



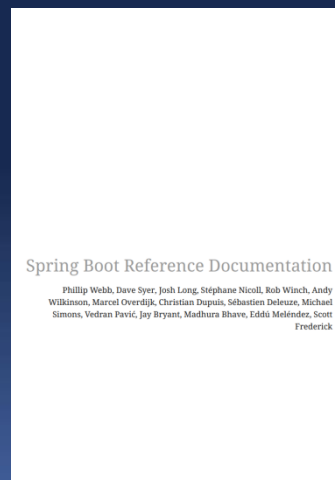
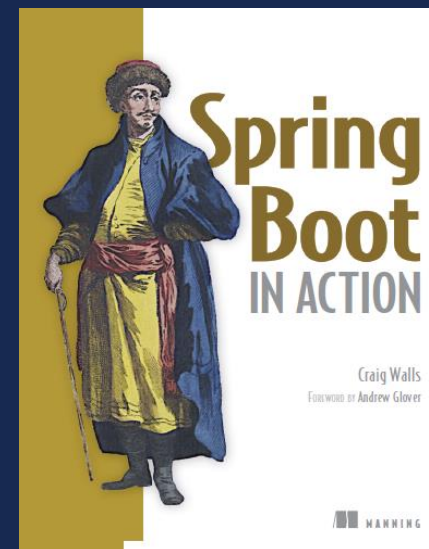
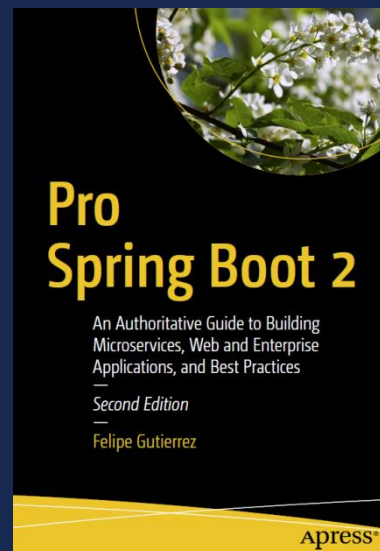
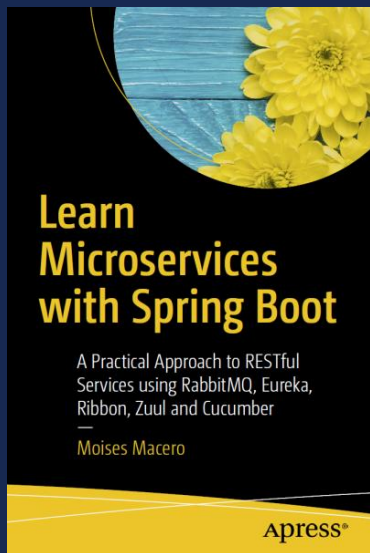
Spring Boot

Unidade 3 – API Rest – Consulta à Banco de Dados



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUVSP
aparecidovfreitas@gmail.com

Bibliografia



Introdução

- Na unidade anterior, desenvolvemos uma **API Rest** que retorna um recurso correspondente à uma lista de cursos;
- A **API** desenvolvida **não** utilizava Banco de Dados e, assim, o desenvolvimento baseou-se em uma Lista em memória;
- Nesta unidade modificaremos nossa API para utilizar uma implementação em Banco de Dados em substituição à lista em memória usada nas unidades anteriores.

Introdução

- Assim, substituiremos o código da API abaixo com por outro equivalente com acesso a Banco de Dados.

```
@RestController
public class CursosController {
    @RequestMapping("/cursos")
    public List<CursoDto> listaCursos() {

        Curso curso1 = new Curso (1,
            "Sistemas de Informação",
            "2020-07-17 09:20:39",
            "2020-07-17 09:20:39");
        Curso curso2 = new Curso (2,
            "Gestão de Tecnologia da Informação",
            "2020-07-17 09:21:27",
            "2020-07-17 09:21:27");

        //return (Arrays.asList(curso1,curso2));
        return converter(Arrays.asList(curso1,curso2));

    }

    //metodo que recebe lista de cursos e retorna lista de cursoDto
    public static List<CursoDto> converter(List<Curso> listaCursos ) {

        List<CursoDto> listaCursosDto = new ArrayList<CursoDto>();
        int n = listaCursos.size();

        for (int i = 0; i < n; i++) {
            CursoDto c = new CursoDto(listaCursos.get(i).getIdCurso(), listaCursos.get(i).getNomeCurso());
            listaCursosDto.add(c);
        }
        return listaCursosDto;
    }
}
```

JPA – Java Persistence API

- **JPA** significa **Java Persistence API** e corresponde à uma especificação Java para prover persistência de dados em bancos de dados relacionais;
- Tem como base a **JDBC – Java DataBase Connectivity** e basicamente provê abstração dos detalhes que usualmente são necessários para que código Java interaja com Bancos de Dados Relacionais;
- A principal função da **JPA** é facilitar a conversão dos dados recuperados no formato relacional para o formato orientado a objetos (**mapeamento objeto-relacional**).



API – JPA

- Realiza o **mapeamento objeto-relacional** entre as classes Java e as tabelas do Banco de Dados;
- Escreve** as consultas SQL independentemente dos detalhes de implementação da Linguagem de cada Banco de Dados;
- Corresponde a uma **referência**, à qual deve ser seguida por todas as soluções de persistência relacional.



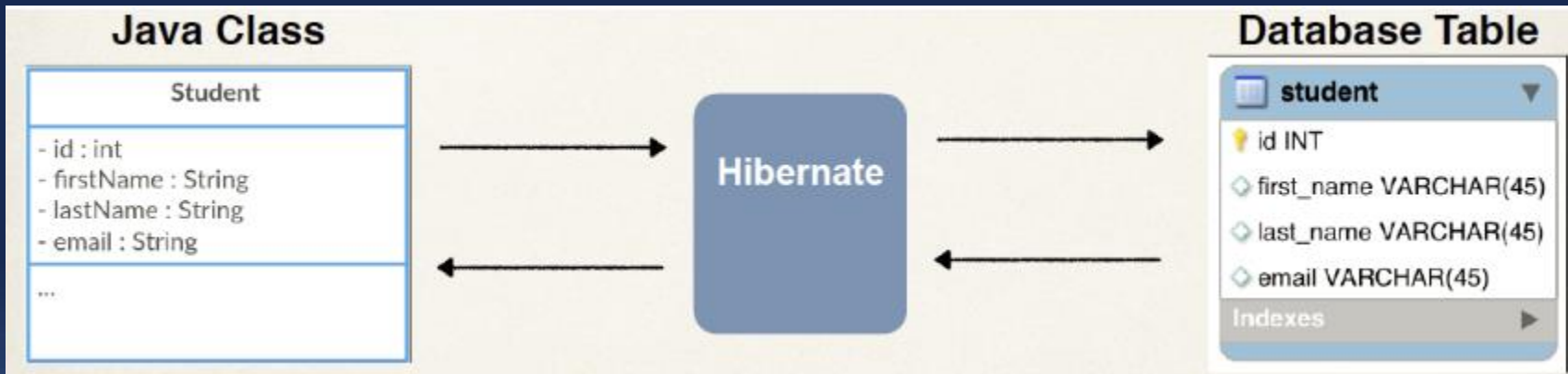
Implementações da JPA

- **JPA** é uma especificação e, como tal, não se pode programar diretamente com ela;
- Ou seja, será necessário que se disponha de uma solução que a implemente;
- Atualmente, a implementação mais usual no mercado é o **Hibernate**;
- Há também outras implementações tais como: **OpenJPA**, **EclipseLink JPA** e outras.



Framework Hibernate

- **Hibernate** é uma solução de mapeamento objeto-relacional para ambientes Java;
- Mapeamento objeto-relacional é uma técnica que permite o mapeamento de objetos do modelo de domínio da aplicação para tabelas do modelo relacional;
- Hibernate provê a implementação de referência da Java Persistence API (JPA).



O que é Spring Data JPA ?

- **Spring Data** é parte do **Framework Spring**;
- A meta das abstrações do **Spring Data Repository** é reduzir de forma significativa a quantidade de código requerida para se implementar a **camada de acesso aos dados** para vários modelos e fabricantes de banco de dados;
- Spring Data JPA **não** é um **JPA provider** ! Na verdade, Spring Data JPA é uma **biblioteca/framework** que adiciona uma **camada extra de abstração no topo da implementação do JPA provider**, como por exemplo, o **Hibernate**.



Vê ! Qual então a diferença entre
Hibernate e Spring Data JPA ?



Hibernate x Spring Data JPA

- Spring Data JPA **não** é um **JPA provider** !
- Na verdade, Spring Data é uma **biblioteca/framework** que adiciona uma **camada extra de abstração no topo da implementação do JPA provider**, como por exemplo, o **Hibernate**;
- Com **Spring Data JPA**, pode-se usar qualquer JPA provider, tais como Hibernate, Eclipse Link ou qualquer outro provider;
- Assim, **Spring Data JPA requer** um JPA provider, como por exemplo, **Hibernate**.



Configurando o Spring Data JPA

- Combinando-se o Spring Boot com o **Spring Data JPA**, pode-se configurar, de forma rápida e fácil, um repositório de dados com uma API Rest;
- Voltando ao nosso projeto, faremos inicialmente a configuração do **Spring Data JPA** em nosso projeto;
- Para isso, iremos incluir os módulos do **Spring Data JPA** em nosso projeto, alterando-se o arquivo **pom.xml**.

Configurando o Spring Data JPA – pom.xml

```

</parent>
<groupId>br.com.qualitsys</groupId>
<artifactId>scpe</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>scpe</name>
<description>Demo project for Spring Boot</description>

<properties>
  <java.version>1.8</java.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
      <exclusion>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
    </exclusions>
  </dependency>

```



Configurando o Spring Data JPA

- Por padrão, o **Spring Data JPA** utiliza o **Hibernate** como **JPA provider**, porém isso também pode ser configurado e, portanto, poderia-se usar no projeto, por exemplo, o Eclipse Link;
- Utilizaremos nessa unidade, o **Hibernate** por ser uma implementação largamente utilizada no mercado, o qual é padrão quando se usa o Spring Data JPA;
- Precisamos também incluir no projeto a dependência do Banco de dados. Nesta unidade, utilizaremos o Banco de Dados **MySQL**.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Configurando o connector MySQL no projeto

```
<groupId>br.com.qualitsys</groupId>
<artifactId>scpe</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>scpe</name>
<description>Demo project for Spring Boot</description>

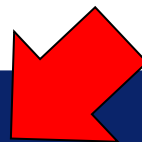
<properties>
  <java.version>1.8</java.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
  </dependency>
```



Configurando o acesso ao Banco de Dados

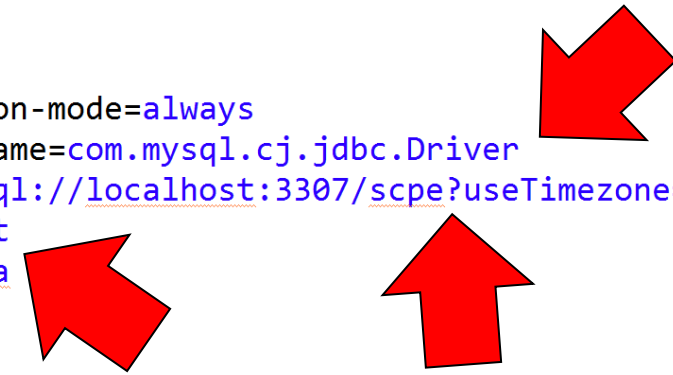


- Iremos agora fazer a configuração das informações do Banco de Dados, tais como nome do Banco de Dados, usuário e senha;
- Essas configurações são feitas no arquivo **application.properties** que se encontra na pasta: **src/main/resources** do nosso projeto;
- A entrada `spring.jpa.hibernate.ddl-auto=update` faz com que as alterações processadas pelo Hibernate sejam automaticamente refletidas no banco de dados.

```
server.port=5555

# data source
spring.datasource.initialization-mode=always
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3307/scpe?useTimezone=true&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=maua

# jpa
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true
```



Testando a configuração



- Vamos reiniciar a aplicação para checar se o Spring Data foi inicializado:

```
: Tomcat initialized with port(s): 5555 (http)
: Starting service [Tomcat]
: Starting Servlet engine: [Apache Tomcat/9.0.36]
: Initializing Spring embedded WebApplicationContext
: Root WebApplicationContext: initialization completed in 1590 ms
: Initializing ExecutorService 'applicationTaskExecutor'
: HHH000204: Processing PersistenceUnitInfo [name: default]
: spring.jpa.open-in-view is enabled by default. Therefore, database queries may
: HHH000412: Hibernate ORM core version 5.4.17.Final
: HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
: HikariPool-1 - Starting...
: LiveReload server is running on port 35729
: Tomcat started on port(s): 5555 (http) with context path ''
: Triggering deferred initialization of Spring Data repositories...
: HikariPool-1 - Start completed.
: HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
: Initialized JPA EntityManagerFactory for persistence unit 'default'
: Spring Data repositories initialized!
: Started ScpeApplication in 4.128 seconds (JVM running for 4.599)
: Initializing Spring DispatcherServlet 'dispatcherServlet'
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 6 ms
```



Configuração das Entidades – JPA

- As classes de **Domínio** da aplicação serão mapeadas para entidades na **JPA**;
- Assim, em nossa aplicação a classe `Curso` receberá a anotação **@Entity**;
- Adicionalmente, em cima do atributo que representa a chave primária incluiremos a anotação **@Id**.
- Caso se queira que a chave seja gerada automaticamente de forma sequencial, deve-se também incluir a anotação **@GeneratedValue** sobre o campo que representa a chave primária. Essa anotação pode receber dois atributos opcionais: **strategy** ou **generator**;
- O atributo opcional `Strategy` nos permite definir a estratégia a ser utilizada para a geração da chave primária, podendo ser: **AUTO**, **IDENTITY**, **SEQUENCE** ou **TABLE**;
- Ao se usar: **strategy=Generation.Type.IDENTITY**, estamos definindo que a chave será auto-incrementada.

Configuração da Entidade Curso



```
package br.com.qualitsys.scpe.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Curso {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)

    private Integer idCurso;
    private String nomeCurso;
    private String timestampCurso;
    private String datetimeCurso;

    public Integer getIdCurso() {
        return idCurso;
    }

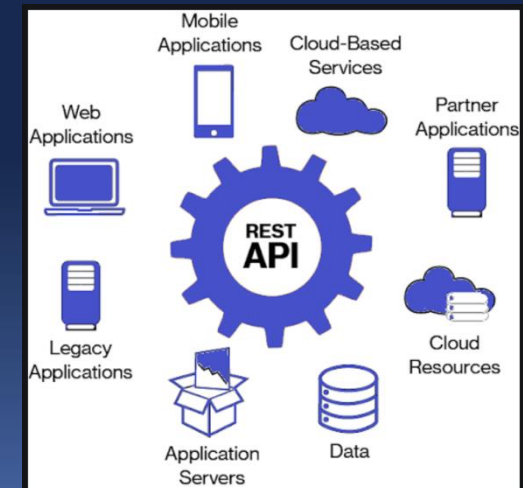
    public void setIdCurso(Integer idCurso) {
        this.idCurso = idCurso;
    }

    public String getNomeCurso() {
```



Concluindo a API /cursos

- Até aqui configuramos a **JPA** do projeto, fizemos o **mapeamento** da classe **Curso** para a correspondente tabela no banco de dados;
- Fizemos também toda a configuração do **Hibernate** e do banco de dados **MySQL** em nosso projeto;
- Agora, iremos concluir a **api** mapeada na url **/cursos**, **substituindo** a lista em memória pelos **registros do banco de dados**.



Spring Data e acesso ao BD

- Com **Spring Data** o acesso aos dados do Banco de Dados pode ser feito por meio de um padrão chamado **Repository**;
- No projeto, criaremos uma **Interface** (**não** uma classe) que será herdada de uma **interface** do **Spring Data** a qual já possui uma série de métodos prontos e abstraídos para a nossa aplicação;
- No nosso package principal **br.com.qualitsys.scpe**, criaremos então uma interface chamada **CursoRepository** que será armazenada no package **br.com.qualitsys.scpe.repository**.

Spring Data e acesso ao BD



New Java Interface

Java Interface
Create a new Java interface.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected

Extended interfaces:

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments



Interface CursoRepository



- Como estamos criando uma interface, **não** precisamos colocar nenhuma anotação em cima dela, pois o **Spring** já a encontra automaticamente;
- Essa interface deverá ser **filha** de alguma interface do **Spring Data**;
- No nosso caso, ela será filha da interface **JpaRepository**;
- Como estamos trabalhando nesse caso com herança, todos os métodos da JpaRepository são portanto herdados (métodos para carregar registros, alterar, excluir, etc...)

```
package br.com.qualitsys.scpe.repository;  
import org.springframework.data.jpa.repository.JpaRepository;  
import br.com.qualitsys.model.Curso;  
  
public interface CursoRepository extends JpaRepository<Curso, Integer> {  
  
}
```




Interface CursoRepository



- A interface JpaRepository é uma interface genérica;
- O primeiro tipo parametrizado da Interface é a Classe que essa interface Repository irá trabalhar;
- O segundo tipo parametrizado é o tipo da chave primária da entidade Curso.

```
package br.com.qualitsys.scpe.repository;  
import org.springframework.data.jpa.repository.JpaRepository;  
import br.com.qualitsys.model.Curso;  
  
public interface CursoRepository extends JpaRepository<Curso, Integer> {  
}
```

Two large red arrows point upwards from the bottom of the code block. The first arrow points to the 'Curso' parameter in the generic type 'CursoRepository', and the second arrow points to the 'Integer' parameter in the generic type 'JpaRepository'.

Injetando o Repository

- O próximo passo agora é injetar a interface Repository na classe CursosController;
- Essa injeção é materializada pela anotação @AutoWired;
- Como a API é de consulta, faremos o mapeamento do endpoint com @GetMapping

Codificando o controller



```
package br.com.qualitsys.scpe.controller;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.qualitsys.scpe.controller.dto.CursorDto;
import br.com.qualitsys.scpe.model.Cursor;
import br.com.qualitsys.scpe.repository.CursorRepository;

@RestController
public class CursosController {

    @Autowired
    private CursorRepository cursoRepository;

    @GetMapping("/cursos")
    public List<CursorDto> listaCursos() {
        List<Cursor> cursos = cursoRepository.findAll();
        return converter(cursos);
    }
}
```



Codificando o controller

```
//metodo que recebe lista de cursos e retorna lista de cursoDto
public static List<CursoDto> converter(List<Curso> listaCursos ) {

    List<CursoDto> listaCursosDto = new ArrayList<CursoDto>();
    int n = listaCursos.size();

    for (int i = 0; i < n; i++) {
        CursoDto c = new CursoDto(listaCursos.get(i).getIdCurso(), listaCursos.get(i).getNomeCurso());
        listaCursosDto.add(c);
    }
    return listaCursosDto;
}
```

Aplicação

```
package br.com.qualitsys.scpe;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class ScpeApplication extends SpringBootServletInitializer{

    public static void main(String[] args) {
        SpringApplication.run(ScpeApplication.class, args);
    }
}
```

Classe de Domínio Curso



```
package br.com.qualitsys.scp.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Curso {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)

    private Integer idCurso;
    private String nomeCurso;
    private String timestampCurso;
    private String datetimeCurso;

    public Integer getIdCurso() {
        return idCurso;
    }

    public void setIdCurso(Integer idCurso) {
        this.idCurso = idCurso;
    }

    public String getNomeCurso() {
        return nomeCurso;
    }
}
```



Classe de Domínio Curso

```
public void setIdCurso(Integer idCurso) {  
    this.idCurso = idCurso;  
}  
  
public String getNomeCurso() {  
    return nomeCurso;  
}  
  
public void setNomeCurso(String nomeCurso) {  
    this.nomeCurso = nomeCurso;  
}  
  
public String getTimestampCurso() {  
    return timestampCurso;  
}  
  
public void setTimestampCurso(String timestampCurso) {  
    this.timestampCurso = timestampCurso;  
}  
  
public String getDatetimeCurso() {  
    return datetimeCurso;  
}  
  
public void setDatetimeCurso(String datetimeCurso) {  
    this.datetimeCurso = datetimeCurso;  
}  
}
```

Interface CursoRepository

```
package br.com.qualitsys.scpe.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import br.com.qualitsys.scpe.model.Curso;  
  
public interface CursoRepository extends JpaRepository<Curso,Integer>{  
  
}
```

Inserção de registros no BD



- Considerando que a API que estamos desenvolvendo nesta unidade é de consulta, faremos a gravação no banco de dados por meio de SQL fora da aplicação.

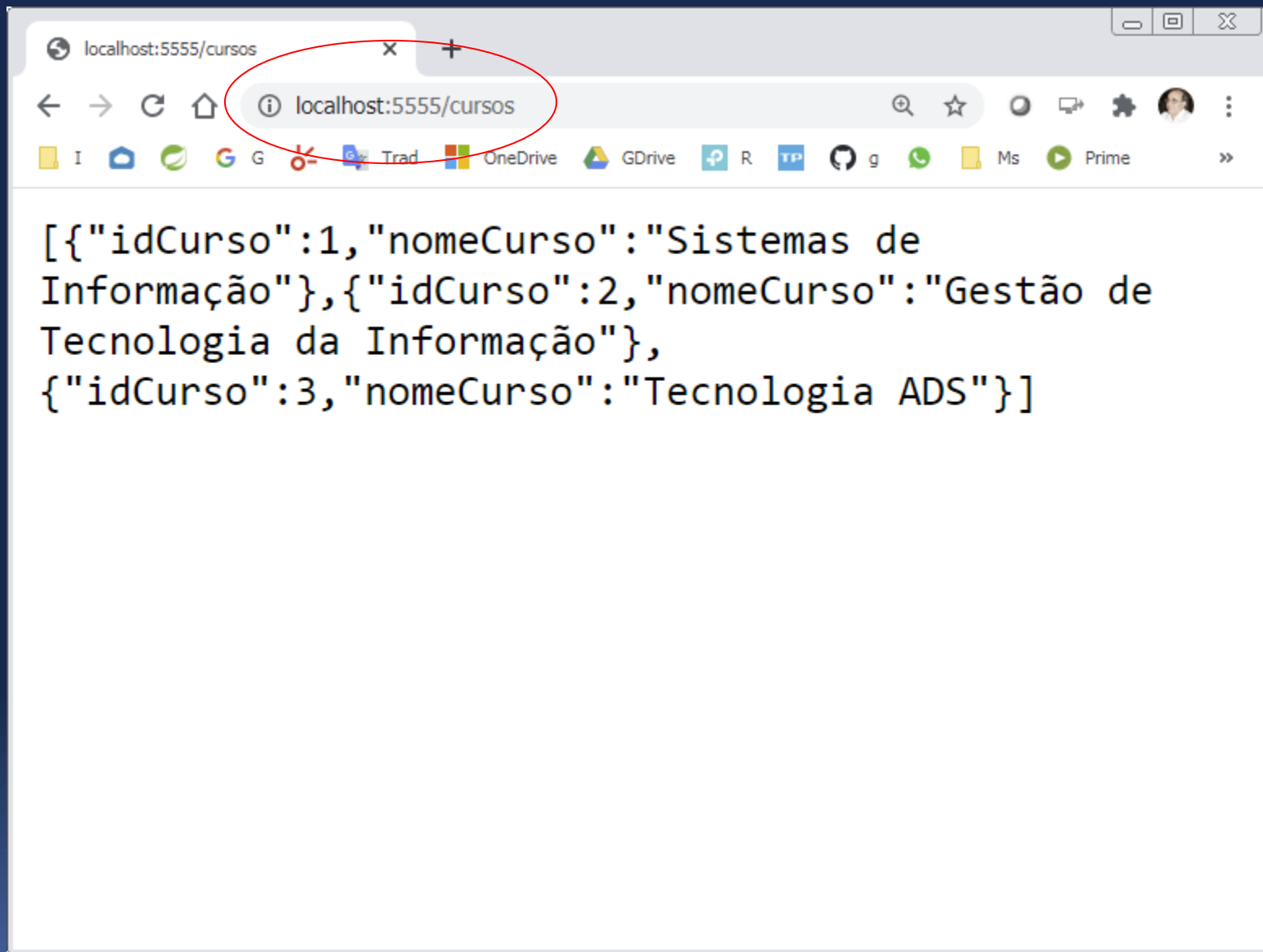
```
use scte;  
  
describe curso;  
  
insert into curso values(1, now(), 'Sistemas de Informação' , CURRENT_TIMESTAMP);  
insert into curso values(2, now(), 'Gestão de Tecnologia da Informação' , CURRENT_TIMESTAMP);  
insert into curso values(3, now(), 'Tecnologia ADS' , CURRENT_TIMESTAMP);  
  
select * from curso;
```

Field	Type	Null	Key	Default	Extra
id_curso	int(11)	NO	PRI	NULL	auto_increment
datetime_curso	varchar(255)	YES		NULL	
nome_curso	varchar(255)	YES		NULL	
timestamp_curso	varchar(255)	YES		NULL	

id_curso	datetime_curso	nome_curso	timestamp_curso
1	2020-07-20 21:51:15	Sistemas de Informação	2020-07-20 21:51:15
2	2020-07-20 21:51:18	Gestão de Tecnologia da Informação	2020-07-20 21:51:18
3	2020-07-20 21:51:21	Tecnologia ADS	2020-07-20 21:51:21
NULL	NULL	NULL	NULL



Executando a aplicação



Executando em Tomcat externo

arquivo war foi gerado na pasta target

```
Command Prompt
Volume Serial Number is 14D9-25F0

Directory of E:\Integrator_Projetos\scpe\target

20-Jul-20  04:54 PM      <DIR>          .
20-Jul-20  04:54 PM      <DIR>          ..
20-Jul-20  08:35 PM      <DIR>          classes
20-Jul-20  04:54 PM      <DIR>          generated-sources
20-Jul-20  04:54 PM      <DIR>          generated-test-sources
20-Jul-20  08:35 PM      <DIR>          m2e-wtp
20-Jul-20  04:54 PM      <DIR>          maven-archiver
20-Jul-20  04:54 PM      <DIR>          maven-status
20-Jul-20  04:54 PM      <DIR>          scpe-0.0.1-SNAPSHOT
17-Jul-20  08:49 PM      11,273,641 scpe-0.0.1-SNAPSHOT.war.original
20-Jul-20  04:54 PM      <DIR>          surefire-reports
20-Jul-20  08:35 PM      <DIR>          test-classes
          1 File(s)      11,273,641 bytes
          11 Dir(s)    350,667,403,264 bytes free

E:\Integrator_Projetos\scpe\target>
```

Renomeando arquivo war para scpe.war

```

C:\> Command Prompt

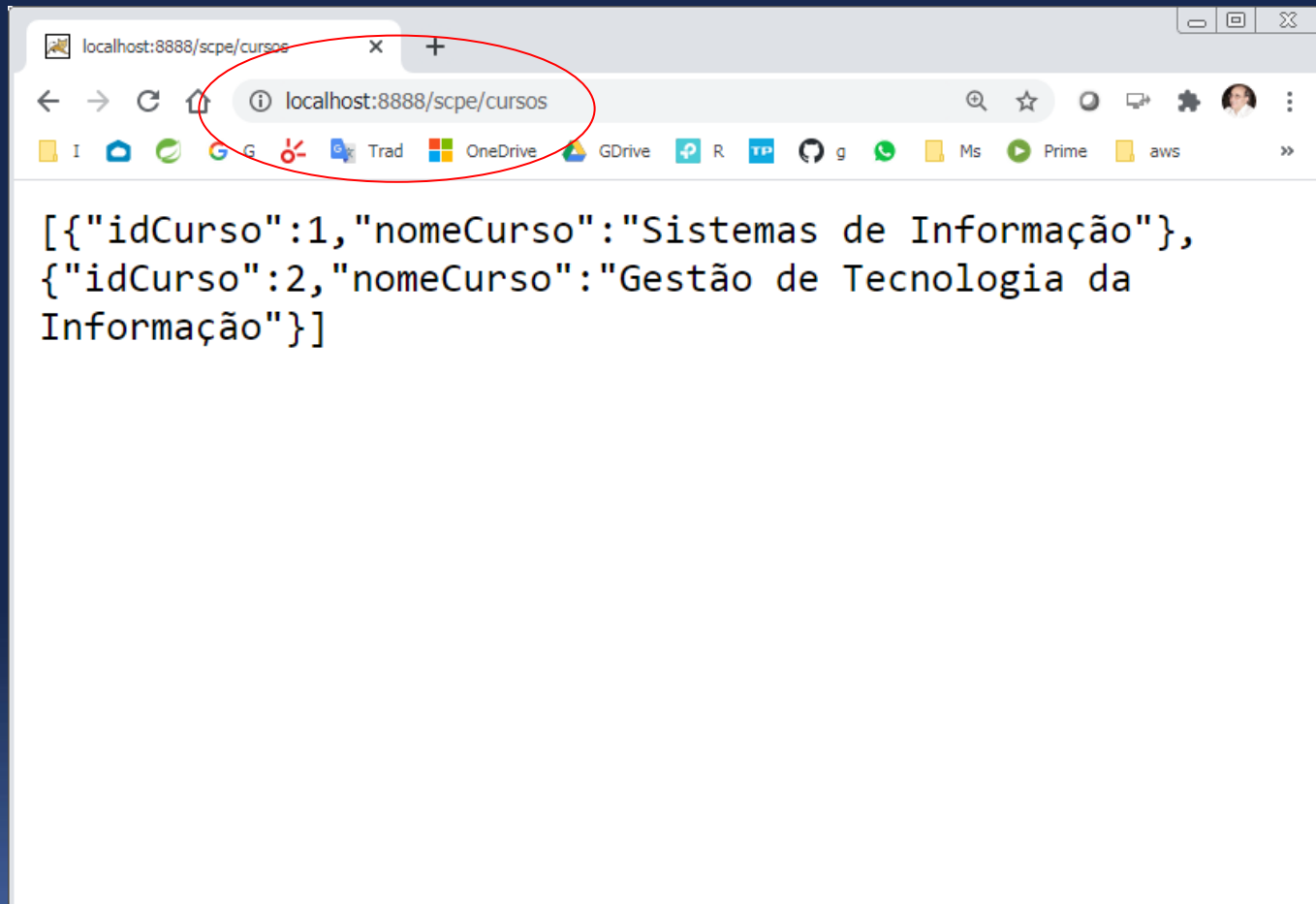
Directory of E:\Integrator_Projetos\scpe\target

21-Jul-20  09:34 AM    <DIR>          .
21-Jul-20  09:34 AM    <DIR>          ..
20-Jul-20  08:35 PM    <DIR>          classes
20-Jul-20  04:54 PM    <DIR>          generated-sources
20-Jul-20  04:54 PM    <DIR>          generated-test-sources
20-Jul-20  08:35 PM    <DIR>          m2e-wtp
20-Jul-20  04:54 PM    <DIR>          maven-archiver
20-Jul-20  04:54 PM    <DIR>          maven-status
20-Jul-20  04:54 PM    <DIR>          scpe-0.0.1-SNAPSHOT
17-Jul-20  08:49 PM             11,273,641 scpe-0.0.1-SNAPSHOT.war.original
17-Jul-20  08:49 PM             11,273,641 scpe.war
20-Jul-20  04:54 PM    <DIR>          surefire-reports
20-Jul-20  08:35 PM    <DIR>          test-classes
           2 File(s)      22,547,282 bytes
          11 Dir(s)  350,644,850,688 bytes free

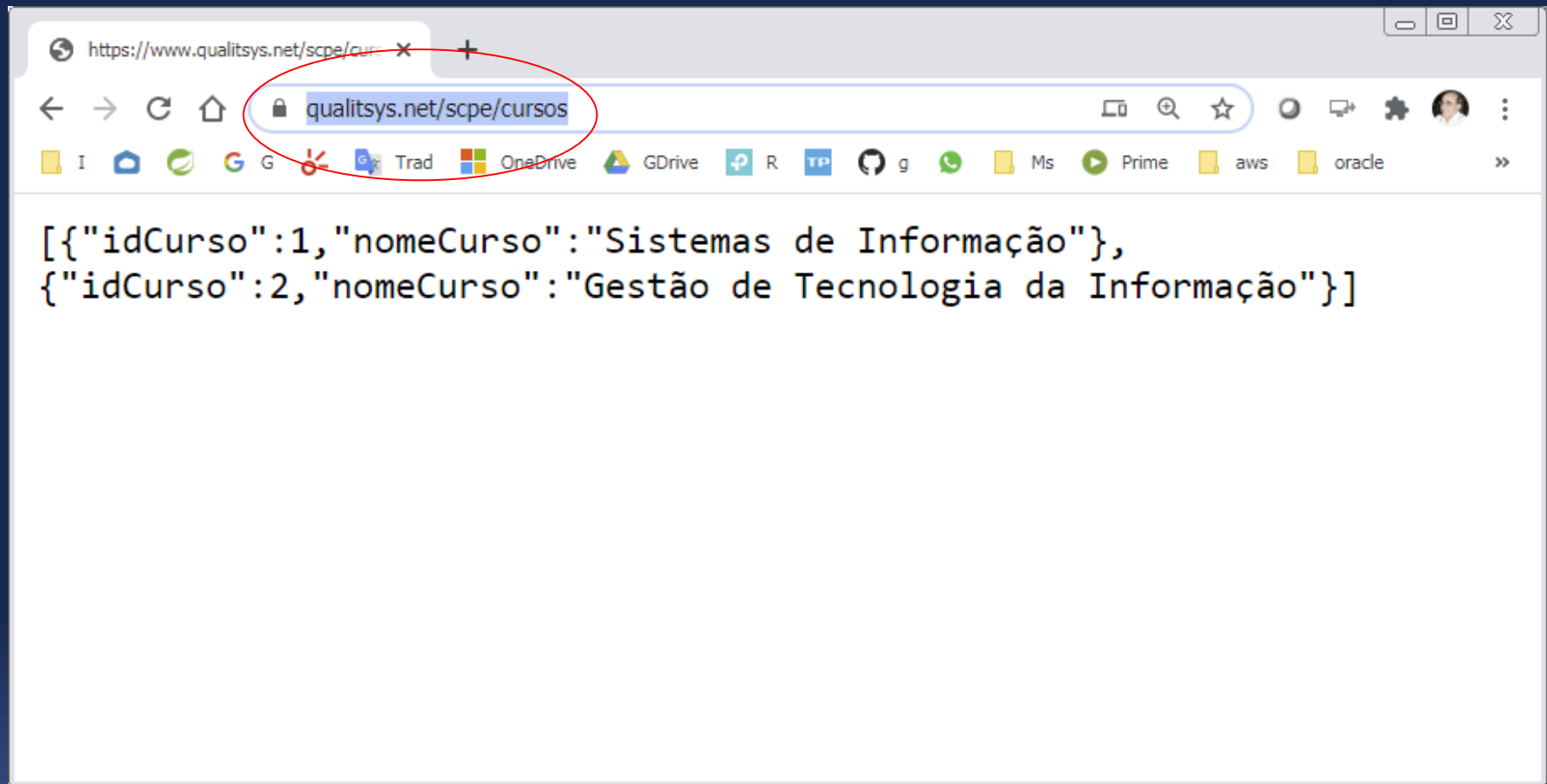
E:\Integrator_Projetos\scpe\target>
  
```

Portando num Tomcat externo

- Basta copiar o arquivo `scpe.war` para a pasta `webapps` do Tomcat externo!



Portando na nuvem – integrator.com.br









Salvando projeto no Repositório git local

Dados (E:) ▾ Integrator_Repository ▾

elp

y ▾ Share with ▾ Burn New folder

Name ^	Date modified	Type
 .git	20-Jul-20 3:49 PM	File folder
 hellospringboot	19-Jul-20 6:44 PM	File folder
 listcursos	20-Jul-20 3:25 PM	File folder
 scpe 	21-Jul-20 9:58 AM	File folder
 README	20-Jul-20 3:41 PM	Markdown Source File

Enviando projeto para github



```
Command Prompt
E:\Integrator_Repository>git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    scpe/

nothing added to commit but untracked files present (use "git add" to track)
E:\Integrator_Repository>
```



Enviando projeto para github



```
Command Prompt
E:\Integrator_Repository>git pull
Already up to date.

E:\Integrator_Repository>git push
Enumerating objects: 38, done.
Counting objects: 100% (38/38), done.
Delta compression using up to 4 threads
Compressing objects: 100% (24/24), done.
Writing objects: 100% (37/37), 54.41 KiB | 6.04 MiB/s, done.
Total 37 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/avfreitas/Integrator_Repository.git
  57e504e..050c598  master -> master

E:\Integrator_Repository>_
```



Consultando github



avfreitas/Integrator_Repository

Search or jump to...

Pull requests Issues Marketplace Explore

avfreitas / Integrator_Repository

Watch 0 Star 0 Fork 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags

Go to file Add file Code

avfreitas Update README.md efc3e1c 17 seconds ago 6 commits

hellospringboot	v1.0.0 - 19/07/2020	2 days ago
listcursos	listcursos - v1.0.0 - api s/JPA	19 hours ago
scpe	rest /cursos V0.1 21/07/20	4 minutes ago
README.md	Update README.md	17 seconds ago

README.md

Integrator_Repository

Hellospringboot - projeto de configuração (setup básico) do Spring Boot listcursos - api Rest com dados em memória (list) scpe/cursos - api Rest com consulta a BD MySQL

About

No description, website, or topics provided.

Readme

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Languages



Validando repositório git

```
Command Prompt
E:\Integrator_Repository>git pull
Already up to date.

E:\Integrator_Repository>git push
Everything up-to-date

E:\Integrator_Repository>_
```