

Teste - Desenvolver uma api rest utilizando Java Spring Boot e MongoDB, o cenário é livre.

1 - Qual a diferença entre Spring e Spring Boot ?

Spring Framework

Spring é um dos mais usados Java EE Frameworks para criação de aplicativos. Para a plataforma Java, o framework Spring fornece um modelo elaborado de programação e configuração. O objetivo é simplificar o desenvolvimento do Java EE e ajuda os desenvolvedores a serem mais produtivos no trabalho. Pode ser usado em qualquer tipo de plataforma de implantação. Leva em consideração as crescentes necessidades das empresas atuais e se esforça para atendê-las.

Ao contrário de outros frameworks, o Spring se concentra em várias áreas de um aplicativo e fornece uma ampla gama de recursos.

Um dos principais recursos do framework Spring é a injeção de dependência. Isso ajuda a simplificar as coisas, permitindo-nos desenvolver aplicativos fracamente acoplados.

Spring Boot

Embora o framework Spring se concentre em fornecer flexibilidade para você, o Spring Boot visa encurtar o tamanho do código e fornecer a maneira mais fácil de desenvolver um aplicativo da web. Com a configuração de anotação e os códigos padrão, o Spring Boot reduz o tempo envolvido no desenvolvimento de um aplicativo. Isso ajuda a criar um aplicativo independente com configuração menor ou quase zero.

Como sabemos, todo framework se baseia em alguns princípios. No caso do Boot, são quatro:

1. Prover uma experiência de início de projeto (getting started experience) extremamente rápida e direta;
2. Apresentar uma visão bastante opinativa (opinionated) sobre o modo como devemos configurar nossos projetos Spring mas, ao mesmo tempo, flexível o suficiente para que possa ser facilmente substituída de acordo com os requisitos do projeto;
3. Fornecer uma série de requisitos não funcionais já pré-configurados para o desenvolvedor como, por exemplo, métricas, segurança, acesso a base de dados, servidor de aplicações/servlet embarcado, etc.;
4. Não prover nenhuma geração de código e minimizar a zero a necessidade de arquivos XML.
- 5.

2 - Quais as vantagens de utilizar o Spring Boot ?

As vantagens adicionais que vêm com o Spring Boot são de grande valor para os desenvolvedores, pois oferecem a conclusão de projetos com esforços mínimos. Para todos os problemas que surgem do framework Spring, o Spring Boot é a solução.

- O Spring Boot não exige a implantação de arquivos WAR.
- Cria aplicativos independentes.
- Ajuda a incorporar o Tomcat, Jetty ou Undertow diretamente.
- Não requer configuração XML.
- Tem como objetivo reduzir o LOC.
- Oferece recursos prontos para produção.
- É mais fácil de lançar.
- Personalização e gerenciamento mais fáceis.

3 - Como escrever testes de integração ?

Descrição: É a forma de se testar a combinação das unidades em conjunto.

Objetivo: Nesse caso, a ideia é encontrar falhas na junção destas unidades. Pode ser que a classe X funcione bem sozinha, mas ao ser utilizada pela classe Y, ela deixe de funcionar.

Exemplo: Colocar todo o software para rodar e começar a usar diversas funcionalidades consideradas centrais no seu programa para confirmar que ele roda e as principais funcionalidades fazem o esperado.

Benefícios: além de testar funcionalmente, pode assegurar performance e confiabilidade. Ajudam a garantir que o trabalho de um desenvolvedor não está afetando o trabalho de outro e em equipes grandes isso pode fazer toda a diferença se forem realizados com frequência.

Teste Integrado (ou "teste de integração"): você tem as seguintes funções (novamente simplistas):

def soma(a, b):

```
    return a + b

def multiplicacao(a, b):
    return a * b

def minhaFuncaoNadaVer(a, b):
    return soma(
        multiplicacao(a, b),
        multiplicacao(a + b, b)
    )
```

Após você ter feito Testes Unitários para as funções soma e multiplicação, agora você vai fazer um Teste de Integração da função minhaFuncaoNadaVer, afinal ela integra duas funções do seu sistema (soma e multiplicacao).

Em Python:

```
assert minhaFuncaoNadaVer(12, 6) == 180
assert minhaFuncaoNadaVer(1, 90) == 8280
```

4 - Desenvolver uma api rest utilizando Java Spring Boot e MongoDB, o cenário é livre.

5 - As ferramentas deste projeto devem ser as seguintes:

- Java
- Spring Boot
- Undertow como webserver
- Gradle
- MongoDB

Links que podem ajudar no processo de desenvolvimento:

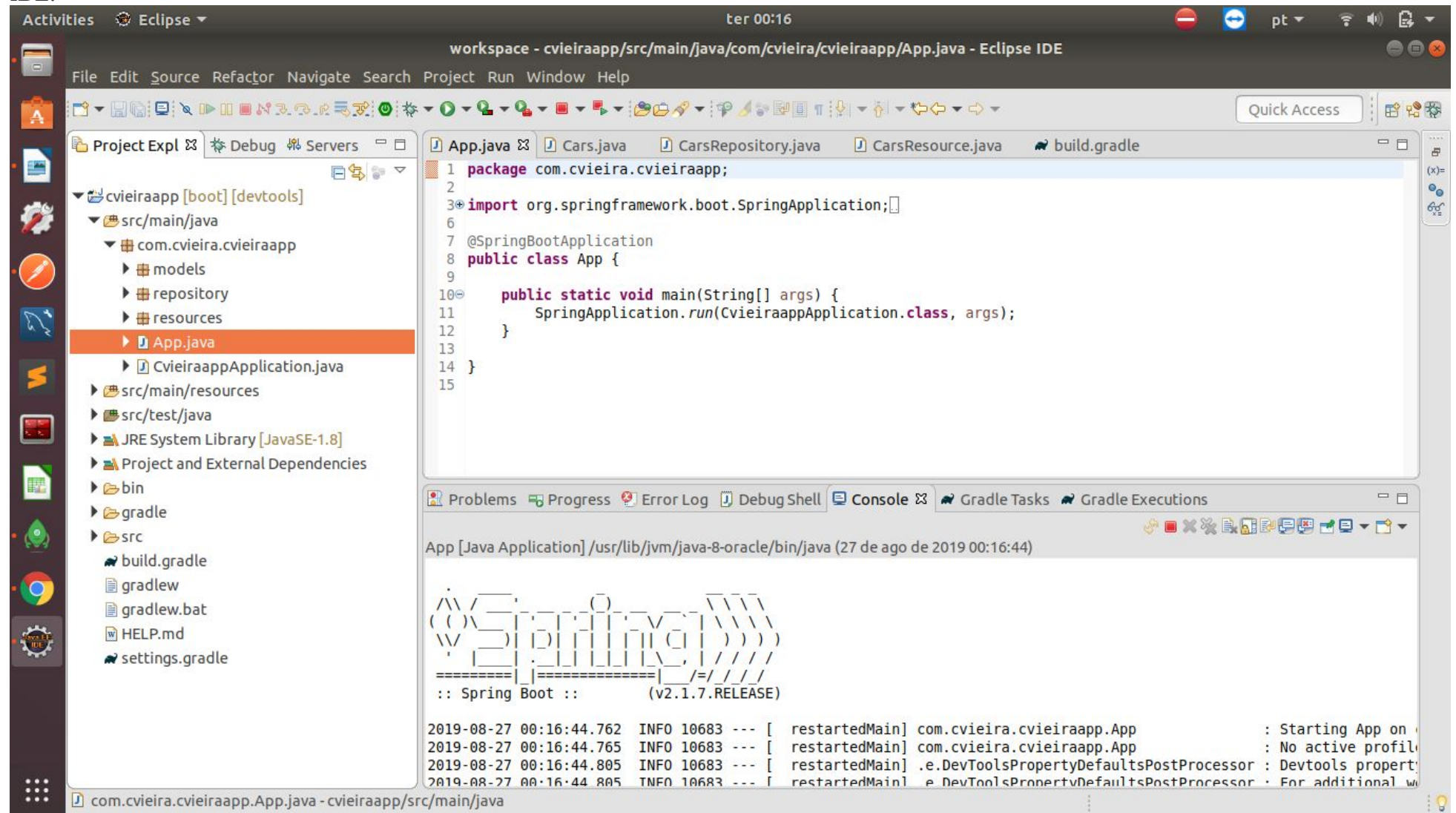
- <https://www.mongodb.com/cloud/atlas>
- <https://www.mongodb.com/products/compass>
- <https://www.mockable.io>
- <http://ptsv2.com/>
- <https://www.getpostman.com/>

Ferramentas Utilizadas

- 1 – Ambiente Linux Ubuntu 18.04
- 2 – IDE Eclipse Version: 2019-06 (4.12.0) Build id: 20190614-1200
- 3 - Versão Java 1.8
- 4 - Spring framework - Spring boot version '2.1.7.RELEASE'
 - a - Spring Boot Data MongoDB Starter
 - b - Spring Boot Data REST Starter
 - c - Spring Boot Undertow Starter
 - d - Spring Boot Web Starter
- 5 - Gradle - Sistema avançado de automatização de builds que une o melhor da flexibilidade do Ant com o gerenciamento de dependências e as convenções do Maven.
- 6 – MongoDB - Bancos de dados NoSQL - orientado a documentos livre, de código aberto e multiplataforma, escrito na linguagem C++.
- 7 - Robo 3T 1.3.1 Shell-centric MongoDB management tool.
- 8 - Postman helps you develop APIs faster Postman for Linux Version 6.7.1 linux 4.15.0-58-generic / x64
- 8 - Git

REST api's – Informações sobre o Veículo
CRUD(Create,Read,Update and Delete)

IDE:



Postman:

The screenshot displays the Postman application interface. The top bar shows the time as 00:23 and the status as 'pt'. The main workspace is titled 'My Workspace' and 'Invite'. The left sidebar shows a 'Collections' view with a 'Trash' section and a 'Spring Boot Mongo...' collection containing 6 requests. The main workspace shows a GET request to 'localhost:8080/cars' with a status of '200 OK', time of '27 ms', and size of '741 B'. The response body is displayed in 'Pretty' format, showing a JSON array of car data.

Request Details:

- Method: GET
- URL: localhost:8080/cars
- Status: 200 OK
- Time: 27 ms
- Size: 741 B

Response Body (JSON):

```
[{"id": "5d64918e827a0b61fdc0bfce", "marca": "Asia Motors", "modelo": "Jipe Rocsta GT 4x4 2.2 Diesel", "combustivel": "Diesel", "ano": "1994", "valor": 18578}, {"id": "5d649214827a0b61fdc0bfcf", "marca": "VW - Volkswagen", "modelo": "Gol (novo) 1.0 Mi Total Flex 8V 2p", "combustivel": "Gasolina", "ano": "2014", "valor": 20812}, {"id": "5d6497ce827a0b10a2df48d7", "marca": "Renault", "modelo": "DUSTER 1.6 Hi-Flex 16V Mec", "combustivel": "Gasolina", "ano": "2014", "valor": 20812}]
```

Robo 3T:

The screenshot shows the Robo 3T application window. The top bar indicates the time is 00:26 and the application version is 1.3. The left sidebar shows a tree view of the database structure, with 'cvieira01' selected, containing 'Collections (1)' and 'Indexes (1)'. The 'cars' collection is highlighted. The main window displays a query: `db.getCollection('cars').find({})`. The results are shown in a table with 2 columns and 2 rows. The first row represents a car with ID '5d64918e827a0b61fdc0bfce', brand 'Asia Motors', model 'Jipe Rocsta GT 4x4 2.2 Diesel', fuel 'Diesel', year '1994', and value '18578.0'. The second row represents a car with ID '5d649214827a0b61fdc0bfce', brand 'VW - Volkswagen', model 'Gol (novo) 1.0 Mi Total Flex 8V 2p', fuel 'Gasolina', year '2014', and value '20812.0'. Both cars are of class 'com.cvieira.cvieiraapp.models.Cars'.

Activities Robo3t ter 00:26
Robo 3T - 1.3

File View Options Window Help

cvieira01 (1)
cvieira01
Collections (1)
cars
Indexes (1)
id
Functions
Users

db.getCollection('cars').find({})

cars 0.032 sec.

```
/* 1 */
{
  "_id" : ObjectId("5d64918e827a0b61fdc0bfce"),
  "marca" : "Asia Motors",
  "modelo" : "Jipe Rocsta GT 4x4 2.2 Diesel",
  "combustivel" : "Diesel",
  "ano" : "1994",
  "valor" : 18578.0,
  "_class" : "com.cvieira.cvieiraapp.models.Cars"
}

/* 2 */
{
  "_id" : ObjectId("5d649214827a0b61fdc0bfce"),
  "marca" : "VW - Volkswagen",
  "modelo" : "Gol (novo) 1.0 Mi Total Flex 8V 2p",
  "combustivel" : "Gasolina",
  "ano" : "2014",
  "valor" : 20812.0,
  "_class" : "com.cvieira.cvieiraapp.models.Cars"
}

/* 3 */
{
```

Logs

EXECUÇÃO DE TESTE: Read

The screenshot shows the Postman application interface. The top bar indicates the time is 09:11 and the user is logged in as 'pt'. The main workspace is titled 'My Workspace' and shows a collection of requests. The selected request is a GET request to 'localhost:8080/cars'. The response is displayed in the 'Body' tab, showing a JSON array of three car objects.

Request Details:

- Method: GET
- URL: localhost:8080/cars
- Environment: No Environment

Response Details:

- Status: 200 OK
- Time: 390 ms
- Size: 741 B

Response Body (JSON):

```
[{"id": "5d64918e827a0b61fdc0bfce", "marca": "Asia Motors", "modelo": "Jipe Rocsta GT 4x4 2.2 Diesel", "combustivel": "Diesel", "ano": "1994", "valor": 18578}, {"id": "5d649214827a0b61fdc0bfcf", "marca": "VW - Volkswagen", "modelo": "Gol (novo) 1.0 Mi Total Flex 8V 2p", "combustivel": "Gasolina", "ano": "2014", "valor": 20812}, {"id": "5d6497ce827a0b10a2df48d7", "marca": "Renault", "modelo": "DUSTER 1.6 Hi-Flex 16V Mec", "combustivel": "Gasolina", "ano": "2014", "valor": 20812}]
```


EXECUÇÃO DE TESTE: Create

The screenshot displays the Postman application window. The top bar shows the system clock at 09:16 and the application name 'Postman'. The main interface is divided into several sections:

- Left Sidebar:** Contains a 'Filter' search bar, 'History' and 'Collections' tabs, and a list of collections. The 'Spring Boot Mongo...' collection is expanded, showing a list of requests including GET, DEL, PUT, and POST requests to localhost:8080/cars.
- Top Bar:** Includes 'File', 'Edit', 'View', and 'Help' menus, along with buttons for 'New', 'Import', 'Runner', and 'My Workspace'.
- Request Editor:** The 'POST' method is selected for the endpoint 'localhost:8080/cars'. The 'Body' tab is active, showing a JSON payload:

```
{  "marca": "Honda",  "modelo": "Accord Sedan EX 2.0 16V 156cv Aut",  "combustivel": "Gasolina",  "ano": "2012",  "valor": "50235.00"}
```
- Response Section:** The status is '200 OK', with a time of '87 ms' and a size of '278 B'. The response body is displayed in 'Pretty' JSON format:

```
{  "id": "5d651f0ed6cbe64b327e04f1",  "marca": "Honda",  "modelo": "Accord Sedan EX 2.0 16V 156cv Aut",  "combustivel": "Gasolina",  "ano": "2012",  "valor": 50235}
```

Activities Robo3t

ter 09:45

Robo 3T - 1.3

File View Options Window Help

cvieira01 (1)

- cvieira01
 - Collections (1)
 - cars**
 - Functions
 - Users

* db.getCollection('cars'... x

cvieira01 mongodb.cvieira.com.br:27017 cvieira01

db.getCollection('cars').find({'marca': 'Honda'})

cars 0.028 sec.

```
/* 1 */
{
  "_id" : ObjectId("5d651f0ed6cbe64b327e04f1"),
  "marca" : "Honda",
  "modelo" : "Accord Sedan EX 2.0 16V 156cv Aut",
  "combustivel" : "Gasolina",
  "ano" : "2012",
  "valor" : 50235.0,
  "_class" : "com.cvieira.cvieiraapp.models.Cars"
}
```

Logs

Activities

Postman

ter 09:56

Postman

File Edit View Help

New

Import

Runner

My Workspace

Invite

Upgrade

Filter

History

Collections

Trash

Spring Boot Mongo...
6 requests

GET localhost:8080/

GET localhost:8080/cars

DEL localhost:8080/cars/5d64...

PUT localhost:8080/cars/5d64...

GET localhost:8080/cars/5d64...

POST localhost:8080/cars

Api Kevin

0 requests

This collection is empty.
Add requests to this collection and create folders to organize them

GET localhost:8080

GET localhost:8080

POST localhost:8080

GET localhost:8080

PUT localhost:8080

DEL localhost:8080

+

...

No Environment

localhost:8080/cars/5d64936d827a0b61fdc0bfd0

Examples (0)

GET

localhost:8080/cars/5d651f0ed6cbe64b327e04f1

Send

Save

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

Cookies

Code

Comments (0)

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body

Cookies (1)

Headers (3)

Test Results

Status: 200 OK

Time: 679 ms

Size: 278 B

Save

Download

Pretty

Raw

Preview

JSON

```
1 {
2   "id": "5d651f0ed6cbe64b327e04f1",
3   "marca": "Honda",
4   "modelo": "Accord Sedan EX 2.0 16V 156cv Aut",
5   "combustivel": "Gasolina",
6   "ano": "2012",
7   "valor": 50235
8 }
```

Learn

Build

Browse

EXECUÇÃO DE TESTE: Update

The screenshot displays the Postman application interface. The top bar shows the system time as 10:00 and the application name 'Postman'. The main workspace is titled 'My Workspace' and includes an 'Invite' button. The left sidebar contains a 'Filter' input, 'History', and 'Collections' tabs. Under 'Collections', there is a 'Trash' section and two collections: 'Spring Boot Mongo...' with 6 requests and 'Api Kevin' with 0 requests. The 'Spring Boot Mongo...' collection is expanded, showing a list of requests. The main panel displays a PUT request to 'localhost:8080/cars/5d651f0ed6cbe64b327e04f1'. The request body is a JSON object:

```
{  "marca": "Honda",  "modelo": "Accord Sedan EX 2.0 16V 156cv Aut",  "combustivel": "Gasolina/Alcool",  "ano": "2012",  "valor": "51111.00"}
```

. The response is also a JSON object:

```
{  "id": "5d651f0ed6cbe64b327e04f1",  "marca": "Honda",  "modelo": "Accord Sedan EX 2.0 16V 156cv Aut",  "combustivel": "Gasolina/Alcool",  "ano": "2012",  "valor": "51111"}
```

. The status bar at the bottom indicates a successful response with status 200 OK, time 346 ms, and size 285 B. The bottom right corner has buttons for 'Learn', 'Build', and 'Browse'.

Activities Postman

ter 10:00

Postman

File Edit View Help

New Import Runner

My Workspace Invite

No Environment

Filter

History Collections

Trash

Spring Boot Mongo... 6 requests

GET localhost:8080/

GET localhost:8080/cars

DEL localhost:8080/cars/5d64...

PUT localhost:8080/cars/5d64...

GET localhost:8080/cars/5d64...

POST localhost:8080/cars

Api Kevin 0 requests

This collection is empty. Add requests to this collection and create folders to organize them

localhost:8080/cars/5d64936d827a0b61fdc0bfd0

Examples (0)

PUT localhost:8080/cars/5d651f0ed6cbe64b327e04f1

Send Save

Params Authorization Headers (1) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary JSON (application/json)

Beautify

1 {
2 "marca": "Honda",
3 "modelo": "Accord Sedan EX 2.0 16V 156cv Aut",
4 "combustivel": "Gasolina/Alcool",
5 "ano": "2012",
6 "valor": "51111.00"
7 }

Body Cookies (1) Headers (3) Test Results

Status: 200 OK Time: 346 ms Size: 285 B Save Download

Pretty Raw Preview JSON

1 {
2 "id": "5d651f0ed6cbe64b327e04f1",
3 "marca": "Honda",
4 "modelo": "Accord Sedan EX 2.0 16V 156cv Aut",
5 "combustivel": "Gasolina/Alcool",
6 "ano": "2012",
7 "valor": "51111"
8 }

Learn Build Browse

EXECUÇÃO DE TESTE: Delete

The screenshot shows the Postman application interface. The top bar displays the time as 10:01 and the language as pt. The main workspace shows a collection named "Spring Boot Mongo..." with 6 requests. The selected request is a DELETE request to the endpoint `localhost:8080/cars/5d651f0ed6cbe64b327e04f1`. The request is configured with the following parameters:

KEY	VALUE	DESCRIPTION
Key	Value	Description

The response is displayed in the "Body" tab, showing a status of 200 OK, a time of 39 ms, and a size of 75 B. A tooltip for the 200 OK status is visible, stating: "Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request the response will contain an entity describing or containing the result of the action."