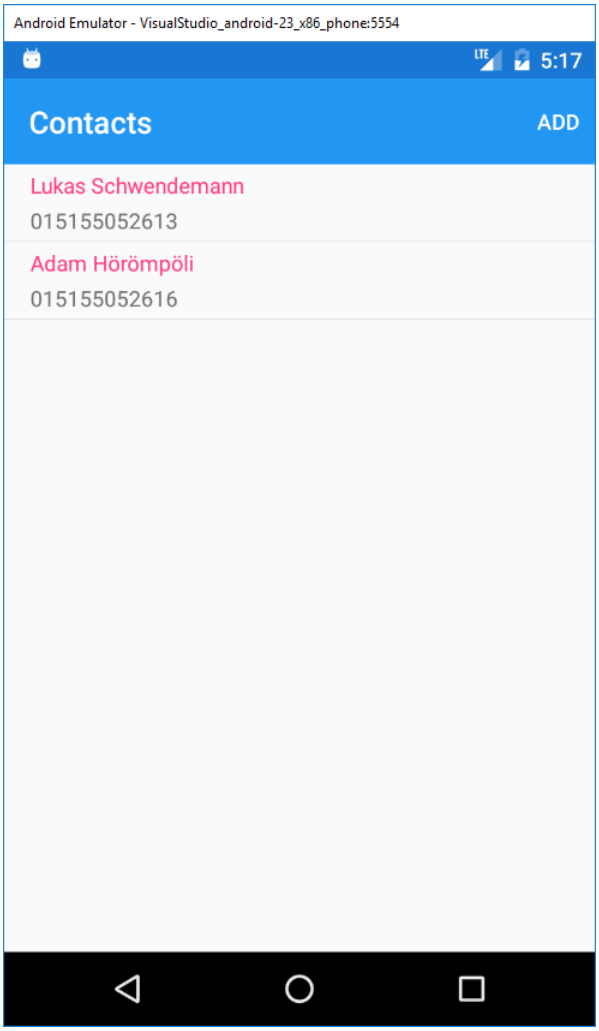
Hands On Lab – Kontaktverwaltung in Xamarin.Forms

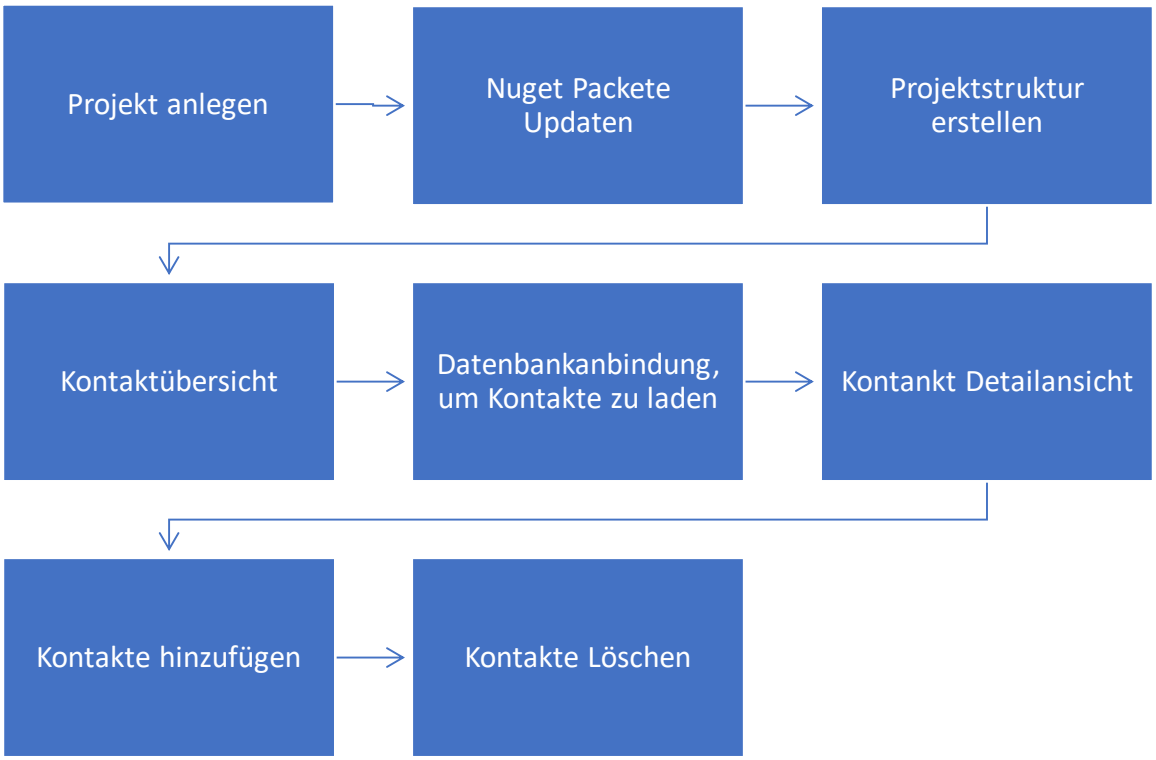
Ziel: Erstellung einer Kontaktverwaltung unter Verwendung von Xamarin.Forms, welche die selbst hinterlegten Daten mittels SQLite persistiert. Die eingetragenen Kontakte können über eine Liste eingesehen und über eine Detailansicht bearbeitet werden.



Voraussetzungen:

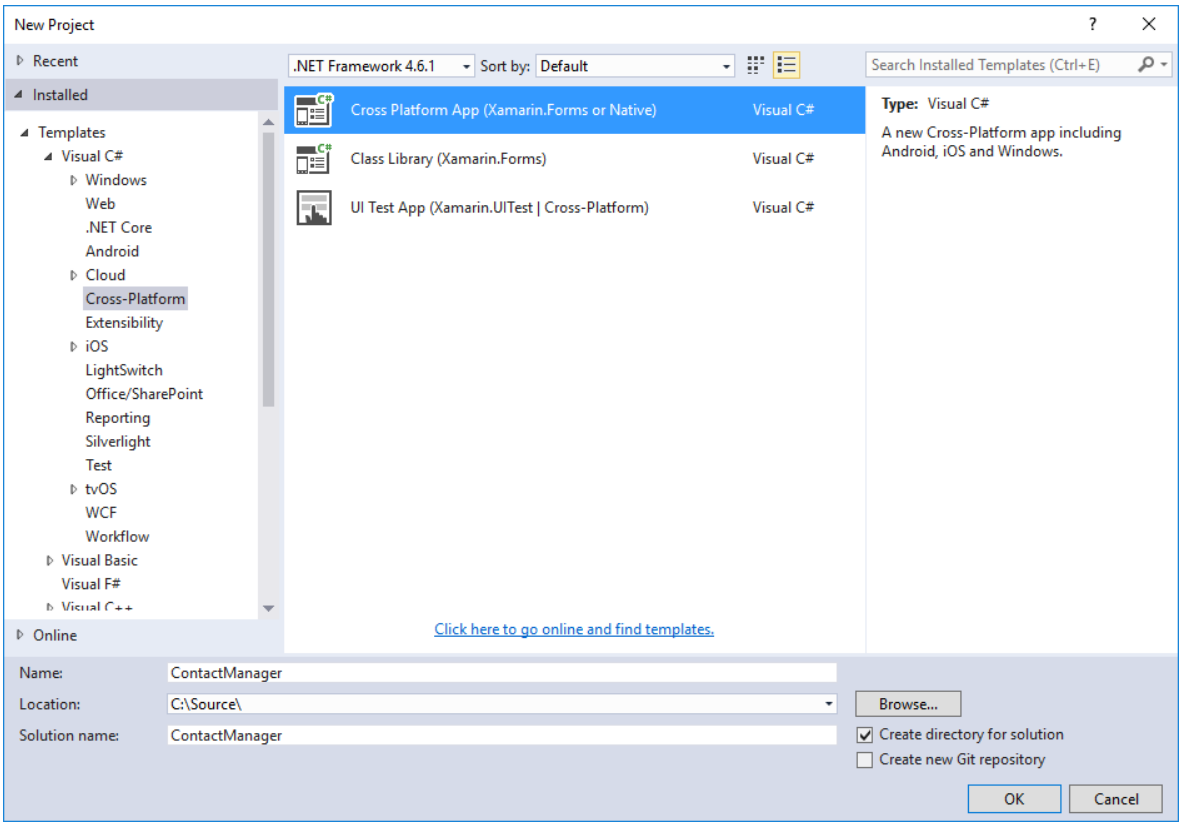
- Mindestens Windows 8.1, besser Windows 10 (für UWP Entwicklung)
- Visual Studio 2015 oder 2017 (ab Community Edition)
- Android Emulator mit Android SDK und Emulator in derselben Version

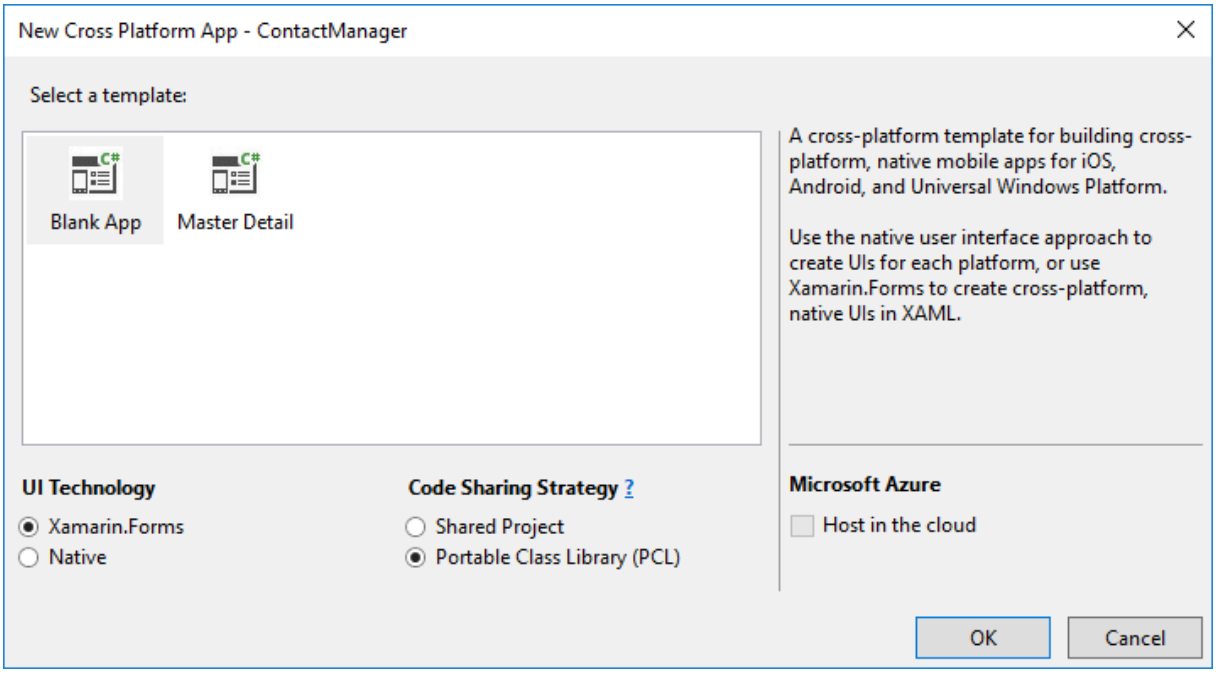
Ablauf



Nach jedem Prozess-Schritt sollte das Ergebnis durch einen Start der Anwendung überprüft werden, dabei auch den Unterschiedlichen Plattformen Beachtung schenken. Wenn die Anwendung einwandfrei auf einer Plattform läuft heißt das nicht, dass sie das auch auf allen anderen Plattformen tut.

1. Erstellen der Solution (5 Min)



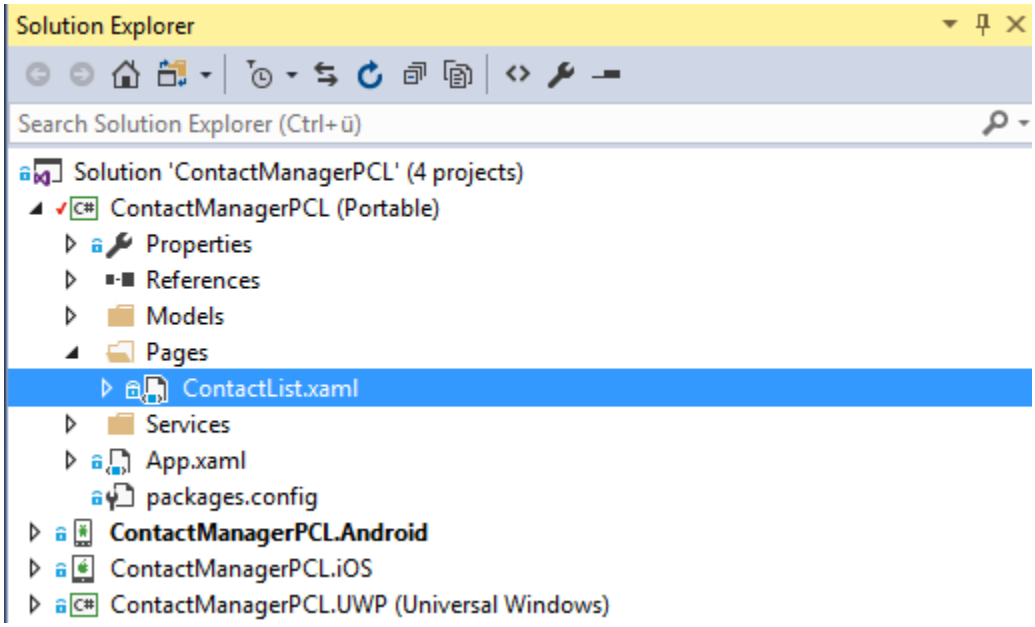


2. Update der NuGet Packete (5 Min)

- UWP (bei Visual Studio 2015 kleiner als Version 5.3
- Zuerst das Xamarin.Forms Paket, danach die Android Abhängigkeiten aktualisieren
- Anwendungen auf jeder Plattform einmal starten, um die Funktionalität sicher zu stellen
    - Android mit der Konfiguration AnyCpu
    - UWP mit der Konfiguration x86 oder x64

3. Erstellen der Projektstruktur (5 Min)

- Erstellen der Ordner Models, Pages und Services.
- Umbenennen der MainPage.xaml (sowohl den Dateinamen als auch die Klasse) nach ContactList.xaml. Danach die ContactList.xaml in den neu vorher erstellten Ordner Pages ziehen.



4. Erstellen der Kontaktübersichtsseite (20 Min)

4.1 Ziel

Anzeigen einer ListView direkt nach dem Start der Anwendung, in welcher Mock-Objekte in Form eines Kontaktes angezeigt werden (gesamter Name und Telefonnummer der Person). Ein Kontakt besteht generell aus den Eigenschaften Vorname, Nachname, gesamter Name, Telefonnummer und der E-Mail-Adresse.

4.2 Vorgehen

1. Erstellen einer neuen Klasse Contact im Models Order
    - FirstName
    - LastName
    - FullName (Berechnet aus FirstName und LastName)
    - TelephoneNumber
    - Email
2. In der ContactList.xaml das Initiale Label entfernen, sodass nur noch das xml Elemente ContentPage vorliegt.
3. Eine ListView hinzufügen, unter welchem ein DataTemplate mit einer TextCell verwendet wird. Das ListView Element benötigt an dieser Stelle noch einen Namen (Attribut „x:Name“), welche als Referenz im Code-Behind der ContactList dient.

```
<ListView x:Name="ContactListView">
        <ListView.ItemTemplate>
            <DataTemplate>
                <TextCell Text="{Binding FullName}" Detail="{Binding
TelephoneNumber}"></TextCell>
            </DataTemplate>
        </ListView.ItemTemplate>
</ListView>
```

4. Das Event Appearing der ContentPage verwenden, um nach dem Laden der Page den ItemSource der ListView auf eine Liste mit beliebigen Kontakten zu setzen.

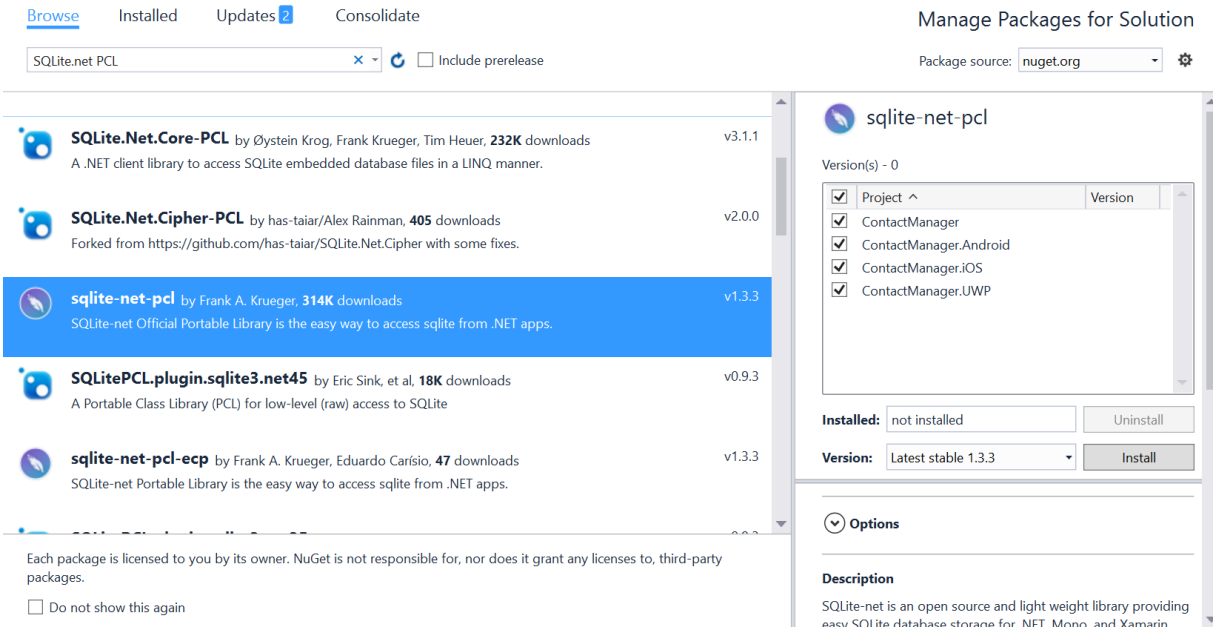
5. Datenbankanbindung zum Befüllen der ListView (50 Min)

5.1 Ziel

Die Daten innerhalb der ListView werden aus einer Datenbank geladen. Diese werden einmalig in die Datenbank geschrieben, um eine erste Liste von Kontakten anzuzeigen. Als Datenbank wird SQLite verwendet, auf welches mit dem Framework SQLite.Net-PCL zugegriffen wird. Ein einzelner Kontakt muss dabei über eine eindeutige ID identifiziert werden können.

5.2 Vorgehen

1. Erstellen eines Interfaces IContactRepository mit den Methoden
    - Task<IList<Contact>> GetContacts()
    - Task SaveContact(Contact contact)
    - Task DeleteContact(Guid id)
2. Erstellen einer Klasse ContactRepository, welches das Interface IContactRepository implementiert.
3. Installieren des NuGet Paketes "sqlite-net-pcl" in der benötigten PCL und allen Plattformprojekten:



4. Die Ablage der Dateien ist plattformspezifisch, weshalb ein weiteres Interface benötigt wird, um die SQLite-Datei zu lokalisieren: Erstellen des Interfaces IFileLocator im Services Ordner mit der Methode
    - string GetFilePath(string fileName)
5. Hinzufügen der Klasse FileLocatorAndroid im Android Projekt und das Interface IFileLocator Implementieren:
    - Mit der Funktion System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal) kann auf den Android spezifischen Pfad zur Dateiablage zugegriffen werden.
    - Der Rückgabewert der Funktion muss den gesamten Pfad inklusive des Dateinamens beinhalten
    - Globale Abhängigkeit auf Namespace-Ebene registrieren. Dieser wird vom Xamarin DependencyService benötigt, welcher durch das Attribut “[assembly: Xamarin.Forms.Dependency(typeof(FileLocatorIos))]” oberhalb des Namespace geschieht. Dadurch kann der DependencyService die Abhängigkeiten später im SharedCode auflösen.
6. Hinzufügen der Klasse FileLocatorUwp im UWP Projekt und implementieren des Interfaces IFileLocator:
    - Mit dem Property Windows.Storage.ApplicationData.Current.LocalFolder.Path kann auf den Windows spezifischen Pfad zur Dateiablage zugegriffen werden.
    - Globale Abhängigkeit auf Namespace-Ebene für den Xamarin DependencyService [assembly: Xamarin.Forms.Dependency(typeof(FileLocatorUwp))] registrieren.
7. Hinzufügen der Klasse FileLocatorIos und implementieren des Interfaces IFileLocator im iOS Projekt:
    - Mit der Funktion System.Environment.GetFolderPath(Environment.SpecialFolder.Personal) kann auf den iOS spezifischen Pfad zur Dateiablage zugegriffen werden.

    - Im übergeordneten Ordner von *Personal* wird ein Ordner *Library* mit dem Unterordner *Databases* erwartet, in welcher die Datenbanken abgelegt werden. Dieser Pfad muss manuell zusammengebaut werden.
    - im Gegensatz zu den Anderen Plattformen muss dieser Ordner explizit erstellt werden
        - Directory.Exists(libFolder)
        - Directory.CreateDirectory(libFolder)
    - Globale Abhängigkeit auf Namespace-Ebene für den Xamarin DependencyService *[assembly: Xamarin.Forms.Dependency(typeof(FileLocatorIos))]* registrieren
8. Implementierung der GetContacts Methode im ContactRepository:
    - Globale „SQLiteConnection“ definieren
    - Im Konstruktor:
        - Instanziieren eines FileLocators mit Hilfe des DependencyService (*DependencyService.Get<IFileLocator>()*)
        - Abrufen des Pfades und dabei den Dateinamen mit Endung ".db3" festlegen
        - Instanziieren der *SQLiteConnection* mit dem abgerufenen Pfad
        - Anlegen der Tabelle *Contract* (*_connection.CreateTable<Contact>()*)
    - Verwendung der Connection in GetContacts (_connection.Table<Contract>()), um den gesamten Inhalt der Tabelle abzurufen
    - Contact braucht ein zusätzliches Property ID als GUID, um Einträge eindeutig identifizieren zu können. Dieser muss mit dem Attribut "PrimaryKey" gekennzeichnet werden.
    - Mock-Daten in Tabelle anlegen, da aktuell noch keine Daten existieren (_connection.Insert).
9. Laden der Daten in die ContactList:
    - Instanziieren einer der Klasse ContactRepository in ContactList
    - Sobald die Page mit der ListView erscheint (Event: Appearing), muss das Attribut ItemSource von ListView gesetzt werden, damit die Daten aus dem Repository geladen und angezeigt werden können.

6. Ergänzung der Detailansicht von Kontakten (20 Min)

6.1 Ziel

Die ContactList Page reagiert auf das Antippen eines einzelnen Elements der ListView und navigiert zu einer neuen Detailansicht. Auf dieser werden alle Informationen eines Kontaktes (bis auf die ID) mit entsprechenden Beschriftungen und bearbeitbaren Feldern angezeigt.

6.2 Vorgehen

- Eine neue ContentPage (XAML) als "ContactDetails" anlegen und mit dem Titel „Contact details“ versehen:
    - Globale Variable "_contact" definieren.
    - In einem Stacklayout jeweils ein "Label" (zur Beschreibung des Feldes) und ein "Entry" (für die gewünschte Eingabe hinzufügen)
    - Den Feldern (Entry) mittels dem Attribut x:Name einen Namen geben, um im Code-Behind darauf zugreifen zu können.
- Im Konstruktor nimmt die ContactDetails Page einen Contact entgegen und befüllt die "Text" Properties der Entries mit den entsprechenden Werten.
- Auf das ListView Event ItemTabbed reagieren, indem die neue ContactDetailsPage instanziiert wird und dorthin navigiert wird (Navigation.PushAsync(page)). Hinweis: ItemTappedEventArgs.Item enthält das Objekt vom Typ Contact
- Für die Navigation innerhalb der App wird eine „NavigationPage“ benötigt. In App.xaml.cs der ContactList eine NavigationPage zuweisen, welche als Parameter die ContactList enthält. (new NavigationPage(new ContactList()))

7. Hinzufügen von Kontakten (10 Min)

7.1 Ziel

Auf der „ContactList“ Seite gibt es einen Toolbar-Knopf. Durch das Betätigen wird auf eine Detailseite ohne Inhalt navigiert. Die Detailseite selber hat einen zusätzlichen Toolbar-Knopf zum Speichern des aktuellen Kontaktes. Nach dem Betätigen werden alle Daten gespeichert und es wird zurück auf die „ContactList“ Seite navigiert.

7.2 Vorgehen

- Auf der ContactList Page im XAML Code ein ToolbarItem (`<ContentPage.ToolbarItems> <ToolbarItem /> </ContentPage.ToolbarItems>`) mit einem passenden Anzeigetext hinzufügen.
- Auf das Event „Click“ reagieren, indem mit einem neu instanziierten Contact auf die Details Seite navigiert wird.
- Auf der ContactDetails Page ein ToolbarItem hinzufügen, um zu speichern.
- Auf das Event „Click“ reagieren
    - die Einträge auslesen und in den globalen „Contact“ schreiben
    - Das Repository zum Speichern verwenden
    - Zurücknavigieren auf die ListView (PopAsyc()).
- Speichern des Repositories unter Verwendung der SQLite Connection implementieren. Dabei muss darauf geachtet werden, dass bereits angelegte Kontakte aktualisiert werden und nicht vorhandene Kontakte neu angelegt werden. (Insert / Update)
- Entfernen der Implementierung zur Erstellung der Mock-Daten.

8. Löschen von Kontakten (5 Min)

8.1 Ziel

Die Detailseite hat einen zusätzlichen Toolbar-Knopf zum Löschen. Nach dem Betätigen wird der Kontakt gelöscht und es wird zurück auf die „ContactList“ Seite navigiert.

8.2 Vorgehen

- Ein weiteres ToolBarItem in der ContactsDetails Page hinzufügen. Dieser wird mit "Delete" beschriftet und dem Click Event versehen, welches das Repository zum Löschen aufruft.
- Repository mit der SQLiteConnection implementieren (Delete).
- Nach dem Löschen zurück zur Übersicht navigieren.