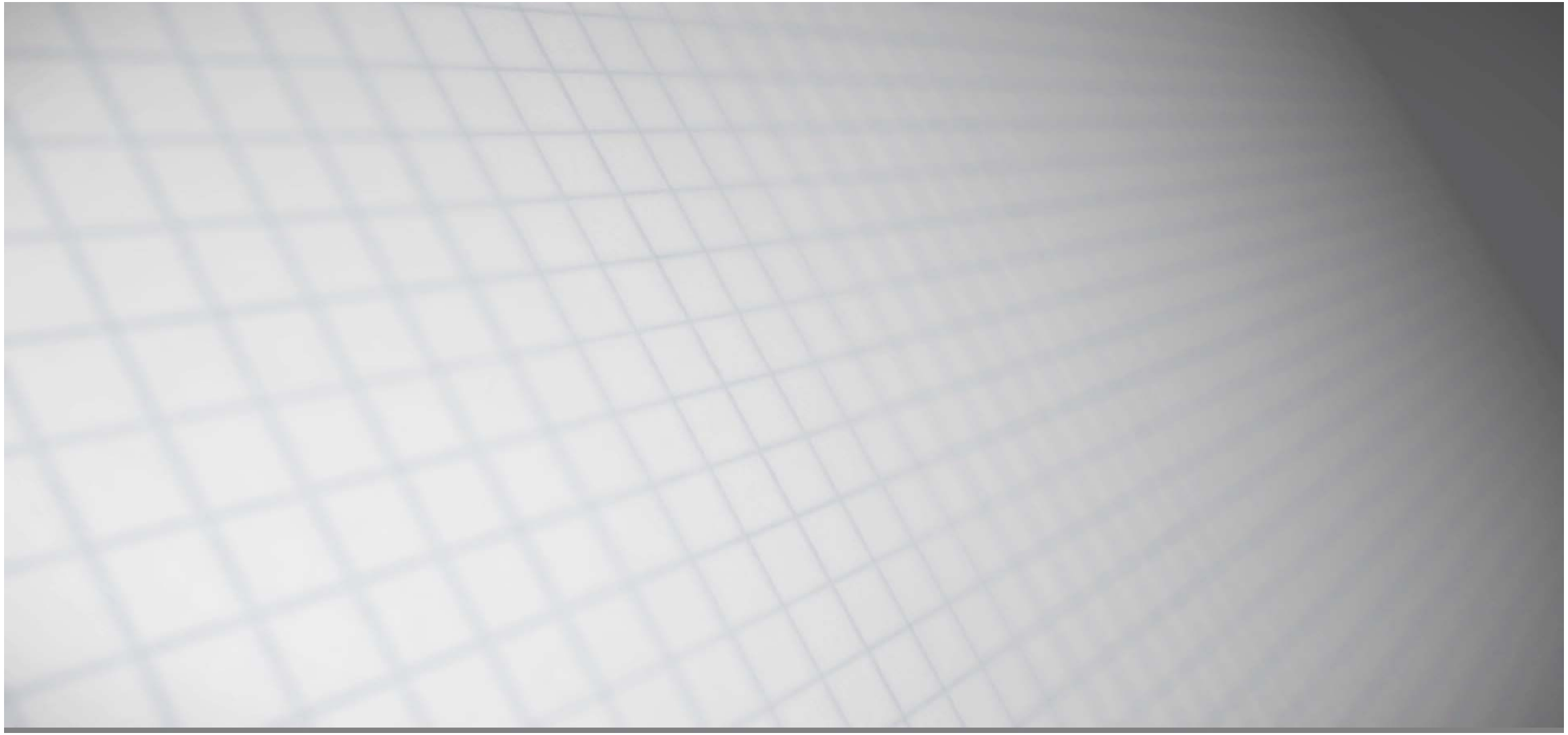
REST API Documentation Using OpenAPI (Swagger)

Modern technology for modern web frontends

Martyn Kemp, Consultingwerk Ltd.
martyn.kemp@consultingwerk.de



Consultingwerk Ltd.

- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany, subsidiary in UK
- Customers in Europe, North America, Australia and South Africa
- Vendor of developer tools and consulting services
- 27 years of Progress experience (V5 … OE11)
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration

Agenda

- OpenAPI (formerly known as Swagger)
- OpenAPI Swagger Structure
- OpenAPI Swagger Online Editor
- Consultingwerk Swagger
- Using Consultingwerk Swagger with RESTful Services

OpenAPI

- **What is OpenAPI?**
  - Swagger is a framework of API developer tools for the OpenAPI Specification (OAS)
  - Allows you to describe your entire API
  - Enables development across the entire API lifecycle, from design and documentation, to test and deployment
  - A powerful definition format for describing and creating RESTful API's, which are easy to understand, readable and language agnostic

OpenAPI

- **What is Swagger?**
  - Swagger is a set of open-source tools built around the OpenAPI Specification
  - Enables you to design, build, document and consume your REST API's
  - Can be written in JSON or YAML (Yet Another Markup Language)

OpenAPI

- **Faster, StandardizedAPI Design**
  - Design APIs in a powerful and intuitive editor that is built for speed and efficiency, without any loss in design consistency
- **Centralized, SecureAPI Collaboration**
  - Seamlessly work across multiple teams on your API development with controlled, centralized access and an optimized collaborative workflow
- **Hosted, InteractiveAPI Documentation**
  - Autogenerate interactive API Documentation straight from the contract and securely host it, making them easy to use and adopt by internal and external users

OpenAPI

- **Design**
  - Commonly achieved through the Swagger Editor
  - http://editor2.swagger.io
  - Using the on line editor, you can edit existing API's or design new API's, which visually renders your Swagger Definition with real time feedback and error handling
- **Build**
  - Generate code from your API’s using the Swagger Codegen Tool
- **Document**
  - Using the Swagger UI, you can visualise and interact with your Swagger Definitions

Swagger – Resources

- Swagger - https://swagger.io/
  - Swagger Editor https://swagger.io/swagger-editor/
    - It’s clean, efficient, and armed with a number of features to help you design and document your RESTful interfaces, straight out of the box
  - Swagger Codegen https://swagger.io/swagger-codegen/
    - Can simplify your build process by generating server stubs and client SDKs from your OpenAPI specification

Swagger – Resources

- Swagger - https://swagger.io/
  - Swagger UI https://swagger.io/swagger-ui/
    - Allows your development team or your end consumers — to visualize and interact with the API's resources without having any of the implementation logic in place
  - Swagger Inpector https://swagger.io/swagger-inspector/
    - An inspection tool for easily calling and validating REST, GraphQL and SOAP based web services to ensure they function correctly

Swagger – Resources

- **Swagger Hub -** https://swaggerhub.com
  - Platform for Designing and Documenting API's
  - Fastest way for teams to collaborate on their API Development
- **Swagger Blog -** https://swaggerhub.com/blog/
  - Area for keeping up to date on activities and events within the Swagger Community
- **Swagger Docs**
  - https://swagger.io/docs/ (specification/2-0/describing-parameters/)
- **Swagger Pet Store Sample**
  - https://editor.swagger.io/

Agenda

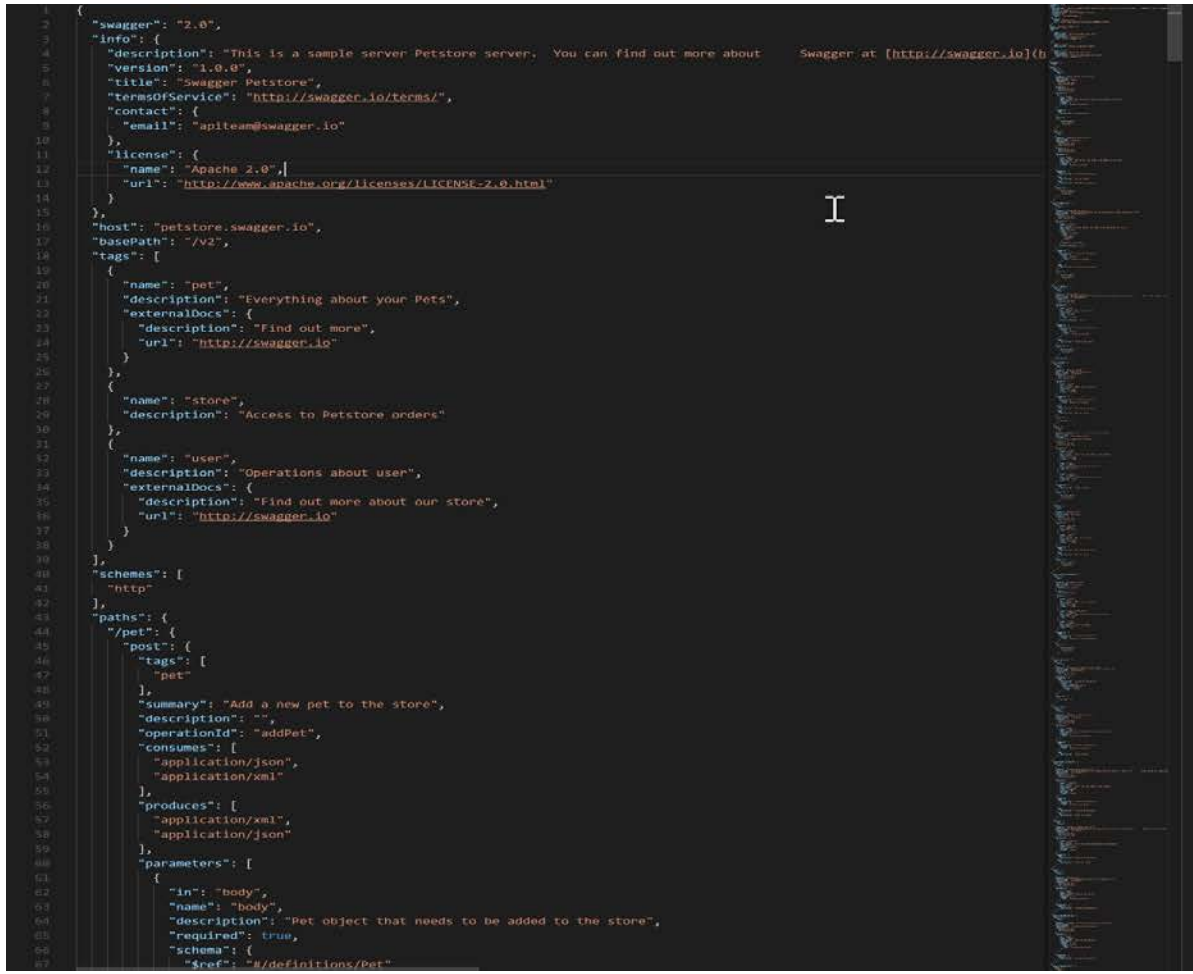
- OpenAPI (formerly known as Swagger)
- OpenAPI Swagger Structure
- OpenAPI Swagger Online Editor
- Consultingwerk Swagger
- Using Consultingwerk Swagger with RESTful Services

Swagger – Basic Structure

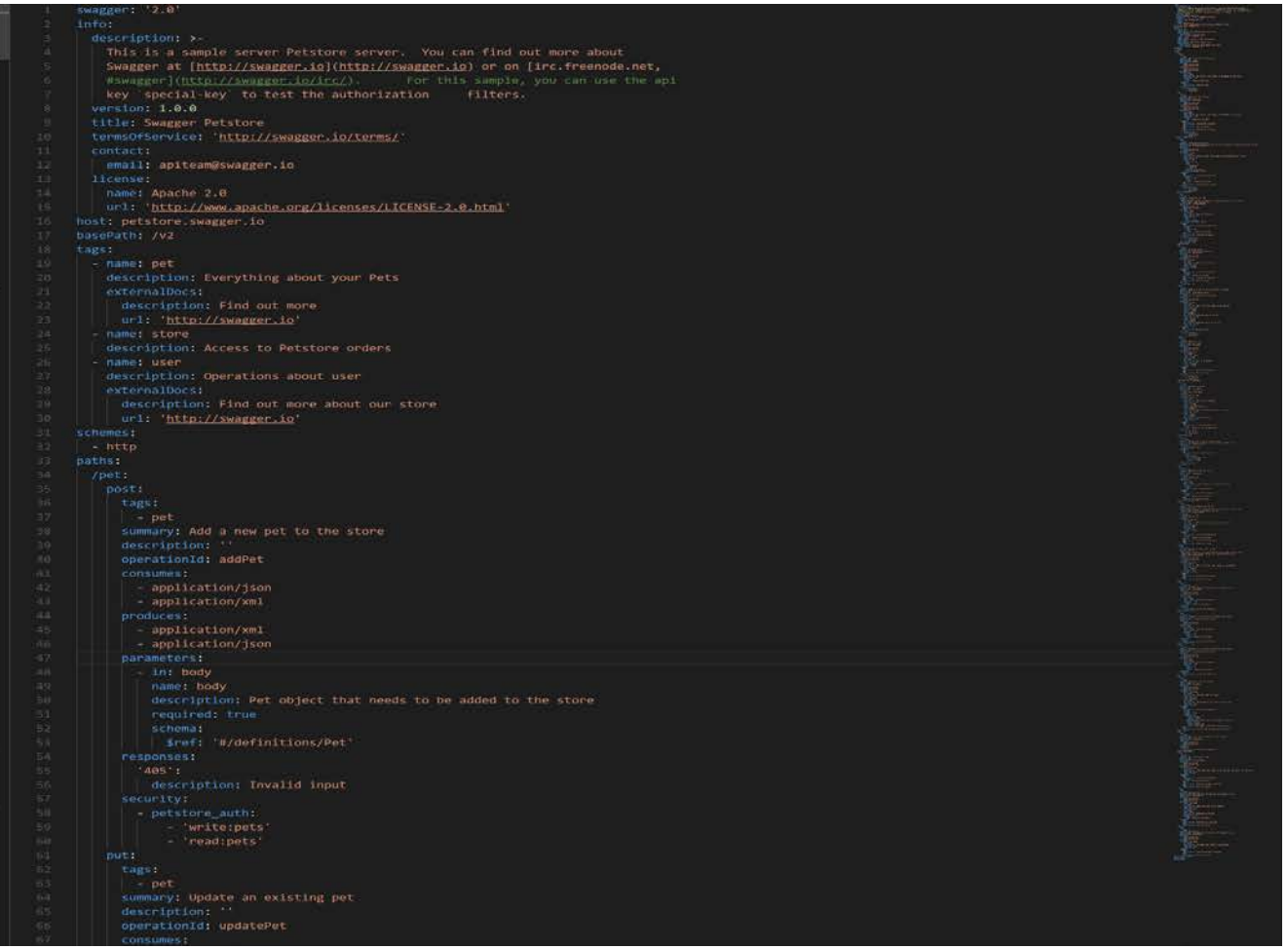
- Metadata
- Base URL
- Consumes, Produces
- Paths
- Operations
- Parameters & Parameter Types
- Responses & Response Media Types
- Input, Output Models (Definitions)

Swagger – Specification

JSON



YAML



Swagger – Metadata

- Metadata
  - Details can be found at https://swagger.io/docs/specification/2-0/basic-structure/
  - Every Swagger specification starts with the Swagger version. A Swagger version defines the overall structure of an API specification — what you can document and how you document it

```
swagger: "2.0"
```

  - Then you need to specify the “info” metadata tag. This is an object that should contain “title” and “version”

```
info: {
        description: “Customer API definition.
                        Some additional text to further enhance your description.”,
        title: “Customer API”,
        version: “1.0.0”
}
```

Swagger – Base URL

- Base URL
  - The base URL for all API calls is defined using schemes, host and basePath:

```
"schemes": [
"http"
],
"host": “machine:port",
"basePath": "\/web\/Resource",
```

    - ***Note: if “host” is not specified, then it is assumed to be the same “host” value that is currently serving the API Documentation***
  - basePath:
    - “basePath” is basically the URL prefix for ALL API Paths relative to the “host” (and must begin with “/”
    - An Example of the concatenated path would be “http://machine:port/web/resource”
    - ***Note: If both “host” and “scheme” are omitted, then the values are derived from machine serving the API Document***

Swagger – Consumes & Produces

- Consumes, Produces
  - The “consumes” and “produces” sections define the MIME (Multipurpose Internet Mail Extensions) types supported by the API

```
"consumes": [
  "application\/json"
],
"produces": [
  "application\/json"
],
```

  - The value of consumes and produces is an array of MIME types
  - Global MIME types can be defined on the root level of an API specification and are inherited by all API operations. However, these can be overridden at operation level

Swagger – Paths

- Paths
  - In the Swagger definition API, paths are resources that your API exposes
    - All paths are relative to basePath. The full request URL is constructed as scheme://host/basePath/path
  - A single path can support multiple operations, however, Swagger defines a unique operation as a combination of a path and an HTTP method. This means that two GET or two POST methods for the same path are not allowed
  - This means that you have to define alternative paths for addition operations, e.g:

```
"/Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity": {
“get”: {}, “delete”: {}, “post”: {}, “put”: {}
}
“/Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity/count": {
"put": { ……..

"/Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity/SubmitData": {
"post": { ……..
```

Swagger – Operations

- **Operations**
  - For each path, you define operations (HTTP methods) that can be used to access that path. Swagger 2.0 supports get, post, put, patch, delete, head, and options
  - Addition resources, such as "Submit", "Count", "Invoke" (Invokable Methods) are defined as additional paths

```
"/Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity/count": {
"put": { ……..

"/Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity/SubmitData": {
"post": { ……...

"/Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity/PutCustomerOnHold": {
"put": {……...
```

Swagger – Parameters

- Parameters
  - API operation parameters are defined under the parameters section in the operation definition. Each parameter has name, value type (for primitive value parameters) or schema (for request body), and optional description

```
"parameters": [
  {
    "in": "body",
    "name": "dsCustomer",
    "required": true,
    "description": "Record to Submit",
    "schema": {
      "$ref": "#/definitions/dsCustomer"
    }
  }
],
```

  - **Note: *Parameters is an Array of one or more objects***

Swagger – Parameter Types

- Parameter Types
  - Swagger distinguishes between the following parameter types based on the parameter location. The location is determined by the parameter's "**in**" key, for example:
    - query parameters, such as /customers?filter={filter}
    - path parameters, such as /customer/{custnum}

```
"/Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity?filter={filter}":
"parameters": [ { "name": "filter", "in": "path", …. } ]

"/Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity? {custnum}":
"parameters": [ { "name": "custnum", "in": "query", …. } ]
```

  - Required and Optional Parameters

    By default, Swagger treats all request parameters as optional. You can add required: true to mark a parameter as required. Note that path parameters must have required: true, because they are always required

```
"parameters": [ {……, "required": true, …. }]
```

Swagger – Responses

- Responses
  - An API specification needs to specify the responses for all API operations. Each operation must have at least one response defined, usually a successful response. A response is defined by its HTTP status code and the data returned in the response body and/or headers
  - An example response for a “Count” operation:

```
"responses": {
  "200": {
    "description": "successful operation",
    "schema": {
      "$ref": "#\/definitions\/Count"
    }
  },
  "401": { "description": "Unauthorized“ },
  "404": {  "description": "Not found“ }
}
```

Swagger – Media Types

- Response Media Types
  - An API specification needs to specify the responses for all API operations. Each operation must have at least one response defined, usually a successful response. A response is defined by its HTTP status code and the data returned in the response body and/or headers
  - An API can respond with various media types. JSON is the most common format, but certainly not the only one

```
"produces": [
    "application/json"
]
```

Swagger – Response Body

- Response Body
  - The schema keyword is used to describe the response body
  - A schema can define:
    - An “object” or “array” – this format is usually represented with JSON and XML API’s
    - A primitive such as a number or string – this is typically used for plain text responses
  - A schema can be define in two possible ways
    - inline
    - $sref (preferred choice as this prevents duplicate and groups definitions together)

Swagger – inline Response Body

- inline

```
"responses": {
  "200": {
    "description": "successful operation",
    "schema": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "eCustomer": {
          "type": "array",
          "items": {
            "additionalProperties": false,
            "properties": {
              "_id": {
                "type": "string",
                "x-semanticType": "Internal"
              },
                .....
```

Swagger – $ref Response Body

- $sref

```
"responses": {
  "200": {
    "description": "successful operation",
    "schema": {
      "$ref": "#\/definitions\/dsCustomer"
    }
  },
```

Swagger – Input & Output Models

- Input and Output Models
  - The global definitions section lets you define common data structures used in your API. They can be referenced via $ref whenever a schema is required — both for request body and response body
  - For example, lets supposed we have the following object:

```
{
  “CustNum": 13,
  “Name": “Freddie Krugger“,
  “Comments”: “Friendly Psychopath …..”
}
```

Swagger – Input & Output Models

- Input and Output Models
  - The above JSON object can then be represented as:

```
"definitions": {
  "dsCustomer": {
   "type": "object",
   "additionalProperties": false,
   "properties": {
    "eCustomer": {
     "type": "array",
     "items": {
      "additionalProperties": false,
      "properties": {
       "CustNum": {
        "type": "integer",
        "x-ablType": "INTEGER",
        "default": 0,
        "x-title": "Cust Num",
        "x-enabledState": "add"
       },
       "Name": {
        "type": "string",
        "x-ablType": "CHARACTER",
```

Swagger – Input & Output Models

- Input and Output Models
  - And is then referenced in the “request” (or even in the “parameters” as seen previously) body schema:

```
"/Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity": {
    "get": {
      ....
      "responses": {
        "200": {
          "description": "successful operation",
          "schema": {
            "$ref": "#/definitions/dsCustomer"
          }

    "delete": {
                ....
      "parameters": [
        {
          "in": "body",
          "name": "dsCustomer",
          "required": true,
          "description": "Record to Delete",
          "schema": {
```

Swagger – Grouping Operations

- Grouping Operations With Tags
  - The OpenAPI specification allows API operations to be grouped together by the use of "tags". For example, Swagger UI uses tags to sort and group the displayed operations
  - “Tags” can be defined globally and referenced within the relevant “Operation”
    - Note: The “tags” attributes should match
  - “Tags” can be defined within each “Operation” (without global “tags”)

Swagger – Grouping “tags”

- Global “tags”

```
"tags": [
  {
   "name": "Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity",
   "description": "Access to Sports2000 Customers",
  },
  {
   "name": "Consultingwerk.SmartComponentsDemo.OERA.Sports2000.ItemBusinessEntity",
   "description": "Access to Sports2000 Items"
  },
  ....
 ],

"\Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity": {
    "get": {
     "tags": [ "Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity“ ],
    },
    "delete": {
     "tags": [ "Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity"]
```

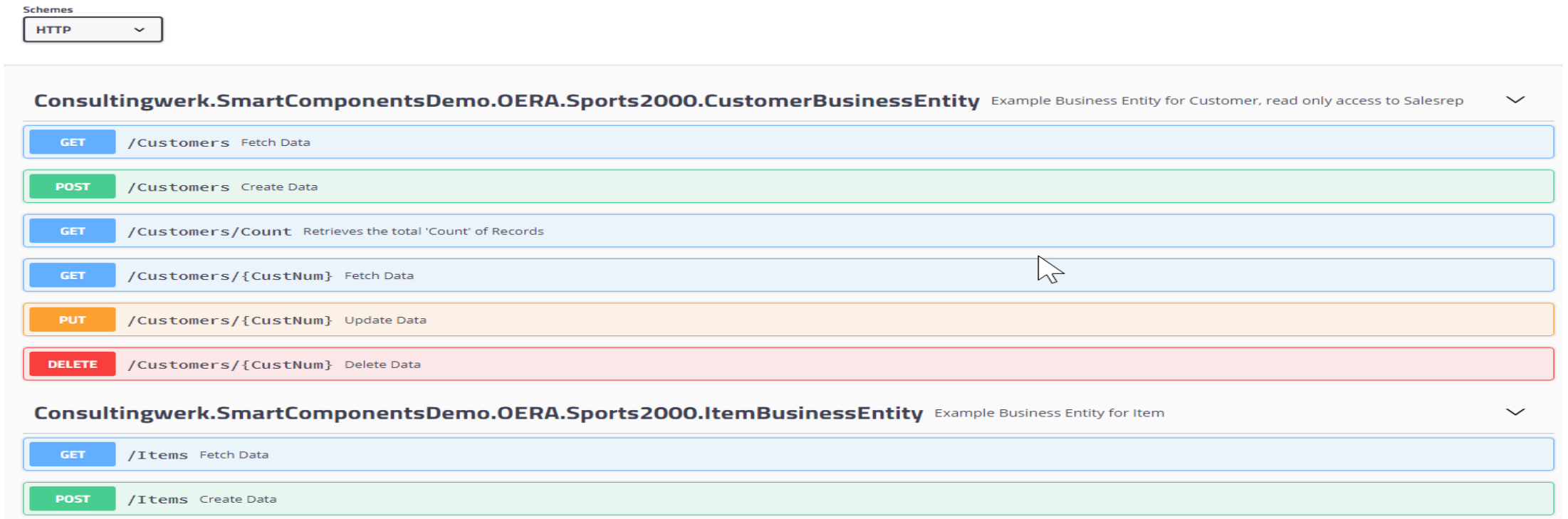
Swagger – Operation “tags”

- Operation “tags”

```
"\Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity": {
    "get": {
     "tags": [
      "Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity"
     ],
     ....
    },
    "delete": {
     "tags": [
      "Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CustomerBusinessEntity"
     ],
```

Swagger – “tags”

- Regardless of which method you invoke, the final display shall always be represented as seen below



- The benefit of using “Global ‘tags’” is that duplication is reduced (e.g. Doc Ref, Descriptions)

Agenda

- OpenAPI (formerly known as Swagger)
- OpenAPI Swagger Structure
- OpenAPI Swagger Online Editor
- Consultingwerk Swagger
- Using Consultingwerk Swagger with RESTful Services

Swagger – Demo

- Demo
  - Demonstrate the Swagger Editor

- http://localhost:8820/web/Consultingwerk/OERA/Swagger/swagger.html?EntityName=Consultingwerk.SmartComponentsDemo.OERA.Sports2000.CountryBusinessEntity

Swagger – Manual or Dynamic Design

- So why not manual design?
  - Difficult to maintain (bad memories….)
  - Version Control
  - Etc ….
- Dynamic
  - No need to worry about changes
  - Bug Fixes and Enhancements are centralised (meaning ALL future OpenAPI definitions automatically see changes)

- We do not hand crank our OpenAPI definitions. Our OpenAPI definitions are generated dynamically through OpenEdge code

Agenda

- OpenAPI (formerly known as Swagger)
- OpenAPI Swagger Structure
- OpenAPI Swagger Online Editor
- Consultingwerk Swagger
- Using Consultingwerk Swagger with RESTful Services

Consultingwerk Swagger

- Consultingwerk now provides OpenAPI (Swagger) Documentation as part of the SmartComponent Library framework
- Consultingwerk provides two Variations of the OpenAPI (Swagger) Documentation
  - Swagger REST API Documentation for JSDO Generic Service
    - https://documentation.consultingwerkcloud.com/display/SCL/Swagger+REST+API+Documention+for+JSDO+Generic+Service
  - Swagger REST API Documentation for RESTful Services
    - Uses the HATEOAS Driven REST API's standard
      - HATEOAS (Hypermedia as the Engine of Application State)
      - Reference: https://restfulapi.net/hateoas/
    - https://documentation.consultingwerkcloud.com/display/SCL/Swagger+REST+API+Documentation+for+RESTful+Services

Consultingwerk Swagger REST API for JSDO Generic Service

- REST API Documentation for JSDO Generic Service
  - The Consultingwerk Swagger Generator produces REST API Swagger Documentation for SmartComponent Library Business Entities, allowing the consumer to understand and interact with the remote service through the Generic Service Interface for the JSDO or Pacific WebSpeed WebHandler based JSDO Generic Service.
  - The Consultingwerk Swagger Generator is accessible dynamically through a Web Handler and can produce static Swagger definitions through a plugin for the Business Entity Designer.
  - The Consultingwerk Swagger Generator is also accessible from within the Business Entity Designer.

Agenda

- OpenAPI (formerly known as Swagger)
- OpenAPI Swagger Structure
- OpenAPI Swagger Online Editor
- Consultingwerk Swagger
- Using Consultingwerk Swagger with RESTful Services

Consultingwerk Swagger REST API for RESTful Services

- REST API Documentation for RESTful Services
  - RESTful services provide an alternative HTTP and JSON based interface for third party applications or mobile applications (the JSDO generic service is specialized to be used by JSDO based clients).
  - The Consultingwerk Swagger RESTful Services Generator produces REST API Swagger Documentation for SmartComponent Library Business Entities exposed as RESTful services, allowing the consumer to understand and interact with the remote service.
  - Based on the HATEOAS standard, this architectural style lets you use hypermedia links in the response contents so that the client can dynamically navigate to the appropriate resource by traversing the hypermedia links.

Consultingwerk Swagger REST API for RESTful Services – Demo

- Swagger REST API Documentation for RESTful Services
  - http://localhost:8820/web/SwaggerEntities/html
  - http://localhost:8820/web/SwaggerEntities/json

Consultingwerk Swagger REST API for RESTful Services – Demo

- Defining a manual Swagger REST API Document for a RESTful Service
  - Use the Swagger Editor to create a Employees Definition
  - Based on Employee's Entity
  - Change the Root/static/swagger.html to point at Employees.json
  - http://localhost:8820/static/swagger/swagger.html

Questions