

WEBSERVICES

Verteilte Systeme

Prof. Dr. Georg Hinkel
17.05.2023

GLIEDERUNG

Datum	Vorlesung	Übungsblatt	Abgabe
19.04.2024	Einführung	HamsterLib	06.05.2024
26.04.2024	Netzwerkprogrammierung	Theorie	
03.05.2024	World Wide Web	HamsterRPC 1	20.05.2024
10.05.2024	Remote Procedure Calls	Theorie	
17.05.2024	Webservices	HamsterRPC 2	03.06.2024
24.05.2024	Fehlertolerante Systeme	Theorie	
31.05.2024	Transportsicherheit	HamsterREST	17.06.2024
07.06.2024	Architekturen für Verteilte Systeme	Theorie	
14.06.2024	Internet der Dinge	HamsterIoT	01.07.2024
21.06.2024	Namen- und Verzeichnisdienste	Theorie	
28.06.2024	Authentifikation im Web	HamsterAuth	15.07.2024
05.07.2024	Infrastruktur für Verteilte Systeme	Theorie	
12.07.2024	Wrap-Up	HamsterCluster (Bonus)	16.08.2024

Agenda

- Wiederholung Service-orientierte Architekturen (SOA)
- WebService Definition Language (WSDL), SOAP
- Representational State Transfer (REST)
- Open API (Swagger)

Lernziele

- WSDL und SOAP erklären können
- RESTful Webservices analysieren können
- OpenAPI Spezifikationen erklären können

- Beobachtung: Port 80/443 werden häufig in der Firewall geöffnet
 - World Wide Web
 - Keine/wenig zusätzlichen Konfigurationen mehr notwendig
 - Außerdem: Präferenz für textuelle Nachrichtenformate (XML, JSON)
- Überlegung: Implementiere Verteilte Systeme über HTTP
 - Service-orientierte Architekturen (SOAP)
 - RESTful Webservices

„TYRANNEI“ VON JAVASCRIPT

- JavaScript ursprünglich entwickelt 1995 von Brendan Eich (Netscape)
 - Entwicklungszeit: 10 Tage, Grundidee: Portierung von Scheme (Lisp-Dialekt)
 - Fokus zu Beginn: clientseitige Validierung von Nutzereingaben
- Entwicklungsschub im Rahmen des Browserkriegs
 - Dadurch sehr hohe Verbreitung „out-of-the-box“
 - Später zunehmend Aufkommen von Skripten, um Roundtrips zu sparen
 - Verbesserung der Performance
- Konsequenz: JavaScript als einzig zuverlässig vorhandene, clientseitige Scripting-Technologie
 - Im Gegensatz zu allen anderen Arten von Softwareprojekten
 - ➔ Optimierung von verteilten Systemen auf JavaScript
 - ➔ Verwendung von XML/JSON als Nachrichtenformat

Mittlerweile WebAssembly,
ermöglicht prinzipiell Freiheit der
Programmiersprache

HAUPTTREIBER GESCHÄFTSANWENDUNGEN

Wiederholung

- Funktionalität
 - Flexibles Abbilden heutiger und künftiger Geschäftsprozesse
 - Integration existierender Systeme (Legacy)
 - Interoperabilität mit Fremdsystemen
- Niedrige Kosten
 - Verringerung der Entwicklungszeit (time-to-market)
 - Verringerung der Entwicklungskosten (insb. in der Wartung)
 - Verringerung der Betriebs-/Managementkosten (total cost of ownership)

➔ Wiederverwendung als wichtiger Lösungsansatz

SERVICE-ORIENTIERTE ARCHITEKTUREN (SOA)

Wiederholung

- Architekturansatz für Geschäftsanwendungen
 - Aufteilung der benötigten Funktionalität in fachlich und organisatorisch getrennte Dienste
 - Separate Entwicklung der einzelnen Dienste
- Selbstbeschreibung
 - Web Service Definition Language (WSDL), W3C-Standard
 - Definiert Parameter und Rückgaben durch XML Schema
- Zentraler Registrierung der Dienste (Service Registry)
 - Aufruf der Dienste über zwischengeschaltete Stelle (Enterprise Service Bus)
- Anwendungen bestehen aus Integrationen von Diensten

- Geschäftsprozess = komplexe Interaktion zwischen Diensten
- Idee: Formale Modellierung von Geschäftsprozessen
 - Erlaubt automatische Ausführung → Web Service Orchestration
 - Programmieren im Großen (Web Services als Einheiten)
- WS-BPEL (Business Process Execution Language)
 - OASIS Standard
 - Mittlerweile nicht mehr bedeutend
- BPMN (Business Process Model and Notation)
 - OMG Standard, verwandt zu UML Aktivitätsdiagramm
 - ISO/IEC 19510

Hierfür notwendig: Standardisierter
Nachrichtenaustausch, formale
Definition der einzelnen Dienste

WEB SERVICE DEFINITION LANGUAGE (WSDL)

- W3C-Empfehlung zur abstrakten Beschreibung von Web-Services (2000)
 - Aktueller Stand WSDL 2.0 von 2007
- Abstrakte Dienstbeschreibung inkl. Parameter, Rückgabe, Fehler
- Technische Transportinformationen
 - Aufrufsemantik, Adresse

```
<types>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.tmsws.com/v1"
    targetNamespace="http://www.example.com" >
    <xs:element name="request"> ... </xs:element>
    <xs:element name="response"> ... </xs:element>
  </xs:schema>
</types>
<interface name="Interface1">
  <fault name="Error1" element="tns:response"/>
  <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
    <input messageLabel="In" element="tns:request"/>
    <output messageLabel="Out" element="tns:response"/>
  </operation>
</interface>
```

XML Schema zur Beschreibung der Daten in XML

Definition von Operationen

```
<binding name="HttpBinding" interface="tns:Interface1"
  type="http://www.w3.org/ns/wsdl/http">
  <operation ref="tns:Get" http:method="GET"/>
</binding>
...
<service name="Service1" interface="tns:Interface1">
  <endpoint name="HttpEndpoint"
    binding="tns:HttpBinding"
    address="http://www.example.com/interface1"/>
  ...
</service>
```

Beschreibung der Transportschicht(en)

Beschreibung der Endpunkte, mehrere Endpunkte möglich

- Entwicklung von XMLHttpRequest als JavaScript-API zum Zugriff auf XML-Webservices aus JavaScript
 - Ursprünglich proprietär von Microsoft (1999)
 - Später standardisiert von W3C (2006)
 - Erlaubt Webservices aus JavaScript heraus aufzurufen
- Idee: Interaktive Webseiten, die Content dynamisch per Webservice nachladen
 - Webservices per WSDL standardisiert → Abruf der Daten in XML Format
 - Vorläufer von Single-Page-Anwendungen
 - Konzept als „Asynchronous JavaScript And XML“ (Ajax) bekannt geworden

SOAP

Ehemals Simple Object Access Protocol

- W3C-Empfehlung, Protokoll zum Austausch von XML-Daten
 - 1999, Ziel: Standardisierter Nachrichtenaustausch unabhängig vom Transportprotokoll
 - Definition eines Nachrichtenformats, ähnlich wie RPCs
 - Transportschicht per HTTP, TCP, SMTP oder JMS

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Header>
    <m:RequestID xmlns:m="http://www.lecture-db.de/soap">a3f5c109b</m:RequestID>
  </s:Header>
  <s:Body>
    <m:DbResponse xmlns:m="http://www.lecture-db.de/soap">
      <m:title value="DOM, SAX und SOAP">
        <m:Choice value="1">Arbeitsbericht Informatik</m:Choice>
        <m:Choice value="2">Seminar XML und Datenbanken</m:Choice>
      </m:title>
    </m:DbResponse>
  </s:Body>
</s:Envelope>
```

- Häufigste Kritik: SOAP als nutzloser Ballast
 - Die meisten Dienste nutzen HTTP als Transportschicht → Firewalls
 - Redundanzen bei Verwendung von HTTP als Transportschicht
 - HTTP definiert schon Adressen, warum dann nochmal Adressen in SOAP Envelop?
 - HTTP definiert schon Header, warum dann nochmal Header in SOAP Envelop?
 - HTTP definiert schon Verben, warum dann nochmal Operationsnamen im SOAP Envelop?
 - HTTP definiert schon Statuscode, warum dann nochmal Status in SOAP Envelop?
 - HTTP definiert schon Body, warum dann nochmal in XML einpacken?
- Redundanzen führen zu Inkonsistenzen, führen zu Verwirrung
- Schlechtere Laufzeit, langsamere Entwicklung

REPRESENTATIONAL STATE TRANSFER (REST)

Einführung

- Ziel: Erfüllung der wichtigsten Anforderungen für Web-basierte Architekturen mit niedriger Einstiegshürde
 - Dissertation Roy Fielding, 2000 „Architectural Styles and the Design of Network-based Software Architectures“
- Idee: Direkte Nutzung von HTTP, aber Architekturrichtlinien zur Umsetzung
 - Client-Server: Kommen wir nachher noch drauf
 - Zustandslose Webservices, d.h. jede Anfrage enthält alle für die Verarbeitung relevanten Informationen
 - Explizite Caching-Informationen, Server definiert genau ob und wie lange Antworten gecacht werden dürfen
 - Einheitliche Schnittstelle, in der Praxis faktisch immer HTTP(S)
 - Mehrschichtige Systeme: 3-Tier, N-Tier oder Microservices
- Webservices, die nach REST-Prinzipen entwickelt wurden, heißen RESTful Webservices

REPRESENTATIONAL STATE TRANSFER (REST)

Einheitliche Schnittstelle

- Adressierbarkeit von Ressourcen
 - Jede logische Ressource bekommt eigene URL
 - Standardisiert Zugriffsweg
- Beispiel: Aufruf statischer Webseiten
 - URL enthält physischen Dateipfad
 - Content enthält Links zu anderen Ressourcen
- Praxis bei Webservices: Identifier des Objektes ist Teil der URL
 - <https://www.example.com/orders/57/items>

REPRESENTATIONAL STATE TRANSFER (REST)

Einheitliche Schnittstelle

- Verwende HTTP-Methoden (hauptsächlich Standardmethoden aus HTTP/1.1) zur Beschreibung der Operationen

Verb	Bedeutung	Anmerkung
GET	Abrufen der angefragten Ressource	Seiteneffektfrei
POST	Ressource übertragen, Verwendung auch für andere Zwecke mit Seiteneffekten	
PUT	Ressource anlegen	Idempotent
PATCH	Ressource aktualisieren	
DELETE	Ressource löschen	Idempotent
HEAD	Nur Header abfragen → Metadaten	Seiteneffektfrei
OPTIONS	Prüft die zur Verfügung stehenden Optionen	Seiteneffektfrei
TRACE	Echo der Anfrage	Seiteneffektfrei

Zustand der Ressource wird nicht geändert

Wiederholte Aufruf derselben URL ist seiteneffektfrei

REPRESENTATIONAL STATE TRANSFER (REST)

Hypermedia as the Engine of Application State (HATEOAS)

- Problem: Woher weiß der Client, welche Ressourcen auf dem Server existieren?
- Antwort bisher: Steht im WSDL-Dokument
- Problem laut Fielding: zu unflexibel, Änderungen einer Ressource machen Beschreibung obsolet
 - Stattdessen Navigation über Links, die im Inhalt zurückgeliefert werden
 - Vorteil: Links können vom Objekt abhängen
 - Wichtig e.g. bei Paging
 - Problem durch Generierung der Service-Beschreibung eingedämmt

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

```
{
  "account": {
    "account_id": "123abc",
    "balance": {
      "currency": "EUR",
      "value": 100.0
    },
    "links": {
      "deposit": "/accounts/123abc/deposit",
      "withdraw": "/accounts/123abc/withdraw",
      "transfer": "/accounts/123abc/transfer",
      "close": "/accounts/123abc/close"
    }
  }
}
```


- Problem: Webservice liefert logisch sehr viele Elemente zurück (Suchergebnisse)
 - Darstellung im Client begrenzt, zeige nur eine Auswahl der Elemente
 - Links auf vorherige/erste/nächste/letzte Page in der Response
 - Standardisierter „Ort“ für Links, um Pagination-Code wiederverwenden zu können
- Lösung: dedizierte Response Header

```
curl https://api.github.com/repos/grpc/grpc/issues -v
```



Verbose, erzwingt
Darstellung der Header

```
HTTP/1.1 200 OK
```

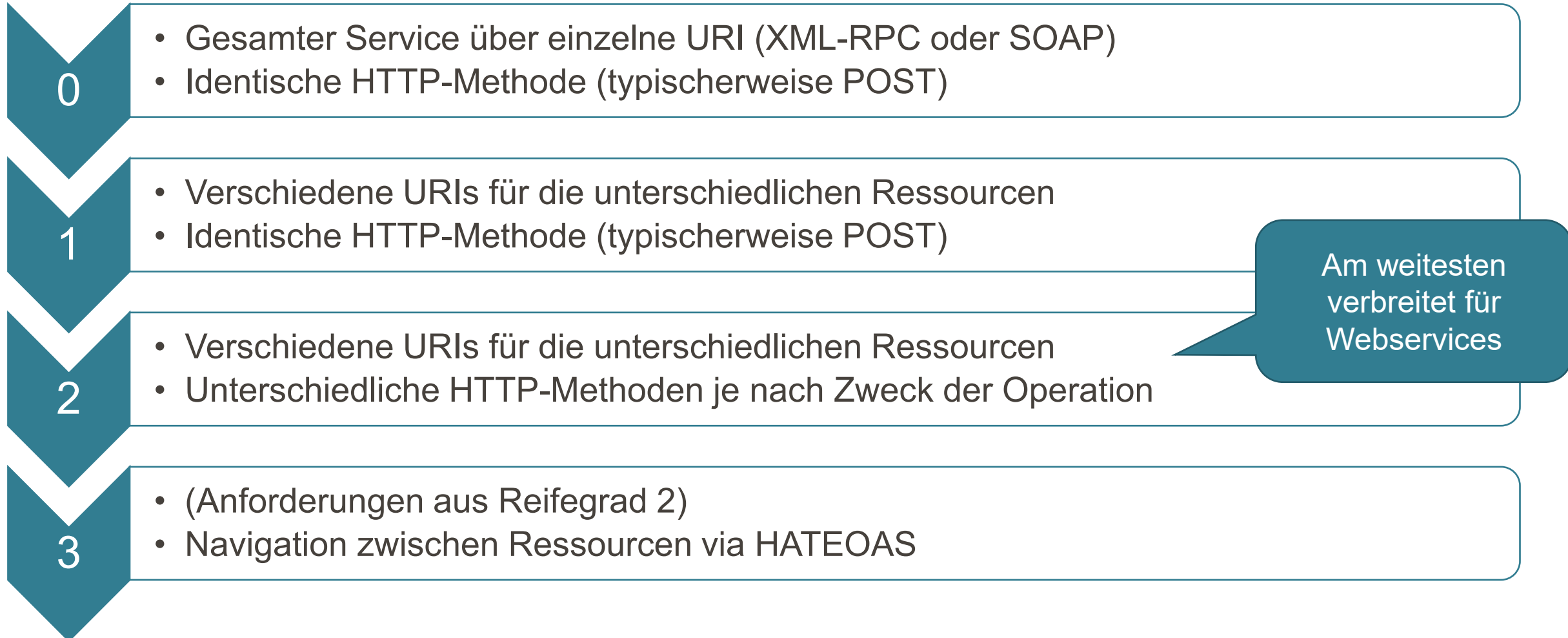
```
...
```

```
Link: <https://api.github.com/repositories/27729880/issues?page=2>; rel="next",  
<https://api.github.com/repositories/27729880/issues?page=27>; rel="last"
```

```
...
```

REPRESENTATIONAL STATE TRANSFER (REST)

Richardson Maturity Model (RMM)



- HTTP-Implementierungen heute üblicherweise durch Webserver gekapselt, Programmierung über spezifische Web-Frameworks
- Typische Bestandteile
 - Controller: Verantwortlich für Anfragen an Gruppen von Adressen („Routen“)
 - Dependency Injection: Webserver löst Instanzen für Controller typischerweise per DI auf
 - Request-Pipeline, in die generisch Code injiziert werden kann (e.g. Authentifizierung)
 - Template-Engine: Nutzlast oft template-gesteuert, insb. bei HTML-Seiten
 - Serializer: Typisierte Objekte werden in Netzdatenformat konvertiert, falls kein HTML

PROGRAMMIERUNG VON WEBSEITEN

Beispiel: SpringBoot → Apache Tomcat

```
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
```

```
@SpringBootApplication
```

```
@RestController
```

```
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
    @GetMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
        return String.format("Hello %s!", name);
    }
}
```

Routen können
Platzhalter für
Parameter enthalten

PROGRAMMIERUNG VON WEBSEITEN

Beispiel ASP.NET Core → Kestrel, IIS oder http.sys

```
using Microsoft.AspNetCore.Mvc;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
var app = builder.Build();
app.MapControllers();
app.Run();

[ApiController]
[Route("hello")]
public class DemoController : ControllerBase
{
    [HttpGet]
    public string Hello( string? name = "World")
    {
        return $"Hello {name}!";
    }
}
```

OPENAPI SPEZIFIKATION

Früher: Swagger Spezifikation

- Spezifikationsformat für REST-basierte Webservices
 - OpenAPI Initiative, hervorgegangen aus Projekt Swagger, 2011
- Open-Source-Tools
 - Swagger Editor: Editor zum Erstellen von API-Definitionen
 - **Swagger CodeGen**: Generiert clients (und server stubs) in verschiedenen SDKs
 - **Swagger UI**: Erzeugt Dokumentation
- Weitere kostenpflichtige Tools
 - SwaggerHub für Kollaboration
 - SwaggerHub Enterprise: Standardisierung der APIs über Unternehmensgrenzen hinweg
 - Swagger Inspector: Test-Tool
- Mittlerweile Tools um API-Spezifikationen vom Webserver generieren zu lassen



BEISPIEL

Siehe Vorlesung World Wide Web

`using Microsoft.AspNetCore.Mvc;`

```
var builder = WebApplication.CreateBuilder(args);  
builder.Services.AddControllers();  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();  
var app = builder.Build();  
app.MapControllers();  
app.UseSwagger();  
app.UseSwaggerUI();  
app.Run();
```

```
[ApiController]  
[Route("hello")]  
public class DemoController : ControllerBase  
{  
    [HttpGet]  
    public string Hello( string? name = "World")  
    {  
        return $"Hello {name}!";  
    }  
}
```

BEISPIEL

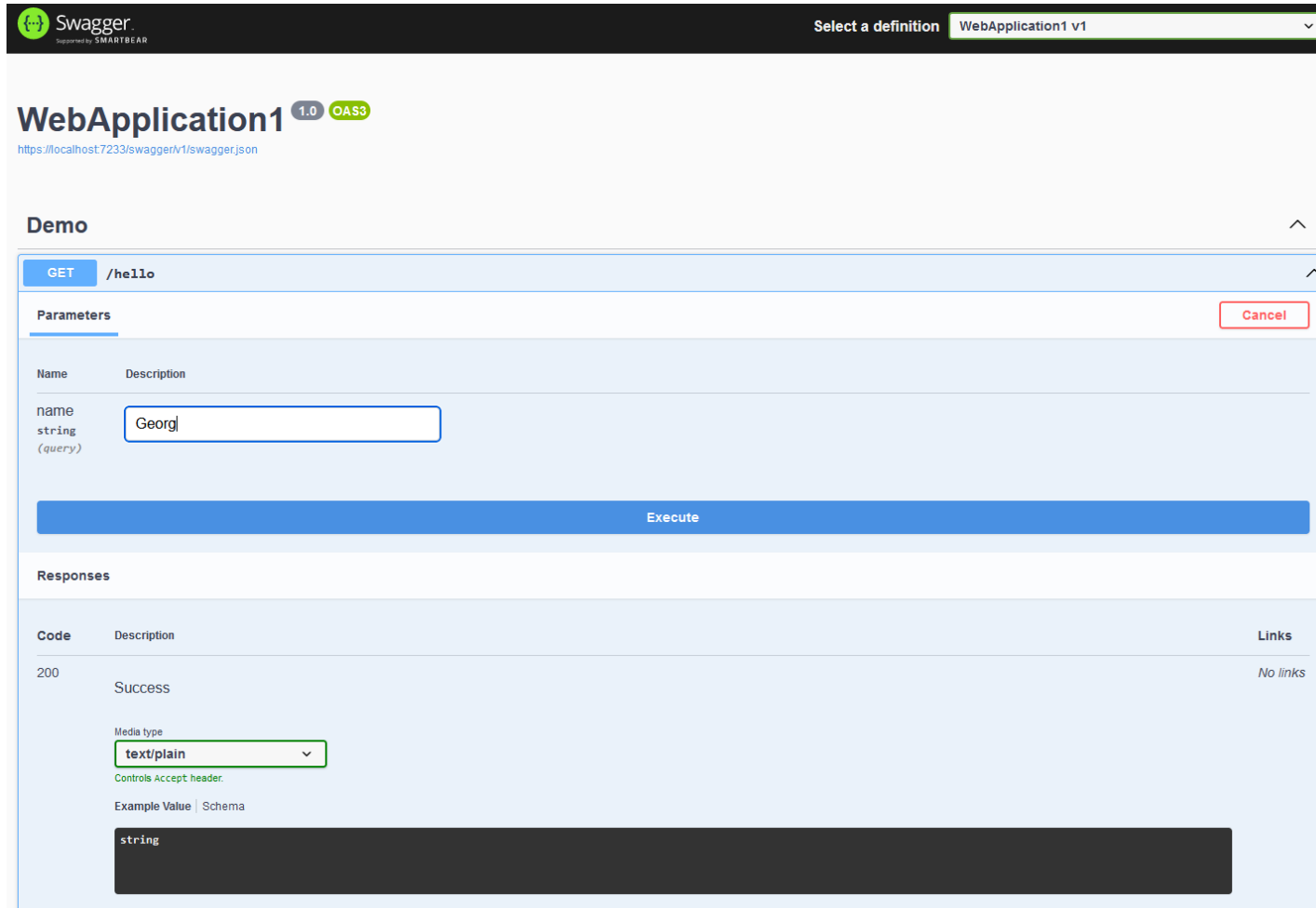
OpenAPI Spezifikation

```
{
  "openapi": "3.0.1",
  "info": { "title": "WebApplication1", "version": "1.0" },
  "paths": {
    "/hello": {
      "get": {
        "tags": [ "Demo" ],
        "parameters": [ { "name": "name", "in": "query", "schema": { "type": "string", "default": "World" } } ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "text/plain": { "schema": { "type": "string" } },
              "application/json": { "schema": { "type": "string" } },
              "text/json": { "schema": { "type": "string" } }
            }
          }
        }
      }
    }
  },
  "components": { }
}
```

Details typischerweise über Annotationen
und Inline-Dokumentation

BEISPIEL

Swagger UI



The image shows the Swagger UI interface for a web application. At the top, the Swagger logo is on the left, and a dropdown menu labeled "Select a definition" shows "WebApplication1 v1". Below this, the title "WebApplication1" is displayed with a "1.0" version tag and an "OAS3" specification tag. A URL "https://localhost:7233/swagger/v1/swagger.json" is provided. The "Demo" section is active, showing a "GET /hello" endpoint. Under the "Parameters" tab, a query parameter "name" of type "string" is defined, with a text input field containing "Georg". A blue "Execute" button is located below the parameters. The "Responses" section shows a "200" status code with a "Success" description. A "Media type" dropdown is set to "text/plain". Below this, there are tabs for "Example Value" and "Schema", with the "Example Value" tab currently selected, showing a "string" type in a dark box.

Swagger
Supported by SMARTBEAR

Select a definition WebApplication1 v1

WebApplication1 1.0 OAS3

<https://localhost:7233/swagger/v1/swagger.json>

Demo

GET /hello

Parameters Cancel

Name	Description
name string (query)	<input type="text" value="Georg"/>

Execute

Responses

Code	Description	Links
200	Success	No links

Media type
text/plain

Controls Accept header.

Example Value | Schema

string

WAS IST DANN DER UNTERSCHIED ZU GRPC?

Vergleich RPC - Webservices

WEBSERVICES VS. GRPC

Kategorie	gRPC	RESTful Webservices
Schnittstellenbeschreibung	Proto-Files, „verpflichtend“	Open API, optional
Transportprotokoll	HTTP/2.0, QUIC	Ab HTTP/1.1
Datenformat	Protocol Buffer	JSON
Adressierung	Methode	Ressource
Methodenauswahl	Adresse	Adresse + Verb
Ressourcenauswahl	Parameter	Adresse
Standardisierung	De-facto / Industriestandard	HTTP als W3C-Standard
Kommunikationsmuster	Request/Reply, Publish/Subscribe, Duplex	Nur Request/Reply
Hauptvorteil	Performance	Universalität

- Problem: RESTful Webservices sind nur Request/Reply
 - Server kann keine Antwort ohne zugehörige Frage senden
- Lösung: Websockets
 - Aufbau des Websockets mittels „normalem“ HTTP Request
 - Authentifizierung mittels HTTP Header
 - TLS wie bei HTTP(S)
 - Adressierung mittels HTTP Adresse
 - Gleicher Port wie RESTful Services
 - Aber: Anderes URI-Schema ws:// (unverschlüsselt) bzw. wss:// (verschlüsselt)
 - Server antwortet mit Status 101 Protocol Upgrade zu Web Socket
 - Anschließend wird der TCP Socket beliebig weiter verwendet

- Web-Services als Mittel, um Dienste per HTTP zugänglich zu machen
 - SOAP als frühes Protokoll, WSDL als Schema
- Heute verbreiteter: RESTful Webservices, Paradigma zur Verwendung von HTTP
- OpenAPI/Swagger als Spezifikationsformat für RESTful Web-Services



- Warum wird für Dienste oft HTTP als Transportschicht eingesetzt?
- Erläutern Sie den Einsatzzweck von WSDL und SOAP!
- Erläutern Sie die Redundanzen beim Versenden von SOAP Nachrichten über HTTP!
- Wann sind Webservices „RESTful“?
- Welche Formate gibt es, um Webservices zu spezifizieren?
- Was verbirgt sich hinter der Abkürzung HATEOAS?
- Beurteilen Sie eine gegebene Schnittstelle nach dem Richardson Maturity Model!

FEHLERTOLERANTE SYSTEME

Verteilte Systeme

Prof. Dr. Georg Hinkel
24.05.2024