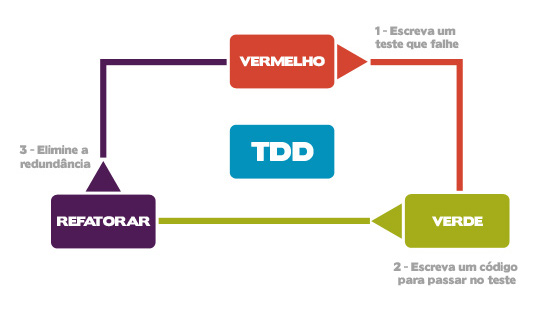
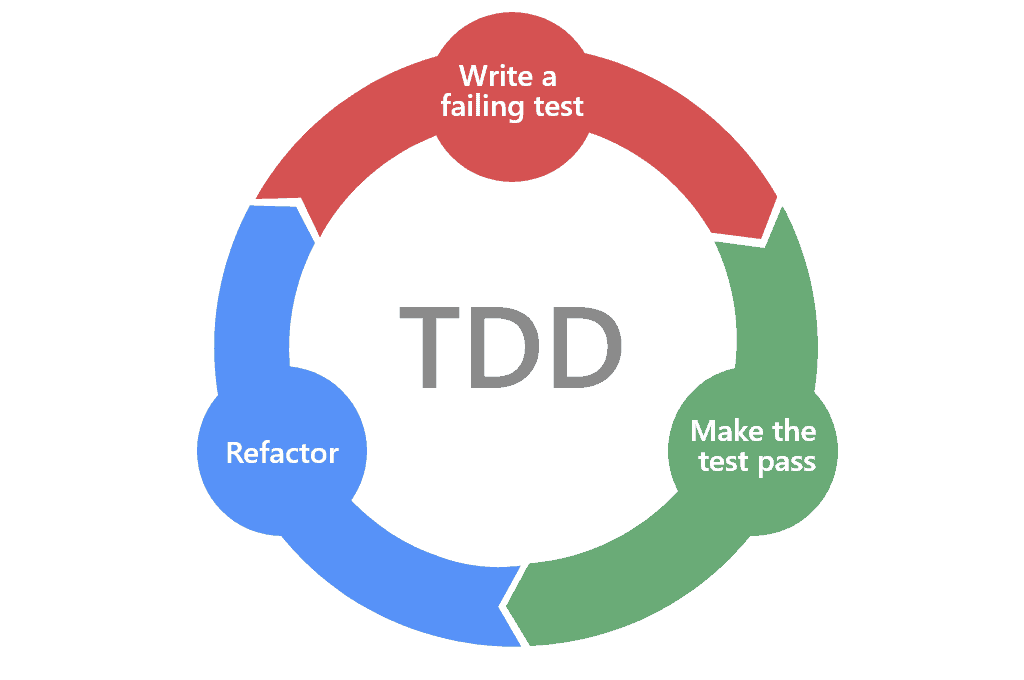
**TDD – Test Driven Development**

Metodologia para desenvolvimento de aplicações baseado na implementação e execução de testes realizados pela própria equipe de desenvolvimento.

**TDD** é uma sigla para **Test Driven Development**, ou *Desenvolvimento Orientado a Testes*. A ideia do TDD é que você trabalhe em ciclos. Estes ciclos ocorrem na seguinte ordem:



* **Red:** escreva um pequeno teste automatizado que, ao ser executado, irá falhar;
* **Green:** implemente um código que seja suficiente para ser aprovado no;
* **Refactor:** refatore o código, deixando-o mais funcional e mais limpo.



**Todo projeto criado com Spring Boot já possui uma classe padrão para desenvolvimento de testes:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

package br.com.cotiinformatica;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import java.util.Locale;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.web.servlet.MockMvc;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.github.javafaker.Faker;

import br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

import br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

import br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

@SpringBootTest

@AutoConfigureMockMvc

class ApiUsuariosApplicationTests {

@Autowired

private MockMvc mock;

@Autowired

private ObjectMapper objectMapper;

@Test

public void postAutenticarTest() throws Exception {

PostAutenticarDTO dto = new PostAutenticarDTO();

dto.setEmail("admin@cotiinformatica.com.br");

dto.setSenha("@Admin123");

mock.perform(

post("/api/autenticar")

.contentType("application/json")

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isOk());

}

@Test

public void postCriarContaTest() throws Exception {

PostCriarContaDTO dto = new PostCriarContaDTO();

Faker faker = new Faker(new Locale("pt-BR"));

dto.setNome(faker.name().fullName());

dto.setEmail(faker.internet().emailAddress());

dto.setSenha(faker.internet().password(8, 20));

mock.perform(

post("/api/criar-conta")

.contentType("application/json")

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isCreated());

}

@Test

public void postRecuperarSenhaTest() throws Exception {

PostRecuperarSenhaDTO dto = new PostRecuperarSenhaDTO();

dto.setEmail("teste@cotiinformatica.com.br");

mock.perform(

post("/api/recuperar-senha")

.contentType("application/json")

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isOk());

}

}

**Executando:**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Para construirmos a API, vamos usar para conexão com o banco de dados a biblioteca do **Spring Data** através do **Hibernate & JPA**



**Primeiro, precisamos criar um banco de dados no PostGreSQL:**

Abrindo o PgAdmin

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

/pom.xml

Instalando as bibliotecas necessárias para

desenvolvermos com Hibernate & JPA.

<dependency>

<groupId>**org.springframework.boot**</groupId>

<artifactId>**spring-boot-starter-data-jpa**</artifactId>

</dependency>

<dependency>

<groupId>**org.springframework.boot**</groupId>

<artifactId>**spring-boot-autoconfigure**</artifactId>

</dependency>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Configurando o DATA SOURCE do projeto:**

Fazendo a configuração para conexão no banco de dados.

/application.properties

server.port=8083

spring.datasource.url=**jdbc:postgresql://localhost:5432/bd\_apiusuarios**

spring.datasource.driver-class-name=**org.postgresql.Driver**

spring.datasource.username=**postgres**

spring.datasource.password=**coti**

spring.jpa.properties.hibernate.dialect=**org.hibernate.dialect.PostgreSQLDialect**

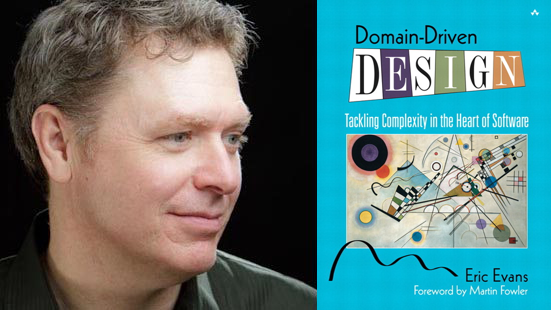
spring.jpa.hibernate.ddl-auto=**update**

spring.jpa.show-sql=**true**

spring.jpa.properties.hibernate.format\_sql=**true**

**DDD – Domain Driven Design**

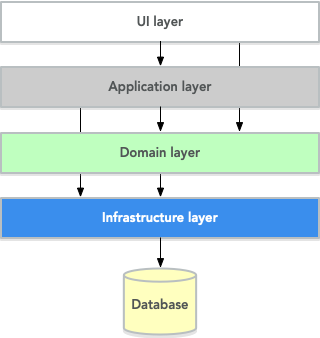
Domain Driven Design significa Projeto Orientado a Domínio. Ele veio do título do livro escrito por Eric Evans, dono da Domain Language,  uma empresa especializada em treinamento e consultoria para desenvolvimento de software. O livro de Evans é um grande catálogo de Padrões, baseados em experiências do autor ao longo de mais de 20 anos desenvolvendo software utilizando técnicas de Orientação a Objetos.



A principal ideia do DDD (Domain Driven Design) é a de que o mais importante em um software não é o seu código, nem sua arquitetura, nem a tecnologia sobre a qual foi desenvolvido, mas sim o problema que o mesmo se propõe a resolver, ou em outras palavras, a regra de negócio.

Ela é a razão do software existir, por isso deve receber o máximo de tempo e atenção possíveis.

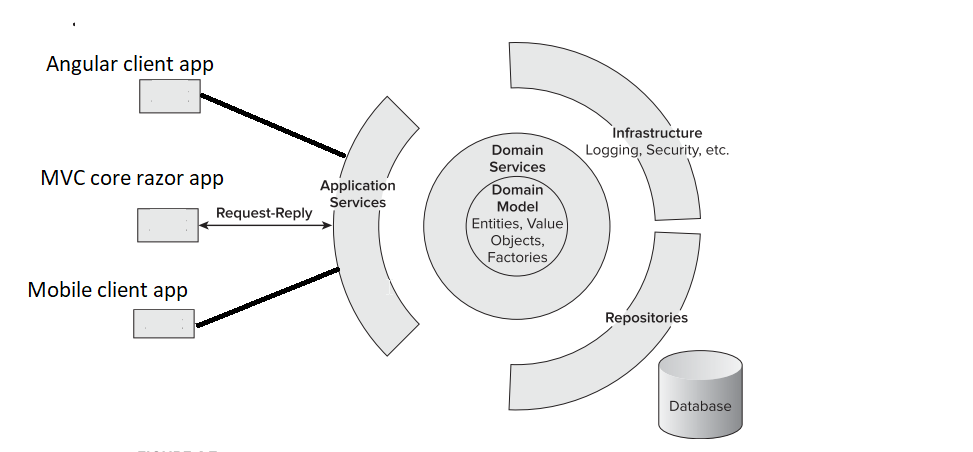
Em praticamente todos os projetos de software, a complexidade não está localizada nos aspectos técnicos, mas sim no negócio, na atividade que é exercida pelo cliente ou problema que o mesmo possui.



Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente

A principal parte do design DDD é o domínio. Ou seja, no domínio implementamos o modelo de entidades do projeto bem como as regras de negócio do sistema.



**Domínio:**

Modelo do domínio, serviços e interfaces dos repositórios. Esta é a parte central do aplicativo. A linguagem obiqua é usada nessas classes, interfaces e assinaturas dos métodos,  e todo conceito dentro dessa camada é familiar para o especialista no domínio.

O Domínio é composto de:

**Domain Models** (Modelos de domínio)

O padrão *Entity* define que todo sistema possui ao menos uma entidade, ou seja, todo sistema possui ao menos um objeto o qual irá refletir um conceito do domínio, **com sua respectiva identidade e estado**.

**Domain Services** (Serviços de domínio)

Os serviços do domínio (*Domain Services*) são classes que tem como objetivo serem uma alternativa para o **desacoplamento de código**. Os serviços do domínio surgem em cenários onde a escolha de dar responsabilidade a uma classe ou outra poderia causar problemas com **acoplamento do código**.

Ou ainda, quando uma responsabilidade nova não se encaixa em nenhuma das Entidades já definidas.

**Primeiro, vamos criar os modelos de domínio através de classes JAVABEAN que representem as entidades do sistema:**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** br.com.cotiinformatica.domain.models;

**import** java.util.Date;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

@Setter

@Getter

@NoArgsConstructor

@AllArgsConstructor

@ToString

**public** **class** Usuario {

**private** Integer idUsuario;

**private** String nome;

**private** String email;

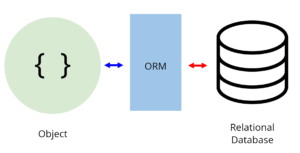
**private** String senha;

**private** Date dataHoraCriacao;

}

**ORM – Mapeamento Objeto Relacional**

Nesta etapa, precisamos mapear os modelos de domínio de forma a traduzir quais tabelas eles representam no banco de dados.



Este mapeamento será feito em Java através da biblioteca **javax.persistence** (pertence a JPA – Java Persistence API)

**package** br.com.cotiinformatica.domain.models;

**import** java.util.Date;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

**import** javax.persistence.Temporal;

**import** javax.persistence.TemporalType;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

@Entity

@Table(name = "usuario")

@Setter

@Getter

@NoArgsConstructor

@AllArgsConstructor

@ToString

**public** **class** Usuario {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "idusuario")

**private** Integer idUsuario;

@Column(name = "nome", length = 150, nullable = **false**)

**private** String nome;

@Column(name = "email", length = 100, nullable = **false**,

unique = **true**)

**private** String email;

@Column(name = "senha", length = 50, nullable = **false**)

**private** String senha;

@Temporal(TemporalType.***TIMESTAMP***)

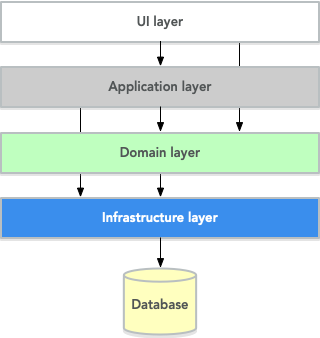
@Column(name = "datahoracriacao", nullable = **false**)

**private** Date dataHoraCriacao;

}

**Infraestrutura**

Camada do projeto DDD onde desenvolvemos a parte de conexão com o banco de dados e quaisquer outros serviços que sejam necessários para que o domínio possa executar as suas regras de negócio.



Criando o repositório para conexão com o banco de dados.

Estes repositórios serão feitos através de interfaces:

/repositories/IUsuarioRepository.java

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** br.com.cotiinformatica.infra.repositories;

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

**import** org.springframework.stereotype.Repository;

**import** br.com.cotiinformatica.domain.models.Usuario;

@Repository

**public** **interface** IUsuarioRepository

**extends** JpaRepository<Usuario, Integer> {

}

**JPQL – Java Persistence Query Language**

Sintaxe para construção de consultas em banco de dados através da JPA (Java Persistence API), tem como principal característica o uso de uma sintaxe própria para consulta de dados que não utiliza SQL.

**package** br.com.cotiinformatica.infra.repositories;

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

**import** org.springframework.data.jpa.repository.Query;

**import** org.springframework.data.repository.query.Param;

**import** org.springframework.stereotype.Repository;

**import** br.com.cotiinformatica.domain.models.Usuario;

@Repository

**public** **interface** IUsuarioRepository **extends** JpaRepository<Usuario, Integer> {

**@Query("select u from Usuario u where u.email = :pEmail")**

Usuario findByEmail(@Param("pEmail") String email);

**@Query("select u from Usuario u where u.email = :pEmail**

**and u.senha = :pSenha")**

Usuario findByEmailAndSenha(**@Param("pEmail")** String email,

**@Param("pSenha")** String senha);

}

**Executando:**

Criando as tabelas no banco de dados:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Uma imagem contendo Calendário

Descrição gerada automaticamente

Hibernate:

**create table usuario (**

**idusuario serial not null,**

**datahoracriacao timestamp not null,**

**email varchar(100) not null,**

**nome varchar(150) not null,**

**senha varchar(50) not null,**

**primary key (idusuario)**

**)**

Hibernate:

**alter table usuario**

**drop constraint UK\_5171l57faosmj8myawaucatdw**

Hibernate:

**alter table usuario**

**add constraint UK\_5171l57faosmj8myawaucatdw unique (email)**

**Domain Services** (Serviços de domínio)

Os serviços do domínio (*Domain Services*) são classes que tem como objetivo serem uma alternativa para o **desacoplamento de código**. Os serviços do domínio surgem em cenários onde a escolha de dar responsabilidade a uma classe ou outra poderia causar problemas com **acoplamento do código**.

Ou ainda, quando uma responsabilidade nova não se encaixa em nenhuma das Entidades já definidas.

* Primeiro, vamos criar interfaces, através do qual serão desenvolvidos os **serviços de domínio**:

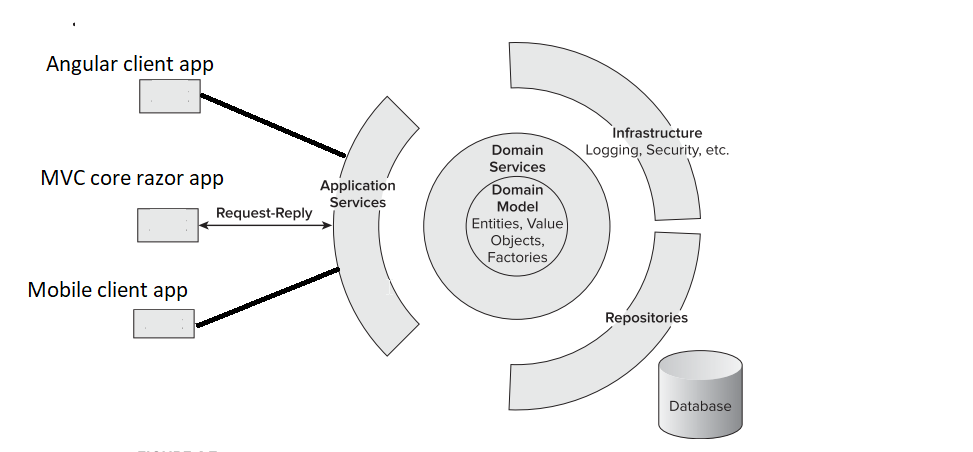
/domain/interfaces/IUsuarioDomainService.java

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente



**package** br.com.cotiinformatica.domain.interfaces;

**import** br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseRecuperarSenhaDTO;

**public** **interface** IUsuarioDomainService {

// Método para implementar as regras de negócio para

// autenticação do usuário

ResponseAutenticarDTO autenticar(PostAutenticarDTO dto);

// Método para implementar as regras de negócio para

// criação da conta do usuário

ResponseCriarContaDTO criarConta(PostCriarContaDTO dto);

// Método pata implementar as regras de negócio para

// recuperação da senha do usuário

ResponseRecuperarSenhaDTO recuperarSenha

(PostRecuperarSenhaDTO dto);

}

Implementando a interface de serviço de domínio:

/services/UsuarioDomainService.java

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** br.com.cotiinformatica.domain.services;

**import** org.springframework.stereotype.Service;

**import** br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseRecuperarSenhaDTO;

**import** br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

@Service

**public** **class** UsuarioDomainService **implements** IUsuarioDomainService {

@Override

**public** ResponseAutenticarDTO autenticar(PostAutenticarDTO dto) {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** ResponseCriarContaDTO criarConta(PostCriarContaDTO dto) {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** ResponseRecuperarSenhaDTO recuperarSenha

(PostRecuperarSenhaDTO dto) {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

As classes de serviço de domínio no DDD irão constantemente acessar a infraestrutura do projeto. Ou seja, irão acessar interfaces que compõem a camada de infraestrutura do sistema como por exemplo, as interfaces de repositório.

**package** br.com.cotiinformatica.domain.services;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseRecuperarSenhaDTO;

**import** br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

**import** br.com.cotiinformatica.infra.repositories.IUsuarioRepository;

@Service

**public** **class** UsuarioDomainService **implements** IUsuarioDomainService {

**@Autowired //injeção de dependência**

**private IUsuarioRepository usuarioRepository;**

@Override

**public** ResponseAutenticarDTO autenticar(PostAutenticarDTO dto) {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** ResponseCriarContaDTO criarConta(PostCriarContaDTO dto) {

//verificar se já existe um usuário

//cadastrado com o email informado

**if**(usuarioRepository.findByEmail(dto.getEmail()) != **null**)

**throw** **new** IllegalArgumentException

("O email informado já está cadastrado.

Tente outro.");

**return** **null**;

}

@Override

**public** ResponseRecuperarSenhaDTO recuperarSenha

(PostRecuperarSenhaDTO dto) {

// **TODO** Auto-generated method stub

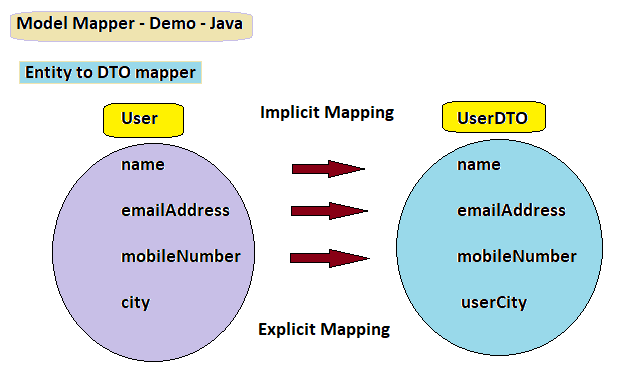
**return** **null**;

}

}

Instalando no projeto a biblioteca **ModelMapper**

Utilizada para fazermos a transferência de dados entre objetos, geralmente de DTOs para modelos de entidade e vice versa.



/pom.xml

<dependency>

<groupId>**org.modelmapper.extensions**</groupId>

<artifactId>**modelmapper-spring**</artifactId>

<version>**3.0.0**</version>

</dependency>

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

**Voltando na classe de serviço de domínio:**

**package** br.com.cotiinformatica.domain.services;

**import** java.math.BigInteger;

**import** java.security.MessageDigest;

**import** java.security.NoSuchAlgorithmException;

**import** java.util.Date;

**import** org.modelmapper.ModelMapper;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** br.com.cotiinformatica.application.dtos.GetUsuarioDTO;

**import** br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseRecuperarSenhaDTO;

**import** br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

**import** br.com.cotiinformatica.domain.models.Usuario;

**import** br.com.cotiinformatica.infra.repositories.IUsuarioRepository;

@Service

**public** **class** UsuarioDomainService **implements** IUsuarioDomainService {

@Autowired //injeção de dependência

**private** IUsuarioRepository usuarioRepository;

@Override

**public** ResponseAutenticarDTO autenticar(PostAutenticarDTO dto) {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** ResponseCriarContaDTO criarConta(PostCriarContaDTO dto) {

//verificar se já existe um usuário

//cadastrado com o email informado

**if**(usuarioRepository.findByEmail(dto.getEmail()) != **null**)

**throw** **new** IllegalArgumentException

("O email informado já está cadastrado.

Tente outro.");

//transferir os dados do DTO para

//a classe de modelo de entidade

ModelMapper modelMapper = **new** ModelMapper();

Usuario usuario = modelMapper.map(dto, Usuario.**class**);

usuario.setSenha(*criptografarSenha*(usuario.getSenha()));

usuario.setDataHoraCriacao(**new** Date());

//gravando no banco de dados

usuarioRepository.save(usuario);

ResponseCriarContaDTO response

= **new** ResponseCriarContaDTO();

response.setStatus(201);

response.setMensagem("Usuário cadastrado com sucesso");

response.setDataHoraCadastro(**new** Date());

response.setUsuario(modelMapper.map

(usuario, GetUsuarioDTO.**class**));

**return** response;

}

@Override

**public** ResponseRecuperarSenhaDTO recuperarSenha

(PostRecuperarSenhaDTO dto) {

// **TODO** Auto-generated method stub

**return** **null**;

}

//Método para realizar a criptografia da senha

**private** **static** String criptografarSenha(String value) {

MessageDigest md;

**try** {

md = MessageDigest.*getInstance*("MD5");

} **catch** (NoSuchAlgorithmException e) {

**throw** **new** RuntimeException(e);

}

BigInteger hash = **new** BigInteger(1, md.digest

(value.getBytes()));

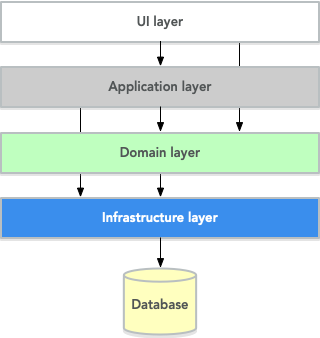
**return** hash.toString(16);

}

}

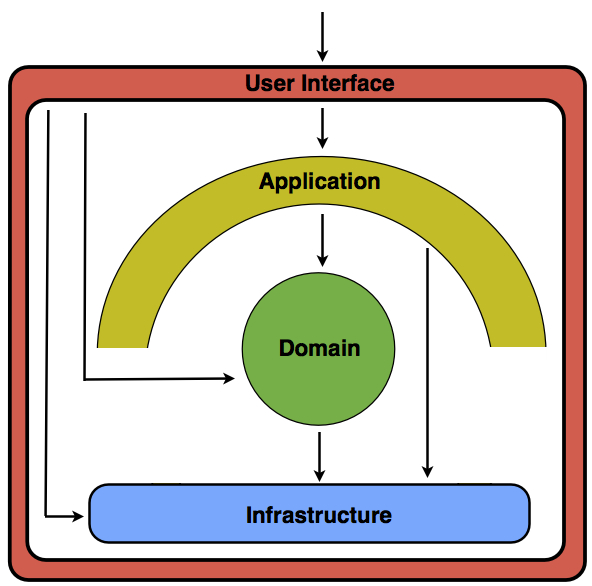
**Application**

Camada do DDD que conecta o domínio com o mundo externo e orquestra os fluxos da aplicação. Neste projeto, a camada de aplicação está representada pelas classes de controle (controladores):



Na camada de aplicação não é implementada nenhuma regra de negócio, ela somente coordena a execução de uma tarefa e delega para os objetos de domínio na camada inferior.

Camada responsável por fazer a(s) aplicação(s) se comunicar diretamente com o Domínio.



A camada de aplicação precisa acessar os serviços de domínio, mas ela não irá acessar as classes de serviço de domínio mas sim as interfaces:

**package** br.com.cotiinformatica.application.controllers;

**import** javax.validation.Valid;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RestController;

**import** br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

**import** br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

**import** io.swagger.annotations.Api;

**import** io.swagger.annotations.ApiOperation;

@Api(tags = "Criação de conta de usuários")

@RestController

**public** **class** CriarContaController {

**@Autowired // injeção de dependência**

**private IUsuarioDomainService usuarioDomainService;**

@ApiOperation("ENDPOINT para cadastro de usuários.")

@PostMapping("/api/criar-conta")

**public** ResponseEntity<ResponseCriarContaDTO> post

(@Valid @RequestBody PostCriarContaDTO dto) {

**return** ResponseEntity.*status*

(HttpStatus.***NOT\_IMPLEMENTED***).body(**null**);

}

}

**Finalizando o controlador:**

**package** br.com.cotiinformatica.application.controllers;

**import** javax.validation.Valid;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RestController;

**import** br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

**import** br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

**import** io.swagger.annotations.Api;

**import** io.swagger.annotations.ApiOperation;

@Api(tags = "Criação de conta de usuários")

@RestController

**public** **class** CriarContaController {

@Autowired // injeção de dependência

**private** IUsuarioDomainService usuarioDomainService;

@ApiOperation("ENDPOINT para cadastro de usuários.")

@PostMapping("/api/criar-conta")

**public** ResponseEntity<ResponseCriarContaDTO> post

(@Valid @RequestBody PostCriarContaDTO dto) {

ResponseCriarContaDTO response = **null**;

**try** {

response = usuarioDomainService.criarConta(dto);

**return** ResponseEntity.*status*

(HttpStatus.***CREATED***).body(response);

}

**catch**(IllegalArgumentException e) {

response = **new** ResponseCriarContaDTO();

response.setStatus(400);

//BAD REQUEST (CLIENT ERROR)

response.setMensagem(e.getMessage());

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***)

.body(response);

}

**catch**(Exception e) {

response = **new** ResponseCriarContaDTO();

response.setStatus(500); //INTERNAL SERVER ERROR

response.setMensagem(e.getMessage());

**return** ResponseEntity.*status*

(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(response);

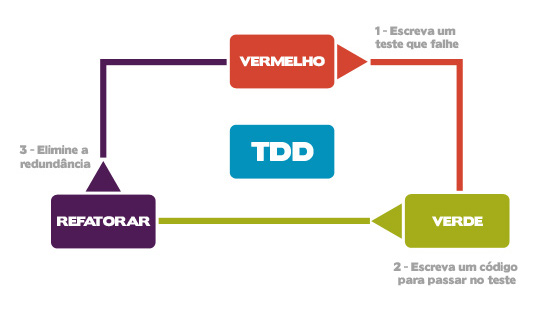
}

}

}

Voltando ao **TDD**, vamos executar os testes:

Test Driven Development



**Executando os testes:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**No PgAdmin:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Executando a API:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

<http://localhost:8083/swagger-ui/index.html>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Site

Descrição gerada automaticamente

<http://localhost:8083/v2/api-docs>

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**POST** http://localhost:8083/api/criar-conta

**REQUEST**

**{**

**"email": "sergio.coti@gmail.com",**

**"nome": "Sergio Mendes",**

**"senha": "@Admin123"**

**}**

**RESPONSE**

**{**

**"status": 201,**

**"mensagem": "Usuário cadastrado com sucesso",**

**"usuario": {**

**"idUsuario": 3,**

**"nome": "Sergio Mendes",**

**"email": "sergio.coti@gmail.com"**

**},**

**"dataHoraCadastro": 1678323293263**

**}**