



5ª EDIÇÃO

CONGRESSO de TI

CONGRESSO ONLINE DE TECNOLOGIA DA INFORMAÇÃO

FACISA

FCM



DANIEL ABELLA

[www.daniel-abella.com](http://www.daniel-abella.com)

# CONSTRUINDO UMA MICROSERVICE ARCHITECTURE COM SPRING BOOT

PROF. DANIEL ABELLA

[DANIEL@DANIEL-ABELLA.COM](mailto:DANIEL@DANIEL-ABELLA.COM)

[WWW.DANIEL-ABELLA.COM](http://WWW.DANIEL-ABELLA.COM)





## PALESTRANTE **DANIEL ABELLA**

- Professor do Curso de Sistemas de Informação da UniFacisa
- Coordenador da Especialização em:
  - Gerenciamento de Projetos, Desenvolvimento Mobile e Qualidade de Software da UniFacisa
- Líder Técnico em Projetos de P&D&I no Virtus/UFCG
- Sócio da Empresa Pluslot
- Experiência com Gerenciamento de Projetos e Desenvolvimento de Software
- Certificado PMP, PMI-ACP, CSM, SCEA, ITIL, CTFL, entre outros

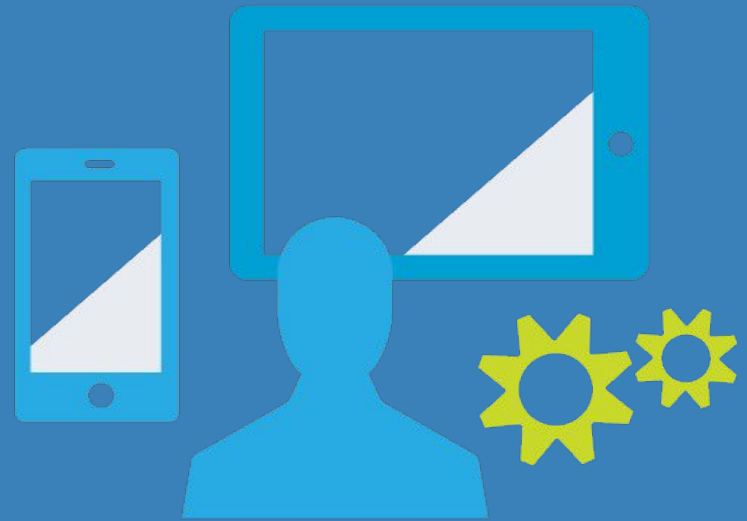


## RESUMO DA PALESTRA

- Construindo uma Microservice Architecture com Spring Boot
  - Spring Boot
  - Spring Data
  - MongoDB
  - Microservices Architecture



# 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





# 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



## WEBSERVICE REST

- Estilo arquitetural
  - Não é um padrão
  - Baseado em URIs
- Proposto na tese de Roy Fielding



Roy Fielding



## WEBSERVICE REST

- Sem regras rígidas
- Usa abertamente o HTTP (Verbos HTTP)
- Conceito do livro Java SOA Cookbook:



*“The idea of REST is essentially a reverse-engineering of how the Web Works”*





## 5 MANDAMENTOS FUNDAMENTAIS DO REST

- 1) Dê a todos os recursos um Identificador
  - Na Web, há um conceito unificado para IDs: **A URI**.
  - URIs compõe um namespace global, e utilizando URIs para identificar seus recursos chave significa ter um ID único e global.





## 5 MANDAMENTOS FUNDAMENTAIS DO REST

- 1) Dê a todos os recursos um Identificador
  - Exemplo de URIs
    - <http://example.com/customers/1234>
    - <http://example.com/orders/2007/10/776654>
    - <http://example.com/products/4554>
    - <http://example.com/processes/salary-increase-234>



## 5 MANDAMENTOS FUNDAMENTAIS DO REST

- 2) Vincule as coisas
- 3) Utilize métodos padronizados
- 4) Recursos com múltiplas representações
- 5) Comunique sem estado

Fonte: <https://www.infoq.com/br/articles/rest-introduction>



## USO DOS VERBOS HTTP

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

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



# ABORDAGEM INCORRETA

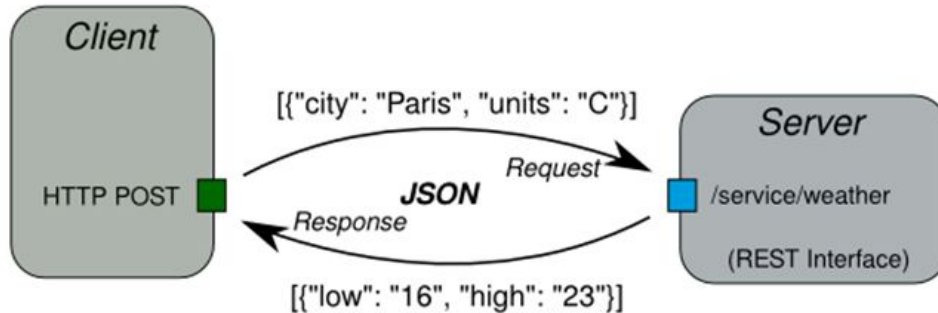
- Comumente encontramos REST aplicados equivocadamente

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 / REST / HTTP



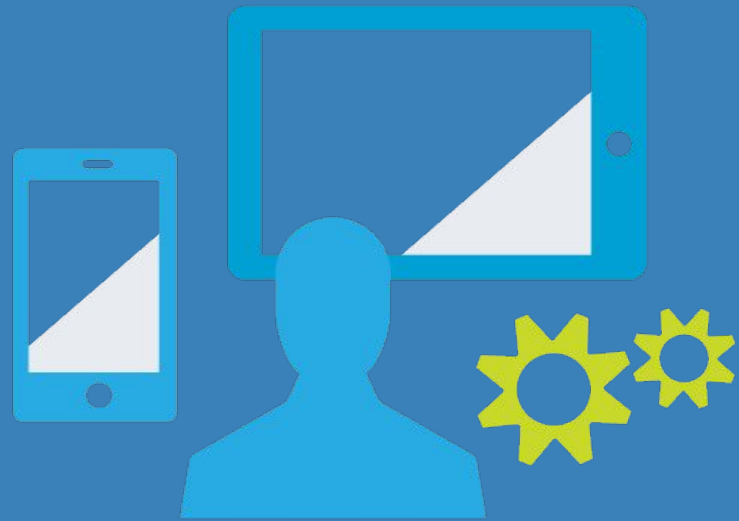
- 100 – Continue
- 200 – OK
- 201 – Created
- 301 – Moved Permanently
- 303 – See Other
- 304 – Not Modified
- 400 – Bad Request
- 401 – Unauthorized
- 403 – Forbidden
- 404 – Not Found
- 405 – Method Not Allowed
- 500 – Internal Server Error

VAMOS ENTENDER MELHOR



JSON, HTTP, REST,  
Quanta coisa!

# FRAMEWORK SPRING BOOT







## FRAMEWORK **SPRING BOOT**

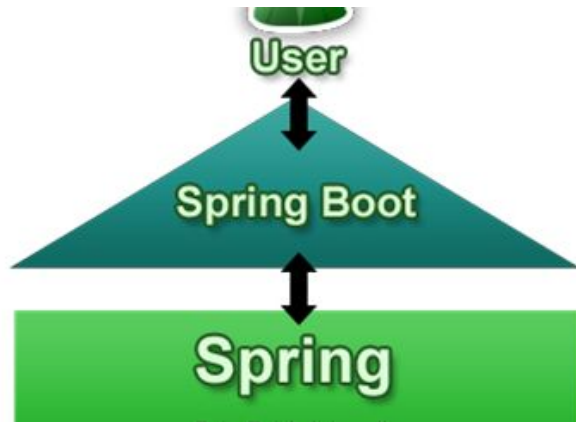
- **Motivação:** Setup do projeto em menor tempo possível
- Configuração por default do projeto
  - Mude o que não está no padrão
- Pronto para produção!
- A sua aplicação (.jar, por exemplo) já conta com um servidor de aplicações embarcado (*embedded*)
  - Quando você for executar é só *java -jar meu.jar* que este já executada no servidor embarcado!





## FRAMEWORK **SPRING BOOT**

- É uma pilha do Spring que facilita todo o trabalho com o Spring Framework





## PREPARANDO AMBIENTE DA AULA

- Instalar o Java e do Postman
- *Download* e configuração do Eclipse
- Instalar o Git
- Clone do projeto
  - [github.com/daniel-abella/springboot-course](https://github.com/daniel-abella/springboot-course)
- Importação do projeto

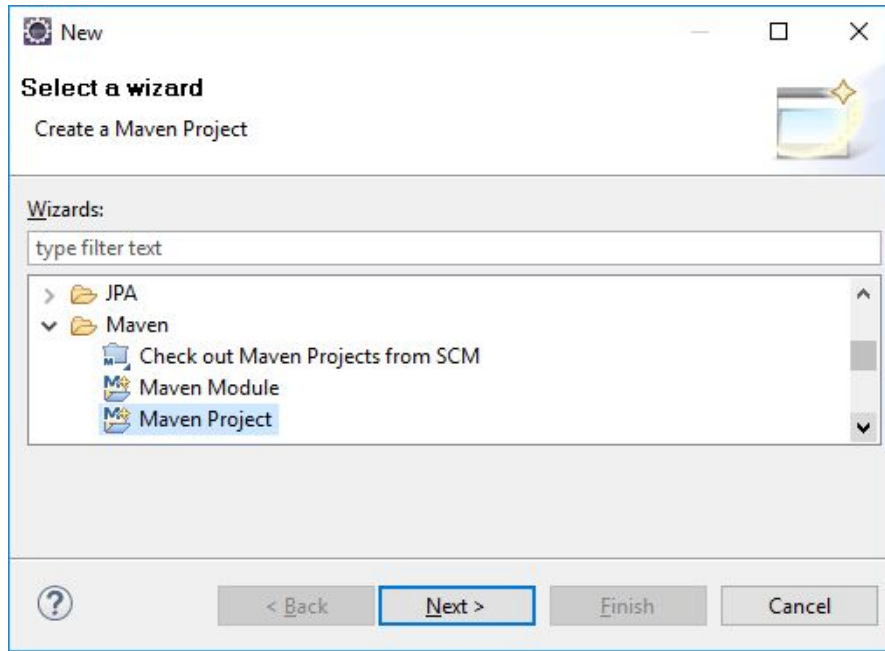
Vamos começar?  
Cadê o Spring Boot?  
Podemos?  
E agora? Podemos?





# CRIANDO PROJETO **SPRING BOOT**

- Ctrl + N
- Selecione *Maven Project* conforme imagem abaixo





# CRIANDO PROJETO **SPRING BOOT**

- Marque a opção *Create a simple project*
- Clique no botão *Next* e siga para próxima tela

New Maven Project

**New Maven project**

Configure project

Artifact

Group Id: com.danielabella.tap

Artifact Id: springboot-sample

Version: 0.0.1-SNAPSHOT

Packaging: jar

Parent Project

Group Id: org.springframework.boot

Artifact Id: spring-boot-starter-parent

Version: 1.3.2.RELEASE

Browse... Clear

► Advanced

? < Back Next > Finish Cancel

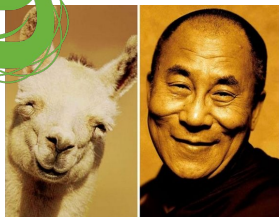


# CRIANDO PROJETO SPRING BOOT

- Adicione as seguintes dependências e plugins:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

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



## FICOU PARECIDO?

- ▼ springboot-sample
  - src/main/java
  - src/main/resources
  - src/test/java
  - src/test/resources
  - > JRE System Library [JavaSE-1.6]
  - > .settings
  - > src
  - target
  - .classpath
  - .project
  - pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.2.RELEASE</version>
  </parent>
  <groupId>com.danielabella.si.tap</groupId>
  <artifactId>springboot-2</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

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



# HANDS ON SPRING BOOT #EXEMPLO0

- O projeto com o exemplo está disponível: **exemplo0**

```
@Controller
@SpringBootApplication
public class Application {

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

    @RequestMapping(value = "/", method = RequestMethod.GET)
    @ResponseBody
    public String home() {
        return "hello World";
    }
}
```





## HANDS ON **SPRING BOOT** #EXEMPLO0

- Ao executar a classe do slide anterior, o webservice está disponível em:
  - <http://localhost:8080>



# HANDS ON **SPRING BOOT** #EXEMPLOO

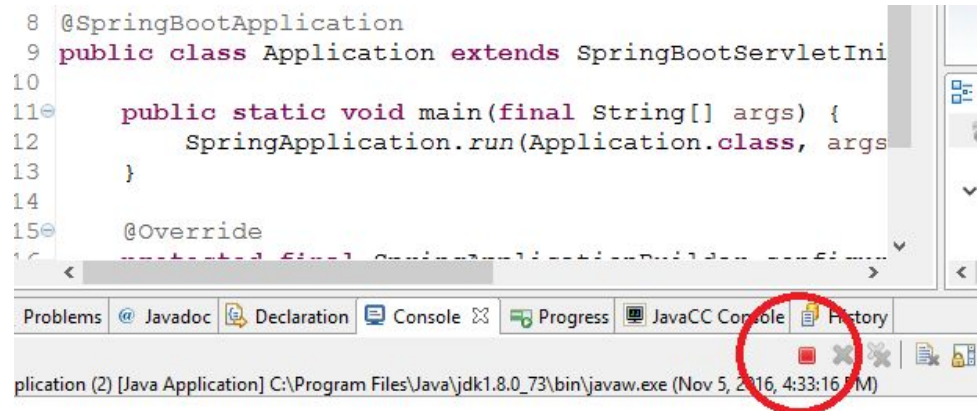
- **Common Problems and Solutions 1:** Caso não consiga executar adequadamente, possivelmente existe algum processo sendo executado na porta 8080 (Exception abaixo).
  - **Como resolver?**
    - Em **src/main/resources** crie um arquivo chamado **application.properties**
      - Insira a linha abaixo e seja feliz :)
        - `server.port=8082`
      - Agora está executando na 8082

```
Caused by: java.net.BindException: Address already in use: bind
    at sun.nio.ch.Net.bind0(Native Method) ~[na:1.8.0_73]
    at sun.nio.ch.Net.bind(Net.java:433) ~[na:1.8.0_73]
    at sun.nio.ch.Net.bind(Net.java:425) ~[na:1.8.0_73]
    at sun.nio.ch.ServerSocketChannelImpl.bind(ServerSocketChannelImpl.java:223) ~[na:1.8.0_73]
    at sun.nio.ch.ServerSocketAdaptor.bind(ServerSocketAdaptor.java:74) ~[na:1.8.0_73]
    at org.apache.tomcat.util.net.NioEndpoint.bind(NioEndpoint.java:340) ~[tomcat-embed-core-8.0.30.jar:8.0.30]
    at org.apache.tomcat.util.net.AbstractEndpoint.start(AbstractEndpoint.java:765) ~[tomcat-embed-core-8.0.30.jar:8.0.30]
    at org.apache.coyote.AbstractProtocol.start(AbstractProtocol.java:473) ~[tomcat-embed-core-8.0.30.jar:8.0.30]
    at org.apache.catalina.connector.Connector.startInternal(Connector.java:986) ~[tomcat-embed-core-8.0.30.jar:8.0.30]
    ... 14 common frames omitted
```



## HANDS ON SPRING BOOT #EXEMPLO0

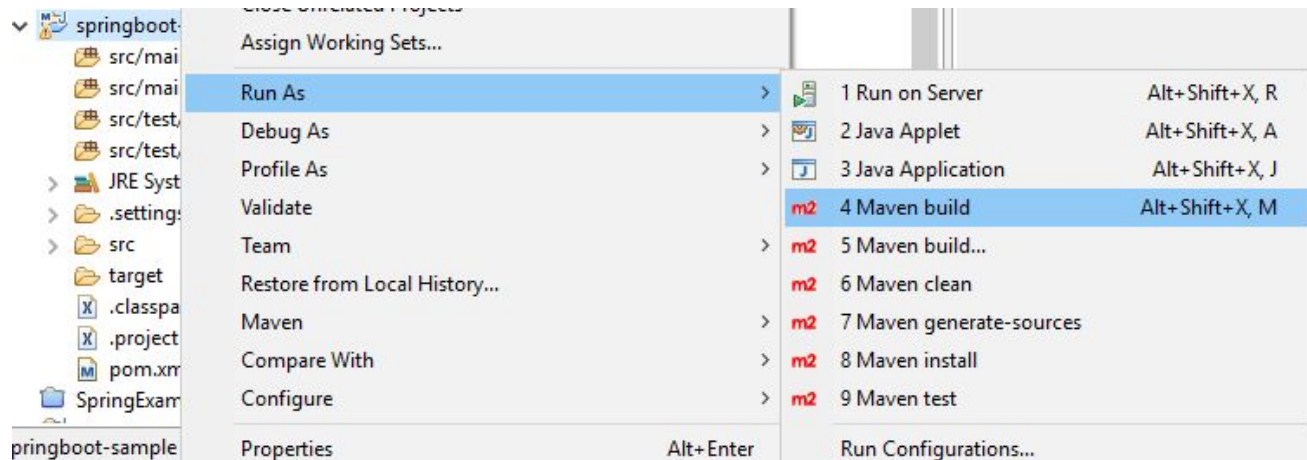
- **Common Problems and Solutions 2:** Executei a aplicação uma vez, esqueço de parar a aplicação e a executo novamente. Vai acontecer a mesma *exception* do slide anterior, porque a porta vai estar sendo ocupada pela primeira execução.
- **Como resolver?** Pare (destacado na imagem abaixo) a primeira execução da aplicação pelo amor de Deus!





# EMPACOTANDO APLICAÇÃO SPRING BOOT

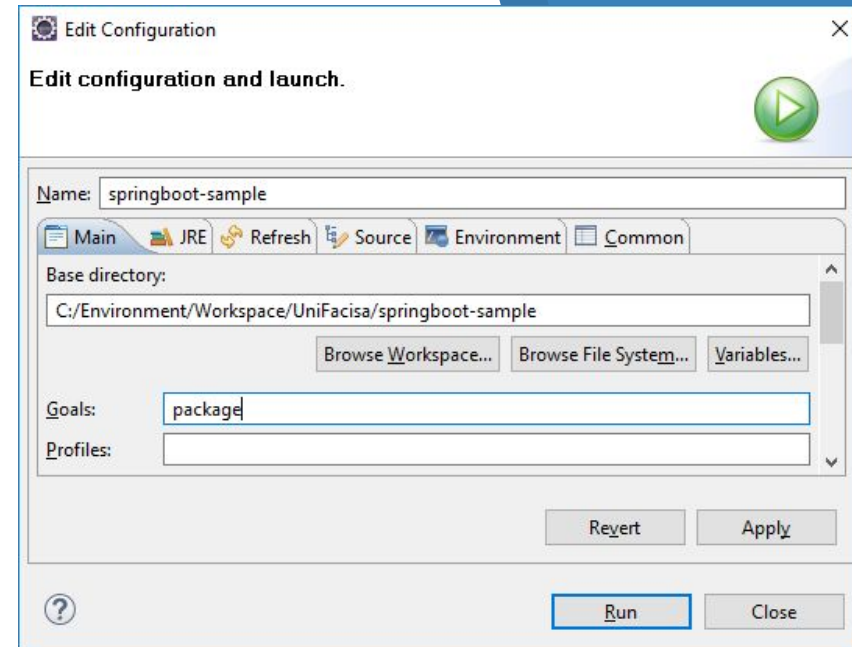
- Terminei minha aplicação Spring Boot, como gero o .jar para enviar para o meu cliente? E como ele vai executar lá? [1]
  - Clique no projeto com o botão direito
    - Selecione a opção *Run as* e depois *Maven configurations*





# EMPACOTANDO APLICAÇÃO SPRING BOOT

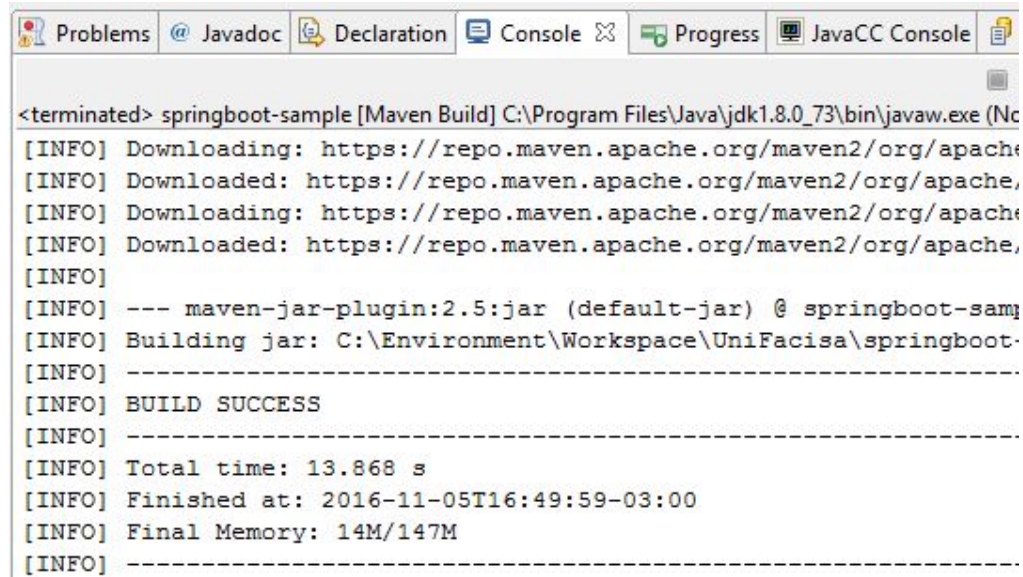
- Terminei minha aplicação Spring Boot, como gero o .jar para enviar para o meu cliente? E como ele vai executar lá? [2]
  - Na tela seguinte no campo goals escreva a palavra package
    - Indica que vamos empacotar a aplicação (gerar nosso jar)





# EMPACOTANDO APLICAÇÃO SPRING BOOT

- Terminei minha aplicação Spring Boot, como gero o .jar para enviar para o meu cliente? E como ele vai executar lá? [3]
  - Dando tudo certo, você terá um BUILD SUCCESS na View Console



```
<terminated> springboot-sample [Maven Build] C:\Program Files\Java\jdk1.8.0_73\bin\javaw.exe (Nc
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache,
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache,
[INFO]
[INFO] --- maven-jar-plugin:2.5:jar (default-jar) @ springboot-sam
[INFO] Building jar: C:\Environment\Workspace\UniFacisa\springboot-
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13.868 s
[INFO] Finished at: 2016-11-05T16:49:59-03:00
[INFO] Final Memory: 14M/147M
[INFO] -----
```



## EMPACOTANDO APLICAÇÃO **SPRING BOOT**

- *Terminei minha aplicação Spring Boot, como gero o .jar para enviar para o meu cliente? E como ele vai executar lá? [4]*
  - O seu .jar está no diretório *target* do projeto
  - Para executar, basta apenas executar no prompt:
    - `java -jar meujar.jar`
  - Simples assim!



# **HANDS ON SPRING BOOT**

**UMA ABORDAGEM  
PRÁTICA**







## HANDS ON SPRING BOOT #EXEMPLO1

- O projeto com o exemplo está disponível: **exemplo1**
- Basicamente o mesmo que o **exemplo0**, porém acessível em:
  - <http://localhost:8080/recurso/teste>

```
@Controller
@SpringBootApplication
@RequestMapping(value = "/recurso")
public class Application {

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

    @RequestMapping(value = "/teste", method = RequestMethod.GET)
    @ResponseBody
    public String home() {
        return "hello World";
    }
}
```

## 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=slrPZlvvTDw>
- 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** #EXEMPLO2

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

```
import java.io.Serializable;

@Entity
@Table
public class User implements Serializable {

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

    public User() {
    }

}
```



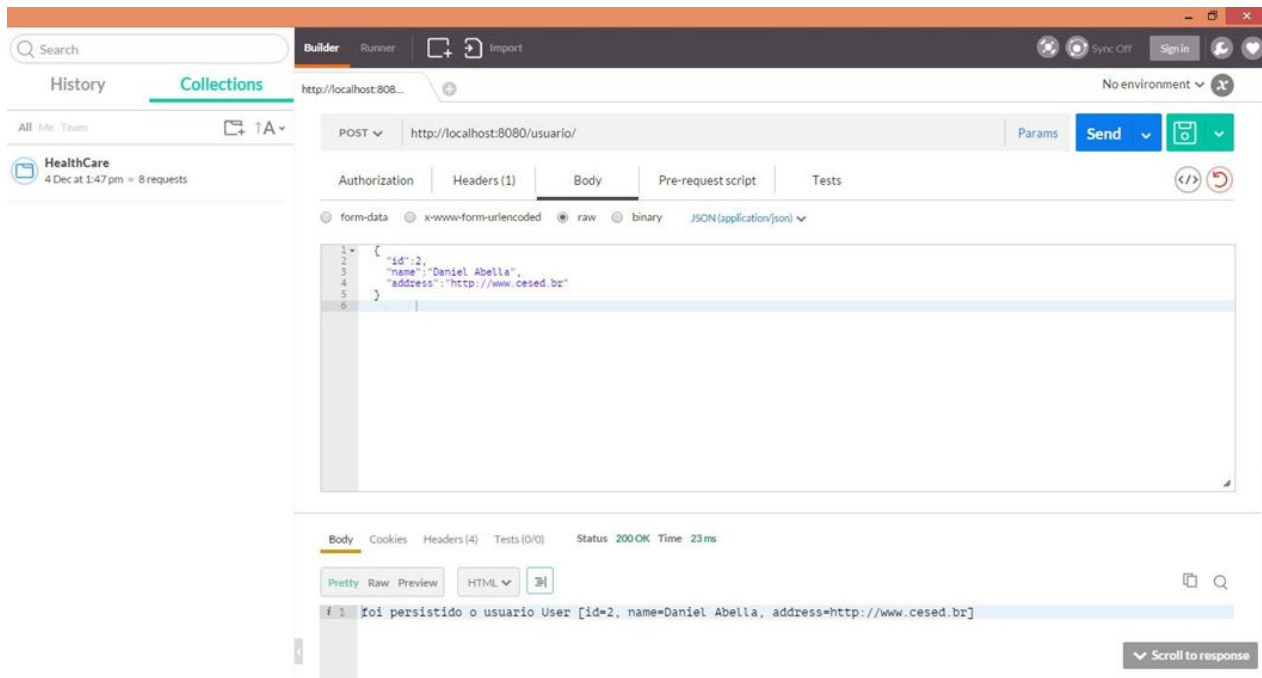
## HANDS ON SPRING BOOT #EXEMPLO2

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

```
public class UserController {  
  
    @RequestMapping(value="/usuario", method = RequestMethod.GET)  
    public String listarTodosUsuarios() {  
        return "todos";  
    }  
  
    @RequestMapping(value="/usuario/{identificador}", method = RequestMethod.GET)  
    public String obterInformacaoUsuario(@PathVariable(value="identificador") String id) {  
        return "Ola, Meu Amigo Desenrolado que possui o ID = " + id;  
    }  
  
    @RequestMapping(value="/usuario", method = RequestMethod.POST)  
    public String createParticipant(@RequestBody User user) {  
  
        try {  
            System.out.println(user);  
            return "foi persistido o usuario " + user;  
        } catch (Exception e) {  
            return "problema";  
        }  
    }  
}
```

# TESTE POSTMAN #EXEMPLO2

- Vamos fazer um POST





## HANDS ON SPRING BOOT #EXEMPLO3

- Vamos evoluir o **exemplo2** 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);
    }
}
```

- O código segue ainda no próximo *slide*



## HANDS ON SPRING BOOT #EXEMPLO3

- Evoluindo **exemplo2** apresentado anteriormente (continuação)

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

    User userFake = new User(1, "Daniel", "End1");

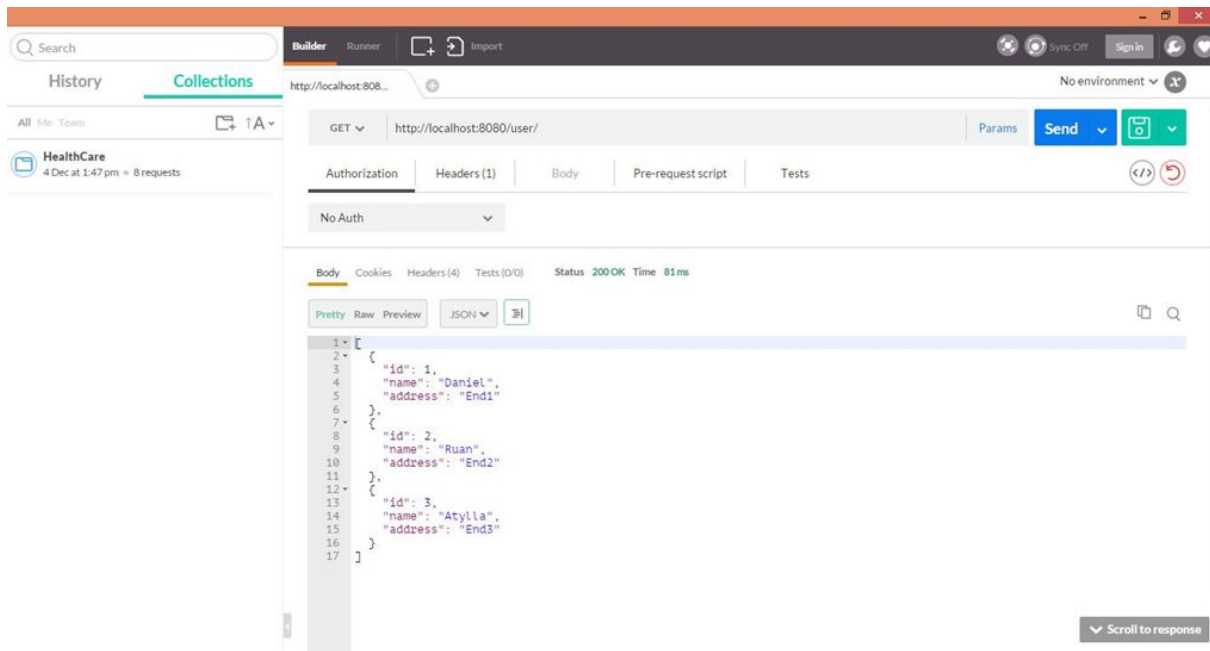
    return userFake == null ?
        new ResponseEntity<User>(HttpStatus.NOT_FOUND) :
        new ResponseEntity<User>(userFake, HttpStatus.OK);
}
```

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

    try {
        //salvar
        return new ResponseEntity<String>(HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<String>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

# TESTE POSTMAN #EXEMPLO3

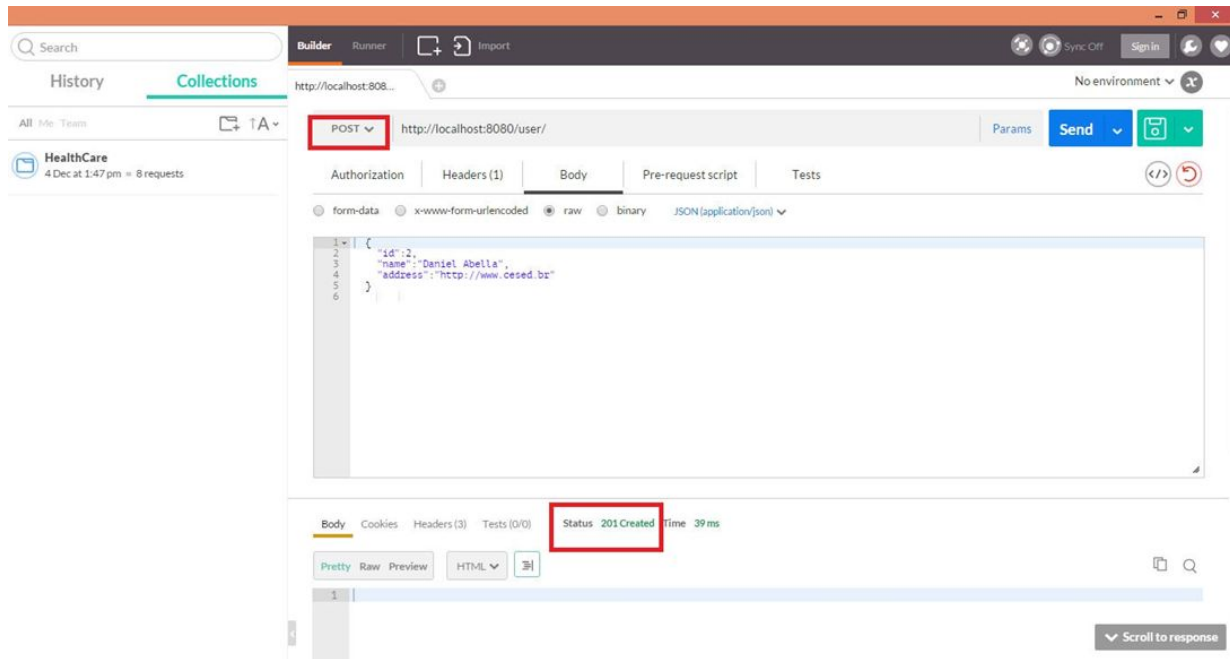
- Vamos fazer um GET





# TESTE POSTMAN #EXEMPLO3

- Vamos fazer um POST





## HANDS ON **SPRING BOOT** #EXEMPLO4

- Vamos evoluir o **exemplo3** apresentado anteriormente
  - Vamos receber parâmetros e variáveis
  - Lembra que o GET você poderia passar uma variável?
    - localhost:8080/usuario/123 (onde 123 era o id)



## HANDS ON SPRING BOOT #EXEMPLO4

- Vamos evoluir o **exemplo3** apresentado anteriormente
  - Agora, quando chamamos localhost:8080/usuario/daniel
    - Temos os dados de Daniel

```
@RequestMapping(value="/app/usuario/{nome}", method = RequestMethod.GET)  
public String olaMundoPersonalizado(@PathVariable String nome) {  
    return "Nome inserido " + nome;  
}
```



## HANDS ON SPRING BOOT #EXEMPLO4

- Vamos evoluir o **exemplo3** apresentado anteriormente
  - Podemos usar RequestParams
    - Mas não invente de passar senha e usuários por aí

```
@RequestMapping(value="/v2", method = RequestMethod.GET)
public String olaMundoPersonalizadoV2(
    @RequestParam(value="nome",defaultValue="nome padrão") String nome,
    @RequestParam(value="sobrenome",defaultValue="sobrenome padrão") String sobrenm) {

    return "Olá " + nome + " " + sobrenm;
}
```

`http://localhost:9094/v2?nome=daniel&sobrenome=abella`

# RELEMBRANDO

JAVA PERSISTENCE API

PARA INTEGRARMOS  
AO NOSSO PROJETO  
SPRING BOOT



# CONCEITO **MAPEAMENTO OBJETO-RELACIONAL**

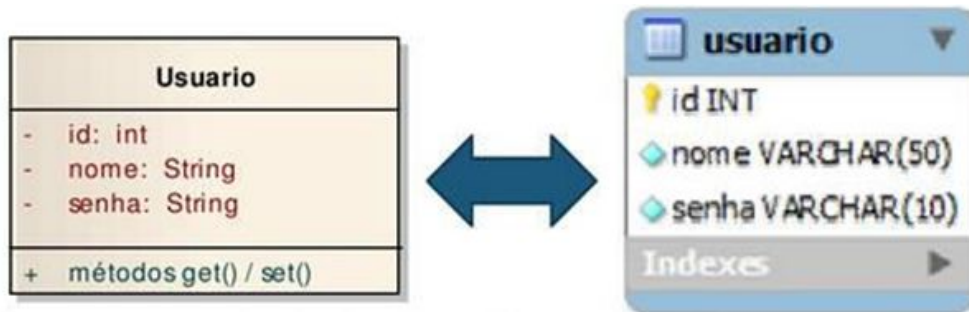


- **Cenário**
  - Perda de tempo na codificação de queries SQL
- **Problema**
  - Apesar do padrão ANSI, possui diferenças significativas entre fabricantes
  - Diferença da POO com o Modelo Relacional



## 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)
- Associação do Objeto
  - Equivalente a: Foreign Key (SQL)





## MAPEANDO UMA ENTIDADE

```
@Entity
@Table(name="produto")
public class Produto implements Serializable{
    @Id
    @GeneratedValue
    private int id;
    @Column(name="nome")
    private String nome;
    @Column(name="descricao")
    private String descricao;
    @Column(name="valor_unitario")
    private float precoUnitario;
```





## MAPEANDO UMA ENTIDADE

**@Column**(name="nome", lenght=10)

- Validação do tamanho do campo

**@Column**(name="senha", nullable=false)

- Validação de campo NOT NULL

**@Column**(name="valor", precision=5, scale=2)

- precision: quantidade de dígitos antes da vírgula
- scale: quantidade de dígitos após a vírgula

**@Column**(name="qualquer", insertable=false, updatable=false)

- Restringe inserção e atualização do campo

# HANDS ON SPRING BOOT

UMA ABORDAGEM  
**PRÁTICA** (RETORNO)





# HANDS ON **SPRING BOOT** #EXEMPLO5

- Vamos entender objetivamente o **exemplo5**
  - **Passo 1:** Mapeando a entidade User

```
import java.io.Serializable;

@Entity
@Table
public class User implements Serializable {

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

    public User() {
    }
}
```



## HANDS ON **SPRING BOOT** #EXEMPLO5

- Vamos entender objetivamente o **exemplo5**
  - **Passo 2:** Criando o Repository
- Repository é um padrão apresentado por Martin Fowler no livro *Patterns of Enterprise Application Architecture*
  - **Objetivo:** Serve de abstração para a persistência dos dados
  - Código apresentado no próximo slide





## HANDS ON SPRING BOOT #EXEMPLO5

- Vamos entender objetivamente o **exemplo5**
  - **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 #EXEMPLO5

- Vamos entender objetivamente o **exemplo5**
  - **Passo 3:** Criando a classe Service
    - Abriga as regras de negócio

```
@Service
@Validated
public class UserService {

    @Autowired
    private UserRepository repository;

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

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



## HANDS ON SPRING BOOT #EXEMPLO5

- Vamos entender objetivamente o **exemplo5**
  - **Passo 3:** Criando a classe Service
    - Abriga as regras de negócio
    - O que necessita de transação, anotar com @Transactional

```
@Transactional
public User save(@NotNull User usuario) {

    User existing = repository.findOne(usuario.getId());

    if(existing == null)
        existing = repository.save(usuario);

    return existing;
}

public UserRepository getRepository() {
    return repository;
}
}
```



# HANDS ON **SPRING BOOT** #EXEMPLO5

- Vamos entender objetivamente o **exemplo5**
  - **Passo 4:** Criando o Controller
    - Classe que recebe as requisições HTTP (métodos de *callback*)

```
@RestController
public class UserController {

    private final UserService participantService;

    @Autowired
    public UserController(final UserService participantService) {
        this.participantService = participantService;
    }

    @RequestMapping(value="/user", method = RequestMethod.GET)
    public ResponseEntity< List<User> > listAllParticipants() {
        return new ResponseEntity< List<User> >
            (participantService.listAllParticipants(), HttpStatus.OK);
    }
}
```



## TESTE **POSTMAN** #EXEMPLO5

- Vamos fazer testes com os métodos expostos





## HANDS ON **SPRING BOOT** #EXEMPLO5

- Vamos aprimorar o Repository do **exemplo5**?
  - 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);  
}
```

# HANDS ON SPRING BOOT

COLOCANDO  
MONGODB **NA**  
**DISPUTA**





## HANDS ON **SPRING BOOT** #EXEMPLO7

- A mudança é muito simples, são três passos:
  - **Passo 1:** Adicionar novas dependências no pom.xml
    - A primeira delas é o módulo do spring boot para trabalhar com mongodb
    - A segunda delas é o mongo embarcado (embedded)
      - Ela se torna dispensável quando você possui o MongoDB disponível

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

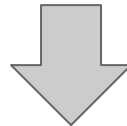
```
<dependency>
  <groupId>de.flapdoodle.embed</groupId>
  <artifactId>de.flapdoodle.embed.mongo</artifactId>
</dependency>
```



## HANDS ON SPRING BOOT #EXEMPLO7

- A mudança é muito simples, são dois passos:
  - **Passo 2:** Mudar o nosso Repository
    - Não estendemos *JpaRepository*, mas *MongoRepository*

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



```
public interface UserRepository extends MongoRepository<User, String> {  
    public User findByName(String name);  
}
```



## HANDS ON **SPRING BOOT** #EXEMPLO7

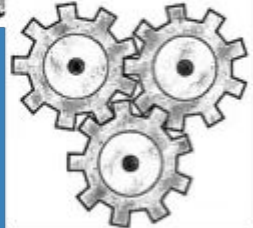
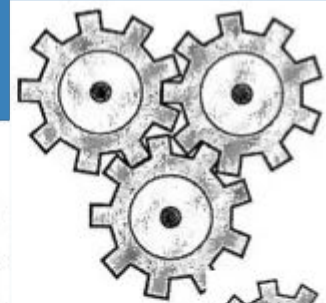
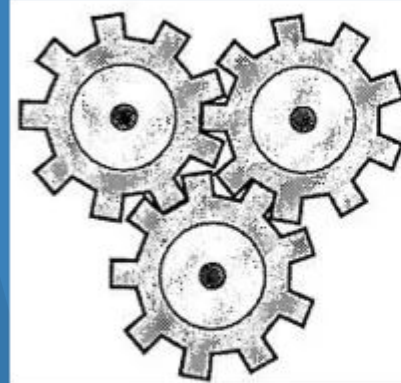
- A mudança é muito simples, são dois passos:
  - **Passo 3:** O import da annotation @Id é modificado
    - Antes importávamos @Id do pacote javax.persistence (JPA)
    - Agora importamos de org.springframework.data.annotation (Spring)

- **Simples assim!**



# MICROSERVICES ARCHITECTURE

**MICRO**



**SERVICES**

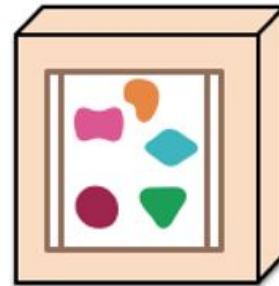


# ARQUITETURA MONOLÍTICA

- É desenvolvido em um único módulo;
- O módulo está incluso a interface com o usuário, as regras de negócio, persistência de dados e comunicações com outros sistemas;
- Componente do núcleo podem acessar os demais componentes por estarem no mesmo módulo.



Funcionalidades em um único processo...



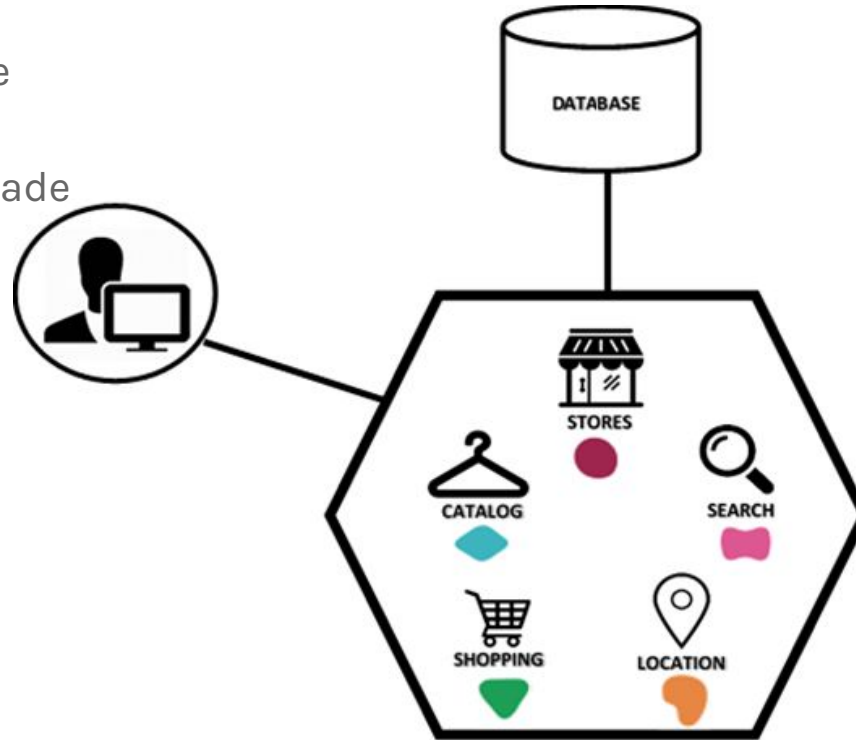
...em uma única Instância de servidor...





# LIMITAÇÕES DE ARQUITETURA MONOLÍTICA

- Escalabilidade
- Acoplamento
- Manutenibilidade



Monolithic  
Architecture

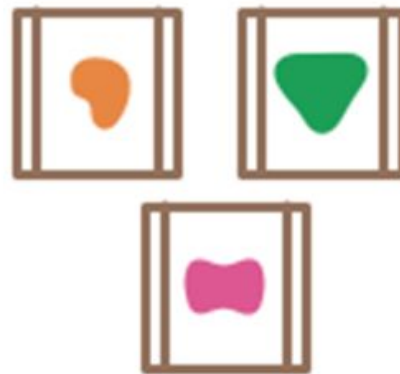


# ARQUITETURA **MICROSERVICES**

- Desenvolver aplicações de software em um conjuntos de serviços.
  - Independente
  - Implementável
  - Escalável



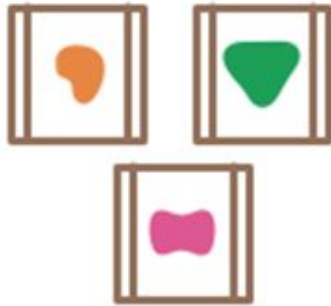
**Martin Fowler**



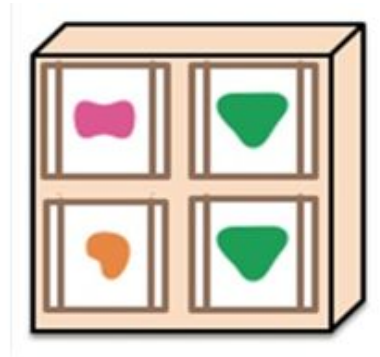


# ARQUITETURA **MICROSERVICES**

- Realiza a segmentação de uma parte específica ou de toda aplicação;
- Usando uma abordagem em um conjunto de serviços;
- Cada um é executado em seu próprio processo.



Cada serviço em seu processo...

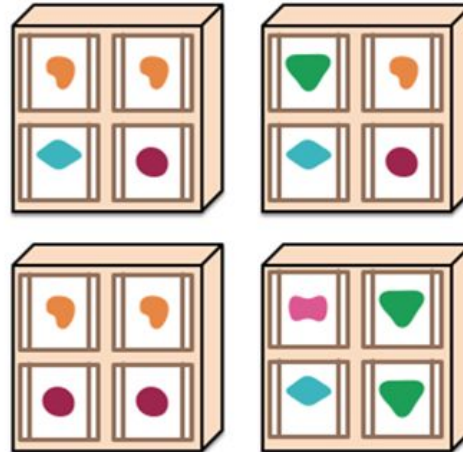
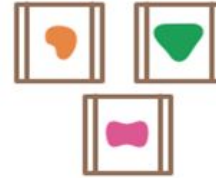


...Instância de servidor com os serviços...

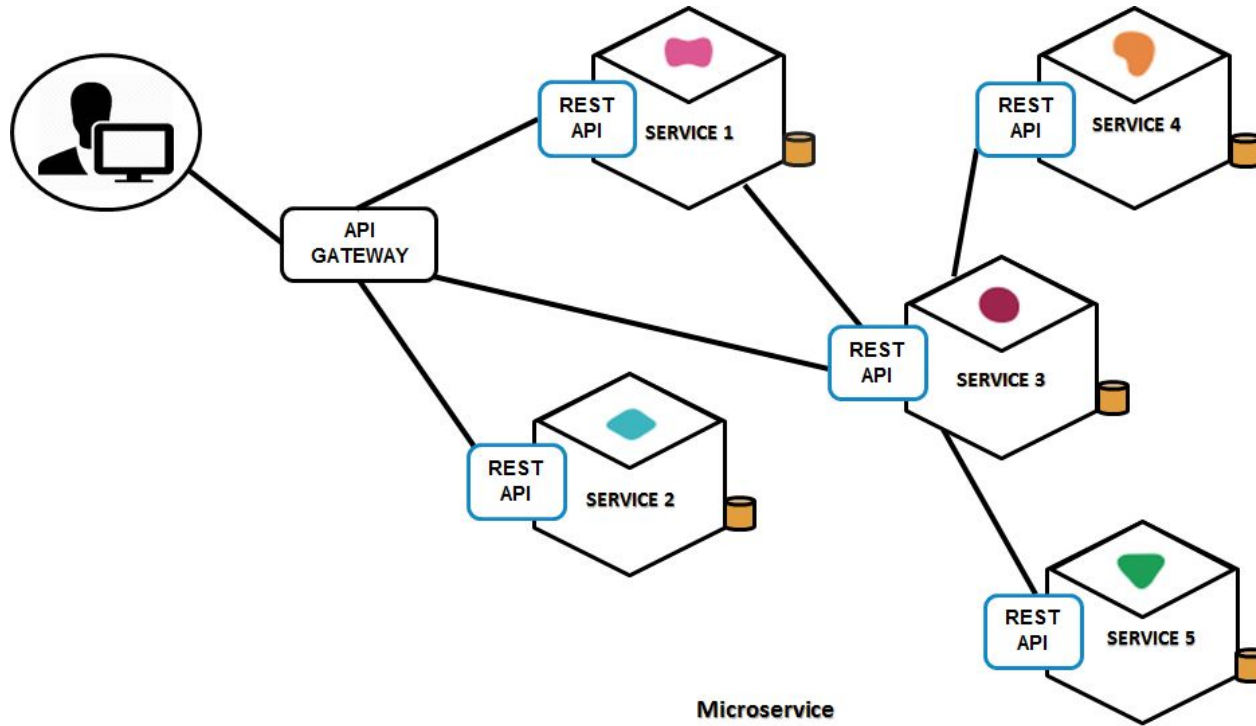


# ARQUITETURA **MICROSERVICES**

- Vantagens
  - Isolamento
  - Autonomia
  - Escalabilidade
  - Serviço único
  - Manutenção



# ARQUITETURA **MICROSERVICES**



Microservice  
Architecture



# 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>



**OBRIGADO!**

Dúvidas?

daniel@daniel-abella.com

www.daniel-abella.com