Using Swagger in TCRS

Objectives

- Give an overview of the purpose and benefits of Swagger
- Use the @Api and @ApiOperation annotations to define a simple REST endpoint in TCRS
- Give an overview of the how the Swagger UI is configured and used, including the potential need for CORS headers to allow the client to access the api-docs URI.
- Use the Swagger-UI facility to invoke REST service endpoints for experimentation and testing
- Give an overview of the use of the Swagger Editor to create Swagger documentation of a design prior to coding

Objectives

- Give an overview of the use of the Swagger Editor to create skeleton JAX-RS service classes that include JAX-RS and Swagger annotations
- Give an overview of the use of the Swagger Editor to create JAX-RS client interface library source code

Documenting REST

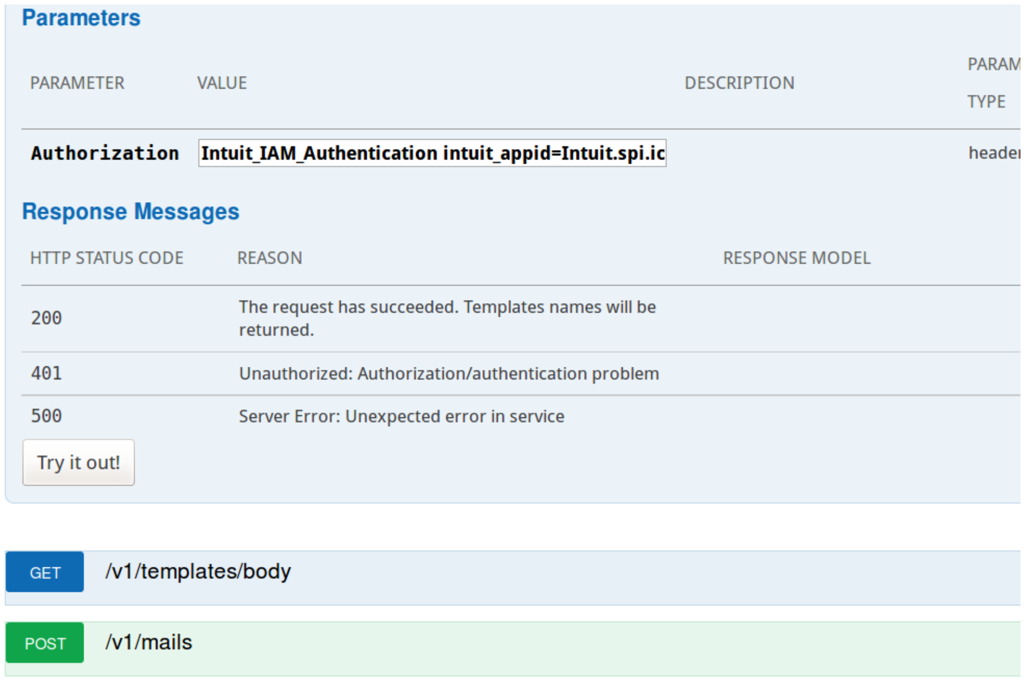
- REST lacks a formal, standardized or widely accepted, interface specification
  - Compare with WSDL for SOAP type web services
  - WADL exists, but is rarely used
  - Fielding's intention was essentially self-documenting (HATEOAS) services, once the data types are known—this is rarely realized
- Swagger is an open source project addressing this, and more

Swagger Features

- Provides both machine and human readable documentation
- Language independent, with libraries for JAX-RS and many other languages used for servers
- JSON machine readable docs can be presented as an interactive page using the “Swagger UI”
  - Supports manual, interactive, experimentation / testing
  - Presents human readable, descriptive, documentation
- In JAX-RS, documentation is derived from both Swagger specific and JAX-RS annotations

Swagger UI Appearance

- Services presented in the Portal should provide a Swagger UI “playground” for experimentation
  - Should be connected to an e2e environment, not the live customer-facing service
  - URIs are listed with
  - parameters &
- methods
- Text describes
- how to use, and
- effects of operations


Parameters

PARAMETER	VALUE	DESCRIPTION	PARAM TYPE
Authorization	Intuit_IAM_Authentication intuit_appid=Intuit.spi.ic		header

Response Messages

HTTP STATUS CODE	REASON	RESPONSE MODEL
200	The request has succeeded. Templates names will be returned.	
401	Unauthorized: Authorization/authentication problem	
500	Server Error: Unexpected error in service	

Annotations

- Key Swagger annotations are:
- @Api—indicates that a class is a root resource and adds descriptive fields
- @ApiOperation—indicates a method is an entry point, and adds descriptive fields
- @ApiResponses—indicates HTTP status responses returned by a request, and gives description
- @ApiModelProperty—indicates allowed values for enumeration-type string values

Generating Annotations

- Annotations can be added manually to source code
  - TCRS “documents” example resource has examples
- Swagger editor can be used to create swagger docs, and template code, from YAML markup using a context sensitive editor
  - editor.swagger.io/

Generating Code

- Using Swagger Editor, YAML document can be used to create skeleton server code, and client library automatically
  - Generation is available for many languages / frameworks
  - Generated code, obviously, does not include business logic
  - Generated server code for JAX-RS includes both JAX-RS and Swagger annotations

Example YAML For Swagger

- First, define the version of Swagger, and provide basic information about the API

```
swagger: '2.0'
info:
  title: Fruits API
  description: Interact with fruits
  version: "1.0.0"

produces:
  - application/json
  - application/xml
```

Example YAML For Swagger

- Then each resource path the service recognizes is listed, with a definition of its behavior

```
paths:
  /fruit:
    get:
      summary: get all fruits
      responses:
        200:
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/Fruit'
```

Example YAML For Swagger

```
paths:         These parts are not repeated for
  /fruit:  the different HTTP methods listed
    post:
      summary: Create a new Fruit
      parameters:
        - in: body
          name: The entity body
          description: A new Fruit
          required: true
          schema:
            $ref: "#/definitions/Fruit"
      responses:
        201:
          description: New fruit created ok
```

Example YAML For Swagger

```
paths:      □ Not repeated ..
  /fruit/{id}:
    get:    □ Summary and description omitted
      parameters:
        - name: id
          in: path
          required: true
          type: number
          format: integer
      responses:
        200:
          description: A piece of fruit
          schema:
           $ref: '#/definitions/Fruit'
```

Example YAML For Swagger

- Items of the form `$ref: '#/definitions/Fruit'` refer to data types in the definitions section

```
definitions:
  Fruit:
    description: Healthy edible thing
    properties:
      name:
        type: string
        description: name of the fruit
```

Swagger YAML Specification

- The YAML used by the Swagger Editor is mapped one-to-one with the JSON format of the Swagger documentation. The full specification of the Swagger JSON format may be found on the specification's github page at:
-
- `https://github.com/swagger-api/`
  `swagger-spec/blob/master/versions/2.0.md`

Example Generated Code

- Generated code includes annotations and placeholder methods for api entry points

```
@Path("/fruit")
@Api(value="/fruit", description="the fruit API")
public class FruitApi {
  @GET
  @ApiOperation(value="get all fruits", notes = "",
    response=Fruit.class, responseContainer="List")
  @ApiResponses(value={
    @ApiResponse(code=200, message="OK") })

  public Response fruitGet() {
      // do some magic!  Your business logic here
```

Configuring Swagger

- Swagger supports two main modes of operation
- Live generation of api-docs
  - Behavior is added to your service responding to GET requests on resources in the tree …/api-docs
  - Intuit preferred approach, avoids risk of version mismatch
- Static generation of api-docs
  - JSON text files for api-docs generated during build phase (e.g. with maven plugin)
  - Inconvenient to combine static document tree with service
  - Some minimal saving of load on the server at runtime

Configuring Swagger

- In general, Swagger 2.x can be installed in a Jersey application as:
  - A servlet, com.wordnik.swagger.jersey.config.JerseyJaxrsConfig declared in web.xml, having no mapping (essentially tied into the servlet lifecycle and taking over from there)
  - A set of provider classes in the package com.wordnik.swagger.jersey.listing
    - ApiListingResourceJSON,
    - JerseyApiDeclarationProvider,
    - JerseyResourceListingProvider

Configuring Swagger

- Swagger configuration is already handled in TCRS
- Swagger configuration is highly version dependent, expect to read the documentation for up to date information
- Swagger also needs to be configured for the base address on which the api-docs are published
- Swagger configuration documentation is under: https://github.com/swagger-api/swagger-core

Swagger, Portal, and CORS

- Your service should be documented on the Portal, including a Swagger playground
- If your Service is on host X, the Portal is on host Y, and someone wants to use the playground on host Z, then the Swagger UI code is loaded onto Z from Y (the portal), but then requests are made to X from Z, using code from Y
  - This is a ***Cross Origin Request***, and the browser will block it by default
- Swagger's …/api-docs URIs should probably be configured to permit cross origin requests

Lab Exercise

- Use the Swagger Editor to create a definition of a simple service that responds to URIs under /fruits as follows
  - GET /fruits/<id> — returns a JSON structure representing a fruit. The structure has name and color as String properties
  - GET /fruits/<id>/name — returns the name of the fruit
- Generate the Java / JAX-RS service code, download it and examine it—what would you do to move this into TCRS?