Revisão Spring boot

```java
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    @NotBlank
    private String nome;

    @NotNull
    @NotBlank
    private String senha;

    private String permissao;

    public Usuario(String nome, String senha) {
        super();
        this.nome = nome;
        this.senha = senha;
    }



}
```

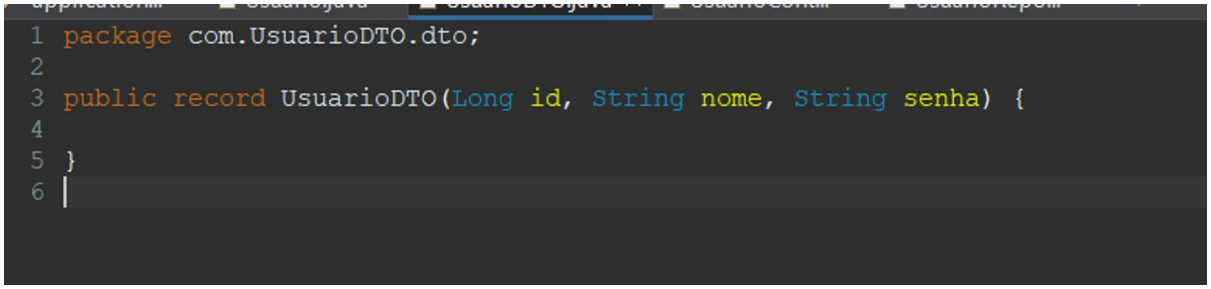
Camada Entidade: Esta é a primeira camada onde ela tem a principal função de definição de atributos e persistência de dados. Ela é anotada por @Entity para indicar que é mapeada para tabelas, também podem ser indicadas por @Id para indicar chave primária.

`@NoArgsConstructor`: Essa anotação gera automaticamente um construtor sem argumentos para a classe.

`@AllArgsConstructor`: Essa anotação gera um construtor com todos os argumentos.

```java
package com.UsuarioDTO.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.UsuarioDTO.entities.Usuario;

public interface UsuarioRepository extends JpaRepository<Usuario, Long>{

}
```

Camada Repositório: Repository é uma camada implementada por interface onde define o JPA, no caso uma API que geralmente serve como uma ponte para as camadas entidade e service.



```java
package com.UsuarioDTO.dto;

public record UsuarioDTO(Long id, String nome, String senha) {

}
```

Camada DTO: A principal função do DTO é simplificar os dados de uma ou mais camadas e garantir a segurança entre a transferência de dados.

Camada Service: Nesta camada temos o @Service que serve para definirmos que esta camada é a service. Nela injetamos diversos métodos (GET, POST, PUT, DELETE)

```java
    @Autowired
    public UsuarioController(UsuarioService usuarioService) {
        this.usuarioService = usuarioService;
    }
    @PostMapping
    public ResponseEntity<UsuarioDTO> criar(@RequestBody @Valid UsuarioDTO usuarioDTO){
        UsuarioDTO salvarUsuario = usuarioService.salvar(usuarioDTO);
        return ResponseEntity.status(HttpStatus.CREATED).body(salvarUsuario);
    }
    @PutMapping("/{id}")
    public ResponseEntity<UsuarioDTO> alterar(@PathVariable Long id, @RequestBody @Valid UsuarioDTO usuarioDTO){
        UsuarioDTO alteraUsarioDTO = usuarioService.atualizar(id, usuarioDTO);
        if(alteraUsarioDTO != null) {
            return ResponseEntity.ok(alteraUsarioDTO);
        }
        else {
            return ResponseEntity.notFound().build();
        }
    }
    @DeleteMapping("/{id}")
    public ResponseEntity<Usuario> delete(@PathVariable Long id){
        boolean delete = usuarioService.delete(id);
        if(delete) {
            return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
        }
        else {
            return ResponseEntity.notFound().build();
        }

    }
    @GetMapping("/{id}")
    public ResponseEntity<Usuario> buscarPorId(@PathVariable Long id){
        Usuario usuario = usuarioService.buscarPorId(id);
        if(usuario != null) {
            return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
        }
        else {
            return ResponseEntity.notFound().build();
        }
    }
    @GetMapping
    public ResponseEntity<List<Usuario>> buscaTodos(){
        List<Usuario> usuario = usuarioService.buscarTodos();
        return ResponseEntity.ok(usuario);
    }
}
```

Camada Controller: Temos o @RestController que serve para marcar a classe como um controlador REST do Spring. Também temos o @RequestMapping, esta anotação define um

mapeamento de URL base para todas as rotas definidas nesta classe. Na camada controller é usada para aplicar os métodos definidos no service.

Application: Serve para configurar o H2 ou qualquer outro banco de dados que você inserir.