Web Services

Motivation

- Software isn't an encapsulated system
  - Interfaces to other systems (data exchange, functionality)
  - Call external functionality, that cannot be run locally (e.g.: Cognitive Services, www.microsoft.com/cognitive-services)
- Different ways:
  - Share .NET-Assemblies (NuGet, limited to .NET)
  - File exchange
  - Webservices (Interoperable, less requirements to communication partner)
- Webservices
  - Interface, callable through HTTP
  - Standardized format
  - SOAP
    - XML, huge overhead
    - Metadata (WSDL)
  - REST
    - JSON or XML, lightweight
    - No (?) Metadata - see below

Creating a SOAP Webservice with WCF

- Create a new project **WCF Service Application**
- Delete IService1
- Add a new Web Service, call it CurrencyCalculatorService
- Implement 1-2 methods
- Show request and response in WCF Test Client

REST

- HTTP-Verbs: GET, POST, PUT, DELETE, ...

Web API

- Creation of REST-Webservices
- Alternative: WCF (Windows Communication Foundation)
  - More possibilities, more configuration necessary

New project

- New project:
  - ASP.NET, **CurrencyConverter.WebApi** (Solution: **CurrencyConverter**)
  - Empty, activate Web API, No Authentication
- Business Logic
  - Copy project **CurrencyConverter.BL**
  - Copy project **CurrencyConverter.Domain**
  - Add references
- New controller
  - **Web API 2 Controller - Empty**
  - **CurrenciesController.cs**

```
private ICurrencyCalculator Logic { get; set; } = BLFactory.GetCalculator();

[HttpGet]
public CurrencyData GetBySymbol(string symbol)
{
    return Logic.GetCurrencyData(symbol);
}
```

Routing

- How to call the REST-Service?
- App_Start/WebApiConfig.cs
  - Version 1: `csharp config.Routes.MapHttpRoute(...)`
  - Version 2: Attribute-Routing
- Choose version 2 - maybe delete the DefaultApi-Route

```
  [HttpGet]
* [Route("currencies/{symbol}")]
  public CurrencyData GetBySymbol(string symbol)
  {
    ...
```

Aufruf

- Browser: **http://localhost:/currencies/EUR**
  - Different response depending on browser (Edge: JSON, Chrome: XML)
  - Reason: HTTP-Header **Accept**: application/json or application/xml
- Show Postman
  -
  - Send GET-Request, change Accept-Header

Statuscodes

- REST Services should respond with correct HTTP-Status codes

- No Exception-Details (Security)
- What's the correct status for "not found"? -> 404

```
    [HttpGet]
    [Route("currencies/{symbol}")]
    public CurrencyData GetBySymbol(string symbol)
    {
        if (!Logic.CurrencyExists(symbol))
           throw new HttpResponseException(HttpStatusCode.NotFound);
        return Logic.GetCurrencyData(symbol);
    }
```

GetAll

```
[HttpGet]
[Route("currencies")]
public IEnumerable<CurrencyData> GetAll()
{
    return Logic.GetCurrencies().Select(symbol => Logic.GetCurrencyData(symbol));
}
```

RateOfExchange

- Route: currencies/EUR/rates/USD)

```
[HttpGet]
[Route("currencies/{srcSymbol}/rates/{targSymbol}")]
public double RateOfExchange(string srcSymbol, string targSymbol)
{
    if (!Logic.CurrencyExists(srcSymbol) || !Logic.CurrencyExists(targSymbol))
        throw new HttpResponseException(HttpStatusCode.NotFound);
    return Logic.RateOfExchange(srcSymbol, targSymbol);
}
```

Post- und Put-Methoden

```
[HttpPut]
[Route("currencies")]
public void Update([FromBody] CurrencyData data)
{
    if (!Logic.CurrencyExists(data.Symbol))
        throw new HttpResponseException(HttpStatusCode.NotFound);
    Logic.Update(data);
}

[HttpPost]
[Route("currencies")]
```

```
public HttpResponseMessage Insert([FromBody] CurrencyData data)
{
    if (Logic.CurrencyExists(data.Symbol))
        return Request.CreateErrorResponse(
            HttpStatusCode.Conflict,
            "currency " + data.Symbol + " already exists");

    Logic.Insert(data);

    // return link to newly created symbol
    var response = Request.CreateResponse(HttpStatusCode.Created);
    string uri = Url.Link("CurrenciesRoute",
        new { symbol = data.Symbol });
    response.Headers.Location = new Uri(uri);

    return response;
}
```

Name route **GetBySymbol** ("GetBySymbolRoute"), to be able to reference to it

```
[Route("currencies/{symbol}", Name="GetBySymbolRoute")]
public CurrencyData GetBySymbol(string symbol)
{
  ...
```

Test

- Postman: POST, http://localhost:1870/currencies
- Body: raw, JSON
- Response: 201 (Created) + Header.Location gesetzt

```
{
  "Symbol": "NOK",
  "Name": "Norwegian Krone",
  "Country": "Norway",
  "EuroRate": 8.97
}
```

C#-Client

- New Project:
  - Console Application
  - **CurrencyConverter.CSharpClient**

GetSymbol abfragen

```
private const string BASE_URI = "http://localhost:1870";

private static async void TestRestService()
{
    using (var client = new HttpClient())
    {
        client.DefaultRequestHeaders.Add("Content", "application/json");
        var json = await client.GetStringAsync(BASE_URI + "/currencies/USD");
        Console.WriteLine(json);
    }
}
```

- Show error (query for a wrong symbol -> Exception, 404)
- Create model class
  - Copy JSON-Response in Browser, Debugger
  - Visual Studio:
    - New class **CurrencyData**
    - Edit > Paste Special > Paste JSON as Classes
- Nuget: Add Newtonsoft.Json

```
var data = JsonConvert.DeserializeObject<CurrencyData>(json);
```

Exchangerate updaten

```
// update exchange rate
data.EuroRate = 1.06f;
var content = new StringContent(
    JsonConvert.SerializeObject(data),
    Encoding.Default,
    "application/json");

var response = await client.PutAsync(BASE_URI + "/currencies", content);
```

Create UWP client

Create Swagger-Metadata

- Motivation: Why Metadata?
  - Documentation
  - Generation of Proxy-Klassen
- **Swagger**: https://swagger.io
  - Swagger UI: Documentation
  - Swagger CodeGen: Proxy generation
- .NET-Implementierung of Swagger: **Swashbuckle**

Create Swagger Metadaten

- Add NuGet **Swashbuckle** to WebApi-Project
  - Hint: Swashbuckle.Core just adds metadata, no documentation
- Discuss App_Start/SwaggerConfig.cs
- Browser:
  - JSON-Metadata: http://localhost:1870/swagger/docs/v1
- Browser: http://localhost:1870/swagger
  - Currencies > Test methods incl. errors
  - Optional: Enhance documentation with XML: http://www.wmpratt.com/swagger-and-asp-net-web-api-part-1/

C#-Client with Swagger

- Store Swagger-Metadata as file (http://localhost:1870/swagger/docs/v1)
- Visual Studio: CSharpClient-Project
  - Context menu > Add Rest API Client
  - Select an existing Swagger metadata file
  - Client Namespace: **CurrencyConverter.CSharpClient.Swagger**
- Errors because of concurring Currency Data implementations
  - Delete (old) CurrencyData class, update usings

```
private static async void TestSwagger()
{
    using (ICurrencyConverterWebApi converter = new CurrencyConverterWebApi(new Uri(BASE_URI
    {
        var data = await converter.Currencies.GetBySymbolAsync("EUR");

        // create new entry
        await converter.Currencies.InsertAsync(new CurrencyData
        {
            Symbol = "ATS",
            Name = "Schilling",
            Country = "Austria",
            EuroRate = 7.1
        });
    }
}
```

- InsertAsync will throw an exception, because of different answer than **200 OK**
- Add attribute in service

```
[SwaggerResponse(HttpStatusCode.Created)]
[SwaggerResponse(HttpStatusCode.Conflict)]
public HttpResponseMessage Insert([FromBody] CurrencyData data)
{
    ...
```

```
}

[SwaggerResponse(HttpStatusCode.NotFound)]
public CurrencyData GetBySymbol(string symbol)
{
    ...
}
```

- Store new Swagger metadata as file
- Delete generated Swagger classes
- Create new Swagger proxies
- Show Get...WithOperationResponse-Methode

```
var result = await converter.Currencies.GetBySymbolWithOperationResponseAsync("XXX");
if (result.Response.IsSuccessStatusCode)
{
    Console.WriteLine($"{result.Body.Name}: {result.Body.EuroRate}");
}
else
{
    Console.WriteLine($"Request {result.Request.RequestUri} returned {result.Response.Status
}
```

Deploy to Azure