Antes de criar as camadas é necessário que se coloque as dependências:

Spring Boot DevTools

Lombok

Spring Web

Validation

MySql Driver

H2 Database

Spring Data

CAMADA ENTIDADES -

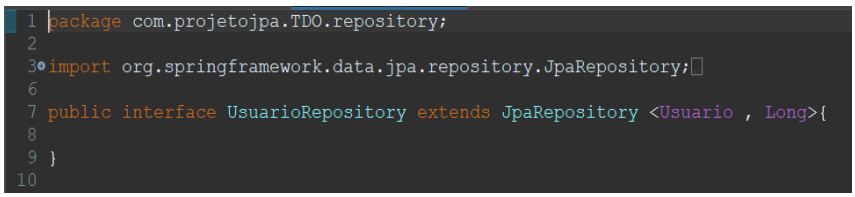
```java
package com.projetojpa.TDO.entities;

import jakarta.persistence.Entity;

@Data
@NoArgsConstructor
@Entity
@Table(name="usuario")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    @NotBlank
    private String nome;

    @NotNull
    @NotBlank
    private String senha;

    private String permisso;

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


}

```

- @Data serve para acesso e manipulação dos dados
- @NoArgsConstructor serve para gerar um construtor sem parâmetros
- @Entity nada mais é que a anotação que exemplifica que aquela camada é a de entidades
- @Table serve para dar nome a tabela

- @Id e @GeneratedValue serve para atribuir uma coluna na tabela, sendo essa coluna gerada automaticamente
- @NotNull e @NotBlank serve para dizer que o valor não pode ser nulo e branco

- Private String nome, senha e permissão são colunas que são privados, do tipo String onde pode haver números ou não, e o nome das colunas
- Por último o método construtor que atribui um valor para as colunas trazendo assim mais segurança

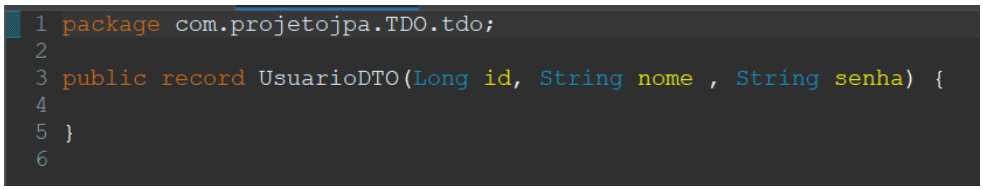
REPOSITORY -



```
package com.projetojpa.TDO.repository;

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

public interface UsuarioRepository extends JpaRepository <Usuario , Long>{

}

```

- Extends serve para herança
- JpaRepository serve como biblioteca que habilita os verbos para a camada service e assim, esta tem 2 parametros, Usuario e Long.

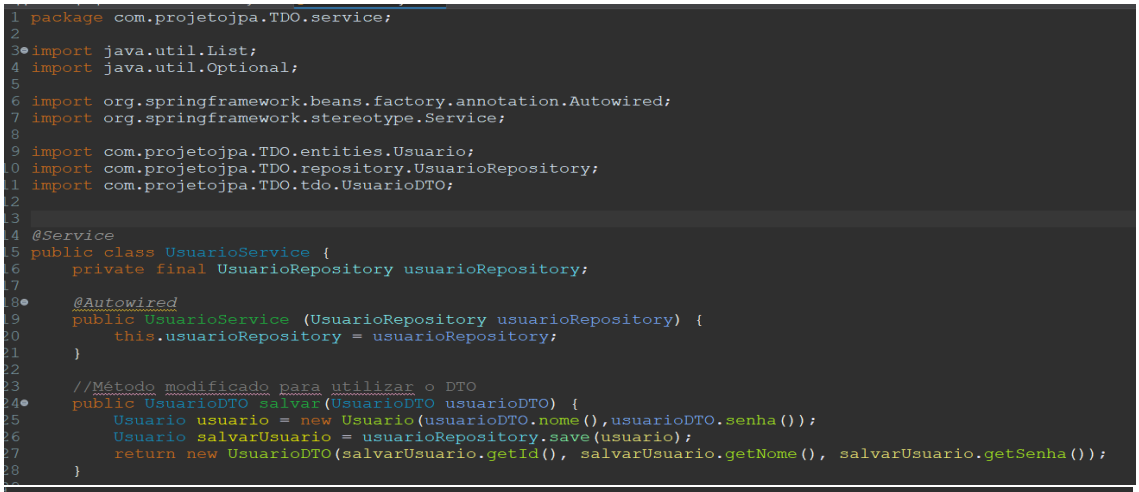
DTO -



```
package com.projetojpa.TDO.tdo;

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

}

```

- Padrão para transferência de dados entre as camadas, trazendo assim, mais segurança aos dados

SERVICE -



```java
package com.projetojpa.TDO.service;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.projetojpa.TDO.entities.Usuario;
import com.projetojpa.TDO.repository.UsuarioRepository;
import com.projetojpa.TDO.tdo.UsuarioDTO;


@Service
public class UsuarioService {
    private final UsuarioRepository usuarioRepository;

    @Autowired
    public UsuarioService (UsuarioRepository usuarioRepository) {
        this.usuarioRepository = usuarioRepository;
    }

    //Método modificado para utilizar o DTO
    public UsuarioDTO salvar(UsuarioDTO usuarioDTO) {
        Usuario usuario = new Usuario(usuarioDTO.nome(),usuarioDTO.senha());
        Usuario salvarUsuario = usuarioRepository.save(usuario);
        return new UsuarioDTO(salvarUsuario.getId(), salvarUsuario.getNome(), salvarUsuario.getSenha());
    }
```

```java
    public UsuarioDTO atualizar(Long id, UsuarioDTO usuarioDTO) {
        Usuario existeUsuario = usuarioRepository.findById(id).orElseThrow(() -> new RuntimeException("Usuario não encontrado"));

        existeUsuario.setNome(usuarioDTO.nome());
        existeUsuario.setSenha(usuarioDTO.senha());

        Usuario updateUsuario = usuarioRepository.save(existeUsuario);
        return new UsuarioDTO(updateUsuario.getId(), updateUsuario.getNome(), updateUsuario.getSenha());
    }

    public boolean deletarUsuario(Long id) {
        Optional <Usuario> existeUsuario = usuarioRepository.findById(id);
        if (existeUsuario.isPresent()) {
            usuarioRepository.deleteById(id);
            return true;
        }
        return false;
    }

    public List<Usuario> buscaTodosUsuario(){
        return usuarioRepository.findAll();
    }

    public Usuario BuscaUsuarioId(Long id) {
        Optional <Usuario> Usuario = usuarioRepository.findById(id);
        return Usuario.orElse(null);
    }
}
```

- @Service serve para mostrar que aquela é a camada Service
- @Autowired serve para a injeção de dependência e conecta ao repository
- Métodos já modificados para ser utilizado o DTO
  - O verbo salvar recebe o “pacote de dados” que já foi passado pelo controller e o DTO e assim, transporta os dados até a camada entidades para que sejam salvos
  - O verbo atualizar recebe o “pacote de dados” que já foi passado pelo controller e o DTO e assim, transporta os dados até a camada entidades para que sejam atualizados
- Os verbos buscar e deletar não sofreram alteração e, portanto, recebem os dados diretamente do controller porque só servem para conferir se tem algum dado e se for para deletar, apagar este dado, de modo que não influenciam na entrada de dados

CONTROLLER -

```java
package com.projetojpa.TDO.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.projetojpa.TDO.entities.Usuario;
import com.projetojpa.TDO.service.UsuarioService;
import com.projetojpa.TDO.tdo.UsuarioDTO;
import jakarta.validation.Valid;

@RestController
@RequestMapping("/usuario")
public class UsuarioController {

    private final UsuarioService usuarioService;

    @Autowired
    public UsuarioController (UsuarioService usuarioService) {
        this.usuarioService = usuarioService;
    }
```

```java
    @GetMapping("/{id}")
    public ResponseEntity<Usuario> buscaUsuarioControlId(@PathVariable Long id){
        Usuario usuario = usuarioService.BuscaUsuarioId(id);
        if (usuario != null) {
            return ResponseEntity.ok(usuario);
        }
        else {
            return ResponseEntity.notFound().build();
        }
    }

    @GetMapping
    public ResponseEntity<List<Usuario>> buscaTodosUsuarioCOntrol(){
        List<Usuario> Usuario = usuarioService.buscaTodosUsuario();
        return ResponseEntity.ok(Usuario);
    }

    @PostMapping
    public ResponseEntity<UsuarioDTO> criar(@RequestBody @Valid UsuarioDTO usuarioDTO){
        UsuarioDTO salvarUsuario = usuarioService.salvar(usuarioDTO);
        return ResponseEntity.status(HttpStatus.CREATED).body(salvarUsuario);
    }
```

```java
    @PutMapping("/{id}")
    public ResponseEntity<UsuarioDTO> alterar (@PathVariable Long id, @RequestBody @Valid UsuarioDTO usuarioDTO){
        UsuarioDTO alteraUsuarioDTO = usuarioService.atualizar(id, usuarioDTO);
        if(alteraUsuarioDTO != null) {
            return ResponseEntity.ok(alteraUsuarioDTO);
        }
        else {
            return ResponseEntity.notFound().build();
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Usuario> apagaUsuarioControl (@PathVariable Long id){
        boolean apagar = usuarioService.deletarUsuario(id);
        if(apagar) {
            return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
        }
        else {
            return ResponseEntity.notFound().build();
        }
    }
}
```

- @RestController serve para dizer que esta é a camada controller que recebe os dados
- @RequestMapping serve para mapear as requisições da tabela
- @Autowired serve para a injeção de dependência e conecta ao repository
- Métodos já modificados para ser utilizado o DTO

- O verbo post recebe os dados e assim, transporta os dados até a camada service para que sejam salvos
- O verbo put recebe os dados e assim, transporta os dados até a camada service para que sejam alterados

- Os verbos get e delete não sofreram alteração e, portanto, recebem os dados e servem para conferir se tem algum dado e se for para deletar, apagar este dado, de modo que não influenciam na entrada de dados

APPLICATION -

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa
spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.defer-datasource-initialization=true
spring.jpa.hibernate.ddl-auto=update
spring.h2.console.enabled=true

```

- Serve para que todas as camadas recebam a aplicação