**AVALIAÇÃO (AC2): CLEAN ARCHITECTURE E CI/CD**

**Objetivo**:

* Aplicar Clean Architecture (CA) e Domain Driven Development (DDD) em Arquitetura de Software.
* Realizar Testes de Software, considerando as camadas do CA e DDD.
* Implantar arquitetura DEVOPS (CI/CD) com Jenkins, visando 3 pipelines: DEV, Homologação (Staging) e Produção.
* Utilizar as Tecnologias Spring Boot (Java e Maven) e Jenkins (FreeStyle e Pipeline) para implementação da Prática.
* Rodar a API via Tecnologia Docker.

**Atividade 7.1: CA e DDD: Gerando o projeto**

1. Criar uma aplicação Spring Boot considerando as seguintes dependências e recursos:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

1. Abrir a aplicação no Eclipse e verificar o pom.xml do projeto e analisar se as dependências supramencionadas foram geradas.
2. Incluir o plugin do Jacoco (você já sabe fazer isso). Resgatar do seu projeto de AC1. plugin do Jacoco versão 0.8.12.
3. Printar a evidência da aplicação rodando (executar Run/Clean Install, e depois Run/ Java Aplication).

Run/clean:

Texto

Descrição gerada automaticamente

Rub/install maven:

Texto

Descrição gerada automaticamente

Run/java application:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

**Atividade 7.2: CA e DDD – Camadas: Entity (com Value Objets) e Repository com Spring Data**

1. Resgatar a sua aplicação TDD desenvolvida em AC1.
2. Definir as Entities da aplicação.
3. Para cada Entity verificar quais serão os Value Objects, com objetivo de identificar os objetos de valor que caracterizam cada Entity, conforme discutido em aula.

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story** | **BDD** | **Entity** | **Value Objects** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Criar a estrutura do item “C” supra definido no Eclipse (Java Spring Boot). Veja um exemplo considerando a arquitetura de CA/DDD para a Entity “USER”. Considerar seu projeto, OK?

Texto

Descrição gerada automaticamente com confiança média

**Atividade 7.3: Arquitetura CA/DDD: Criando o Pacote Entity (inclui os value objects) e Repository.**

1. Desenvolver a Entity com “annotations” para persistência em base de dados, seguindo o padrão ORM (mapeamento Objeto Relacional).

No pacote Entity criar o Objeto de Valor, conforme exemplo.   
Obs: seguir o objeto de valor do seu projeto ok? Pode ser email, mas veja a coerência com seu projeto.

**package** ac2\_project.example.ac2\_ca.entity;

**import** java.util.Objects;

**import** jakarta.persistence.Embeddable;

@Embeddable

**public** **class** User\_Email {

**private** String emailAddress;

// Construtor padrão necessário para o JPA

**protected** User\_Email() {}

**public** User\_Email(String emailAddress) {

**if** (emailAddress == **null** || !emailAddress.matches("^[A-Za-z0-9+\_.-]+@[A-Za-z0-9.-]+$")) {

**throw** **new** IllegalArgumentException("Email inválido");

}

**this**.emailAddress = emailAddress;

}

**public** String getEmailAddress() {

**return** emailAddress;

}

// Sobrescreva equals e hashCode para garantir comparação por valor

@Override

**public** **boolean** equals(Object o) {

**if** (**this** == o) **return** **true**;

**if** (o == **null** || getClass() != o.getClass()) **return** **false**;

User\_Email email = (User\_Email) o;

**return** Objects.*equals*(emailAddress, email.emailAddress);

}

@Override

**public** **int** hashCode() {

**return** Objects.*hash*(emailAddress);

}

}

1. Criar a Classe Entity

package ac2\_project.example.ac2\_ca.entity;

import jakarta.persistence.Embedded;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "tb\_user")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

@Embedded

private User\_Email email;

// Getters e Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public User\_Email getEmail() {

return email;

}

public void setEmail(User\_Email email) {

this.email = email;

}

}

1. Na camada Repository, criar:

package ac2\_project.example.ac2\_ca.repository;

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

import org.springframework.stereotype.Repository;

import ac2\_project.example.ac2\_ca.entity.User;

@Repository

public interface User\_Repository extends JpaRepository<User, Long> {

1. Na pasta “Resources” do Spring Boot. Resgatar o arquivo “application.properties. Configurar o banco em memória H2 para testar a verificar o ORM funcionando.

# data source

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.url=jdbc:h2:mem:blog-praticandoAPI

spring.datasource.username=sa

spring.datasource.password=

# jpa

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=update

# h2

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

1. Salvar a aplicação. Run/ Maven Clean. Run/Maven Install. Run/JavaApplication. Veja o servidor (container web Tomcat – Rodando). Veja que o ORM funcionou com JPA e que o H2 está disponível em localhost h2/console.

Texto

Descrição gerada automaticamente

1. Faça rodar o ORM do seu Projeto, conforme exemplo abaixo, evidenciando que o ORM foi realizado considerado a Classe Entity como Table na base de dados e o Value Object como coluna da tabela.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente