

APP-SCHIENE - ABGABE SPRINT 3

INHALT

1	Einleitung	1
2	Unit Testing	1
3	Xamarin UI Test Framework.....	2

1 EINLEITUNG

Da in den vorherigen Sprints die kompletten Funktionalitäten implementiert wurden, wurde der 3.Sprint verwendet um Einblicke in das Testen von Apps zu sammeln.

Geplant waren einerseits normale Tests für die MVVM-Implementierung und das Testen der UI selbst.

In Zusammenhang mit Xamarin und C# hat man einerseits die Möglichkeit normale Unit Tests durchzuführen, aber zusätzlich mit dem Xamarin UI Test Framework hat man die Möglichkeit auch die UI – interaktiv zu testen.

2 UNIT TESTING

Das Testen der MVVM-Architektur war komplizierter wie gedacht. Problem war der NavigationService der von jedem ViewModel im Konstruktor gebraucht wird. Dieser wird aber erst zur Laufzeit injected und kann nicht manuell in den Test-Cases injected werden.

```
public LoginViewModel(IDataAccessService _DataAccessProxy, INavigationService _NavigationProxy)
{
```

3 XAMARIN UI TEST FRAMEWORK

Beim UI Testen verlief alles reibungslos.

Xamarin bietet die Möglichkeit UI Test Projekte zu erstellen, in diesen können User-Inputs und Benutzerinteraktionen simuliert werden. Diese Simulation bzw. die Tests selbst können LIVE am Smartphone / Emulator mitverfolgt werden.

Beispiel Testfall für einen inkorrekten Login:

```
[Test]
[Category("Login")]
public void LoginIncorrect()
{
    app.EnterText("txtUsername", "wat");
    app.EnterText("txtPassword", "watf");

    app.DismissKeyboard();
    app.Tap("btnLogin");

    app.WaitForElement("Error", "No errors info occured", TimeSpan.FromSeconds(5));

    var result = app.Query("Error");
    Assert.IsTrue(result.Any(), "LoginIncorrect test failed");
}
```

Jedes UI-Element, dass angesprochen wird bekommt für das Testen extra eine sogenannte „AutomationId“ über diese erfolgt dann der Befehlsaufruf („Tap“ für Klick, „EnterText“ für Texteingabe und noch andere wie .ScrollUp, ... etc.)

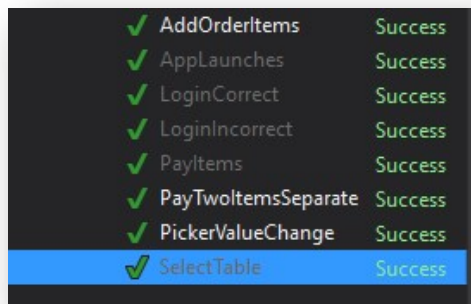
```
<Button AutomationId="btnLogin" Grid.Row="7" x:Name="btnLogin"/>
```

```
app.Tap("Bill");
app.WaitForElement("Pay", "Pay View konnte nicht geöffnet werden", TimeSpan.FromSeconds(5));

result = app.Query("Pay");
Assert.IsTrue(result.Any(), "Cannot open pay view");
```

Wie im Beispiel zu sehen wird hier auf den Button Bill geklickt und 5 Sekunden gewartet, bevor dann überprüft wird, ob sich die nächste View tatsächlich geöffnet hat.

Auf diese Art kann man verschiedenste Szenarien (Geschäftsfälle) durchspielen.



✓ AddOrderItems	Success
✓ AppLaunches	Success
✓ LoginCorrect	Success
✓ LoginIncorrect	Success
✓ PayItems	Success
✓ PayTwoItemsSeparate	Success
✓ PickerValueChange	Success
✓ SelectTable	Success

Da das Testen ja am Smartphone / Emulator simuliert wird, können die Testcases auch teilweise als Showcases für Präsentationen verwendet werden. Die Vorstellung beginnt mit Klick auf „Run unit tests“.