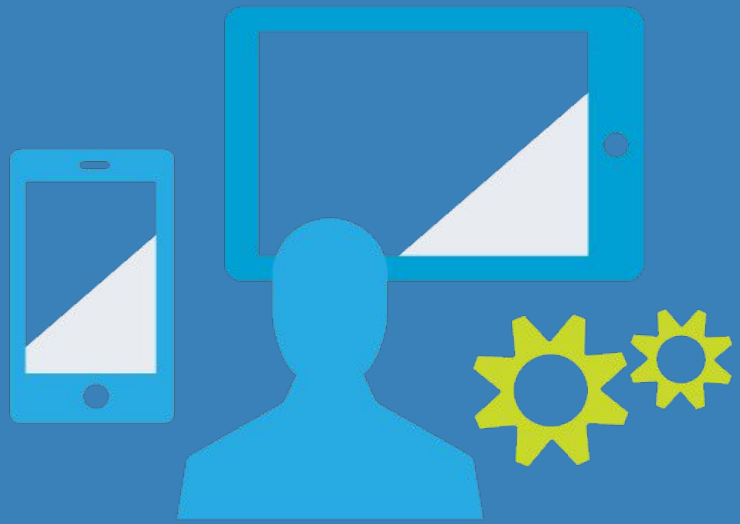
WEBSERVICES REST
COM SPRING BOOT

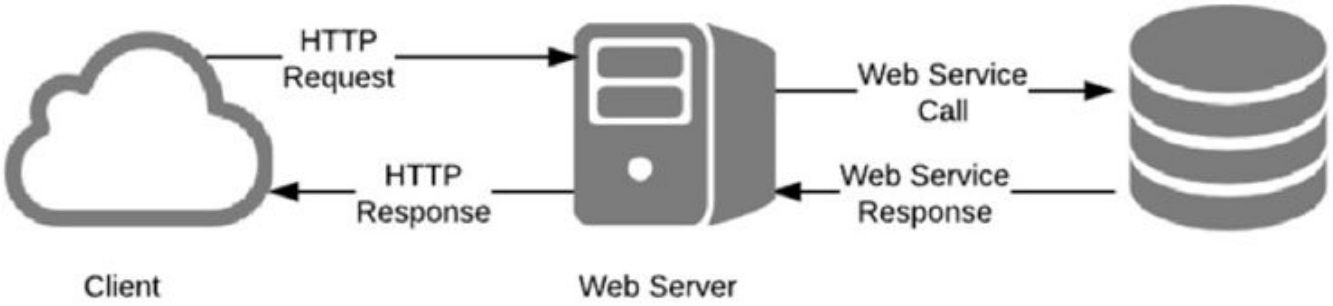
PROF. DANIEL ABELLA

FUNDAMENTAÇÃO TEÓRICA



WEBSERVICE REST

- Sistema que utiliza o protocolo HTTP do mesmo jeito que uma aplicação web
  - Requests e Responses
  - Suporta vários formatos como JSON, XML, entre outros



USO DOS **VERBOS HTTP**

http://locahost:8080/api/bookmarks

RESTful		
Verbo	**URI (substantivo)**	**Ação**
POST	/bookmarks	Criar
GET	/bookmarks/1	Visualizar
GET	/bookmarks/	Visualizar
PUT	/bookmarks/1	Alterar
DELETE	/bookmarks/1	Apagar

HTTP	SQL	CRUD
POST	INSERT	CREATE
GET	SELECT	RETRIEVE
PUT	UPDATE	UPDATE
DELETE	DELETE	DELETE

WEBSERVICE REST

- Fluxo requisição/resposta

- 100 – Continue
- 200 – OK
- 201 – Created
- 301 – Moved Permanently
- 303 – See Other
- 304 – Not Modified
- 400 – Bad Request
- 401 – Unauthorized
- 403 – Forbidden
- 404 – Not Found
- 405 – Method Not Allowed
- 500 – Internal Server Error

ABORDAGEM INCORRETA

- Comumente encontramos REST aplicados equivocadamente

Não RESTful

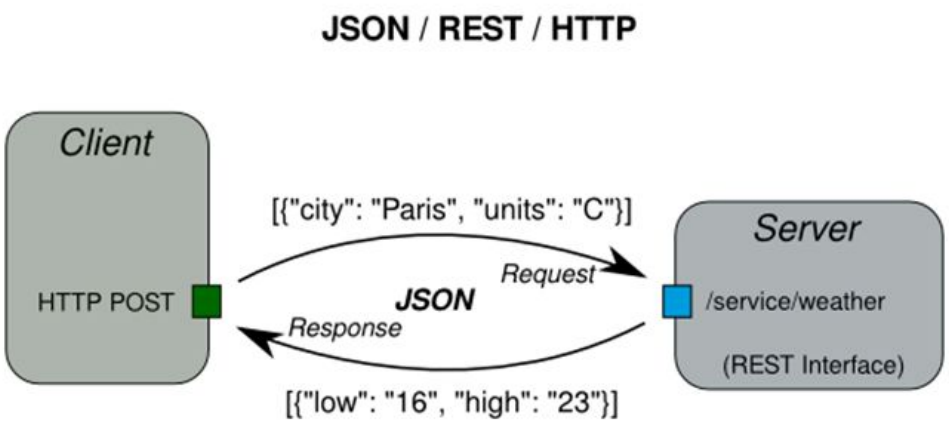
Verbo	URI (substantivo)	Ação
POST	/bookmarks/create	Criar
GET	/bookmarks/show/1	Visualizar
POST	/bookmarks/update/1	Alterar
GET/POST	/bookmarks/delete/1	Apagar

RESTful

Verbo	URI (substantivo)	Ação
POST	/bookmarks	Criar
GET	/bookmarks/1	Visualizar
PUT	/bookmarks/1	Alterar
DELETE	/bookmarks/1	Apagar

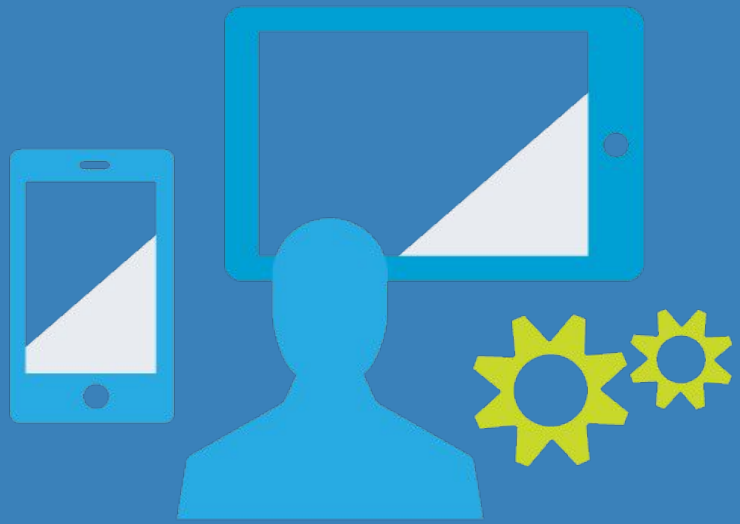
COMUNICAÇÃO REST

JSON



- 100 – Continue
- 200 – OK
- 201 – Created
- 301 – Moved Permanently
- 303 – See Other
- 304 – Not Modified
- 400 – Bad Request
- 401 – Unauthorized
- 403 – Forbidden
- 404 – Not Found
- 405 – Method Not Allowed
- 500 – Internal Server Error

FRAMEWORK
SPRING BOOT



FRAMEWORK SPRING BOOT

- **Motivação:** Setup do projeto em menor tempo possível
- Configuração por default do projeto
  - Mude o que não está no padrão

- Pronto para produção!
- A sua aplicação (.jar, por exemplo) já conta com um servidor de aplicações embarcado (*embedded*)
  - Quando você for executar é só *java -jar meujar.jar* que este já executada no servidor embarcado!

IMPORTANDO SEU **PROJETO**

- Download dos exemplos no meu Github
- File > Import > Existing Maven Projects
- Selecionar o exemplo1, exemplo2 ou exemplo3
- Renomear o projeto
- Clicar no pom.xml e alterar o artifact id e group id

HANDS ON
SPRING BOOT

EXEMPLO UM

HANDS ON SPRING BOOT

- Importar o projeto exemplo1
- Classe Application
  - Porta padrão é a 8080

HANDS ON SPRING BOOT #EXEMPLO1

- A partir de agora para realizar os testes usaremos a ferramenta **Postman**
  - Assista o vídeo para entender como funciona! https://www.youtube.com/watch?v=slrPZIvgTDw

- Nos permite realizar requisições GET, POST, PUT, DELETE, além dos outros verbos HTTP
  - Permite criar *collections* de testes
    - Se você quiser repetir os vários testes que fez previamente, basta consultar a *collection* criada.
    - Quer aprender? (Espero que sim!)
      - https://www.youtube.com/watch?v=bF8q8wvLs8A ou
      - https://www.getpostman.com/docs/collections

HANDS ON SPRING BOOT #EXEMPLO1

- Vamos começar a trabalhar com JSON e outros verbos HTTP
  - Primeiro modelamos a nossa entidade (User)

```
public class User implements Serializable {

    private static final long serialVersionUID = -7799369695818057571L;

    private int id;
    private String name;
    private String address;

    public User() {
    }
```

HANDS ON SPRING BOOT #EXEMPLO1

- Vamos começar a trabalhar com JSON e outros verbos HTTP
  - Primeiro modelamos a nossa entidade (User)
  - Agora criamos nosso controller (User Controller)

```
 9 @RestController
10 public class UserController {
11
12     @RequestMapping(value="/usuario", method = RequestMethod.GET)
13     public String listarTodosUsuarios() {
14         //iria ao bd, listaria todos os usuarios e retornaria.
15         return "todos";
16     }
```

HANDS ON SPRING BOOT

EXEMPLO DOIS

HANDS ON SPRING BOOT #EXEMPLO2

- Vamos evoluir o **exemplo1** apresentado anteriormente
  - Modificamos apenas o *UserController*
  - Agora usará dados *fake* (sem banco de dados ainda)

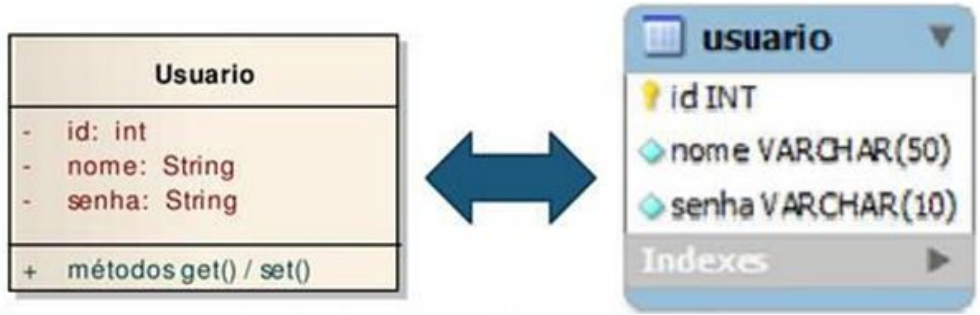
```
import java.util.ArrayList;

@RestController
public class UserController {

    @RequestMapping(value="/user", method = RequestMethod.GET)
    public ResponseEntity< List<User> > listAllUsers() {

        //dados fake
        List<User> listaUsuariosFake = new ArrayList<User>();
        listaUsuariosFake.add(new User(1,"Daniel","End1"));
        listaUsuariosFake.add(new User(2,"Ruan","End2"));
        listaUsuariosFake.add(new User(3,"Atylla","End3"));

        return new ResponseEntity< List<User> >(listaUsuariosFake, HttpStatus.OK);
    }
```

HANDS ON SPRING BOOT

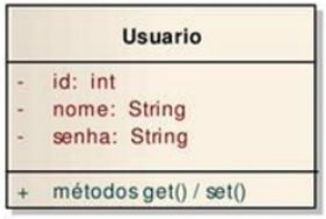
EXEMPLO TRES

CONCEITO MAPEAMENTO OBJETO-RELACIONAL

- **Criação do Objeto**
  - **Equivalente a:** Insert (SQL)
- **Alteração do Objeto**
  - **Equivalente a:** Update (SQL)
- **Remoção do Objeto**
  - **Equivalente a:** Delete (SQL)



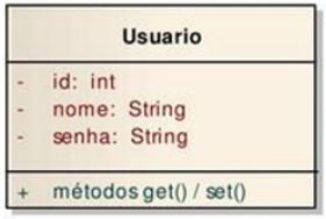
CONCEITO MAPEAMENTO OBJETO-RELACIONAL



	id	nome	senha

```java
public class Usuario {

    private int id;
    private String nome;
    private String senha;

    public Usuario() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }
}
```

CONCEITO MAPEAMENTO OBJETO-RELACIONAL



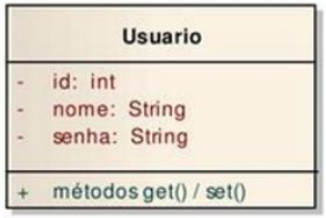
	id	nome	senha
	1	Daniel	123

```
Usuario usuario1 = new Usuario();
usuario1.setId(1);
usuario1.setNome("Daniel");
usuario1.setSenha("123");
```

	id	nome	senha
	1	Daniel	123
	2	Weslev	123

```
Usuario usuario1 = new Usuario();
usuario1.setId(1);
usuario1.setNome("Daniel");
usuario1.setSenha("123");

Usuario usuario2 = new Usuario();
usuario2.setId(2);
usuario2.setNome("Wesley");
usuario2.setSenha("123");
```

CONCEITO MAPEAMENTO OBJETO-RELACIONAL



	id	nome	senha
	1	Daniel	123

```
Usuario usuario1 = new Usuario();
usuario1.setId(1);
usuario1.setNome("Daniel");
usuario1.setSenha("123");
```

	id	nome	senha
	1	Daniel	12345

```
usuario1.setSenha("12345");
```

ESTRUTURA PROJETO

Entidade

Repository

Operações no BD

Service

Regras de Negócio

Controller

Endpoints

GET,POST,PUT, DELETE

200, 201, 404, 500

ESTRUTURA PROJETO

UserEntidade

UserRepository

Operações no BD

UserService

Regras de Negócio

UserController

Endpoints

GET,POST,PUT, DELETE

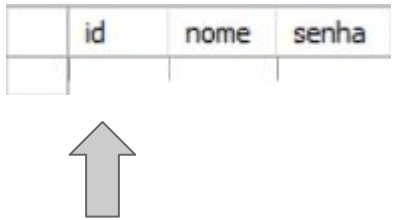
200, 201, 404, 500

HANDS ON SPRING BOOT #EXEMPLO3

- Vamos entender objetivamente o **exemplo3**

  - **Passo 1:** Mapeando a entidade User

```java
@Entity
public class User implements Serializable {

    private static final long serialVersionUID = -7799369695818057571L;

    @Id
    private String id;
    private String name;
    private String address;
```



Chave primária

HANDS ON SPRING BOOT #EXEMPLO3

- Vamos entender objetivamente o **exemplo3**
  - **Passo 2:** Criando o Repository

1 2 3

```
public interface UserRepository extends JpaRepository<User, String> {


}
```

  - **Pontos de Discussão:**
    - #1: Cria uma interface que estenda JpaRepository
    - #2: Entidade na qual se relaciona o Repository (User)
    - #3: Tipo do campo anotado com @Id na entidade

```
@Id
private String id;
```

HANDS ON SPRING BOOT #EXEMPLO3

- Vamos entender objetivamente o **exemplo3**
  - **Passo 3:** Criando a classe Service
    - Abriga as regras de negócio

```
@Service
@Validated
public class UserService {

    @Autowired
    private UserRepository repository;

    public User buscarUsuarioPorId(String id) {
        return repository.findOne(id);
    }

    public List<User> listarTodosUsuarios() {
        return repository.findAll();
    }

    @Transactional
    public User inserirUsuario(User usuario) {
        return repository.save(usuario);
    }
```

HANDS ON SPRING BOOT #EXEMPLO3

- Vamos entender objetivamente o **exemplo3**
  - **Passo 4:** Criando o Controller
    - Classe que recebe as requisições HTTP (métodos de *callback*)

```java
@RestController
public class UserController {

    @Autowired
    private UserService userService;

    @RequestMapping(value = "/user", method = RequestMethod.GET)
    public ResponseEntity<List<User>> listarUsuarios() {

        List<User> listaUsuarios = userService.listarTodosUsuarios();

        return new ResponseEntity<List<User>>(listaUsuarios, HttpStatus.OK);
    }
```

HANDS ON SPRING BOOT #EXEMPLO3

- **Passo 4:** Criando o Controller

```
@RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
public ResponseEntity<User> obterUsuario(@PathVariable String id) {

    User user = userService.buscarUsuarioPorId(id);

    if (user == null) {
        return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
    } else {
        return new ResponseEntity<User>(user, HttpStatus.OK);
    }
}

@RequestMapping(value = "/user", method = RequestMethod.POST)
public ResponseEntity<String> criarUsuario(@RequestBody User user) {

    try {
        userService.inserirUsuario(user);

        return new ResponseEntity<String>(HttpStatus.CREATED);

    } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<String>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

TESTE POSTMAN #EXEMPLO3

- Vamos fazer testes com os métodos expostos

HANDS ON
SPRING BOOT

EXEMPLO QUATRO

HANDS ON SPRING BOOT #EXEMPLO3

- Vamos aprimorar o Repository do **exemplo3**?
  - Imagine que eu gostaria de criar:
    - Método buscar pelo nome (atributo name de User)

```
public interface UserRepository extends JpaRepository<User, String> {

    public User findByName(String name);

}
```

QUICK REFERENCE CARD

- Query Methods

UNIFACISA
CENTRO UNIVERSITÁRIO

Desenvolvendo WebServices REST com Spring Boot
Guia de Referência Rápida
Curso de Sistemas da Informação - UniFacisa
Professor: Daniel Abella (daniel@daniel-abella.com / www.daniel-abella.com)

1. Verbos HTTP

RESTful

Verbo	URI (substantivo)	Ação
POST	/bookmarks	Criar
GET	/bookmarks/1	Visualizar
GET	/bookmarks/	Visualizar
PUT	/bookmarks/1	Alterar
DELETE	/bookmarks/1	Apagar

CRIANDO SEU PROJETO

- File > Import > Existing Maven Projects
- Selecionar o exemplo1, exemplo2 ou exemplo3
- Renomear o projeto
- Clicar no pom.xml e alterar o artifact id e group id

REFERÊNCIAS

- https://www.caelum.com.br/apostila-java-web/uma-introducao-pratica-ao-jpa-com-hibernate/#14-1-mapeamento-objeto-relacional
- FLOWER, Martin, Inversion of Control Containers and the Dependency Injection pattern. Disponível em : http://www.martinfowler.com/articles/injection.html
- Lobo, Henrique. Vire o Jogo com Spring Framework. Casa do Código.
- http://www.springbyexample.org/
- http://projects.spring.io

Vídeos criados para o assunto

- https://www.youtube.com/watch?v=EWtr124ZYGc&t=436s (parte 1)
- https://www.youtube.com/watch?v=dc7IksBIcNY&t=1s (parte 2)

OBRIGADO!

Dúvidas?

daniel@daniel-abella.com

www.daniel-abella.com