

Arquitetura de Serviços - Trabalho 3

- 1. Questões conceituais
- 2. Questões práticas
 - 2.1. Solução
 - 2.1.1. Inserir um restaurante
 - 2.1.2. Retornar todos os restaurantes
 - 2.1.3. Retornar um restaurante pelo `id`
 - 2.1.4. Consultar restaurante pelos atributos do endereço, por exemplo, consultar pela cidade retornando os restaurantes existentes na cidade
 - 2.1.5. Atualizar restaurante, por exemplo, atualizar o endereço do restaurante
 - 2.1.6. Apagar um restaurante pelo seu `id`
- 3. Anexo

Professor: Leonardo Guerreiro Azevedo Integrantes:

- Breno Rage Aboud
- Wesley Santos da Silva
- Luis Carlos Rojas Torres
- Álvaro Jr

NOTA: O repositório deste trabalho se encontra em

<https://github.com/LuisCarlosRojasTorres/MSA/blob/main/Trabalho3.md> [LINK](#)

1. Questões conceituais

1. Defina especificação OpenAPI.

- A OAS(OpenAPI Specification) é uma especificação que define padrões para descrever e documentar APIs REST facilitando o consumo por parte dos clientes de API descrevendo de maneira clara dos os componentes de uma API como endpoints, parametros, payloads e métodos HTTP. A especificação pode ser construída com YAML ou JSON, dois tipos de linguagem de máquina que são lidas pelas ferramentas de exibição da documentação. A OAS é uma iniciativa open-source fomentada pela OpenAPI e mantida pela comunidade.

2. Qual a vantagem de se especificar o contrato de um serviço

- Especificar um contrato de um serviço trás várias vantagens técnicas ao processo de desenvolvimento de software como a melhora na interoperabilidade do serviço, também melhora as condições para reuso do serviço. Porém a maior vantagem é no campo organizacional, a especificação do serviço propicia uma melhor comunicação entre os stakeholders envolvidos no uso daquele serviço pois deixa claro quais são as condições de uso como os parametros de entrada e saída, a descrição do serviço, o método HTTP do serviço, o endpoint entre outras melhorias no processo de desenvolvimento.

2. Questões práticas

- Elabore o contrato OpenAPI para o serviço web RESTful referente a tarefa de implementação de serviços RESTful. Considere a especificação do contrato para os endpoints do serviço e para os respectivos esquemas:
 - Inserir um restaurante
 - Retornar todos os restaurantes:
 - Retornar um restaurante pelo `id`
 - Consultar restaurante pelos atributos do endereço, por exemplo, consultar pela cidade retornando os restaurantes existentes na cidade
 - Atualizar restaurante, por exemplo, atualizar o endereço do restaurante
 - Apagar um restaurante pelo seu `id`
- Inclua os seguintes itens na sua resposta a esta tarefa:
 - Especificação OpenAPI do serviço em formato YAML e em formato JSON.
 - Explicação dos passos utilizados para elaborar o contrato.
 - Printscreen da especificação no Swagger Editor.
 - Printscreen da execução do serviço empregando o Swagger Editor.

2.1. Solução

- Para apresentar a solução deste trabalho se utilizara a seguinte data (também utilizada no trabalho anterior):

name	Restaurant1	Restaurant2	Restaurant3
adress			
postalCode	111	222	33
streetAddress	Rua Um número 1111	Rua Dois número 2222	Rua Tres número 3333
adressLocality	Cidade1	Cidade2	Cidade1
addressRegion	Region1	Region2	Region3
addressCountry	Country1	Country2	Country3
url	www.rest1.com.br	www.rest2.com.br	www.rest3.com.br
menu	www.rest1.com.br/menu	www.rest2.com.br/menu	www.rest3.com.br/menu
telephone	111-1111	222-2222	333-3333
priceRange	\$11-111	\$22-222	\$33-333

- Esta data convertida no formato json ficaria da seguinte forma:

```
restaurantsList = [  
  {  
    "id": 1,  
    "name": "Restaurant1",  
    "address": {  
      "postalCode": "111",  
      "streetAddress": "Rua Um número 1111",  
      "addressLocality": "Cidade1",  
      "addressRegion": "Region1",  
      "addressCountry": "Country1",  
    },  
    "url": "www.rest1.com.br",  
    "menu": "www.rest1.com.br/menu",  
  },  
]
```

```
"telephone": "11 1111-1111",
"priceRange": "$$"
},
{
  "id": 2,
  "name": "Restaurant2",
  "address": {
    "postalCode": "222",
    "streetAddress": "Rua Dois número 2222",
    "addressLocality": "Cidade2",
    "addressRegion": "Region2",
    "addressCountry": "Country2",
  },
  "url": "www.rest2.com.br",
  "menu": "www.rest2.com.br/menu",
  "telephone": "22 22222-2222",
  "priceRange": "$$$"
},
{
  "id": 3,
  "name": "Restaurant3",
  "address": {
    "postalCode": "111",
    "streetAddress": "Rua Um número 3333",
    "addressLocality": "Cidade1",
    "addressRegion": "Region3",
    "addressCountry": "Country3",
  },
  "url": "www.rest3.com.br",
  "menu": "www.rest3.com.br/menu",
  "telephone": "33 33333-3333",
  "priceRange": "$$$$"
}
]
```

Por outro lado, precisa-se definir os **schemas** para facilitar a definição das interfaces.

```
components:
  schemas:
    Restaurants:
      type: object
      properties:
        name:
          type: string
          example: "Restaurant DummyName"
        address:
          $ref: '#/components/schemas/Address'
        url:
          type: string
          example: "www.dummy-restaurant1.com.br"
        menu:
          type: string
```

```

    example: "Region1"
  telephone:
    type: string
    example: "XX XXXXX-XXXX"
  priceRange:
    type: string
    example: "$$"
  xml:
    name: restaurants
Address:
  type: object
  properties:
    postalCode:
      type: string
      example: "111"
    streetAddress:
      type: string
      example: "Rua Um número 1111"
    addressLocality:
      type: string
      example: "Cidade1"
    addressRegion:
      type: string
      example: "Region1"
    addressCountry:
      type: string
      example: "Country1"
  xml:
    name: address

```

2.1.1. Inserir um restaurante

Especificação OpenAPI do serviço em formato YAML e em formato JSON

- YAML

```

/restaurants:
  post:
    tags:
      - restaurants
    description: Add a new restaurant to the list
    operationId: addRestaurant
    requestBody:
      description: Create a new restaurant in the list
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Restaurants'
        application/xml:
          schema:
            $ref: '#/components/schemas/Restaurants'

```

```

    required: true
  responses:
    '200':
      description: Successful operation
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Restaurants'
        application/xml:
          schema:
            $ref: '#/components/schemas/Restaurants'
    '400':
      description: Invalid input

```

- JSON

```

"/restaurants": {
  "post": {
    "tags": [
      "restaurants"
    ],
    "description": "Add a new restaurant to the list",
    "operationId": "addRestaurant",
    "requestBody": {
      "description": "Create a new restaurant in the list",
      "content": {
        "application/json": {
          "schema": {
            "$ref": "#/components/schemas/Restaurants"
          }
        },
        "application/xml": {
          "schema": {
            "$ref": "#/components/schemas/Restaurants"
          }
        }
      }
    },
    "required": true
  },
  "responses": {
    "200": {
      "description": "Successful operation",
      "content": {
        "application/json": {
          "schema": {
            "$ref": "#/components/schemas/Restaurants"
          }
        },
        "application/xml": {
          "schema": {
            "$ref": "#/components/schemas/Restaurants"
          }
        }
      }
    }
  }
}

```

```
    }
  },
  "400": {
    "description": "Invalid input"
  }
}
```

Explicação dos passos utilizados para elaborar o contrato

- Tags (**tags**): As tags são usadas para agrupar operações relacionadas. Neste caso, a operação **POST** também está associada à tag **restaurants**.
- Description (**description**): A descrição fornece detalhes adicionais sobre o que a operação faz. Aqui, descreve-se que a operação **POST** é responsável por inserir um restaurante à lista de restaurantes.
- OperationId (**operationId**): O **operationId** é um identificador único para a operação. Neste caso, o **operationId** é **addRestaurant**.
- Request Body (**requestBody**): O **requestBody** descreve o corpo da solicitação que a operação espera receber. No **POST**, é um objeto **restaurant** definido no **schema Restaurants**. Pode ser JSON, XML ou outro formato.
- Responses (**responses**): As respostas definem os códigos de status HTTP que a operação pode retornar, juntamente com uma descrição de cada código. No exemplo do **POST**, temos respostas para sucesso (código 200) e para casos de entrada inválida (código 400) ou restaurante não encontrado (código 404)....

Printscreen da especificação no Swagger Editor

```
paths:
  /restaurants:
    get:
    post:
      tags:
        - restaurants
      description: Add a new restaurant to the list
      operationId: addRestaurant
      requestBody:
        description: Create a new pet in the store
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Restaurants'
          application/xml:
            schema:
              $ref: '#/components/schemas/Restaurants'
        required: true
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Restaurants'
            application/xml:
              schema:
                $ref: '#/components/schemas/Restaurants'
        '400':
          description: Invalid input
```

Printscreen da execução do serviço empregando o Swagger Editor

- Printscreen antes de executar o comando **POST**

POST

/restaurants

⌵

Add a new restaurant to the list

Parameters

Cancel

No parameters

Request body required

application/json

⌵

Create a new pet in the store

```
{
  "name": "Restaurant DummyName",
  "address": {
    "postalCode": "123",
    "streetAddress": "Rua Dummy número 1234",
    "addressLocality": "CidadeX",
    "addressRegion": "RegionX",
    "addressCountry": "CountryX"
  },
  "url": "www.dummy-restaurant1.com.br",
  "menu": "Region1",
  "telephone": "XX XXXXX-XXXX",
  "priceRange": "$$"
}
```

Execute

- Printscreen depois de executar o comando **POST**

Code	Description	Links
200	<div>Successful operation</div> <div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "name": "Restaurant DummyName", "address": { "postalCode": "123", "streetAddress": "Rua Dummy número 1234", "addressLocality": "CidadeX", "addressRegion": "RegionX", "addressCountry": "CountryX" }, "url": "www.dummy-restaurant1.com.br", "menu": "Region1", "telephone": "XX XXXXX-XXXX", "priceRange": "\$\$" }</pre></div>	No links
400	<div>Invalid input</div>	No links

2.1.2. Retornar todos os restaurantes

Especificação OpenAPI do serviço em formato YAML e em formato JSON

- YAML:


```

/restaurants:
  get:
    tags:
      - restaurants
    description: Get a list of all the restaurants
    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Restaurants'
      '400':
        description: Invalid ID supplied

```

- JSON:

```

"/restaurants": {
  "get": {
    "tags": [
      "restaurants"
    ],
    "description": "Get a list of all the restaurants",
    "responses": {
      "200": {
        "description": "Successful operation",
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/Restaurants"
            }
          }
        }
      },
      "400": {
        "description": "Invalid ID supplied"
      }
    }
  }
},

```

Explicação dos passos utilizados para elaborar o contrato

- Tags (**tags**): As tags são usadas para agrupar operações relacionadas. Neste caso, a operação **GET** também está associada à tag **restaurants**.
- Description (**description**): A descrição fornece detalhes adicionais sobre o que a operação faz. Aqui, descreve-se que a operação **GET** é responsável por obter a lista de todos os restaurantes.

- OperationId (**operationId**): O **operationId** é um identificador único para a operação. Neste caso, o **operationId** é **addRestaurant**.
- Responses (**responses**): As respostas definem os códigos de status HTTP que a operação pode retornar, juntamente com uma descrição de cada código. No exemplo do **POST**, temos respostas para sucesso (código 200) e para casos de entrada inválida (código 400) ou restaurante não encontrado (código 404).

Printscreen da especificação no Swagger Editor

```
paths:
  /restaurants:
    get:
      tags:
        - restaurants
      description: Get a list of all the restaurants
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Restaurants'
        '400':
          description: Invalid ID supplied
```

Printscreen da execução do serviço empregando o Swagger Editor

- Printscreen antes de executar o comando **GET**

GET

/restaurants

Get a list of all the restaurants

Parameters

Cancel

No parameters


Execute

Responses

Code	Description	Links
200	<div>Successful operation</div> <div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "id": 1, "name": "Restaurant DummyName", "address": { "postalCode": "111", "streetAddress": "Rua Um número 1111", "addressLocality": "Cidade1", "addressRegion": "Region1", "addressCountry": "Country1" }, "url": "www.dummy-restaurant1.com.br", "menu": "Region1", "telephone": "XX XXXXX-XXXX", "priceRange": "\$\$" }</pre></div>	

No links

- Printscreen depois de executar o comando GET

Code	Details
200	<div>Response body</div> <pre>[{ "address": { "addressCountry": "Country1", "addressLocality": "Cidade1", "addressRegion": "Region1", "postalCode": "111", "streetAddress": "Rua Um número 1111" }, "id": 1, "menu": "www.rest1.com.br/menu", "name": "Restaurant1", "priceRange": "\$\$", "telephone": "11 11111-1111", "url": "www.rest1.com.br" }, { "address": { "addressCountry": "Country2", "addressLocality": "Cidade2", "addressRegion": "Region2", "postalCode": "222", "streetAddress": "Rua Dois número 2222" } }]</pre> <div> Download</div>

2.1.3. Retornar um restaurante pelo **id**

Especificação OpenAPI do serviço em formato YAML e em formato JSON

- YAML:

```
/restaurants/{restaurant_id}:
  get:
    tags:
      - restaurants
    summary: Get restaurant by ID
    description: Returns a single restaurant
    operationId: restaurant_id
    parameters:
      - name: restaurant_addr
        in: path
        description: ID of the restaurant to return
        required: true
        schema:
          type: string
    requestBody:
      description: ID of the restaurant to return
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Restaurants'
        application/xml:
          schema:
            $ref: '#/components/schemas/Restaurants'
    responses:
```

```

'200':
  description: A single restaurant
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/Restaurants'
'400':
  description: Invalid input
'404':
  description: Restaurant not found

```

- JSON:

```

{
  "/restaurants/{restaurant_id}": {
    "get": {
      "tags": [
        "restaurants"
      ],
      "summary": "Get restaurant by ID",
      "description": "Returns a single restaurant..",
      "operationId": "findRestaurant",
      "parameters": [
        {
          "name": "restaurant_id",
          "in": "path",
          "description": "ID of the restaurant to return",
          "required": true,
          "schema": {
            "type": "integer"
          }
        }
      ],
      "requestBody": {
        "description": "single restaurant request body",
        "required": true,
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/Restaurants"
            }
          },
          "application/xml": {
            "schema": {
              "$ref": "#/components/schemas/Restaurants"
            }
          }
        }
      },
      "responses": {
        "200": {

```

```
    "description": "A single restaurant",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/Restaurants"
        }
      }
    },
    "400": {
      "description": "Invalid input"
    },
    "404": {
      "description": "Restaurant not found"
    }
  }
}
```

Explicação dos passos utilizados para elaborar o contrato

- Tags (tags): As tags são utilizadas para agrupar operações relacionadas. Neste caso, a operação GET também está associada à tag restaurants.
- Summary (summary): O sumário fornece uma descrição breve e clara da funcionalidade da operação. Aqui, é indicado que a operação GET é responsável por obter um restaurante com base em seu ID.
- Description (description): A descrição fornece detalhes adicionais sobre o propósito da operação. Neste caso, descreve-se que a operação GET permite recuperar um restaurante específico com base em seu identificador único (ID).
- OperationId (operationId): O operationId é um identificador único para a operação. Aqui, o operationId é definido como findRestaurant.
- Parameters (parameters): Os parâmetros especificam as informações necessárias para a operação. No exemplo do GET, o parâmetro de caminho restaurant_id é essencial para identificar o restaurante desejado.
- Responses (responses): As respostas definem os códigos de status HTTP que a operação pode retornar, juntamente com uma descrição para cada código. No caso do GET, são fornecidas respostas para sucesso (código 200), entrada inválida (código 400) e restaurante não encontrado (código 404).

Printscreen da especificação no Swagger Editor

```
paths:
  /restaurants/{restaurant_id}:
    get:
      tags:
        - restaurants
      summary: Get restaurant by ID
      description: Returns a single restaurant
      operationId: restaurant_id
      parameters:
        - name: restaurant_id
          in: path
          description: ID of the restaurant to return
          required: true
          schema:
            type: string
      requestBody:
        description: ID of the restaurant to return
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Restaurants'
          application/xml:
            schema:
              $ref: '#/components/schemas/Restaurants'
      responses:
        '200':
          description: A single restaurant
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Restaurants'
        '400':
          description: Invalid input
        '404':
          description: Restaurant not found
```

Printscreen da execução do serviço empregando o Swagger Editor

- Printscreen antes de executar o comando **GET**

GET

/api/restaurants/{restaurant_id} Get a restaurant by ID. getRestaurantById

Returns a single restaurant.

Parameters

Cancel

Name	Description
restaurant_id * required (path)	ID of the restaurant to return

1

Execute

Clear

Responses


Response content type application/json

Curl

```
curl -X GET "http://localhost:5000/api/restaurants/1" -H "accept: application/json"
```

- Printscreen depois de executar o comando GET

Server response

Code	Details
200	<div>Response body<pre>{ "address": { "addressCountry": "Country1", "addressLocality": "Cidade1", "addressRegion": "Region1", "postalCode": "111", "streetAddress": "Rua Um número 1111" }, "id": 1, "menu": "www.rest1.com.br/menu", "name": "Restaurant1", "priceRange": "\$\$", "telephone": "11 11111-1111", "url": "www.rest1.com.br"}</pre><div> Download</div></div> <div>Response headers<pre>access-control-allow-headers: Content-Type,Authorization access-control-allow-methods: GET,PUT,POST,DELETE,OPTIONS access-control-allow-origin: * connection: close content-length: 349 content-type: application/json date: Fri05 Apr 2024 13:59:16 GMT server: Werkzeug/3.0.1 Python/3.10.5</pre></div>
Responses	
Code	Description
200	A single restaurant
404	Restaurant not found

2.1.4. Consultar restaurante pelos atributos do endereço, por exemplo, consultar pela cidade retornando os restaurantes existentes na cidade

Especificação OpenAPI do serviço em formato YAML e em formato JSON

```
/restaurants/{restaurant_addr}:
  get:
    tags:
      - restaurants
    summary: Get restaurants by city
    description: Returns a list of restaurants in a specific city.
    operationId: getRestaurantsByCity
    parameters:
      - name: restaurant_addr
```

```

    in: path
    description: City name to filter restaurants
    required: true
    schema:
      type: string
  requestBody:
    description: Restaurant object to restaurant_addr
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Restaurants'
      application/xml:
        schema:
          $ref: '#/components/schemas/Restaurants'
  responses:
    '200':
      description: A list of restaurants in the specified city
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Restaurants'
    '400':
      description: Invalid input
    '404':
      description: Restaurant not found

```

- JSON:

```

{
  "/restaurants/{restaurant_addr}": {
    "get": {
      "tags": [
        "restaurants"
      ],
      "summary": " Get restaurants by city",
      "description": "Returns a list of restaurants in a specific city.",
      "operationId": "getRestaurantsByCity",
      "parameters": [
        {
          "name": "restaurant_addr",
          "in": "path",
          "description": "City name to filter restaurants",
          "required": true,
          "schema": {
            "type": "string"
          }
        }
      ],
      "requestBody": {
        "description": "City name to filter restaurants and retrieve a

```

```
list of restaurants in the specified city.",
  "required": true,
  "content": {
    "application/json": {
      "schema": {
        "$ref": "#/components/schemas/Restaurants"
      }
    },
    "application/xml": {
      "schema": {
        "$ref": "#/components/schemas/Restaurants"
      }
    }
  }
},
"responses": {
  "200": {
    "description": "A list of restaurants in the specified city",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/Restaurants"
        }
      }
    }
  },
  "400": {
    "description": "Invalid input"
  },
  "404": {
    "description": "Restaurant not found"
  }
}
}
```

Explicação dos passos utilizados para elaborar o contrato

- Tags (tags): As tags são utilizadas para agrupar operações relacionadas. Neste caso, a operação GET também está associada à tag restaurants.
- Summary (summary): O sumário fornece uma descrição breve e clara da funcionalidade da operação. Aqui, é indicado que a operação GET é responsável por obter uma lista de restaurantes com base na cidade.
- Description (description): A descrição fornece detalhes adicionais sobre o propósito da operação. Neste caso, descreve-se que a operação GET permite recuperar uma lista de restaurantes localizados em uma cidade específica.
- OperationId (operationId): O operationId é um identificador único para a operação. Aqui, o operationId é definido como getRestaurantsByCity.

- Parameters (parameters): Os parâmetros especificam as informações necessárias para a operação. No exemplo do GET, o parâmetro de caminho `restaurant_addr` é essencial para filtrar os restaurantes pela cidade.
- Request Body (requestBody): O `requestBody` descreve o corpo da solicitação que a operação espera receber. No GET, isso inclui o nome da cidade para filtrar os restaurantes e recuperar uma lista dos restaurantes na cidade especificada. Pode ser JSON, XML ou outro formato.
- Responses (responses): As respostas definem os códigos de status HTTP que a operação pode retornar, juntamente com uma descrição para cada código. No caso do GET, são fornecidas respostas para sucesso (código 200), entrada inválida (código 400) e restaurante não encontrado (código 404).

Printscreen da especificação no Swagger Editor

```
paths:
  /restaurants/{restaurant_addr}:
    get:
      tags:
        - restaurants
      summary: Get restaurants by city
      description: Returns a list of restaurants in a specific city.
      operationId: getRestaurantsByCity
      parameters:
        - name: restaurant_addr
          in: path
          description: City name to filter restaurants
          required: true
          schema:
            type: string
      requestBody:
        description: Restaurant object to restaurant_addr
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Restaurants'
          application/xml:
            schema:
              $ref: '#/components/schemas/Restaurants'
      responses:
        '200':
          description: A list of restaurants in the specified city
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Restaurants'
        '400':
          description: Invalid input
        '404':
          description: Restaurant not found
```

Printscreen da execução do serviço empregando o Swagger Editor

- Printscreen antes de executar o comando `GET`

GET

/api/restaurants/{restaurant_addr}

Get restaurants by city.

getRestaurantsByCity

Returns a list of restaurants in a specific city.

Parameters

Cancel

Name

Description

restaurant_addr * required

City name to filter restaurants

(path)

Cidade1

Execute

Clear

Responses

Response content type

application/json

Curl

curl -X GET "http://localhost:5000/api/restaurants/Cidade1" -H "accept: application/json"

Request URL

http://localhost:5000/api/restaurants/Cidade1

- Printscreen depois de executar o comando GET

Request URL

`http://localhost:5000/api/restaurants/Cidade1`

Server response

Code	Details
200	<p>Response body</p> <pre>[{"address": {"addressCountry": "Country1", "addressLocality": "Cidade1", "addressRegion": "Region1", "postalCode": "111", "streetAddress": "Rua Um n\u00famero 1111"}, "id": 1, "menu": "www.rest1.com.br/menu", "name": "Restaurant1", "priceRange": "\$\$", "telephone": "11 11111-1111", "url": "www.rest1.com.br"}, {"address": {"addressCountry": "Country3", "addressLocality": "Cidade1", "addressRegion": "Region3", "postalCode": "111", "streetAddress": "Rua Um n\u00famero 3333"}, "id": 3, "menu": "www.rest3.com.br/menu", "name": "Restaurant3", "priceRange": "\$\$\$\$ ", "telephone": "33 33333-3333", "url": "www.rest3.com.br"}]</pre> <p>Response headers</p> <pre>access-control-allow-headers: Content-Type,Authorization access-control-allow-methods: GET,PUT,POST,DELETE,OPTIONS access-control-allow-origin: * connection: close content-length: 765 content-type: application/json date: Fri05 Apr 2024 13:59:38 GMT server: Werkzeug/3.0.1 Python/3.10.5</pre>

Responses

Code	Description
200	A list of restaurants in the specified city
404	No restaurants found in the specified city

2.1.5. Atualizar restaurante, por exemplo, atualizar o endereço do restaurante

Especificação OpenAPI do serviço em formato YAML e em formato JSON

- YAML:

```
/restaurants/{restaurant_id}:
  put:
    tags:
      - restaurants
    summary: Update a restaurant
    description: Updates the information of a specific restaurant.
```

```

operationId: updateRestaurant
parameters:
  - name: restaurant_id
    in: path
    description: ID of the restaurant to update
    required: true
    schema:
      type: integer
requestBody:
  description: Restaurant object to update
  required: true
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/Restaurants'
    application/xml:
      schema:
        $ref: '#/components/schemas/Restaurants'
responses:
  '200':
    description: Restaurant updated successfully
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Restaurants'
  '400':
    description: Invalid input
  '404':
    description: Restaurant not found

```

- JSON:

```

{
  "/restaurants/{restaurant_id}": {
    "put": {
      "tags": [
        "restaurants"
      ],
      "summary": "Update a restaurant",
      "description": "Updates the information of a specific restaurant.",
      "operationId": "updateRestaurant",
      "parameters": [
        {
          "name": "restaurant_id",
          "in": "path",
          "description": "ID of the restaurant to update",
          "required": true,
          "schema": {
            "type": "integer"
          }
        }
      ]
    }
  }
}

```

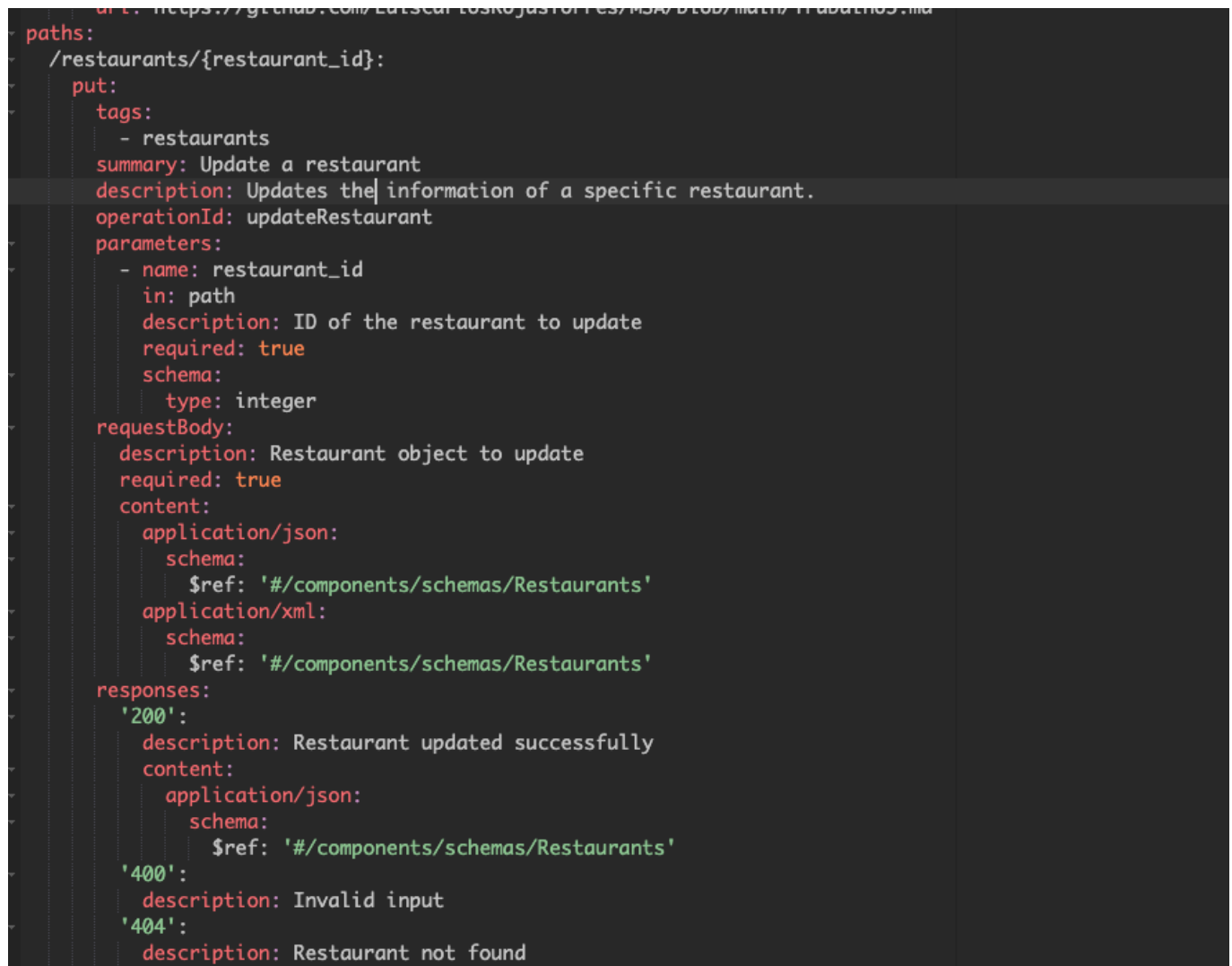
```
    ],
    "requestBody": {
      "description": "Restaurant object to update",
      "required": true,
      "content": {
        "application/json": {
          "schema": {
            "$ref": "#/components/schemas/Restaurants"
          }
        },
        "application/xml": {
          "schema": {
            "$ref": "#/components/schemas/Restaurants"
          }
        }
      }
    },
    "responses": {
      "200": {
        "description": "Restaurant updated successfully",
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/Restaurants"
            }
          }
        }
      },
      "400": {
        "description": "Invalid input"
      },
      "404": {
        "description": "Restaurant not found"
      }
    }
  }
}
```

Explicação dos passos utilizados para elaborar o contrato

- Tags (tags): As tags são usadas para agrupar operações relacionadas. Neste caso, a operação PUT também está associada à tag "restaurants".
- Summary (summary): O resumo fornece uma breve descrição do que a operação faz. No caso da operação PUT, o resumo indica que ela é usada para atualizar um restaurante.
- Description (description): A descrição fornece detalhes adicionais sobre o que a operação faz. Aqui, descreve-se que a operação PUT é responsável por atualizar as informações de um restaurante específico.

- **OperationId (operationId):** O operationId é um identificador único para a operação. Neste caso, o operationId é "updateRestaurant".
- **Parameters (parameters):** Os parâmetros especificam quais informações a operação espera receber. No exemplo do PUT, temos um parâmetro de caminho (path parameter) chamado restaurant_id, que identifica o restaurante a ser atualizado.
- **Request Body (requestBody):** O requestBody descreve o corpo da solicitação que a operação espera receber. No PUT, isso geralmente inclui os dados que serão usados para atualizar o restaurante. Pode ser JSON, XML ou outro formato.
- **Responses (responses):** As respostas definem os códigos de status HTTP que a operação pode retornar, juntamente com uma descrição de cada código. No exemplo do PUT, temos respostas para sucesso (código 200) e para casos de entrada inválida (código 400) ou restaurante não encontrado (código 404)....

Printscreen da especificação no Swagger Editor



...

Printscreen da execução do serviço empregando o Swagger Editor

- Printscreen antes de executar o comando **PUT**

PUT

/restaurants/{restaurant_id} Update a restaurant

Updates the information of a specific restaurant.

Parameters

Try it out

Name	Description
restaurant_id * required	ID of the restaurant to update
integer (path)	<input type="text" value="1"/>

Request body required

application/json

Restaurant object to update

Example Value

Schema

```
{
  "id": 1,
  "name": "Restaurant DummyName",
  "address": {
    "postalCode": "111",
    "streetAddress": "Rua Um número 1111",
    "addressLocality": "Cidade1",
    "addressRegion": "Region1",
    "addressCountry": "Country1"
  },
  "url": "www.dummy-restaurant1.com.br",
  "menu": "Region1",
  "telephone": "XX XXXXX-XXXX",
  "priceRange": "$$"
}
```

Curl

```
curl -X 'PUT' \
'http://127.0.0.1:5000/api/restaurants/1' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "id": 1,
  "name": "Restaurant DummyName Added",
  "address": {
    "postalCode": "111",
    "streetAddress": "Rua Um número 1111",
    "addressLocality": "Cidade1",
    "addressRegion": "Region1",
    "addressCountry": "Country1"
  },
  "url": "www.dummy-restaurant1.com.br",
  "menu": "Region1",
  "telephone": "XX XXXXX-XXXX",
  "priceRange": "$$"
}'
```

Request URL

```
http://127.0.0.1:5000/api/restaurants/1
```

Server response

Code

Details

200

Response body

```
{
  "address": {
    "addressCountry": "Country1",
    "addressLocality": "Cidade1",
    "addressRegion": "Region1",
    "postalCode": "111",
    "streetAddress": "Rua Um número 1111"
  },
  "id": 1,
  "menu": "Region1",
  "name": "Restaurant DummyName Added",
  "priceRange": "$$",
  "telephone": "XX XXXXX-XXXX",
  "url": "www.dummy-restaurant1.com.br"
}
```



Download

...

2.1.6. Apagar um restaurante pelo seu **id**

Especificação OpenAPI do serviço em formato YAML e em formato JSON

- YAML:

```
paths:
  /restaurants/{restaurant_id}:
    delete:
      tags:
        - restaurants
      summary: Delete a restaurant by ID
      description: Deletes a restaurant specified by its ID.
      operationId: deleteRestaurant
```

```
parameters:
  - name: restaurant_id
    in: path
    description: ID of the restaurant to delete
    required: true
    schema:
      type: integer
responses:
  '200':
    description: Restaurant deleted successfully
  '404':
    description: Restaurant not found
```

- JSON:

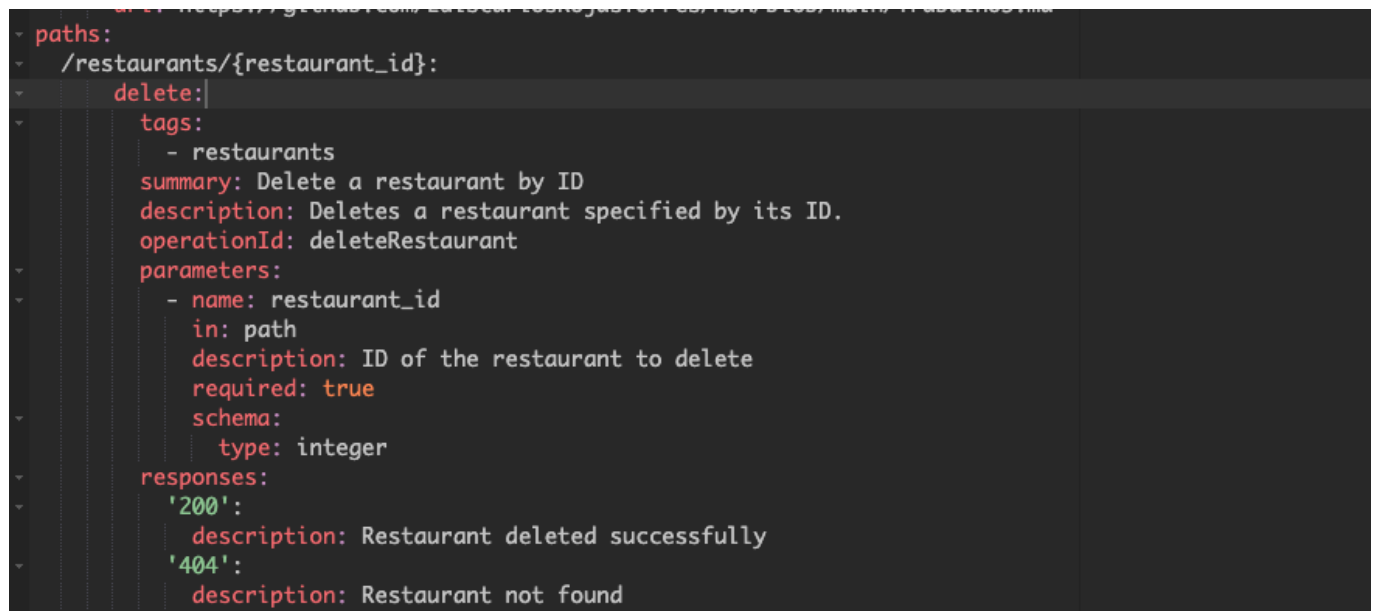
```
"paths": {
  "/restaurants/{restaurant_id}": {
    "delete": {
      "tags": [
        "restaurants"
      ],
      "summary": "Delete a restaurant by ID",
      "description": "Deletes a restaurant specified by its ID.",
      "operationId": "deleteRestaurant",
      "parameters": [
        {
          "name": "restaurant_id",
          "in": "path",
          "description": "ID of the restaurant to delete",
          "required": true,
          "schema": {
            "type": "integer"
          }
        }
      ],
      "responses": {
        "200": {
          "description": "Restaurant deleted successfully"
        },
        "404": {
          "description": "Restaurant not found"
        }
      }
    }
  }
}
```

...

Explicação dos passos utilizados para elaborar o contrato

- Tags (tags): As tags são utilizadas para agrupar operações relacionadas em categorias. No exemplo, a operação DELETE está associada à tag "restaurants", indicando que esta operação pertence ao contexto dos restaurantes.
- Summary (summary): O resumo fornece uma breve descrição do que a operação faz.
- Description (description): A descrição fornece uma explicação mais detalhada sobre a operação. No exemplo, descreve-se que a operação DELETE é responsável por excluir um restaurante específico com base no seu ID.
- OperationId (operationId): O operationId é um identificador único para a operação. Ele pode ser usado para fazer referência à operação em outros lugares da documentação ou do código. Neste caso, o operationId é "deleteRestaurant".
- Parameters (parameters): Parâmetros são informações que a operação espera receber. No DELETE, geralmente especificamos parâmetros de caminho (path parameters) para identificar o recurso a ser excluído. Aqui, temos o parâmetro restaurant_id no caminho, indicando o ID do restaurante a ser excluído.
- Responses (responses): As respostas definem os possíveis códigos de status HTTP que a operação pode retornar, juntamente com uma descrição do que cada código significa. No exemplo, temos duas respostas possíveis: 204, indicando que o restaurante foi excluído com sucesso, e 404, indicando que o restaurante não foi encontrado....

Printscreen da especificação no Swagger Editor



Printscreen da execução do serviço empregando o Swagger Editor

- Printscreen ao de executar o comando **DELETE**

Name	Description
restaurant_id * required	ID of the restaurant to delete
integer (path)	<input type="text" value="1"/>
<div>ExecuteClear</div>	
Responses	
Curl	
<pre>curl -X 'DELETE' \ 'http://127.0.0.1:5000/api/restaurants/1' \ -H 'accept: */*'</pre>	
Request URL	
<pre>http://127.0.0.1:5000/api/restaurants/1</pre>	
Server response	
Code	Details
200	<div>Response body</div> <pre>{ "resultMessage": "Restaurante Restaurant1, com o ID: 1 foi deletado com sucesso" }</pre> <div>Download</div> <div>Response headers</div> <pre>content-length: 82 content-type: application/json</pre>

3. Anexo

- Codigo completo do Server-side:

```
from flask import Flask, jsonify, make_response, request
from flask_cors import CORS, cross_origin

app = Flask(__name__)
CORS(app, origins=['http://localhost:8000'], always_send=False)
HOST = "localhost"
PORT = 5000

restaurantsList = [
{
  "id": 1,
  "name": "Restaurant1",
  "address": {
    "postalCode": "111",
    "streetAddress": "Rua Um número 1111",
    "addressLocality": "Cidade1",
    "addressRegion": "Region1",
    "addressCountry": "Country1",
```

```
,
"url": "www.rest1.com.br",
"menu": "www.rest1.com.br/menu",
"telephone": "11 11111-1111",
"priceRange": "$$"
},
{
  "id": 2,
  "name": "Restaurant2",
  "address": {
    "postalCode": "222",
    "streetAddress": "Rua Dois número 2222",
    "addressLocality": "Cidade2",
    "addressRegion": "Region2",
    "addressCountry": "Country2",
  },
  "url": "www.rest2.com.br",
  "menu": "www.rest2.com.br/menu",
  "telephone": "22 22222-2222",
  "priceRange": "$$$"
},
{
  "id": 3,
  "name": "Restaurant3",
  "address": {
    "postalCode": "111",
    "streetAddress": "Rua Um número 3333",
    "addressLocality": "Cidade1",
    "addressRegion": "Region3",
    "addressCountry": "Country3",
  },
  "url": "www.rest3.com.br",
  "menu": "www.rest3.com.br/menu",
  "telephone": "33 33333-3333",
  "priceRange": "$$$$"
}
]

@app.route('/')
def index():
    return "Trabalho N2! Arquitetura de Microserviços"

@app.route("/api/restaurants", methods=["GET"])
def get_restaurants():
    city = request.args.get("addressLocality")
    if city is None:
        return jsonify(restaurantsList)
    else:
        print(" - City: " + city)
        ans = list()
        for restaurant in restaurantsList:
            if restaurant["address"]["addressLocality"] == city:
                ans.append(restaurant)
```

```
        if len(restaurant) == 0:
            return not_found()
        return jsonify({"Restaurant selected by address new way": ans})

@app.route("/api/restaurants/<int:restaurant_id>", methods=["GET"])
def get_restaurant_by_index(restaurant_id):
    print("get_restaurant_by_index")
    try:
        id = int(restaurant_id)
        for restaurant in restaurantsList:
            if restaurant["id"] == id:
                if len(restaurant) == 0:
                    return not_found()
                return jsonify(restaurant)
        return not_found()
    except (ValueError, TypeError):
        return not_found()

@app.route("/api/restaurants/<restaurant_addr>", methods=["GET"])
def get_restaurant_by_City(restaurant_addr):
    print("get_restaurant_by_City")
    ans = list()
    for restaurant in restaurantsList:
        if restaurant["address"]["addressLocality"] == restaurant_addr:
            ans.append(restaurant)

    if len(restaurant) == 0 or len(ans) == 0:
        return not_found()
    return jsonify(ans)

@app.route("/api/restaurants/cities", methods=["GET"])
def get_AllCities():
    ans = list()
    for restaurant in restaurantsList:
        if restaurant["address"]["addressLocality"] not in ans:
            print("- City:" + restaurant["address"]["addressLocality"])
            ans.append(restaurant["address"]["addressLocality"])

    if len(ans) == 0:
        return not_found()
    ans.sort()
    return jsonify({"cities": ans})

@app.errorhandler(404)
def not_found():
    return make_response(jsonify({"error": "Restaurante não encontrado!"}), 404)

@app.errorhandler(400)
def bad_request(field):
    return make_response(jsonify({"error": "0 {} é mandatório!".format(field)}), 400)

@app.route("/api/restaurants", methods=["POST"])
```



```
def create_restaurant():
    if not request.json or not "name" in request.json or not
request.json["name"]:
        return bad_request("Nome")
    newRestaurant = {
        "id": restaurantsList[-1]["id"] + 1,
        "name": request.json["name"],
        "address": request.json.get("address", ""),
        "url": request.json.get("url", ""),
        "menu": request.json.get("menu", ""),
        "telephone": request.json.get("telephone", ""),
        "priceRange": request.json.get("priceRange", ""),
    }
    restaurantsList.append(newRestaurant)
    return jsonify(newRestaurant), 201

@app.route("/api/restaurants/<int:restaurant_id>", methods=["PUT",
"OPTIONS"])
def update_restaurant(restaurant_id):
    restaurant = [restaurant for restaurant in restaurantsList if
restaurant["id"] == restaurant_id]
    if len(restaurant) == 0:
        return not_found()
    if not request.json or not "name" in request.json:
        return bad_request("Name")
    if "name" in request.json and type(request.json["name"]) != str:
        return bad_request("Name")
    if "address" in request.json and type(request.json["address"]) !=
dict:
        return bad_request("Address")
    if "url" in request.json and type(request.json["url"]) is not str:
        return bad_request("URL")
    if "menu" in request.json and type(request.json["menu"]) is not str:
        return bad_request("Menu")
    if "telephone" in request.json and type(request.json["telephone"]) is
not str:
        return bad_request("Telephone")
    if "priceRange" in request.json and type(request.json["priceRange"])
is not str:
        return bad_request("Price Range")

    restaurant[0]["name"] = request.json.get("name", restaurant[0]
["name"])
    restaurant[0]["address"] = request.json.get("address", restaurant[0]
["address"])
    restaurant[0]["url"] = request.json.get("url", restaurant[0]["url"])
    restaurant[0]["menu"] = request.json.get("menu", restaurant[0]
["menu"])
    restaurant[0]["telephone"] = request.json.get("telephone",
restaurant[0]["telephone"])
    restaurant[0]["priceRange"] = request.json.get("priceRange",
restaurant[0]["priceRange"])

    return jsonify(restaurant[0])
```

```
@app.route("/api/restaurants/<int:restaurant_id>", methods=["DELETE"])
def delete_restaurants(restaurant_id):
    try:
        id = int(restaurant_id)
        for restaurant in restaurantsList:
            if restaurant["id"] == id:
                if len(restaurant) == 0:
                    return not_found()
                restaurantsList.remove(restaurant)
                return jsonify({"resultMessage": "Restaurante {}, com o
ID: {} foi deletado com sucesso".format(restaurant["name"],
restaurant["id"])}))
        return not_found()
    except (ValueError, TypeError):
        return not_found()

@app.after_request
def after_request(response):
    response.headers.add('Access-Control-Allow-Origin', '*')
    response.headers.add('Access-Control-Allow-Headers', 'Content-
Type,Authorization')
    response.headers.add('Access-Control-Allow-Methods',
'GET,PUT,POST,DELETE,OPTIONS')
    return response

if __name__ == "__main__":
    app.run(host=HOST, port=PORT, debug=True)
    app.register_error_handler(400, bad_request)
    app.register_error_handler(404, not_found)
```