

# Tutorial: Como Criar uma API REST para Listar e Buscar Livros na Biblioteca Digital (Quarkus + Jakarta REST)

## Introdução — O que e Por Que

Neste tutorial, você aprenderá a **criar uma API REST** no projeto **Biblioteca Digital** desenvolvido em **Quarkus** com **Jakarta EE**, estendendo a aplicação original que utiliza **JSF (Jakarta Faces)**.

A proposta é permitir que o sistema disponibilize seus dados para **outros aplicativos**, como apps móveis ou painéis em React, através de endpoints REST que retornam informações em **formato JSON**.

Este tutorial atende ao objetivo de **aprofundar o conhecimento em arquitetura RESTful**, integrando **camadas de serviço, repositório e entidades existentes** no projeto, sem modificar sua estrutura principal.

## Objetivo

- Expor os dados da entidade **Livro** via API REST.
- Reutilizar o **BibliotecaService** existente.
- Permitir **listar todos os livros e buscar um livro por ID**.
- Retornar os resultados no formato **JSON**, compatível com aplicações externas.

## Conceitos Fundamentais

Antes de começar, é importante entender três conceitos base usados neste tutorial:

1. **REST (Representational State Transfer):**  
Um estilo arquitetural que permite comunicação entre sistemas por meio de HTTP (GET, POST, PUT, DELETE).
2. **JAX-RS (Jakarta RESTful Web Services):**  
Especificação do Jakarta EE usada para implementar APIs RESTful.
3. **Quarkus RESTEasy Reactive:**  
Implementação performática e moderna do JAX-RS no Quarkus, integrada com JSON via Jackson.

## Pré-Requisitos

- Projeto “Biblioteca Digital” clonado e funcionando.
- Java 17 ou superior instalado.
- Maven configurado (`mvn -v` para verificar).
- PostgreSQL em execução.
- IDE (VS Code, IntelliJ ou Eclipse).

## Etapa 1 – Adicionando a Dependência REST no Projeto

Para habilitar endpoints REST e serialização JSON, adicione no `pom.xml` a seguinte dependência (depois da seção do Hibernate e CDI):

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-rest-jackson</artifactId>
</dependency>
```

💡 **Importante:** Nas versões mais recentes do Quarkus (3.x), a biblioteca antiga `quarkus-resteasy-reactive-jackson` foi substituída por `quarkus-rest-jackson`.

Após salvar o arquivo, execute:

```
mvn clean install
```

para baixar a nova dependência.

## Etapa 2 – Criando a Classe Resource (Controlador REST)

Dentro da pasta do projeto:

```
src/main/java/com/biblioteca/api/
```

crie o arquivo **LivroResource.java** com o conteúdo abaixo:

```
package com.biblioteca.api;
```

```
import com.biblioteca.model.Livro;
import com.biblioteca.service.BibliotecaService;
import jakarta.inject.Inject;
```

```
import jakarta.ws.rs.*;
import jakarta.ws.rs.core.MediaType;
import java.util.List;

@Path("/api/livros")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class LivroResource {

    @Inject
    BibliotecaService bibliotecaService;

    // Lista todos os livros cadastrados
    @GET
    public List<Livro> listarLivros() {
        return bibliotecaService.listarTodosLivros();
    }

    // Busca um livro específico pelo ID
    @GET
    @Path("/{id}")
    public Livro buscarPorId(@PathParam("id") Long id) {
        return bibliotecaService.buscarLivroPorId(id);
    }
}
```

## Etapa 3 – Reutilizando o Código Existente

O **BibliotecaService** já contém os métodos de negócio que serão reutilizados:

```
@ApplicationScoped
public class BibliotecaService {

    @Inject
    LivroRepository livroRepository;

    public List<Livro> listarTodosLivros() {
        return livroRepository.listarTodos();
    }

    public Livro buscarLivroPorId(Long id) {
        return livroRepository.buscarPorId(id);
    }
}
```



Assim, o Resource atua apenas como **ponte** entre o frontend (ou um cliente HTTP externo) e o serviço.

## Etapa 4 – Testando a API REST

Execute o projeto em modo de desenvolvimento:

```
mvn quarkus:dev
```

Abra seu navegador e acesse:

-  **Listar todos os livros:**  
<http://localhost:8080/api/livros>
-  **Buscar livro por ID:**  
<http://localhost:8080/api/livros/1>

## Etapa 5 – Exemplo de Retorno JSON

**GET /api/livros**

```
[
  {
    "id": 1,
    "titulo": "Dom Casmurro",
    "isbn": "123-4567890125",
    "dataPublicacao": "1899-01-01",
    "autor": {
      "nome": "Machado de Assis",
      "email": "machado@academia.com"
    }
  },
  ...
]
```

```
{
  "id": 2,
  "titulo": "O Guarani",
  "isbn": "123-4567890123",
  "dataPublicacao": "1857-01-01",
  "autor": {
    "nome": "José de Alencar",
    "email": "alencar@literatura.com"
  }
}
```

## Etapa 6 – Verificação de Funcionamento

Para confirmar que a API está operacional, use uma das ferramentas abaixo:

- **Navegador:** abra o link `http://localhost:8080/api/livros`
- **Postman:** realize uma requisição GET para o mesmo endpoint.
- **cURL (terminal):**
- `curl -X GET http://localhost:8080/api/livros`

O retorno deve ser um JSON contendo todos os livros do banco `import.sql`.

## Conclusão





Com este tutorial, foi criada uma API REST completa utilizando **Jakarta REST (JAX-RS)** e **Quarkus**, estendendo o projeto original de JSF.

A implementação reutiliza os serviços existentes e permite comunicação com outras aplicações via HTTP.

Esse aprimoramento:

- **Moderniza** o sistema.
- **Aumenta a integração** com outros módulos (móvel, frontend, BI, etc.).
- **Demonstra domínio técnico** de arquiteturas RESTful e boas práticas de Quarkus.

## Referências Bibliográficas

-  Quarkus REST Guide
-  Jakarta RESTful Web Services Specification
-  PostgreSQL JDBC Documentation
-  REST API Design Principles — Martin Fowler