**Criar aplicação Quarkus**

* Acessar io.quarkus e adicionar as dependências necessárias
  + RESTEasy JAX-RS
  + RESTEasy Jacson
* Adicionar extensões pelo terminal
  + mvn quarkus:list-extensions
  + mvn quarkus:add-extension -Dextensions=”quarkus-resteasy-jackson”

**Aplicação Quarkus**

* Criar Movie Resource
  + Criar método para criar lista de Movies
  + Criar endpoint para listar os filmes
  + Criar endpoint para adicionar um filme à lista
  + Criar endpoint para mostrar o tamanho da lista de filmes
  + Criar endpoint para atualizar filme
  + Criar endpoint para deletar o filme
* Criar Movie.class
  + Long id
  + String title

**Open API Quarkus**

* @Tag(name = “Movie Resource”, desciption = “Movie REST APIs”)
  + Colocar uma tag para iniciar a documentação
  + Pode colocar na classe
* @Operation(

operationId = “getMovies”,

summary = “Get Movies”,

description = “Get all movies inside the list”)

* + Serve para fazer uma breve descrição do método que será usado na requisição
* @APIResponse(

responseCode = “200”,

description = “Operation Completed”,

content = @Content(mediaType = MediaType.APPLICATION.JSON)

* + Serve para colocar as descrições caso ocorra algum status code.
* @Operation(  
   operationId = "insertMovie",  
   summary = "Insert Movie",  
   description = "Insert movie inside the list"  
  )  
  @APIResponse(  
   responseCode = "201",  
   description = "Operation Completed",  
   content = @Content(mediaType = MediaType.*APPLICATION\_JSON*)  
  )  
  @POST  
  public Response insertMovie(  
   @RequestBody(  
   description = "Movie to create",  
   required = true,  
   content = @Content(schema = @Schema(implementation = Movie.class))  
   )  
   Movie newMovie) {  
   movieList.add(newMovie);  
   return Response.*status*(Response.Status.*OK*.getStatusCode()).build();  
  }

**Swagger UI**

* As configurações já vão ser replicadas devido a implementação do OPEN API, basta acessar o endpoint localhost:8080/q/Swagger-ai
* **Path:**
  + Caso queiramos alterar o path da documentação precisamos alterar o arquivo application.properties
    - quarkus.swagger-ui.path=/api-docs

**Microprofile RestClient**

* **Utilizado a API** <https://www.tvmaze.com/>
* **Criar classe TvMazeResource**
  + **Criar lista de TvSerie**
  + **Criar método para listar as séries**
  + **Criar método para povoar a lista de TvSerie**
  + **Criar método para povoar o SET de genres**
* **Criar classe TvSerie**
  + **Criar variáveis:**
    - **Long id**
    - **String name**
    - **URL url**
    - **String summary**
    - **STring language**
    - **Set<String> genres**
    - **URL officialSite**
  + **Criar construtor recebendo todas as variáveis**
* **Criar variável para acessar a api TV MAZE Proxy**
  + **Inserir anotação @RestClient**
* **Criar classe TvMazeProxy**
  + **Inserir a anotação @RegisterRestClient(configkey = “tvmaze”)**
    - **Inserir no application.properties as configurações necessárias para puxar a url.**
      * **restclient-tvmaze/mp-rest/url=https://api.tvmaze.com/**
* Criar a classe EpisodeProxy
  + Inserir nessa classe a lista de episódios pelo id retirado do TvMazeProxy

**CORS – Cross Origin Resource Sharing**

**Quando vamos expor nossa aplicação para fora do ambiente local, as vezes ao tentarem acessar nossa api, as aplicações podem apresentar erro de CORS e isso acontece, pois, nossa API não está pronta para receber tais requisições. Então é necessário que preparemos nossa API para suportar, com os seguintes passos, no arquivo application properties:**

* **application.properties**
  + **quarkus.http.cors=true**
  + **quarkus.http.cors.origins=\***
  + **quarkus.http.cors.methods=GET, POST, PUT, DELETE**
  + **quarkus.http.cors.exposed-headers=Content-Disposition**
  + **quarkus.http.cors.access-control-max-age=24H**
  + **quarkus.http.cors.access-control-allow-credentials=true**

**Conectando ao banco de dados com Quarkus**

* Alterar o application.properties
  + quarkus.datasource.db-kind=mysql
  + quarkus.datasource.username=root
  + quarkus.datasource.password=Nary0000
  + quarkus.datasource.jdbc.url=jdbc:mysql://localhost:3306/banco
  + quarkus.hinernate-orm.database.generation=drop-and-create

**Criando NamedNativeQueries**

* @NamedNativeQuery(name=”LISTAR\_CLIENTE”, query=”SELECT \* FROM CLIENTE”, resultClass=Cliente.class
* @NamedNativeQuery(name=”INSERIR\_CLIENTE”, query=”INSERT INTO CLIENTE(nome, idade)VALUE(?1,?2)

**Usando NamedNativeQuery**

* **Listando cliente:**
  + List<Cliente> listarCliente = getEntityManager().createNamedQuery(“LISTAR\_CLIENTE”, Cliente.class).gerResultList();
* **Inserindo Cliente:**
  + Query query = getEntityManager().createNamedQuery(“INSERIR\_CLIENTE”).setParameter(1, cliente.getNome()).setParameter(2, cliente.getIdade());

**Quarkus Reactive SQL Clientes with MySql**