BeerCompanion

# konzept

Für jedes getrunkene Bier wird ein Counter erhöht. Der Benutzer kann dann Gruppen hinzufügen, und seinen Counter mit der von Anderen vergleichen.

# umsetzung:

App für WP, Android und iOS. Sehr viel gemeinsamer Quellcode mit Xamarin in C#. Web API in PHP mit mySQL die alle Daten mit einer Datenbank abgleicht.

# daten:

Grundsätzlich gibt es vier Objekte:

Drinker: Der Trinker, der der das Bier trinkt  
DrinkerCycle: Eine Gruppe von Trinkern  
UserInformation: Die Informationen des aktiven Benutzers  
Beer: das ist klar

Das Datenbankschema ist leicht anders, spielt aber keine Rolle für die Entwicklung der Platformspezifischen Apps.

# mvvm-pattern:

Das MVVM Pattern ist beschdeee: Jede Seite bekommt grundsätzlich ein Objekt zugewiesen. Alle Informationen die auf der Seite dargestellt werden, sind Properties dieses Objetes (dem ViewModel). Alle Anweisungen (Buttons etc) auf der Seite werden durch ICommands direkt mit dem ViewModel verknüpft, dass dann die geforderten Aktionen ausführt. Verändert das ViewModel nun Properties (= weist neue Werte zu) wird automatisch ein Event, der PropertyChanged Event aufgerufen. Auf Windows Phone Masterrace werden dann ebenfalls automatisch die Werte auf der Seite angepasst.

Die Ganze Logik der Anwendung inkl. der Anweisungen wird somit abstrahiert. Das schwierigste bei der Umsetzung auf den andren Platformen wird sein, dieser PropertyChanged Event korrekt zu verarbeiten.

# quellcode:

Der Code ist in mehreren Projekten organisiert. Ich werde kurz einzeln durchgehen. Alle jeweils nachfolgenden Projekte erben von den jeweils davor gehenden.

Nicht erwähnt werden die Projekte im Order Test, die sind dazu da, die API und die Anwendung zu testen. Sie sind für die Entwicklung plattformspezifischer Apps nicht relevant.

## Common: Alle Framework spezifischen Objekte.

**Singleton:** Ein Pattern, am besten kurz Nachschlagen.   
**Logging:** Hilfsobjekte für einfaches Logging

## Data: Communikation mit der API mittels spezieller Objekte (den Entities)

**Entities:** Alle Objekte die zur Kommunikation verwendet werden  
**Enums:** sollte klar sein, comon **Services:** Den DataService, der die Kommunikation mit der API erledigt.

## Business: Der eigentliche Quellcode.

**Converter:** Übernehmen die Konvertierung der Objekte von der API in Objekte, die im Business vernünftig gebraucht werden können. (Entities <-> Models)  
**Enums:** …  
**Models:** Das Gegenstück zu den Entities; alle Objekte die in der View und im Business gebraucht werden  
**Repository:** Übernehmen das Datenmanagement; stellen der Anwendung alle benötigten Daten als Models zur Verfügung.  
**Services:** Platformspezifische Implementationen als Interfaces

## View: Übernimmt die Darstellung mittels MVVM

**Enums & Models & Services:** Nur in View gebraucht Objekte  
**Utils:** Sofern Code von ViewModels ausgelagert werden kann, bitte dort hinein  
**ViewModels:** Übernehmen die Darstellung der Informationen. Bitte beachte das ProgressViewModel das Informationen enthält, ob ein Hintergrundprozess am laufen ist.

## Presentation: Platformspezifischer Code (native)

Jede Platform muss folgendes bereitstellen:

4 Seiten:

Main: Darstellung Counter und Auflistung aller Gruppen  
Wizard: Wird beim ersten Start der Anwendung. Nach dem Abschluss muss der Name und die persönliche Farbe festgelegt sein.  
DrinkerCycle: Darstellung einer einzelnen Gruppe: Anzeige Statistiken und alle Mitglieder. Bei der Darstellung der Mitglieder müssen neue bestätigt werden können, und bestehende wieder entfernt werden können  
Settings: Einstellungen

2 Services:

IStorageService und IInteractionService müssen implementiert werden. Es ist alles schön auskommentiert ☺

2 Packages:

Mvvm light von Laurent Buignon (so ungefähr)  
Newtonsoft.JSON

1 Object:

Kreiere ein ViewModelLocator wie im Presentation.WinUniversal im Ordner ViewModel. Registriere die platformspezifischen Services gegen die Interfaces wie im Beispiel. Die Implementierung des INavigationService und des IDialogService kommen vom Mvvm Light Package.

Beim INavigationService vergiss nicht, mittels .Configure alle Seiten zu registrieren. Ein Beispiel wie das geht findest du im NavigationHelper im Ordner Platform.

# tl;dr

Alle Wichtige steht im Punkt direkt oben dran, ausserdem ist es empfehlenswert sich den mvvm-pattern Teil durchzusehen.

# schlusswort

Danke für die Hilfe, viel Spass.

Wenn ihr am Bugfixen seid: Ihr werdet keine finden alles was Business oder weiter nach unten betrifft. Wenn also ein Fehler auftritt, der wahrscheinlich damit zu tun hat, hat er wahrscheinlich nichts damit zu tun.

Schreibfehler gibt es fiele, darfst sie behalten!