Semana Integrada da Computação

Desenvolvimento de APIs REST com

Mateus Fuchs

INTRODUÇÃO

O que é uma API?

Uma API (Interface de Programação de Aplicativos) é um conjunto de regras e protocolos que permite que diferentes programas de software se comuniquem entre si.

Ela define como solicitações de um software devem ser feitas e como as respostas serão estruturadas.

CONFUSO?

PODEMOS PENSAR EM UMA API COMO UM CONTROLE REMOTO DE UMA TELEVISÃO

Botões (Funções da API): O controle remoto tem vários botões, cada um com uma função específica, como aumentar o volume, mudar de canal, ligar e desligar a TV. Cada botão representa uma função ou ação que você pode realizar.

Manual do Usuário (Documentação da API): O manual que acompanha o controle remoto fornece informações sobre como usar os botões e quais ações eles executam. Isso é semelhante à documentação de uma API, que explica como usar a API e quais solicitações podem ser feitas.

Pressionar um Botão (Chamada da API): Quando você pressiona um botão no controle remoto, você está fazendo uma chamada à API do controle remoto, solicitando uma ação específica, como aumentar o volume. É como fazer uma solicitação à API.

Ação na TV (Resposta da API): Quando você pressiona um botão, a TV responde executando a ação desejada, como aumentar o volume ou mudar de canal. Isso é semelhante à resposta que você recebe ao usar uma API, que pode ser informações ou uma ação executada pelo sistema.

CONFUSO?

PODEMOS PENSAR EM UMA API COMO UM CONTROLE REMOTO DE UMA TELEVISÃO

Você usa o controle remoto para interagir com a TV sem precisar conhecer todos os detalhes internos complexos do funcionamento da TV ou do controle. O controle remoto fornece uma interface simples (os botões) para acessar as funcionalidades da TV, escondendo a complexidade técnica subjacente.

Da mesma forma, em programação, as APIs fornecem uma interface simples para interagir com sistemas ou serviços complexos, ocultando os detalhes de implementação. Isso facilita o uso desses sistemas sem a necessidade de conhecimento detalhado sobre como eles funcionam internamente.

EXEMPLOS REAIS

PayPal Developer

OpenWeather

API do Google Maps: Permite que os desenvolvedores integrem recursos de mapeamento, como mapas interativos, direções e informações sobre locais, em seus aplicativos ou sites.

API do PayPal: Permite que aplicativos e sites processem pagamentos online de forma segura, facilitando transações financeiras.

API do OpenWeatherMap: Fornece dados meteorológicos em tempo real e previsões para cidades em todo o mundo, permitindo que os desenvolvedores incorporem informações meteorológicas em seus aplicativos.

O que é o REST?

As APIs **REST** (Representational State Transfer) são um estilo arquitetural para projetar sistemas de comunicação na web. Elas são baseadas em princípios simples e usam o protocolo **HTTP** (Hypertext Transfer Protocol) para permitir a interação e comunicação eficiente entre diferentes aplicativos e sistemas distribuídos na internet.

Alguns conceitos importantes sobre a arquitetura REST serão introduzidos posteriormente neste curso.

SPRING BOOT

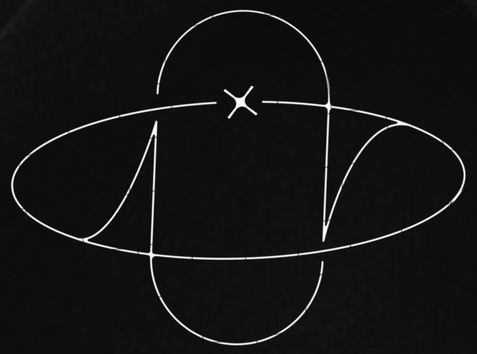
O que é o Spring boot?
O Spring Boot é uma ferramenta que ajuda os programadores a construir aplicativos de computador mais rapidamente e com menos complicações. É como ter uma caixa de ferramentas cheia de utensílios úteis que facilitam a construção de diferentes partes de um aplicativo.

Por que usaremos o Spring Boot para o desenvolvimento de APIs REST?
Usaremos o Spring Boot para desenvolver APIs REST porque ele simplifica muito o processo. Em vez de criar cada parte de uma API do zero, o Spring Boot fornece partes pré-construídas que funcionam bem juntas. Isso economiza tempo e esforço, permitindo que os programadores se concentrem mais em fazer a API funcionar da maneira certa e menos em criar as partes técnicas complicadas.

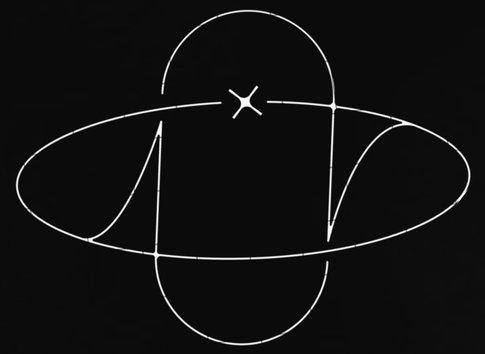
SPRING BOOT

Importante notar que o Spring Boot é especialmente utilizado no mundo da programação em Java, o que significa que é um conjunto de ferramentas específicas para desenvolvedores que trabalham com a linguagem de programação Java.

Iremos aprender as ferramentas e particularidades do Spring Boot no decorrer deste curso, principalmente na etapa do desenvolvimento da nossa própria API.



UM BREVE NIVELAMENTO



A INTERNET E A ARQUITETURA CLIENTE-SERVIDOR

A internet é uma rede global de computadores interconectados que permite a troca de informações e recursos em todo o mundo. Ela é como uma grande teia de conexões de computadores que se comunicam uns com os outros. Algumas coisas importantes sobre a internet incluem:

- **Protocolo TCP/IP:** É o conjunto de regras que os computadores usam para se comunicar na internet. Ele quebra informações em pacotes para serem enviados e recompõe esses pacotes no destino.

- **Navegador da Web:** Um programa de computador que permite que você acesse informações na internet. Exemplos incluem o Google Chrome e o Mozilla Firefox.

- **URL (Uniform Resource Locator):** Um endereço que identifica recursos na web, como sites e páginas específicas. Por exemplo, "https://www.google.com.br/" é uma URL.

A INTERNET E A ARQUITETURA CLIENTE-SERVIDOR

A arquitetura cliente-servidor é um modelo fundamental na internet. Ela descreve como os computadores se comunicam uns com os outros. Aqui estão os conceitos-chave:

- **Servidor:** Um computador ou sistema que fornece serviços, recursos ou informações a outros computadores, chamados de clientes. Um exemplo é um servidor web que hospeda sites.

- **Cliente:** Um computador ou sistema que solicita e consome serviços, recursos ou informações de um servidor. Por exemplo, um navegador da web atua como cliente ao solicitar páginas da web a um servidor web.

- **Comunicação:** Os clientes enviam solicitações aos servidores e os servidores respondem com os dados ou recursos solicitados.

- **Modelo de Requisição-Resposta:** Na arquitetura cliente-servidor, a comunicação ocorre principalmente por meio do modelo de requisição e resposta. O cliente faz uma solicitação e o servidor responde a essa solicitação com os dados ou ações apropriados.

PROTOCOLO HTTP

HTTP (Hypertext Transfer Protocol) é um protocolo de comunicação usado para transferir dados na World Wide Web (Web). Ele funciona com base no modelo de requisição e resposta entre clientes e servidores. Vamos ver os principais componentes do HTTP:

Verbos HTTP (Métodos):

- **GET**: Usado para solicitar dados de um servidor. Por exemplo, ao abrir uma página da web, seu navegador faz uma solicitação GET para obter a página.
- **POST**: Usado para enviar dados para o servidor. Por exemplo, ao enviar um formulário da web, os dados são enviados ao servidor com uma solicitação POST.
- **PUT**: Usado para atualizar dados em um servidor. Por exemplo, ao editar um perfil de usuário, as informações atualizadas são enviadas com uma solicitação PUT.
- **DELETE**: Usado para solicitar a remoção de recursos do servidor. Por exemplo, ao excluir um post em uma rede social, uma solicitação DELETE pode ser enviada

URI (Uniform Resource Identifier) é um identificador único que aponta para um recurso na web. É semelhante a um endereço. Por exemplo, "https://www.exemplo.com/recurso" é uma URI que aponta para um recurso específico em um servidor web.

PROTOCOLO HTTP

Headers HTTP:

- Os cabeçalhos HTTP são informações adicionais que são incluídas nas solicitações e respostas HTTP para fornecer detalhes sobre a transação. Alguns cabeçalhos comuns incluem:
  - **User-Agent:** Identifica o cliente, como o navegador da web ou aplicativo.
  - **Content-Type:** Indica o tipo de mídia (por exemplo, JSON, HTML) no corpo da mensagem.
  - **Authorization:** Usado para autenticação, geralmente incluindo informações de credenciais.
  - **Accept:** Especifica os tipos de mídia que o cliente aceita na resposta.
  - **Cache-Control:** Controla como os recursos devem ser armazenados em cache pelo navegador.

Corpo (Body) da Mensagem:

- O corpo da mensagem é a parte opcional de uma solicitação ou resposta HTTP onde os dados reais são transmitidos. Isso é comum em solicitações POST e PUT, onde os dados do formulário ou do payload são enviados como parte do corpo da mensagem.

PROTOCOLO HTTP

Os códigos de status HTTP são códigos numéricos de três dígitos que são retornados pelo servidor em uma resposta HTTP para indicar o resultado da solicitação feita pelo cliente. Esses códigos fornecem informações sobre o status da solicitação e geralmente são agrupados em cinco classes principais:

1xx (Informational - Informativo): Indica que a solicitação foi recebida e o servidor está continuando a processá-la.
- Exemplo: 100 (Continue) - Indica que o servidor recebeu a solicitação e o cliente pode continuar com a solicitação.

2xx (Successful - Bem-sucedido): Indica que a solicitação foi recebida, entendida e aceita com sucesso.
- Exemplo: 200 (OK) - Indica que a solicitação foi bem-sucedida e os dados solicitados estão sendo retornados.

3xx (Redirection - Redirecionamento): Indica que o cliente deve tomar uma ação adicional para completar a solicitação.
- Exemplo: 301 (Moved Permanently) - Indica que o recurso foi movido permanentemente para um novo local, e o cliente deve redirecionar suas solicitações para esse novo local.

4xx (Client Error - Erro do Cliente): Indica que a solicitação do cliente continha informações inválidas ou não pode ser cumprida pelo servidor.
- Exemplo: 404 (Not Found) - Indica que o recurso solicitado não foi encontrado no servidor.

5xx (Server Error - Erro do Servidor): Indica que ocorreu um erro no servidor ao processar a solicitação do cliente.
- Exemplo: 500 (Internal Server Error) - Indica que ocorreu um erro interno no servidor que impediu o processamento da solicitação.

JSON

O **JSON** é um formato de dados leve e amplamente utilizado para representar informações estruturadas. Ele foi originalmente derivado da notação de objetos do JavaScript, mas se tornou uma notação de dados independente e é compatível com várias linguagens de programação.

```
{
  "nome": "Maria",
  "idade": 25,
  "cidade": "Dourados",
  "telefones": [
    "12-3456-7890",
    "98-7654-3210"
  ]
}
```

E FINALMENTE, AS APIS REST

Em uma **API REST**, tudo é considerado um **recurso**. Recursos podem ser objetos, dados, serviços ou qualquer coisa que você queira expor para acesso via API. Cada recurso é identificado por um **URI** (Uniform Resource Identifier), ou também chamado de **Endpoint**.

As operações em recursos são mapeadas para **verbos HTTP**, que incluem **GET** (para recuperar dados), **POST** (para criar novos recursos), **PUT** (para atualizar recursos) e **DELETE** (para remover recursos). Esses verbos são usados para indicar a ação que você deseja realizar no recurso.

Em uma arquitetura REST, o estado do sistema é representado em um formato específico, como **JSON** ou XML. Isso significa que quando você faz uma solicitação para um recurso, a resposta contém o estado atual desse recurso.

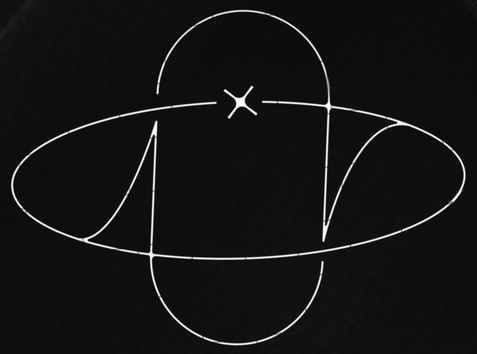
As URIs são projetadas para serem significativas e representativas dos recursos que elas identificam. Por exemplo, "/clientes/123" é uma URI que representa o cliente com o ID 123.

Exemplo de API REST: https://docs.awesomeapi.com.br/api-de-moedas

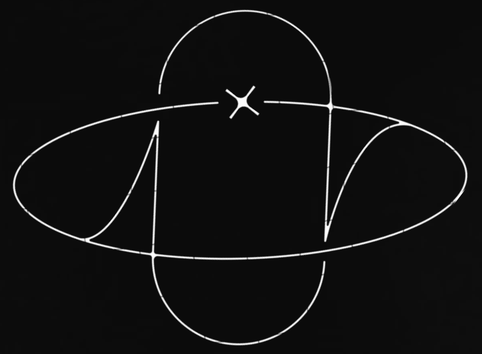
COMO TESTAR UMA API REST?

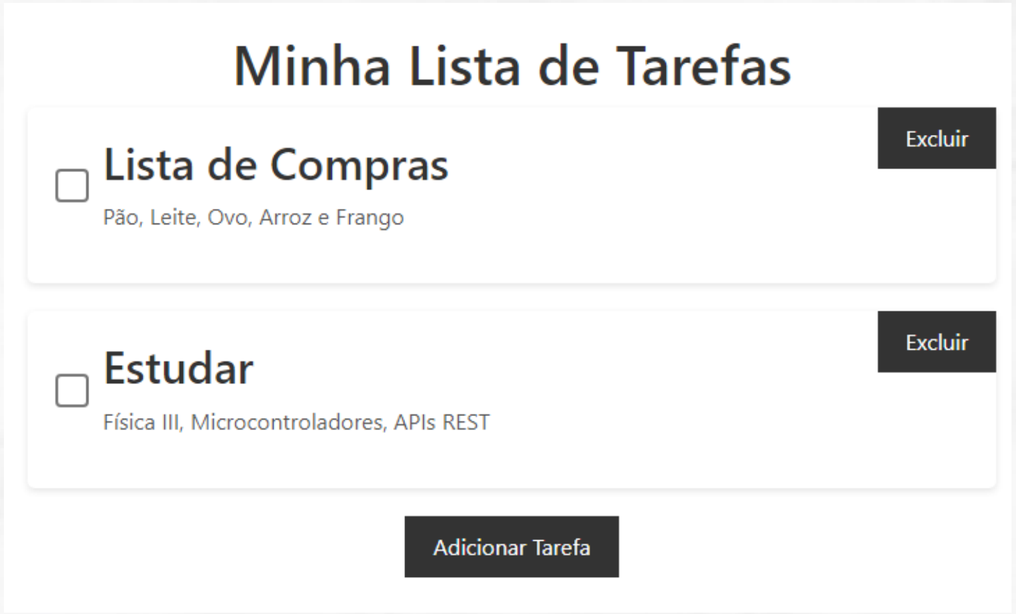
Para testar uma API REST é necessário fazer requisições HTTP para o servidor da API usando um link com o recurso desejado. Embora seja possível realizar solicitações HTTP por meio do navegador, essa abordagem apresenta desafios significativos. Os navegadores geralmente limitam a variedade de métodos HTTP disponíveis, não oferecem recursos avançados de automação de testes e dificultam a visualização e a organização de várias solicitações. Essas limitações tornam o processo de teste de APIs menos eficiente e menos flexível.

O Postman é uma ferramenta multifuncional que simplifica o teste e a interação com APIs. Com o Postman, é possível realizar uma série de tarefas, incluindo testar APIs com solicitações HTTP, automatizar testes com scripts, organizar solicitações em coleções, criar documentação interativa, simular ambientes de desenvolvimento, monitorar APIs etc. Iremos usar o Postman pa testar a API REST que criaremos neste curso.



VAMOS AO PROJETO





PROJETO

LISTA DE TAREFAS

Este projeto é um exemplo de uma API REST para gerenciamento de tarefas, desenvolvida utilizando o Spring Boot. O seu principal objetivo é permitir que os usuários criem, visualizem e excluam tarefas, proporcionando uma forma organizada de gerenciar suas responsabilidades diárias.

Devido à sua simplicidade, é uma excelente escolha para aprender os conceitos iniciais de APIs REST e familiarizar-se com o funcionamento do Spring Boot.

MÃOS À OBRA

Verifiquem se as seguintes ferramentas estão instaladas:

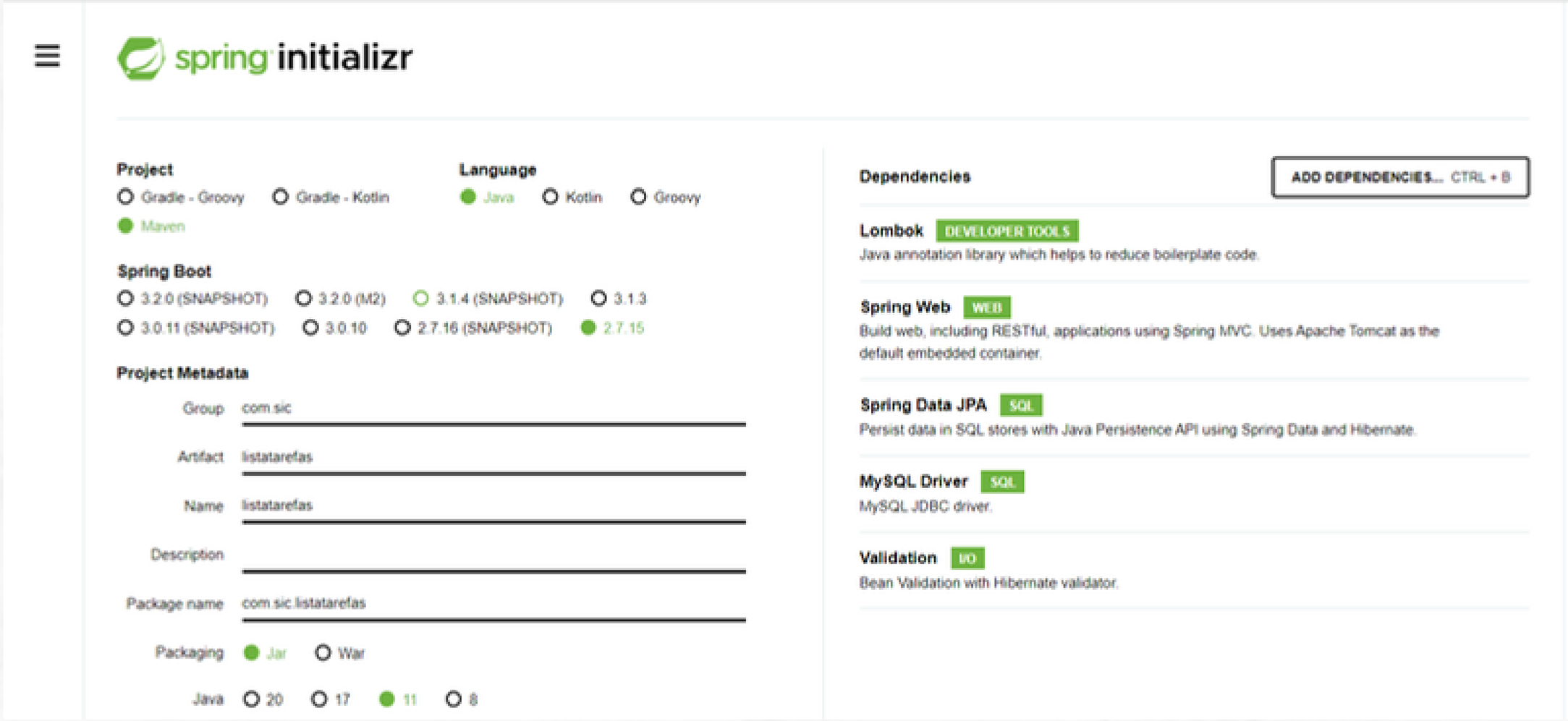
-Java 8 (digite **java -version** no prompt de comando)

-IntelliJ IDEA Community Edition

-Postman

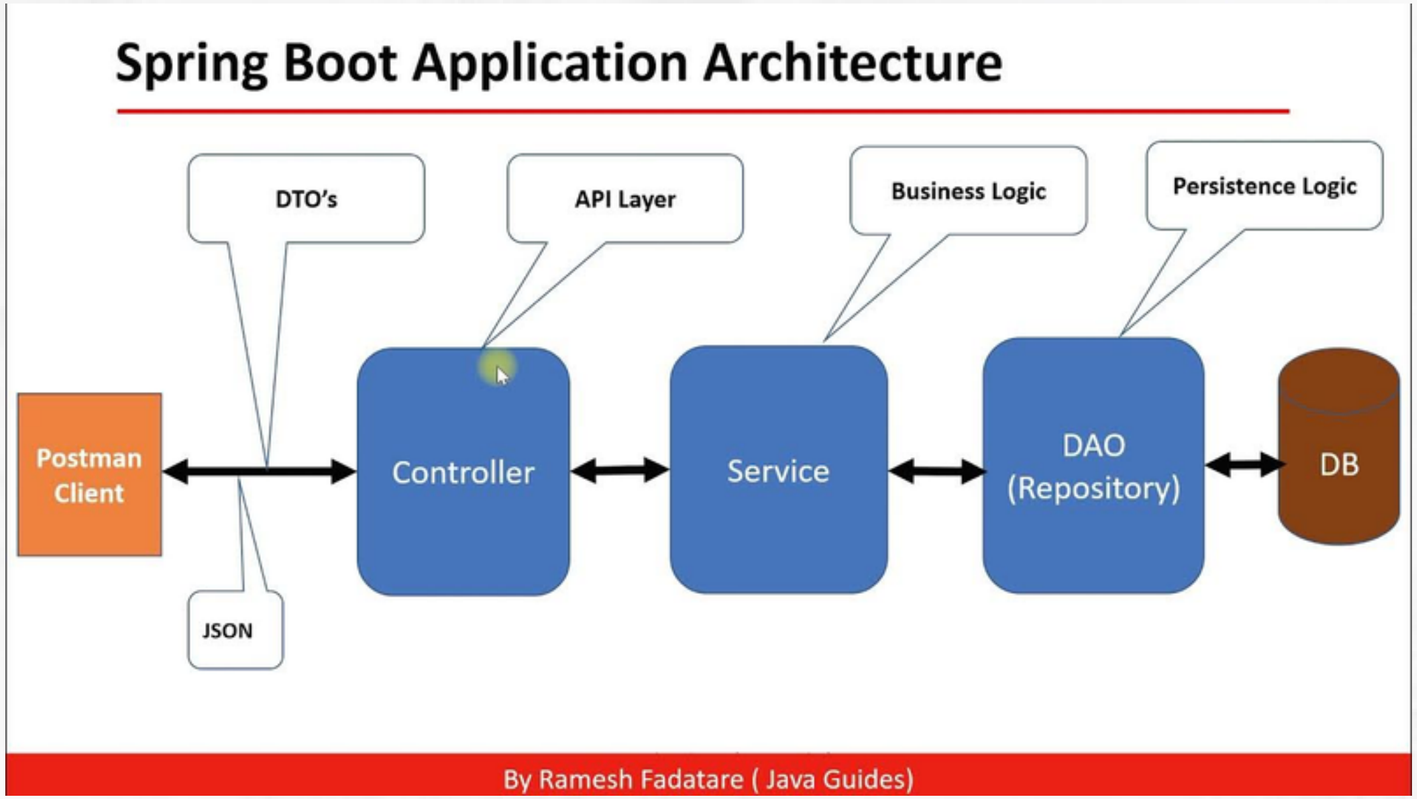
-MySQL (E crie um banco de dados para o projeto)

https://start.spring.io/



ARQUITETURA DO PROJETO

Nós dividiremos o nosso projeto em camadas, mais especificamente em quatro: Model, Repository, Service e Controller.



ARQUITETURA DO PROJETO

Model: Nesta parte, definimos como os dados são estruturados. Por exemplo, se estamos construindo uma API para gerenciar tarefas, aqui definimos o que uma "tarefa" deve conter, como título, descrição e data de conclusão. É como criar um formulário em que precisamos especificar quais informações queremos coletar.

Service: Aqui, colocamos a lógica do nosso aplicativo. Por exemplo, se alguém pedir para criar uma nova tarefa, é esta camada que verifica se todos os campos necessários estão preenchidos corretamente. É como a parte da nossa mente que toma decisões com base nas informações que temos.

Controller: Esta camada é como um gerente que lida com as solicitações dos usuários. Quando alguém faz uma solicitação à nossa API, como "mostre todas as tarefas", a camada de controle recebe essa solicitação e a direciona para o lugar certo (ou seja, o serviço). É como o atendente que encaminha você para o departamento certo em uma empresa.

Repository: Aqui é onde guardamos as informações de forma permanente, como em um banco de dados. Quando alguém cria uma nova tarefa, por exemplo, a camada de serviço pode usar esta camada para guardar essa tarefa em um local seguro. É como a prateleira onde armazenamos documentos importantes.

VAMOS CRIAR NOSSO MODEL

No mesmo pacote que a classe de Application do Spring Boot, vamos criar um pacote “model”

Em seguida iremos criar uma classe “Tarefa” que será a única entidade do projeto.

```java
package com.cursosic.cursosic.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;
import javax.validation.constraints.NotBlank;

11 usages
@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Tarefa {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @NotBlank
    private String titulo;

    @NotBlank
    @Column(columnDefinition = "TEXT")
    private String conteudo;

    private Boolean checado;
}
```

VAMOS CRIAR NOSSO REPOSITORY

No mesmo pacote que a classe de Application do Spring Boot, vamos criar um pacote “repository”

Em seguida iremos criar uma interface “TarefaRepository” .

```java
package com.cursosic.cursosic.repository;

import com.cursosic.cursosic.model.Tarefa;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

3 usages
@Repository
public interface TarefaRepository extends JpaRepository<Tarefa, Long> {
}
```

VAMOS CRIAR NOSSO SERVICE

No mesmo pacote que a classe de Application do Spring Boot, vamos criar um pacote "service"

Em seguida iremos criar uma classe "TarefaService" .

```java
package com.sic.listatarefas.service;

import com.sic.listatarefas.model.Tarefa;
import com.sic.listatarefas.repository.TarefaRepository;
import org.springframework.stereotype.Service;

import java.util.List;

no usages
@Service
public class TarefaService {

    5 usages
    final TarefaRepository repo;

    no usages
    public TarefaService(TarefaRepository repo){this.repo = repo;}

    no usages
    public List<Tarefa> getAll(){return repo.findAll();}

    no usages
    public Tarefa getById(Integer id) {return repo.findById(id).orElse( other: null);}

    no usages
    public Tarefa save(Tarefa tarefa){return repo.save(tarefa);}

    no usages
    public void deleteById(Integer id){repo.deleteById(id);}

}
```

VAMOS CRIAR NOSSO CONTROLLER

No mesmo pacote que a classe de Application do Spring Boot, vamos criar um pacote “controller”

Em seguida iremos criar uma classe “TarefaController” .

```java
@CrossOrigin()
@RestController
@RequestMapping("/")
public class TarefaController {

    5 usages
    final TarefaService service;

    no usages
    public TarefaController(TarefaService service){
        this.service = service;
    }

    no usages
    @GetMapping("/tarefas")
    public List<Tarefa> getTarefas(){
        return service.getAll();
    }

    no usages
    @GetMapping("/tarefas/{id}")
    public ResponseEntity<?> getTarefaById(@PathVariable Integer id){
        Tarefa tarefa = service.getById(id);
        if(tarefa == null){return ResponseEntity.notFound().build();}
        else{return ResponseEntity.ok(tarefa);}
    }
```

VAMOS CRIAR NOSSO CONTROLLER

No mesmo pacote que a classe de Application do Spring Boot, vamos criar um pacote “controller”

Em seguida iremos criar uma classe “TarefaController” .

```
    @PostMapping("/tarefas")
    public ResponseEntity<?> saveTarefa(@RequestBody @Valid Tarefa tarefa){
        try{
            return ResponseEntity.ok(service.save(tarefa));
        }catch(Exception e){
            return ResponseEntity.badRequest().build();
        }
    }

    no usages
    @DeleteMapping("/tarefas/{id}")
    public ResponseEntity deleteTarefa(@PathVariable Integer id){
        try {
            service.deleteById(id);
            return ResponseEntity.ok().build();
        }catch(Exception e){
            return ResponseEntity.notFound().build();
        }
    }
}
```

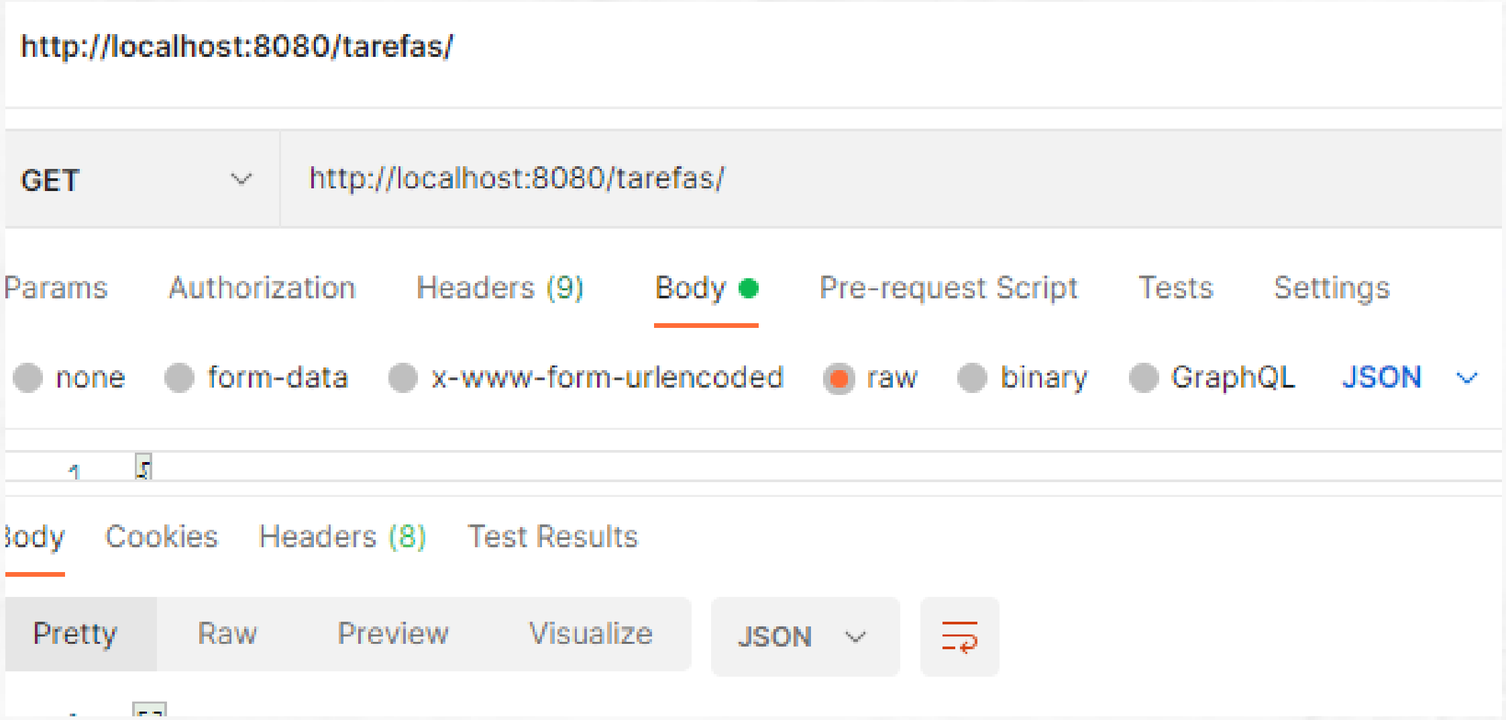
CONEXÃO COM O BD

Dentro do arquivo application.properties, iremos colocar as informações necessárias para a comunicação da nossa API com o banco de dados.

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/cursosic
spring.datasource.username=root
spring.datasource.password=admin
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql: true
server.port=8080
```

TESTE COM O POSTMAN

Nossa API está pronta! agora vamos testar suas funcionalidades com o Postman.



INTEGRAÇÃO COM UM FRONT END

Acesse https://github.com/fuchsMateus/ListaDeTarefas/ para ter acesso a um front end feito para usar com esta API.

Vamos ver na prática como a nossa API é consumida por um cliente.

RECOMENDAÇÕES PARA IMPLANTAÇÃO

Para colocar uma API REST em produção:

- Escolha um lugar confiável para hospedá-la, como na nuvem.
- Mantenha a comunicação segura usando HTTPS.
- Use automação para facilitar a implantação e atualizações.
- Fique de olho no desempenho e na disponibilidade da sua API.
- Faça cópias de segurança dos dados regularmente. Proteja sua API com login e permissões corretas.
- Mantenha tudo atualizado e documentado.
- Teste como sua API se comporta sob pressão e tenha um plano de recuperação em caso de problemas sérios.
- Esteja ciente das regras de segurança e treine sua equipe para manter tudo funcionando bem.

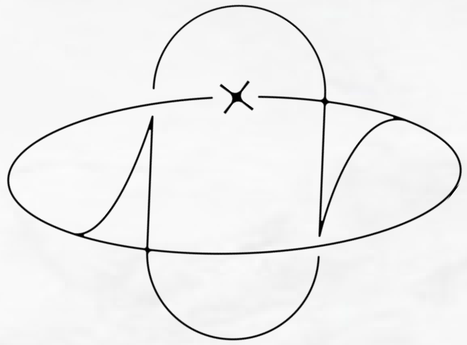
RECOMENDAÇÕES PARA IMPLANTAÇÃO

Minha recomendação é hospedar a API em algum provedor de **cloud**, como o Heroku, AWS EC2, Microsoft Azure, Oracle Cloud etc.

PRÓXIMOS PASSOS PARA INGRESSAR NESTA ÁREA

Para se aprofundar mais nesta área como um desenvolvedor Java Back End focado em APIs REST, aqui vão alguns conceitos sugeridos para serem estudados, que são extensivamente requeridos em vagas de emprego:

Testes unitários;
Swagger;
Spring Security;
Docker;
Kubernetes;
Git e Github;
Bancos de dados não relcionais;
Computação em nuvem;
Microserviços;
Scrum;
E aqui um roadmap para continuar os estudos de Spring Boot: https://roadmap.sh/spring-boot



RESUMO DO CURSO E CONSIDERAÇÕES FINAIS