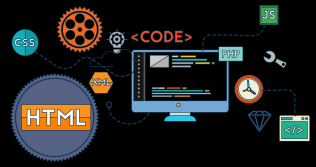
Tópicos Avançados de Programação

Genilson Medeiros

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

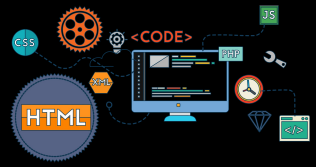
@Service



O @Service é uma anotação do Spring que marca uma classe como um Service dentro da arquitetura da aplicação.

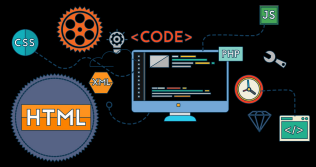
Ela indica que a classe contém lógica de negócio (regras, cálculos, validações) e deve ser gerenciada pelo Spring como um bean.

Service




```java
@Service  3 usages  new *
public class UserService {

    private final UserRepository userRepository;  3 usages

    public UserService(UserRepository userRepository) {  no usages  new *
        this.userRepository = userRepository;
    }

    public User register(User user) {  1 usage  new *
        if (userRepository.existsByEmail(user.getEmail())) {
            throw new DuplicateEmailException(
                    String.format(Strings.DUPLICATE_EMAIL, user.getEmail()));
        }

        return userRepository.save(user);
    }
}
```

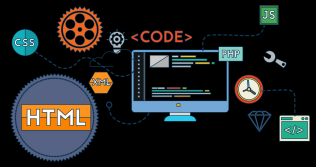
Strings



gm2-social-network C:\exemplos\uni
- .idea
- .mvn
- src
  - main
    - java
      - gm2.br.gm2_social_netwo
        - controller
        - dto
        - entity
        - exception
        - repository
        - service
        - utils
          - Constants
          - Strings

```
1  package gm2.br.gm2_social_network.utils;
2
3  public final class Strings {  no usages  new *
4      public static final String DUPLICATE_EMAIL = "Já existe um usuário com o e-mail %s";
5  }
6
```

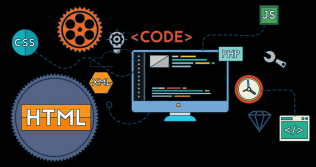
Strings



gm2-social-network C:\exemplos\uni
- .idea
- .mvn
- src
  - main
    - java
      - gm2.br.gm2_social_netwo
        - controller
        - dto
        - entity
        - exception
        - repository
        - service
        - utils
          - Constants
          - Strings

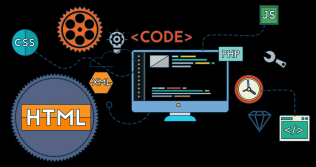
```
1  package gm2.br.gm2_social_network.utils;
2
3  public final class Strings {  no usages  new *
4      public static final String DUPLICATE_EMAIL = "Já existe um usuário com o e-mail %s";
5  }
6
```

Controller



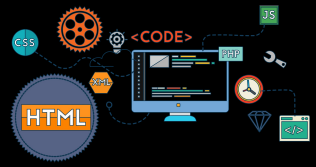

```java
@RestController  no usages  new *
@RequestMapping("/user")
public class UserController {

    private final UserService userService;  2 usages

    private UserController(UserService userService) {  no usages  new *

        this.userService = userService;
    }

    @PostMapping()  no usages  new *
    public ResponseEntity<User> register(@RequestBody User user) {

        return new ResponseEntity<>(userService.register(user), HttpStatus.CREATED);
    }
}
```

DTO - Data Transfer Object



É uma **classe simples** (normalmente só com atributos, getters e setters) usada para **transportar dados** entre diferentes camadas da aplicação (por exemplo: Controller → Service → Client) ou entre sistemas (API → Frontend).

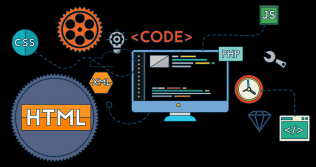
DTO - Data Transfer Object



Para que serve

- **Evitar expor diretamente entidades do banco** (como as classes JPA).
- **Transportar só os dados necessários** (reduzindo payload em APIs, por exemplo).
- **Facilitar validações** específicas da entrada/saída.
- **Adaptar dados** para a necessidade de cada camada (ex.: renomear campos, juntar informações de várias entidades).
- **Desacoplar** a lógica de negócio da forma como os dados são enviados/recebidos.

DTO - Data Transfer Object




```java
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data  2 usages  new *
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class UserRequestDto {
    private String name;
    private String email;
    private String photo;
    private String message;
}
```

```java
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data  5 usages  new *
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class UserResponseDto {
    private Long id;
    private String name;
    private String email;
    private String photo;
    private String message;
}
```

@Builder


```
public UserResponseDto register(UserRequestDto user) {  1 usage  new *
    if (userRepository.existsByEmail(user.getEmail())) {
        throw new DuplicateEmailException(
                String.format(Strings.DUPLICATE_EMAIL, user.getEmail()));
    }

    User newUSer = userRepository.save(toUser(user));

    return fromUser(newUSer);
}
```

```
private User toUser(UserRequestDto user) {  1 u
    return User.builder()
            .name(user.getName())
            .email(user.getEmail())
            .message(user.getMessage())
            .photo(user.getPhoto())
            .build();
}

private UserResponseDto fromUser(User user) {
    return UserResponseDto.builder()
                .id(user.getId())
                .name(user.getName())
                .email(user.getEmail())
                .message(user.getMessage())
                .photo(user.getPhoto())
            .build();
}
```