Jörg Neumann

Flexible Smart-Client-Architekturen für .NET-Desktop-Entwickler

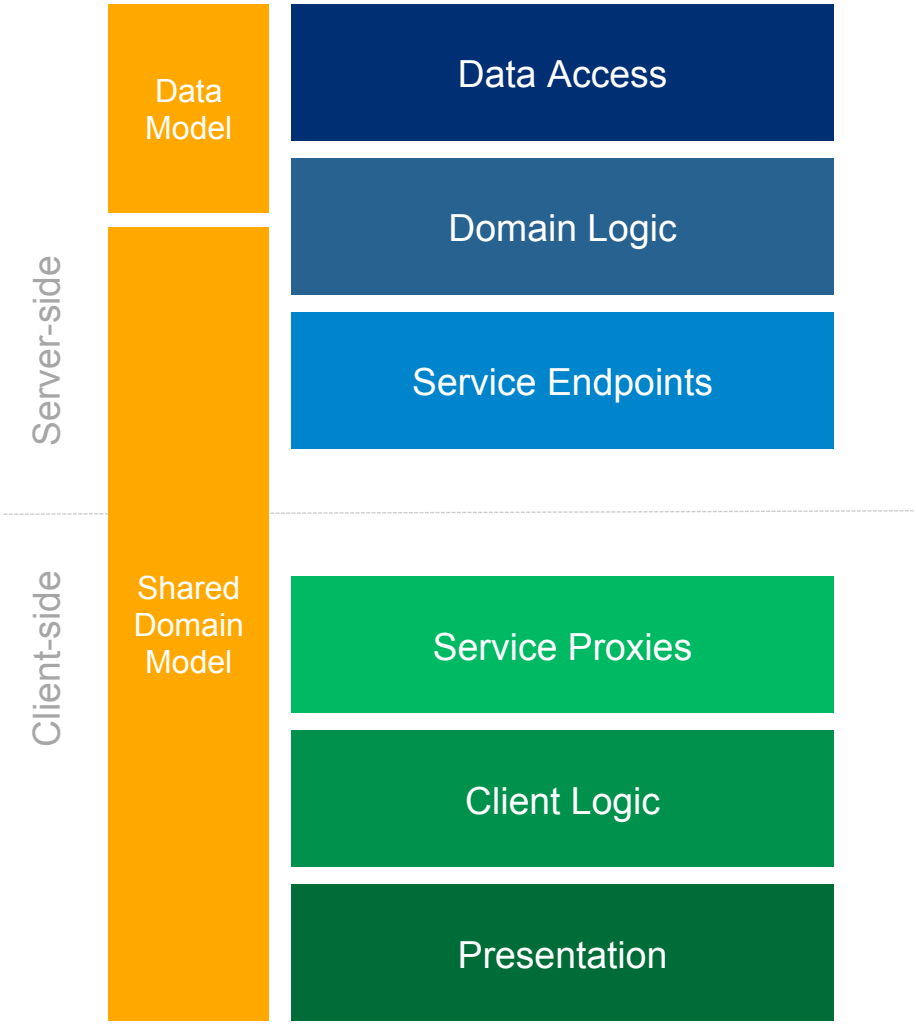
Die Serverseite

Jörg Neumann

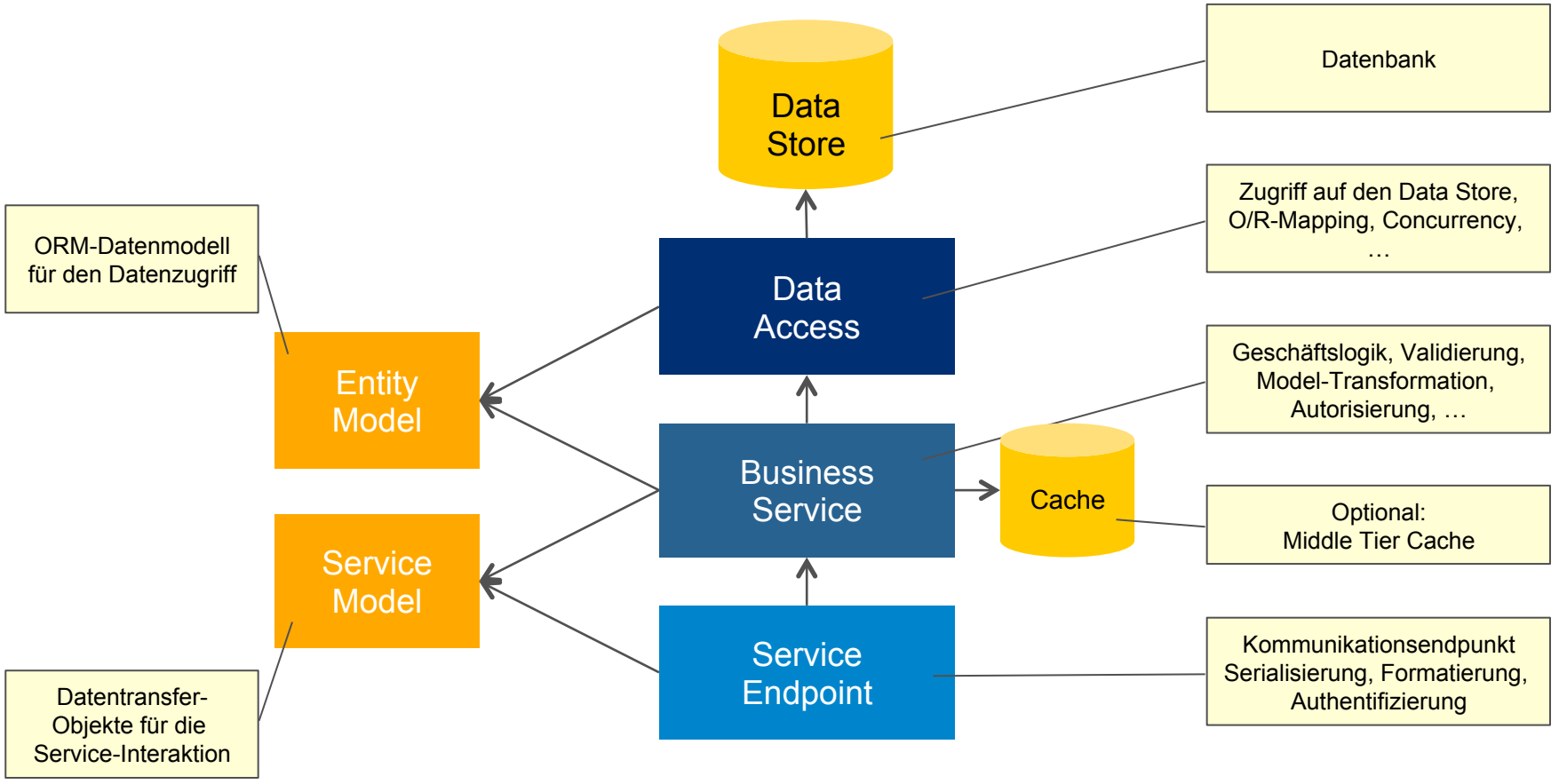
Microsoft Most Valuable Professional

- Principal Consultant bei Acando
- Associate bei Thinktecture
- MVP „Windows Platform Development“
- Beratung, Training, Coaching
- Buchautor, Speaker
- Themen: Xamarin, Windows Store Apps, WPF
- Mail: Joerg.Neumann@Acando.de
- Blog: www.HeadWriteLine.BlogSpot.com

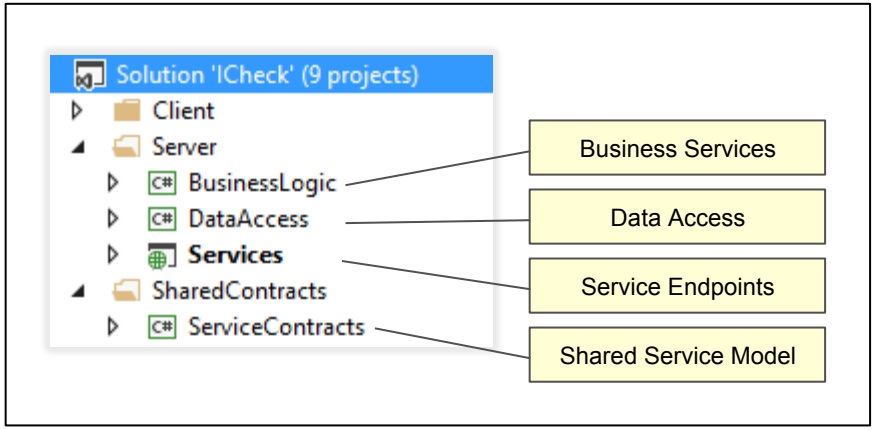
Architektur



Server-Architektur



Projektaufbau



Demo

Projektaufbau

Datenzugriff

- Entity Framework Code-First
  - Zugriffsklasse leitet von **DbContext** ab
  - Konfiguration erfolgt über **DbContext.Configuration**
  - Zugriff erfolgt über **DbSet<T>**

Datenzugriff

- Entity Framework Code-First
  - Zugriffsklasse leitet von **DbContext** ab
  - Konfiguration erfolgt über **DbContext.Configuration**
  - Zugriff erfolgt über **DbSet<T>**

```
public class ConferenceDudeContext : DbContext
{
    public ConferenceDudeContext()
        : base("ConferenceDudeContext")
    {
        Configuration.ProxyCreationEnabled = false;
        Configuration.AutoDetectChangesEnabled = false;
        Configuration.LazyLoadingEnabled = false;
        Configuration.ValidateOnSaveEnabled = false;
    }

    public DbSet<Speaker> Speakers { get; set; }
}
```

Models

- Das Mapping erfolgt über Klassen- und Property-Attribute
  - Namespace **System.ComponentModel.DataAnnotations**
- Mapping
  - **Table**: Tabellenname in der Datenbank
  - **Key**: Eindeutiger Schlüssel
  - **DatabaseGenerated**: Datenbankgenerierter Schlüssel
  - …
- Attribute zur Validierung
  - **Required**: Pflichtfeld
  - **StringLength**: Feldlänge
  - **EmailAddress**: Inhalt muss eine gültige Mail-Adresse sein
  - **RegularExpression**: Inhalt muss mit RegEx matchen
  - …

Models

```
[Table("Speakers")]
public class Speaker
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    [StringLength(100)]
    public string Name { get; set; }

    [StringLength(100)]
    public string Company { get; set; }

    [EmailAddress]
    [StringLength(100)]
    public string EMail { get; set; }

    [RegularExpression(@"\d{2,5}/\d+")]
    [StringLength(100)]
    public string Phone { get; set; }
    …
}
```

DEMO

ENTITY FRAMEWORK CODE-FIRST

Service-orientierte Operationen

- Bilden einen konkreten Business Case ab
  - Kein Forms-over-Data-Anzatz mit einfachen CRUD-Operationen
- Beispiel: **GetCustomerOrder()** liefert
  - **CustomerOrder**-Objekt (inkl. Customer-, Order- & OrderDetails-Daten)
- Beispiel: **GetSmallCustomerList()** liefert
  - Ein Subset der **Customer**-Entität
- Vorteile
  - Es werden nur die Daten übertragen, die benötigt werden
  - Unnötige Roundtrips werden vermieden
  - Datenbank-Abfragen können minimiert werden
- Herausforderungen
  - O/R-Mapper bilden Datenbanktabellen auf Klassen ab
  - Eine Klasse kann hierbei nur auf eine Tabelle gemapped werden

Lösung

- Trennung von Service- und Data Model
  - Datenzugriffschicht und Service haben separate Models
- Models müssen im Business Layer gemapped werden
  - AutoMapper erledigt dies sehr effizient
  - Über NuGet erhältlich
- Mappings erstellen
  - Statische Mapping-Klasse im Business-Layer-Projekt erstellen

Mapping-Klasse erstellen

```
using AutoMapper;
using System.Collections.Generic;
using System.Linq;
using biz = BusinessServices.Model;
using dat = DataServices.Model;

namespace BusinessServices
{
  internal static class ModelMapping
  {
    static ModelMapping()
    {
      Mapper.CreateMap<biz.SpeakerDto, dat.Speaker>().IgnoreAllNonExisting();
      Mapper.CreateMap<dat.Speaker, biz.SpeakerDto>().IgnoreAllNonExisting();

      Mapper.AssertConfigurationIsValid();
    }

    …
  }
}
```

Extension-Methods implementieren

```
…
public static biz.SpeakerDto Map(this dat.Speaker item)
{
  return Mapper.Map<dat.Speaker, biz.SpeakerDto>(item);
}

public static dat.Speaker Map(this biz.SpeakerDto item)
{
  return Mapper.Map<biz.SpeakerDto, dat.Speaker>(item);
}

public static IEnumerable<biz.SpeakerDto> Map(this IEnumerable<dat.Speaker> items)
{
  var destItems = new List<biz.SpeakerDto>();
  items.ToList().ForEach(item => destItems.Add(item.Map()));
  return destItems;
}

public static IEnumerable<dat.Speaker> Map(this IEnumerable<biz.SpeakerDto> items)
{
  var destItems = new List<dat.Speaker>();
  items.ToList().ForEach(item => destItems.Add(item.Map()));
  return destItems;
}
```

Mapping anwenden

```
public class ConferenceManager : IConferenceManager
{
  IConferenceDataService _dataService;
  …
  public IEnumerable<SpeakerDto> GetSpeakerList()
  {
    return = _dataService.GetSpeakerList().Map();
  }

  public SpeakerDto GetSpeakerById(int id)
  {
    return _dataService.GetSpeakerById(id).Map();
  }
  …
}
```

Demo

Model-Mapping

RESTful Services

Prinzipien eines RESTful Service:

- Abfragen von Informationen
- Einfügen, Ändern und Löschen von Daten
- Identifikation der Daten findet über die URI statt
- Kommunikation basiert auf HTTP-Verbs
  - GET -> Select
  - POST -> Insert
  - PUT -> Update
  - DELETE -> Delete

Was ist ASP.NET Web API?

- HTTP-basierte Services
- Für breite Reichweite an Technologien entworfen
- Verwendet HTTP als Anwendungsprotokoll und nicht als Transportprotokoll

Design-Prinzipien

- HTTP-konformes Programmiermodell
- Leichtgewichtig, testbar, skalierbar
- Einfaches Mapping von Ressourcen zu URIs
- Unterstützung für unterschiedliche Formate
- Verwendung von HTTP-Status-Codes
- Dependency Injection-Support

Implementierung

- Klasse von **ApiController** ableiten
  - URI orientiert sich an Namenskonvention
  - …/api/Customer -> CustomerController
- Action-Methoden implementieren
  - Ressourcen bereitstellen / Operationen durchführen
- Actions zu HTTP-Verbs mappen
  - Methodennamenpräfix (Get…, Put…, Post…, Delete…)
  - Attribute verwenden (**HttpGet**, **HttpPost**, **HttpPut**, **HttpDelete**)

Demo

WebAPI-Controller

Output-Format konfigurieren

- Formatter integrieren
  - z.B. JSON-formatierten Output
  - **HttpConfiguration.Formatters**-Eigenschaft

```
public static class WebApiConfig
{
  public static void Register(HttpConfiguration config)
  {
    var formatter = new JsonMediaTypeFormatter();
    config.Formatters.Clear();
    config.Formatters.Add(formatter);
    …
  }
}
```

DEMO

FORMATTERS

Konfiguration

- **WebApiConfig**-Klasse
  - Liegt im Ordner **App_Start**
  - Endpunkte, Actions und Routen konfigurieren

```
public static class WebApiConfig
{
  public static void Register(HttpConfiguration config)
  {
    …
  }
}
```

Routen konfigurieren

- Route definieren

```
public static void Register(HttpConfiguration config)
{
  config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
  );
}
```

- Controller-Methode:

```
[HttpGet]
public IList<Customer> GetCustomer() { … }

[HttpGet]
public IList<Customer> GetCustomerById(int id) { … }
```

- Zugriff
  - http://localhost:1334/api/Customers
  - http://localhost:1334/api/Customers/1

Action-basiertes Routing

- Unterschiedliche Aktionen über einen Controller abbilden
  - Action wird in URI übergeben
- Routing Template:

```
config.Routes.MapHttpRoute(
  name: "DefaultApi",
  routeTemplate: "api/{controller}/{action}/{id}",
  defaults: new { id = RouteParameter.Optional });
```

- Methode wird mit **ActionName**-Attribute deklariert

```
[HttpGet, ActionName("list")]
public IList<Customer> GetCustomers() { … }

[HttpGet, ActionName("groups")]
public IList<Customer> GetCustomerGroups() { … }
```

- Zugriff
  - http://localhost:1334/api/customers/list
  - http://localhost:1334/api/customers/groups

DEMO

ACTION-BASED ROUTING

Attribut-basiertes Routing

- Unterschiedliche Routing-Strategien in einem Controller
  - **Route**-Attribut bestimmt über Methode das Routing
- Konfiguration:

```
config.MapHttpAttributeRoutes();
config.Routes.MapHttpRoute(
  name: "DefaultApi",
  routeTemplate: "api/{controller}/{id}",
  defaults: new { id = RouteParameter.Optional });
```

- Controller

```
[HttpGet]
public Customer GetCustomerById(int id) { … }

[HttpGet, Route("api/Customers/Count")]
public int GetCustomerCount() { … }
```

- Zugriff
  - http://localhost:1334/api/customers/1
  - http://localhost:1334/api/customers/count

Demo

Attribute-based Routing

Action Results

- HTTP sendet bei der Rückgabe Statuscodes im Header
  - OK, Not Found, …
- Eine Web API-Controller Action kann zurückgeben:
  - void
  - Ein benutzerdefinierter Typ
  - HttpResponseMessage
  - IHttpActionResult

Action Results: void

- Implementierung

```
public class ValuesController : ApiController
{
    public void Post()
    {
    }
}
```

- Ergebnis
  - HTTP/1.1 204 No Content

Action Results: HttpResponseMessage

- Implementierung

```
public HttpResponseMessage Get()
{
  HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.OK, "value");
  response.Content = new StringContent("hello", Encoding.Unicode);
  response.Headers.CacheControl = new CacheControlHeaderValue()
  {
    MaxAge = TimeSpan.FromMinutes(20)
  };
  return response;
}
```

- Ergebnis

```
HTTP/1.1 200 OK
Cache-Control: max-age=1200
Content-Length: 10
Content-Type: text/plain; charset=utf-16
Server: Microsoft-IIS/8.0 Date: Mon, 27 Jan 2014 08:53:35 GMT

hello
```

Action Results: HttpResponseMessage II

- Objekte können über **CreateResponse()** zurückgegeben werden
  - Web API verwendet den konfigurierten Formatter, um die Daten in den HTTP Response Body zu schreiben

```
public HttpResponseMessage GetSpeakerList()
{
  IEnumerable<SpeakerDto> speakers = _manager.GetSpeakerList();
  HttpResponseMessage response =
    Request.CreateResponse(HttpStatusCode.OK, speakers);
  return response;
}
```

Action Results: IHttpActionResult

- Erstellung des HTTP-Results kann in eine separate Klasse ausgelagert werden
- Gut für Unit Tests
- Macht den Code klarer

```
public IHttpActionResult GetSpeakerById(int id)
{
    var speaker = _manager.GetSpeakerList().FirstOrDefault();
    if (speaker == null)
    {
        return NotFound();
    }
    return Ok(speaker);
}
```

Demo

Action Results

Dependency Injection

- Controller testbar machen
  - Repository Pattern
  - Business-Logik in Controller injizieren

```
public class CustomersController : ApiController
{
  public CustomersController(ICustomerManager manager)
  {
    …
  }
}
```

Dependency Injection

- Web API enthält Support für Dependency Injection
  - **HttpConfiguration.DependencyResolver**-Eigenschaft
  - Nimmt einen IoC-Container entgegen (z.B. **AutoFac**)

```
public static class WebApiConfig
{
  public static void Register(HttpConfiguration config)
  {
    …
    config.DependencyResolver =
      ContainerConfiguration.GetWebApiDependencyResolver();
  }
}
```

AutoFac konfigurieren

```
internal class ContainerConfiguration
{
  public static AutofacWebApiDependencyResolver GetWebApiDependencyResolver()
  {
    var builder = new ContainerBuilder();
    var config = GlobalConfiguration.Configuration;

    // Controller-Assembly registrieren
    builder.RegisterApiControllers(Assembly.GetExecutingAssembly());
    builder.RegisterWebApiFilterProvider(config);

    // Business-Logik-Assembly registrieren
    var businessLogicAssembly = Assembly.Load("BusinessServices");
    builder.RegisterAssemblyTypes(businessLogicAssembly)
      .AsImplementedInterfaces()
      .InstancePerRequest();

    var container = builder.Build();
    var resolver = new AutofacWebApiDependencyResolver(container);

    return resolver;
  }
}
```

DEMO

DEPENDENCY INJECTION

Web API Services konsumieren

- NuGet-Package
  - Microsoft ASP.NET Web API Client Libraries
  - Erweitert die Möglichkeiten der **HttpClient**-Klasse
  - Bringt **JSON.NET** mit
- Service-Proxy implementieren
  - Manuelle Implementierung der HTTP-Anfragen
  - ggf. Datentype/Listentypen für den UI-Layer wandeln
  - Methoden Asynchron implementieren

GET-Operationen

```
public partial class SessionServiceClient : ISessionService
{
  private HttpClient httpClient;

  public SessionServiceClient()
  {
    httpClient = new HttpClient();
    httpClient.BaseAddress =
      new Uri(ConfigurationManager.AppSettings["ServicesBaseUrl"]);
  }

  public async Task<ObservableCollection<Speaker>> GetSpeakerListAsync()
  {
    var response = await httpClient.GetAsync("speakers/list");
    response.EnsureSuccessStatusCode();
    return await response.Content.ReadAsAsync<ObservableCollection<Speaker>>();
  }

  public async Task<Speaker> GetSpeakerByIdAsync(int id)
  {
    var response = await httpClient.GetAsync("speakers/list?id=" + id);
    return await response.Content.ReadAsAsync<Speaker>();
  }
  …
}
```

POST-, PUT-, DELETE-Operationen

```
public partial class SessionServiceClient : ISessionService
{
  …
  public async Task<Speaker> AddSpeakerAsync(Speaker speaker)
  {
    var response = await httpClient.PostAsJsonAsync("speakers/list", speaker);
    return = await response.Content.ReadAsAsync<Speaker>();
  }
  public async Task UpdateSpeakerAsync(Speaker speaker)
  {
    await httpClient.PutAsJsonAsync("speakers/list", speaker);
  }
  public async Task DeleteSpeakerAsync(int id)
  {
    await httpClient.DeleteAsync("speakers/list?id=" + id);
  }
}
```

DEMO

CLIENT-SIDE ACCESS

Weitere Optionen

- Gewünschtes Format im HTTP-Header angeben
  - **DefaultRequestHeaders.Accept**-Eigenschaft

```
public class SpeakerServiceClient
{
  public SpeakerServiceClient(string baseUri)
  {
    _httpClient = new HttpClient();
    _httpClient.BaseAddress = new Uri(baseUri);
    _httpClient.DefaultRequestHeaders.Accept.Add(
      new MediaTypeWithQualityHeaderValue("application/json"));
  }
  …
}
```

Authentifizierung konfigurieren

- Controller wird mit **Authorize**-Attibut versehen

```
[Authorize]
public class SpeakersController : ApiController
{ … }
```

- Authentifizierungsmethode in **web.config** deklarieren

```
<configuration>
  <system.web>
    <authentication mode="Windows"/>
    …
  </system.web>
  …
</configuration>
```

Authentifizierung im Client konfigurieren

- Authentifizierung wird über **HttpClientHandler**-Klasse konfiguriert
- **HttpClient** wird mit **HttpClientHandler**-Instanz initialisiert

```
public class SpeakerServiceClient
{
  public SpeakerServiceClient(string baseUri)
  {
    var handler = new HttpClientHandler() { UseDefaultCredentials = true };
    _httpClient = new HttpClient(handler);
    _httpClient.BaseAddress = new Uri(baseUri);
    _httpClient.DefaultRequestHeaders.Accept.Add(
      new MediaTypeWithQualityHeaderValue("application/json"));
  }
  …
}
```


Q & A

Sie brauchen Unterstützung?

- Training, Coaching, Reviews, Entwicklungsunterstützung
  - joerg.neumann@acando.de