WAY2LEARN

LAB- Designing API using RAML

Use Case :

Show the full list of artists.
Show the list of artists by nationality.
Show the full list of albums.
Show the list of albums by genre.
Show a particular artist's albums collection.
Search for a song by title.
Show a particular album's songs list.
Enter new Artists, Albums and Songs

1) Create A new RAML file with Title JukeBox

2)Give the base URI as http://localhost:8081/api and version as v1

3) We want to expose songs ,artists and albums resources such that user will be able to get all resources and also create a new Resource.

So, Modify your RAML as shown below : (Dont Copy Paste. Use CTRL+SPACE and design it as shown below)

```
/songs:
  get:
  post:

/artists:
  get:
  post:

/albums:
  get:
  post:
```

4) We want an ability to get a song by songid, artists by artist id and album by album id.

So, modify the RAML As shown below :

```
/songs:
  get:
  post:
  /{songid}:
    get:

/artists:
  get:
  post:
  /{artistId}:
    get:

/albums:
  get:
  post:
  /{albumId}:
    get:
```

5) We want all albums of a particular artist identified by artistID and all songs in an album identified by albumId.

So, Modify /artists and /albums to look like below :

```
/artists:
  get:
  post:
  /{artistId}:
    get:
    /albums:
      get:

/albums:
  get:
  post:
  /{albumId}:
    get:
    /songs:
      get:
```

5) For get /songs, we want to pass following query parameters :

`query, orderBy,order,offset,limit`

So modify get : under /songs as below :

```
/songs:
  description: Collection of available Songs in Jukebox.
  get:
    description: Get a list of Songs.
    queryParameters:
        query:
        orderBy:
        order:
        offset:
        limit:
```

6) You should be able to describe query parameters.For, query parameter "query", describe it as show below :

```
query:
  description: |
            JSON array
[{"field1","value1","operator1"},{"field2","value2","operator2"},...,{"fieldN","valueN"
,"operatorN"}] with valid searchable fields: songTitle
  example: |
            "[\"songTitle\", \"XYZ \", \"like\"]"
```

7) Similarly, describe other query parameters also as shown below :

```
      orderBy:
         description: |
           Order by field: songTitle
         type: string
         required: false
       order:
         description: Order
         enum: [desc, asc]
         default: desc
         required: false
       offset:
         description: Skip over a number of elements by specifying an offset value
         type: integer
         required: false
         example: 20
         default: 0
       limit:
         description: Limit the number of elements on the response
         type: integer
         required: false
         example: 80
         default: 10
```

7) Describe the responses for get /songs as shown below :

```
responses:
      200:
        body:
          application/json:
            example: |
              !include jukebox-include-songs.sample
```

8) Describe post for /songs as below :

```
post:
    description: Add a new song to Jukebox.
    queryParameters:
      access_token:
            description: "The access token provided by the authentication application"
            example: AABBCCDD
            required: true
            type: string
    body:
      application/json:
        example: |
           !include jukebox-include-song-new.sample
        schema: !include jukebox-include-song.schema
    responses:
      200:
        body:
          application/json:
            example: |
              { "message": "The song has been properly entered" }
```

9) Similarly, you can describe get /songs/{songId } as shown below :

```
/{songid}:
    get:
      description: |
          Get the Song
          with songId ={songId}
      responses:
          200:
            body:
              application/json:
                example: |
                  !include jukebox-include-song-retrieve.sample
          404:
            body:
              application/json:
                example: |
                  {"message": "song not found" }
```

10) for get /artists and get /albums also, we want same query parameters and they also have similar response types. So, to avoid duplicates , we can defined Schemas ,resource Types and traits as shown below :

```
schemas:
 - song: !include jukebox-include-song.schema
 - artist: !include jukebox-include-artist.schema
 - album: !include jukebox-include-album.schema


resourceTypes:
  - collection:
      description: Collection of available <<resourcePathName>> in Jukebox.
      get:
        description: Get a list of <<resourcePathName>>.
        responses:
          200:
            body:
              application/json:
                example: |
                  <<exampleCollection>>
      post:
        description: |
          Add a new <<resourcePathName|!singularize>> to Jukebox.
        queryParameters:
          access_token:
            description: "The access token provided by the authentication application"
            example: AABBCCDD
            required: true
            type: string
        body:
          application/json:
            schema: <<resourcePathName|!singularize>>
            example: |
              <<exampleItem>>
        responses:
          200:
            body:
              application/json:
                example: |
                  { "message": "The <<resourcePathName|!singularize>> has been properly
entered" }
  - collection-item:
      description: Entity representing a <<resourcePathName|!singularize>>
      get:
        description: |
          Get the <<resourcePathName|!singularize>>
          with <<resourcePathName|!singularize>>Id =
          {<<resourcePathName|!singularize>>Id}
        responses:
          200:
            body:
              application/json:
```

```
            example: |
              <<exampleItem>>
        404:
          body:
            application/json:
              example: |
                {"message": "<<resourcePathName|!singularize>> not found" }
```

11) Configure traits as below :

```
traits:
  - searchable:
      queryParameters:
        query:
          description: |
            JSON array
[{"field1","value1","operator1"},{"field2","value2","operator2"},...,{"fieldN","valueN"
,"operatorN"}] <<description>>
          example: |
            <<example>>
  - orderable:
      queryParameters:
        orderBy:
          description: |
            Order by field: <<fieldsList>>
          type: string
          required: false
        order:
          description: Order
          enum: [desc, asc]
          default: desc
          required: false
  - pageable:
      queryParameters:
        offset:
          description: Skip over a number of elements by specifying an offset value for the
query
          type: integer
          required: false
          example: 20
          default: 0
        limit:
          description: Limit the number of elements on the response
          type: integer
          required: false
          example: 80
          default: 10
```

12) Modify / songs resource as below

```
/songs:
  type:
    collection:
      exampleCollection: !include jukebox-include-songs.sample
      exampleItem: !include jukebox-include-song-new.sample
  get:
    is: [
          searchable: {description: "with valid searchable fields: songTitle", example:
"[\"songTitle\", \"Get L\", \"like\"]"},
          orderable: {fieldsList: "songTitle"},
          pageable
        ]
  /{songId}:
    type:
      collection-item:
        exampleItem: !include jukebox-include-song-retrieve.sample
```