FECAP

PROJETO EcoIA Game

Requisitos da disciplina Modelagem de Software e Arquitetura de Sistemas

São Paulo

2024

INTEGRANTES DO PROJETO e RA’S

Guilherme de Amorim Medeiros.....23024555
Lorena Pereira Bernardo.....23025048
Matheus Sampaio Duarte.....23034588
Paulo Carvalho Silva Junior.....23024564

Contents

1. INTRODUÇÃO

Os testes são fundamentais para garantir a qualidade e funcionalidade do software, abrangendo requisitos como funcionalidades, desempenho e segurança. Eles incluem diversos tipos, como unitários, de integração e de aceitação do usuário. A escolha de estratégias e ferramentas adequadas, incluindo a automação, é crucial. A prática de integração contínua e entrega contínua agiliza o processo de desenvolvimento e identificação de problemas. Os testes são realizados em todas as etapas do ciclo de vida do desenvolvimento. Integrá-los em cada etapa é essencial para garantir qualidade consistente do produto. Essa introdução fornece uma visão geral dos elementos essenciais de uma rotina de testes básica.Apresentar 2 testes unitários.

1.1. Apresentar 2 testes unitários.

Primeiro Teste:

Teste de Criptografia e Descriptografia do Banco de Dados

Primeira parte:

```java
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.util.Base64;

class CryptographyTest {
    @Test
    void testDescriptografarDados() {
        // Dados de teste
        String dadosOriginais = "João Silva";
        String chaveBase64 = "chaveBase64";
        String ivBase64 = "ivBase64";

        // Criptografa os dados de teste
        String dadosCriptografados = criptografarDados(dadosOriginais, chaveBase64, ivBase64);

        // Descriptografa os dados criptografados
        String dadosDescriptografados = descriptografarDados(dadosCriptografados, chaveBase64, ivBase64);

        // Verifica se os dados descriptografados correspondem aos dados originais
        Assertions.assertEquals(dadosOriginais, dadosDescriptografados);
    }

    @Test
    void testDescriptografarDadosComErro() {
        // Dados de teste inválidos
        String dadosCriptografados = "dadosCriptografadosInvalidos";
        String chaveBase64 = "chaveBase64Invalida";
        String ivBase64 = "ivBase64Invalido";

        // Tenta descriptografar os dados inválidos
        String dadosDescriptografados = descriptografarDados(dadosCriptografados, chaveBase64, ivBase64);

        // Verifica se a função retorna null em caso de erro
        Assertions.assertNull(dadosDescriptografados);
    }
```

Segunda Parte:

```java
    private String criptografarDados(String dados, String chaveBase64, String ivBase64) {
        try {
            // Decodifica a chave e o IV de Base64 para bytes
            byte[] chave = Base64.decode(chaveBase64, Base64.DEFAULT);
            byte[] iv = Base64.decode(ivBase64, Base64.DEFAULT);

            // Inicializa o objeto Cipher para criptografia com o algoritmo AES/CBC/PKCS5PADDING
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
            cipher.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(chave, "AES"),
                    new IvParameterSpec(iv));

            // Criptografa os dados
            byte[] dadosCriptografadosBytes = cipher.doFinal(dados.getBytes(StandardCharsets.UTF_8));

            // Codifica os dados criptografados em Base64
            return Base64.encodeToString(dadosCriptografadosBytes, Base64.DEFAULT);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    private String descriptografarDados(String dadosCriptografados, String chaveBase64,
                                        String ivBase64) {
        try {
            // Decodifica a chave e o IV de Base64 para bytes
            byte[] chave = Base64.decode(chaveBase64, Base64.DEFAULT);
            byte[] iv = Base64.decode(ivBase64, Base64.DEFAULT);

            // Inicializa o objeto Cipher para descriptografia com o algoritmo AES/CBC/PKCS5PADDING
            Cipher decipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
            decipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(chave, "AES"),
                    new IvParameterSpec(iv));

            // Descriptografa os dados criptografados
            byte[] dadosDescriptografadosBytes = decipher.doFinal(Base64.decode(dadosCriptografados,
                    Base64.DEFAULT));

            // Converte os dados descriptografados de bytes para String
            return new String(dadosDescriptografadosBytes, StandardCharsets.UTF_8);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Segundo Teste:

Teste para gerar o Nickname do Usuário automaticamente

Primeira Parte:

```java
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.util.Base64;

class CryptographyTest {

    // Teste para verificar se o IV (Initialization Vector) gerado aleatoriamente tem o tamanho correto
    @Test
    void testGerarIVAleatorio() {
        byte[] iv = gerarIVAleatorio();
        Assertions.assertEquals(16, iv.length);
    }

    // Teste para verificar se a chave gerada a partir do nome tem o tamanho correto
    @Test
    void testGerarChaveNome() {
        String dados = "João Silva";
        String chaveNome = gerarChaveNome(dados);
        byte[] chaveBytes = Base64.getDecoder().decode(chaveNome);
        Assertions.assertEquals(32, chaveBytes.length);
    }

    // Teste para verificar se a chave gerada a partir do apelido tem o tamanho correto
    @Test
    void testGerarChaveNickname() {
        String dados = "JoãoSilva";
        String chaveNickname = gerarChaveNickname(dados);
        byte[] chaveBytes = Base64.getDecoder().decode(chaveNickname);
        Assertions.assertEquals(32, chaveBytes.length);
    }
```

Segunda Parte:

Teste para criptografar os dados do Usuário antes de enviar para o Anco de Dados

```java
    // Teste para verificar se a criptografia do nome não retorna nulo
    @Test
    void testCriptografarNome() {
        String nome = "João Silva";
        String chaveNome = gerarChaveNome(nome);
        byte[] iv = gerarIVAleatorio();
        String dadosCriptografados = criptografarNome(nome, chaveNome, iv);
        Assertions.assertNotNull(dadosCriptografados);
    }

    // Teste para verificar se a criptografia do apelido não retorna nulo
    @Test
    void testCriptografarNickname() {
        String nickname = "JoãoSilva";
        String chaveNickname = gerarChaveNickname(nickname);
        byte[] iv = gerarIVAleatorio();
        String dadosCriptografados = criptografarNickname(nickname, chaveNickname, iv);
        Assertions.assertNotNull(dadosCriptografados);
    }
}
```

1.1. Apresentar um teste de sistema.

Os Testes de Componentes foram realizados por meio de vídeo via arquivo: <TesteQualDevOps_atividade_PI.Mp4> conforme o link abaixo:

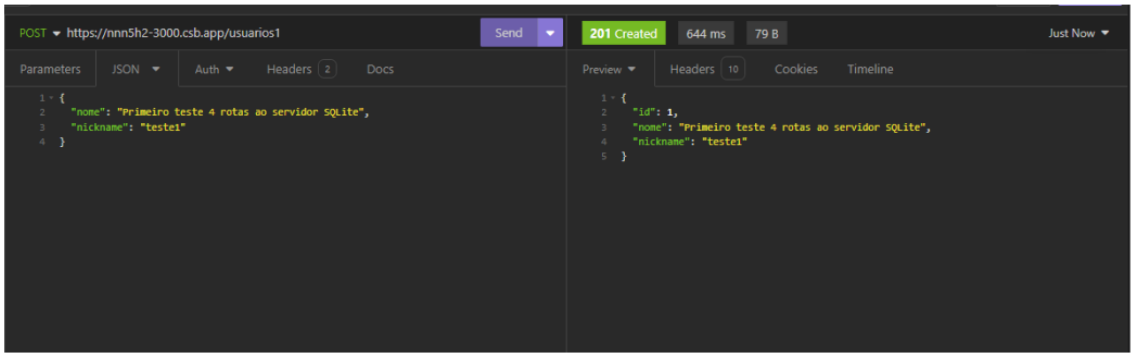
https://github.com/2024-1-NADS3/Projeto12/tree/main

1.2. Apresentar 2 testes de componentes

Teste Sistêmico para validação do CRUD da tabela Usuário do Banco de Dados

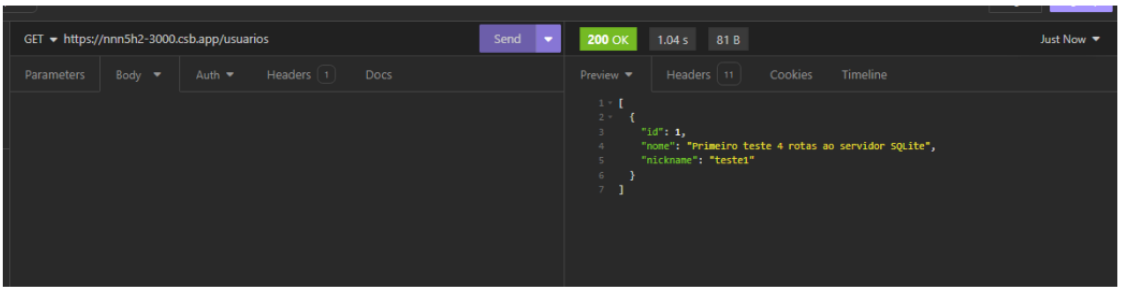
Teste de Cadastro do Usuário utilizando o método Post por meio de uma requisição HTTP via JSON

POST USUÁRIO



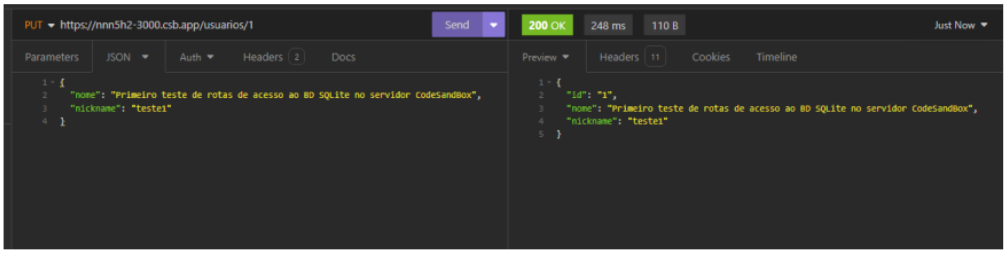
Teste de Consulta do Usuário utilizando o método Get por meio de uma requisição HTTP via JSON

GET USUÁRIO

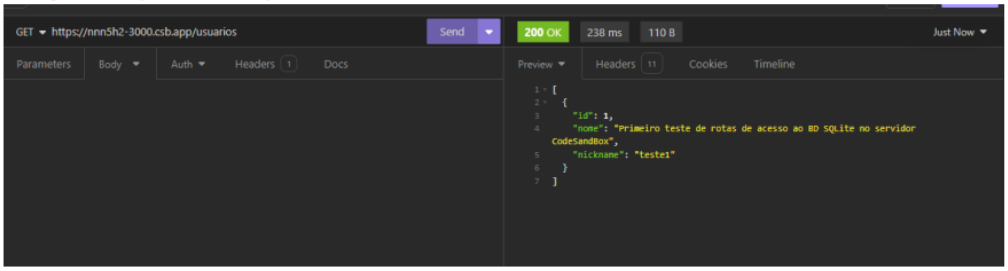


Teste de Atualização do Usuário utilizando o método Put por meio de uma requisição HTTP via JSON

PUT USUÁRIO

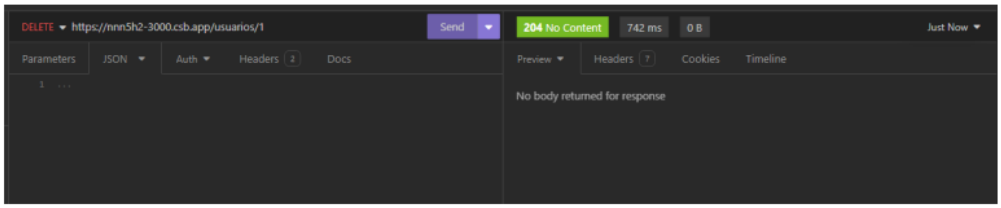


⇨ Get gerado após atualização:

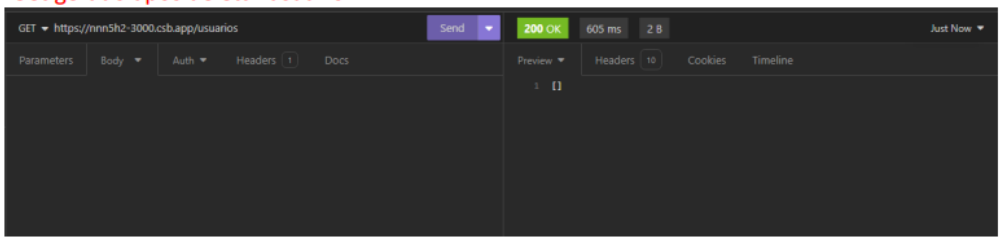


Teste de Exclusão do Usuário utilizando o método Delete por meio de uma requisição HTTP via JSON

DELETE USUÁRIO



⇨ Get gerado após deletar usuário:



2. Qualidade de Software

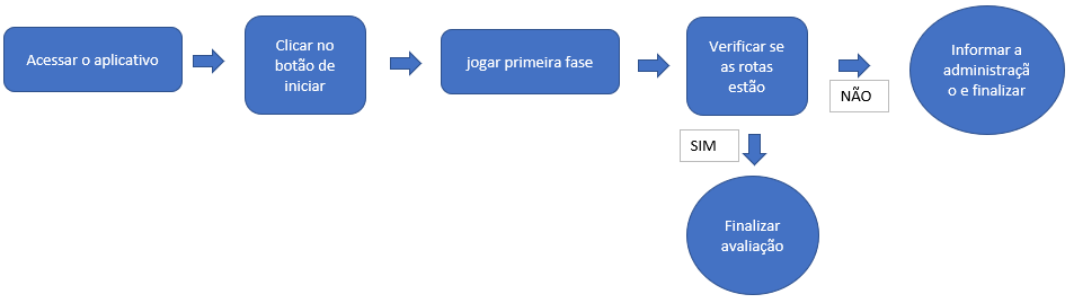
2.1. Indicar 4 atributos de qualidade de software e informar como foi aplicado no projeto integrador (PI)

Na nossa aplicação utilizamos conceitos de **Usabilidade** para a avaliação da interface do usuário em relação a princípios de design. Adicionalmente realizamos teste de **Confiabilidade e Desempenho** para identificar e corrigir possíveis problemas na aplicação rapidamente e melhorar a eficiência do software em termos de tempo de resposta. Além disso implementamos recursos de criptografia para melhorar o atributo de **Segurança** da nossa aplicação protegendo os dados e garantindo a integridade das informações.

2.2. Apresentar um Modelo que qualidade de software

Modelo de Qualidade de Software		
Atributos	**Objetivo**	**Métodos de Avaliação**
Usabilidade	Melhorar a experiência do usuário garantindo uma interface intuitiva e fácil de usar. Métodos de Avaliação:	Avaliação heurística com base em princípios de design reconhecidos.
		Testes de usabilidade com usuários reais para coletar feedback direto sobre a interface do usuário.
		- Análise de métricas como taxa de erro do usuário, eficiência na realização de tarefas e satisfação do usuário.
Confiabilidade	Assegurar que a aplicação é capaz de operar sem falhas durante um determinado período sob condições específicas.	Testes de carga para verificar como o sistema se comporta sob volumes elevados de uso.
		Testes de stress para determinar os limites da capacidade do sistema.
		Monitoramento contínuo da aplicação para identificação e correção rápida de falhas.
Usabilidade	Otimizar a eficiência do software, garantindo tempo de resposta rápido e uso eficiente dos recursos.	Monitoramento de performance em tempo real.
		Testes de benchmarking comparando a performance do software com os padrões da indústria.
		Otimização baseada nos resultados dos testes, ajustando configurações de software e hardware conforme necessário.
Usabilidade	Proteger os dados dos usuários e garantir a integridade e confidencialidade das informações.	Implementação de criptografia de dados em trânsito e em repouso.
		Realização de testes de penetração para identificar vulnerabilidades de segurança.
		Adoção de práticas de segurança de software como desenvolvimento orientado à segurança e revisões regulares de código

2.3. Apresentar um Processo (plano) de gerenciamento de qualidade de software



4. REFERÊNCIAS BIBLIOGRÁFICAS

SOMMERVILLE, I. **Engenharia de Software.** 11ª Edição. São Paulo: Pearson Addison-Wesley, 2017.