Einfügen einer neuen Datenklasse in DiversityPhone

In DiversityPhone werden Datenklassen, die Gegenstand der Synchronisation sind, über verschiedenen Sichten angelegt. Die Grundlage ist hierfür das MVVM-Pattern (siehe auch: <http://msdn.microsoft.com/de-de/magazine/dd419663.aspx> ) . Grundlage dafür ist die Darstellung der Oberfläche in einer View, die von der Ausführungslogik getrennt ist. Diese wird im Viewmodel organisiert. Jede View ist genau einem ViewModel zugeordnet. Als Backend fungiert die lokale Datenhaltung in einer SQLCE-Datenbank. Diese verwaltet die Datenklasse in einer Tabelle und organisiert den Zugriff auf die Daten über linq2sql (siehe: <http://msdn.microsoft.com/de-de/library/vstudio/bb386976.aspx>).

# Import der Zugehörigen Tabelle aus der Serverdatenbank

**DiversityService/Model**

Für jede Datenbanktabelle, die verwendet werden soll muss eine Datenklasse angelegt werden. Diese muss für die Verwendung durch Petapoco (ORM) geeignet sein.

Näheres: <http://www.toptensoftware.com/petapoco/>

# Einfügen der nötigen Zugriffsmethoden

**Im DiversityService Projekt**  
In das IDiversityService Interface

[OperationContract]

IEnumerable<Term> GetStandardVocabulary();

In die DiversityService Implementierung

IEnumerable<Term> GetStandardVocabulary();

# Anlegen der (reduzierten) Modellklasse

**DiversityPhone/Model**

Dies ist nur nötig für Klassen, die auch clientseitig verwendet werden. Das Datenmodell kann dahingehend modifiziert werden, dass nur eine Teilmenge der Felder verwendet werden muss. Werden zusätzliche Felder clientseitig benötigt, werden sie an dieser Stelle hinzugefügt. Die Anpassung der Datenmodelle wird von der Klasse DiversityService übernommen (siehe DiversityService (TODO)).

Für die Synchronisation Implementieren alle editierbaren Klassen das Interface IModifyable

public enum ModificationState

{

New,

Unmodified,

Modified

}

public interface IModifyable

{

/// <summary>

/// Tracks the persistance status of an object.

/// null - Newly created, not yet persisted

/// false - persisted remotely, local copy unchanged

/// true - persisted only locally OR the local copy has been changed

/// </summary>

ModificationState ModificationState { get; set; }

}

Wenn verschiedene Tabllen eine identische oder vergleichbare Struktur haben, können diese zu einer Klasse zusammengefasst werden z.B. Term.

Die Klassen müssen für Linq2SQL mit entsprechenden Attributen versehen werden.

Näheres: <http://msdn.microsoft.com/en-us/library/bb386971.aspx>

Datenklassen werden über T4 Text Templates generiert. Diese Übernehmen die Generierung von Boilerplate code für INotifyPropertyChanging / -Changed Implementierung und die Linq2SQL Spaltenattribute. Als Beispiele können alle vorhandenen Templates (DiversityPhone.ServiceReference Projekt, \*.tt Dateien) dienen.

# Anlegen einer Tabelle im DatenKontext

**Im DiversityPhone Projekt,   
DiversityDataContext**

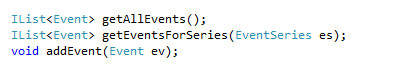
public Table<Event> Events;

# Einfügen der nötigen Zugriffsmethoden

**Im DiversityPhone Projekt,   
Ordner Services**

In das IOfflineStorage bzw. IOfflineFieldData Interface  
In die OfflineStorage Implementierung

Standard: Klassenweise Gruppierung



# ViewModels der gewünschten Seiten/Elemente anlegen

**Im DiversityPhone Projekt,   
Im Ordner ViewModels**

Für alle Seiten, die angezeigt werden sollen müssen ViewModels angelegt werden.

Es werden insgesamt 3 Klassen für jede Felddatenklasse des Typs EventSeries, Event, Specimen und IdentificationUnit angelegt, sowie 2 Klassen für alle weiteren editierbaren Klassen. Für Read-Only-Klassen werden keine eigenen Seiten zur Ansicht und somit auch keine ViewModels benötigt.

Darüber hinaus werden für folgende Prozesse ViewModels angelegt:

* UserProfil
* Kartenauswahl
* Kartenansicht
* Multimediaaufnahme
* MultimediaAnsicht
* OberflächenCustomizing

Basisklasse [Felddatenklasse]VM z.B.

public class EventVM : ElementVMBase<Event>

Enthält:

* Property Model mit Verweis auf das Objekt der zugrunde liegenden Klasse
* Property Description mit anzuzeigendem Text
* Property Icon für das Icon

Die ViewModels für Seiten leiten alle direkt oder indirekt von PageVMBase ab.

Dieses enthält:

* Messenger:   
  Die IMessageBus implementierung, die verwendet werden soll
* ActivationObservable

Observable, die immer feuert, wenn die Seite besucht oder verlassen wird, zu der das ViewModel gehört.

Klasse zum Editieren Edit[Felddatenklasse]VM z.B.:

public class EditEVVM : EditPageVMBase<Event>

Enthält:

* Property Current mit Verweis auf das VM der zugrunde liegenden Klasse
* Property IsEditable, das angibt, ob die Seite aktuell im Bearbeitungsmodus ist.
* ReactiveCommands ToggleEditable, Save und Delete
* Editierbare Felder der zugrunde liegenden Klasse
* CanSaveSubject zum erlauben/verhindern der Bearbeitung des aktuellen Objekts
* Methode UpdateModel() die beim Speichern aufgerufen wird, um die vorgenommen Änderungen auf die Model instanz zu übertragen

Klasse zur Auswahl von Kindklassen (nur ES, EV, CS , IU) View[Felddatenklasse]VM z.B.

public class ViewEVVM : ViewPageVMBase<T>

Enthält:

* Property Current mit Verweis auf das VM der zugrunde liegenden Klasse
* Liste von ViewModells der Kindklasse IList<KindklasseVM>
* ReactiveCommand Add

# Seiten- ViewModels im ViewModelLocator registrieren

**Im DiversityPhone Projekt,   
Im Ordner ViewModels**

* IOC Constructor Lambda
* Zugriffs-Property:

public ViewEVVM ViewEV { get { return \_ioc.Resolve<ViewEVVM>(); } }

# View erstellen

**Im DiversityPhone Projekt,   
Im Ordner View**

PortraitPage Template

Template in der Templateklasse erstellen/wählen

🡪DataContext der Seite per Binding auf das ViewModel setzen:

DataContext="{Binding Source={StaticResource Locator}, Path=ViewEV}"

# Seite registrieren

**Im DiversityPhone Projekt,   
Im Ordner Services**

🡪Zum Page enum hinzufügen

🡪Im NavigationService eintragen (Navigate() und evtl. noch mehr Message Listener für Select oder Edit Nachrichten)

# Dokumentieren

Anpassungsschritte der neu eingefügten Klassen im Dokumentationsfile

.docx hinterlegen.