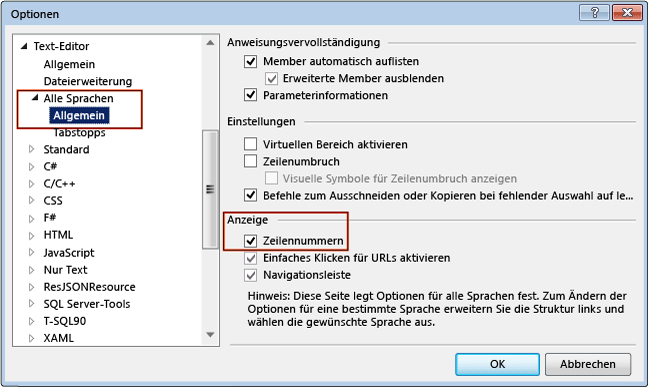
Dresden

Der Editor muss 12 Zeilen anzeigen.



Zeilennummern für den Code anzeigen

Wählen Sie in der Menüleiste **Extras**, **Optionen**. Erweitern Sie den Knoten **Text-Editor**, und wählen Sie entweder den Knoten für die Sprache, die Sie verwenden, oder **Alle Sprachen** aus, um die Zeilennummern in allen Sprachen zu aktivieren. Sie können die Zeilennummer auch im Feld **Schnellstart** eingeben.

1. 

Visual Studio Projekt, rechte Maustaste, hinzufügen Klasse- wetter.vb anlegen und mit der Ort Eigenschaft versehen.

Public Class wetter

Public Property Ort As String

End Class

Die Datei Mainpage.xaml.vb öffnen. Das Event Loaded erzeugen lassen. Auswahl mittlere DropDown Oben Mainpage Ereignisse, Rechts Dropdown Loaded auswählen und Click



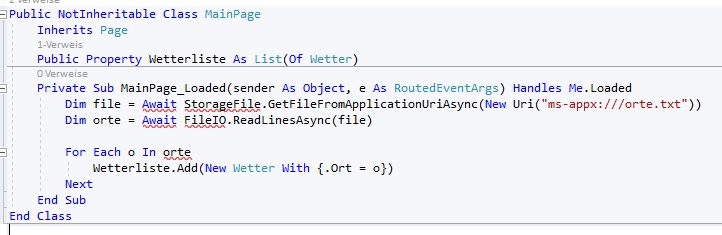
Ergebnis:

Public NotInheritable Class MainPage

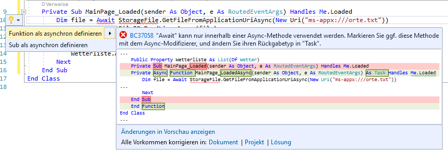
Inherits Page

Private Sub MainPage\_Loaded(sender As Object, e As RoutedEventArgs) Handles Me.Loaded

Property WetterListe für generische Liste der Orte anlegen



Fehler im Code per Maus Cursor setzen. [STRG]+[.] drücken und korrekturen Anwenden



2 Aktionen sind nötig

Methode als Async markieren

Windows.Storages Namensraum inkludieren

Öffnen der Datei MainPage.XAML

Einfügen des XAML RotatorTile Steuerelements



Das Element markieren [STRG]+[.] drücken und per Intellisense den Namensraum einfügen lassen. xmlns:Controls="using:Microsoft.Toolkit.Uwp.UI.Controls"

Die blaue Unterwellung verschwindet.

Achtung: XAML ist im Gegensatz zu VB.NET Case Sensitiv. Verwendete Eigenschaftsnamen müssen exakt den Namen im Code entsprechen. Es empfiehlt sich die CamelCase Notation.

XAML Coder ergänzen, so das im Ergebnis folgende Code im Visual Studio Editor steht

<Grid Background="CornflowerBlue">

<Controls:RotatorTile

Direction="Up"

RotationDelay="0:0:5"

ExtraRandomDuration="0:0:0"

ItemsSource="{x:Bind WetterListe, Mode=OneWay}"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch" >

<Controls:RotatorTile.ItemTemplate>

<DataTemplate x:DataType="local:Wetter">

<TextBlock Text="{x:Bind Ort, Mode=OneWay}" FontSize="28" Foreground="Yellow"/>

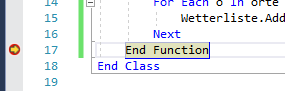
</DataTemplate>

</Controls:RotatorTile.ItemTemplate>

</Controls:RotatorTile>

</Grid>

Mit F5 das Programm ausführen. Es werden keine Orte angezeigt. Mit F9 im Code an Zeile 17 (siehe Bild) einen Breakpoint setzen und die Daten überprüfen. Einfach den Cursor auf die Variable WetterListe bewegen.

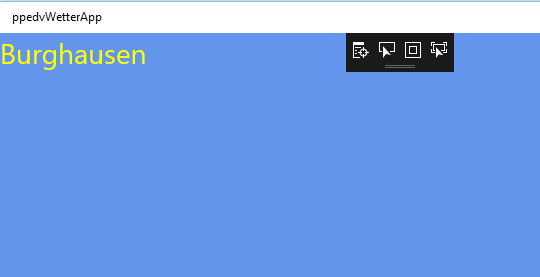


Es wird angezeigt per Popup Count=12

Code Ändern von generischer Liste auf Observable Collection um die späte Änderung der Daten dem Userinterface auch mitteilen zu können.

Public Property WetterListe As ObservableCollection(Of Wetter) = New ObservableCollection(Of Wetter)

Nun rotiert die App von unten kommend alle 5 Sekunden einen neuen Ortstnamen

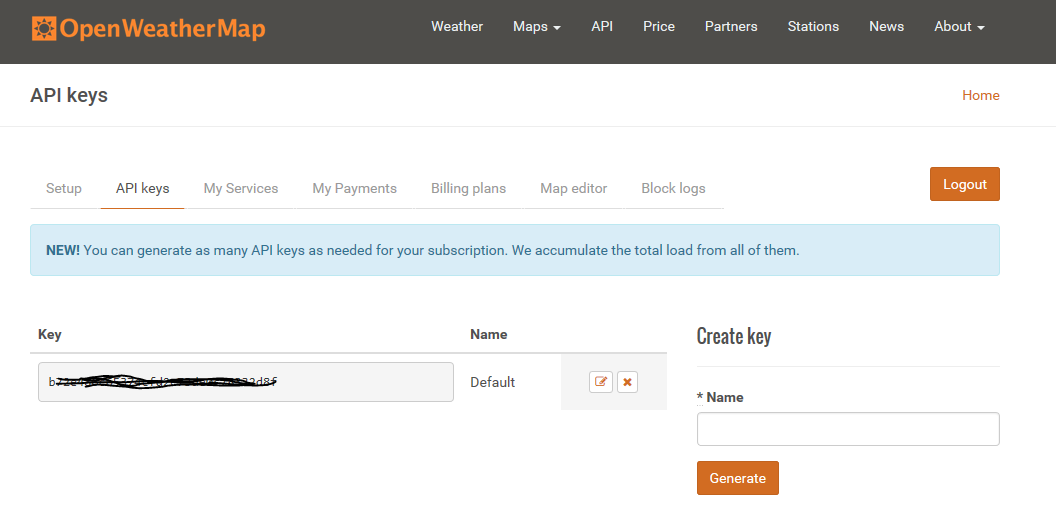


# Lab 2 Wetter Service einbinden

Dauer: 15 Min

Ein öffentlich verfügbarer REST Service liefert die Wetter Daten im JSON Format.

Benutzer Anlegen und Anmelden bei <https://openweathermap.org/api> über Sign Up.



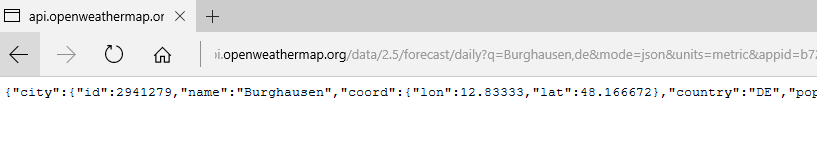
Api Key abrufen und lokal speichern, in beliebiger Textdatei.

Es wird der 16 Tage Service <https://openweathermap.org/forecast16> genutzt.

Die Url von der Website kopieren und in einem Texteditor (zb Notepad) bearbeiten. API Key einfügen. Die Url kopierne und zum Test im Browser einfügen.

http://api.openweathermap.org/data/2.5/forecast/daily?q=Burghausen&units=metric&appid=b72e4xxxx6537xxxx8f

Erwartetes Ergebnis



Diese Zeichenfolge markieren ([STRG]+[A]) kopieren ([STRG]+[C]). Im Projekt (Visual Studio) eine Textdatei hinzufügen mit dem Namen daten.json. In diese den Inhalt der Zwischenablage einfügen ([STRG]+[V]) und speichern.



Diese Wetterdaten im JSON Format können nun pro Ort geladen und zur Anzeige gebracht werden. Dazu wird die Klasse HttpClient instanziiert. Namensraum einbinden per Intellisense. Bisherigen Code ergänzen.

Dim client = New HttpClient()

Dim file = Await StorageFile.GetFileFromApplicationUriAsync(New Uri("ms-appx:///orte.txt"))

Dim orte = Await FileIO.ReadLinesAsync(file)

For Each Ort In orte

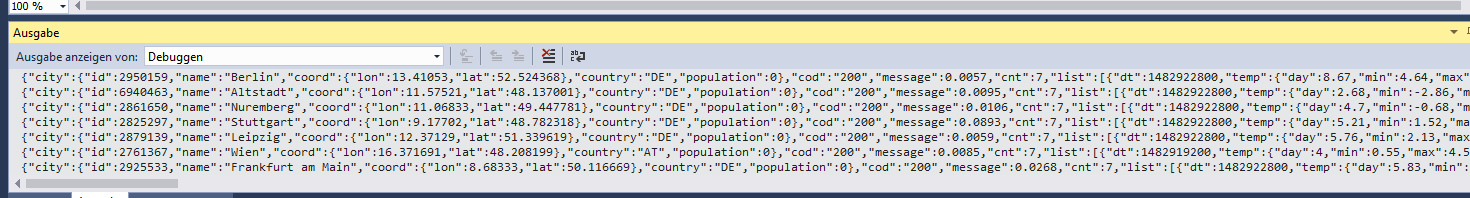
WetterListe.Add(New Wetter With {.Ort = Ort})

Dim json = Await client.GetStringAsync($"http://api.openweathermap.org/data/2.5/forecast/daily?q={Ort}&units=metric&appid=b72e486c65374cfd2e73de0c78332d8f")

Debug.WriteLine(json)

Next

Mit F5 die Anwendung starten. Die Ausgabe erfolgt im Visual Studio Ausgabe Fenster

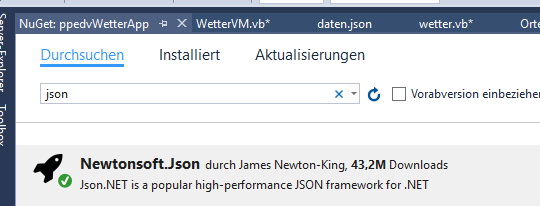


# Lab 3 MVVM Ansatz verwirklichen

Dauer: 40 Min

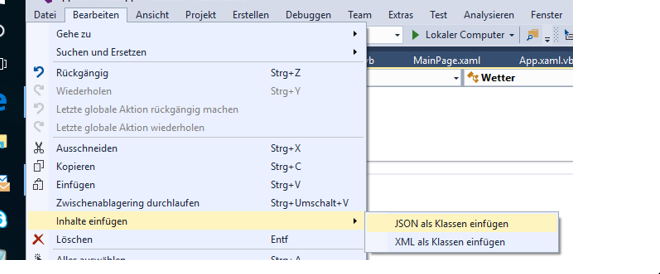
Mit dem Model View ViewModel Archtitektur Pattern, werden in der Regel alle XAML Anwendungen realisiert. Dabei werden die Daten an das Userinterface gebunden und dieses bei Veränderung automatisch aktualisiert. In diesem Beispiel dauert das Laden der zwölf Wetter Datensätze einfach länger. Das User Interface wird aber sofort angezeigt.

Der JSON String wird direkt in ein Viewmodel deserialisiert. Dafür kommt die Bibliothek Newtonsoft.Json zum Einsatz. Per Visual Studio-rechtsclick auf Projekt - Nuget Paketmanager, durchsuchen (json) und installieren.



Im ersten Schritt wird das Model erzeugt. Dies Klassen und Eigenschaften werden analog der JSON Daten erstellt. Einen kompletten JSON Datensatz, z.B. aus der gespeicherten Datei daten.json in die Zwischenablage kopieren. Mauscursor auf letzte Zeile in Wetter.vb setzen.

Visual Studio Menü - Bearbeiten –Inhalte einfügen- JSON als Klassen einfügen.



Ergebnis ist die Modelklasse mit ca 50 Zeilen Code.

Im nächsten Schritt das Viewmodel als Klasse mit dem Namen WetterVM.vb hinzugefügt. Das Viewmodel hält die Daten als Liste von RootObject Objekten (siehe wetter.vb) und stelle ein Event bereit um die Daten zu laden. Diesen kann man Großteils aus der Datei MainPage.xaml.vb und dem loaded Event kopieren, da er dort nicht mehr benötigt wird.

Public WetterListe As ObservableCollection(Of Rootobject) = New ObservableCollection(Of Rootobject)

Public Async Sub loadVMAsync()

Dim client = New HttpClient()

Dim file = Await StorageFile.GetFileFromApplicationUriAsync(New Uri("ms-appx:///orte.txt"))

Dim orte = Await FileIO.ReadLinesAsync(file)

For Each Ort In orte

Dim wetter As Rootobject

Dim json = Await client.GetStringAsync($"http://api.openweathermap.org/data/2.5/forecast/daily?q={Ort}&units=metric&appid=b72e486c65374cfd2e73de0c78332d8f")

wetter = JsonConvert.DeserializeObject(Of Rootobject)(json)

wetter.city.name = Ort 'Fix für Wetterstationsname

WetterListe.Add(wetter)

Next

End Sub

Den Code aus der Methode laoded in mainpage.xaml.vb auskommentieren. Markieren des Code Teils [STRG]+[K] [STRG] +[C]

Alternativ per Visual Studio Menü



Nun fehlt nur noch der View, also die MainPage.XAML. Es kommen die neuen Compiled Bindings zum Einsatz. Die Page bekommt als Property eine Objekt vom Typ des Viewmodels. In der loaded Funktion wird dann die das Viewmodel als Instanz zugewiesen und gefüllt.

Im Codebehind des Viewmodels werden zwei Zeilen Code eingefügt.

Public Property Wetter As WetterVM = New WetterVM

Private Async Function MainPage\_LoadedAsync(sender As Object, e As RoutedEventArgs) As Task Handles Me.Loaded

Wetter.loadVMAsync()

Im XAML der View muss dann der Binding Pfad verändert werden. Die Wurzelklasse heist im Model (automatisch generiert) eben RootObject

ItemsSource="{x:Bind Wetter.WetterListe, Mode=OneWay}"

…

<DataTemplate x:DataType="local:Rootobject">

<TextBlock Text="{x:Bind city.name, Mode=OneWay}" FontSize="28" Foreground="Yellow"/>

Wenn alles korrekt ist, muss Visual Studio Intellisense die Namen der Felder nach x:Bind vorschlagen.

Zusätzlich wird rein für Forschungszwecke ein Counter eingebaut. Hier per Stackpanel ein Textblock und das RotatorTile gruppiert.

<StackPanel Orientation="Horizontal" Height="30" VerticalAlignment="Top">

<TextBlock Height="30" Text="{x:Bind Wetter.WetterListe.Count,Mode=OneWay}" Width="100"></TextBlock>

<Controls:RotatorTile

Wenn Sie nun F5 drücken wird eine Exception beim deserialisieren auftreten.

Newtonsoft.Json.JsonSerializationException ist aufgetreten.

HResult=0x80131500

Nachricht = Cannot deserialize the current JSON array (e.g. [1,2,3]) into type 'ppedvWetterApp.List' because the type requires a JSON object (e.g. {"name":"value"}) to deserialize correctly.

To fix this error either change the JSON to a JSON object (e.g. {"name":"value"}) or change the deserialized type to an array or a type that implements a collection interface (e.g. ICollection, IList) like List<T> that can be deserialized from a JSON array. JsonArrayAttribute can also be added to the type to force it to deserialize from a JSON array.

Path 'list', line 1, position 159.

bei Newtonsoft.Json.JsonConvert.DeserializeObject(String value, Type type, JsonSerializerSettings settings)

bei Newtonsoft.Json.JsonConvert.DeserializeObject[T](String value, JsonSerializerSettings settings)

bei Newtonsoft.Json.JsonConvert.DeserializeObject[T](String value)

bei ppedvWetterApp.WetterVM.VB$StateMachine\_2\_loadVMAsync.MoveNext() in c:\win10\ppedvWetterApp\ppedvWetterApp\WetterVM.vb: Zeile19

Das Format der JSON Daten steht teilweise im Wiederspruch mit reservierten Namen von .net oder der Programmiersprache. In manchenFällen kann der JSON Klassen Generator auch den Typ nicht korrekt erkennen.

Folgende Aktionen in wetter.vb durchführen

Klasse List in Liste umbenennen

In Klasse Rootobjekt Eigenschaft list() -> Public Property wlist As List(Of Liste)

Serialisierungs Attribut wlist voranstellen <JsonProperty("list")>

In Klasse Liste weather -> Public Property weather As List(Of Weather)

Wenn nun die Anwendung mit F5 gestarte wird sieht man den Zähler von 0 auf 12 im linken oberen Eck der APP hochzählen.

Um die Tagesdaten horizontal zum Ort anzuzeigen wird ein Gridview Element eingebaut. Die Feldnamen in der Bindung ergeben sich immer aus dem Datenmodell. Auch die adressierung per Index ist möglich (weather[0])

<Grid Background="CornflowerBlue">

<StackPanel>

<TextBlock Height="30" Text="{x:Bind Wetter.WetterListe.Count,Mode=OneWay}" Width="100"></TextBlock>

<Controls:RotatorTile Height="400"

Direction="Up"

RotationDelay="0:0:5"

ExtraRandomDuration="0:0:0"

ItemsSource="{x:Bind Wetter.WetterListe, Mode=OneWay}"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch" >

<Controls:RotatorTile.ItemTemplate>

<DataTemplate x:DataType="local:Rootobject">

<StackPanel>

<TextBlock Text="{x:Bind city.name, Mode=OneWay}" FontSize="28" Foreground="Yellow"/>

<GridView ItemsSource="{x:Bind wlist,Mode=OneWay}" HorizontalAlignment="Stretch" >

<GridView.ItemTemplate>

<DataTemplate x:DataType="local:Liste">

<Grid CornerRadius="3" Background="#7FFFFFFF" Width="100" Height="200">

<TextBlock Text="{x:Bind dt}"/>

<TextBlock Text="{x:Bind temp.day, Mode=OneWay}" Margin="0,50,0,0"/>

<TextBlock Text="{x:Bind weather[0].description, Mode=OneWay}" Margin="0,30,0,0" />

</Grid>

</DataTemplate>

</GridView.ItemTemplate>

</GridView>

</StackPanel>

</DataTemplate>

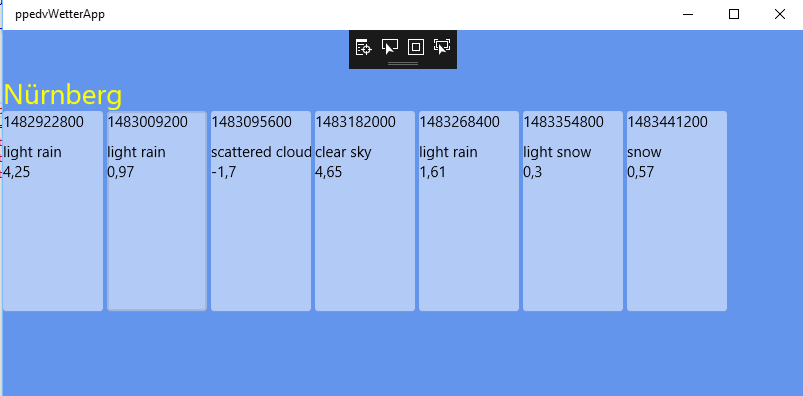
</Controls:RotatorTile.ItemTemplate>

</Controls:RotatorTile>

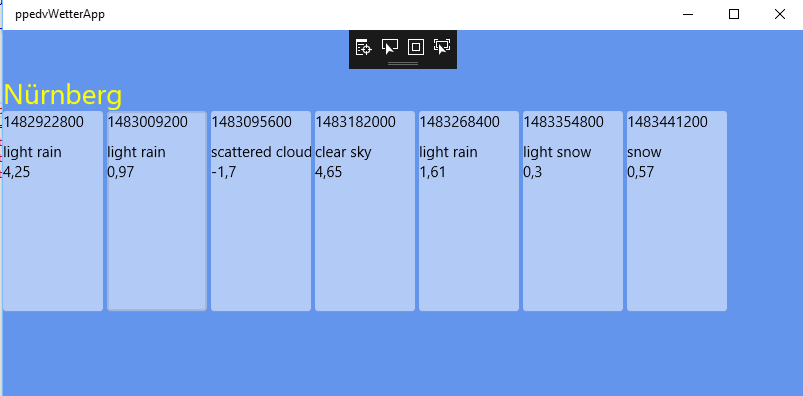
</StackPanel>

</Grid>

Der folgende Screen zeigt das soll Resultat. Die Zeit ist 1/1000 Sekunden seit 1.1.1970 angegeben.



End Class



Um die Anzeige des Datums menschlesbar zu machen wird das Feature Function Binding verwendet. Eine Funktion wird der der Model Klasse Liste hinzugefügt, die ein formatiertes Datum liefert.

Public Function Int2Datum(s As Integer) As String

Dim d = New DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc)

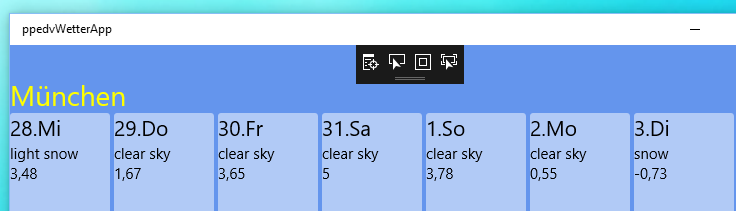
Dim dd = String.Format("{0:d.ddd}", d.AddSeconds(s))

Return dd

End Function

Das Binding des Feldes dt in der MainPage.XAML wird geändert

<TextBlock Text="{x:Bind Int2Datum(dt), Mode=OneWay}" FontSize="14" />



Nach dem Scroll, flackert die Gridviewliste schnell und kurz und schiebt dann die Felder neu von unten rein. Das könnte man im passenden Template abstellen, erledigt sich aber mit der weiteren Aufgabenstellung.

# Lab 4 UI Design

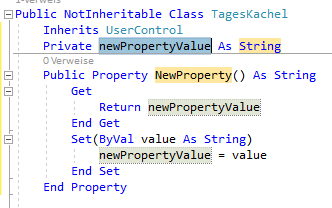
Dauer: 30 Min

Die App soll auf verschiedenen Screen Größen im Vollbild Modus laufen und dabei immer den maximalen Platz ausnutzen. Mittels Grid Layout lassen sich variable Größe per \* definieren.

Das Gridview Listen Steuerelement wird durch ein Layout Grid ersetzt. Für Wiederverwendbarkeit einer „Temp Kachel“ wird ein Benutzer Steuerelement hinzugefügt.

Visual Studio- Rechtsclick auf Projekt- hinzufügen neues Element –Benutzersteuerlement mit dem Namen TagesKachel.XAML.

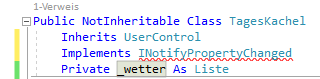
Die Datei TagesKachel.xaml.vb editieren. Im Editor tippen [PROP] [TAB] [TAB]



Durch die markierten Eigenschaften per Tab navigieren und die Bezeichnung ändern in \_wetter vom Typ Liste und Wetter.

Das Usercontrol ist so ein eigenes kleines Viewmodel, das per Property Wetter später in der MainPage.xaml gebunden wird. Um die Änderungen des Propertys Wetter zuverlässig in die UI zu reflektieren bekommt das Viewmodel (das ident mit dem View ist) das Interface InotifypropertChanged Implementiert.

In der nach Inherits folgenden Zeile Implements InotifypropertChanged einfügen



Per Intelisense [STRG] [.] auf dem unterwellten Objekt den Implementierungscode automatisch einfügen lassen.

Nach der Zuweisung im Setter den Code für die Benachrichtigung einfügen

RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs("Wetter"))

Der Codebhind des Usercontrols sieht nun so aus

Public NotInheritable Class TagesKachel

Inherits UserControl

Implements INotifyPropertyChanged

Private \_wetter As Liste

Public Property Wetter() As Liste

Get

Return \_wetter

End Get

Set(ByVal value As Liste)

\_wetter = value

RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs("Wetter"))

End Set

End Property

Public Event PropertyChanged As PropertyChangedEventHandler Implements INotifyPropertyChanged.PropertyChanged

End Class

Im XAML View wird die Wetter Kachel per Grid definiert.

<Viewbox Stretch="Fill">

<Grid CornerRadius="3" Background="#7FFFFFFF" Padding="5,0,5,0"

VerticalAlignment="Stretch" HorizontalAlignment="Stretch">

<Grid.RowDefinitions>

<RowDefinition Height="30\*"/>

<RowDefinition Height="100\*"/>

<RowDefinition Height="10\*"/>

</Grid.RowDefinitions>

<TextBlock Text="{x:Bind Wetter.Int2Datum(Wetter.dt), Mode=OneWay}" FontSize="14" />

<StackPanel Orientation="Horizontal" Grid.Row="1">

<TextBlock Text="{x:Bind Wetter.temp.AsInt(Wetter.temp.day),FallbackValue=0,TargetNullValue=0, Mode=OneWay}" FontSize="20"/>

<TextBlock Text="°" Grid.Row="1" FontSize="20"/>

<Image Source="{x:Bind Wetter.temp.AsPfad(Wetter.weather[0].icon),Mode=OneWay}" Margin="-10"></Image>

</StackPanel>

<TextBlock Text="{x:Bind Wetter.weather[0].description, Mode=OneWay}" FontSize="8" Grid.Row="2"/>

</Grid>

</Viewbox>

Es muss noch in der Datei Wetter.vb in der Klasse Temp die Funktion AsPfad zur Konvertierung in den Pfad für die Bildquelle hinzugefügt werden. Zusätzlich coden sie eine Funktion AsInt zum runden der Temperatur auf Ganzzahl in die gleich Klasse.

Public Function AsPfad(icon As String) As ImageSource

Dim bi = New BitmapImage(New Uri($"ms-appx:///Assets/{icon}.png", UriKind.RelativeOrAbsolute))

Return bi

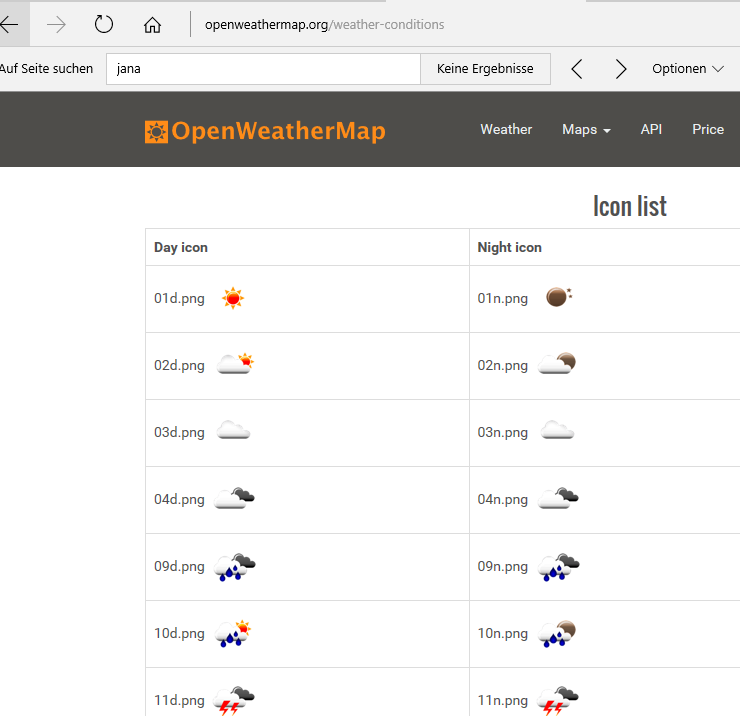
End Function

Public Function AsInt(t As Double) As String

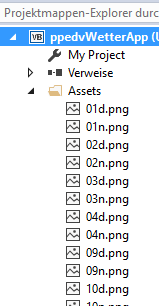
Return Math.Round(t).ToString

End Function

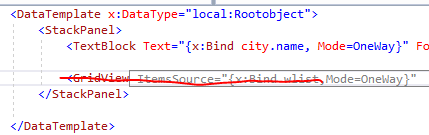
Die Bilder werden von der Website <https://openweathermap.org/weather-conditions> in das Verzeichnis Assets gespeichert. Rechts Click speichern unter



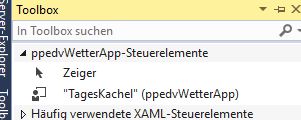
In Visual Studio Projekt Explorer Ansicht



In der Datei MainPage.XAML wird das Gridview entfernt und durch das folgende XAML Schnippsel ersetzt.



Ziehen Sie zumindest einmal aus der Toolbox das UserControl um sich die Tipparbeit zu sparen



<Grid VerticalAlignment="Stretch">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="100\*"/>

<ColumnDefinition Width="100\*"/>

<ColumnDefinition Width="100\*"/>

<ColumnDefinition Width="100\*"/>

<ColumnDefinition Width="100\*"/>

</Grid.ColumnDefinitions>

<local:TagesKachel Margin="20,0,20,0" Grid.Column="0" Wetter="{x:Bind wlist[0],Mode=OneWay}" />

<local:TagesKachel Margin="20,0,20,0" Grid.Column="1" Wetter="{x:Bind wlist[1],Mode=OneWay}" />

<local:TagesKachel Margin="20,0,20,0" Grid.Column="2" Wetter="{x:Bind wlist[2],Mode=OneWay}" />

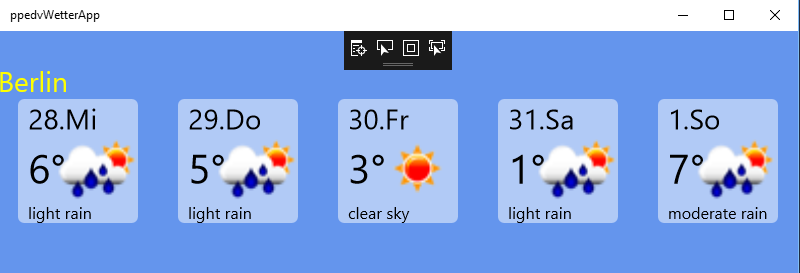
<local:TagesKachel Margin="20,0,20,0" Grid.Column="3" Wetter="{x:Bind wlist[3],Mode=OneWay}" />

<local:TagesKachel Margin="20,0,20,0" Grid.Column="4" Wetter="{x:Bind wlist[4],Mode=OneWay}" />

</Grid>

Letzendlich wird noch im MainPage Loaded Event der Code für den Fullscreen Mode eingefügt

ApplicationView.PreferredLaunchWindowingMode = ApplicationViewWindowingMode.FullScreen



# Lab 5 publizieren in den Store

Dauer: 25 Min

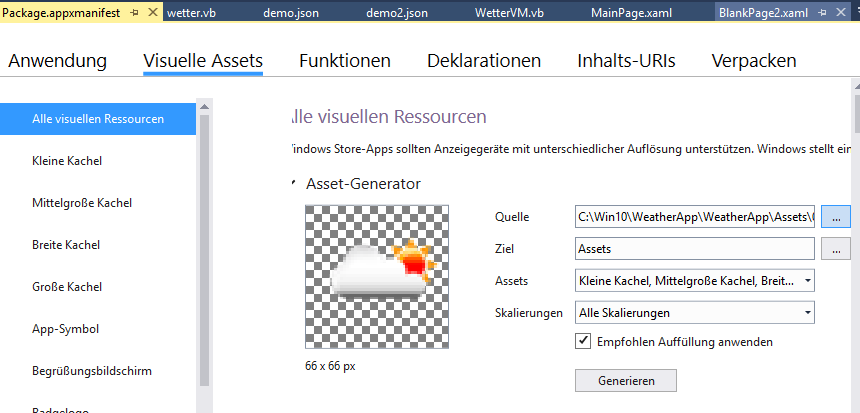
Store Ready

Im letzten Schritt wird die App in den Microsoft Store publiziert. Dazu benötigt man eine Microsoft ID

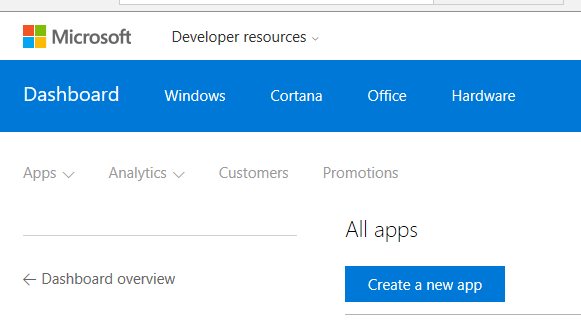
* *https://****signup****.live.com/*

Weiters einen Microsoft Developer Account

* <https://developer.microsoft.com/registration>

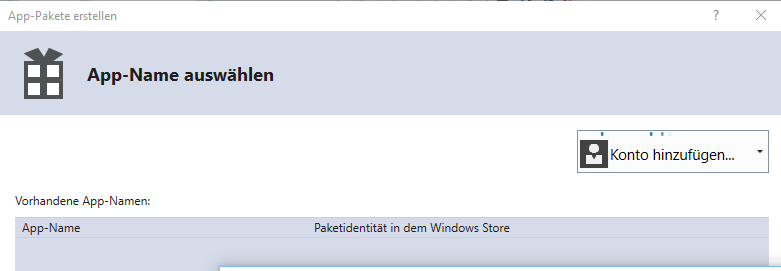


Visual Studio -Projekt -Store-App Packet hochladen



Reserve Product Name

Visual Studio Menü-Projekt -Store -App Paket erstellen



Es wird die eigene Microsoft ID (in der Regel die email Adresse) hinzugefügt. Dazu kommt ein extra Popup Dialog hoch. In diesem wird die Identität bestätgt in dem man das Passwort eingibt.

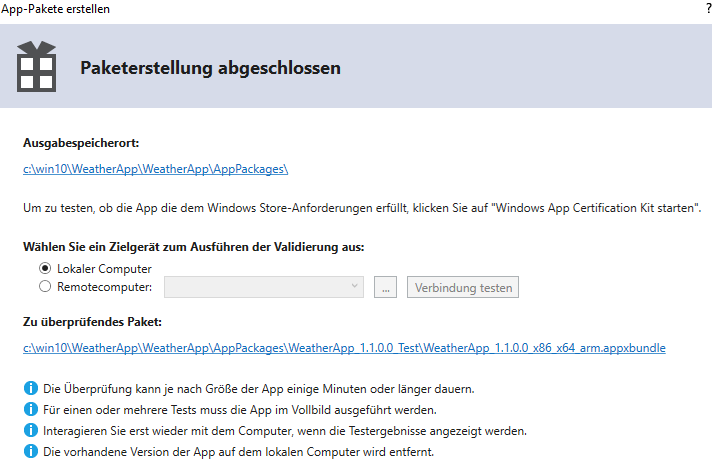
Der reservierte App Name erscheint in der Liste „vorhanden App Namen“ und wird ausgewäht.

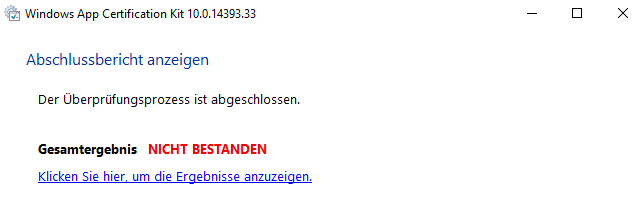
Der nächste Dialog wird betätigt. Aktivitt wird im Ausgabe Fenster angezeigt und ein animiertes Icon in der Visual Studio Status Leiste



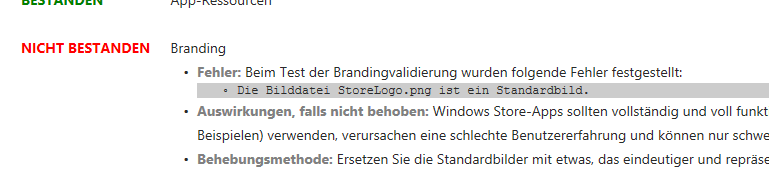
Dauer einige Minuten

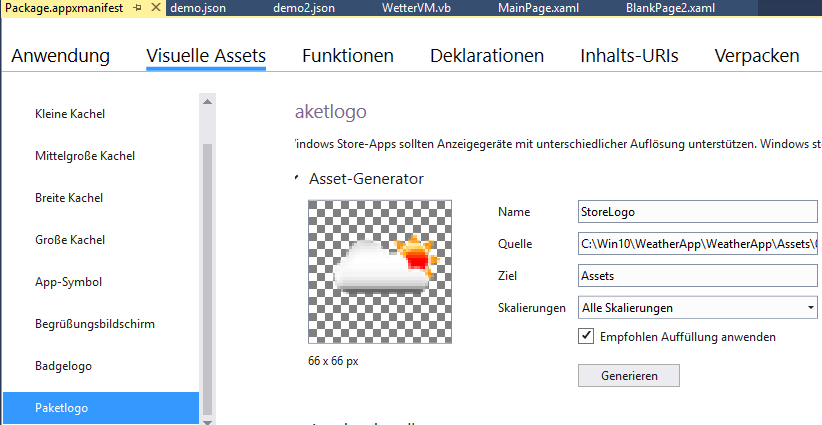
Nächster Schritt wird mit dem App Certificaion Kit ein Reihe von Vortests durchgeführt. Dies prüft Regeln um in den Store aufgenommen werden zu können





Im Browser öffnen



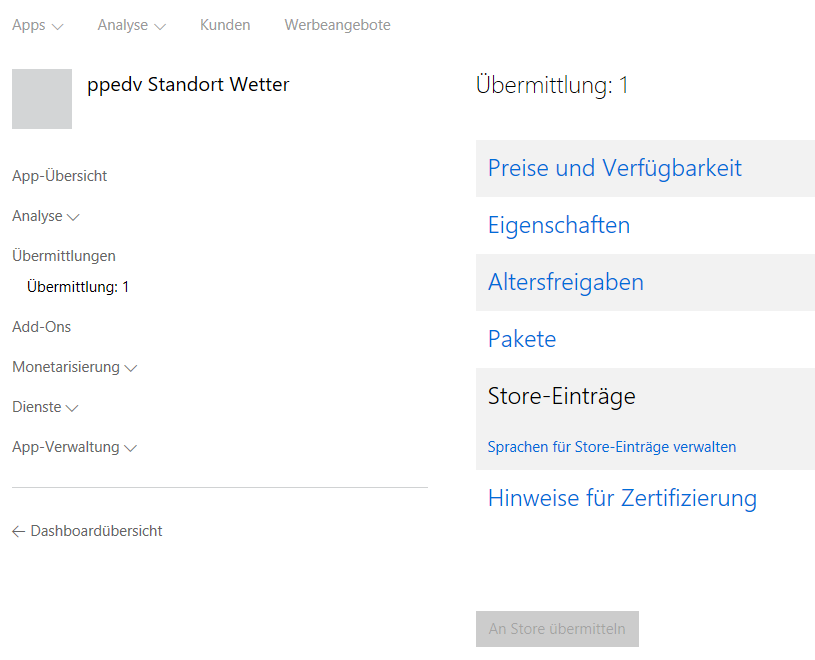


Nach diesem Fehler, storelogo.png aus Projekt löschen und die App Pakete neu erstellen (wie Schritt xx)

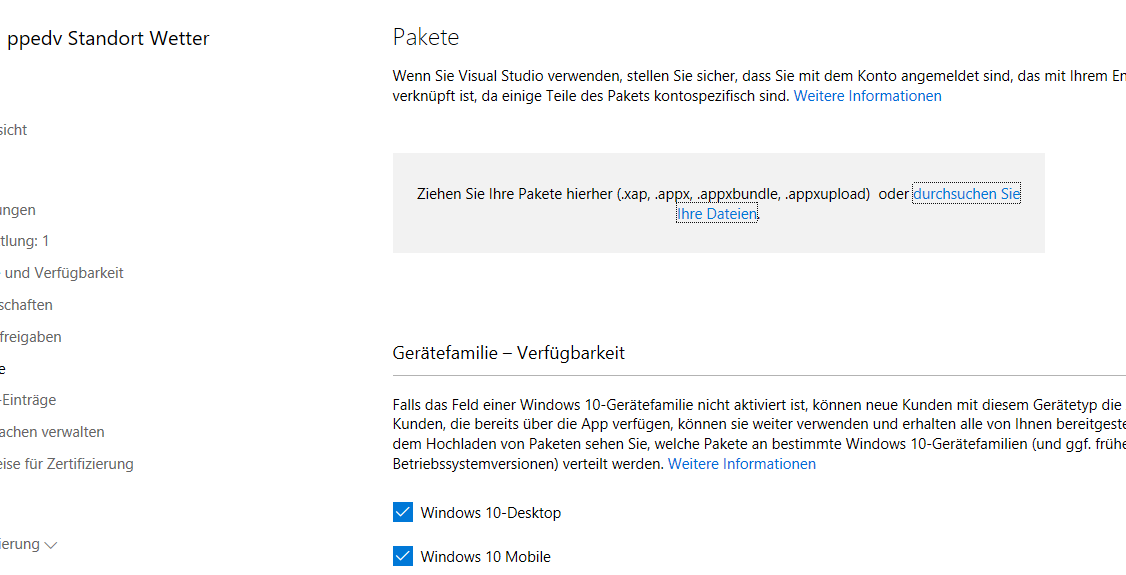
App Paket hochladen

Visual Studio Menü-Projekt -Store -App Paket hochladen

Führen Sie alle einzelschritte durch mit minimal erforderlichen Einstellungen (zb gratis)



Hochladen der appx Pakete

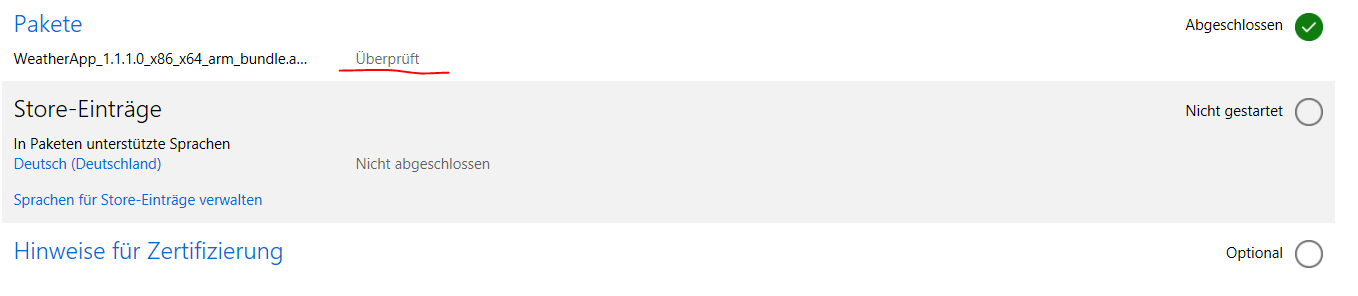


Pfad nutzen wie Schritt Paket erstellen definiert und appxupload Datei per darg und Drop

c:\win10\WeatherApp\WeatherApp\AppPackages\

WeatherApp\_1.1.1.0\_x86\_x64\_arm\_bundle.appxupload

Enthält alle Build Versionen wie ARM und co

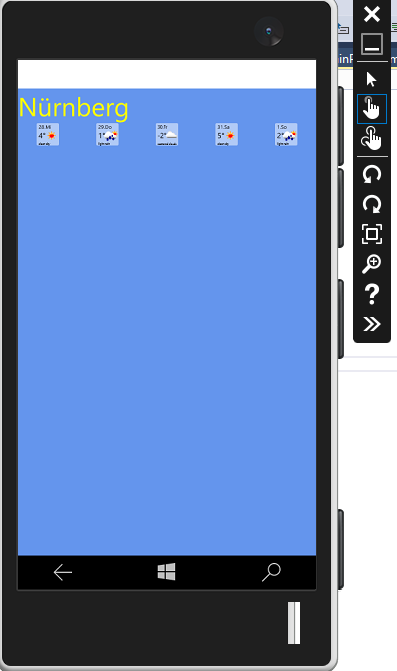


Mit dem Windows Werkzeug Snipping lassen sich Screenshots für den Store erstellen. Im Windows Startmenü Snipp eingeben und das Programm starten. Zum speichern Datei, Dateinamen eingeben und Drop Down PNG als Typ auswählen.

Optional Mobil Emulator starten

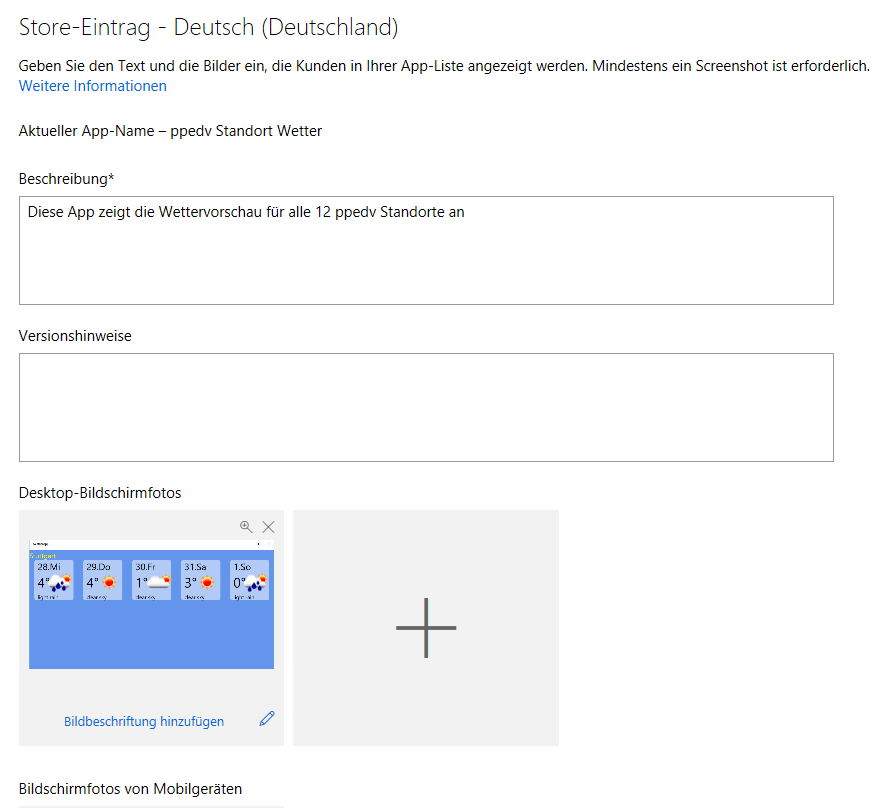
In der Regel nutzen die genutzen Anwendungen den Arbeitsspeicher weitesgehend aus. Um den Windows phone emulator nutzen zu können, empfiehlt sich ein Neustart des Computers. Sofort danach Visual Studio starten, das Projekt laden und im Emulator starten. Am besten den Emulator Prozess laufen lassen.

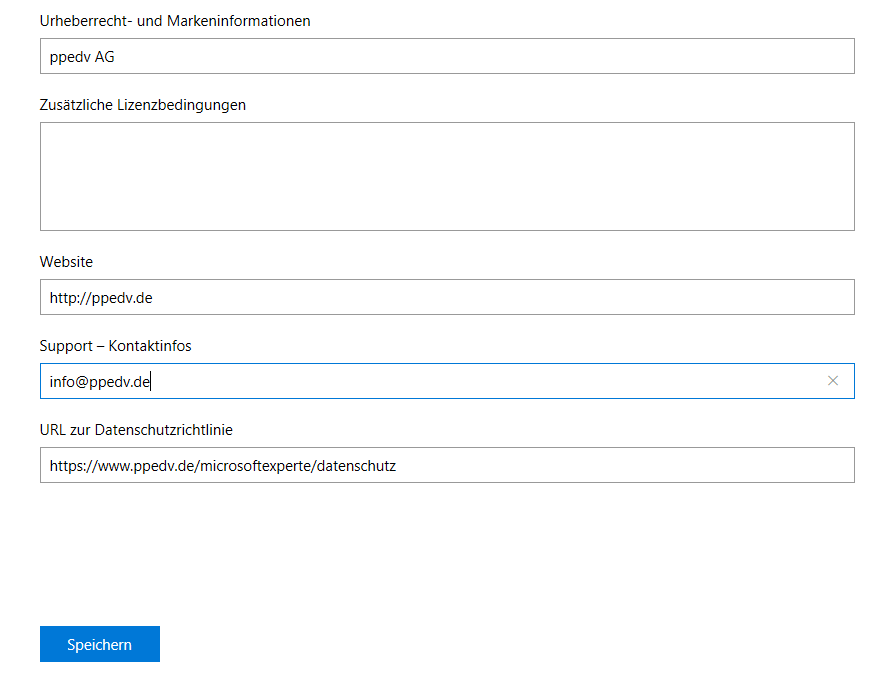




Einstellungen Sprachen

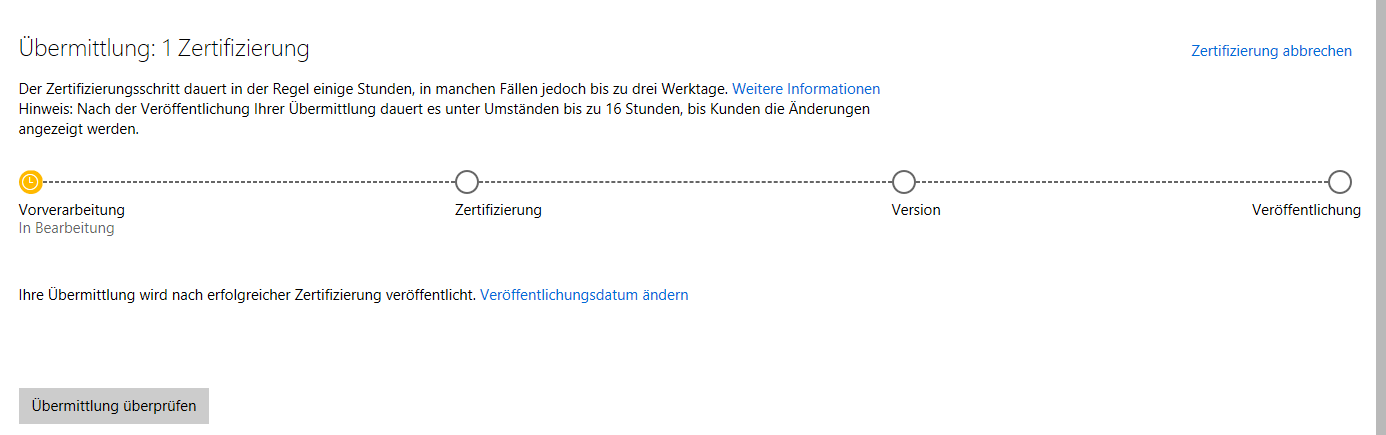
Beschreibender Text pro Sprache





Hinweise für Tester frei lassen

An Store Übermitteln



Ab hier dauert es Stunden oder Tage

Das komplette Projekt ist über Github abrufbar

<https://github.com/hannespreishuber/ppedvWetterApp>

© ppedv AG 2017