

# Curso: Programação Orientada a Objetos com Java

## Projeto MongoDB com Spring Boot

### Objetivo geral:

- Compreender as principais diferenças entre paradigma orientado a documentos e relacional
- Implementar operações de CRUD
- Refletir sobre decisões de design para um banco de dados orientado a documentos
- Implementar associações entre objetos
  - Objetos aninhados
  - Referências
- Realizar consultas com Spring Data e MongoRepository

### Instalação do MongoDB

#### Checklist Windows:

- <https://www.mongodb.com> -> Download -> Community Server
- Baixar e realizar a instalação com opção "Complete"
  - **ATENÇÃO:** optaremos no curso por **NÃO** instalar o Compass por enquanto
- <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/> -> Set up the MongoDB environment
  - Criar pasta \data\db
  - Acrescentar em PATH: C:\Program Files\MongoDB\Server\3.6\bin (adapte para sua versão)
- Testar no terminal: mongod

#### Checklist Mac:

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/>

- Instalar brew:
  - <https://brew.sh> -> executar o comando apresentado na primeira página
- Instalar o MongoDB:
  - **brew install mongodb**
- Criar pasta /data/db:
  - **sudo mkdir -p /data/db**
- Liberar acesso na pasta criada
  - **whoami** (para ver seu nome de usuário, exemplo: nelio)
  - **sudo chown -Rv nelio /data/db**
- Testar no terminal:
  - **mongod**

### Instalação do Mongo Compass

#### Referências:

<https://www.mongodb.com/products/compass>

## Primeiro commit - projeto criado

### ATUALIZAÇÃO

#### ATENÇÃO: VERSÃO DO SPRING BOOT:

1.5.x: <https://github.com/acenelio/workshop-spring-boot-mongodb>

2.x.x: <https://github.com/acenelio/workshop-springboot2-mongodb>

Quando houver alguma atualização para compatibilidade com Spring Boot 2, será mostrado no início do vídeo e também aqui no material de apoio.

#### Erro comum: arquivo corrompido do Maven (invalid LOC header)

Recomendação: apague os arquivos e voltar ao STS e deixar o Maven refazer o download

- Vídeo: <https://www.youtube.com/watch?v=Fn1oXbDtOg>

#### Checklist:

- File -> New -> Spring Starter Project
  - Escolher somente o pacote Web por enquanto
- Rodar o projeto e testar: <http://localhost:8080>
- Se quiser mudar a porta padrão do projeto, incluir em application.properties: `server.port=${port:8081}`

## Entity User e REST funcionando

#### Checklist para criar entidades:

- Atributos básicos
- Associações (inicie as coleções)
- Construtores (não inclua coleções no construtor com parâmetros)
- Getters e setters
- hashCode e equals (implementação padrão: somente id)
- Serializable (padrão: 1L)

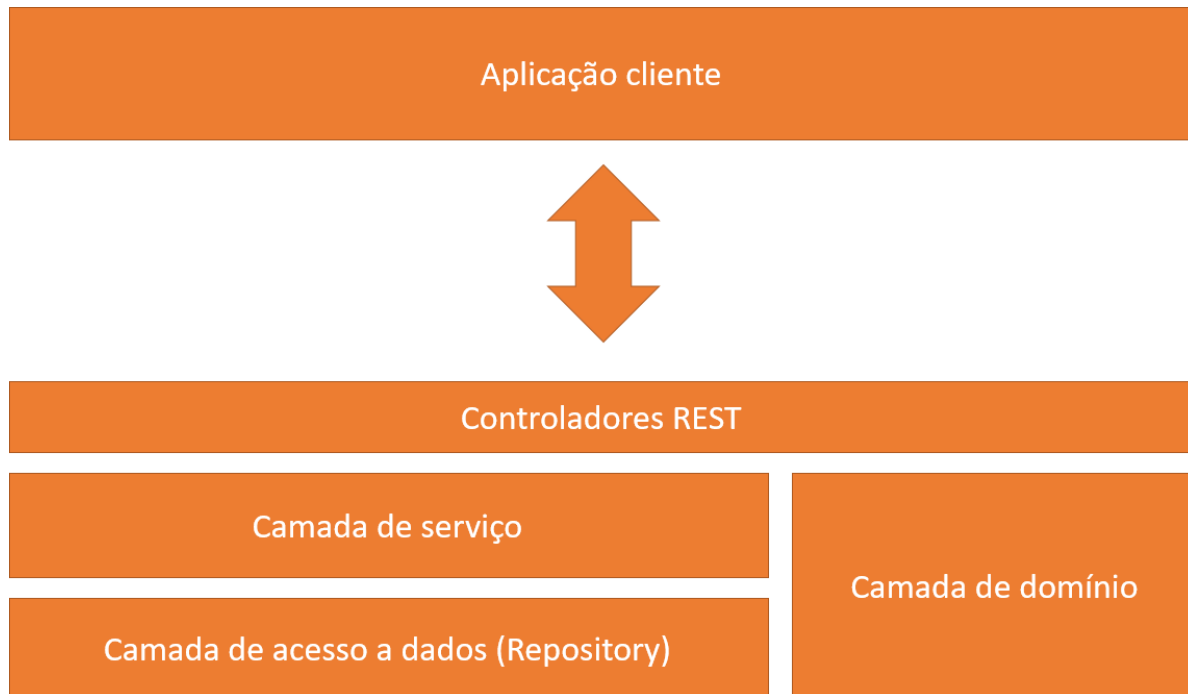
#### Checklist:

- No subpacote domain, criar a classe User
- No subpacote resources, criar uma classe UserResource e implementar nela o endpoint GET padrão:

```
@RestController
@RequestMapping(value="/users")
public class UserResource {

    @RequestMapping(method=RequestMethod.GET)
    public ResponseEntity<List<User>> findAll() {
        List<User> list = new ArrayList<>();
        User maria = new User("1001", "Maria Brown", "maria@gmail.com");
        User alex = new User("1002", "Alex Green", "alex@gmail.com");
        list.addAll(Arrays.asList(maria, alex));
        return ResponseEntity.ok().body(list);
    }
}
```

## Conectando ao MongoDB com repository e service



### Referências:

<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>  
<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-nosql.html>  
<https://stackoverflow.com/questions/38921414/mongodb-what-are-the-default-user-and-password>

### Checklist:

- Em pom.xml, incluir a dependência do MongoDB:

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

- No pacote repository, criar a interface UserRepository
- No pacote services, criar a classe UserService com um método findAll
- Em User, incluir a anotação @Document e @Id para indicar que se trata de uma coleção do MongoDB
- Em UserResource, refatorar o código, usando o UserService para buscar os usuários
- Em application.properties, incluir os dados de conexão com a base de dados:  
spring.data.mongodb.uri=mongodb://localhost:27017/workshop\_mongo
- Subir o MongoDB (comando mongod)
- Usando o MongoDB Compass:
  - Criar base de dados: workshop\_mongo
  - Criar coleção: user
  - Criar alguns documentos user manualmente
- Testar o endpoint /users

## Operação de instanciação da base de dados

### ATUALIZAÇÃO

Para o projeto ficar compatível com Spring Boot 2.x.x:

```
userRepository.saveAll(Arrays.asList(maria, alex, bob));
```

#### Checklist:

- No subpacote config, criar uma classe de configuração Instantiation que implemente CommandLineRunner
- Dados para copiar:

```
User maria = new User(null, "Maria Brown", "maria@gmail.com");
```

```
User alex = new User(null, "Alex Green", "alex@gmail.com");
```

```
User bob = new User(null, "Bob Grey", "bob@gmail.com");
```

## Usando padrão DTO para retornar usuários

#### Referências:

<https://pt.stackoverflow.com/questions/31362/o-que-é-um-dto>

**DTO (Data Transfer Object):** é um objeto que tem o papel de carregar dados das entidades de forma simples, podendo inclusive "projetar" apenas alguns dados da entidade original. Vantagens:

- Otimizar o tráfego (trafegando menos dados)
- Evitar que dados de interesse exclusivo do sistema fiquem sendo expostos (por exemplo: senhas, dados de auditoria como data de criação e data de atualização do objeto, etc.)
- Customizar os objetos trafegados conforme a necessidade de cada requisição (por exemplo: para alterar um produto, você precisa dos dados A, B e C; já para listar os produtos, eu preciso dos dados A, B e a categoria de cada produto, etc.).

#### Checklist:

- No subpacote dto, criar UserDTO
- Em UserResource, refatorar o método findAll

## Obtendo um usuário por id

### ATUALIZAÇÃO

Para o projeto ficar compatível com Spring Boot 2.x.x:

Classe **UserService**, troque o código antigo:

```
public User findById(String id) {  
    User user = repo.findOne(id);  
    if (user == null) {  
        throw new ObjectNotFoundException("Objeto não encontrado");  
    }  
    return user;  
}
```

Por este:

```
import java.util.Optional;  
  
(...)  
  
public User findById(String id) {  
    Optional<User> obj = repo.findById(id);  
    return obj.orElseThrow(() -> new ObjectNotFoundException("Objeto não encontrado"));  
}
```

#### Checklist:

- No subpacote service.exception, criar ObjectNotFoundException
- Em UserService, implementar o método findById
- Em UserResource, implementar o método findById (retornar DTO)
- No subpacote resources.exception, criar as classes:
  - StandardError
  - ResourceExceptionHandler

## Inserção de usuário com POST

#### Checklist:

- Em UserService, implementar os métodos insert e fromDTO
- Em UserResource, implementar o método insert

## Deleção de usuário com DELETE

### ATUALIZAÇÃO

Para o projeto ficar compatível com Spring Boot 2.x.x:

```
repo.deleteById(id);
```

**Checklist:**

- Em UserService, implementar o método delete
- Em UserResource, implementar o método delete

## Atualização de usuário com PUT

### ATUALIZAÇÃO

Para o projeto ficar compatível com Spring Boot 2.x.x:

Classe **UserService**, troque o código antigo:

```
public User update(User obj) {  
    User newObj = repo.findOne(obj.getId());  
    updateData(newObj, obj);  
    return repo.save(newObj);  
}
```

Por este:

```
public User update(User obj) {  
    User newObj = findById(obj.getId());  
    updateData(newObj, obj);  
    return repo.save(newObj);  
}
```

**Checklist:**

- Em UserService, implementar os métodos update e updateData
- Em UserResource, implementar o método update

## Criando entity Post com User aninhado

**Nota:** objetos aninhados vs. referências

**Checklist:**

- Criar classe Post
- Criar PostRepository
- Inserir alguns posts na carga inicial da base de dados

## Projeção dos dados do autor com DTO

**Checklist:**

- Criar AuthorDTO
- Refatorar Post
- Refatorar a carga inicial do banco de dados
  - **IMPORTANTE:** agora é preciso persistir os objetos User antes de relacionar

## Referenciando os posts do usuário

### Checklist:

- Em User, criar o atributo "posts", usando @DBRef
  - Sugestão: incluir o parâmetro (lazy = true)
- Refatorar a carga inicial do banco, incluindo as associações dos posts

## Endpoint para retornar os posts de um usuário

### Checklist:

- Em UserResource, criar o método para retornar os posts de um dado usuário

## Obtendo um post por id

### Checklist:

- Criar PostService com o método findById
- Criar PostResource com método findById

## Acrescentando comentários aos posts

### Checklist:

- Criar CommentDTO
- Em Post, incluir uma lista de CommentDTO
- Refatorar a carga inicial do banco de dados, incluindo alguns comentários nos posts

## Consulta simples com query methods

### Referências:

<https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/>  
<https://docs.spring.io/spring-data/data-document/docs/current/reference/html/>

### Consulta:

*"Buscar posts contendo um dado string no título"*

### Checklist:

- Em PostRepository, criar o método de busca
- Em PostService, criar o método de busca
- No subpacote resources.util, criar classe utilitária URL com um método para decodificar parâmetro de URL
- Em PostResource, implementar o endpoint

## Consulta simples com @Query

### Referências:

<https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/>  
<https://docs.spring.io/spring-data/data-document/docs/current/reference/html/>  
<https://docs.mongodb.com/manual/reference/operator/query/regex/>

**Consulta:**

*"Buscar posts contendo um dado string no título"*

**Checklist:**

- Em PostRepository, fazer a implementação alternativa da consulta
- Em PostService, atualizar a chamada da consulta

## Consulta com vários critérios

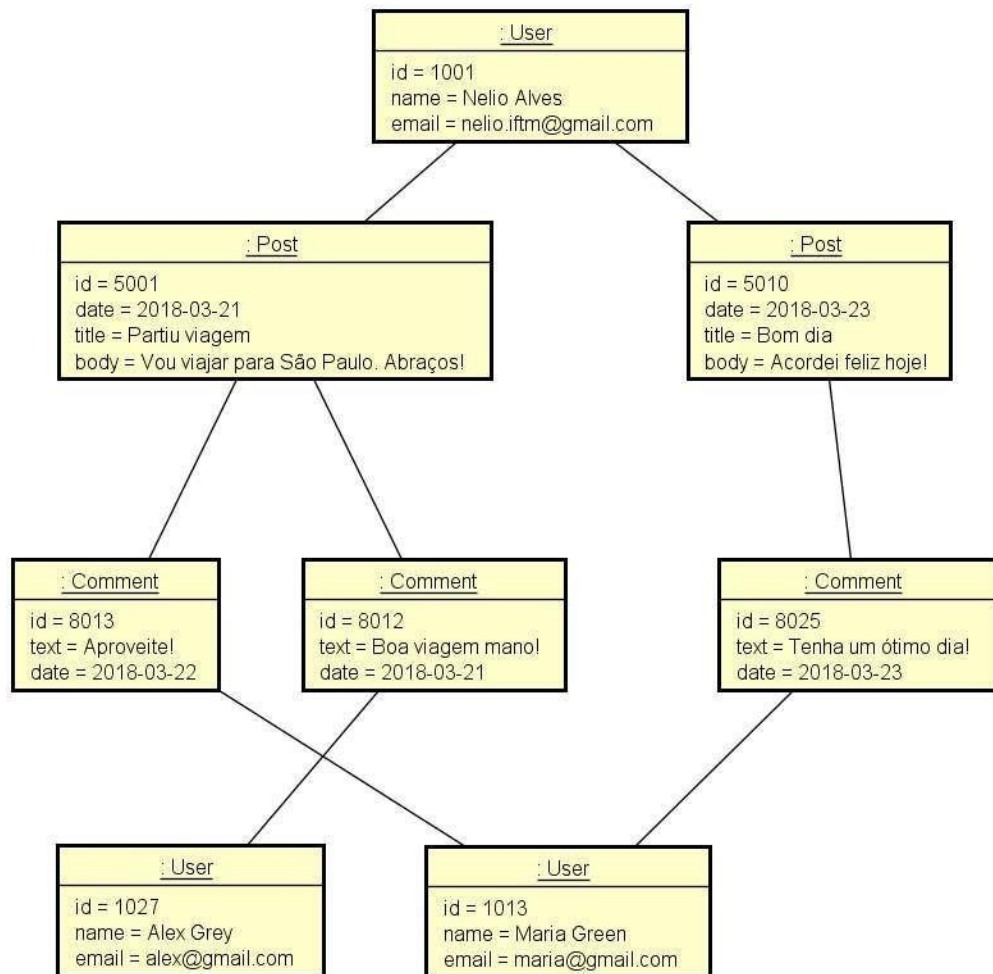
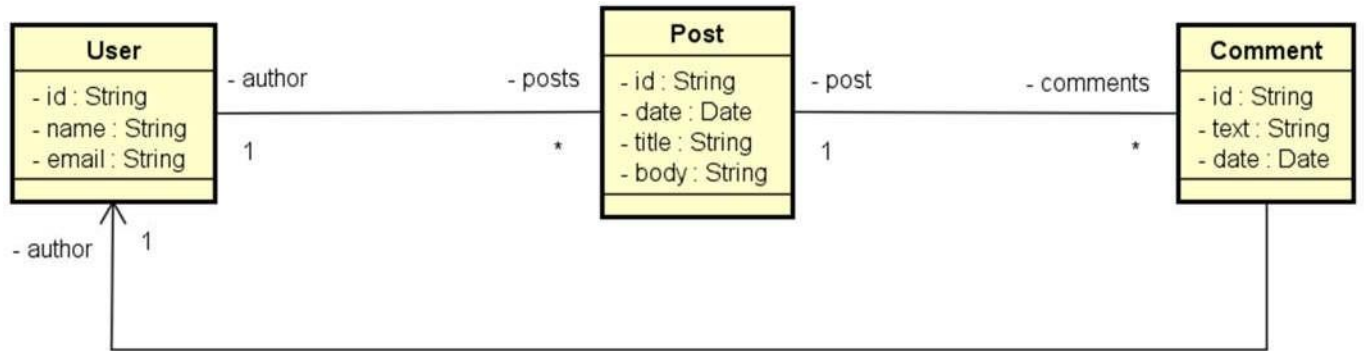
**Consulta:**

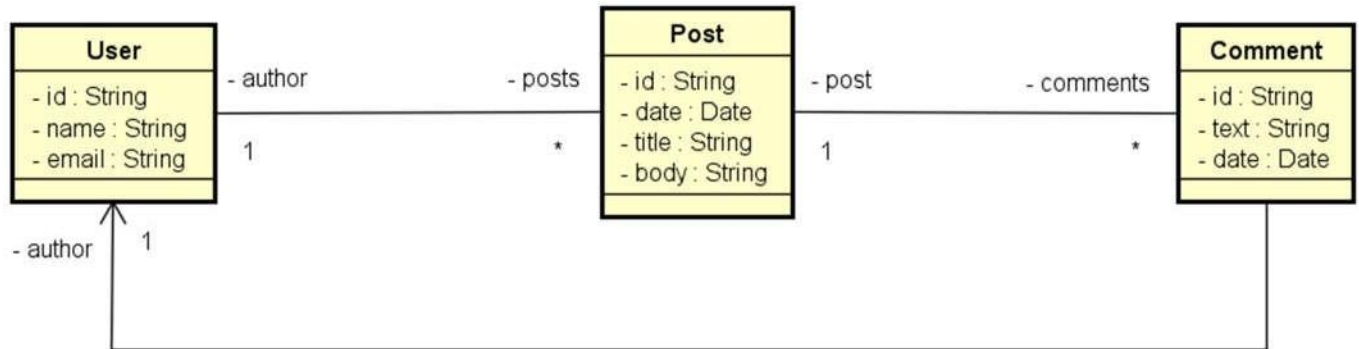
*"Buscar posts contendo um dado string em qualquer lugar (no título, corpo ou comentários) e em um dado intervalo de datas"*

**Checklist:**

- Em PostRepository, criar o método de consulta
- Em PostService, criar o método de consulta
- Na classe utilitária URL, criar métodos para tratar datas recebidas
- Em PostResource, implementar o endpoint







**user**

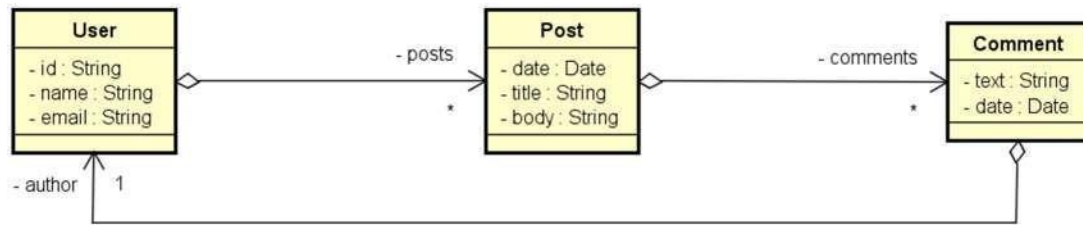
id	name	email
x-o-x	x-o-x	x-o-x
1001	Maria Brown	maria@gmail.com
x-o-x	x-o-x	x-o-x
x-o-x	x-o-x	x-o-x
1013	Alex Green	alex@gmail.com
x-o-x	x-o-x	x-o-x
1027	Bob Grey	bob@gmail.com

**post**

id	date	title	body	author_id
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
5001	2018-03-21	Partiu viagem	Vou viajar para São Paulo. Abraços!	1001
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
5010	2018-03-23	Bom dia	Acordei feliz hoje!	1001
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x

**comment**

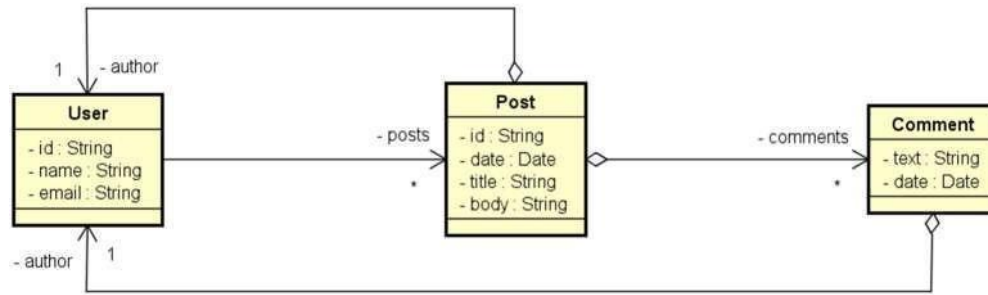
id	text	date	post_id	author_id
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
8012	Boa viagem mano!	2018-03-21	5001	1013
8013	Aproveite!	2018-03-22	5001	1027
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x
8025	Tenha um ótimo dia!	2018-03-23	5010	1013
x-o-x	x-o-x	x-o-x	x-o-x	x-o-x



```

{
  "id": "1001",
  "name": "Maria Brown",
  "email": "maria@gmail.com",
  "posts": [
    {
      "date": "2018-03-21",
      "title": "Partiu viagem",
      "body": "Vou viajar para São Paulo. Abraços!",
      "comments": [
        {
          "text": "Boa viagem mano!",
          "date": "2018-03-21",
          "author": {
            "id": "1013",
            "name": "Alex Green"
          }
        },
        {
          "text": "Aproveite!",
          "date": "2018-03-22",
          "author": {
            "id": "1027",
            "name": "Bob Grey"
          }
        }
      ]
    }
  ],
  {
    "date": "2018-03-23",
    "title": "Bom dia",
    "body": "Acordei feliz hoje!",
    "comments": [
      {
        "text": "Tenha um ótimo dia!",
        "date": "2018-03-23",
        "author": {
          "id": "1013",
          "name": "Alex Green"
        }
      }
    ]
  }
]
}

```



```

{
  "id": "1001",
  "name": "Maria Brown",
  "email": "maria@gmail.com",
  "posts": ["5001", "5010"]
}
{
  "id": "5001",
  "date": "2018-03-21",
  "title": "Partiu viagem",
  "body": "Vou viajar para São Paulo. Abraços!",
  "author": {
    "id": "1001",
    "name": "Maria Brown"
  },
  "comments": [
    {
      "text": "Boa viagem mano!",
      "date": "2018-03-21",
      "author": {
        "id": "1013",
        "name": "Alex Green"
      }
    },
    {
      "text": "Aproveite!",
      "date": "2018-03-22",
      "author": {
        "id": "1027",
        "name": "Bob Grey"
      }
    }
  ]
}
{
  "id": "5010",
  "date": "2018-03-23",
  "title": "Bom dia",
  "body": "Acordei feliz hoje!",
  "author": {
    "id": "1001",
    "name": "Maria Brown"
  },
  "comments": [
    {
      "text": "Tenha um ótimo dia!",
      "date": "2018-03-23",
      "author": {
        "id": "1013",
        "name": "Alex Green"
      }
    }
  ]
}
}

```