Projektowanie i Konstrukcja Systemów Rozproszonych

ServiceStack

Resource Oriented Architecture

- Architektura oprogramowania i styl programowania kładący nacisk na zasoby oraz interfejsy do ich modyfikacji
- REST nakłada dodatkowe ograniczenia wzorowane na sieci WWW na architekturę oprogramowania ROA

ROA – cechy

- Rezygnacja z podejścia RPC na rzecz udostępniania zasobów
- Reprezentacja zasobu powinna zawierać odnośniki do innych powiązanych zasobów
- Każdy zasób powinien udostępniać własne URI
- Kombinacje zasobów powinny być reprezentowane pod własnym URI

REST – cechy

- Wykorzystanie głównie HTTP/1.1 i URI
- Preferowanie danych w URI, niż w nagłówkach HTTP
- Klient-serwer
- Bezstanowość
- Zdolność do przechowywania podręcznego (*Cacheability*)
- Tolerancja systemów warstwowych (np. proxy)

REST – cechy (2)

- Założenia co do interfejsów zasobów
  - Identyfikacja zasobów w żądaniach (np. URI)
  - Reprezentacja zasobów (np. JSON, XML)
  - Manipulacja zasobami poprzez ich reprezentację
  - Samoopisujące wiadomości (np. MIME)
  - Hiperlinki jako podstawowy mechanizm zmiany stanu klienta

ROA/REST – zalety

- Łatwe sterowanie wydajnością i skalowalnością (np. poprzez dobrze poznane mechanizmy dla HTTP)
- Prostota interfejsów (ograniczenie do *HTTP verbs*)
- Łatwa modyfikacja zasobów i ich reprezentacji
- Jasna i przejrzysta komunikacja

ROA/REST – wady

- Założenie o bezstanowości serwerów
  - Konieczność przesyłania pełnych reprezentacji zasobów
- W praktyce: ograniczenia płynące z działania w ramach HTTP
- Sesja wyłącznie po stronie klienta (odporność na awarie, trwałość)

ServiceStack

- Framework do tworzenie usług webowych, nastawiony na REST
  - Konwencja nad konfiguracją
  - Brak generowania kodu
  - Dedykowane POCO DTO
  - .NET i mono
  - Łatwy w testowaniu
  - Zorientowany na wiadomości (*Request/Reply*)
  - OpenSource (v3.x)

ServiceStack – cechy

- Formatowanie do XML, JSON, SOAP i innych
- Wbudowane narzędzia walidacji żądań
- Automatyczna obsługa wyjątków
- Caching (Memcached i Redis wspierane standardowo)
- Uwierzytelnianie i autoryzacja (wbudowany OAuth dla Twitter/Facebook)

ServiceStack – HelloWorld

- Każda usługa składa się z następujących elementów
  - Request DTO
  - Implementacja usługi
  - Response DTO
  - URI usługi (routing żądań)

ServiceStack – HelloWorld (2)

- DTO Request

```
using ServiceStack.ServiceHost;
[Route("/hello")]
[Route("/hello/{Name}")]
public class Hello :
 IReturn<HelloResponse>
{
    public string Name { get; set; }
}
```

- DTO Reply

```
class HelloResponse
{
   public string Result { get; set; }
}
```

ServiceStack – HelloWorld (3)

- Usługa

```
public class HelloService : Service
{
    public HelloResponse Get(HelloRequest request)
    {
        return "Hello" + request.Name;
    }
}
```

- Klient

```
string response = client.Get(new HelloRequest("Chuck"));
```

ServiceStack – tworzenie API

- W usłudze – implementacja metod odpowiadających czasownikom HTTP
  - Możliwe dodawanie niestandardowych
  - Różne typy DTO dla różnych czasowników

```
[Route("/reqstars")]
public class AllReqstars : IReturn<List<Reqstar>> { }
public class ReqstarsService : Service
{
    public List<Reqstar> Any(AllReqstars request)
    {return Db.Select<Reqstar>();}
}
```

ServiceStack – tworzenie API (2)

```
[Route("/users", "GET,POST")]
[Route("/api/admin/users/{Name}", "GET,PUT,DELETE")]
public class User : IReturn<IList<UserDto>> { }

[Route("/myUser", "GET")]
public class MyUser : IReturn<UserDto> { }

Public class UsersService : Service {
    public UserDto Get (MyUser request) {}
    public object Delete (User request) {}
    public object Put (User request) {}
    [SetStatus(HttpStatusCode.Accepted, "New user was created")]
    public object Post (User request) {} }
```

ServiceStack – Metadane

- Generowane pod adresem `/metadata`
- Lista usług i punktów wejścia
- Adresy dla każdego formatu danych (z przykładami)
- WSDL dla usług SOAP
- Adresy opisów XSD typów danych DTO
- Adresy do przykładowych implementacji klienckich

ServiceStack – Metadane (2)

- Możliwe formatowanie do: XML, JSON, JSV, CSV, SOAP

www.servicestack.net/ServiceStack.Hello/servicestack/json/metadata?op=Hello

Hello Web Services

<back to all web services

Hello

ServiceStack's Hello World web service.

REST Paths

The following REST paths are available for this service.

```
All Verbs /hello/{Name}
All Verbs /hello/{Name*}
All Verbs /hello
```

To override the Content-type in your clients HTTP **Accept** Header, append **?format=json**

To embed the response in a **jsonp** callback, append **?callback=myCallback**

HTTP + JSON

The following are sample HTTP requests and responses. The placeholders shown need to be replaced with actual values.

```
POST /json/syncreply/Hello HTTP/1.1
Host: c60-116.icpnet.pl
Content-Type: application/json
Content-Length: length

{"Name":"String"}
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: length

{"Result":"String"}
```

Hosting

- IIS
- Self-hosting
- Linux: daemon i FastCGI

```
public class AppHost : AppHostHttpListenerBase {
    public AppHost()
    : base("HttpListener Self-Host", typeof(HelloService).Assembly) {}

    public override void Configure(Funq.Container container) {
        Routes.Add<Hello>("/hello").Add<Hello>("/hello/{Name}");
        Plugins.Add (…);
    }
}
```

Hosting (2)

```
static void Main(string[] args)
{
    var listeningOn = args.Length == 0 ? "http://*:1337/" : args[0];
    var appHost = new AppHost();
    appHost.Init();
    appHost.Start(listeningOn);

    Console.WriteLine("AppHost Created at {0}, listening on {1}",
        DateTime.Now, listeningOn);

    Console.ReadKey();
}
```

Routing żądań

- Dopasowanie zmiennych i czasowników HTTP

```
[Route("/hello/{Name}", "GET POST")]
```

- Wildcard

```
[Route("/hello/{Name*}")]
```

  - `/hello`
  - `/hello/name`
  - `/hello/my/name/is  //Name = my/name/is`

- Ścieżki ignorowane

```
[Route("/contacts/{Id}/{ignore}", "GET")]
```

  - `/contacts/1/john-doe – dopasuje tylko „1”`

Odpowiedzi

- Usługa może zwrócić:
  - Obiekt DTO serializowany do wybranego formatu
  - Ręcznie wybrane: `HttpResult, HttpError, CompressedResult`
- Dopasowanie formatu odpowiedzi
  - Nagłówek „`Accept`” lub `/srv?format=[json|xml|...]`

Odpowiedzi (2)

```
public object Get(Hello request) {
    //1. Returning a custom Response Status and Description with Response DTO body:
    var responseDto = ...;
    return new HttpResult(responseDto, HttpStatusCode.Conflict) {
        StatusDescription = "Computer says no",
    };
    //2. Throw or return a HttpError:
    throw new HttpError(HttpStatusCode.Conflict, "SomeErrorCode");
    //3. Modify the Request's IHttpResponse
    base.Response.StatusCode = (int)HttpStatusCode.Redirect;
    base.Response.AddHeader("Location", "http://path/to/new/uri");
}
//4. Using a Request or Response Filter
[AddHeader(ContentType = "text/plain")]
public string Get(Hello request)
{ return "Hello, {0}!".Fmt(request.Name); }
```

Wtyczki

- Standardowe
  - Metadane
  - HTML Info Format
  - Razor Markdown
- Opcjonalne
  - MVC Razor
  - Validation
  - Authentication
  - Registration
  - Request Logger

Kontener IoC

- Kontener IoC używany dla wszystkich zależności
  - Usług
  - Filtrów żądań
  - Walidatorów
- Domyślnie singleton
- Przykład rejestracji własnego typu

```
public override void Configure(Container container)
{
    container.RegisterAutoWired<MyType>();
    container.RegisterAutoWiredAs<MyType,IMyType>();
}
```

Obsługa błędów

- Wyjątki CLR przekazywane end-to-end, ale z serializacją po drodze (np. JSON)
- Biblioteka do obsługi błędów w JavaScript
- Przykład (ważna konwencja nazw)

```
public class Hello {}

    public class HelloResponse //Follows naming convention and is in the
same namespace as 'Hello'

{

    public ResponseStatus ResponseStatus { get; set; } //Exception gets
serialized here

}
```

Obsługa błędów (2)

- Wyjątki CLR mapowane na kody HTTP
  - `MyEx : ArgumentException -> 400 BadRequest`
  - `UnauthorizedAccessException -> 403 Forbidden`
  - `MyEx : NotImplementedException -> 405 MethodNotAllowed`
  - `Inne wyjątki -> 500 InternalServerError`
- Ręczne zwrócenie błędu zamiast DTO

```
public object Get(User request) {

       throw HttpError.NotFound("User {0} does not exist".Fmt
(requst.Name));

}
```

Obsługa błędów (3)

- Wyjątki odtwarzane po stronie klienta

```
try {
    var client = new JsonServiceClient(BaseUri);
    var response = client.Send<UserResponse>(new User());
} catch (WebServiceException webEx)
{ /*  webEx.StatusCode  = 400
      webEx.ErrorCode   = ArgumentNullException
      webEx.Message     = Value cannot be null. Parameter name: Name
      webEx.StackTrace  = (Server StackTrace - if DebugMode is enabled)
      webEx.ResponseDto = (populated Response DTO)
      webEx.ResponseStatus   = (your populated Response Status DTO)
      webEx.GetFieldErrors() = (individual errors for each field if any)
```

Walidacja – *Fluent Validation*

- Separacja logiki biznesowej od walidacji przychodzących żądań
- Przykład

```
[Route("/users")]
public class User
{
    public string Name { get; set; }
    public string Company { get; set; }
    public int Age { get; set; }
    public int Count { get; set; }
}
```

Walidacja – *Fluent Validation* (2)

```
public class UserValidator : AbstractValidator<User> {
    public UserValidator() {
        //Validation rules for all requests
        RuleFor(r => r.Name).NotEmpty();
        RuleFor(r => r.Age).GreaterThan(0);
        //Validation rules for GET request
        RuleSet(ApplyTo.Get, () => {
        RuleFor(r => r.Count).GreaterThan(10);});
        //Validation rules for POST and PUT request
        RuleSet(ApplyTo.Post | ApplyTo.Put, () => {
                RuleFor(r => r.Company).NotEmpty();});
    }
}
```

Walidacja – *Fluent Validation* (3)

- Request:

```
POST localhost:50386/validated { "Name": "Max" }
```

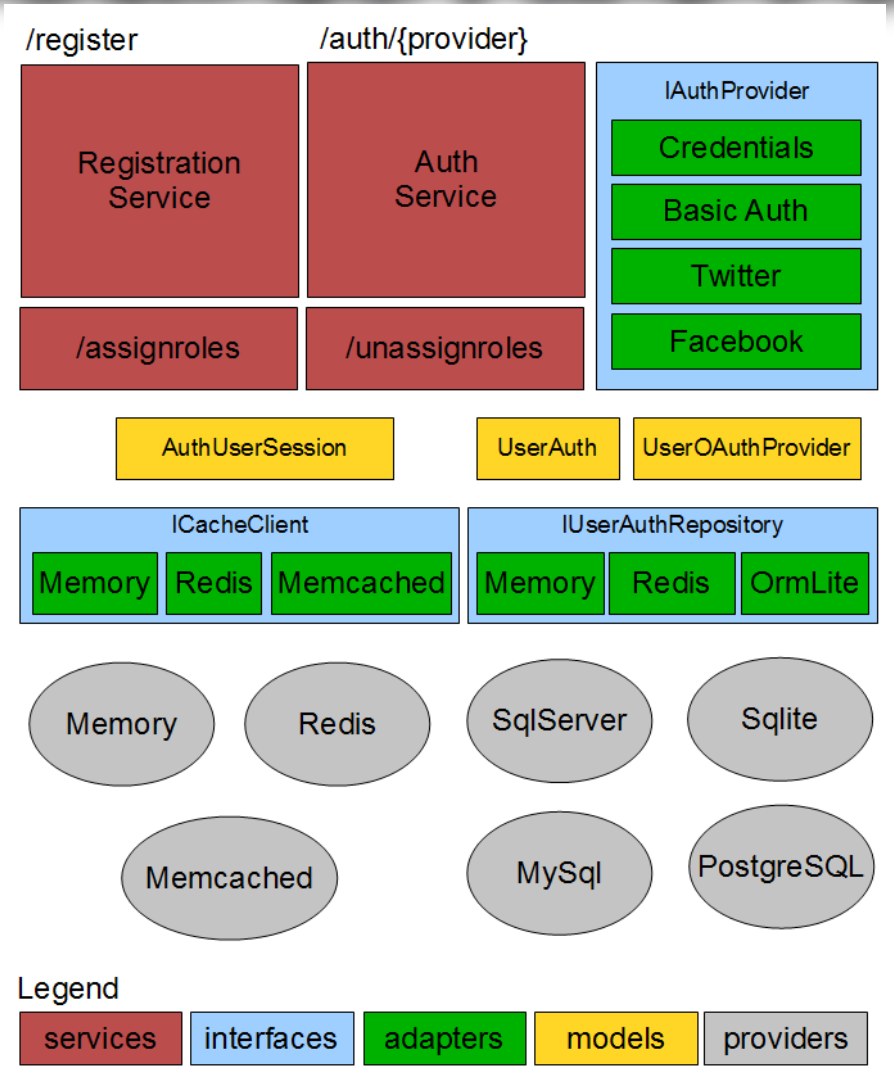
- Response:

```
{   "ErrorCode": "GreaterThan",
    "Message": "'Age' must be greater than '0'.",
    "Errors": [ {
            "ErrorCode": "GreaterThan",
            "FieldName": "Age",
            "Message": "'Age' must be greater than '0'."
        },   {
            "ErrorCode": "NotEmpty",
            "FieldName": "Company",
            "Message": "'Company' should not be empty." }]}
```

Bezpieczeństwo

- Powiązanie kilku aspektów
  - Uwierzytelnianie i autoryzacja
    - Zarządzanie użytkownikami
    - Standardy uwierzytelniania
  - Sesja użytkownika
    - Cache
    - Przechowywanie

Bezpieczeństwo (2)



Bezpieczeństwo (3)

- Uwierzytelnianie
  - Credentials – usługa uwierzytelniająca
  - Basic Auth – HTTP Basic Authentication
  - Digest Auth – HTTP Digest Authentication
  - Custom Credentials – własne implementacje
  - OAuth: Twitter, Facebook, Yammer
  - OpenId: Google, Yahoo
  - OAuth2: Google, LinkedIn

Bezpieczeństwo (4)

- Przechowywanie danych uwierzytelniania
  - OrmLite
  - Redis
  - In Memory
  - Mongo DB
  - Raven DB
  - NHibernate
- Caching: In Memory, Memcached, Redis, AWS, Azure

Bezpieczeństwo – przykład

- Konfiguracja

```
public override void Configure(Container container)
{
    Plugins.Add(new AuthFeature(() => new AuthUserSession(),
      new IAuthProvider[] { new CredentialsAuthProvider() }));

    Plugins.Add(new RegistrationFeature());

    container.Register<ICacheClient>(new MemoryCacheClient());
    var userRep = new InMemoryAuthRepository();
    container.Register<IUserAuthRepository>(userRep);
}
```

Bezpieczeństwo – przykład (2)

- Żądanie uwierzytelnienia (CredentialsAuthProvider)

```
POST localhost:6339/auth/credentials?format=json

{

    "UserName": "admin",

    "Password": "test",

    "RememberMe": true

}
```

Bezpieczeństwo – przykład (3)

- Po uwierzytelnieniu klient otrzymuje Auth Cookie, na jego podstawie znajdowana jest sesja
- Możliwe własne implementacje *IUserAuthSession*
- Obsługa sesji

```
public class SecuredService : Service {
    public object Get(Secured request)
    {
        IAuthSession session = this.GetSession();
        return new SecuredResponse() { Test = "Hi " + session.FirstName };
    }
}
```

Bezpieczeństwo – przykład (4)

- Wbudowane podstawowe zarządzanie prawami, możliwe do określenia na poziomie DTO lub usługi

```
[Authenticate]

[RequiredRole("Admin")]

[RequiredPermission("CanAccess")]

[RequiredPermission(ApplyTo.Put | ApplyTo.Post, "CanAdd")]

[RequiredPermission(ApplyTo.Delete, "AdminRights", "CanDelete")]

public class Secured {

    public bool Test { get; set; }

}

public class SecuredService : Service {

    public object Get(Secured request) { ...
```

Request / response filters

- Globalne filtrowanie żądań lub odpowiedzi, np.

```
this.RequestFilters.Add((httpReq, httpResp, requestDto) => {
    var sessionId = httpReq.GetCookieValue("user-session");
    if (sessionId == null)
        httpResp.ReturnAuthRequired();
});


this.ResponseFilters.Add((req, res, dto) => {
    if (req.ResponseContentType == ContentType.Csv)
        res.AddHeader(HttpHeaders.ContentDisposition,
            string.Format("attachment;filename={0}.csv",
req.OperationName));
});
```

Klienci usług

- Usługi silnie typowane na podstawie DTO

```
HelloResponse response = client.Get(new Hello { Name = "World!" });
response.Result.Print();


client.GetAsync(new Hello { Name = "World!" },
    r => r.Result.Print(), (r, ex) => { throw ex; });


var client = new JsonServiceClient(baseUri) {
    UserName = UserName,
    Password = Password };
var request = new Secured { Name = "test" };
var response = client.Send<SecureResponse>(request);
```

Inne – Auto Mapper

- Wspiera konwersję między DTO a obiektami biznesowymi

```
var dto = viewModel.TranslateTo<MyDto>();

var dto = new MyDto { A = 1, B = 2 }.PopulateWith(viewModel);

var dto = new MyDto { A = 1, B = 2 }
.PopulateWithNonDefaultValues(viewModel);
```

Inne – OrmLite

- Prosty ORM: 1 klasa – 1 tabela
- Wspiera SQL Server, Oracle, MySQL, PostgreSQL, SQLite, Firebird
- Przykład:

```
var people = db.SelectParam<Person>(q => q.Age == 27);

var person = db.GetByIdParam<Person>(1);

db.CreateTable<Employee>();

db.Insert(new Employee { Id = 1, Name = "Employee 1" });
```

Inne

- Wewnętrzne logowanie na podstawie wielu dostawców logu
- Automatyczne logowanie żądań
- Wbudowany mini profiler
- Cache również dla odpowiedzi usług
- Klient w JavaScript