

CETEJ35 - Java Web - JAVA_XXX (2024_01)

[Meus cursos](#) / [CETEJ35 - Web \(2024_01\)](#) / [Semana 07: 14/10 a 20/10](#) / [API Reativa](#)

API Reativa

✓ **Feito:** Ver

✓ **Feito:** Gastar pelo menos 20 minutos na atividade

A fazer: Passar pela atividade até o fim

Fecha: segunda-feira, 2 dez. 2024, 00:00

Nesta aula vamos construir uma nova aplicação que funciona como uma API, recebendo dados usando a arquitetura REST. Para fazer isso, vamos usar uma nova tecnologia do Spring - o Spring WebFlux.

CONTROLADOR

A última parte desse projeto é o controlador. Seguindo o modelo MVC, o controlador tem a responsabilidade de gerenciar a interação entre o usuário e a API. Nesse projeto, nosso controlador age como um *delegador* de solicitações. Dessa forma, os métodos do controlador apenas expõem os endereços da API e repassam as solicitações de e para a interface de persistência.

Usar controladores para delegar solicitações é um padrão muito comum em aplicações CRUD. Mas, você sabia que tem uma forma mais simples de fazer isso? O **Spring Data REST** é um projeto do Spring Data que permite criar APIs diretamente da interface do repositório. Desse modo, não é necessário criar um controlador para delegar solicitações.



O código na figura a seguir mostra a classe do controlador, **TodoRestController**, na sua forma final.

```
14 @RestController
15 public class TodoRestController {
16
17     private final TodoRepository repository;
18
19     public TodoRestController(final TodoRepository repository) {
20         this.repository = repository;
21     }
22
23     @GetMapping("/todos")
24     public Flux<Todo> lerTodos() {
25         return repository.findAll();
26     }
27
28     @GetMapping("/todos/{feito}")
29     public Flux<Todo> lerByFeito(@PathVariable boolean feito) {
30         return repository.findByFeito(feito);
31     }
32
33     @PostMapping("/todos")
34     public Mono<Todo> criar(@RequestBody Todo todo) {
35         return repository.save(todo);
36     }
37
38     @DeleteMapping("/todos/{id}")
39     public Mono<Void> deletar(@PathVariable String id) {
40         return repository.deleteById(id);
41     }
42
43     @PutMapping("/todos/{id}")
44     public Mono<Todo> atualizar(@PathVariable String id) {
45
46         return repository
47             .findById(id)
48             .map(todoAtual -> new Todo(id,
49                                     todoAtual.titulo(),
50                                     todoAtual.descricao(),
51                                     !todoAtual.feito()))
52             .flatMap(repository::save)
53             .onTerminateDetach();
54     }
55 }
56 }
```



A classe `TodoRestController` tem cinco métodos. Na linha 24, o método `lerTodos()` é responsável por listar as *tarefas* existentes na base de dados. Observe o tipo de retorno - é o mesmo retorno que usamos no `TodoRepository`. Observe que o corpo do método apenas repassa a solicitação para uma instância de `TodoRepository`, responsável por acessar a base de dados. O resultado de `TodoRepository.findAll()` é retornado diretamente para o solicitante.

O segundo método, na linha 29, é uma variação do primeiro. Enquanto `lerTodos()` retorna todas as *tarefas* na base de dados, `lerByFeito(boolean)` retorna apenas as tarefas que tiverem o atributo `feito` com o valor definido no parâmetro do método - *true* ou *false*. O corpo do método `lerByFeito(boolean)` apenas repassa a solicitação, chamando o método `TodoRepository.findByFeito(boolean)`, que criamos anteriormente.

O terceiro método, na linha 34, é responsável por salvar uma tarefa na base de dados. Note que o método `criar(Todo)` tem um retorno diferente dos dois métodos anteriores. Enquanto os métodos anteriores poderiam retornar mais de uma *tarefa*, o método `criar(Todo)` retorna apenas uma *tarefa*. O tipo `reactor.core.publisher.Mono` funciona da mesma forma que o `reactor.core.publisher.Flux`. Porém, o `reactor.core.publisher.Mono` suporta carregar apenas um objeto. Nesse caso, o retorno corresponde à *tarefa* que foi criada.

Os dois métodos seguintes, `criar(Todo)` (linha 34) e `deletar(String)` (linha 39), funcionam exatamente como os métodos anteriores - repassando a solicitação para o método correspondente no `TodoRepository`. Contudo, o método `criar(Todo)` recebe um objeto do tipo `Todo` como parâmetro - representando a *tarefa* que será criada. Já o método `deletar(String)` recebe uma `String` como parâmetro - representando o identificador da *tarefa* que será removida. Outra diferença é que o método `deletar(String)` não retorna uma *tarefa*, mas sim um retorno vazio - representado pelo objeto `Void`. Isso faz sentido porque a *tarefa* acabou de ser eliminada e, portanto, não existem dados para retornar.

Você pode **personalizar o tipo de retorno** se preferir, por exemplo, adicionando mais informações sobre a solicitação.

O último método, **atualizar(String)** (linha 44), alterna o estado da **tarefa** entre *feito* e *não feito*. O corpo do método recupera a **tarefa** correspondente ao identificador informado (linha 47). Em seguida, uma nova **tarefa (Todo)** é criada com os mesmos valores da tarefa existente (linhas 48 a 51). Porém, o valor do atributo **feito** é alternado (linha 51). Depois, a tarefa alterada é salva (linha 52) e os assinantes (*subscribers*) dessa operação são liberados (linha 53).

A classe tem ainda um construtor que recebe **TodoRepository** como argumento. Esse construtor é usado pelo Spring para injetar a dependência no atributo **repository**. Assim, como vimos anteriormente, a injeção de dependência é uma característica marcante do Spring. Note que não precisamos instanciar **TodoRepository** em nenhum momento.

Nesse ponto, você deve estar se perguntando: "E quanto às anotações?". Sem as anotações, nosso código é apenas mais um código Java. São as anotações que fazem o framework entender que esse código deve ser gerenciado pelo Spring. As anotações usadas aqui **não são** diferentes das anotações usadas pelo Spring MVC. Por isso, esse mesmo controlador poderia ser usado tanto para uma API com Spring MVC quanto para uma API com WebFlux. O detalhe, é claro, é o tipo de retorno dos métodos que precisaria ser adequado.

Agora, vamos entender o papel de cada anotação nessa classe:

- **org.springframework.web.bind.annotation.RestController** (linha 14) identifica essa classe como um controlador que atende solicitações do tipo REST.
- **org.springframework.web.bind.annotation.GetMapping** (linhas 23 e 28) associa o método subsequente com uma solicitação **HTTP** do tipo **GET** para a URL informada como parâmetro **na anotação**. Observe que na linha 28, a URL tem também um parâmetro variável (**feito**), que deve ser informado como parte da URL na solicitação.
- **org.springframework.web.bind.annotation.PostMapping** (linha 33), **org.springframework.web.bind.annotation.DeleteMapping** (linha 38) e **org.springframework.web.bind.annotation.PutMapping** (linha 43) associam o método subsequente com uma solicitação **HTTP** do tipo **POST**, **DELETE** e **PUT**, respectivamente, para as URLs informadas como parâmetro nas anotações.
- **org.springframework.web.bind.annotation.PathVariable** (linhas 29, 39 e 44) associa o parâmetro da URL com o parâmetro do método atual.
- **org.springframework.web.bind.annotation.RequestBody** (linha 34) associa um objeto **JSON** enviado no corpo da solicitação REST com o objeto informado no parâmetro do método atual (**Todo**).

O código desenvolvido nesta Seção está disponível no **Github**, na branch **semana06-40-controlador**.

[Retroceder](#)[Avançar](#)[◀ Verificação de aprendizado - Integração](#)[Atividade II WebConf ▶](#)[✉ Contate o suporte do site](#)


Você acessou como RAFAEL ROCHA DA SILVA PROENÇA (Sair)
CETEJ35 - Web (2024_01)

Tema
Adaptable
Boost
Clássico
Campus
Apucarana

- Campo Mourão
- Cornélio Procopio
- Curitiba
- Dois Vizinhos
- Francisco Beltrão
- Guarapuava
- Londrina
- Medianeira
- Pato Branco
- Ponta Grossa
- Reitoria
- Santa Helena
- Toledo
- UTFPR
- Ajuda
- Chat UTFPR
- Calendário Acadêmico
- Biblioteca
- e-Mail
- Nuvem (OwnCloud)
- Produção Acadêmica
- Secretaria Acadêmica
- Sistemas Corporativos
- Sistema Eletrônico de Informação - SEI
- Suporte ao usuário
- Criação de curso
- Comunidade
- Português - Brasil (pt_br)
- Deutsch (de)
- English (en)
- Português - Brasil (pt_br)

Resumo de retenção de dados

Baixar o aplicativo móvel.

 Dê um feedback sobre este software 



Universidade Tecnológica Federal do Paraná - UTFPR
Suporte ao usuário