Versione 01

VERIFICHE E APPROVAZIONI

VERS	REDAZIONE		CONTROLLO APPROVAZIONE		AUTORIZZAZIONE EMISSIONE	
	NOME	DATA	NOME	DATA	NOME	DATA
V01	Mauro Antonaci	15/07/2018	Mauro Antonaci	15/09/2018	Fabrizio Barbero	15/09/2018

STATO DELLE VARIAZIONI

VERS	PARAGRAFO O PAGINA	DESCRIZIONE DELLA VARIAZIONE
V01	Tutto il documento.	Versione iniziale del documento.

Sommario

1. Scopo del documento

Questo documento ha lo scopo di illustrare alcune linee guida, e gli strumenti a disposizione, per la realizzazione di servizi *REST* in modalità *contract-first*, ovvero a partire da un artefatto di definizione delle specifiche delle API.

2. Riferimenti

- [§1] OpenApi specification v.2.0: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md
- [§2] Progetto Swagger-codegen su gitlab:§ https://github.com/swagger-api/swagger-codegen
-

3. Campo di applicazione

Le presenti linee guida sono applicabili in tutti gli scenari di realizzazione di API utilizzando lo stack applicativo basato su *jaxrs-resteasy*.

4. Destinatari ed utilizzo del documento

Questo documento è destinato ai progettisti/sviluppatori che, avendo la necessità di realizzare servizi *REST* (siano essi *stand-alone* o *backend* di una *Single Page Application*), decidano di utilizzare la tecnologia *JAX-RS*, ed in particolare il framework *resteasy*.

A corredo del documento vengono forniti:

- Un *tool* di generazione degli artefatti *jaxrs*, basato su *swagger*.

Il modo più efficace di utilizzare queste linee guida pertanto è il seguente:

- Prendere conoscenza delle specifiche *JAX-RS* e del framework *resteasy* in autonomia, consultando la documentazione relativa alle specifiche e al framework;
- Prendere conoscenza delle specifiche *OpenAPI* per la formalizzazione delle API *REST*
- Leggere queste linee guida ed esaminare il progetto di esempio allegato alle linee guida di *JAX-RS*
- Provare ad implementare una propria applicazione, partendo da uno scheletro di progetto che può essere derivato dal progetto di esempio di cui sopra, dal quale si tolgono/modificano manualmente le parti applicative

5. Requisiti/Vincoli su cui si basano le linee guida

Le presenti linee guida sono ispirate dalla presenza di requisiti o vincoli specifici relativi al contesto di applicazione, ovvero:

- i sistemi informativi degli enti della PA piemontese
- le caratteristiche degli ambienti di esecuzione della server farm CSI

- i processi di sviluppo e rilascio della software factory CSI
- altre linee guida in vigore in CSI

In questo capitolo sono elencati tali requisiti/vincoli e nei successivi capitoli si farà riferimento ad essi per giustificare alcune delle scelte fatte. Non è pertanto indispensabile conoscere approfonditamente questo capitolo ai fini dell'applicazione delle linee guida, ma se ne consiglia ugualmente la lettura.

Esistono requisiti/vincoli di varia forza:

- alcuni vincoli sono imprescindibili in quanto il loro mancato rispetto potrebbe rendere non possibile, ad esempio, il dispiegamento dell'applicativo, oppure la corretta gestione del prodotto in esercizio, oppure la gestione del progetto da un punto di vista di processo
- alcuni vincoli sono da considerarsi meno tassativi, in quanto il loro mancato rispetto non comprometterebbe gli aspetti di cui sopra mentre invece il loro rispetto potrebbe garantire, ad esempio, una maggiore efficienza del processo di sviluppo, migliori performance, etc..

Pertanto per ciascuno dei vincoli verrà anche indicato il livello di *importanza*, oltre che l'aspetto che il rispetto di tale vincolo influenza.

Nei paragrafi seguenti sono elencati i vari requisiti/vincoli e si specifica anche il grado di copertura del requisito fornito da queste linee guida, che potrà essere:

- requisito totalmente soddisfatto
- requisito non soddisfatto o non soddisfacibile (può essere solo per quei requisiti che non siano dichiarati indispensabili)
- requisito non ancora preso in considerazione (sarà preso in considerazione in future versioni di linea guida o in linee guida accessorie)

5.1 Generazione completa degli artefatti jaxrs di definizione interfaccia

Deve essere possibile generare completamente almeno i seguenti artefatti jax-rs:

- Lo strato model (i dto)
- Le resource classes

- Importanza: obbligatoria
- Elementi influenzati: sorgenti

Questo requisito risulta totalmente soddisfatto grazie all'implementazione di generatore fornita.

5.2 Rispetto delle linee guida di naming delle classi

Le classi generate a partire dalla specifica delle API devono rispettare le indicazioni relative a:

- Naming dei package
- Naming delle classi

- Importanza: obbligatoria
- Elementi influenzati : sorgenti

Questo requisito risulta totalmente soddisfatto grazie all'implementazione fornita.

5.3 Rispetto delle best practices di naming di metodi/attributi/proprietà

Le classi generate a partire dalla specifica delle API devono rispettare le linee guida di naming degli artefatti del linguaggio target (es. le linee guida java)

- Importanza: obbligatoria
- Elementi influenzati : sorgenti

Questo requisito risulta totalmente soddisfatto grazie all'implementazione di generatore fornita.

5.4 Possibilità di round-trip-engineering

Affinché un metodo di sviluppo API *contract-first* risulti efficace, è necessario poter mettere in pratica un meccanismo di *round-trip-enginnering* tale per cui ad ogni modifica del modello delle API sia possibile rigenerare le classi che costituiscono l'interfaccia in modo da rispecchiare le variazioni apportate.

- Importanza: obbligatoria
- Elementi influenzati : sorgenti

Questo requisito risulta parzialmente soddisfatto grazie all'implementazione di generatore fornita. In particolare risulta totalmente soddisfatto per quanto riguarda le classi di interfaccia mentre le classi di implementazione delle interfacce *jax-rs* devono essere adeguate manualmente a fronte della variazione delle classi di interfaccia, ma questo non rappresenta un particolare problema.

6. Generatore *jaxrs-resteasy-eap-csi*

In questo capitolo viene descritto il plugin di *swagger* realizzato in CSI per generare artefatti *jaxrs-resteasy* secondo le linee guida CSI. Si rammenta che questo strumento rappresenta una estensione custom del generatore standard *swagger*: la maggior parte delle opzioni di lancio è descritta anche nella pagina del progetto GIT corrispondente (#2).
Allo stato attuale, però se ne consiglia l'utilizzo per la generazione delle sole classi di interfaccia, seguendo le indicazioni fornite in questo documento.
Questo documento fa riferimento alla versione 1.0.0.001 del generatore.

6.1 Installazione e lancio del generatore

Il generatore è distribuito tramite due jar:

- Il primo jar (swagger-codegen-cli.jar) contiene le classi core del generatore swagger
- Il secondo jar (csi-java-swagger-codegen-<versione>.jar) contiene l'estensione per il "linguaggio" jaxrs-resteasy-csi

L'installazione consiste nel copiare in una cartella a piacere tali jar.

Nell'esempio che segue si presuppone che il generatore sia lanciato da un *prompt* di comando posizionato nella cartella "*buildfiles*" del progetto, nella quale è consigliabile di inserire un file .bat contenente i comandi per lanciare il generatore.
Per lanciare il generatore da *windows* occorre eseguire, ad esempio, i seguenti comandi:

```
REM La directory dove sono presenti i jar del generatore csi-swagger
set CSI_SWAGGER_GEN_HOME=c:\devtools\csi-swagger-codegen

set CLI_JAR=%CSI_SWAGGER_GEN_HOME%\swagger-codegen-cli.jar
```

```
set CUSTOM_GEN_JAR=%CSI_SWAGGER_GEN_HOME%\csi-java-swagger-codegen-1.0.0.jar

set SWAGGER_CP=%CLI_JAR%;%CUSTOM_GEN_JAR%

java -cp %SWAGGER_CP% io.swagger.codegen.SwaggerCodegen generate -i ..\src\yaml\myapi.yaml -l jaxrs-resteasy-eap-csi -o .. --config swagger_config_java.json
```

Note:

- La variabile CSI_SWAGGER_GEN_HOME deve puntare alla cartella che contiene i *jar* del generatore
- Il file referenziato nella opzione "-i" rappresenta la specifica OpenApi/swagger della API di cui si vuole generare le classi
- L'opzione "-l" permette di impostare il "linguaggio" in cui generare: nel caso specifico il valore "jaxrs-resteasy-eap-csi" permette di utilizzare il generatore custom per servizi *resteasy* che segue le specifiche CSI.
- Il file json referenziato dalla opzione "--config", il cui contenuto è descritto di seguito, permette di modificare parzialmente il comportamento del generatore

6.1.1 Il file di opzioni JSON

Le opzioni con cui è possibile modificare il comportamento del generatore per un determinato "linguaggio", sono ottenibili tramite il comando:

```
java -cp %SWAGGER_CP% io.swagger.codegen.SwaggerCodegen config-help -l jaxrs-resteasy-eap-csi
```

Di seguito un esempio di file *json*, di cui si commentano alcune *property*.

```
{
"java8": false,
"useBeanValidation": true,
"hideGenerationTimestamp": true,
"apiPackage": "it.csi.test.tstjaxrs.business.anagrafica",
"modelPackage":  "it.csi.test.tstjaxrs.dto.anagrafica",
"invokerPackage": "it.csi.test.tstjaxrs.business.anagrafica",
"implFolder": "implfolder",
"sourceFolder": "src/java",
"useSwaggerFeature": false,
"generateJbossDeploymentDescriptor": false,
"generateImplArtifacts": false,
"generateSwaggerMetadata": false,
"serializableModel": true,
"attributeSerializationStrategy": "implicit-snake-case",
"accessHttpHeaders": true,
"accessHttpRequest": false
}
```

- **apiPackage**: se impostato permette di personalizzare il package in cui vengono generate le classi relative alle API/risorse.
- **modelPackage**: se impostato permette di personalizzare il package in cui vengono generate le classi relative ai DTO referenziati dalle API.
- **sourceFolder**: se impostato permette di personalizzare il percorso, al di sotto della cartella impostata nella opzione "-o" del comando, in cui verranno create le classi java. L'impostazione "src/java", ad esempio, permette di far generare le classi secondo le linee guida CSI vigenti
- **attributeSerializationStrategy**: permette di configurare il modo in cui sono serializzate le proprietà:
  - "explicit-as-modeled": le proprietà vengono serializzate esattamente come definite nello yaml, grazie all'utilizzo della *annotation* della libreria *jackson* (

@JSONProperty("nome_campo")). E' l'impostazione che permette la maggior flessibilità ma introduce anche un lock-in importante del progetto nei confronti della libreria *jackson*
    - "implicit-camel-case": le proprietà vengono serializzate con la strategia di default per jaxrs, ovvero con i nomi degli attributi in *camelCase*.
    - "implicit-snake-case": le proprietà vengono serializzate automaticamente in *camel_case* (es. la property "codiceFiscale" sarà automaticamente mappata con "codice_fiscale"). Lo stesso risultato sarebbe realizzabile anche con una strategia "explicit-as-modeled" e impostando i nomi delle property in camel case direttamente nello yaml, ma l'impostazione implicita permette di non annotare le singole properties dei DTO con l'annotazione @JSONProperty("nome_campo"), minimizzando il lock-in del progetto con la libreria jackson. Affinchè tale impostazione abbia effetto è però necessario inserire esplicitamente nel file JacksonConfig.java la seguente impostazione:
      `this.objectMapper.setPropertyNamingStrategy(PropertyNamingStrategy.CAMEL_CASE_TO_LOWER_CASE_WITH_UNDERSCORES); )`
- **accessHttpHeaders**: se impostato a "true" permette di aggiungere all'elenco di parametri delle classi associate alle API un parametro di tipo `javax.ws.rs.core.HttpHeaders`, che potrà essere utilizzato nella *business logic* per accedere agli attributi dell'*header http* della richiesta
- **accessHttpRequest**: se impostato a "true" permette di aggiungere all'elenco di parametri delle classi associate alle API un parametro di tipo `javax.servlet.http.HttpServletRequest`, che potrà essere utilizzato nella *business logic* per accedere alla richiesta *http* nella sua interezza.

N.B: allo stato attuale è consigliabile utilizzare il generatore esclusivamente per la creazione delle classi di interfaccia, ovvero:

- API (classi annotate jax-rs)
- Model

A tale scopo si consiglia di impostare le file *json* di configurazione le seguenti proprietà:

- `"useSwaggerFeature": false,`
- `"generateJbossDeploymentDescriptor": false,`
- `"generateImplArtifacts": false,`
- `"generateSwaggerMetadata": false,`

6.2 Output del generatore

Il generatore genera una serie di classi a partire dal contenuto della specifica *OpenAPI / yaml*. In sintesi il generatore genera:

- Per ogni oggetto modellato nelle *definitions* nello *yaml* ➔ una classe che rappresenta un DTO (classi "*model*")
- Per ogni *path* definito nella sezione "paths":
    - una *interface* che rappresenta un sottoinsieme delle API, secondo le specifiche *jaxrs*.
    - Una *ResourceClass jaxrs*, che implementa l'*interface* del punto precedente, nella quale deve essere inserita la logica di implementazione delle varie API e che devono essere registrate nella *RestApplication*.

6.2.1 Le classi model

Sono DTO direttamente derivanti dalle *definitions* contenute nel file yaml.
Possono essere sovrascritte tranquillamente ad ogni rigenerazione, in quanto non contengono codice inserito a mano.

6.2.1.1 Strategia di serializzazione degli attributi

Resteasy utilizza la libreria jackson per la serializzazione in json. Nel caso in cui si debba utilizzare una strategia di serializzazione degli attributi differente da quella di default (che prevede la serializzazione in "camelCase") è possibile aggiungere sulle *properties* una *annotation* @JsonProperty, indicando il nome della *property JSON* da utilizzare per quella proprietà java.

Es:

```
@JsonProperty("data_nascita")
public Date getDataNascita() {
  return dataNascita;
}
public void setDataNascita(Date dataNascita) {
  this.dataNascita = dataNascita;
}
```

Questa *annotation* è specifica della libreria *jackson*. Utilizzando esplicitamente tale *annotation* si crea un *lock-in* non desiderabile e diffuso con una libreria aggiuntiva rispetto alla pura specifica *jaxrs*.
Nel caso sia necessario utilizzare la serializzazione basata, ad esempio, su una strategia "*snake_case*" è possibile, al fine di limitare tale *lock-in*, impostarla come meccanismo di default, impostando l'opzione *attributeSerializationStrategy* al valore *"implicit-snake-case"*.
In questo modo non verrà generata *l'annotation* @JsonProperty("<nome_property>").
Affinchè la serializzazione avvenga di default utilizzando lo *snake_case* per le *property* dei *dto* è però necessario aggiungere nella classe *JacksonConfig* la seguente impostazione:

```
this.objectMapper.setPropertyNamingStrategy(
      PropertyNamingStrategy.CAMEL_CASE_TO_LOWER_CASE_WITH_UNDERSCORES);
```

Questa configurazione rappresenta una soluzione sempre specifica della libreria Jackson, ma meno pervasiva di quella basata su *annotation*, in quanto limitata ad una impostazione globale e non diffusa in tutte le classi DTO.
Nel caso invece in cui si desideri utilizzare la serializzazione di default di *resteasy*, ovvero quella in cui gli attributi vengono serializzati in *camelCase*, è possibile impostare l'opzione *attributeSerializationStrategy* al valore *"implicit-camel-case"*. Anche in questo caso non vengono aggiunte le annotation @JsonProperty e non è necessario aggiungere alcuna impostazione nel file JacksonConfig.

6.2.1.2 Validazione

A fronte della presenza nello yaml di constraint di validazione su vari elementi dell'interfaccia il generatore inserisce nelle classi corisspondenti alcune annotazioni che permettono di implementare a runtime tali constraint.
In particolare i punti interessati sono:

- I campi dei DTO
- I parametri di input dei metodi corrispondenti ai vari path

6.2.2 Le resource classes (interface + implementation)

Il generatore, per ogni path distinto presente nella definizione OpenAPI, genera:

- Una interface che rappresenta la (una possibile) trasposizione jax-rs delle API definite, e che può essere sovrascritta a piacere, a fronte di rigenerazione, in quanto non se ne prevede una modifica manuale
- Una classe di implementazione di tale interfaccia che invece deve essere generata solo una volta poiché contiene l'implementazione specifica delle varie API (o il codice che delega tale logica ai bean di spring retrostanti)

Per poter utilizzare tale classe di implementazione secondo le linee guida *CSI jax-rs*, è necessario:

1. Implementare le indicazioni riportate nelle linee guida *CSI jax-rs* relativamente alla integrazione con *spring* (paragrafo 7.2.7.2 delle linee guida in versione V3)
2. Registrare tale classe tra i *singleton* della *RestApplication*, (come descritto al paragrafo 7.2.5 delle linee guida in versione V3)