

Apresentação Backend do Projeto SIGE - Rubrum

Sumário

- O que é Java Spring – pronto
- Vantagens do Spring – pronto
- Spring e APIs – pronto
- O que têm em cada pacote da nossa API – pronto
- Como se comunicar com uma api – pronto
- Exemplo de aplicação -

1- O que é Java Spring

Spring é o framework – conjunto de ferramentas que definem a estrutura do projeto – mais popular para a linguagem Java. Seu foco é fornecer uma base sólida para o desenvolvimento de uma aplicação web de forma facilitada, garantindo velocidade, segurança e flexibilidade para o projeto.

2- Vantagens do Spring

A principal vantagem de se usar Spring é reduzir o código para as funcionalidades básicas da aplicação, o que, em um projeto grande e com diversas entidades como o SIGE, no qual o gasto de tempo e esforço necessários para implementar as funções de CRUD é enorme, poupa uma grande quantidade de recursos dos programadores.

Um exemplo de recurso que fornece essa otimização é a classe *Repository*, responsável por realizar operações com o banco de dados com qualquer entidade sem a necessidade de escrever código. A classe já vem com métodos para salvar, editar, excluir e recuperar entidades no banco de dados, além de permitir que novas operações sejam adicionadas sem que seja necessário escrever código novo.

```
public class SchoolController {

    @Autowired
    private SchoolRepository repository;

    @PostMapping("/save")
    public ResponseEntity<String> save(@RequestBody @Valid SchoolRequestDTO data) {
        School school = new School(data);
        repository.save(school);

        return ResponseEntity.ok("escola criada com sucesso!");
    }
}
```

Exemplo de uso da classe Repository.

```
public interface SchoolRepository extends JpaRepository<School, String> {
    public School findByName(String name);
}
```

Exemplo de operação adicionada a classe Repository.

3- Spring e APIs

O Java Spring estrutura a aplicação em forma de *Rest API*, mas o que exatamente significa isso?

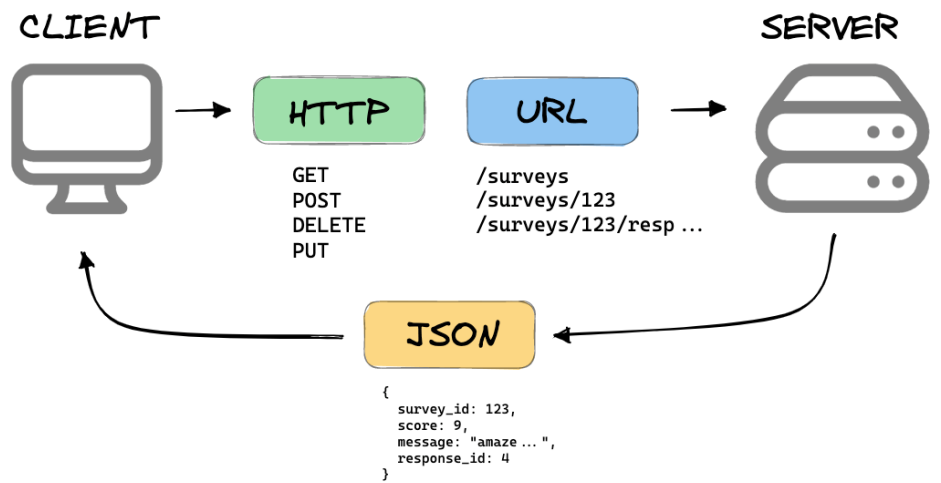
API significa *Application Programming Interface*, ou interface de programação de aplicações. As APIs são programas que conseguem se comunicar com outros, sem a necessidade de entender como esse programa foi implementado.

Uma API funciona como se fosse um contrato. Se uma parte faz um pedido estruturado de forma específica, isso determina como a outra responderá.

Como exemplo, imaginemos que você queira as informações de uma escola. Para isso, a API exige que você faça um pedido contendo o id dessa escola em um link específico (como “sigeapi.com/escolas”).

Ser uma API **Rest** significa que os pedidos feitos a essa aplicação devem ser gerenciados por Http (precisam ser feitos usando “GET” e “POST”).

WHAT IS A REST API?



mannhowie.com

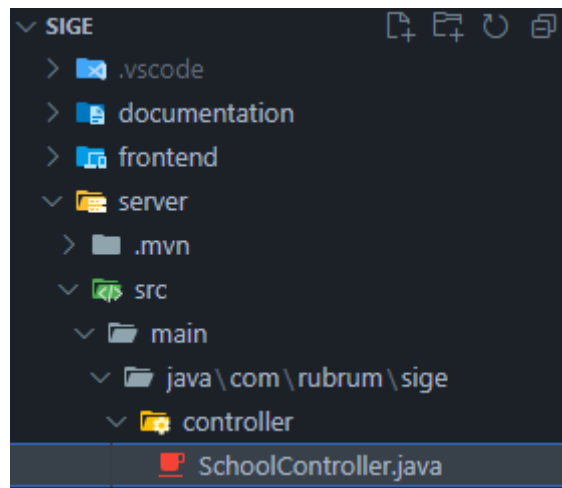
Diagrama exemplificando o uso de uma API. [1]

4- O que têm nos pacotes da nossa API

Controller

O pacote mais importante para a equipe que está desenvolvendo o frontend é, sem dúvidas, o pacote *controller*. As classes nesse pacote, separadas por entidades do banco de dados, contêm todos os pedidos que podem ser feitos à API.

Como exemplo, veremos a classe “*SchoolController*”, responsável por gerenciar os pedidos relacionados as escolas.



Local da classe “SchoolController”

```
@Validated
@RestController
@RequestMapping("school")
public class SchoolController {

    @Autowired
    private SchoolRepository repository;

    @PostMapping("/save")
    public ResponseEntity<String> save(@RequestBody @Valid SchoolRequestDTO data) {
        School school = new School(data);
        repository.save(school);
        return ResponseEntity.ok(body:"escola criada com sucesso!");
    }


    @GetMapping("/{name}")
    public ResponseEntity<SchoolResponseDTO> getByName(@PathVariable String name) throws BadRequestException {
        School school = repository.findByName(name);
        if (school == null) {
            throw new BadRequestException("nome não encontrado.");
        }
        return ResponseEntity.ok(new SchoolResponseDTO(school));
    }
}
```

Parte do código da classe “SchoolController”. [Código na Íntegra](#)

Coisas importantes para se reparar no arquivo:

- O “*RequestMapping*” logo acima da assinatura da classe indica o link para acessar seus métodos. Nesta classe, supondo que a API está no *localhost:8080*, o link ficaria algo do tipo “*localhost:8080/school*”.
- O “*PostMapping*” ou “*GetMapping*” acima da assinatura do método indica o link para acessar aquele método. No *save*, por exemplo, o link ficaria algo do tipo “*localhost:8080/school/save*”.
- Em métodos em que o link contém um valor entre chaves – como no método *getByName* que possui o valor *{name}* - você deve substituir esse valor pelo atributo requerido. Supondo que você queira requisitar as informações da escola “*escolaexemplo*”, o link ficaria algo do tipo “*localhost:8080/school/escolaexemplo*”.

É muito importante que você repare nos métodos que recebem um objeto de parâmetro assim como o método *save*. Para usar esse tipo de método, você deve fazer um pedido a API fornecendo um objeto com os valores iguais ao do objeto parâmetro.



```
public record SchoolRequestDTO(String name, String palette) {}
```

Objeto parâmetro do método *save*, contendo o nome e a paleta de cores de uma escola.

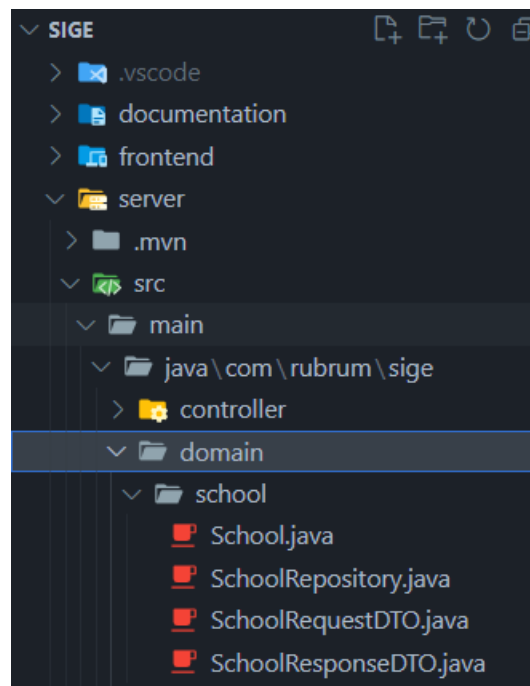
```
{  
  "name": "escolaexemplo",  
  "palette": "vermelho, azul, verde"  
}
```

Exemplo de objeto fornecido para a API do SIGE.

Alguns métodos retornam um objeto ao frontend assim como o *getByName*. A resposta da API será igual a um JSON da classe retornada.

Domain

O pacote *domain* contém as classes que normalmente estariam no *model* do padrão mvc. Nele estão as classes que se comunicam com o banco de dados, as classes que representam entidades do banco de dados e as classes que representam pedidos e respostas que a API pode receber ou responder.



Local do pacote domain.

As classes “...Repository” são as responsáveis por se comunicar com o banco de dados, lembrando que os métodos do CRUD não estarão escritos diretamente no arquivo, já que são herdados de uma classe do Spring.

As classes “...RequestDTO” representam um pedido que a API pode receber, enquanto as “...ResponseDTO” são as respostas. DTO significa *Data Transfer Object* (ou objeto de transferência de dados), e é usado nos objetos que servem para transferir dados do cliente para API e vice-versa.

As classes com o nome de uma entidade (neste caso a “School.java”) representam sua respectiva tabela do banco de dados.

Esses são os dois principais pacotes e os únicos que você precisa entender para conseguir usar a API. Não é necessário que decore o que cada arquivo faz, mas têm que se lembrar de como e onde procurar o que você precisa enviar à API para receber o que precisa.

5- Como Se Comunicar Com a API

A forma mais simples para se comunicar com a API e fazer um pedido (requisição Http) a ela é com o Javascript, utilizando o **fetch**. Ele é uma interface do Javascript para fazer requisições Http de maneira nativa, sem precisar importar nada no projeto.

O fetch é usado como uma função que recebe um URL e uma **estrutura**.

```
fetch("localhost:8080/escola", minhaEstrutura);
```

Exemplo de uso do fetch.

Essa estrutura do fetch é onde estão os dados que a API requer e informações sobre o que o pedido contém. Essa estrutura é separada em três principais partes: o método, o cabeçalho e o corpo.

- O método contém a operação desejada (GET ou POST, por exemplo);
- O cabeçalho normalmente contém informações sobre a requisição, como que tipo de dados ela está enviando à API e que tipo de dado você aceita como resposta.

```
const cabecalho = {  
  "Content-type": "application/json; charset=UTF-8",  
  "Accept": "application/json"  
}
```

- No corpo estão os dados que você pretende enviar à API.

```
const corpo = {  
  "name": "escolaExemplo",  
  "palette": "vermelho, azul, verde"  
}
```

```
1  
2 const cabecalho = {  
3   "Content-type": "application/json; charset=UTF-8",  
4   "Accept": "application/json"  
5 }  
6  
7 const corpo = {  
8   "name": "escolaExemplo",  
9   "palette": "vermelho, azul, verde"  
10 }  
11  
12 const minhaEstrutura = {  
13   "method": "POST", // Método  
14   "headers": cabecalho, // Cabeçalho  
15   "body": corpo // Corpo  
16 }  
17  
18 const escola = fetch("localhost:8080/escola/save", minhaEstrutura);
```

Exemplo completo de fetch à API do SIGE.

Existe, porém, um pequeno detalhe de quando se vai fazer um fetch. Como você está se comunicando com uma aplicação externa ao frontend, o código fica descompassado,

uma vez que o Javascript não sabe que precisa **esperar** a API terminar de executar seu método e responder. É como se o frontend pedisse uma caneta ao backend e tentasse começar a escrever enquanto o ele ainda está tirando a caneta da bolsa.

Para que o Javascript entenda que precisa esperar o resultado do fetch antes de prosseguir com o código, é necessário colocar “**await**” antes do fetch.

Já para lidar com a informação recebida da API, adicione um “**.then(função)**” depois do fetch. A função usada de parâmetro no *then* recebe a resposta da API como argumento.

```
1
2  const cabecalho = {
3    "Content-type": "application/json; charset=UTF-8",
4    "Accept": "application/json"
5  }
6
7  const corpo = {
8    "name": "escolaExemplo",
9    "palette": "vermelho, azul, verde"
10 }
11
12 const minhaEstrutura = {
13   "method": "POST", // Método
14   "headers": cabecalho, // Cabeçalho
15   "body": corpo // Corpo
16 }
17
18 const resposta = function(resp) {
19   return resp.message; // escola criada com sucesso!
20 }
21
22 const escola = await fetch("localhost:8080/escola/save", minhaEstrutura).then(resposta);
```

Exemplo de fetch com await e then.

```
public class SchoolController {

    @Autowired
    private SchoolRepository repository;

    @PostMapping("/save")
    public ResponseEntity<String> save(@RequestBody @Valid SchoolRequestDTO data) {
        School school = new School(data);
        repository.save(school);

        return ResponseEntity.ok("escola criada com sucesso!");
    }
}
```


Fontes

- <https://spring.io/why-spring>
- <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>
- <https://mannhowie.com/rest-api> [1]
- https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API/Using_Fetch