Spike: MODKBEKBJ-46 - Spring initiative investigation

~~MODKBEKBJ-46~~ - SPIKE: PoC - integrate springmvc-raml-plugin into the sample spring application CLOSED

Goal:

The main goal of the spike was investigation if the proposed plugin meets project initial requirements.

Initial requirements:

1. for endpoints it has to generate interfaces or classes that could be extended to implement the necessary logic **[MUST]**;
2. support !include directive in RAML files like below:

```
types:
  note: !include note.json
  noteCollection: !include noteCollection.json
  errors: !include raml-util/schemas/errors.schema
```

3. support definition of a single element in collection as $ref

Investigation results:

Initial requirement:

> 1. *for endpoints it has to generate interfaces or classes that could be extended to implement the necessary logic [MUST];*

Result:

The plugin provides several rules an ability to generate interfaces. To achiev this you have to specify the rule name in pom.xml file

```
<configuration>
  <rule>com.phoenixnap.oss.ramlplugin.raml2code.rules.Spring4ControllerInterfaceRule</rule>
</configuration>
```

Here is the raml file which were used to test:

```
#%RAML 1.0
title: Drinks Services
version: ${project.version}
mediaType: application/json
baseUri: /

types:
  drink:
    type: object
    properties:
      type:
        type: string
        description: The drink type.
      name:
        type: string
        description: The drink name.
  drinkCollection:
    type: object
    properties:
      drinks:
        type: array
        items: drink
  drinkUpload:
```

```
    type: object
    properties:
      name:
        type : string
  healthCheck:
    type: object
    properties:
      timestamp:
        type: string
      status:
        type: string

/drinks:
  description: Provides interaction with 'Drink' resource
  get:
    description: Retrieves a list of available drinks.
    responses:
      200:
        body:
          application/json:
            type: drinkCollection
      400:
        description: |
          The request sent to the server contains data which is not as expected e.g. incorrect drink type
      404:
        description: |
          The requested resource was not found
  post:
    description: Creates a new drink.
    body:
      application/json:
        type: !include examples/postDrink.json
    responses:
      201:
        body:
          application/json:
            type: drink

  /{drinkName}:
    uriParameters:
      drinkName:
        displayName: DrinkName
        description: |
          The name of the drink for info
          required: true
          type: string
    get:
      description: Retrieves details for a specific drink.
      responses:
        200:
          body:
            application/json:
              type: drink
        404:
          description: "Drink Name not found"
          body:
            text/plain:
              example: "Drink Name not found"
        400:
          description: "Internal server error"
          body:
            text/plain:
              example: "Internal server error"
    put:
      description: Modify an existing drink.
      body:
        application/json:
          type: drinkUpload
      responses:
        200:
          body:
```

```
            application/json:
              type: drinkUpload
        400:
          description: "Bad request"
          body:
            text/plain:
              example: "Bad request"
        404:
          description: "User not found"
          body:
            text/plain:
              example: "User not found"
        500:
          description: "Internal server error"
          body:
            text/plain:
              example: "Internal server error"
    delete:
      description: Delete an existing drink.
      responses:
        204:
          description: "No Content"
          body:
            text/plain:
             example: "No Content"
        404:
          description: "Not Found"
          body:
            text/plain:
              example: "Not found"
/healthCheck:
  description: Provides server health check
  get:
    description: Retrieves the server's health status.
    responses:
      200:
        description: The health check item has been successfully retrieved.
        body:
          application/json:
            type: healthCheck
      400:
        description: "Bad request"
        body:
          text/plain:
            example: "Bad request"
      404:
        description: "Not found"
        body:
          text/plain:
            example: "Not found"
```

this are the Controllers generated by plugin:

DrinkController

```
/**
 * Provides interaction with 'Drink' resource
 * (Generated with springmvc-raml-parser v.2.0.4)
 *
 */
@RestController
@Validated
@RequestMapping(value = "/drinks", produces = "application/json")
public interface DrinkController {


    /**
     * Retrieves a list of available drinks.
     *
     */
    @RequestMapping(value = "", method = RequestMethod.GET)
    public ResponseEntity<DrinkCollection> getDrinks();

    /**
     * Creates a new drink.
     *
     */
    @RequestMapping(value = "", method = RequestMethod.POST)
    public ResponseEntity<Drink> createDrink(
        @javax.validation.Valid
        @org.springframework.web.bind.annotation.RequestBody
        CreateDrinkRequest createDrinkRequest);

    /**
     * Retrieves details for a specific drink.
     *
     */
    @RequestMapping(value = "/{drinkName}", method = RequestMethod.GET)
    public ResponseEntity<Drink> getDrinkByName(
        @PathVariable
        String drinkName);

    /**
     * Modify an existing drink.
     *
     */
    @RequestMapping(value = "/{drinkName}", method = RequestMethod.PUT)
    public ResponseEntity<DrinkUpload> updateDrinkByName(
        @PathVariable
        String drinkName,
        @javax.validation.Valid
        @org.springframework.web.bind.annotation.RequestBody
        DrinkUpload drinkUpload);

    /**
     * Delete an existing drink.
     *
     */
    @RequestMapping(value = "/{drinkName}", method = RequestMethod.DELETE)
    public ResponseEntity<?> deleteDrinkByName(
        @PathVariable
        String drinkName);

}
```

HealthCheckController

```
**
 * Provides server health check
 * (Generated with springmvc-raml-parser v.2.0.4)
 *
 */
@RestController
@Validated
@RequestMapping(value = "/healthCheck", produces = "application/json")
public interface HealthCheckController {


    /**
     * Retrieves the server's health status.
     *
     */
    @RequestMapping(value = "", method = RequestMethod.GET)
    public ResponseEntity<HealthCheck> getHealthCheck();

}
```

and Models

Drink

```
public class Drink implements Serializable
{

    final static long serialVersionUID = 7968831422593218332L;
    /**
     * The drink type.
     *
     */
    protected String type;
    /**
     * The drink name.
     *
     */
    protected String name;

    /**
     * Creates a new Drink.
     *
     */
    public Drink() {
        super();
    }

    /**
     * Creates a new Drink.
     *
     */
    public Drink(String type, String name) {
        super();
        this.type = type;
        this.name = name;
    }

    /**
     * Returns the type.
     *
     * @return
     *     type
     */
    @NotNull
```

```
    public String getType() {
        return type;
    }

    /**
     * Set the type.
     *
     * @param type
     *     the new type
     */
    public void setType(String type) {
        this.type = type;
    }

    /**
     * Returns the name.
     *
     * @return
     *     name
     */
    @NotNull
    public String getName() {
        return name;
    }

    /**
     * Set the name.
     *
     * @param name
     *     the new name
     */
    public void setName(String name) {
        this.name = name;
    }

    public int hashCode() {
        return new HashCodeBuilder().append(type).append(name).toHashCode();
    }

    public boolean equals(Object other) {
        if (other == null) {
            return false;
        }
        if (other == this) {
            return true;
        }
        if (this.getClass()!= other.getClass()) {
            return false;
        }
        Drink otherObject = ((Drink) other);
        return new EqualsBuilder().append(type, otherObject.type).append(name, otherObject.name).
isEquals();
    }

    public String toString() {
        return new ToStringBuilder(this).append("type", type).append("name", name).toString();
    }

}
```

HealthCheck

```
public class HealthCheck implements Serializable
{

    final static long serialVersionUID = 8277845917257136452L;
    protected String timestamp;
```

```
protected String status;

/**
 * Creates a new HealthCheck.
 *
 */
public HealthCheck() {
    super();
}

/**
 * Creates a new HealthCheck.
 *
 */
public HealthCheck(String timestamp, String status) {
    super();
    this.timestamp = timestamp;
    this.status = status;
}

/**
 * Returns the timestamp.
 *
 * @return
 *     timestamp
 */
@NotNull
public String getTimestamp() {
    return timestamp;
}

/**
 * Set the timestamp.
 *
 * @param timestamp
 *     the new timestamp
 */
public void setTimestamp(String timestamp) {
    this.timestamp = timestamp;
}

/**
 * Returns the status.
 *
 * @return
 *     status
 */
@NotNull
public String getStatus() {
    return status;
}

/**
 * Set the status.
 *
 * @param status
 *     the new status
 */
public void setStatus(String status) {
    this.status = status;
}

public int hashCode() {
    return new HashCodeBuilder().append(timestamp).append(status).toHashCode();
}

public boolean equals(Object other) {
    if (other == null) {
        return false;
    }
    if (other == this) {
```

```
            return true;
        }
        if (this.getClass()!= other.getClass()) {
            return false;
        }
        HealthCheck otherObject = ((HealthCheck) other);
        return new EqualsBuilder().append(timestamp, otherObject.timestamp).append(status, otherObject.
status).isEquals();
    }

    public String toString() {
        return new ToStringBuilder(this).append("timestamp", timestamp).append("status", status).
toString();
    }

}
```

Summary

Pros: ability to create Controllers and Models from the raml file

Cons: model properties do not have Jackson annotations like

```
@JsonProperty("customerId")
@JsonPropertyDescription("Customer ID using the KB")
@NotNull
private String customerId;
```

TODO:

- [ ] investigate annotations creation/adding to the model properties
- [ ] propose the solution (possibly POC) to have needed annotations

- Rules description can be found here https://github.com/phoenixnap/springmvc-raml-plugin#rule
- Sample module can be found here. The pom file which uses this rule can be found here

Initial requirement:

> *2. support !include directive in RAML files like below :*
>
> ```
> types:
>   note: !include note.json
>   noteCollection: !include noteCollection.json
>   errors: !include raml-util/schemas/errors.schema
> ```
>
> *3. support definition of a single element in collection as $ref*

Result:

Plugin is able to read `!include` directive with `json` and `schema` extension but the result of the parsing does not feet our expectations.

For instance we declared type in following `raml` file

```
#%RAML 1.0
title: Drinks Services
version: ${project.version}
mediaType: application/json
baseUri: /

types:
  drinkCollection: !include drinkCollection.json
  drink: !include drink.json
```

```
/drinks:
  description: Provides interaction with 'Drink' resource
  get:
    description: Retrieves a list of available drinks.
    responses:
      200:
        body:
          application/json:
            type: drinkCollection
      400:
        description: |
          The request sent to the server contains data which is not as expected e.g. incorrect drink type
      404:
        description: |
          The requested resource was not found
  post:
    description: Creates a new drink.
    body:
      application/json:
        type: !include examples/postDrink.json
    responses:
      201:
        body:
          application/json:
            type: drink

  /{drinkName}:
    uriParameters:
      drinkName:
        displayName: DrinkName
        description: |
          The name of the drink for info
          required: true
          type: string
    get:
      description: Retrieves details for a specific drink.
      responses:
        200:
          body:
            application/json:
              type: drink
        404:
          description: "Drink Name not found"
          body:
            text/plain:
              example: "Drink Name not found"
        400:
          description: "Internal server error"
          body:
            text/plain:
              example: "Internal server error"
    delete:
      description: Delete an existing drink.
      responses:
        204:
          description: "No Content"
          body:
            text/plain:
             example: "No Content"
        404:
          description: "Not Found"
          body:
            text/plain:
              example: "Not found"
```

with next content inside

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Drinks Collection",
  "type": "object",
  "properties": {
    "drinks": {
      "type": "array",
      "id": "drinksListObject",
      "items": {
        "type": "object",
        "$ref": "drink.json"
      },
      "required": [
        "type",
        "name"
      ]
    }
  },
  "totalRecords": {
    "type": "integer"
  }
,
  "required": [
    "items",
    "totalRecords"
  ]
}
```

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Drink Schema",
  "type": "object",
  "properties": {
    "type": {
      "type": "string"
    },
    "name": {
      "type": "string"
    }
  },
  "required": [
    "type",
    "name"
  ]
}
```

The module will create Controller for this `raml` file with one "but"

```
/**
 * Provides interaction with 'Drink' resource
 * (Generated with springmvc-raml-parser v.2.0.4)
 *
 */
@RestController
@Validated
@RequestMapping(value = "/drinks", produces = "application/json")
public interface DrinkController {


    /**
     * Retrieves a list of available drinks.
     *
     */
    @RequestMapping(value = "", method = RequestMethod.GET)
    public ResponseEntity<?> getDrinks();

    /**
     * Creates a new drink.
     *
     */
    @RequestMapping(value = "", method = RequestMethod.POST)
    public ResponseEntity<?> createDrink(
        @Valid
        @RequestBody
        CreateDrinkRequest createDrinkRequest);

    /**
     * Retrieves details for a specific drink.
     *
     */
    @RequestMapping(value = "/{drinkName}", method = RequestMethod.GET)
    public ResponseEntity<?> getDrinkByName(
        @PathVariable
        String drinkName);

    /**
     * Delete an existing drink.
     *
     */
    @RequestMapping(value = "/{drinkName}", method = RequestMethod.DELETE)
    public ResponseEntity<?> deleteDrinkByName(
        @PathVariable
        String drinkName);

}
```

As we can see the the particular type of the response was not defined and because of it models for "Drink" and "DrinkCollection" also were not created. The only model was created is general request, which fields do not satisfy our requirement.

```
@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonPropertyOrder({
    "type",
    "name"
})
public class CreateDrinkRequest {

    @JsonProperty("type")
    private Object type;
    @JsonProperty("name")
    private String name;
    @JsonIgnore
    @Valid
    private Map<String, Object> additionalProperties = new HashMap<String, Object>();

    @JsonProperty("type")
    public Object getType() {
        return type;
    }

    @JsonProperty("type")
    public void setType(Object type) {
        this.type = type;
    }

    @JsonProperty("name")
    public String getName() {
        return name;
    }

    @JsonProperty("name")
    public void setName(String name) {
        this.name = name;
    }

    @JsonAnyGetter
    public Map<String, Object> getAdditionalProperties() {
        return this.additionalProperties;
    }

    @JsonAnySetter
    public void setAdditionalProperty(String name, Object value) {
        this.additionalProperties.put(name, value);
    }

    @Override
    public String toString() {
        return new ToStringBuilder(this).append("type", type).append("name", name).append
("additionalProperties", additionalProperties).toString();
    }

    @Override
    public int hashCode() {
        return new HashCodeBuilder().append(name).append(additionalProperties).append(type).toHashCode();
    }

    @Override
    public boolean equals(Object other) {
        if (other == this) {
            return true;
        }
        if ((other instanceof CreateDrinkRequest) == false) {
            return false;
        }
        CreateDrinkRequest rhs = ((CreateDrinkRequest) other);
        return new EqualsBuilder().append(name, rhs.name).append(additionalProperties, rhs.
additionalProperties).append(type, rhs.type).isEquals();
    }

}
```

Despite of the fact that currently correct type selection is not provided we can probably inject custom logic in the phase of distinguishing the type in Api ActionMetadata.parseResponse().

part of ApiActionMetadata class

```
package com.phoenixnap.oss.ramlplugin.raml2code.data;
...


private void parseResponse(JCodeModel codeModel, String responseContentTypeFilter) {
   RamlResponse response = RamlHelper.getSuccessfulResponse(action);

   if (response != null && response.getBody() != null && !response.getBody().isEmpty()) {
      for (Entry<String, RamlMimeType> body : response.getBody().entrySet()) {
         if (responseContentTypeFilter == null || body.getKey().equals(responseContentTypeFilter)) {
            if (body.getKey().toLowerCase().contains("json") || body.getKey().toLowerCase().equals("body")) {
               // if we have a json type we need to return an object
               // Continue here!
               ApiBodyMetadata responseBody = null;

               RamlDataType type = body.getValue().getType();
               String schema = body.getValue().getSchema();
               // prefer type if we have it.
               String name = StringUtils.capitalize(getName()) + "Response";
               if (type != null && type.getType() != null && !(type.getType() instanceof
JSONTypeDeclaration)) {
                  responseBody = RamlTypeHelper.mapTypeToPojo(codeModel, parent.getDocument(), type.
getType());
               } else if (StringUtils.hasText(schema)) {
                  responseBody = SchemaHelper.mapSchemaToPojo(parent.getDocument(), schema, Config.
getPojoPackage(), name, null);
               }

               if (responseBody != null) {
                  this.responseBody.put(body.getKey(), responseBody);
               }
            }
         }
      }
   }
}
```

the following declaration

```
types:
  drinkCollection: !include drinkCollection.json
```

is already resolved as `JSONTypeDeclaration` , so the logic inside `SchemaHelper.mapSchemaToPojo(...)` can be modified to satisfy our needs.

Summary

Pros: the initials structure is created

Cons: need to customize the the logic

TODO:

- [ ] dive deeper into process of parsing JSONTypeDeclaration
- [ ] provide a possible solution(POC probably) to include types declaration and reference for single collection element

---

Useful links:

- Rules description can be found here https://github.com/phoenixnap/springmvc-raml-plugin#rule

- Sample module can be found here. The pom file which uses this rule can be found here