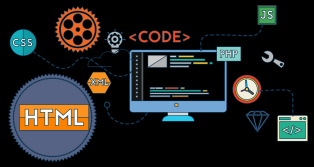
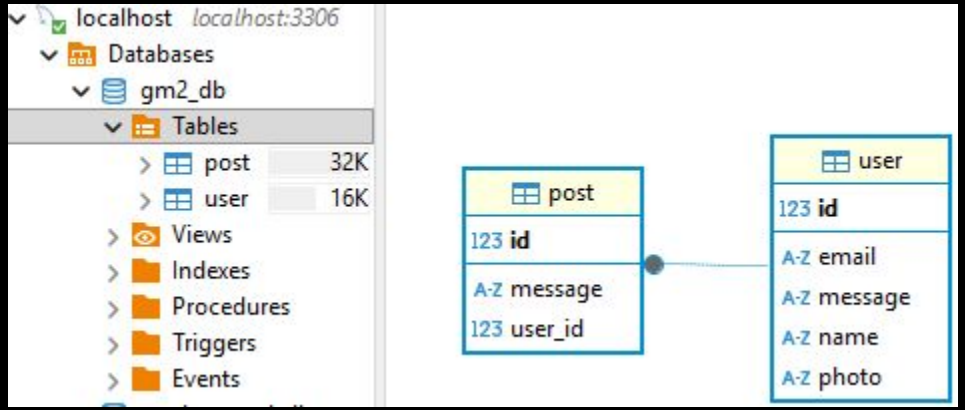
Tópicos Avançados de Programação

Genilson Medeiros

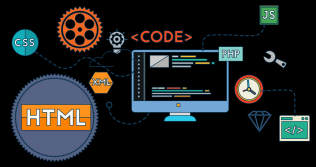
Professor do curso de Sistemas de Informação
Mestre em Ciência e Tecnologia em Saúde
Engenheiro de Software

Relacionamentos



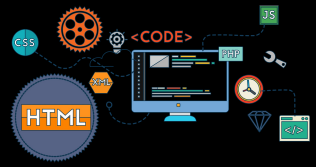


Relacionamentos




```java
@Builder  8 usages  new *
@Data
@Entity
@Table(name = "user")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = Constants.GENERAL_MAX_LENGTH, nullable = false)
    private String name;

    @Column(length = Constants.GENERAL_MAX_LENGTH, nullable = false, unique = true)
    private String email;

    @Column(length = Constants.PHOTO_PATH, nullable = true)
    private String photo;

    @Column(length = Constants.MESSAGE_LENGTH, nullable = true)
    private String message;

    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    private Set<Post> posts = new HashSet<>();
}
```

Relacionamentos



1. `@OneToMany(mappedBy = "user", ...)`

- Indica um **relacionamento um-para-muitos**:
  Um **User** tem **muitos Posts**.

- `mappedBy = "user"`:
  Diz que **essa relação é mapeada pelo atributo `user` dentro da classe `Post`**.
  Isso significa que a **tabela `Post` tem a FK (chave estrangeira)** para `User`.
  Ou seja, essa entidade (User) **não tem a coluna de relacionamento**, ela só referência.

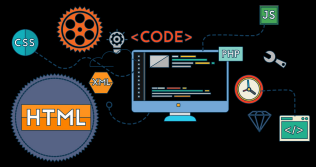
2. `fetch = FetchType.LAZY`

- Define o **carregamento dos dados**.

- LAZY (preguiçoso):
  Os `posts` só são carregados do banco **quando você acessar `getPosts()`**.

- Isso ajuda a performance, evitando carregar todos os `posts` junto com o `user`.

Relacionamentos



3. `cascade = CascadeType.ALL`

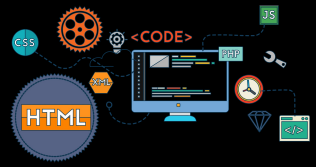
- Define o comportamento em **cascata** para operações do JPA:
    - ALL: Todas as operações (PERSIST, MERGE, REMOVE, REFRESH, DETACH) vão se propagar.
- Exemplo: Se você salvar ou deletar um `User`, os `posts` deste usuário também serão salvos ou deletados automaticamente.

4. `@ToString.Exclude`

- Vem do **Lombok**.
- Exclui `posts` do método `toString()`.

  Isso evita **loop infinito** quando `Post` também tem referência para `User`.

Relacionamentos



5. `@EqualsAndHashCode.Exclude`

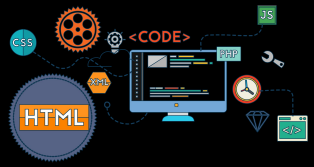
- Também do Lombok.
- Exclui `posts` de `equals()` e `hashCode()`.

Assim, a comparação de dois `User` não vai levar em conta a lista de `posts` (evita loops e melhora performance).

6. `private Set<Post> posts = new HashSet<>();`

- Inicializa `posts` como um `HashSet` vazio.
- `Set` impede duplicação de elementos.
- Já inicializar evita `NullPointerException` quando acessar `getPosts()`.

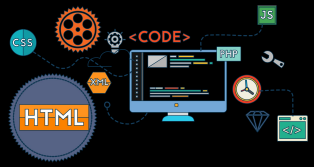
Relacionamentos




```java
@Builder  3 usages  new *
@Data
@Entity
@Table(name = "post")
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = Constants.MESSAGE_LENGTH, nullable = false)
    private String message;

    @ManyToOne(fetch = FetchType.LAZY)
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
}
```

```java
@Repository  no usages  new *
public interface PostRepository extends JpaRepository<Post, Long> {
}
```

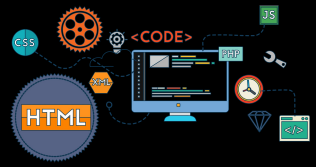
Post DTO




```java
@Builder  2 usages  new *
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PostRequestDto {
    private Long userId;
    private String message;
}
```

```java
@Builder  6 usages  new *
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PostResponseDto {
    private Long id;
    private String message;
}
```

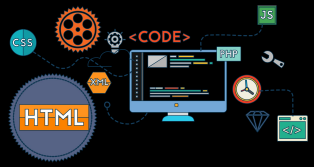
Post Service




```java
@Service  no usages  new *
public class PostService {
    private final PostRepository postRepository;  2 usages
    private final UserRepository userRepository;  2 usages

    public PostService(PostRepository postRepository, UserRepository userRepository) {
        this.postRepository = postRepository;
        this.userRepository = userRepository;
    }

    public PostResponseDto register(PostRequestDto post) {  no usages  new *
        var user = userRepository.getReferenceById(post.getUserId());

        var newPost = postRepository.save(toPost(post, user));
        return fromPost(newPost);
    }
```

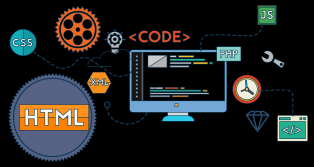
```java
@Builder  11 usages  new *
@Data
@Entity
@Table(name = "user")
@AllArgsConstructor
@NoArgsConstructor
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

Post Service



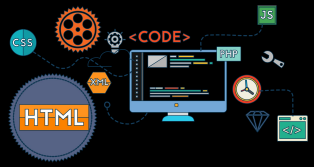

```
private Post toPost(PostRequestDto post, User user) {  1 usa
    return Post.builder()
            .user(user)
            .message(post.getMessage())
            .build();
}

private PostResponseDto fromPost(Post post) {  1 usage  new *
    return PostResponseDto.builder()
                .id(post.getId())
                .message(post.getMessage())
            .build();
}
```

PostController




```java
@RestController  no usages  new *
@RequestMapping("/post")
public class PostController {

    private final PostService postService;  2 usages

    public PostController(PostService postService) {  no usages  new *
        this.postService = postService;
    }

    @PostMapping()  no usages  new *
    public ResponseEntity<PostResponseDto> register(@RequestBody PostRequestDto post) {
        return new ResponseEntity<>(postService.register(post), HttpStatus.CREATED);
    }
}
```

Postman



POST http://localhost:8080/post

Params Authorization Headers (8) Body Pre-request Script

none form-data x-www-form-urlencoded raw binar

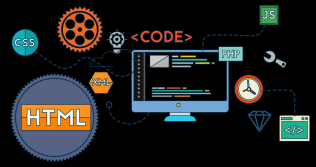
```
1  {
2      "userId": 1,
3      "message": "Em breve teremos novidades"
4  }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2      "id": 2,
3      "message": "Em breve teremos novidades"
4  }
```

Obter todas as postagens



Repository

```
@Repository  3 usages  new *
public interface PostRepository extends JpaRepository<Post, Long> {
    List<Post> findByUserId(Long userId);  1 usage  new *
}
```

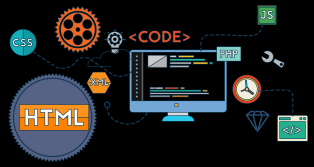
Service

```
public List<PostResponseDto> getAll(Long userId) {  1 usage  new *
    var posts = postRepository.findByUserId(userId);
    return posts.stream().map( Post p-> fromPost(p)).collect(Collectors.toList());
}
```

Controller

```
@GetMapping("{userId}")  no usages  new *
public ResponseEntity<List<PostResponseDto>> getAll(@PathVariable Long userId) {
    return ResponseEntity.ok(postService.getAll(userId));
}
```

Postman



GET http://localhost:8080/post/1

Params Authorization Headers (6) Body Pre-request Script

none form-data x-www-form-urlencoded raw bina

```
1
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1  [
2      {
3          "id": 1,
4          "message": "Bem vindos a minha rede social"
5      },
6      {
7          "id": 2,
8          "message": "Em breve teremos novidades"
9      }
10 ]
```