Chapter 12
WCF – Windows Communication Foundation

WCF – Windows Communication Foundation

1. Was versteht man darunter?

Windows Communication Foundation (WCF) ist ein Framework zum Erstellen dienstorientierter Anwendungen. Mit WCF können wir Daten als asynchrone Nachrichten von einem Dienstendpunkt an einen anderen senden. Ein Dienstendpunkt kann Teil eines fortwährend verfügbaren von IIS gehosteten Diensts oder ein in einer Anwendung gehosteter Dienst sein. Ein Endpunkt kann ein Client eines Diensts sein, der Daten von einem Dienstendpunkt anfordert. Die Nachrichten können einfach aus einem als XML gesendeten einzelnen Zeichen oder Wort oder aus einem komplexen Strom binärer Daten bestehen. Einige Beispielszenarios enthalten Folgendes:

- Ein sicherer Dienst zur Verarbeitung von Geschäftstransaktionen.
- Ein Dienst, der aktuelle Daten für andere Dienste bereitstellt, z. B. für Netzwerkverkehrberichtsdienste oder andere Überwachungsdienste.
- Ein Chatdienst, mit dem zwei Personen in Echtzeit kommunizieren oder Daten austauschen können.
- Etc.

Während das Erstellen solcher Anwendungen vor dem Vorhandensein von WCF möglich war, vereinfacht WCF die Entwicklung von Endpunkten so einfach wie nie zuvor. Zusammengefasst ist WCF so konzipiert, dass es einen verwaltbaren Ansatz zum Erstellen von Webdiensten und Webdienstclients bietet.

2. Erstellen einer Applikation mit Restful WCF API

Im Allgemeinen gibt es mehrere Möglichkeiten, ein WCF zu konfigurieren, da WCFs in einer .config-Datei durch eine Menge von Parametern definiert werden kann. In diesem Konzept wird eine Möglichkeit davon vorgestellt.

2.1. Erstellen einer Restful WCF API

A) Erstelle eine WCF Service Applikation unter VS 2019: *Neues Projekt > WCF Service Applikation (Abb. 1) – Framework: .NET Framework 4*

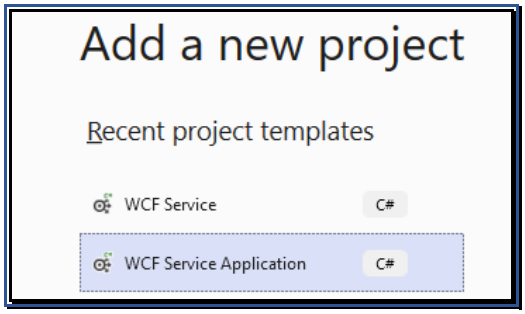

Abb. 1: Erstellen eines Projektes

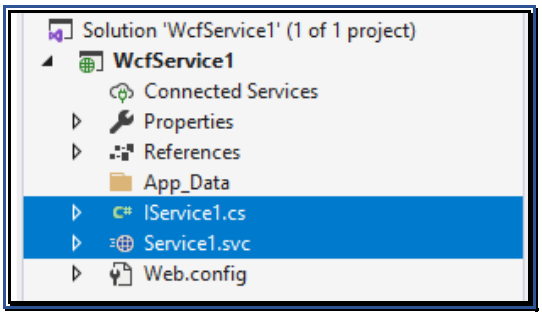

Abb. 2: Löschen der Default-Services

B) Mit der Erstellung der Applikation werden automatisch 2 Files für ein WCF-Service und ein Interface-File erstellt. Lösche diese (Abb. 2)! Erstelle die eigenen WCF und Interface Services:

Rechtsklick auf das Projekt WCFService1und füge ein neues WCF-Service hinzu (Abb. 3). Benenne es (hier: `RestServiceImpl`.svc: *Recht-Klick > Hinzufügen > WCF-Service*

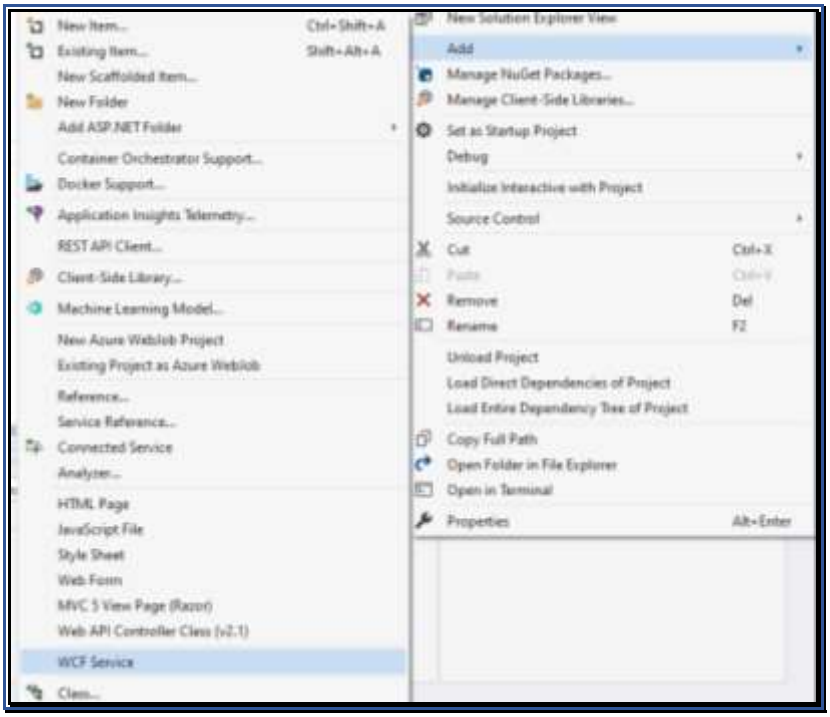

Abb. 3: Einfügen eines leeren WCF-Dienstes

Wir wollen eine API erstellen, die die Daten sowohl in XML- als auch in JSON-Format übermitteln. Im Interface **`IRestServiceImpl`** ist deshalb folgender Code hinzuzufügen. In dieser Datei muss die Library `System.ServiceModel.Web` hinzugefügt werden, damit die Methodenattribute implementiert werden können:

```
public interface IRestServiceImpl
{
    [OperationContract]
    [WebInvoke(Method = "GET",
      ResponseFormat = WebMessageFormat.Xml,
      BodyStyle = WebMessageBodyStyle.Wrapped,
      UriTemplate = "xml/{id}")]

    string XMLData(string id); // Interface-Methode

    [OperationContract]
    [WebInvoke(Method = "GET",
        ResponseFormat = WebMessageFormat.Json,
        BodyStyle = WebMessageBodyStyle.Wrapped,
        UriTemplate = "json/{id}")]

    string JSONData(string id); // Interface-Methode
}
```

In diesem Code sind 2 Interface-Methoden implementiert, die als Parameter einen String übergeben und einen String als Rückgabewert haben. In einem Fall (oben) ist der Rückgabewert in Form einer XML-Datei: `string XMLData(string id);` und im zweiten Fall ist der Rückgabewert in Form eines JSON-Formats: `string JSONData(string id);`

C) In der Datei RestServiceImpl.svc.cs implementieren wir die beiden obigen Methoden aus:

```
public class RestServiceImpl : IRestServiceImpl
{
    #region IRestService Members
    public string JSONData(string id)
    {
        return "You requested product " + id;
    }

    public string XMLData(string id)
    {
        return "You requested product" + id;
    }
    #endregion

}
```

D) Nun muss noch die *Web.config*-Datei um 2 Basis-Teile erweitert werden. Der erste Teil enthält Informationen über den End Point. Die Details sind <services>:

```
<system.serviceModel>
  <services>
      <service name="WcfService1.RestServiceImpl"
             behaviorConfiguration="ServiceBehaviour">
        <endpoint address="" binding="webHttpBinding"
             contract="WcfService1.IRestServiceImpl"
             bindingConfiguration=""
             behaviorConfiguration="EndPointBehavior"/>
      </service>
  </services>
```

Der zweite Teil enthält Informationen über das Verhalten des Services und des End Points <behaviors>

```
    <behaviors>
        <serviceBehaviors>
           <behavior name="ServiceBehaviour">
               <!-- To avoid disclosing metadata information, set the
                    value below to false before deployment -->
               <serviceMetadata httpGetEnabled="true"/>
               <!-- To receive exception details in faults for debugging
                    purposes, set the value below to true.  Set to
                    false before deployment to avoid disclosing
                    exception information -->
               <serviceDebug includeExceptionDetailInFaults="false"/>
           </behavior>
        </serviceBehaviors>
        <endpointBehaviors>
           <behavior name="EndPointBehavior">
               <webHttp helpEnabled="true"/>
           </behavior>
        </endpointBehaviors>
    </behaviors>
```

E) Damit ist das WCF abgeschlossen (speichern) und es kann getestet, also ausgeführt werden. Das WCF-Service liefert folgende HTML-Seite mit der Url der Interface-Klasse, Abb. 4:

Abb. 4: Start des Dienstes

Wir sehen die URL, die uns das WCF zu Verfügung stellt. Unter dieser URL können wir die Methoden aufrufen:

Die Funktion `XMLData(string id)` als Url-Erweiterung liefert das Ergebnis in xml-Format, Abb. 5:

Abb. 5: XML-Format einer Dienst-Anfrage

Die Funktion `string JSONData(string id)` als Url-Erweiterung liefert das Ergebnis in json-Format, Abb. 6:

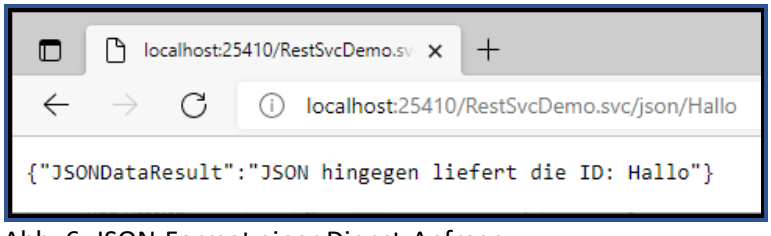

Abb. 6: JSON-Format einer Dienst-Anfrage

F) Zugriff auf eine Klassenbibliothek: Mit diesem WCF können wir auf Assemblies zugreifen, die Daten zur Verfügung stellen. Dazu erstellen wir ein **eigenes Projekt**, Abb.en 7, 8. Dieses Projekt unterscheidet sich vom WCF-Projekt!!!:

r. MT Solution > Add > New Project > Class Library (.NET Framework) C# - Next > Framework: .NET Framework 4 ... (erstellen).

Damit haben wir unsere Anwendung *"Solution WCF-Demo_18_02_22"* um die Bibliothek *BL_Company* erweitert. Dieser Bibliothek haben wir die Klassen *Company* und *Person* hinzugefügt. Diese Bibliothek ist noch dem *WCF* als *Referenz* hinzuzufügen.

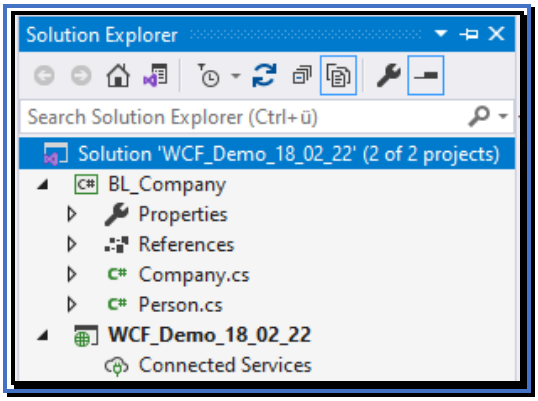

Abb. 7: Erstellen einer Bibliothek

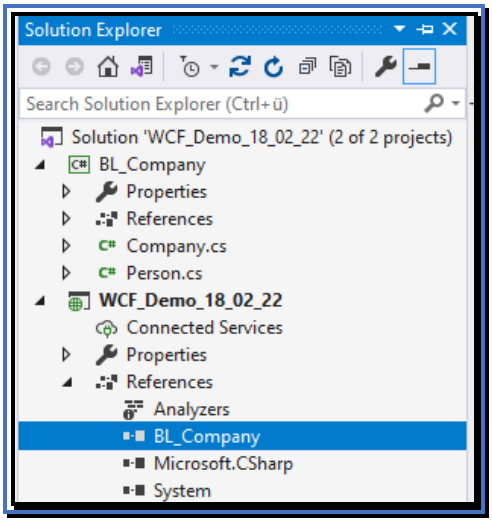

Abb. 8: Die erstellte Bibliothek

Nun kann das WCF auf Methoden dieser Bibliothek zugreifen und in der Folge den Client-Programmen zur Verfügung stellen.

2.2. Erstellen der Applikation

Hier erstellen wir eine Konsole-Applikation `Console App (.NET Framework)`, die auf das Service zugreift. Wir implementieren den Zugriff auf das WCF in der Methode `Main(…)` von `Program`. Wir benötigen dazu die Bibliothek `System.Net;` und das `System.IO.`

a) Erstelle einen Web-Request, der auf das Service zugreift. In der URL spezifizieren wir den Synonym der Methode mit den Übergabeparametern, die wir im Service aufrufen wollen:

```
var request =     (HttpWebRequest)WebRequest.Create
                  ("http://localhost:25410/RestSvcDemo.svc/xml/123");
```

`xml` steht für die Methode `XMLData(string id)`
`123` steht für die Parameter `id`

b) Ermittle über den Response das Ergebnis zu dieser WCF-Service-Methode:

```
HttpWebResponse response = request.GetResponse() as HttpWebResponse;
Stream stream = response.GetResponseStream();
StreamReader rdr = new StreamReader(stream);
string result = rdr.ReadLine();
```

c) Schreibe das Ergebnis in eine XML-Datei und ermittle das Resultat aus der XML-Datei:

```
File.WriteAllText("file.xml", result);
```

2.3. Exekution der Anwendung mit dem WCF

Hier ist es wichtig, dass das WCF gestartet ist, damit die Anwendung auf das WCF zugreifen kann. Da sich das Ergebnis in einer XML-Datei befindet, benötigen wir noch den Namensraum `System.Xml`;
Hier das Ergebnis einer Service-Anfrage, die ein int-Feld ermittelt:

```
▼<JSONCompanyListResponse xmlns="http://tempuri.org/">
  ▼<JSONCompanyListResult
    xmlns:a="http://schemas.microsoft.com/2003/10/Serialization/Arrays"
    xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
      <a:int>33</a:int>
      <a:int>44</a:int>
      <a:int>55</a:int>
    </JSONCompanyListResult>
  </JSONCompanyListResponse>
```

Ein Ergebnis, das ein Feld von Personen ermittelt:

```
▼<XMLPersonListResponse xmlns="http://tempuri.org/">
  ▼<XMLPersonListResult
    xmlns:a="http://schemas.datacontract.org/2004/07/WCF_Demo_18_02_22"
    xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
    ▼<a:Person>
        <a:Gehalt>2000</a:Gehalt>
        <a:Name>DrChess</a:Name>
        <a:PersID>1111</a:PersID>
      </a:Person>
    ▼<a:Person>
        <a:Gehalt>3000</a:Gehalt>
        <a:Name>DrNo</a:Name>
        <a:PersID>2222</a:PersID>
      </a:Person>
    ▼<a:Person>
        <a:Gehalt>4000</a:Gehalt>
        <a:Name>DrShiwago</a:Name>
        <a:PersID>3333</a:PersID>
      </a:Person>
    </XMLPersonListResult>
  </XMLPersonListResponse>
```