



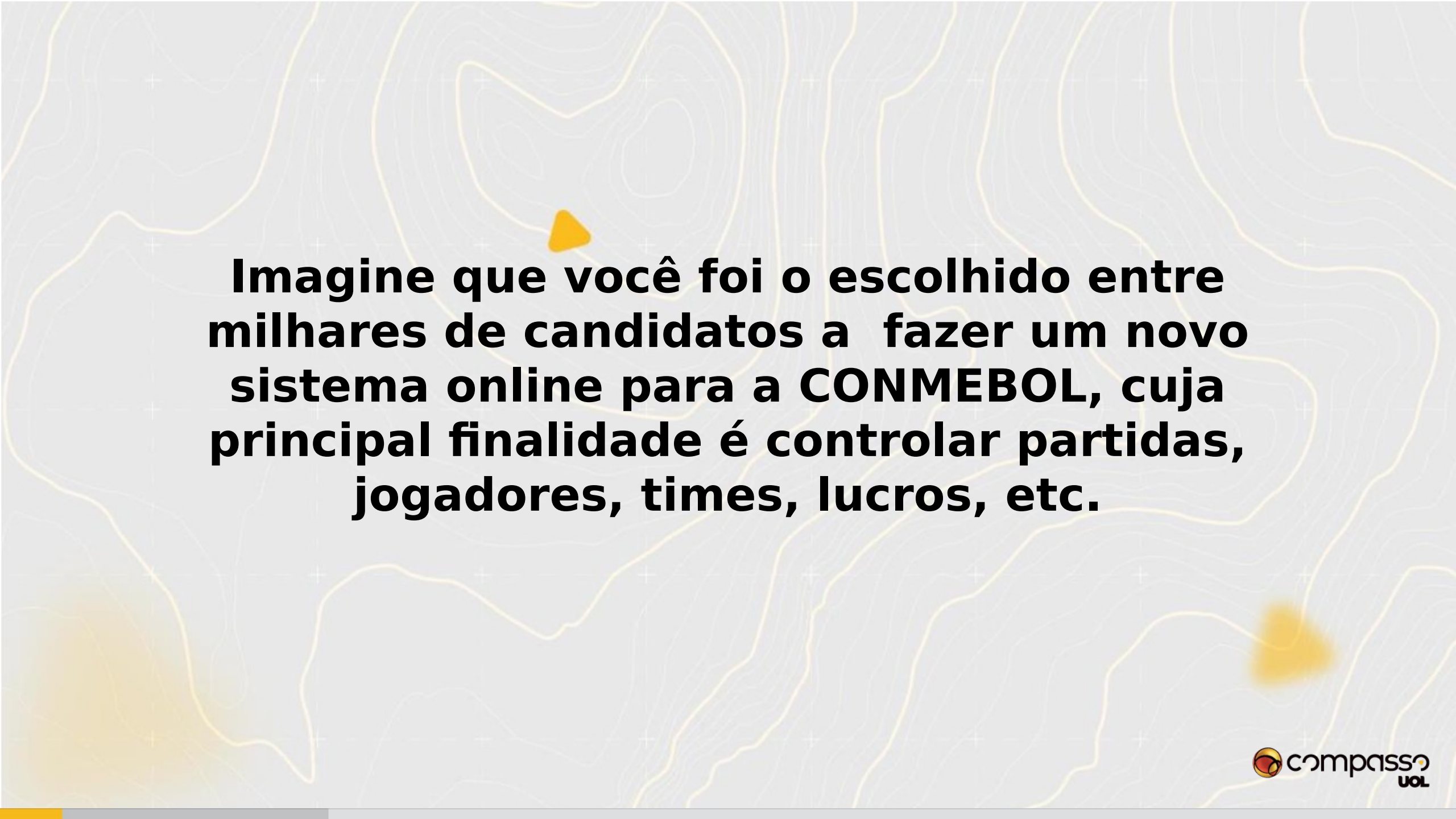
# API Rest: Uma Abordagem com Spring Boot

O que envolve o desenvolvimento de uma API Rest em Spring Boot?

DAIONE CAROLINE BALDUZZI PAVAN/ Dev. Java Jr.

04/2020





**Imagine que você foi o escolhido entre milhares de candidatos a fazer um novo sistema online para a CONMEBOL, cuja principal finalidade é controlar partidas, jogadores, times, lucros, etc.**





# O que é uma API?



**É o acrônimo de *Application Program Interface* ou *API*; é uma interface que estabelece rotinas e padrões de comunicação entre dois ou mais serviços. Os serviços são ações que podem ser executadas.**

Como estamos pensando em um aplicativo para a Conmebol, uma das primeiras funcionalidades que o aplicativo deve ter é a inserção de dados dos jogadores.

Nome	<input type="text"/>
Idade	<input type="text"/>
CPF	<input type="text"/>
Endereço	<input type="text"/>
	<input type="button" value="Enviar"/> <input type="button" value="Limpar"/>

Envia dados

API

## Example API <sup>v1</sup>

[/swagger/v1/swagger.json](#)

### Values

GET /api/Values

GET /api/Get

POST /api/Post

PUT /api/Put

DELETE /api/Delete

**A API pode respeitar um padrão que conhecemos por REST (*Representational State Transfer*). Os princípios do padrão Rest são:**

## Uniform Interface

Uma API deve ser desenvolvida, levando-se em consideração que um sistema é um sistema complexo, que está em constante alteração e que precisa ser manutenível.

Precisa fazer sentido para quem for o cliente que vai consumi-la. Por isso, o uso dos verbos usados nos momentos corretos, utilizar os nomes dos endpoints no plural, etc.

Configuram - se uma boa prática no desenvolvimento

## Client Server

**Separação Cliente/Servidor.** Isso ajuda se acaso a pessoa quiser fazer uma aplicação WEB, disponibilizando a versão Mobile para quem desejar.

## Stateless

O cliente faz a requisição e durante o seu processamento a API não guarda informação alguma. Se a pessoa loga em uma conta, acaba gerando um token de autenticação e através desse token (armazenado, muitas vezes no storage do navegador), é possível validar se o usuário tem permissão de acesso a determinada

## Cacheable

“ As respostas para uma requisição, deverão ser explícitas ao dizer se aquela requisição, pode ou não ser cacheada pelo cliente.”  
(Rocketseat)

## Layered System

O cliente não precisa saber como as requisições são feitas, mas precisa saber que aquela camada de sistema vai processar a requisição recebida.

## Code on Demand

O servidor envia um script pra ser rodado no lado do cliente.



GET -> Requisição que pede para o servidor carregar algum dado.

POST -> Requisição que insere um novo dado

PUT -> Requisição que atualiza um determinado dado;

DELETE - > Requisição de exclusão de um dado.

# STATUS CODE

## **100-> Respostas de informação**

100 (Continue): Essa resposta provisória indica que tudo ocorreu bem até agora e que o cliente deve continuar com a requisição ou ignorar se já concluiu o que gostaria (MDN Web Docs).

## **200-> Respostas de sucesso**

200(OK): Informa que a requisição foi enviada com sucesso.

201 (Created): Avisa que a solicitação de salvar um novo registro, foi realizada com sucesso.

204(No Content): Retorna a informação de que a requisição foi processada, porém não possui corpo de retorno.

## **300-> Mensagens de Redirecionamento.**

308: Permanent Redirect

307: Temporary Redirect

# STATUS CODE

## **400-> Respostas de Erro no lado do cliente.**

400 (Bad Request): Erro de sintaxe faz com que o servidor não entenda a requisição.

401(Unauthorized): O cliente não está autenticado na aplicação para realizar uma requisição.

402 (Payment Required): É preciso pagar para acessar tal conteúdo.

403 (Forbidden): O famoso Forbidden, (ou o status da tristeza) ocorre quando o cliente é autenticado, mas acessa um recurso da api ao qual não está autorizado a acessar.

404 (Not Found): O Servidor não conseguiu encontrar o que o cliente requisitou.

405 (Method Not Allowed); ocorre quando o servidor tenta acessar um endpoint que ainda não foi implementado.

## **500-> Respostas de Erro no lado do servidor**

500 (Internal Sever Error): “O servidor encontrou uma situação com o qual não sabe lidar” (MDN Web Docs).

501: O método da requisição não é suportado pela aplicação.

<https://http.cat/>



## Conceitos e dicas de boas práticas.

EndPoint -> É o endereço pelo qual você vai acessar a requisição que precisa acionar.

Ex.:

<http://localhost:8080/jogadores>

Resource -> É uma entidade que tem dados associados a ela e relacionamentos entre outros resources. O resource é aquela parte da URL em que tu tem o nome do objeto que vai ser processado dentro do .

JSON -> É o modo como as requisições e as respostas trafegam do servidor para o cliente ou do cliente para o servidor.

RestFull-> É a aplicação correta dos princípios do padrão Rest.



# Design Patterns? O que são?



# Design Pattern

Os *Design Patterns* são soluções prontas e testadas para problemas que ocorrem no momento de desenvolver um software. Mas um design pattern seria um código pronto, ou uma espécie de framework? Nada disso, são modelos de solução que não têm relação direta com a programação em si, mas com a arquitetura e a engenharia de uma solução.

O que você ganha ao usar um *Design Pattern*?

1. Ganho de produtividade
2. Organização e manutenção
3. Facilidade em tomar decisões técnicas.



# Design Patterns



## Creational

Como o próprio nome já sugere, é um padrão que foca na criação de objetos. Um exemplo é o padrão *Builder*, onde se usa uma classe que constrói um objeto de outra classe.

## Structural

Como juntar vários objetos para criar uma funcionalidade? A ideia do estrutural é conseguir ordenar da melhor forma possível vários objetos que, ao fim ao cabo, executam uma mesma atividade. (Facade )

## Behavioral

Trata como os objetos se comunicam entre si. É mais próximo do algoritmo. *Strategy*, *Observable* (conectar um conjunto de objetos a um objeto central para que avisa quando um componente desses objetos foi modificado - usado em promisses).

# Design Pattern

## Builder

```
public class Jogador {  
    private int id;  
    private String nome;  
    private int idade;  
    private String cpf;  
    private String email;  
    private String endereco;  
    private String nomeDaMae;  
    private String cpfResponsavel;
```

```
public class Application {  
    public static void main(String[] args) {  
        JogadorBuilder jogador01 = new JogadorBuilder().addCpf("um cpf")  
            .addEmail("um email").addEndereco("um endereco")  
            .addId(1).addIdade(24)  
            .addNome("Um nome");  
  
        JogadorBuilder jogador02 = new JogadorBuilder().addCpf("um cpf2")  
            .addEmail("um email2").addEndereco("um endereco2")  
            .addId(1).addIdade(24)  
            .addNome("Joao");  
  
        System.out.println("JOGADOR 01: " + jogador01.toString());  
        System.out.println(".....");  
        System.out.println("JOGADOR 02: " + jogador02.toString());  
    }  
}
```

```
public class JogadorBuilder{  
    private Jogador jogador;  
  
    public JogadorBuilder() {this.jogador = new Jogador(); }  
  
    public Jogador getJogador() { return jogador; }  
  
    public void limpar() { this.jogador = new Jogador(); }  
  
    public JogadorBuilder addId(int id) { this.jogador.setId(id); return this;}  
  
    public JogadorBuilder addNome(String nome) { this.jogador.setNome(nome); return this; }  
  
    public JogadorBuilder addIdade(int id) { this.jogador.setId(id); return this; }  
  
    public JogadorBuilder addCpf(String cpf) { this.jogador.setCpf(cpf); return this; }  
  
    public JogadorBuilder addEmail(String email) { this.jogador.setEmail(email); return this; }  
  
    public JogadorBuilder addEndereco(String endereco) { this.jogador.setEndereco(endereco); return this; }  
  
    public Jogador resultado(){ return this.getJogador(); }  
  
    @Override  
    public String toString(){ return this.getJogador().toString(); }
```

# Design Pattern

## Interpreter

```
package br.com.matheuscastiglioni.interpreter;

public class Numero implements Operador {

    private int numero;

    public Numero(int numero) {
        this.numero = numero;
    }

    @Override
    public int interpretar() {
        return this.numero;
    }
}
```

```
package br.com.matheuscastiglioni.interpreter;

public interface Operador {

    int interpretar();

}
```

```
public class Somar implements Operador {

    private Numero esquerda;
    private Numero direita;

    public Somar(Numero esquerda, Numero direita) {
        this.esquerda = esquerda;
        this.direita = direita;
    }

    @Override
    public int interpretar() {
        return this.esquerda + this.direita;
    }
}
```



# Spring Boot na prática



# Ponderações Iniciais





# Application.properties X Application.yml

Qual usar?

```
TreinamentoApplication.java × application.properties ×
1
2 spring.datasource.url= mongodb://localhost/conmebol
3 spring.data.mongodb.port=27017
4 spring.data.mongodb.authentication-database=None
5 define.url=http://localhost:8080/api
6 #spring.data.mongodb.username=[username]
7 #spring.data.mongodb.password=[password]debol
8
```

```
TreinamentoApplication.java × application.properties × application.yml ×
1 spring:
2   datasource:
3     className: com.mysql.jdbc.Driver
4     password:
5     platform: mysql
6     url: jdbc:mysql://localhost/testando_teste?createDatabaseIfNotExist=true&useTimezone=true&serverTimezone=UTC
7     username: root
8   jpa:
9     database-platform: org.hibernate.dialect.MariaDBDialect
10    generate-ddl: false
11    hibernate:
12      ddl-auto: update
13    show-sql: true
14  application:
15    name: product-serve
16
17
18 eureka:
19   client:
20     service-url:
21       default-zone: ${EUREKA_URI:http://localhost:8761}
22   instance:
23     prefer-ip-address: true
24
25 server:
26   port: 8082
```



# Hibernate

O **Hibernate** é um *framework* para o [mapeamento objeto-relacional](#) escrito na linguagem [Java](#), mas também é disponível em [.Net](#) com o nome [NHibernate](#). Este framework facilita o mapeamento dos atributos entre uma base tradicional de dados relacionais e o modelo objeto de uma aplicação, mediante o uso de arquivos ([XML](#)) ou anotações Java (veja [Annotation \(java\)](#)) (Wikipédia).

## Algumas anotações

@Transient: Essa anotação expressa que o atributo da entidade não será salvo em banco.

@Entity: Denota que aquela classe vai corresponder a uma tabela no banco de dados.

@Table Anotação que guarda as propriedades das tabelas.

@OneToOne @ManyToMany e @ManyToOne são anotações que especificam o tipo de relação que as tabelas terão quando geradas pelo Spring.

```
@OneToOne
private Logistica logisticaVolta;

@ManyToOne
private Evento evento;

@ManyToOne
private Camiseta camiseta;

@ManyToMany
private List<Restricao> restricoes;

@Column(unique = true)
private String email;
```

```
@Column(name="RETORNO_ANTECIPADO")
private boolean retorno;
```

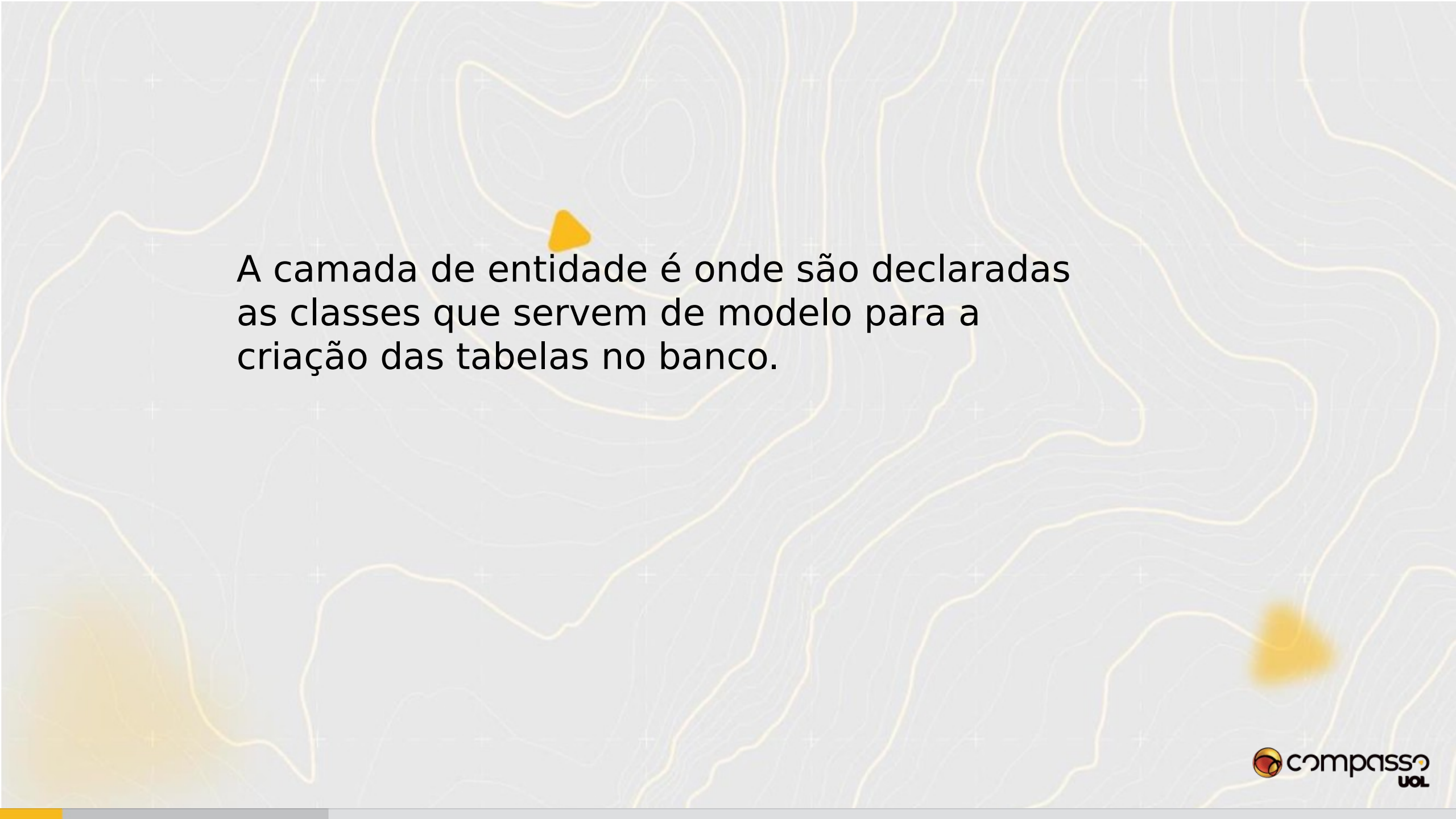
```
private String nome;
@Transient
private int idade;
```

```
@Entity
@Table(name= "resposta")
public class Resposta {
```



# Entidades





A camada de entidade é onde são declaradas as classes que servem de modelo para a criação das tabelas no banco.



# Alguns exemplos de autoincremento

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
@Id  
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator="PRODU_SEQ")  
@SequenceGenerator(sequenceName = "product_sequence", allocationSize = 1, name= "PRODU_SEQ")
```

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private int id;  
private String nome;  
@OneToMany(cascade = CascadeType.REMOVE)  
private List<JogadorEntidade> jogadores;
```

## Cascade

1. **CascadeType.PERSIST** : cascade type `persist` means that `save()` or `persist()` operations cascade to related entities.
2. **CascadeType.MERGE** : cascade type `merge` means that related entities are merged when the owning entity is merged.
3. **CascadeType.REFRESH** : cascade type `refresh` does the same thing for the `refresh()` operation.
4. **CascadeType.REMOVE** : cascade type `remove` removes all related entities association with this setting when the owning entity is deleted.
5. **CascadeType.DETACH** : cascade type `detach` detaches all related entities if a "manual detach" occurs.
6. **CascadeType.ALL** : cascade type `all` is shorthand for all of the above cascade operations.

# Propriedades

```
import javax.persistence.*;
import java.util.Date;

@Entity
@Table(name= "jogadores")
public class JogadorEntidade {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(name = "jogador", nullable = false, unique = true)
    private String nome;
    @Column(name="data_nascimento", nullable = false)
    private Date nascimento;
    @Column(unique = true, nullable =false)
    private String email;

    public int getId() {
        return id;
    }
}
```



# Repository

Finalmente Spring Boot!





# Repository

A camada de repositório (Repository) está relacionada com a consulta, inserção, deleção e edição de dados no banco. Isto é, está ligada diretamente à manutenção da base de dados. Quando você está em um banco de dados, buscando informações, geralmente você escreve a instrução SQL e executa o comando. As interfaces fazem isso no mundo do Spring Boot, e também lançam exceções relacionadas ao banco.

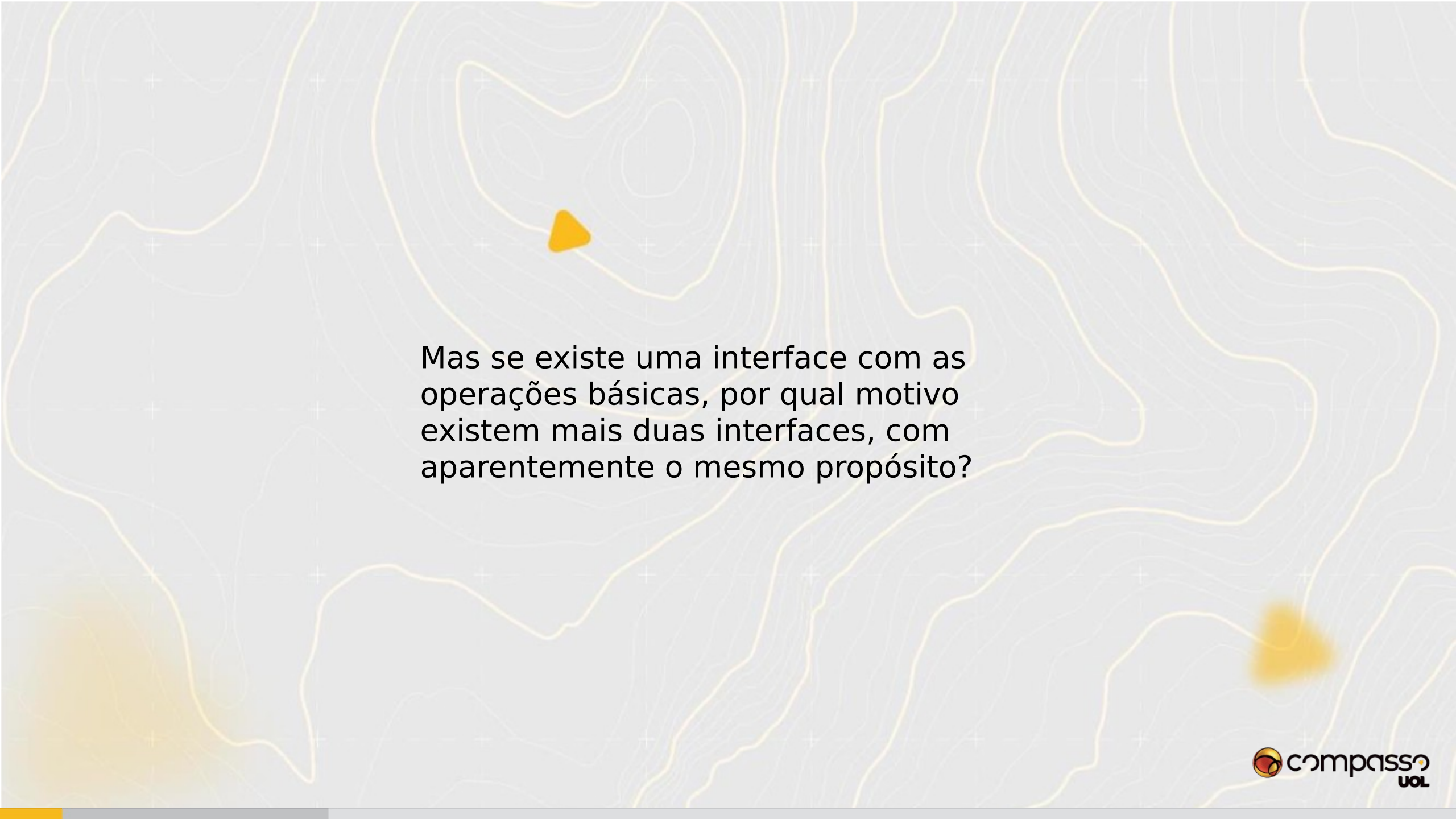
# Tipos de Repository

## CrudRepository

```
public interface JogadorRepository extends CrudRepository<JogadorEntidade, Integer> {  
}
```

```
package org.springframework.data.repository;  
  
import java.util.Optional;  
  
@NoRepositoryBean  
public interface CrudRepository<T, ID> extends Repository<T, ID> {  
    <S extends T> S save(S var1);  
  
    <S extends T> Iterable<S> saveAll(Iterable<S> var1);  
  
    Optional<T> findById(ID var1);  
  
    boolean existsById(ID var1);  
  
    Iterable<T> findAll();  
  
    Iterable<T> findAllById(Iterable<ID> var1);  
  
    long count();  
  
    void deleteById(ID var1);  
  
    void delete(T var1);  
  
    void deleteAll(Iterable<? extends T> var1);  
  
    void deleteAll();  
}
```

A CrudRepository tem em sua abstração as requisições básicas para execução de determinadas ações.



Mas se existe uma interface com as operações básicas, por qual motivo existem mais duas interfaces, com aparentemente o mesmo propósito?



# PagingAndSortingRepository

```
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {  
    Iterable<T> findAll(Sort var1);  
  
    Page<T> findAll(Pageable var1);  
}
```

Ao fazer com que a nossa repository estenda a interface PagingAndSortingRepository, estamos dizendo para a nossa aplicação que ela pode fazer uma busca paginada e adotar parâmetros básicos para que isso ocorra. A PagingAndSortingRepository estende a CrudRepository, possibilitando o acesso ao crud básico.

```
Sort sort = new Sort(new Sort.Order(Direction.ASC, "lastName"));  
Pageable pageable = new PageRequest(0, 5, sort);
```



Então não precisamos de mais  
nenhuma interface repository, né???

# Errado! JpaRepository

```
public interface JogadorRepository extends JpaRepository<JogadorEntidade, Integer> {  
}
```

```
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T> {  
    List<T> findAll();  
  
    List<T> findAll(Sort var1);  
  
    List<T> findById(Iterable<ID> var1);  
  
    <S extends T> List<S> saveAll(Iterable<S> var1);  
  
    void flush();  
  
    <S extends T> S saveAndFlush(S var1);  
  
    void deleteInBatch(Iterable<T> var1);  
  
    void deleteAllInBatch();  
  
    T getOne(ID var1);  
  
    <S extends T> List<S> findAll(Example<S> var1);  
  
    <S extends T> List<S> findAll(Example<S> var1, Sort var2);  
}
```

A JPA Repository serve para acessar todos os outros que as outras interfaces têm disponíveis (porque estende a PagingAndSortingRepository) e possui algumas funcionalidades a mais, como por exemplo o deleteAllInBatch() ou o famoso “delete from jogador” que exclui todos os dados da tabela.

Outro ponto importante é que a partir de seu uso, temos como retorno uma list quando chamamos o findAll, e não mais um Iterable.



Então não precisamos de mais nenhuma repository né?

```
💡  
public interface JogadorRepository extends ReactiveCrudRepository<JogadorEntidade, String> {  
}
```

```
*/  
interface ConferenceRepository extends ElasticsearchRepository<Conference, String> {}
```

```
public interface PlaylistRepository extends ReactiveMongoRepository<Playlist, String>{  
    Mono<Playlist> findByNome(Mono<String> nome);  
}
```



# Classes Request, Response, DTO ou Model



São classes responsáveis por fazer a transmissão da informação, sem expor a entidade. Isso significa que se você resolver usar a entidade para transitar os dados da controller para a sua repository, você estará sujeitando a sua aplicação a problemas sérios de segurança. Além do mais essas classes permitem uma modelagem diferenciada dos dados de resposta e requisição. Por esses motivos, pense com carinho na possibilidade de uso desse tipo de recurso.



```
@Entity
@Table(name = "notification")
public class NotificationEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "NOTIF_SEQ")
    @SequenceGenerator(sequenceName = "notification_seq", allocationSize = 1, name = "NOTIF_SEQ")
    @Column(name = "id", unique = true, nullable = false)
    private Long id;
    @Column(name = "message", nullable = false)
    private String messageToSend;
    @Column(name = "title", nullable = false)
    private String titleFromMessage;
    private String oneSignalId;
    @Column(name = "time_send", nullable = false)
    private LocalDateTime timeToSend;
    private String segmentslist;
```

```
public class NotificationModel {

    private String title;
    private String message;
    private Instant deliveryTime;
    private List<String> segments;
    private String idMessage;
    private Long idInternal;
```

```
@Component
public abstract class UtilNotification {

    public static NotificationEntity notificationModelToNotificationEntity(NotificationModel model){

        NotificationEntity entity = new NotificationEntity();
        entity.setMessageToSend(model.getMessage());
        entity.setOneSignalID(model.getIdMessage());
        entity.setSegmentsList(buildSegmentList(model));
        entity.setTimeToSend(UtilCommons.instantToLocalDateTime(model.getDeliveryTime()));
        entity.setTitleFromMessage(model.getTitle());
        entity.setId(model.getIdInternal());

        return entity;
    }

    public static NotificationModel entityToNotificationModel(NotificationEntity entity){
        NotificationModel model = new NotificationModel();
        model.setIdMessage(entity.getOneSignalID());

        model.setDeliveryTime(entity.getTimeToSend().toInstant(ZoneOffset.UTC));
        model.setMessage(entity.getMessageToSend());
        model.setSegments(Arrays.asList(entity.getSegmentsList().split("regex: ","")));
        model.setTitle(entity.getTitleFromMessage());
        model.setIdInternal(entity.getId());

        return model;
    }
}
```

```
@Component
public abstract class UtilNotification {

    public static NotificationEntity notificationModelToNotificationEntity(NotificationModel model){

        NotificationEntity entity = new NotificationEntity();
        entity.setMessageToSend(model.getMessage());
        entity.setOneSignalID(model.getIdMessage());
        entity.setSegmentsList(buildSegmentList(model));
        entity.setTimeToSend(UtilCommons.instantToLocalDateTime(model.getDeliveryTime()));
        entity.setTitleFromMessage(model.getTitle());
        entity.setId(model.getIdInternal());

        return entity;
    }

    public static NotificationModel entityToNotificationModel(NotificationEntity entity){
        NotificationModel model = new NotificationModel();
        model.setIdMessage(entity.getOneSignalID());

        model.setDeliveryTime(entity.getTimeToSend().toInstant(ZoneOffset.UTC));
        model.setMessage(entity.getMessageToSend());
        model.setSegments(Arrays.asList(entity.getSegmentsList().split("regex: ","")));
        model.setTitle(entity.getTitleFromMessage());
        model.setIdInternal(entity.getId());

        return model;
    }
}
```



```
public NotificationModel insertNotification(NotificationModel model) {  
    NotificationEntity notificationEntity = this.repository  
        .save(UtilNotification  
            .notificationModelToNotificationEntity(model));  
  
    if (notificationEntity.getId() <= 0) {  
        UtilErrors.badRequest("Não foi possível salvar a notificação");  
    }  
    return UtilNotification.entityToNotificationModel(notificationEntity);  
}
```

```
@PostMapping  
public NotificationModel insertMessage(@RequestBody NotificationModel notificationModel) throws Exception {  
  
    NotificationModel model = this.externalService.insertNotification(notificationModel);  
    if(!StringUtils.isEmpty(model.getIdMessage())){  
        return this.internalService.insertNotification(model);  
    }else{  
        UtilErrors.internalServerError("Não foi possível inserir os dados da notificação.");  
    }  
    return null;  
}
```

Procurem criar um pacote com classes do tipo Util! Dentro dela você pode colocar as conversões de objetos, de datas, tratamentos de string, etc. Isso faz com que o código fique mais limpo e organizado.



# Serviços





É a camada da aplicação que contém as regras do negócio. Nela você pode fazer as requisições filtradas para a camada de repositório e devolver para a camada de Controller as respostas que vieram do banco (Corpo, status, etc.)

```
@Service
public class NotificationInternalService {

    @Autowired
    private NotificationRepository repository;

    public NotificationModel insertNotification(NotificationModel model) {
        NotificationEntity notificationEntity = this.repository
            .save(UtilNotification
                .notificationModelToNotificationEntity(model));

        if (notificationEntity.getId() <= 0) {
            UtilErrors.badRequest("Não foi possível salvar a notificação");
        }
        return UtilNotification.entityToNotificationModel(notificationEntity);
    }

    public NotificationModel getNotification(Long id) {
        Optional<NotificationEntity> notificationEntity = this.repository.findById(id);
        if (!notificationEntity.isPresent())
            UtilErrors.notFoundException("Não foi possível encontrar a notificação pesquisada!");

        return UtilNotification.entityToNotificationModel(notificationEntity.get());
    }
}
```

```
@Service
public class NotificationExternalService {

    @Value("${app.id}")
    private String appId;

    @Value("${app.token}")
    private String token;

    @Autowired
    private ConnectionService connectionService;

    public String update(Long id, NotificationModel model) { return null; }

    public String viewNotification(String notificationId) {...}

    public String deleteNotification(String id) {...}

    public String viewNotifications() throws IOException {...}

    public NotificationModel insertNotification(NotificationModel model) throws Exception {...}
}
```





# Controller



```

@RestController
@Api("Camiseta")
@RequestMapping("/eventos/api/camisetas")
public class CamisetaController {

    @Autowired
    private CamisetaService camisetaService;

    private static Logger LOG = LoggerFactory.getLogger(CamisetaController.class);

    @ApiOperation(httpMethod = "GET", value = "Listagem de Camisetas", nickname = "Listagem de Camisetas")
    @GetMapping
    public ResponseEntity<?> listarCamisetas(){
        LOG.debug("Entrou na chamada de listar camisetas. ");
        List<CamisetaDTO> camisetas = this.camisetaService.camisetas();
        LOG.warn("Retornaram as informações para serem verificadas");
        if(camisetas.isEmpty()) {
            LOG.error("Não foi possível listar as camisetas. ");
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        LOG.debug("Retornou a lista de camisetas", camisetas);
        return ResponseEntity.ok(camisetas);
    }

    @ApiOperation(httpMethod = "GET", value = "Recupera Camisa", nickname = "Recupera Camisa")
    @GetMapping("/{id}")
    public ResponseEntity<?> camiseta (@PathVariable int id) {
        LOG.debug("Entrou na chamada de selecionar camiseta. ");
        CamisetaDTO camisetaVar = this.camisetaService.buscarCamiseta(id);
        LOG.warn("Retornara a informação para ser verificada");
        if(camisetaVar == null) {
            LOG.error("Não foi possível retornar a informação. ");
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        LOG.debug("Retornou a informação ", camisetaVar);
        return ResponseEntity.ok(camisetaVar);
    }
}

```

Qual a diferença entre @Controller e  
@RestController?



```

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {

    /**
     * The value may indicate a suggestion for a logical component name,
     * to be turned into a Spring bean in case of an autodetected component.
     * @return the suggested component name, if any (or empty String otherwise)
     * @since 4.0.1
     */
    @AliasFor(annotation = Controller.class)
    String value() default "";
}

```

Por esse motivo, opta-se por usar a `@RestController` já que estende a anotação `@ResponseBody`.

A Anotação `@Controller` tem como extensão outras anotações, mas ela não tem a anotação que serializa os dados para que depois eles possam ser enviados em formato XML ou JSON.

```

@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Controller {

    /**
     * The value may indicate a suggestion for a logical component name,
     * to be turned into a Spring bean in case of an autodetected component.
     * @return the suggested component name, if any (or empty String otherwise)
     */
    @AliasFor(annotation = Component.class)
    String value() default "";
}

```

Dá pra utilizar a anotação `@Controller` e serializar esses objetos, utilizando do seguinte jeito:

```
@Controller
@Api("Aeropostos")
@RequestMapping("/eventos/api/aeroportos")
public class AeroportoController {

    @Autowired
    private AeroportoService aeroportoService;

    private static Logger LOG = LoggerFactory.getLogger(AeroportoService.class);

    @ResponseBody
    @ApiOperation(httpMethod = "GET", value = "Lista de Aeroportos.", nickname = "Aeroportos")
    @GetMapping
    public ResponseEntity<?> listarAeroportos() {
        LOG.debug("Entrou na chamada de api que lista os aeroportos. ");
        List<AeroportoDTO> aeroportos = this.aeroportoService.aeroportosList();
        LOG.debug("Foram encontrados registros.");
        return ResponseEntity.ok(aeroportos);
    }

    @ResponseBody
    @ApiOperation(httpMethod = "GET", value = "Recupera Aeroportos.", nickname = "Recupera")
    @GetMapping("/{id}")
    public ResponseEntity<?> aeroporto(@PathVariable int id) {
        LOG.error("Entrou na chamada que seleciona apenas um aeroporto. ");
        AeroportoDTO aeroportoVar = this.aeroportoService.aeroporto(id).get();
        LOG.debug("Vai retornar o dado selecionado. ");
        return ResponseEntity.ok(aeroportoVar);
    }
}
```

Lembrem-se de que a anotação `@Controller` é indicada quando você vai usar enviar os dados pra uma view. Evitem usá-la caso vocês forem trabalhar com API REST.

# Videografia

## 1. API

[https://www.youtube.com/watch?v=ghTrp1x\\_1As](https://www.youtube.com/watch?v=ghTrp1x_1As)

## 2. Design Patterns

<https://www.youtube.com/watch?v=J-lHpiu-Twk>

<https://www.youtube.com/watch?v=8vq2QB4ogKM>

## 3. Controllers

<https://www.youtube.com/watch?v=qsIkWczoan8>



# Sites Visitados

## 1. API

<https://developer.mozilla.org/pt-BR/docs/Web/API/Cache>

## 2. Design Patterns

<https://brizeno.wordpress.com/category/padroes-de-projeto/interpreter/>

<https://blog.schoolofnet.com/o-que-sao-design-patterns/>

<https://pt.wikipedia.org/wiki/Builder>

<https://refactoring.guru/design-patterns/builder>

<https://blog.matheuscastiglioni.com.br/interpreter-padroes-de-projeto-em-java/>

## 3. HTTP Status

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>

<https://http.cat/>

## 4. Hibernate

<https://howtodoinjava.com/hibernate/hibernate-jpa-cascade-types/>

## 5. Repositórios

<https://www.devmedia.com.br/forum/crudrepository-vs-jparepository/597127>

## 6. Serviços

<https://www.baeldung.com/spring-component-repository-service>

**τέλος**



