HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Festival Planer – Xamarin Projekt

Aktuelle Entwicklungen im Bereich Online-Medien

Kurs: Prof. Dr. Dirk Eisenbiegler

Teilnehmer: Schmidberger, Valentin

263249, MIB 6

Gliederung

** alle Screenshots ohne Quellenangabe, die in den folgenden Seiten vorkommen, sind aus meinem eigenen Projekt.

1. Allgemeine Anforderungen an das Projekt

Die allgemeinen Anforderungen an das Projekt waren wie folgt festgelegt:

Festival Planer als App:

- Die App soll eine Übersicht über eingetragene Festivals beinhalten, jedes Festival besitzt eigene Informationen über
    - Titel, Ticket Anzahl, Preis, Zeitspanne, Beschreibung und Standort
- Die App soll es ermöglichen selbst neue Festivals mit den oben genannten Informationen einzutragen
- Die App soll über den geräteinternen GPS Sensor den Standort abfragen
- Die App soll den Nutzer über Festivals in Form einer Push-Nachricht informieren können
- Die App soll einen Eintrag im Kalender hinzufügen, wenn der Nutzer auf die Push-Nachricht eingeht

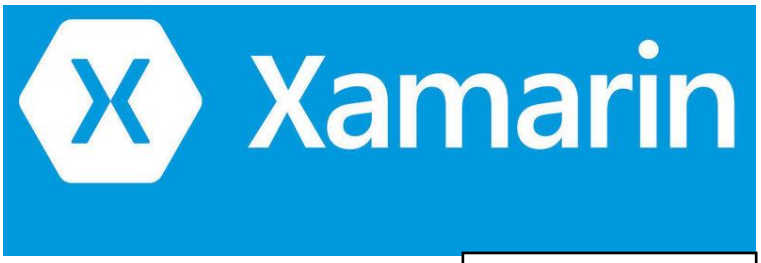
Technische Besonderheiten des Festival Planers:

- Eine native App, die sowohl unter Android als auch iOS lauffähig gemacht werden soll
- Ein Framework welches ermöglicht, die App in einer Entwicklungsumgebung einmal aufzusetzen und beide Betriebssysteme über ein gemeinsames Layer abdeckt
- Die Nutzung einer gemeinsamen Datenbank und eine Bearbeitung persistenter Daten über genannte Datenbank ermöglicht
- Die Nutzung des internen GPS Sensors des Handys
- Die Nutzung der Push-Nachrichten Technologie
- Die Nutzung des internen Kalenders

1.1. Denkbare weitere Features:

- Datenbank aufbauen für User und deren gekauften Festivals / Login Funktion
- Ansicht der App für Nutzer / Ansicht der App für Administratoren
- Abgleich von möglichen Festivals und User Datenbank
    - Sodass nur dann eine Push-Nachricht versendet wird, wenn der Nutzer das Festival noch nicht in seiner eigenen Datenbank besitzt.
- Festivals interaktiver gestalten
- Funktion, dass Nutzer bereits gekaufte Festivals auf der Festival Ansicht wieder verkaufen können
- Mehr visuellen Content in der detaillieren Festival Ansicht
- Zugang zu den Datenbanken verschlüsseln
- Allgemein visuell ansprechender gestalten

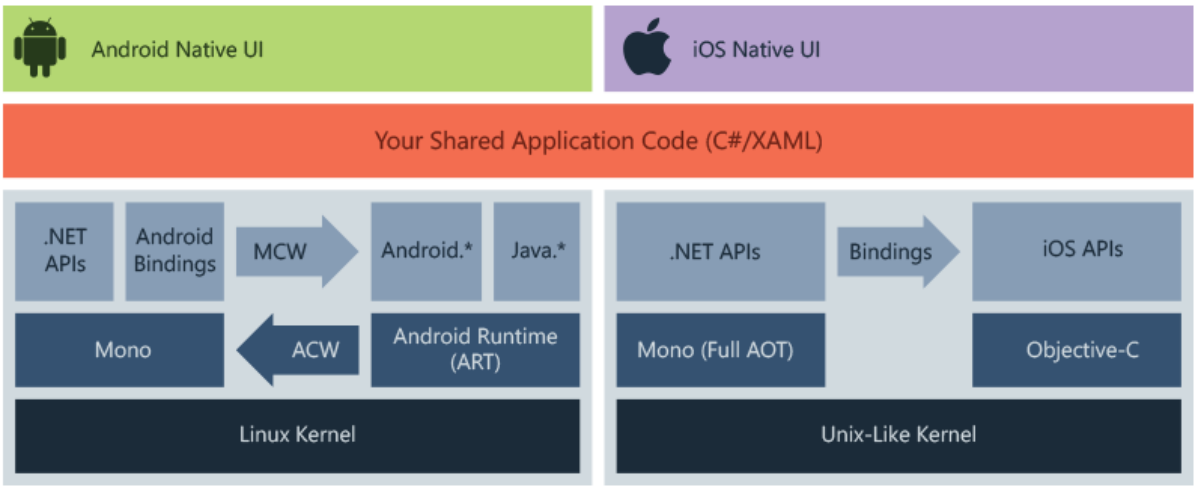
2. Was ist Xamarin



Quelle: Microsoft

Xamarin ist ein Framework, welches im Jahr 2011 von der Firma Mono entwickelt wurde. Mono (Software) ist eine open source Alternative zu Microsofts .NET Framework. Sie ermöglicht die Entwicklung von plattformunabhängiger Software auf den Standards der Common Language Infrastructure und der Programmiersprache C#. Anfang 2013 stellte die Firma Xamarin.Studio vor, welches ermöglichte iOS, Android und OS-X Entwicklungstools in einem Programm zu erstellen. Zusätzlich wurde das Framework in Visual Studio und in das .NET Framework von Microsoft eingebettet. Im Jahr 2016 wurde Xamarin von Microsoft übernommen.

2.1. Funktionsweise von Xamarin



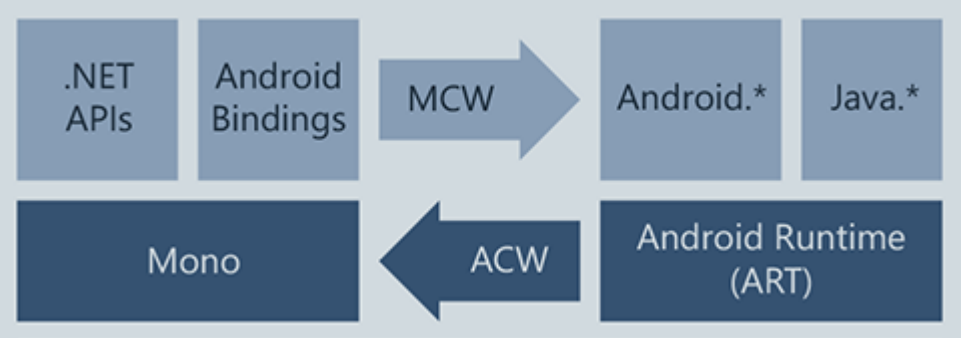
Quelle: Microsoft

In der obigen Abbildung wird eine vereinfachte Funktionsweise von Xamarin dargestellt: Durch das Framework wird es ermöglicht native Benutzeroberflächen für jede Plattform zu erstellen und gleichzeitig auch BackEnd Code in C# plattformübergreifend freizugeben. Mit diesem Ansatz kann man bis zu 80% des Anwendungscodes mithilfe von Xamarin freigebeben werden. (Quelle Microsoft)

2.2. Plattformen

Xamarin unterteilt deren plattformspezifische Bereiche in Unterkategorien, welche immer mit Xamarin.“x“ unterteilt sind. Im folgenden werde ich auf die für mich relevantesten Unterkategorien eingehen.

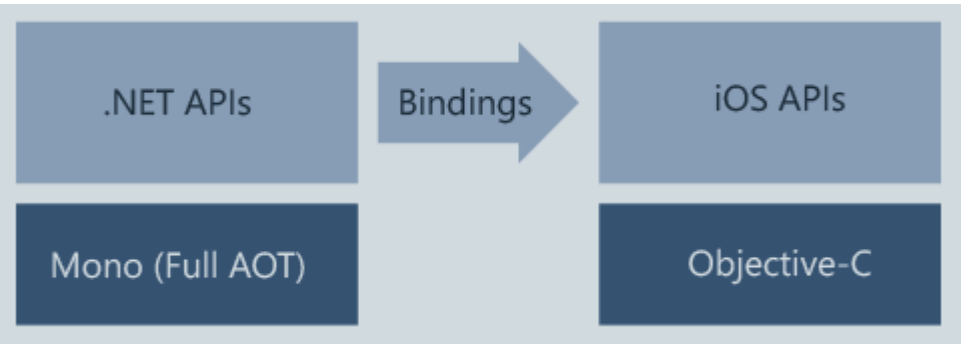
2.3. Xamarin.Android



Quelle: Microsoft

Die Anwendungen in Xamarin werden von C# Code zu einer Zwischensprache names Intermdiate Language umkompoiliert, die dann während der Anwendung Just-In-Time startet und in eine native Assembly kompiliert wird. Xamarin Android Anwendungen werden nach wie vor in der Mono-Umgebung ausgeführt, sie bieten allerdings Android und Java Namespaces für das .NET Framework. Die Mono-Ausführungsumgebung führt Aufrufe in diese Namespaces über verwaltete Callable Wrapper (Managed Callable Wrappers, MCWs) aus und stellt ART Callable Wrappers für Android (Android Callable Wrappers, ACW) zur Verfügung, sodass in beiden Umgebungen Code voneinander aufgerufen werden kann. (Quelle Microsoft)

2.4. Xamarin.iOS



Quelle: Microsoft

Bei Xamarin iOS Applikationen wird der Vorgang anders umgesetzt: C# Code wird vollständig in Apples nativen Assemblycode Ahead-of-time (AOT) kompiliert. Xamarin verwendet Selektoren, um Objective-C für verwalteten C#-Code verfügbar zu machen und Registrars, um verwalteten C#-Code für Objective-C zu verteilen. (Quelle Microsoft)

2.5. Community

Eine direkte Community Plattform gibt es bei Xamarin leider nicht. Allerdings stellt Microsoft eine recht ausführliche Dokumentation zur Verfügung, über die man oft schon den Kern des Anwendungsfalls versteht und darauf aufbauen kann. Auf der anderen Seite ist Xamarin durchaus im Netz vertreten, man findet viele Lösungsansätze auf der Seite Stackoverflow oder Ähnlichem. Des Weiteren findet man auch viele Tutorials über YouTube. Im Großen und Ganzen kann ich aus meiner Zeit mit Xamarin sagen, dass man zu jedem Problem das aufkommt, immer eine passende Erklärung gefunden hat auch wenn man dafür manchmal etwas tiefer im Netz suchen musste.

3. Was ist Xamarin.Forms

Ohne Xamarin.Forms gab es zuvor nur die Unterformen Xamarin.iOS, Xamarin.Androird etc. . Durch Xamarin.Forms wurden diese Unterformen vereint und mithilfe von Xamarin.Forms können Entwickler nun Xamarin.iOS-, Xamarin.Android- und Windows-Anwendungen aus einer einzigen freigegebenen CodeBase erstellen. Zur Laufzeit verfügt Xamarin.Forms über Plattformrenderer, um die plattformübergreifenden Benutzeroberflächenelemente in native Steuerelemente für Xamarin.Android, Xamarin.iOS und UWP (Windows common platform) zu konvertieren. So können Entwickler natives Aussehen, native Servicequalität und native Leistung erreichen, ohne dabei für jede Plattform extra Code schreiben zu müssen. Xamarin.Forms-Anwendungen bestehen im Normalfall aus einer .NET-Bibliothek und einzelnen Plattformprojekten. Die freigegebene Bibliothek enthält die XAML- und C#-Ansichten.

Zusätzlich verfügt Xamarin.Forms über ein großes Ökosystem von Bibliotheken, die der Anwendung verschiedene Funktionen hinzufügen: Bei Xamarin.Essentials handelt es sich zum Beispiel um eine Bibliothek aus dem Hause Microsoft, die plattformübergreifende APIs für native Gerätefeatures bietet. Auch ich habe innerhalb meines Projekts immer wieder auf die Xamarin.Essentials Bibliothek zugegriffen, um

mir Teile des Programmierens deutlich zu erleichtern. Zu den von Xamarin.Essentials zur Verfügung gestellten Hilfsprogrammen gehören die folgenden:

- Geräteinformationen
- Dateisystem
- Beschleunigungsmesser
- Wählhilfe
- Text-zu-Sprache
- Bildschirmsperre

Quelle Microsoft

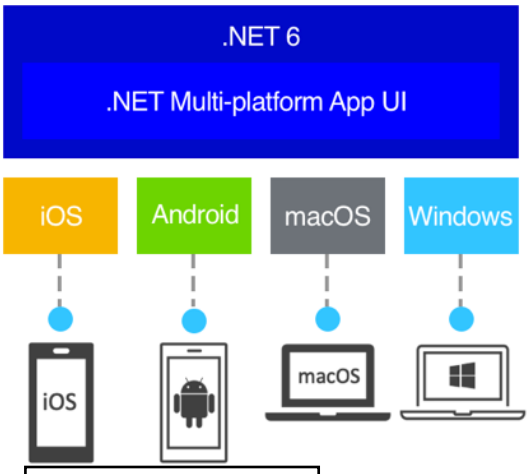
3.1. NuGet - Package Manager

Bei dem NuGet - Packet Manager handelt es sich um ein opensource Pakete Manager für das .NET Ökosystem. Er wurde dafür entwickelt Entwicklern wieder verwendbaren Code zugänglich zu machen. NuGet wird verwendet, um automatisch Dateien und Referenzen zu dem jeweiligen Visual Studio-Projekten hinzuzufügen. Einfach gesagt: ein NuGet-Paket ist eine einzelne ZIP-Datei, die kompilierten Code, andere Dateien im Zusammenhang mit diesem Code und ein beschreibendes Manifest enthält. Entwickler, die Code teilen wollen, erstellen und veröffentlichen Pakete auf einem öffentlichen oder privaten Host. Paketverbraucher erhalten diese Pakete über entsprechende Hosts, fügen diese ihren Projekten hinzu und rufen dann die Funktionalität eines Pakets in ihrem Projektcode auf.

Quelle Microsoft

4. Was ist .NET Maui

.NET Maui ist der Nachfolger von Xamarin (aktuell noch in der Preview Version): Mit .NET MAUI kann man alle Funktionen nutzen die Xamarin.Forms zuvor schon geboten hat, mit dem Zusatz, dass der Entwickler nun aus einer Code Basis heraus auch Apps für macOS und Windows entwickeln kann.

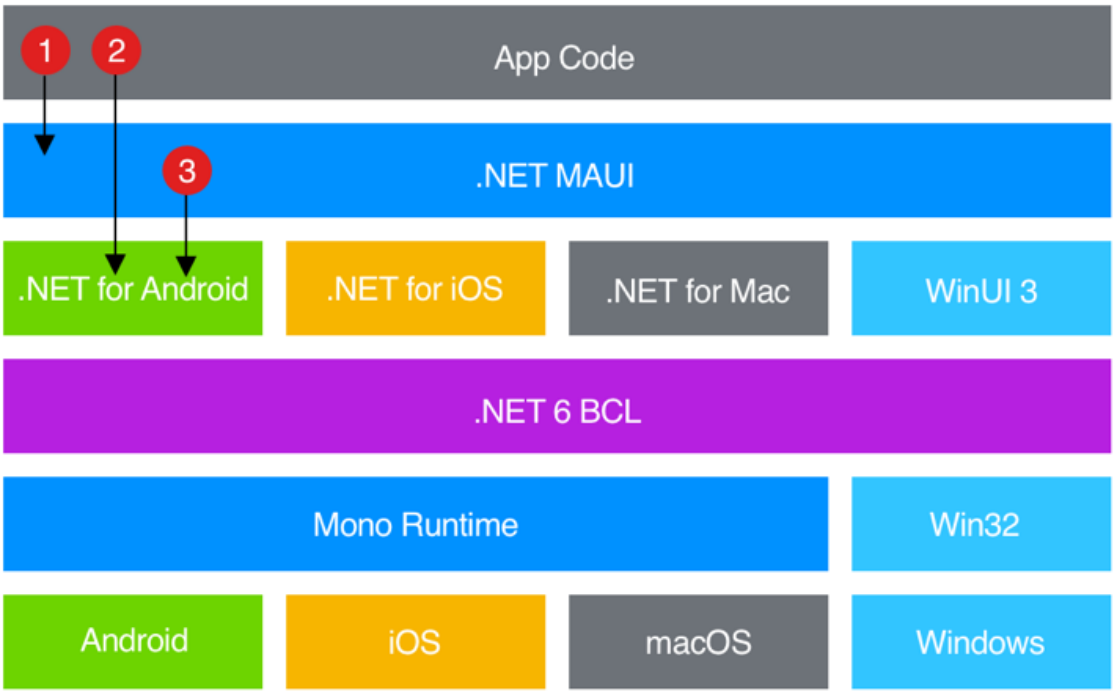


Quelle: Microsoft

Wenn man zuvor mit Xamarin.Forms zum Erstellen plattformübergreifender Benutzeroberflächen gearbeitet hat, erkennt man viele Gemeinsamkeiten. Allerdings gibt es auch einige Unterschiede. Mit einem der wichtigsten Ziele von .NET MAUI kann man jetzt noch mehr App-Logik- und UI-Layouts in einer einzigen Codebasis implementieren.

Auch die .NET Version wurde von .NET 5 Standard auf .NET 6 aktualisiert. .NET 6 bietet eine Reihe plattformspezifischer Frameworks zum Erstellen von Apps: .NET für Android, .NET für iOS, .NET für macOS und Windows UI 3 (WinUI 3) Bibliothek. Diese Frameworks haben Zugriff auf die gleiche .NET 6 Base Class Library (BCL). Diese Bibliothek abstrahiert die Details der zugrunde liegenden Plattform weg von dem eigen geschriebenen Code. Die Base Class Library hängt von der .NET-Laufzeit ab, um die Ausführungsumgebung für Ihren Code bereitzustellen.

.NET MAUI bietet ein einzelnes Framework zum Erstellen der UIs für mobile und Desktop-Apps. Das folgende Diagramm zeigt eine abstrahierte Ansicht der Architektur einer .NET MAUI-App:



Quelle: Microsoft

.NET MAUI-Apps können auf PC oder Mac geschrieben und in systemeigene App-Pakete kompiliert werden:

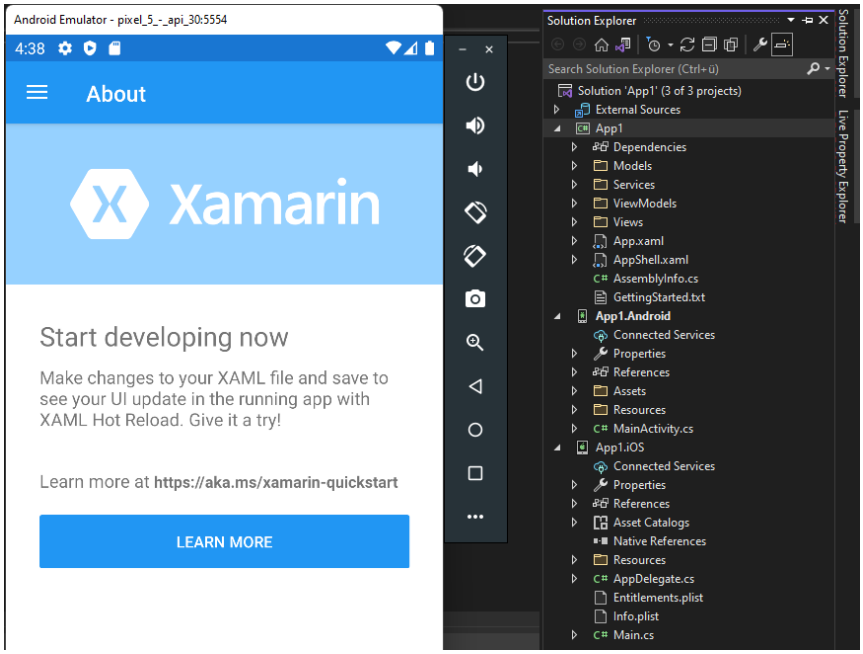
- Für die Anwendungsfälle iOS und Android geht .NET Maui genauso vor wie schon oben bei Xamarin.Forms erläutert.
- Für macOS Apps, die mit .NET MAUI erstellt wurden, wird Mac Catalyst verwendet, eine Lösung von Apple, die mit zusätzlichen AppKit- und Plattform-APIs erweitert wurde, die erforderlich waren.
- Windows Apps, die mithilfe von .NET MAUI erstellt wurden, verwenden die Windows UI 3 (WinUI 3) Bibliothek, um systemeigene Apps zu erstellen, die auf dem Windows Desktop lauffähig sein sollen.

Quelle Microsoft

5. Start des Projekts

5.1. Aufsetzen der Umgebung und erstes Template

Der erste Schritt, den ich anfangs unternahm, war das Herunterladen der Visual Studio Software, da ich zuvor primär in Visual Studio Code gecoded hatte. Das Installieren von Visual Studio und das Hinzufügen von Xamarin.Forms lief reibungslos ab. Alle Abhängigkeiten und Pakete die man für Xamarin.Forms benötigt wurden automatisch



von Visual Studio heruntergeladen. Sobald alles fertig runtergeladen war, startete ich mit dem vorgeschlagenem Template von Microsoft.

5.2. Testversuche Xamarin.Forms | .NET Maui

Da ich im Anfangsstadium noch nicht wirklich Bescheid wusste was genau Xamarin.Forms ist und was es mit dem Nachfolger .NET Maui auf sich hat, lud ich zunächst auch .NET Maui in meine Visual Studio Entwicklungsumgebung. Dementsprechend untersuchte ich in den folgenden Tagen auch das Template von .NET Maui, welches deutlich weniger ausführlich gestaltet war, als das von Xamarin.Forms. Zusätzlich fiel mir immer mehr auf, dass es zu der neuen Plattform

.NET Maui noch nahezu keine Dokumentation oder User Beiträge auf Seiten wie StackOverflow gab.

5.3. Entscheidung für Xamarin.Forms

So entschied ich mich für die Platform Xamarin.Forms, vor allem anhand dieser zwei Gründe:

1. .NET Maui befindet sich nach wie vor noch in der Test Phase und es lauern einige Bugs auf den Entwickler, welche noch nicht gefixt wurden
2. Die Dokumentation und Nutzer Beiträge zu .NET Maui sind noch nahezu nicht vorhanden, die Arbeit würde also oft auf „Versuchen bis es funktioniert“ rauslaufen, was oft sehr viel Zeit in Anspruch nehmen kann

6. Allgemeiner Überblick

6.1. Dive into XAML

XAML ermöglicht Entwicklern das Bauen von Benutzeroberflächen in Xamarin.Forms mithilfe von Markup anstelle von Code. XAML ist nicht zwingend in Xamarin.Forms erforderlich, aber es ist oft deutlich einfacher umzusetzen und visuell kohärenter als gleichwertiger C# Code. XAML eignet sich gut für die Verwendung mit der beliebten MVVM(Model-View-ViewModel)-Anwendungsarchitektur: XAML definiert die Ansicht, die über XAML-basierte Datenbindungen mit der hinterlegten ViewModel-C# Code verknüpft ist. Innerhalb einer XAML-Datei kann man Benutzeroberflächen mit allen Xamarin.Forms Ansichten, Layouts und Seiten sowie benutzerdefinierten Klassen definieren. Hier ein beispielhafter Code einer XAML Datei aus meinem Projekt:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="FestivalPlaner.Views.NewItemPage"
             Shell.PresentationMode="ModalAnimated"
             Title="New Festival"
             xmlns:ios="clr-namespace:Xamarin.Forms.PlatformConfiguration.iOSSpecific;assembly=Xamarin.Forms.Core"
             ios:Page.UseSafeArea="true">
    <ContentPage.Content>
        <StackLayout Spacing="3" Padding="15">
            <Label Text="Enter new Festival" FontSize="Title"/>
            <Label Text="Name" FontSize="Medium" />
            <Entry Text="{Binding Name, Mode=TwoWay}" FontSize="Medium" />
            <Label Text="Start Date" FontSize="Medium" />
            <Editor Text="{Binding StartDate, Mode=TwoWay}" AutoSize="TextChanges" FontSize="Medium" Margin="0" />
            <Label Text="End Date" FontSize="Medium" />
            <Editor Text="{Binding EndDate, Mode=TwoWay}" AutoSize="TextChanges" FontSize="Medium" Margin="0" />
            <Label Text="Place" FontSize="Medium" />
            <Editor Text="{Binding Place, Mode=TwoWay}" x:Name="Place" AutoSize="TextChanges" FontSize="Medium" Margin="0" />
            <Button x:Name="LocationButton" Text="Add Location" Clicked="Button_Clicked"></Button>
            <Label Text="Price" FontSize="Medium" />
            <Editor Text="{Binding Price, Mode=TwoWay}" AutoSize="TextChanges" FontSize="Medium" Margin="0" />
            <Label Text="Ticket Count" FontSize="Medium" />
            <Editor Text="{Binding TicketCountAvailable, Mode=TwoWay}" AutoSize="TextChanges" FontSize="Medium" Margin="0" />
            <StackLayout Orientation="Horizontal">
                <Button Text="Cancel" Command="{Binding Cancel}" HorizontalOptions="FillAndExpand"></Button>
                <Button Text="Save" Command="{Binding Save}" HorizontalOptions="FillAndExpand"></Button>
            </StackLayout>
        </StackLayout>
    </ContentPage.Content>

</ContentPage>
```

Quelle Microsoft

6.2. Dive into C#

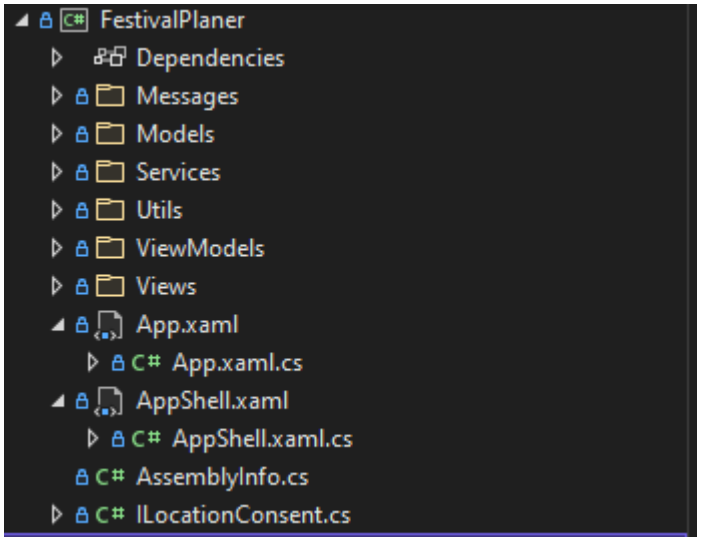
Quelle: Microsoft

C# (Aussprache „C Sharp“) ist eine moderne, objektorientierte und typsichere Programmiersprache. C# ermöglicht das Erstellen performanter Applikation, die in .NET ausgeführt werden können. C# hat seine Wurzeln in der C-Sprache und ist Programmierern, die mit C, C++, Java und JavaScript arbeiten, schnell vertraut. C# bietet zudem Sprachkonstrukte in der Softwarekomponenten erstellt und verwendet werden können. Seit Veröffentlichung der Sprache wurden C# Features hinzugefügt, um neue Workloads und Methoden zur Gestaltung von Software zu unterstützen. Im Kern ist C# aber eine objektorientierte Sprache. Man definiert Typen und deren Verhalten. In der seitlichen Abbildung kann man ein beispielhaften C# Code aus meiner Xamarin.Forms Anwendung studieren. Unter anderem ist ein Button Event aus der darüberliegende XAML Datei zu untersuchen.

```
namespace FestivalPlaner.Views
{
    9 references
    public partial class NewItemPage : ContentPage
    {

        0 references
        public NewItemPage()
        {
            InitializeComponent();
            BindingContext = new NewItemViewModel();
        }

        0 references
        private void Button_Clicked(object sender, EventArgs e)
        {
            Navigation.ShowPopup(new MapPupUp(this)
            {
                IsLightDismissEnabled = false,
                BindingContext = this

            });
        }
        1 reference
        public void OnSafe(string place)
        {
            Place.Text = place;
            LocationButton.IsEnabled = false;
            LocationButton.IsVisible = false;
        }
    }
}
```

Quelle Microsoft

6.3. General Solution



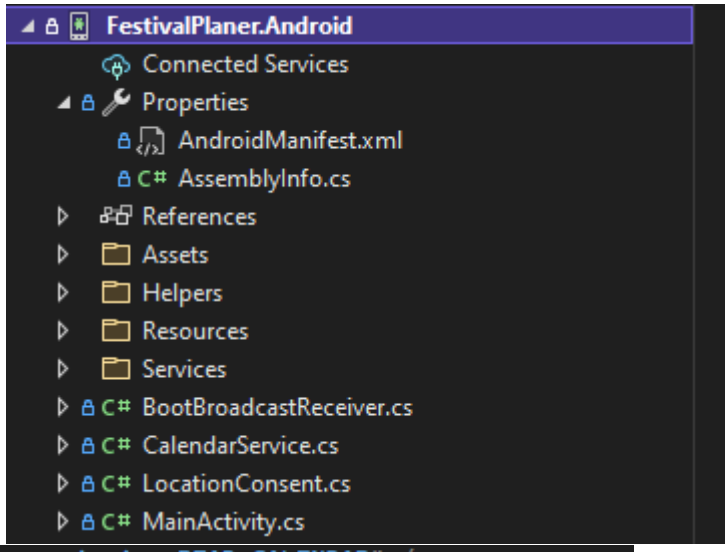
Xamarin.Forms ist in 3 Projekt Ordner/Bereiche aufgeteilt. In den folgenden 3 Abschnitten werde ich jede dieser Projekt Unterteilungen genauer erläutern. Hier soll es zuerst um die „General Solution“ gehen. Innerhalb der „Gerneral Solution“ werden alle XAML / C# Dateien/Skripte angelegt und eine sinnvolle Struktur erstellt. In der allgemeinen Lösung findet man alle UI und Geschäftslogik die für beide Betriebssysteme gleichermaßen funktionieren, in meinem Fall waren das folgende Bereiche:

- Alle Ansichten der App
- Flyout Navigation
- Anbindung an die Datenbank
- Generelle GPS Lösung von Xamarin.Essentials
- Verschiedene Klassen-Strukturen wie *FestivalModel* oder *FestivalMessage*
- Message-Klassen für das Messaging Center

In der obigen Abbildung ist auch schon zu erkennen, dass die Dateien XAML und C# oft im Paar gebaut werden: XAML kann man so verstehen wie HTML, also das Frontend: man kann dort visuelle Elemente hinzufügen wie Buttons, Listen, Textfelder etc. . Die Besonderheit ist aber, dass die XAML Datei auf eine hinterlegte C# Datei verweist. Innerhalb der C# Datei kann man auf alle Elemente, Attribute und sonstige XAML Elemente zugreifen und in Echtzeit anpassen, verändern und auslesen. Sehr ähnlich wie das Zusammenspiel zwischen HTML und Javascript.

6.4. Android Solution

In der Android Solution („FestivalPlaner.Android“) geht es in erster Linie darum, die Berechtigungen für das Android Gerät in der Android Manifest XML Datei festzulegen.




```
<uses-permission android:name="android.permission.READ_CALENDAR" />
<uses-permission android:name="android.permission.WRITE_CALENDAR" />
<uses-permission android:name="android.permission.READ_CALENDAR" />
<uses-permission android:name="android.permission.WRITE_CALENDAR" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
```

Der zweite größere Teil betrifft das Aufrufen von nativen Gerätefeatures. Die Hürde, die dabei überwinden werden muss, ist folgende: Es besteht nur ein sehr kleiner Zusammenhang zwischen der „General Solution“ und der „Android Solution“. Ungefähr neunzig Prozent der Geschäftslogik und Interaktion innerhalb der App befindet sich in der „General Solution“. Nun muss es einen Weg geben, um von der „General Solution“, bestimmte native Features in der „Android Solution“ auszulösen. Die Lösung hierfür war für mich das „Messaging Center“. Damit wird es dem Programmierer ermöglicht

```
MessagingCenter.Subscribe<StopServiceMessage>(this, "ServiceStopped", message =>
{
    if (IsServiceRunning(typeof(AndroidLocationService)))
        StopService(serviceIntent);
});
MessagingCenter.Subscribe<CalendarMessage>(this, "CreateCalendar", message =>
{
    new CalendarService(message.startTime, message.endTime, message.title, message.description);
});
MessagingCenter.Subscribe<DateCheckerMessage>(this, "DateCheckerMessage", message =>
{
    Task.Run(async () =>
    {
        message.isFree =  await CalendarService.CheckIsDateIsAvailable(message.startTime, message.endTime);
    });
});
  });
```

gewisse „Events“ festzulegen, die Plattformübergreifend gelesen werden können. In der obigen Abbildung ist zu sehen, wie ich gewisse Events in der „Android Solution“ abonniere.

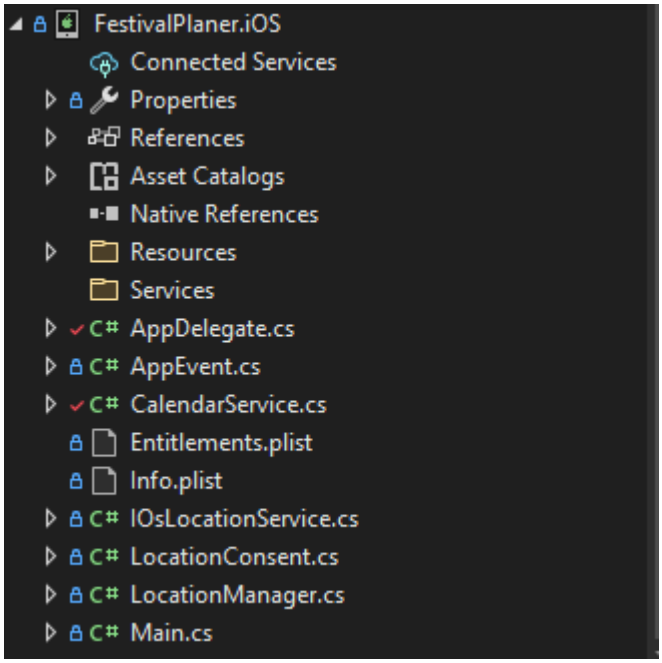
Innerhalb der „General Solution“ kann ich dann wiederum ein solch festgelegtes Event abfeuern. In dem obigen Fall ist es das „CreateCalendar“-Event, welches ich in der Android Solution abonniert habe. Sobald dieses Event gesendet wird, soll in diesem Zusammenhang eine neue Klasse vom Typ „CalendarService“ innerhalb der Android Solution aufgerufen werden. Zusätzlich wurde dem Event eine Klasse vom Typ „CalendarMessage“ übergeben, welche die wichtigen Daten zum Festival beinhaltet.

6.5. iOS Solution

In der iOS Solution („FestivalPlaner.iOS) geht es in erster Linie darum, die Berechtigungen für das iOS Gerät in der Info PLIST Datei festzulegen.

CalendarService.cs | Main.cs | × Info.plist

Property	Type	Value
Targeted device family	Array	(2 items)
	Number	1
	Number	2
Add new entry		
Supported interface orientations	Array	(3 items)
	String	Portrait (bottom button)
	String	Landscape (left button)
	String	Landscape (right button)
Add new entry		
Supported interface orientations (iPad)	Array	(4 items)
	String	Portrait (bottom button)
	String	Portrait (top button)
	String	Landscape (left button)
	String	Landscape (right button)
Add new entry		
Minimum system version	String	8.0
Bundle display name	String	FestivalPlaner
Bundle identifier	String	com.company.Festival
Bundle version	String	1
Launch screen interface file base name	String	LaunchScreen
Bundle name	String	FestivalPlaner
XSAppIconAssets	String	Assets.xcassets/AppIcon.appiconset
Required background modes	Array	(3 items)
	String	App registers for location updates
	String	App downloads content in response to push notifications
	String	App processes data in the background
Add new entry		
Privacy - Location Always and When In U...	String	We need your location to show you near festivals
Privacy - Location Always Usage Descrip...	String	We need your location to show you near festivals
Privacy - Location Usage Description	String	We need your location to show you near festivals
Privacy - Location When In Use Usage D...	String	We need your location to show you near festivals
Main storyboard file base name	String	LaunchScreen
Bundle versions string (short)	String	1.0
Privacy - Calendars Usage Description	String	We need access to your Calender.
Add new entry		



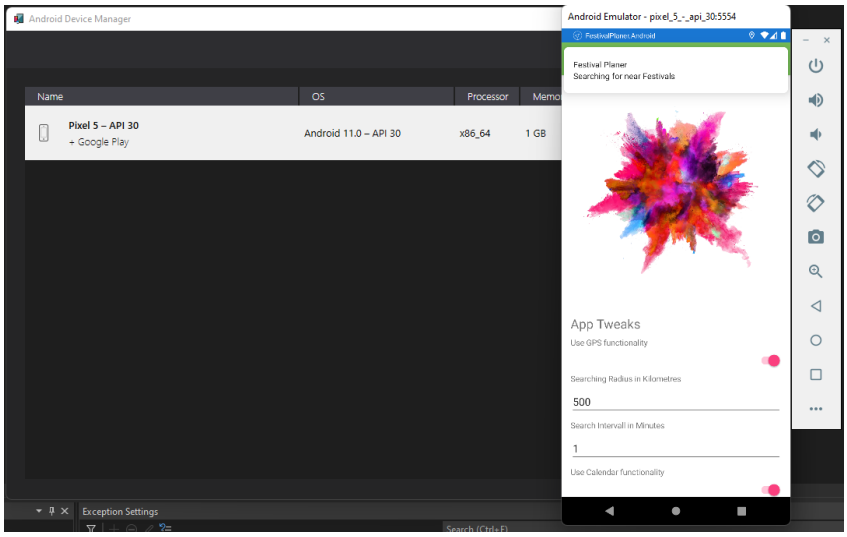
Auch in der iOS Solution wurde das „*Messaging Center*“ genutzt, um zum richtigen Zeitpunkt über die General Solution native, geräteinterne Ressourcen innerhalb der iOS Solution zu nutzen, wie zum Beispiel das Background Tasking und den Apple Kalender.

7. Projektausführung

Bei der Projektausführung ging es für mich einmal um die App innerhalb Android und auf der anderen Seite die App innerhalb iOS. Hier muss man vorab sagen, dass die Ausführung unter Android deutlich einfacher gestaltet ist, als die Ausführung unter iOS.

7.1. Android

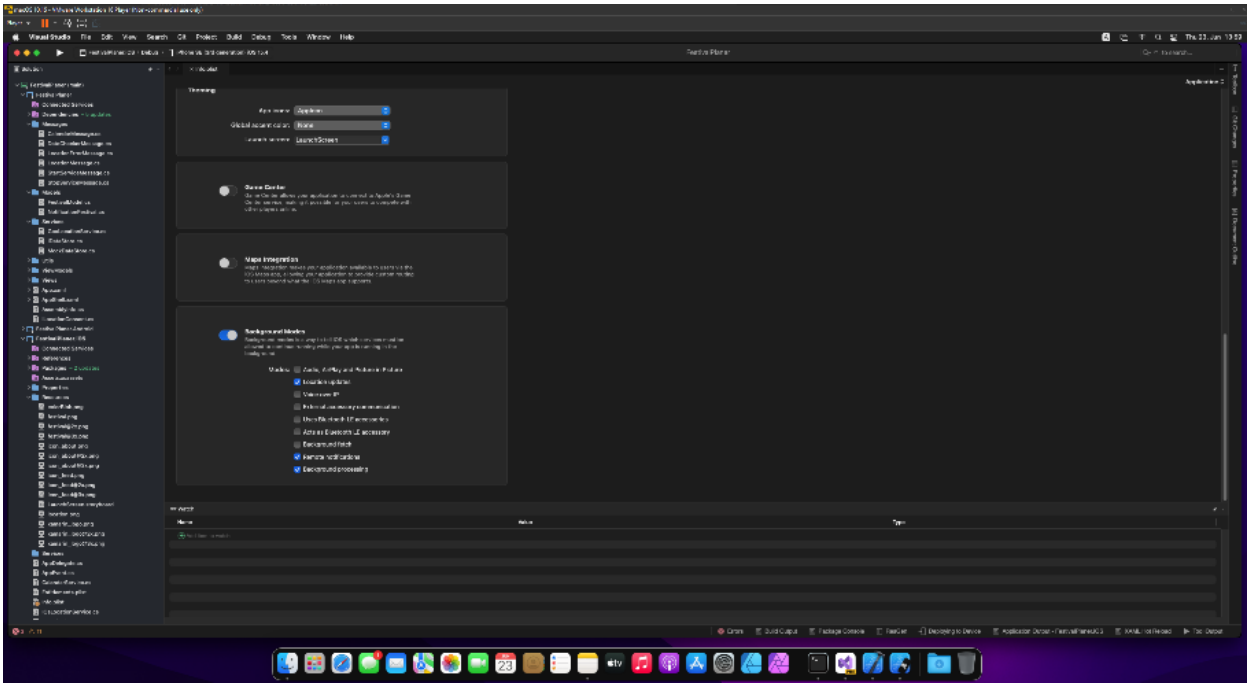
Über ein Windows Rechner innerhalb Visual Studio ist die Ausführung der App unter Android ein leichtes Spiel. Es gibt verschiedene Möglichkeiten die App unter Android



zu exekutieren. Der erste und einfachste Weg ist es, die App über den eingebauten Android Emulator zu starten. Das war auch der erste Ansatz für mich, den ersten Prototyp zu

untersuchen und zu testen. Der zweite Weg ist es, bei einem Android Handy den Developer Modus zu aktivieren, dann kann man das Gerät wie den Emulator problemlos mit der App bespielen. Der letzte und dritte Weg wäre es die App zu einer APK zu konvertieren und dann auf das Endgerät zu installieren.

7.2. iOS



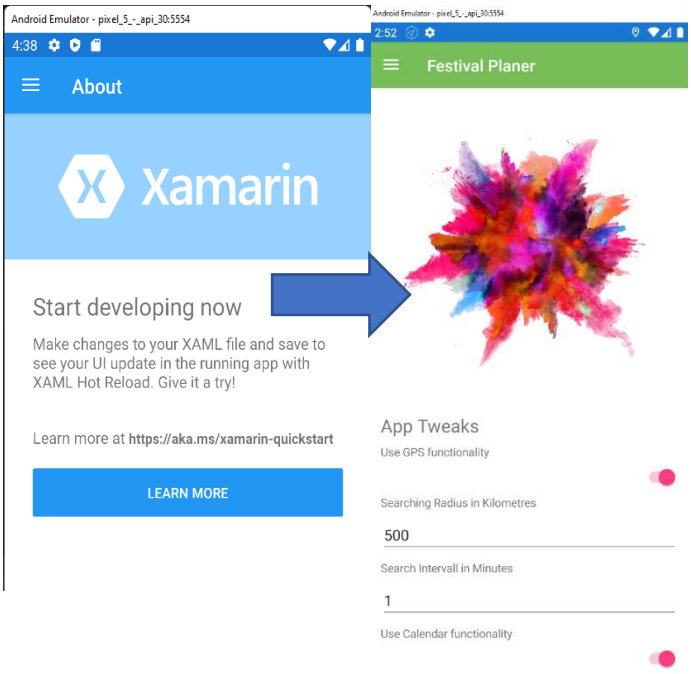
Unter iOS war das Ganze nochmal um einiges schwieriger. Um eine App überhaupt auf ein iOS Gerät spielen zu können, braucht man unter einem Windows Betriebssystem einen gültigen Apple Developer Account. Da ich in den Anfangszeiten allerdings nur ein Iphone besaß, habe ich mich recht bald damit beschäftigt, wie man eine App auf ein Iphone spielen kann, ohne im Besitz eines Apple Developer Accounts zu sein. Die Lösung dafür war es, eine Virtual Machine mit MacOS aufzusetzen und dann über XCode ein temporären Apple Developer Zertifikat zu erstellen. Der nächste Schritt war es dann innerhalb Visual Studio für Mac im Xamarin Projekt, den selben Namen und Bundle Identifier wie in XCode zu nutzen. Damit war es dann möglich, sobald das Zertifikat am Iphone bestätigt wurde, die App auch über das Iphone zu starten. Allerdings war dieser Vorgang mit viel Kopfschmerzen verbunden, da die Geschwindigkeit in der Virtual Machine sehr langsam war und es ab und an auch immer wieder zu unverständlichen Problemen kam. Desweiteren nutzte ich Git als

SourceControl, um den aktuellen Stand dann auch auf die Virtual Machine pullen zu können. Im Zeitfenster von einer Woche war das Zertifikat dann abgelaufen und der Spaß begann von vorne. Im Großen und Ganzen empfehle ich jedem, der eine App in das Apple Universum bringen möchte, einen Developer Account anzulegen, denn dann ist es auch möglich die App per Windows auf das Iphone zu spielen.

8. Umbauen des Templates für meine Anforderungen

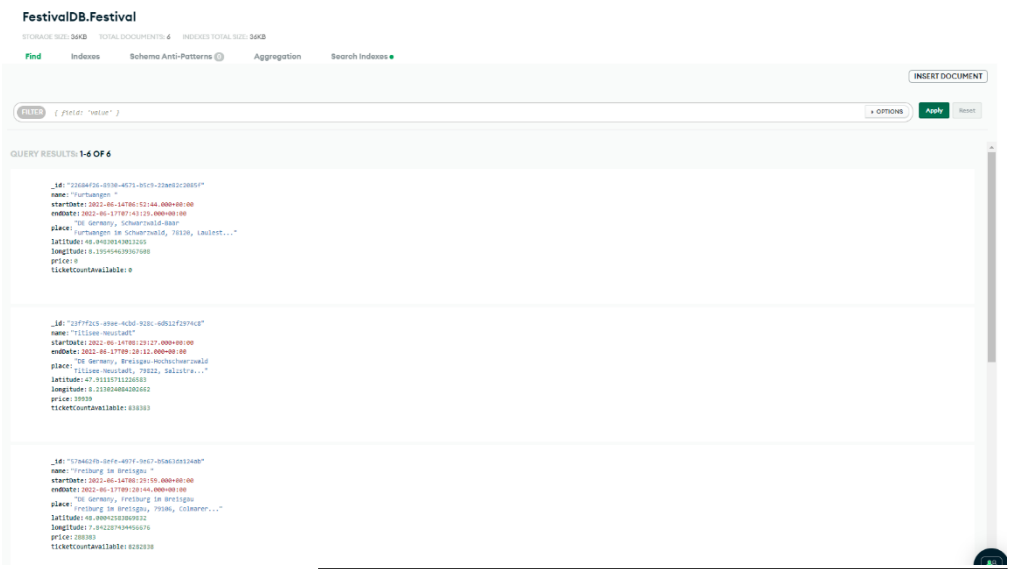
Als die Grundbausteine gelegt wurden, konnte ich anfangen das gegebene Template von Xamarin für meine Zwecke umzubauen. Das Template gab mir schon verschiedene Ansichten der App, eine Fly-Out Navigation und die Möglichkeit eine Liste von Namen, Zahlen und Werte anzulegen, die persistent in der App gespeichert wurden. Das erste, was ich anpasste, waren die verschiedenen Ansichten.

Zuerst begann ich also die Startseite entsprechend meinen Vorstellungen anzupassen. Der nächste Schritt war die Anpassung der farblichen Komponenten: von einem Blau Ton verwandelte ich die App in eine Mischung aus Hell Grün und Gelb. Die weitere Anpassung der App erfolgte nach und nach über die Zeit, den Anforderungen entsprechend, die ich mir selbst gesetzt hatte.



8.1. Implementierung der Datenbank

Der nächste Schritt für mich war es die Anbindung einer Datenbank zu realisieren. Die Datenbank sollte für die App die Festival Informationen für jedes einzelne Festival in einem Objekt persistent abspeichern.



Quelle: MongoDB Cloud, Ansicht Datenbankstruktur

8.2. MongoDB Package

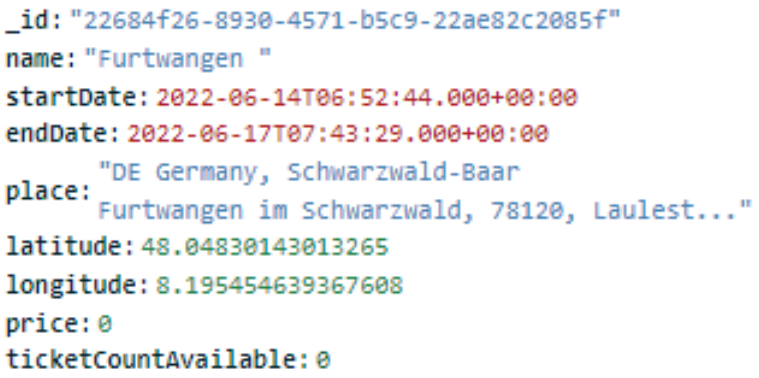
Ich habe mich für die Lösung einer MongoDB Datenbank entschieden, da ich einerseits schon mit MongoDB gearbeitet habe und ich andererseits schon ein passendes Plugin im NugetManager gefunden hatte. Die Anbindung über Code war wie vom Plugin versprochen sehr einfach und verständlich, so hatte ich nach einer Worksession die Anbindung zur Datenbank hergestellt.

Quelle: Wikipedia, MongoDB Logo

8.3. Datenbankstruktur

Danach beschäftigte ich mich mit der passenden Struktur der Datenbank für mein Projekt. Über die Zeit veränderte sich auch nochmal die Struktur da sich herausstellte, dass es innerhalb der App sehr praktisch ist, Längen und Breitengrad mit in den Festival Informationen zu speichern. Die endgültige Struktur sah dann wie folgt aus:

1. Eine eindeutige UUID mit der jedes Festival zu einem einzigartigen Objekt wurde
2. Information wie Name, Standort, Preis und Ticket, die jeweils entweder als String oder Float abgespeichert wurden



3. Information über das Start- und Enddatum, welches vom Typ der Klasse „Date“ von C# abgleitet und abgespeichert wurde.
4. Und zuletzt schon die erwähnte, wichtige Informationen zu Längen und Breitengrad.

9. Implementierung GPS

Nachdem die Anbindung der Datenbank erfolgreich abgeschlossen war und es nun möglich war innerhalb der App neue Festivals mit deren Informationen abzuspeichern, ging es für mich als Nächstes um das Abfragen der Koordinaten über den geräteinternen GPS Sensor. Dabei fand ich wieder Hilfe in der Bibliothek von Xamarin und zwar ganz konkret in der Bibliothek von Xamarin.Essentials.

Unter anderem befand sich in der genannten Bibliothek auch eine Klasse für die Anbindung des GPS Sensors.

```
var request = new GeolocationRequest(GeolocationAccuracy.High, TimeSpan.FromSeconds(45));
cts = new CancellationTokenSource();
actualLocation = await Geolocation.GetLocationAsync(request, cts.Token);
```

Mit diesen 3 Zeilen und mithilfe des Plugins konnte man jeweils Länge und Breitengrad und die aktuelle Höhe des Endgerätes abfragen (Lattitude, Longitude, Altitude). Zusätzlich musste man in der Android und iOS Solution noch bestimmte

Permission freigeben, dass man innerhalb der General Solution die GPS Daten abfragen konnte.

10. Implementierung Maps

Nachdem ich erfolgreich die aktuellen GPS Daten über das Endgerät abfragen konnte, musste ich im nächsten Schritt eine Möglichkeit finden, bei der Erstellung eines Festivals, ebenfalls die gewünschten GPS Koordinaten wo das Festival stattfinden soll, zu übergeben und abzuspeichern. Anfangs dachte ich, es wäre der einfachere Weg eine Scroll Bar zu bauen, die jede Stadt ab zum Beispiel zehntausend Einwohner beinhaltet. Es stellte sich aber heraus, das dieser Weg damit einhergeht, dass man die Einträge alle statisch hard coded. Das war einerseits nicht mein Ziel und würde auf der anderen Seite sehr viel Zeit in Anspruch nehmen. Die nächste Idee die sich entwickelte war, das Ganze über eine integrierte Map Ansicht zu lösen. Der allgemeine Ansatz war dabei, dass der Nutzer beim Erstellen eines neuen Festivals, die Möglichkeit besitzt den Standort über eine Map Ansicht festzulegen. Über das Klicken auf die Map sollte dann eine sogenannte Pin Nadel gesetzt werden und damit den Standort des Festivals festlegen.

10.1. Android Google Maps

Hier muss man ausnahmsweise zugeben, dass die Integration der Apple Maps deutlich einfacher ausgefallen ist, als die Integration der Google Maps. Für das Nutzen der Google Maps API muss man sich zuerst im Google Developer Programm registrieren. Danach legt man über deren Website einen Google Maps API Key an. Dieser wird dann im Android Manifest als Meta Data hinterlegt.

```
<meta-data android:name="com.google.android.geo.API_KEY" android:value="AIzaSyCDo_ClHpHKKEEVJM4imRPNvawZ4VCRcrA"
```

Vermutlich wird sobald innerhalb meiner Anwendung Google Maps aufgerufen wird, der hinterlegte API Key auf seine Gültigkeit überprüft und dann der Zugang auf Google Maps freigegeben.

10.2. iOS Apple Maps

Unter iOS war das Ganze wie oben schon erwähnt, deutlich einfacher umzusetzen. Das

```
Xamarin.FormsMaps.Init();
```

Einzige, was dafür nötig war, war ein Einzeiler in der iOS AppDelegate Datei mit dem man den Zugang zu den Apple Maps „triggern“ konnte. Danach funktionierten die Apple Maps genauso zuverlässig wie Google Maps und lieferten auch dieselben Ergebnisse.

10.3. Vom Pin zur Position und Adresse

Der letzte Baustein der fehlte war die Funktion, die es dem Nutzer ermöglichen sollte, bei der Erstellung eines neuen Festivals, eine bestimmte Position auf der Map festzulegen, um dann dem Festival eine einzigartige Lokalisierung zu übergeben. Die Lösung dafür stellte Xamarin out of the box bereit, mit dem sogenannten „Map_MapClicked“ Event.

```
public async void Map_MapClicked( object sernder, MapClickedEventArgs e)
{

    try
    {
        Map.Pins.Clear();
        var result = await Geocoding.GetPlacemarksAsync(e.Position.Latitude, e.Position.Longitude);
        if (!result.Any()) return;
        pin = new Pin
        {
            Type = PinType.Place,
            Position = new Position(result.FirstOrDefault().Location.Latitude, result.FirstOrDefaul
            Label = result.FirstOrDefault().CountryCode + " " + result.FirstOrDefault().CountryName
            Address = result.FirstOrDefault().Locality + ", " + result.FirstOrDefault().PostalCode
        };
        Map.Pins.Add(pin);
        NewItemViewModel.place = pin.Label + "\n" + pin.Address;
        NewItemViewModel.latitude = result.FirstOrDefault().Location.Latitude;
        NewItemViewModel.longitude = result.FirstOrDefault().Location.Longitude;
    }
    catch (Exception ex)
    {
        await App.Current.MainPage.DisplayAlert("Location not found", ex.ToString(), "Cancel");
    }
```

Über das Map Clicked Event wurde jedes Mal, oben gezeigte Methode ausgeführt. Im nächsten Schritt suchte ich die Lösung, wie man eine Pin Nadel auf der Map setzen konnte. Auch hierfür gab es in der Xamarin.Essentials Bibliothek schon die passende Klasse dazu. Über das Map Clicked Event und dem Übergabeparameter MapClickedEventArgs „e“ hatte ich Zugriff auf die Längen und Breitengrade der geklickten Position. Aus der Kombination der geklickten Position und der gegebenen

Funktion Geocoding.GetPlacemarksAsync aus der Xamarin.Essentials Bibliothek, konnte man dann Dinge wie: Postleitzahl, Ort, Straße, Region, Land und vieles mehr abspeichern. Zusätzlich speicherte ich im Hintergrund, für den Nutzer nicht sichtbar, die zugehörigen Längen und Breitengrade ab.

11. Background Tasking

Der Hintergrundprozess auf mobilen Endgeräten unterscheidet sich maßgeblich von dem traditionellen Konzept des Multitaskings auf dem Desktop Rechner. Desktop Anwendungen verfügen über viel mehr Ressourcen, als mobile Anwendungen. Wie zum Beispiel: Bildschirmfläche, Leistung im Sinne von Strom und Arbeitsspeicher. Anwendungen werden auf dem Desktop Rechner nebeneinander ausgeführt und bleiben dabei leistungsfähig und benutzbar. Auf mobilen Endgeräten sind die Ressourcen viel begrenzter. Schon allein durch die kleine Bildschirmfläche, wird das Prinzip der Nebenläufigkeit verschiedener Programme hinfällig. Die Ausführung mehrer Anwendungen mit voller Geschwindigkeit würden dazu zu Leistungs und Strom Problemen führen. Der Hintergrundbetrieb einer mobilen Anwendung ist also ein ständiger Kompromiss zwischen der Bereitstellung von Ressourcen für die Ausführung der Hintergrundaufgaben und der Aufrechterhaltung der Reaktionsfähigkeit der im Vordergrund befindlichen Anwendung des mobilen Endgeräts. Sowohl iOS als auch Android haben Lösungen für die Hintergrundverarbeitung, aber sie werden auf sehr unterschiedliche Weise gehandhabt.

Quelle Microsoft

11.1. Android Background Task

Über das Messaging Center schicke ich eine Nachricht an die Android Solution, dass der Background Task gestartet werden soll. Dann rufe ich in der Klasse AndroidLocationService.cs folgende Methode auf:

```
0 references
public override StartCommandResult OnStartCommand(Intent intent, StartCommandFlags flags, int startId)
{
    _cts = new CancellationTokenSource();

    Notification notif = DependencyService.Get<INotification>().ReturnNotif();
    StartForeground(SERVICE_RUNNING_NOTIFICATION_ID, notif);

    Task.Run(() => {
        try
        {
            GeoLocationService.Run(_cts.Token).Wait();
        }
        catch (OperationCanceledException)
        {
        }
        finally
        {
            if (_cts.IsCancellationRequested)
            {
                var message = new StopServiceMessage();
                Device.BeginInvokeOnMainThread(
                    () => MessagingCenter.Send(message, "ServiceStopped")
                );
            }
        }
    }, _cts.Token);

    return StartCommandResult.Sticky;
```

Sobald der Command ausgeführt und die Klasse GeoLocationService in der GeneralSolution deren Run Methode erfolgreich ausgeführt wurde, wird ein neuer Background Task gestartet. Solange der Nutzer in den App Tweaks den GPS Toggle nicht auf false setzt, läuft der Background Task also unendlich.

11.2. iOS Background Task

In iOS konnte ich das Konzept, das oben schon erläutert wurde, mehr oder weniger gleich weiterführen. Das Ganze wurde intern in der iOS Solution nochmal etwas komplizierter umgesetzt, aber der Kern des Hintergrund Dienstes ist sehr ähnlich und im folgenden Foto abgebildet:

```
nint _taskId;
CancellationTokenSource _cts;
public bool isStarted = false;

1 reference
public async Task Start()
{
    _cts = new CancellationTokenSource();
    _taskId = UIApplication.SharedApplication.BeginBackgroundTask("com.company.Festival", OnExpiration);

    try
    {
        isStarted = true;
        await GeoLocationService.Run(_cts.Token);

    }
    catch (OperationCanceledException)
    {
    }
    finally
    {
        if (_cts.IsCancellationRequested)
        {
            var message = new StopServiceMessage();
            Device.BeginInvokeOnMainThread(
                () => MessagingCenter.Send(message, "ServiceStopped")
            );
        }
    }

    var time = UIApplication.SharedApplication.BackgroundTimeRemaining;

    UIApplication.SharedApplication.EndBackgroundTask(_taskId);
}
```

12. Kalender Integration

12.1. iOS EventKit

Mit dem iOS EventKit, welches Xamarin bereitstellt, war es mir möglich auf den geräteinternen Kalender zuzugreifen. Diese Schnittstelle baute ich dann in meine Klasse CalendarService ein:

```
public CalendarService(DateTime _startTime, DateTime _endTime, string _title, string _description)
{
    Device.BeginInvokeOnMainThread(() =>
    {
        newEvent = EKEvent.FromStore(eventStore);

        newEvent.AddAlarm(EKAlarm.FromDate((NSDate)(_startTime - TimeSpan.FromDays(1))));
        newEvent.StartDate = (NSDate)_startTime;
        newEvent.EndDate = (NSDate)_endTime;
        newEvent.Title = _title;
        newEvent.Notes = _description;
        newEvent.Availability = EKEventAvailability.Busy;
    });
    startTime = _startTime;
    endTime = _endTime;
    CreateEvent();
}
```

```
public static bool CheckIfDateIsAvailable(DateTime startTime, DateTime endTime)
{
    bool isFree = false;
    var predicate = AppEvent.CurrentEvent.EventStore.PredicateForEvents((NSDate)startTime, (NSDate)endTime
    var possibleEvents = AppEvent.CurrentEvent.EventStore.EventsMatching(predicate);
    if (possibleEvents.ToArray().Length > 0)
        foreach (EKEvent ev in possibleEvents.ToArray())
        {
            if (ev.Calendar.Title != "Deutsche Feiertage")
            {
                Console.WriteLine(ev.Title);
                isFree = false;
                break;
            }
            else isFree = true;
        }
    else
        isFree = true;

    return isFree;
}
```

Die statische Methode CheckIfDateIsAvailable wird über das MessagingCenter aus der General Solution aufgerufen, dann wird erstmals überprüft, ob zur angegebenen Festival Zeit schon vorhandene Events stattfinden, falls dies der Fall ist, returned die Methode ein false. Falls zu dieser Zeit noch keine Events eingetragen wurden, geht es weiter zur Methode CreateEvent:

```
void CreateEvent()
{

    EKCalendar[] calendars = eventStore.GetCalendars(EKEntityType.Event);
    foreach (EKCalendar calendar in calendars)
    {
        if (calendar.Title == "Calendar")
        {
            eKCalendar = calendar;
        }
    }
    Device.BeginInvokeOnMainThread(() =>
    {

        newEvent.Calendar = eKCalendar;
        NSError e;
        eventStore.SaveEvent(newEvent, EKSpan.ThisEvent, out e);


    });
}
```

Innerhalb dieser Methode wird das Festival dann nur noch in den iOS Kalender hinzugefügt.

12.2. Android Kalender API

Bei der Implementierung des Android Kalenders hatte ich mich etwas schwerer getan als anfangs erwartet. Das Hauptproblem dabei war es, dass Android Open Source ist und es eine große Menge an unterschiedlichen Endgeräten und Herstellern gibt. Dadurch gibt es auch eine große Anzahl an unterschiedlichen internen Kalender Lösungen. Nach langem hin und her probieren mit der gestellten Lösung von Xamarin und keinem zufriedenstellenden Ergebnis, bin ich dann schlussendlich auf das Drittanbieter Add On CrossCalendar gestoßen. Das Plugin ermöglichte mir direkten Zugriff auf die Kalender, ohne viel Code selber schreiben zu müssen. Tatsächlich ließ sich der Code in der Android Solution dann sehr ähnlich schreiben, wie in der iOS Solution schon oben gezeigt.

```
public CalendarService(DateTime _startTime, DateTime _endTime, string _title, string _description)
{

    newEvent = new CalendarEvent()
    {
        Name = _title,
        Description = _description,
        Start = _startTime,
        End = _endTime,
        Reminders = new List<CalendarEventReminder> { new CalendarEventReminder() }
    };
    startTime = _startTime;
    endTime = _endTime;
    Task.Run(async () =>
    {
        await CreateEvent();
    });

}
```

```
0 references
public static async Task<bool> CheckIfDateIsAvailable(DateTime startTime, DateTime endTime)
{
    bool isAvailable = false;
    var calendars = await CrossCalendars.Current.GetCalendarsAsync();
    var myCalendar = calendars[0];
    var possibleEvents = await CrossCalendars.Current.GetEventsAsync(myCalendar, startTime, endTime);
    if (possibleEvents.Count == 0)
        isAvailable =  true;
    return isAvailable;
}
```

Die statischee Methode CheckIfDateIsAvailable wird über das MessagingCenter aus der General Solution aufgerufen, dann wird zuerst überprüft, ob zur angegebenen Festival Zeit schon vorhandene Events stattfinden, falls dies der Fall ist, wird kein Eintrag in den Kalender vorgenommen. Falls zu dieser Zeit noch keine Events eingetragen wurden, geht es weiter zur Methode CreateEvent:

```
1 reference
public async Task CreateEvent()
{
    var calendars = await CrossCalendars.Current.GetCalendarsAsync();
    var myCalendar = calendars[0];
    await CrossCalendars.Current.AddOrUpdateEventAsync(myCalendar, newEvent);
}
```

Innerhalb dieser Methode wird das Festival dann nur noch in den Android/Google Kalender hinzugefügt.

13. Verbindung der Elemente im Hintergrundprozess

Nachdem alle Anforderungen innerhalb meiner einzelnen Arbeitsschritte erfolgreich implementiert wurden, ging als Nächstes darum die Elemente logisch, den Anforderungen entsprechend, im Hintergrund zu verbinden.

```
public static async Task Run(CancellationToken token)
{
    await Task.Run(async () =>
    {
        while (Views.FestivalPlaner.gpsToggle)
        {
            token.ThrowIfCancellationRequested();
            try
            {

                await Task.Delay(TimeSpan.FromMinutes(Views.FestivalPlaner.gpsSearchIntervall));
                await GeoLocationService.GetCurrentLocation();
                if (GeoLocationService.actualLocation != null)
                {
                    var message = new LocationMessage();
                    Device.BeginInvokeOnMainThread(() =>
                    {
                        MessagingCenter.Send<LocationMessage>(message, "Location");
                    });
                }
            }
            catch (Exception ex)
            {
                Device.BeginInvokeOnMainThread(() =>
                {
                    var errormessage = new LocationErrorMessage();
                    MessagingCenter.Send(errormessage, ex.ToString());
                });
            }
        }
        return;
    }, token);
}
```

Innerhalb des Tasks Run in der Klasse GeoLocationService wird eine While Schleife im Hintergrund ausgeführt. Innerhalb der Schleife gibt es einen individuell einstellbaren Delay. Zum Beispiel wird die Methode GetCurrentLocation dann alle 30 Minuten aufgerufen. Innerhalb der Methode GetCurrentLocation werden dann die verschiedenen Anforderungen nach der Reihe nach abgearbeitet.

```
public static async Task GetCurrentLocation()
{
    try
    {
        var request = new GeolocationRequest(GeolocationAccuracy.High, TimeSpan.FromSeconds(45));
        cts = new CancellationTokenSource();
        actualLocation = await Geolocation.GetLocationAsync(request, cts.Token);
        isCeckingDatabase = true;
        foreach (FestivalModel festivalModel in App.festivals)
        {
            double nearestLocationFestival = Location.CalculateDistance(actualLocation, new Location(festivalModel.latitude,
            if (nearestLocationFestival < Views.FestivalPlaner.gpsSearchRadius)
            {
                newNearFestival = true;
                foreach (NotificationFestival nearFestival in nearFestivals.ToArray())
                    if (nearFestival.Festival == festivalModel) newNearFestival = false;
                if (newNearFestival)
                {
                    var dateCheckerMessage = new DateCheckerMessage(festivalModel.startDate, festivalModel.endDate);
                    if (Views.FestivalPlaner.calendarToggle)
                    {

                        MessagingCenter.Send(dateCheckerMessage, "DateCheckerMessage");
                        await Task.Delay(TimeSpan.FromSeconds(1));
                    }

                    if (!Views.FestivalPlaner.calendarToggle)
                    {
                        var rndVerficationNumber = new Random().Next();
                        notificationIncrement++;
                        var notification = new NotificationRequest
                        {
                            BadgeNumber = notificationIncrement,
                            Title = "Festival at your location!",
                            Subtitle = festivalModel.name + "\n" + festivalModel.place,
                            Description = "Entfernung: " + nearestLocationFestival + " km",
                            NotificationId = rndVerficationNumber
                        };
                        var tempNearFestival = new NotificationFestival(festivalModel, notification);
                        nearFestivals.Add(tempNearFestival);
                    }
```

```
                else if (dateCheckerMessage.isFree)
                {
                    var rndVerficationNumber = new Random().Next();
                    notificationIncrement++;
                    var notification = new NotificationRequest
                    {
                        BadgeNumber = notificationIncrement,
                        Title = "Festival at your location!",
                        Subtitle = festivalModel.name + "\n" + festivalModel.place,
                        Description = "Entfernung: " + nearestLocationFestival + " km",
                        NotificationId = rndVerficationNumber
                    };
                    var tempNearFestival = new NotificationFestival(festivalModel, notification);
                    nearFestivals.Add(tempNearFestival);
                }
```

1. Zuerst hole ich mir die aktuelle Position des Endgerätes über GetLocationAsync.
2. Danach gehe ich mit einer For Schleife durch die in der Datenbank angelegten Festivals. In der nächsten Zeile überprüfe ich die Distanz zwischen dem Endgerät und dem Festival.
3. Falls die Distanz kleiner ist, als der Threshold-Radius geht es eine Zeile weiter zur nächsten Überprüfung: es gibt einen zweiten Array in dem Festivals gespeichert werden, die schon als Push Nachricht versendet wurden, falls das aktuelle Festival sich schon im Array befindet, wird das nächste Festival in der Datenbank überprüft.
4. Falls es sich um ein Festival handelt, dass bisher noch nicht als Push Nachricht weitergeleitet wurde, geht es zur Überprüfung des Kalenders: Über das MessagingCenter wird eine dateCheckerMessage an die iOS oder Android Solution gesendet. Dann wird im internen Kalender bezüglich des angegebenen Datums überprüft, ob es zu dieser Zeit schon eingetragene Events gibt. Wenn es schon ein Event gibt, wird der Vorgang abgebrochen und das nächste Festival in der Datenbank wird überprüft
5. Wenn allerdings zu dieser Zeit keine Events im Kalender stehen kommt es zur Erstellung einer Push Benachrichtung

Wie ich die die Push Benachrichtungen umgesetzt habe, soll es im nächsten Abschnitt meiner Gliederung gehen.

13.1. Push Benachrichtigungen

Nachdem die Abfrage der Position, die Integration der Maps, die Integration des Kalenders und die Hintergrundaktualisierung dieser Elemente erfolgreich funktionierte, ging es als Nächstes darum, aus deren Ergebnis eine Push Nachricht zu versenden. Die allgemeine Push Nachrichten Funktion bekam ich über ein Drittanbieter Add On durch den NugetPacket – Manager, namens „NotificationCenter“.

Falls das Festival alle oben genannte Schritte erfolgreich durchlaufen ist, wird eine neue Push Benachrichtung erstellt, die ich wiederum in einer eigenen Klasse untergebracht habe, names NotificationFestival, der weitere Verlauf der Push Nachricht wird dann in der Klasse verarbeitet, die in der unteren Abbildung dargestellt ist.

```
public class NotificationFestival
{

    public FestivalModel Festival { get; set; }


    public NotificationRequest NotificationRequest { get; set; }


    public NotificationFestival(FestivalModel festivalModel, NotificationRequest notificationRequest)
    {
        Festival = festivalModel;
        NotificationRequest = notificationRequest;
        Task.Run(() => GeoLocationService.viewModel.ExecuteLoadItemsCommand()).Wait();
        Task.Run(() => this.StartNotification()).Wait();
    }


    public async Task StartNotification()
    {
        await NotificationCenter.Current.Show(this.NotificationRequest);

        NotificationCenter.Current.NotificationTapped += this.OnLocalNotificationTapped;
    }
    private async void OnLocalNotificationTapped(NotificationEventArgs e)
    {

        if (this.NotificationRequest.NotificationId == e.Request.NotificationId)
        {
            GeoLocationService.nearFestivals.Remove(this);
            await Shell.Current.GoToAsync($"{nameof(ItemDetailPage)}?{nameof(ItemDetailViewModel.Item
            if (Views.FestivalPlaner.calendarToggle)
                Device.BeginInvokeOnMainThread(() =>
                {
                    var calendarMessage = new CalendarMessage(this.Festival.startDate, this.Festival.
                    MessagingCenter.Send(calendarMessage, "CreateCalendar");
                });
            NotificationCenter.Current.NotificationTapped -= this.OnLocalNotificationTapped;
            GeoLocationService.notificationIncrement--;

        }
```

Die NotificationFestival Klasse besitzt einerseits das Festival Objekt und andererseits die Notification. Interessant an der Klasse ist das Event OnLocalNotificationTapped. Mit diesem Event konnte ich, sobald der Nutzer auf die Push Nachricht drückt, auf die detaillierte Ansicht des Festivals weiterleiten und den entsprechenden Kalender Eintrag durchführen.

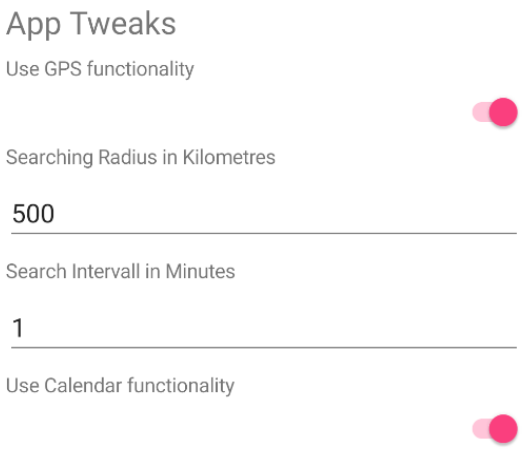
14. Einstellbare Parameter / App Tweaker

Als alle Anforderungen erfüllt waren, ging es für mich noch darum, die App etwas modularer zu gestalten. Mein Ziel war es, dem Nutzer mehr Kontrolle über den Hintergrund Prozess zu geben. Deswegen entschied ich mich innerhalb der Startseite verschiedene Input Elemente über XAML hinzuzufügen, die es dem Nutzer ermöglicht verschiedene Parameter im Hintergrund Prozess anzupassen.

14.1. Background Service, Radius, Zeit-Intervall, Kalender Funktion

Die Parameter die für mich am sinnvollsten erschienen waren:

- GPS Funktion AN/AUS
- Suchradius
- Suchintervall
- Kalender Funktion AN/AUS



Dadurch war es dem Nutzer nun möglich die Hintergrund Suche nach Festivals gänzlich auszuschalten. Oder die GPS Funktion zu nutzen, aber die Kalender Funktionen zu deaktivieren. Oder standardmäßig beide Funktionen zu nutzen, aber den Suchradius und das Intervall nach seinen Belieben zu ändern. Das Suchintervall wird in Minuten übergeben, der Grund dafür war hauptsächlich für meine eigenen Testzwecke. Im Falle einer Veröffentlichung, wäre wahrscheinlich ein Intervall in

Stunden deutlich sinnvoller, da der Hintergrund Prozess sonst zu viel Leistung und Akku verbraucht. Und das stündliche oder noch längeres Intervall der Suche nach Festivals in der Nähe, sollte meiner Meinung nach mehr als ausreichen.

15. Fazit

Im Großen und Ganzen kann ich sagen, dass ich mit Xamarin sehr zufrieden war. Es gab nie wirklich Probleme, die sich nicht lösen ließen. Ich denke ein zentraler Punkt wieso es so gut lief war, dass Xamarin exakt für mein Projekt zugeschnitten war: es geht für das Framework Xamarin primär darum, über eine Code Base, eine native App für sowohl Android als auch iOS zu entwickeln. Auch das große open source Ökosystem von Bibliotheken hat mir meine Arbeit um vieles erleichtert. Die Xamarin.Forms Dokumentation ist inzwischen wirklich umfangreich und sowohl in Englisch als auch Deutsch gut ausformuliert. Der einzige Punkt in den Anforderungen, an der ich etwas länger zu knappern hatte, war der Hintergrundprozess. Dabei kann man diesen Teil auch nicht wirklich auf schlechte Umsetzung von Xamarin setzen, da Background Tasking allgemein auf mobilen Endgeräten schwierig umsetzbar ist und die Betriebssysteme am liebsten gar keine Background Tasks mehr unterstützen möchten. Einerseits wegen des hohen Energieverbrauchs, andererseits weil es inzwischen deutlich effizientere Wege gibt. Wie zu Beispiel den Background Fetch: dieser wird nur stündlich/täglich ausgeführt und über eine REST API läuft. Heißt konkret: nicht mehr das Handy bzw. die App führt die Arbeit aus, sondern irgendwelche Server verarbeiten die Daten und geben dann ein Request über die REST API an die App Schnittstelle weiter. Der Energieverbrauch wird also auf ein Bruchteil des Background Tasks heruntergebrochen.

16. Quellenverzeichnis

- https://docs.microsoft.com/de-de/xamarin/android/internals/architecture
- https://de.wikipedia.org/wiki/Xamarin
- https://docs.microsoft.com/de-de/xamarin/get-started/what-is-xamarin-forms
- https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui
- https://docs.microsoft.com/de-de/nuget/what-is-nuget

- https://docs.microsoft.com/de-de/xamarin/xamarin-forms/xaml/xaml-basics/
- https://docs.microsoft.com/de-de/dotnet/csharp/tour-of-csharp/
- https://docs.microsoft.com/de-de/xamarin/android/app-fundamentals/services/
- https://docs.microsoft.com/de-de/xamarin/ios/platform/eventkit
- https://docs.microsoft.com/de-de/xamarin/ios/internals/architecture
- https://docs.microsoft.com/de-de/xamarin/android/user-interface/controls/calendar