tgm
Die Schule der Technik

JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

Systemtechnik Labor

5BHIT 2017/18

GK10.2 "Mobile Dienste"

Laborprotokoll

Marvin Ertl

12. April 2018

Bewertung:		Version:	1.0
Betreuer:	Michael Borko	Begonnen:	21.03.2018
		Beendet:	12.04.2018

Inhaltsverzeichnis

1 Einführung

Diese Übung gibt einen Einblick in Entwicklungen von mobilen Applikationen.

1.1 Ziele

Das Ziel dieser Übung ist eine Anbindung einer mobilen Applikation an ein Webservices.

Die Anbindung soll mit Hilfe eines RESTful Webservice umgesetzt werden.

1.2 Voraussetzungen

- Grundlagen einer höheren Programmiersprache und XML
- Grundlegendes Verständnis über Entwicklungs- und Simulationsumgebungen
- Verständnis von RESTful Webservices
- Download der entsprechenden Entwicklungsumgebung

1.3 Aufgabenstellung

Es ist eine mobile Anwendung zu implementieren, die sich an das Webservice aus der Übung GK9.3 "Cloud-Datenmanagementänbinden soll. Dabei müssen die entwickelten Schnittstellen entsprechend angesprochen werden.

Es ist freigestellt, welche mobile Implementierungsumgebung dafür gewählt wird. Empfohlen wird aber eine Implementierung auf Android.

1.4 Bewertung

- Gruppengrösse: 1 Person
- Anforderung "**überwiegend erfüllt**"
  - Dokumentation und Beschreibung der angewendeten Schnittstelle
  - Anbindung einer mobilen Applikation an die Webservice-Schnittstelle
  - Registrierung von Benutzern
  - Login und Anzeige einer Willkommensnachricht
- Anforderungen "**zur Gänze erfüllt**"
  - Simulation bzw. Deployment auf mobilem Gerät
  - Überprüfung der funktionalen Anforderungen mittels Regressionstests

2 Implementierungsumgebung

Als Implementierungsumgebung wurde die Universelle Windows-Plattform (UWP) gewählt, um einen Einblick in eine neue Umgebung zu bekommen.

Die fertige Übung ist unter folgendem Link abrufbar: `https://github.com/mertl-tgm/GK10.2`

2.1 UWP

Die UWP bietet die Möglichkeit eine Anwendung nur einmal zu entwickeln und auf allen Windows Geräten auszuführen. Dadurch kann die, in dieser Übung, entwickelte App auf vielen unterschiedlichen Geräten verwendet werden, unter anderem auf mobilen Geräten, wie Smartphones und Tablets, PCs, HoloLens und viele weitere. Im Fall dieser Übung wurde die Anwendung aber nur mit PCs und mobilen Geräten getestet.

Um diese Universellen Apps zu ermöglichen, bietet UWP die selbe Kern-API auf allen Geräten an. Dadurch kann auch ein App Store als einheitliche Vertriebsplattform auf unterschiedliche Geräteformen verwendet werden.

Abbildung 1: Darstellung der UWP

Dennoch kann für jedes Gerät spezieller Code geschrieben werden sodass die Anwendung ein unterschiedliches Verhalten je nach Gerät aufweist. Außerdem bietet UWP universelle Steuerelemente, womit eine Benutzeroberfläche entwickelt werden kann, welche sich an die Auflösung, DPI-Dichte, Eingabeart von Geräten anpasst.

UWP ermöglicht das Entwickeln mittels Visual C++, C#, Visual Basic und JavaScript, wobei mit den Programmiersprachen Visual C++, C# und Visual Basic XAML verwendet werden kann um die Benutzeroberfläche zu definieren. [7]

XAML (Extensible Application Markup Language) sind XML Dateien, welche XAML Objekte initialisieren und Eigenschaften von diesen Objekten festlegen. Hierfür gibt es einen eigene XAML-Namespaces, welcher festlegt, wie die Informationen interpretiert werden. [8]

Bei dieser Übung wurde C# mit XAML zur Umsetzung der Aufgabe verwendet.

2.2 Visual Studio

Als am besten geeignete Tool erweist sich Visual Studio, welches eine optimale Unterstützung für die Entwicklung von UWP Apps bietet. Bei dieser Übung wurde Visual Studio 2017 Enterprise Edition verwendet, wobei darauf zu achten ist, dass zur Verwendung von Emulatoren für andere Geräte, wie zum Beispiel Smartphones, HoloLens, ... , Visual Studio 2015 installiert sein muss. Zwar können die Emulatoren für Visual Studio 2017 gedownloadet werden, diese werden aber im Editor dann nicht angezeigt, wenn Visual Studio 2015 nicht installiert ist.

Im Visual Studio Installer kann im Reiter "Einzelne Komponenten" Emulatoren für die gewünschte Windows 10 Version gedownloadet werden.

Abbildung 2: Installieren von Emulatoren

In Visual Studio kann anschließend die App auf den gewünschten Gerät ausgeführt werden.

3 REST - Schnittstelle

Als REST - Schnittstelle wurde die entwickelte Anwendung für die Übung GK9.3 verwendet. Diese wurde nur leicht angepasst, sodass JSON Objekte gesendet werden.

3.1 Deployen

Die Anwendung ist im Unterverzeichnis "Cloud-Datenmanagement-Json" zu finden. Diese kann einfach deployed werden, da es sich um ein mit Maven verwaltetes Programm handelt.

Mittels folgendem Befehl wird die Anwendung auf den Port 8080 lokal gestartet.

```
1 mvn tomcat7:run-war
```

Für diese Übung wurde das Projekt auf einem Server, welcher unter der IP-Adresse 37.252.185.24 erreichbar ist, aufgesetzt.

3.2 Verwendung

Die REST-Schnittstelle bietet 3 Funktionen, das Anmelden eines neuen Benutzers, den Login mit einem bestehenden Account und das Löschen eines Benutzerkontos.

Bei der Registrierung muss der Vorname, der Nachname, die E-Mail-Adresse, ein Passwort sowie eine Passwort Wiederholung eingegeben werden. Diese Eingaben werden als Parameter an die Schnittstelle gesendet, nachfolgend ist ein Beispielaufruf zu sehen:

```
http://37.252.185.24:8080/ertl/register?vname=Marvin&nname=Ertl&email=mertl@student
↪ .tgm.ac.at&pw=abc123&pwagain=abc123
```

Die Schnittstellen für das Löschen und Einloggen von Benutzer sind ähnlich aufgebaut, beim Einloggen muss die E-Mail sowie das Passwort als Parameter mitgesendet werden und beim Löschen muss die E-Mail-Adresse mitgesendet werden.

```
http://37.252.185.24:8080/ertl/login?email=mertl@student.tgm.ac.at&pw=abc123

http://37.252.185.24:8080/ertl/delete?email=mertl@student.tgm.ac.at
```

Die Rückgabe ist dabei immer ein JSON Objekt, welches aus einem Array mit 2 Werten besteht. Der erste Wert gibt an, ob der Befehl erfolgreich (success) oder nicht erfolgreich (error) war. Der zweite Wert gibt bei einem success weitere Informationen mit, bei einem error wird die Fehlermeldung mitgeliefert.

Nachfolgend ist das Antwort JSON, welches bei einer erfolgreichen Registrierung zurückgeliefert wird.

```
["success","Marvin#Ertl#mertl@student.tgm.ac.at"]
```

4 Mobile Applikation

Zum Entwickeln der Anwendung wurde in Visual Studio ein UWP Projekt erstellt, dieses beinhaltet bereits die App Klasse, welche sich um das Starten/Beenden/Pausieren der Anwendung kümmert. Diese Klasse wird allen drei weiteren Klassen übergeben. Die App besteht insgesamt aus drei Ansichten, den Register Page, dem Login Page und der MainPage, diese Ansichten erben alle von der ”Page” Klasse. Erfolgreiches registrieren sowie einloggen führt zur MainPage, auf welcher der Benutzer begrüßt wird und sein Vorname, Nachname sowie seine E-Mail-Adresse anzeigt. [2]

Dabei zu beachten ist, dass jede Ansicht 2 Dateien hat, eine XAML Datei, welche die Benutzeroberfläche definiert und einer C# Datei, welche den User Input verwaltet sowie die Ansicht.

Zwischen den einzelnen Ansichten wird mittels der ”Navigate” Methode gewechselt, beim Wechsel kann man nur ein Objekt mitgeben, aus diesem Grund wurde das Objekt ”Param” angelegt, dieses beinhaltet die Instanz der App Klasse sowie einen String. Dieser wird verwendet um die Rückmeldung von der REST-Schnittstelle an die nächste Ansicht weiterzugeben.

```
public class Param
{
    public App App { get; set; }
    public string Text { get; set; }
}
```

Eine Page reagiert mittels der Methode ”OnNavigatedTo” auf Parameter, welche übergeben werden. Diese Methode überschreibt dabei eine Methode aus der Page Klasse. Dies geschieht wie folgt:

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    this.param = new Param();
    this.param.App = (App)e.Parameter;

}
```

4.1 Register

Als erstes wurde die Registrierungsansicht mittels den Designer in Visual Studio erstellt.

Dieser erlaubt es einfach aus der Toolbox Objekte mittels Drag&Drop eine Benutzeroberfläche zu gestalten. Wobei jederzeit zwischen den Skalierung auf unterschiedlichen Geräten gewechselt werden kann. Somit kann überprüft werden, wie die Benutzeroberfläche auf diesen Geräten ausschauen würde.

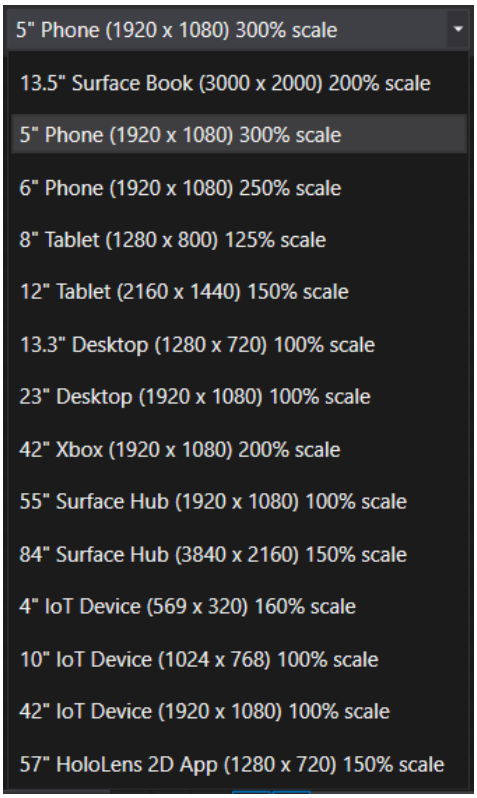

Abbildung 3: Auswahl der Skalierung

Die erstellte Oberfläche sieht im Designer wie folgt aus:

Abbildung 4: Registrierungsoberfläche im Designer

Anschließend kann entweder über den Design oder direkt im XAML Code einige Änderungen vorgenommen werden. Zum Beispiel ein Event auf die Button zu legen. Nachfolgend ist ein Ausschnitt aus der XAML Datei, welcher den Button "Registrieren" definiert, wobei bei einem Click die Methode "RegisterUser" in der C# Datei aufgerufen wird.

```
<Button Grid.Row="0" Grid.Column="1" Content="Registrieren" Click="RegisterUser"
↪  HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Margin="10"/>
```

Die Methode, welche auf das Click Event vom Button reagiert sieht wie folgt aus:

```
private async void RegisterUser(object sender, RoutedEventArgs e)
{
    // ...
}
```

In dieser Methode werden als erstes alle eingegebene Werte aus den Feldern ausgelesen und zu einem String zusammengesetzt, welcher die Parameter für die "register" Methode beinhaltet. Wobei das Passwort mittels SHA512 gehasht wird. [3]

```
public string HashPW(string pw)
{
```

```
        HashAlgorithmProvider objAlgProv =
        ↪ HashAlgorithmProvider.OpenAlgorithm(HashAlgorithmNames.Sha512);
        CryptographicHash objHash = objAlgProv.CreateHash();

        IBuffer buffpw = CryptographicBuffer.ConvertStringToBinary(pw,
        ↪ BinaryStringEncoding.Utf16BE);
        objHash.Append(buffpw);
        IBuffer buffHash = objHash.GetValueAndReset();

        return CryptographicBuffer.EncodeToBase64String(buffpw);
    }
```

Diese Methode erwartet einen String, dieser wird mittels SHA512 gehasht und anschließend zurückgeliefert. Dies wird bereits am Client durchgeführt, um zu verhindern, dass das Passwort während der Übertragung abgefangen wird.

Nun wird ein Request an den Server mit den Parameter gesendet und auf eine Antwort gewartet, falls keine Verbindung mit den Server hergestellt werden kann, wird eine Fehlermeldung ausgegeben.

```
    string param = "vname=" + this.vname.Text + "&nname=" + this.nname.Text + "&email=" +
    ↪ this.email.Text + "&pw=" + this.param.App.HashPW(this.pw.Password) + "&pwagain="
    ↪ + this.param.App.HashPW(this.pwagain.Password);

    Uri geturi = new Uri("http://37.252.185.24:8080/ertl/register?" + param);
    string response = "";
    try
    {
        System.Net.Http.HttpClient client = new System.Net.Http.HttpClient();
        System.Net.Http.HttpResponseMessage responseGet = await client.GetAsync(geturi);
        response = await responseGet.Content.ReadAsStringAsync();
    }
    catch (System.Net.Http.HttpRequestException)
    {
        this.errormessages.NavigateToString("Fehler beim Verbinden mit den Server.");
        return;
    }
```

Als Antwort wird ein JSON Objekt zurückgesendet, diese ist wie bereits oben beschrieben definiert. Zum Verarbeiten des Objektes wurde auf die Library Newtonsoft.Json zurückgegriffen, diese kann einfach mittels dem NuGet Package Manager installiert werden.

Als erstes wird unterschieden ob es ein success oder error war, war die Registrierung erfolgreich wird der Benutzer auf die MainPage weitergeleitet, falls ein Fehler aufgetreten ist, wird der Benutzer darüber informiert.

```
Newtonsoft.Json.Linq.JArray result =
↪ (Newtonsoft.Json.Linq.JArray)JsonConvert.DeserializeObject(response);
if (((string)result[0]).Equals("error"))
{
    this.errormessages.NavigateToString((string)result[1]);
    return;
```

```
6   }
7   else if (((string)result[0]).Equals("success"))
8   {
9       this.errormessages.NavigateToString(response);
10
11      this.param.Text = (string)result[1];
12      Frame.Navigate(typeof(MainPage), this.param);
13      return;
14  }
```

4.2 Login

Bei der Login Ansicht wurde nach dem selbem Prinzip vorgegangen, nur wird beim Anmelden nur das Passwort und die E-Mail-Adresse benötigt. Wie jede Ansicht bietet auch diese ein Menü, diese kann über die drei horizontalen Balken erweitert werden, sodass neben den Buttons auch noch die Bezeichnungen stehen. Diese Button ermöglichen das Wechseln zwischen der Registrierung und dem Login, sowie einen Button zum Beenden der Anwendung.

Bei einer Fehlermeldung wird diese unterhalb der Buttons in einer WebView dargestellt. Mittels der Methode "NavigateToString" kann der Text in der View angezeigt werden, wobei dieser als HTML Code interpretiert wird. Dies wurde gewählt, da mehrere Fehlermeldung von Server nur mittels "
" getrennt zurückgesendet werden.

```
1       this.errormessages.NavigateToString("Fehlermeldung");
```

Dies sieht wie folgt aus:

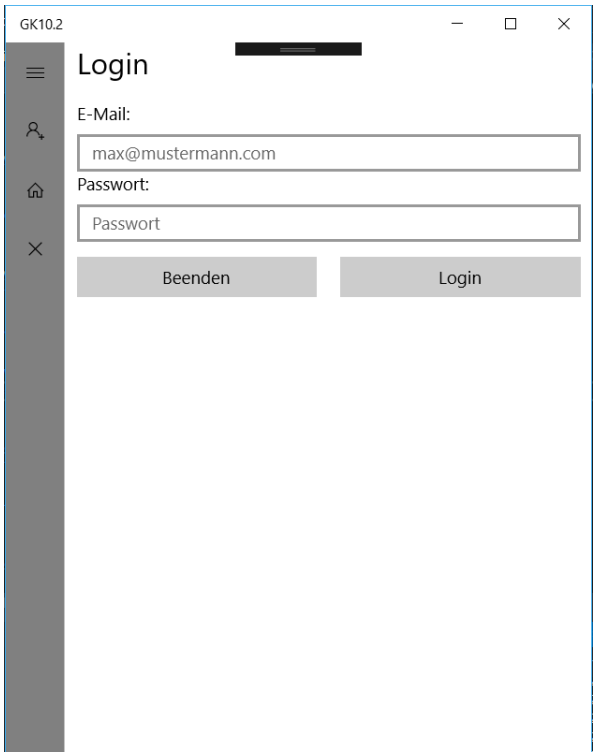

Abbildung 5: Login Ansicht auf Windows 10

Ansonsten ist diese Ansicht gleich aufgebaut, bei der Anfrage an die REST-Schnittstelle wird nun nur das Passwort sowie die E-Mail-Adresse übergeben, ist das Anmelden erfolgreich wird der Benutzer zur MainPage weitergeleitet.

```
1       string param = "email=" + this.email.Text + "&pw=" +
        ↪  this.param.App.HashPW(this.pw.Password);
2
3       Uri geturi = new Uri("http://37.252.185.24:8080/ertl/login?" + param);
```

4.3 Mainpage

Ist die Ansicht, welcher der Benutzer bei einer erfolgreichen Registrierung oder Anmeldung sieht. Wurde ein Emulator und Visual Studio 2015 installiert, hat der Entwickler auch die Möglichkeit die Anwendung in einem Emulator zu starten.

Die MainPage hat eine andere Menubar im Vergleich zum Login oder Registrieren. In dieser hat man die Möglichkeit sich abzumelden, den Account zu löschen oder die Anwendung zu beenden.

Hierfür muss ein Request an die Methode ”delete” gesendet werden, welche als Parameter die E-Mail-Adresse des Accountes hat.

```
1       string param = "email=" + this.emailAdr;
2
3       Uri geturi = new Uri("http://37.252.185.24:8080/ertl/delete?" + param);
```

Dies sieht dann mit einem erfolgreichen Login wie folgt aus:

Abbildung 6: Ausführen der Anwendung im Emulator

4.4 Deployen

Nach einigem Testen kann nun die Anwendung deployed werden, hierfür muss die App als erstes gepackt werden. Dies geschieht über einen Rechtsklick auf das Projekt in Visual Studio. Dort gibt es die Auswahlmöglichkeit ”Store” mit den Unterpunkt ”Create App Package”. Wurde dies ausgewählt werden noch einige Informationen wie zum Beispiel Version, Name, verlangt. Anschließend kann entschieden werden, ob die App gleich direkt in den Store geladen wird, in diesem Fall wurde dies nicht gemacht.

Abschließend muss noch die Plattform ausgewählt werden, auf welcher die App läuft für Smartphones wird ARM benötigt für Desktop x86 oder x64. Es können aber auch alle Plattformen auf einmal ausgewählt werden.

Nun erhält man einen Ordner mit einigen Dateien, die wichtigste davon ist die Datei mit der Endung ”.appxbundle”, dabei handelt es sich um die Installationsdatei. [5] Um die Entwicklerversion zu installieren, gibt es ein PowerShell Script, welches ausgeführt werden muss. Ansonsten kann mittels einem Doppelklick auf die Datei ”.appxbundle” die Installation gestartet werden, dies sieht wie folgt aus:

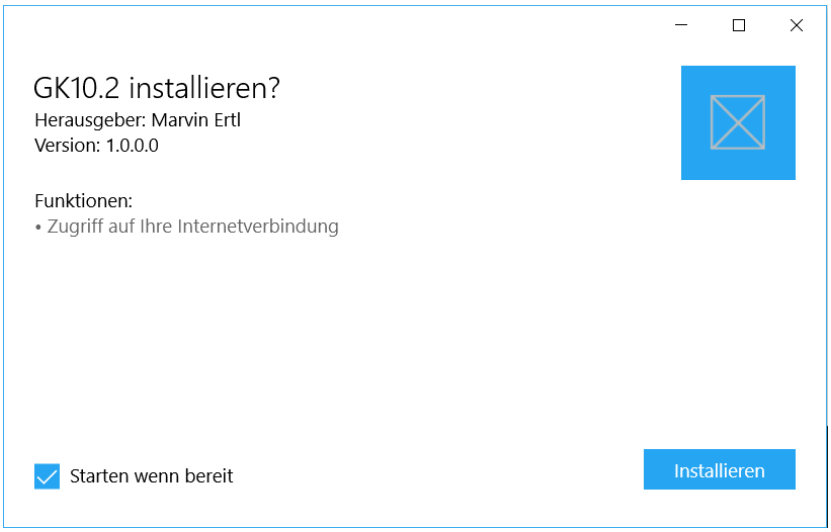

Abbildung 7: Installieren einer App

Darauf zu achten ist, dass unter den Windows Einstellungen in der Kategorie ”Für Entwickler” erlaubt wird, Apps aus anderen Quellen außer dem App-Store zu erlauben.

5 Testen

Um das Testen der Anwendung möglichst einfach zu machen und Regressiv zu wiederholen, wurde auf UI Automation von UWP Apps gesetzt. Dafür muss als erstes der WinAppDriver installiert werden. Dieser muss während des Testen laufen. [4]

5.1 UI Automation

Microsoft UI Automation ist ein Framework um auf Elemente am Desktop zuzugreifen um die Benutzeroberfläche zu bedienen. Aus diesem Grund wurde ein zweites Projekt erstellt, welches sich um das Testen

der Anwendung kümmert. Mittels diesem Framework können nicht nur eigene Anwendungen getesten werden, sondern auch Anwendungen von Drittanbieter oder Windows eigene Anwendungen. [6]

Im Repository des Win App Drivers gibt es auch einige Beispiel, auf Basis des Beispiels für den Windows Rechner wurde eigene Testfälle entwickelt. [4]

5.2 Testaufbau

Das Projekt besteht aus zwei Klassen, eine Klasse GKSession, diese startet die Anwendung und die Klasse ScenarioStandard beinhaltet die Testfälle.

Um die Anwendung starten zu können, muss die App ID der Anwendung gefunden werden, diese kann in den App Package Informationen gefunden werden. [1] Nun kann die Anwendung wie folgt gestartet werden und eine Session erstellt werden:

```
                DesiredCapabilities appCapabilities = new DesiredCapabilities();
appCapabilities.SetCapability("app", AppId);
appCapabilities.SetCapability("deviceName", "WindowsPC");
session = new WindowsDriver<WindowsElement>(new Uri(WindowsApplicationDriverUrl),
↪  appCapabilities);
```

Nun kann in der Scenario Klasse Testfälle definiert werden, diese müssen mittels "[TestMethod]" gekennzeichnet sein. Mittels der Methode "FindElementByName" können nun die Buttons bzw. Eingabefelder gefunden werden und daraufgeklickt bzw. Werte eingegeben werden. Schlussendlich wird überprüft, ob nun "Herzlich Willkommen" in der Anwendung angezeigt wird.

```
    session.FindElementByName("Vorname:").Click();
    session.FindElementByName("Vorname:").SendKeys("Marvin");
    session.FindElementByName("Nachname:").Click();
    session.FindElementByName("Nachname:").SendKeys("Ertl");
    session.FindElementByName("E-Mail:").Click();
    session.FindElementByName("E-Mail:").SendKeys("abc@def.com");
    session.FindElementByName("Passwort:").Click();
    session.FindElementByName("Passwort:").SendKeys("12345");
    session.FindElementByName("Passwort wiederholen:").Click();
    session.FindElementByName("Passwort wiederholen:").SendKeys("12345");
    session.FindElementByXPath("//Button[@Name='Registrieren']").Click();

    Assert.AreEqual("Herzlich Willkommen", session.FindElementByName("Herzlich
    ↪  Willkommen").Text);
```

Anschließend wurde mit einigen Testfällen überprüft ob die Anwendung richtig funktioniert bzw. ob bei einem erneuten Ausführen die Testfälle auch erfolgreich sind. Es wurde überprüft ob sich der Benutzer Registrieren/Anmelden kann und ob der Account gelöscht werden kann. Weiters wurden einige Fehlermeldungen überprüft wie zum Beispiel ein falsche Passwort, einen Benutzer mit bereits vorhanden E-Mail-Adresse registrieren.

Diese Test könnten nun auch mittels einem Continuous Integration Service verbunden werden, sodass bei jedem Build automatisch diese Test ausgeführt werden. Das Ausführen der Testfälle könnte auch von einem anderen Server aus gestreamt werden.

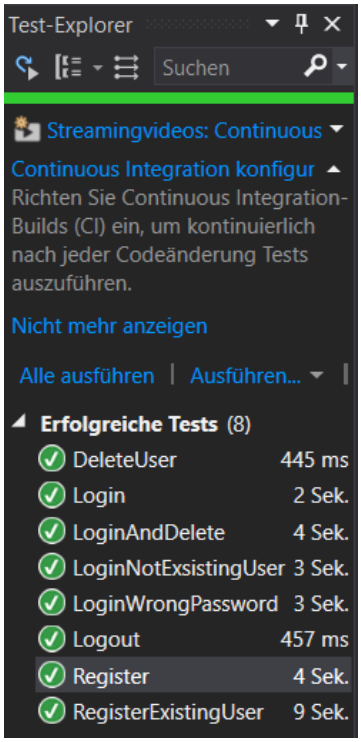

Abbildung 8: Test Explorer in Visual Studio

Literaturverzeichnis

[1] *Identity (Windows 10) - UWP app developer*. [Online; accessed 11. Apr. 2018]. 2018. URL: https://docs.microsoft.com/de-de/uwp/schemas/appxpackage/uapmanifestschema/element-identity.

[2] *Learn how to create a*. [Online; accessed 11. Apr. 2018]. 2018. URL: https://docs.microsoft.com/en-us/windows/uwp/get-started/create-a-hello-world-app-xaml-universal.

[3] *MACs, hashes, and signatures - UWP app developer*. [Online; accessed 11. Apr. 2018]. 2018. URL: https://docs.microsoft.com/en-us/windows/uwp/security/macs-hashes-and-signatures.

[4] *Microsoft/WinAppDriver*. [Online; accessed 11. Apr. 2018]. 2018. URL: https://github.com/Microsoft/WinAppDriver.

[5] *Packaging UWP apps - UWP app developer*. [Online; accessed 11. Apr. 2018]. 2018. URL: https://docs.microsoft.com/en-us/windows/uwp/packaging/packaging-uwp-apps#before-packaging-your-app.

[6] *UI Automation Overview (Windows)*. [Online; accessed 11. Apr. 2018]. 2018. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/ee684076(v=vs.85).aspx.

[7] *What's a Universal Windows Platform (UWP) app? - UWP app developer*. [Online; accessed 11. Apr. 2018]. 2018. URL: https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide.

[8] *XAML overview - UWP app developer*. [Online; accessed 11. Apr. 2018]. 2018. URL: https://docs.microsoft.com/en-us/windows/uwp/xaml-platform/xaml-overview.

Abbildungsverzeichnis