

Entrega 2

Grupo: Adriano de Souza, Carolina Arêas, Daniel Neves, Natália Bruno Rabelo, Vitor Hugo Prado

1. Descrição do escopo do sistema

1.1 Introdução

O Sistema de Gestão Clínica WeB foi projetado para fornecer uma solução abrangente e integrada para o gerenciamento eficaz das operações dentro de uma clínica médica.

1.2 Funções Principais

O sistema inclui uma série de módulos funcionais estrategicamente concebidos para abordar as diversas necessidades operacionais de uma clínica médica, tais como:

Agendamento de Consultas e Exames: Esta funcionalidade lida com a marcação eficiente de consultas e exames, permitindo uma gestão de tempo otimizada. O usuário também tem a possibilidade de editar e reagendar consultas.

Gestão de Usuários: Esta funcionalidade inclui o cadastro, a manutenção e o controle de login para vários tipos de usuários - médicos, administradores e pacientes.

Gestão de Especialidades Médicas: Esta funcionalidade fornece ferramentas para gerenciar e rastrear as várias especialidades médicas oferecidas na clínica.

Gestão de Planos de Saúde: Esta funcionalidade permite o controle eficiente dos diferentes planos de saúde disponíveis, assegurando um atendimento adequado aos pacientes.

1.3 Requisitos Não Funcionais

RNF1: Sistema deverá ser responsivo sendo obrigatório o uso do Bootstrap 5.0 (ou superior);

RNF2: Todas as bibliotecas, scripts, imagens etc., necessários para o funcionamento devem estar disponíveis localmente;

1.4 Requisitos Funcionais

1.4.1 Requisitos – Paciente

RF1: O paciente faz o registro na aplicação, informando nome, CPF, senha e tipo do plano de saúde.

RF2: O paciente autorizado acessa a área privada da aplicação (faz o login) e tem acesso ao seu painel de controle que possui: a lista das consultas (realizadas e não realizadas) e a ação de marcação de consulta.

RF3: O paciente autorizado marca a consulta escolhendo o médico (especialidade médica) e a data.

1.4.2 Requisitos - Médico

RF4: O médico autorizado acessa a área privada da aplicação (faz o login) e tem acesso ao seu painel de controle que possui: a lista das consultas (realizadas e não realizadas) e a ação de realização da consulta.

RF5: O médico realiza a consulta descrevendo o que ocorreu e solicitando exames quando necessário.

RF6: O médico visualiza e edita os dados de uma consulta já realizada por ele.

1.4.3 Requisitos - Administradores

RF7: O administrador acessa a área privada da aplicação (faz login) e tem acesso ao seu painel de controle que possui as seguintes ações: cadastrar médicos, cadastrar pacientes, cadastrar administradores, cadastrar os tipos de plano de saúde e cadastrar as especialidades médicas.

RF8: O administrador cadastra os médicos.

RF9: O administrador cadastra as especialidades.

RF10: O administrador cadastra os tipos de planos de saúde.

RF11: O administrador retira a autorização do paciente para acessar a aplicação.

RF12: O administrador retira a autorização do médico para acessar a aplicação.

RF13: O administrador visualiza as consultas de um médico.

2. Plano de testes

Os testes serão realizados em várias fases do desenvolvimento para garantir que o sistema atenda aos padrões de qualidade e às expectativas do usuário.

- **Testes unitários:** Esses testes verificarão a funcionalidade de cada componente individual do sistema.
- **Testes de integração:** Esses testes verificarão a interação adequada entre os diferentes componentes do sistema.
- **Testes de sistema:** Esses testes verificarão a funcionalidade do sistema como um todo, para garantir que todas as partes estejam funcionando corretamente juntas.
- **Análise estática:** Uma fase no processo de testes que se concentra em melhorar a qualidade do código, aumentando a manutenibilidade, reduzindo a complexidade e corrigindo pequenos erros que podem se tornar grandes problemas no futuro.
- **Testes de performance:** Esses testes serão realizados para garantir que o sistema é capaz de funcionar eficientemente sob carga pesada ou grande volume de dados.

2.1 Artefatos gerados

- Documentação de requisitos do sistema
- Código-fonte Java
- Casos de teste
- Relatório de inspeção de código fonte
- Relatório de teste de mutação
- Relatório de teste estrutural

2.2 Ferramentas utilizadas

- Java JDK para desenvolvimento
- Netbeans para desenvolvimento integrado de ambiente (IDE)
- JUnit e Mockito para testes unitários
- Postman para testes de integração
- SonarLint para análise estática
- Selenium para teste de sistema
- Jacoco para teste de cobertura (estrutural)
- Pitest para teste de mutação (baseado em defeitos)
- JMeter teste de carga

3. Código-fonte original

O código-fonte original do projeto encontra-se neste repositório abaixo:

<https://github.com/nataliaRabelo/ClinicaMaven>

4. Indicação das medidas de cada atributo de qualidade da ISO 25010 seguindo uma escala. Justificar as decisões para indicação das medidas.

A ISO 25010 estabelece vários atributos de qualidade para software, que foram levados em consideração na concepção deste sistema. A seguir estão as medidas de cada atributo de qualidade seguindo uma escala de 0 a 5, onde 5 atinge o nível mais alto, com justificativas para as decisões tomadas:

- **Adequação funcional:**

- Nota: 5
- Justificativa: O sistema atende aos requisitos necessários para a gestão de uma clínica médica, a decisão de incluir determinadas funções foi tomada com base nas necessidades dos usuários finais. Aprofundando essa questão, o sistema implementa de forma correta todas as funcionalidades planejadas e comportamentos esperados.

- **Eficiência do desempenho:**

- Nota: 5
- Justificativa: O sistema é otimizado para processamento rápido e eficiente, a eficaz utilização dos recursos também torna o sistema capaz de suportar a carga de trabalho em uma clínica médica em tempo real e suas demais necessidades.

- **Compatibilidade:**

- Nota: 3
- Justificativa: O sistema foi projetado para ser compatível com sistemas operacionais e bancos de dados comumente utilizados. A escolha do Java como linguagem de programação foi feita devido à sua portabilidade e compatibilidade, no entanto, a versão das ferramentas torna possível ou não o funcionamento do sistema (como exemplo: a versão do Mysql). Além disso, uma nota máxima nesse atributo exigiria garantir que não haja conflitos ou impactos negativos em suas operações individuais, apesar de baixa probabilidade, ainda é possível esse ocorrido.

- **Usabilidade:**

- Nota: 3
- Justificativa: O sistema tem uma interface de usuário intuitiva, tornando-o fácil de usar mesmo para pessoas sem experiência técnica. Porém, a cobertura de proteção contra erros dos usuários é baixa, ilustrando esse fato, apesar de haver validações (exemplo: validação do CPF), são realizadas em poucas funcionalidades, também possui poucas formas de avisos e questionamentos ao usuário, embora na maioria dos casos é avisado quando uma ação teve sucesso, não há questionamentos de confirmação de ações. Além disso possui baixa acessibilidade, como exemplo, não possui funcionalidades para auxiliar durante o uso feito por usuários com deficiência visual,

- **Confiabilidade:**

- Nota: 4
- Justificativa: O sistema possui um backup completo e bem documentado, assim a recuperabilidade do sistema se torna fácil. Porém, apesar de ser um sistema completo, ele foi desenvolvido em um curto período de tempo e depois não sofreu mais análises ou mudanças, e assim não passou por um ciclo de aprendizado e melhoria contínua, tendo como consequência uma falta de maturidade,

- **Segurança:**

- Nota: 2
- Justificativa: O sistema foi projetado para ser robusto e confiável, com redundância de dados e recuperação de falhas para garantir a integridade dos dados. Além disso, não realiza nenhuma forma de divulgação ou uso não autorizado. Porém, apesar de possuir senhas para garantir sua autenticidade, tais senhas apresentam falhas, como exemplo não há exigência de uma senha forte,

a senha pode por exemplo ser “1”, além disso são mostrada de formas explícitas no banco de dados, assim o sistema não consegue assegurar uma das principais formas de segurança.

- **Manutenibilidade:**

- Nota: 5
- Justificativa: O código é estruturado de forma clara, tornando-o fácil de manter, atualizar e testar. Além disso sua transparência, estrutura objetiva e boa documentação torna simples sua análise é possível a reutilização de código, funcionalidades ou recursos.

- **Portabilidade:**

- Nota: 3
- Justificativa: O sistema foi projetado para ser facilmente portado para diferentes sistemas operacionais e navegadores, é capaz de realizar a instalação sem erros ou problemas. Também no caso de um componente, recurso etc ser substituído, o sistema tem a capacidade de suportar essa mudança sem causar impactos significativos. Porém ele não é adaptável a uma versão mobile ou no caso do uso de um tablet, nesses casos a página não é ajustada tornando o sistema incapaz de ser usado, exemplo: alguns botões não aparecem na página pois ela não é ajustada para o tamanho ideal.

As decisões tomadas ao determinar as medidas para cada atributo de qualidade foram baseadas na necessidade de fornecer um sistema de alta qualidade que atenda às necessidades dos usuários e cumpra os padrões de segurança e ética no setor médico.

5. Relatório de inspeção do código-fonte e ou documentação do software.

5.1. Objetivos:

Realizar uma inspeção de código com software adequado para o projeto para analisar a qualidade do código e consequentemente detectar bugs, vulnerabilidades e code smells.

5.2. Metodologia:

Utilizamos o plugin SonarLint para o NetBeans em cada pasta da aplicação para descobrirmos todos os problemas do código.

5.3 Resultados:

Foram encontrados ao todo 363 erros que foram reduzidos a três levando em consideração que 34 restantes são falsos positivos em que acusavam variáveis não utilizadas que, na verdade, estavam sendo utilizadas.

Os três erros não corrigidos foram acordados em não serem resolvidos, os quais dois deles tratam-se de complexidade alta, por motivos de estarmos atuando em um sistema com tecnologia antiga que nos limita ao uso de servlets, onde dois dos erros apontavam alta complexidade em uma classe de servlet, enquanto o último erro restante que consiste em aumentar segurança das senhas, a motivação se deu por não termos interesse em aumentar a segurança das senhas por motivos de pretensão de realizar mais teste.

Abaixo segue a apresentação mais detalhada dos problemas encontrados durante a inspeção de código com auxílio da ferramenta.

5.4.1 Problema 1:

- **Descrição:** O erro “AvoidCommentOutCodeCheck” é caracterizado pela presença de comentários em trechos de código que não deveriam estar comentados.

- **Localização:** O erro se localiza no arquivo AreaAdministrador.jsp.



```
<br><br>

<h3>Olá, Bem vindo(a), Administrador <%= adm.getNome() %>!/</h3>

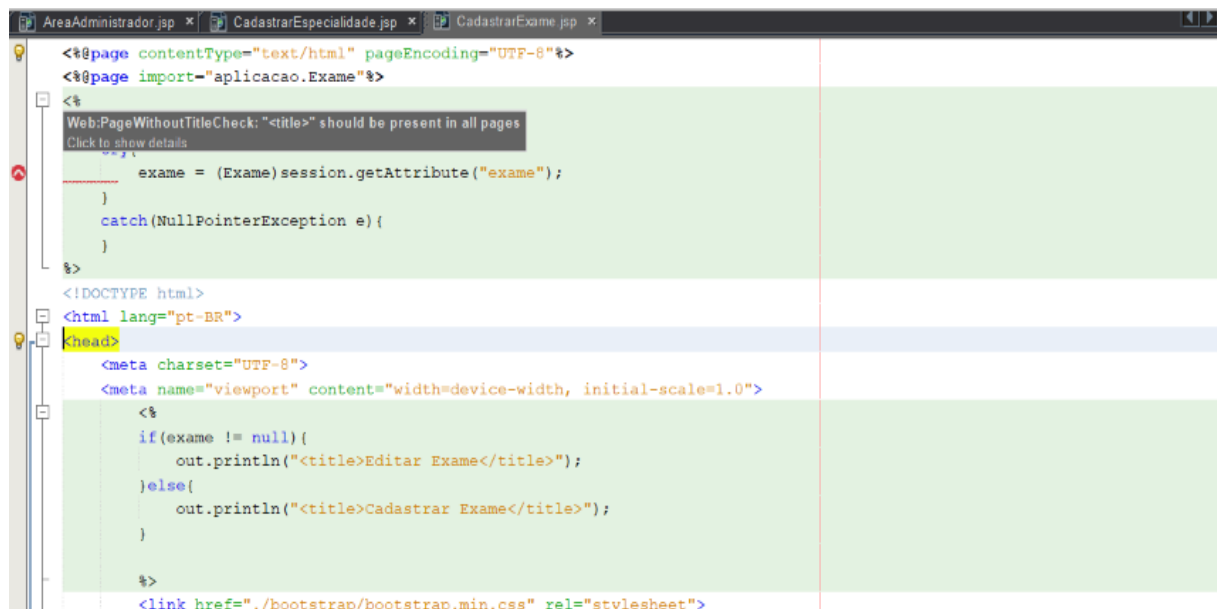
<br><br>

<div style="text-align: center;">
  <div>
    <!--<div class="intern" style="border: 15px solid red; display: inline-block;">-->
    <!--<div class="intern" style="border: 15px solid green; width: 30%;">-->
    <form action="./ControladorAdministrador" method="GET">
      <button class="button button2" type="submit">Ver Pacientes</button>
      <input type="hidden" name="arg" value="Visualizar">
      <input type="hidden" name="type" value="Paciente">
    </form>
    <br>
    <form action="./ControladorAdministrador" method="GET">
      <button class="button button2" type="submit">Ver Médicos</button>
      <input type="hidden" name="arg" value="Visualizar">
      <input type="hidden" name="type" value="Medico">
    </form>
    <br>
    <form action="./ControladorAdministrador" method="GET">
```

- **Prioridade:** Major (Code Smell).
- **Recomendação:** Remoção dos comentários desnecessários.

5.4.2 Problema 2:

- **Descrição:** O erro “PageWithoutTitleCheck” descreve que todas as páginas Web deveriam ter um título.
- **Localização:** Há ocorrência em 3 arquivos diferentes (CadastrarEspecialidade.jsp, CadastrarExame.jsp e CadastrarPlano.jsp). Abaixo o exemplo é demonstrado na página CadastrarExame.jsp.



```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="aplicacao.Exame"%>
<%
Web.PageWithoutTitleCheck: "<title>" should be present in all pages
Click to show details
  exame = (Exame)session.getAttribute("exame");
}
catch (NullPointerException e){
}
%>

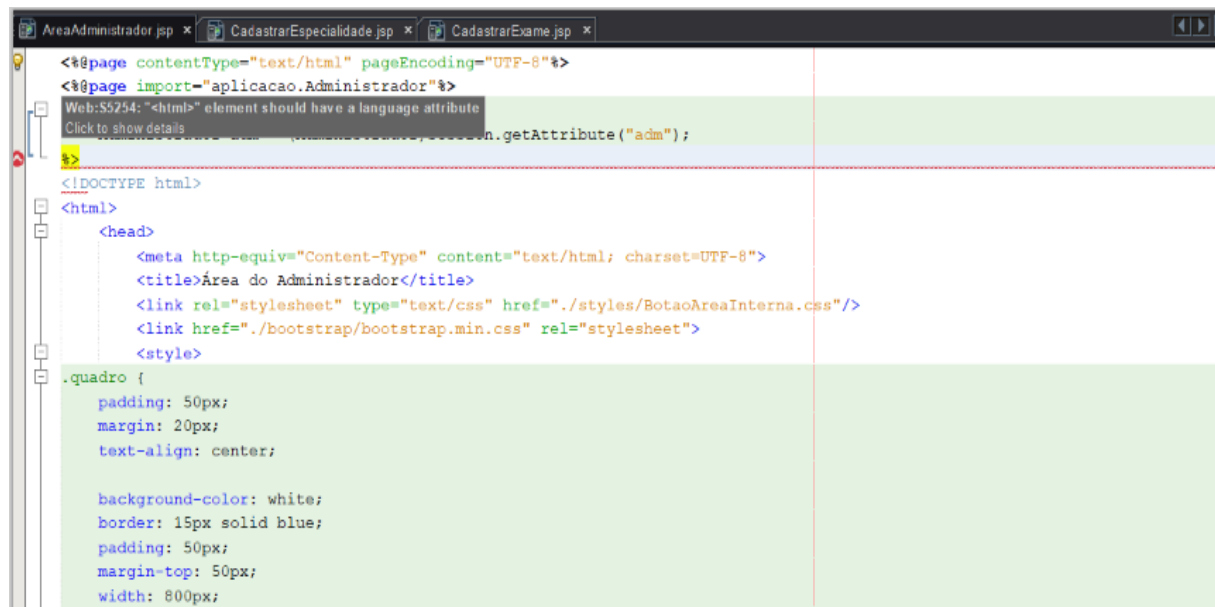
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <%
    if (exame != null){
      out.println("<title>Editar Exame</title>");
    }else{
      out.println("<title>Cadastrar Exame</title>");
    }
  %>
</head>
<link href="./bootstrap/bootstrap.min.css" rel="stylesheet">
```

- **Prioridade:** Major (Bug).
- **Recomendação:** O SonarLint não detectou a condição para mudar o texto exibido como título e acusou como erro.

5.4.3 Problema 3:

- **Descrição:** O erro “S5254” destaca a falta da atribuição de uma idioma no elemento <html>.

- **Localização:** Esse erro teve 15 ocorrências diferentes no código fonte da aplicação. O arquivo AreaAdministrador.jsp é um exemplo.



```
<@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="aplicacao.Administrador"%>
Web:55254: "<html>" element should have a language attribute
Click to show details
<!-- ...n.getAttribute("adm"); ... -->
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Área do Administrador</title>
    <link rel="stylesheet" type="text/css" href="/styles/BotaoAreaInterna.css"/>
    <link href="/bootstrap/bootstrap.min.css" rel="stylesheet">
    <style>
      .quadro {
        padding: 50px;
        margin: 20px;
        text-align: center;

        background-color: white;
        border: 15px solid blue;
        padding: 50px;
        margin-top: 50px;
        width: 800px;
      }
    </style>
```

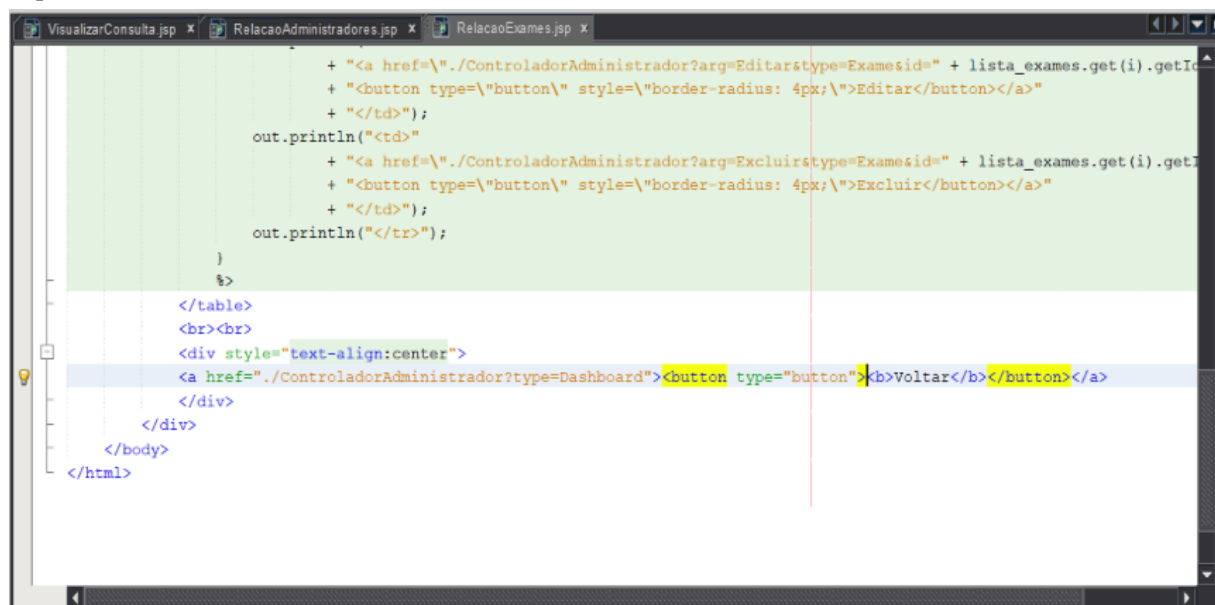
- **Prioridade:** Major (Bug).

- **Recomendação:** Adicionar o parâmetro lang="idioma escolhido" dentro da tag <html>.

5.4.4 Problema 4:

- **Descrição:** O erro "BoldAndItalicTagsCheck" indica casos onde foram usadas as tags ou <i>.

- **Localização:** Existem 27 ocorrências desse erro. Abaixo é mostrado um exemplo no trecho de código no arquivo JSP RelacaoExame.



```
+ "<a href='\"./ControladorAdministrador?arg=Editar&type=Examesid=\" + lista_exames.get(i).getId() + \">Editar</a>\"";
+ "</td>\"";
out.println("<td>\"
+ "<a href='\"./ControladorAdministrador?arg=Excluir&type=Examesid=\" + lista_exames.get(i).getId() + \">Excluir</a>\"";
+ "</td>\"";
out.println("</tr>\"");
}
}
</table>
<br><br>
<div style=\"text-align:center\">
  <a href='\"./ControladorAdministrador?type=Dashboard\"><button type=\"button\"><b>Voltar</b></button></a>
</div>
</div>
</body>
</html>
```

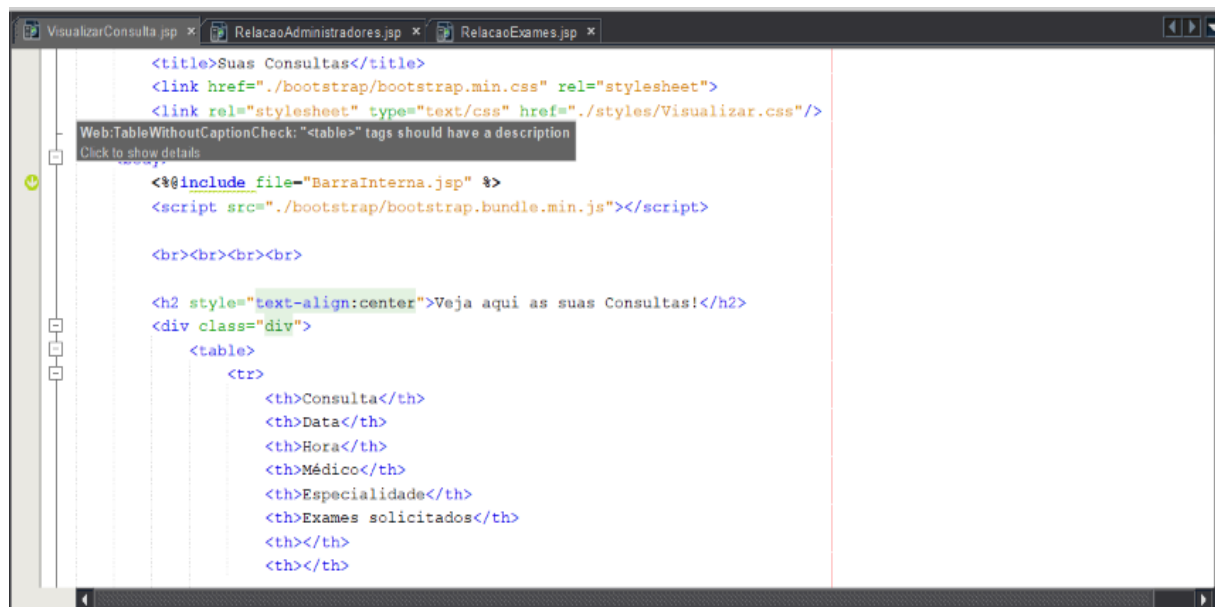
- **Prioridade:** Minor (Bug).

- **Recomendação:** Substituir as tags e <i> por e no código por recomendação.

5.4.5 Problema 5:

- **Descrição:** O erro "TableWithoutCaptionCheck" acusa a falta de legendas para descrever o conteúdo da tabela para possíveis usuários com deficiência visual.

- **Localização:** O erro apresentou 9 ocorrências diferentes e uma delas acontece no arquivo VisualizacaoConsulta.jsp.



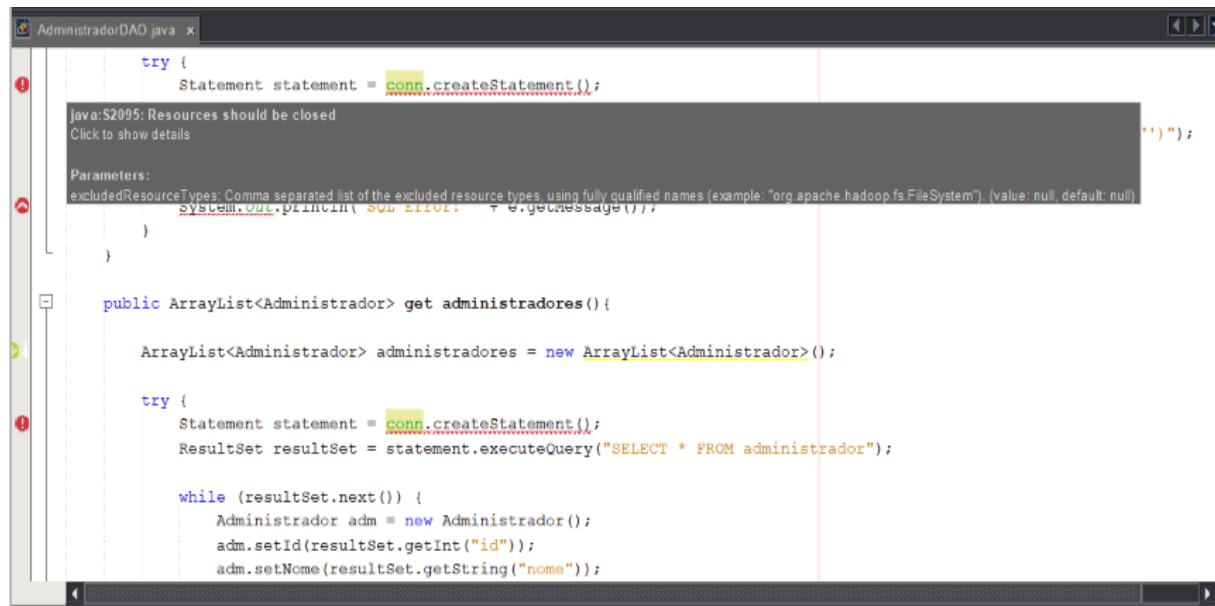
- **Prioridade:** Minor (Bug).

- **Recomendação:** Adicionar a tag <caption> dentro da tag <table> para que haja descrição do conteúdo presente.

5.4.6 Problema 6:

- **Descrição:** O erro “S2095” indica que recursos foram abertos e não foram fechados posteriormente. Isso pode resultar no vazamento de recursos.

- **Localização:** Há 53 eventos detectados pelo SonarLint. Uma das incidências acontece na classe AdministradorDAO.java.



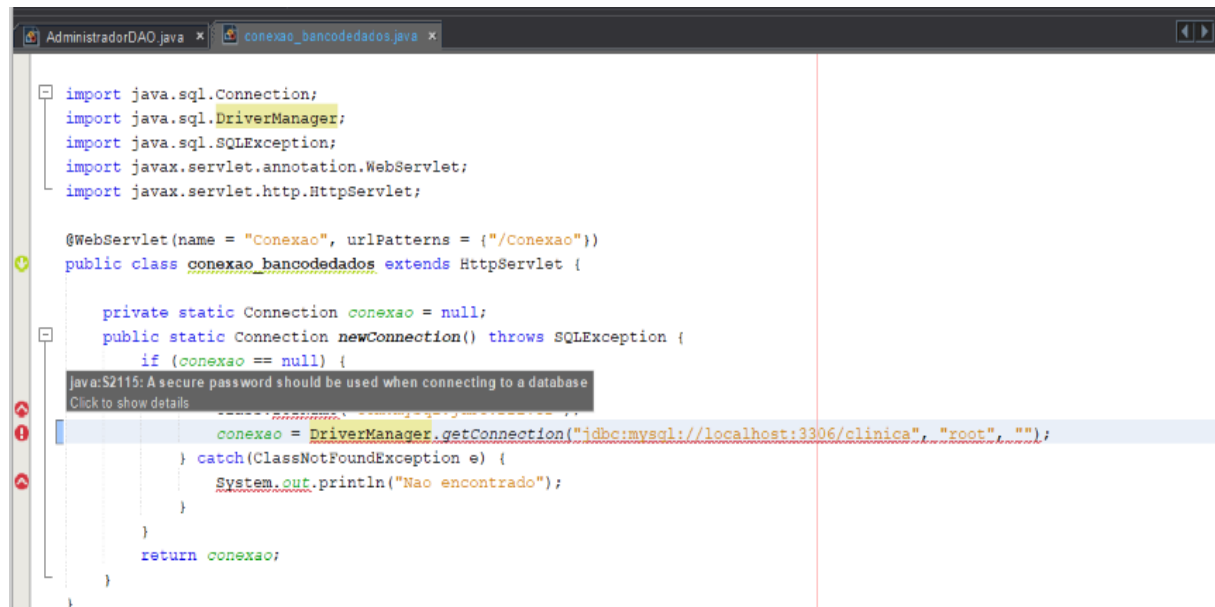
- **Prioridade:** Blocker (Bug).

- **Recomendação:** Garantir que seja feito o fechamento de todos os recursos que implementam a classe *Closeable* ou a sua super interface *AutoCloseable*.

5.4.7 Problema 7:

- **Descrição:** O erro “S2115” aponta que a senha usada para se conectar ao banco de dados é insegura.

- **Localização:** O erro está localizado na classe `conexao_bancodedados.java` que realiza a conexão com o banco de dados.



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

@WebServlet(name = "Conexao", urlPatterns = {"/Conexao"})
public class conexao_bancodedados extends HttpServlet {

    private static Connection conexao = null;

    public static Connection newConnection() throws SQLException {
        if (conexao == null) {
            conexao = DriverManager.getConnection("jdbc:mysql://localhost:3306/clinica", "root", "");
        } catch (ClassNotFoundException e) {
            System.out.println("Nao encontrado");
        }
        return conexao;
    }
}
```

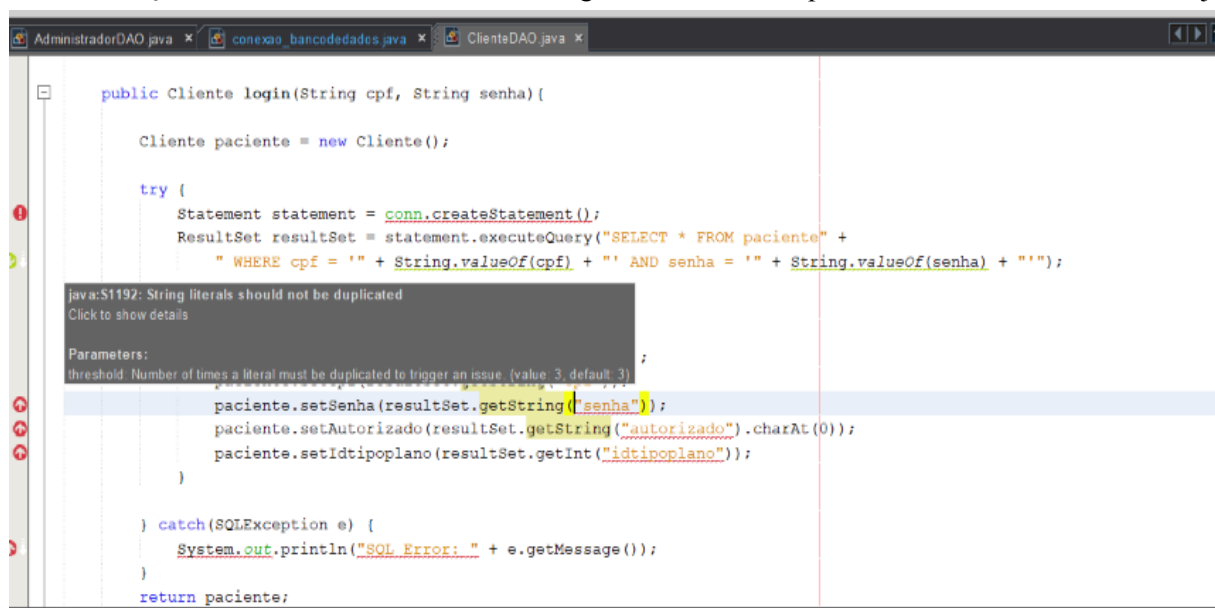
- **Prioridade:** Blocker (Vulnerability).

- **Recomendação:** Utilizar uma senha mais forte, que seja longa e contenha caracteres maiúsculos, minúsculos e caracteres especiais.

5.4.8 Problema 8:

- **Descrição:** O erro “S1192” demonstra a presença literais String duplicados, algo que pode deixar o processo de refatoração propício a erro.

- **Localização:** Existem 33 ocorrências no código-fonte. Um exemplo ocorre na classe `ClienteDAO.java`.



```
public Cliente login(String cpf, String senha){

    Cliente paciente = new Cliente();

    try {
        Statement statement = conn.createStatement();
        ResultSet resultSet = statement.executeQuery("SELECT * FROM paciente" +
            " WHERE cpf = '" + String.valueOf(cpf) + "' AND senha = '" + String.valueOf(senha) + "'");

        paciente.setSenha(resultSet.getString("senha"));
        paciente.setAutorizado(resultSet.getString("autorizado").charAt(0));
        paciente.setIdtipoplano(resultSet.getInt("idtipoplano"));

    } catch (SQLException e) {
        System.out.println("SQL_Error: " + e.getMessage());
    }

    return paciente;
}
```

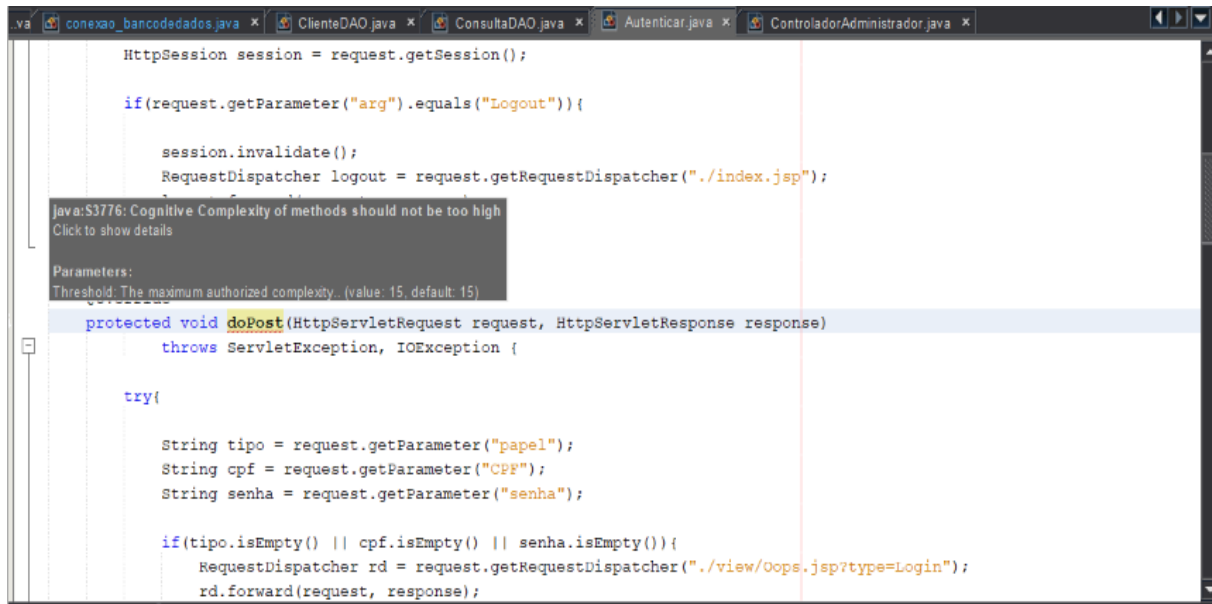
- **Prioridade:** Critical (Code Smell).

- **Recomendação:** Transformar as variáveis em constantes para evitar a probabilidade de erro e permitir que as mesmas sejam referenciadas de qualquer lugar.

5.4.9 Problema 9:

- **Descrição:** O erro “S3776” elucida a existência de métodos de alta complexidade cognitiva e esses métodos são difíceis de manter.

- **Localização:** Esse erro aparece nas classes Autenticar.java e ControladorAdministrador.java.



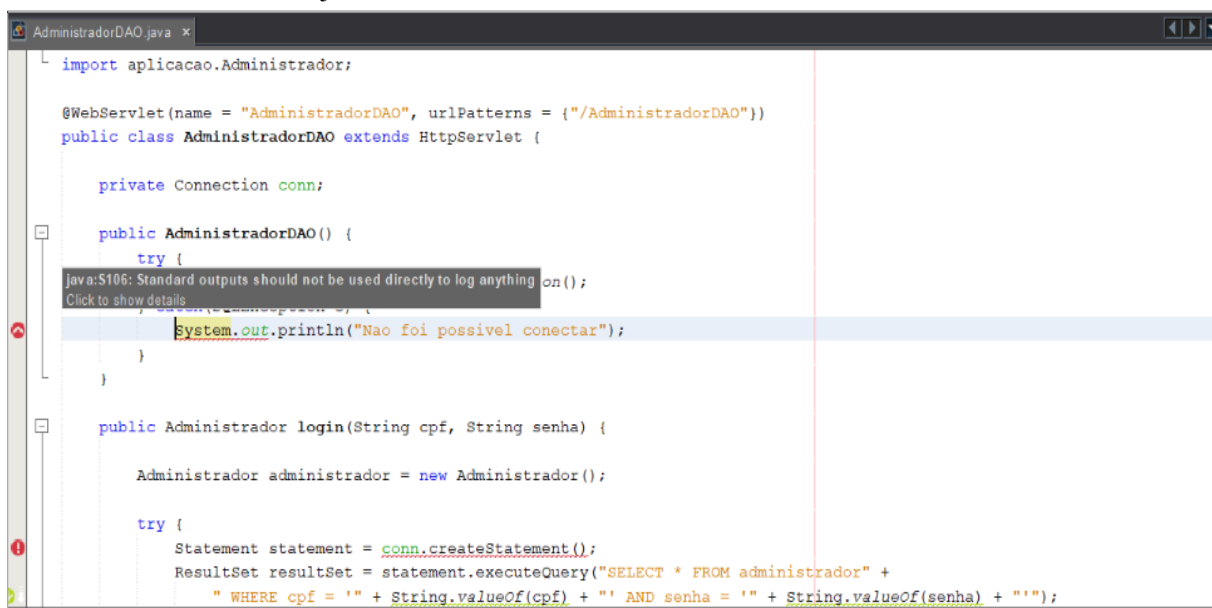
- **Prioridade:** Critical (Code Smell).

- **Recomendação:** Diminuir a complexidade dos métodos criados. Embora a grande quantidade de argumentos do método doPost seja o padrão oferecido pela própria linguagem.

5.4.10 Problema 10:

- **Descrição:** O erro “S106” descreve o problema de exibir mensagens de log em outputs comuns (system.out.println).

- **Localização:** Existem 63 ocorrências desse erro na totalidade do código. Abaixo é mostrado um exemplo na classe AdministradorDAO.java.



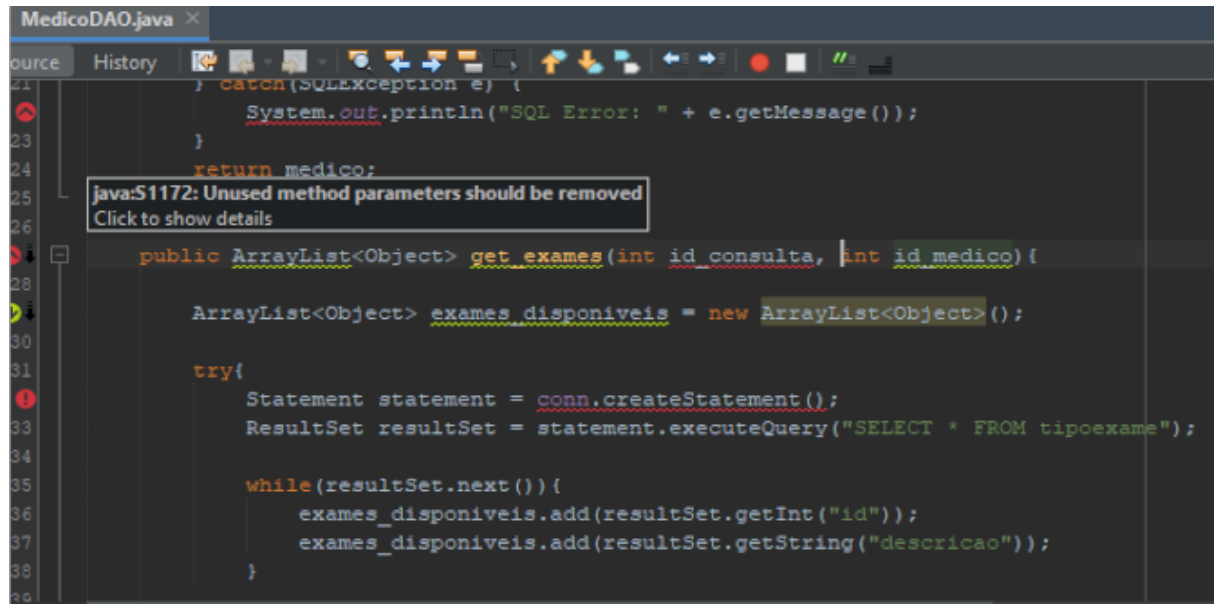
- **Prioridade:** Major (Code Smell).

- **Recomendação:** A solução é salvar os logs para o usuário poder consultá-los posteriormente e usar um formato uniforme para padronização.

5.4.11 Problema 11:

- **Descrição:** O erro “S1172” mostra a ocorrência de inserção de parâmetros que não são usados.

- **Localização:** Esse erro aparece na classe MedicoDAO.java.



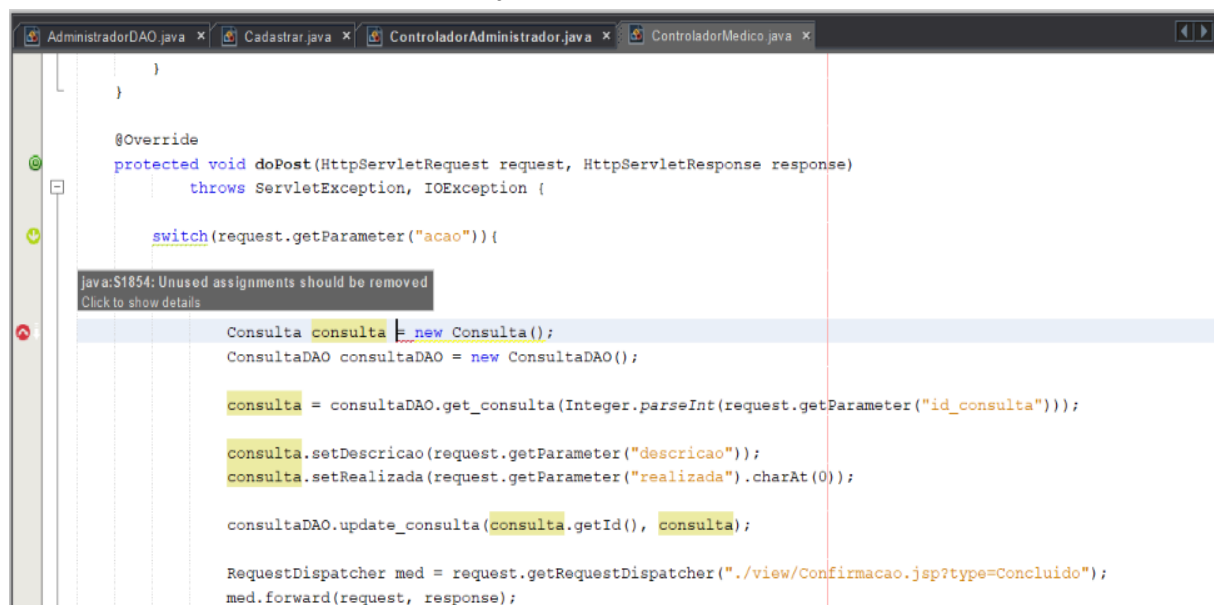
- **Prioridade:** Major (Code Smell).

- **Recomendação:** Remover todos os parâmetros que não estão sendo usados para melhorar a legibilidade do código.

5.4.12 Problema 12:

- **Descrição:** O erro “S1854” indica a ocorrência de atribuições feitas que não são usadas.

- **Localização:** 33 aparições desse erro foram capturadas pelo SonarLint. A imagem mostra uma das ocorrências na classe ControladorMedico.java.



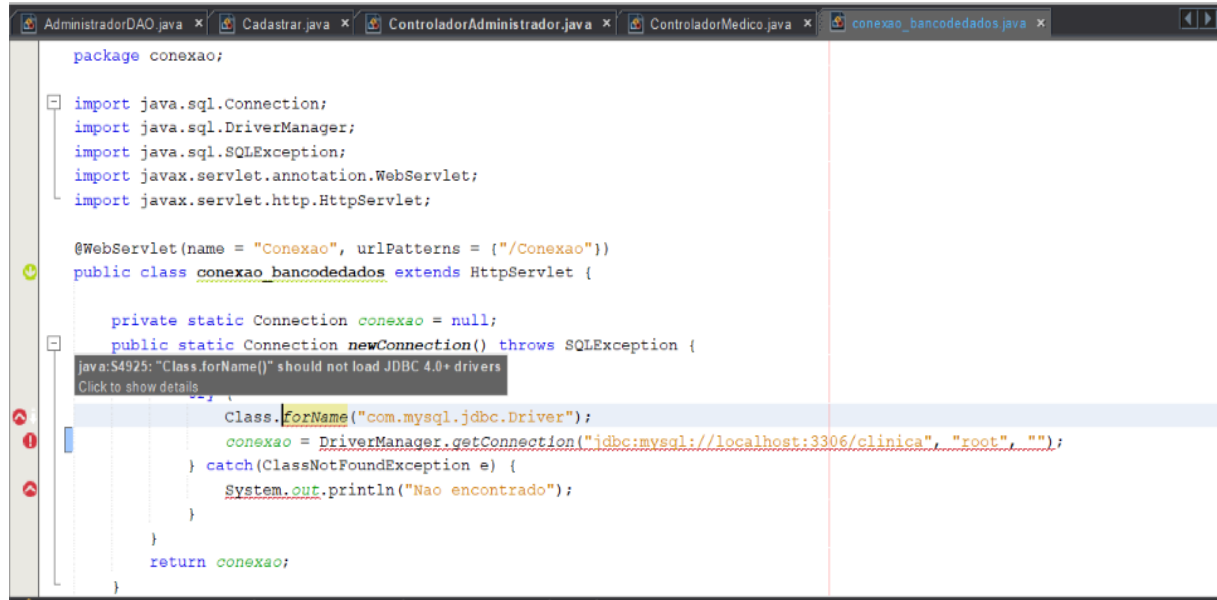
- **Prioridade:** Major (Code Smell).

- **Recomendação:** Mesmo que não seja um erro propriamente dito, pode ser considerado um desperdício de recursos. Portanto, todos os valores calculados devem ser usados ou removidos se não forem ter uso factível.

5.4.13 Problema 13:

- **Descrição:** O erro “S4925” aponta que drivers JDBC 4.0+ não precisam ser carregados pela “Class.forName()”.

- **Localização:** Esse erro aparece na classe de conexão com o banco de dados (conexao_bancodedados.java).



```
package conexao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

@WebServlet(name = "Conexao", urlPatterns = {"/Conexao"})
public class conexao_bancodedados extends HttpServlet {

    private static Connection conexao = null;

    public static Connection newConnection() throws SQLException {
        java:S4925: "Class.forName()" should not load JDBC 4.0+ drivers
        Class.forName("com.mysql.jdbc.Driver");
        conexao = DriverManager.getConnection("jdbc:mysql://localhost:3306/clinica", "root", "");
    } catch (ClassNotFoundException e) {
        System.out.println("Nao encontrado");
    }

    return conexao;
}
```

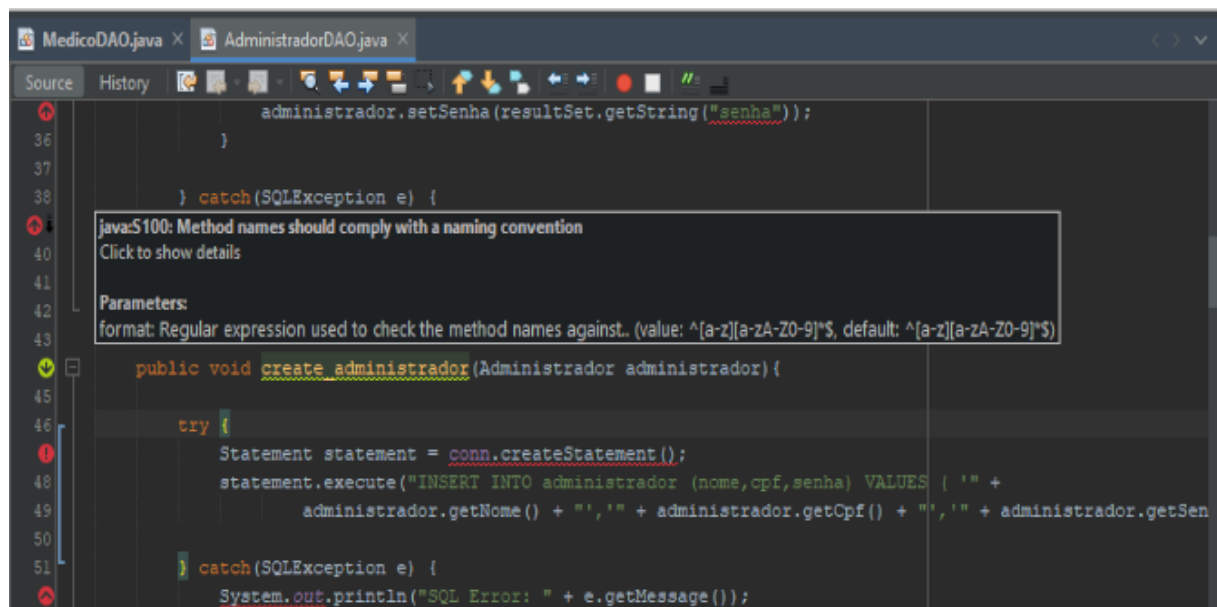
- **Prioridade:** Major (Code Smell).

- **Recomendação:** Não há necessidade de carregar o driver, pois nas versões 4.0 ou acima do driver JDBC isso é feito automaticamente.

5.4.14 Problema 14:

- **Descrição:** O erro “S100” exhibe o uso de nomes de métodos que não seguem a convenção de nomenclatura.

- **Localização:** O erro possui 49 ocorrências e o exemplo abaixo ocorre na classe AdministradorDAO.java.



```
administrador.setSenha(resultSet.getString("senha"));
}
} catch (SQLException e) {
    java:S100: Method names should comply with a naming convention
    Click to show details
    Parameters:
    format: Regular expression used to check the method names against.. (value: ^([a-z][a-zA-Z0-9]*)$, default: ^([a-z][a-zA-Z0-9]*)$)
    public void create_administrador(Administrador administrador) {
        try {
            Statement statement = conn.createStatement();
            statement.execute("INSERT INTO administrador (nome,cpf,senha) VALUES ( '" +
                administrador.getNome() + "','" + administrador.getCpf() + "','" + administrador.getSen
            catch (SQLException e) {
                System.out.println("SQL Error: " + e.getMessage());
            }
        }
    }
}
```

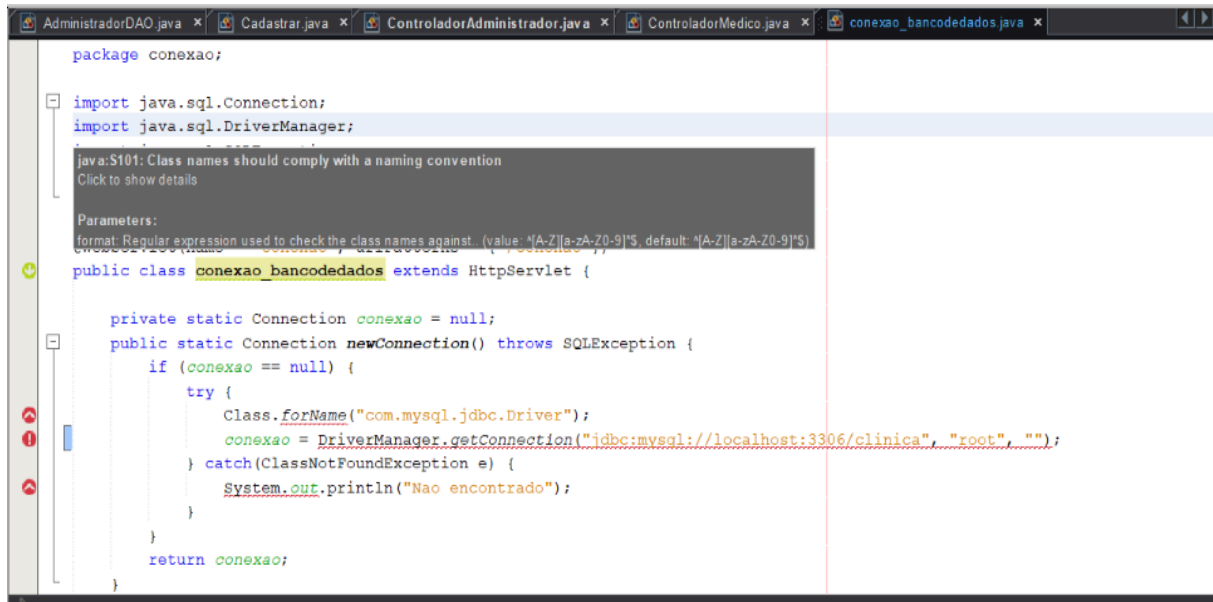
- **Prioridade:** Major (Code Smell).

- **Recomendação:** Adotar um nome que seja complacente com o padrão, como createAdministrador().

5.4.15 Problema 15:

- **Descrição:** O erro “S101” acusa o uso de um nome para a classe que não segue a convenção de nomenclatura.

- **Localização:** Assim como o erro anterior, a classe conexão_bancodedados.java que apresenta o problema.



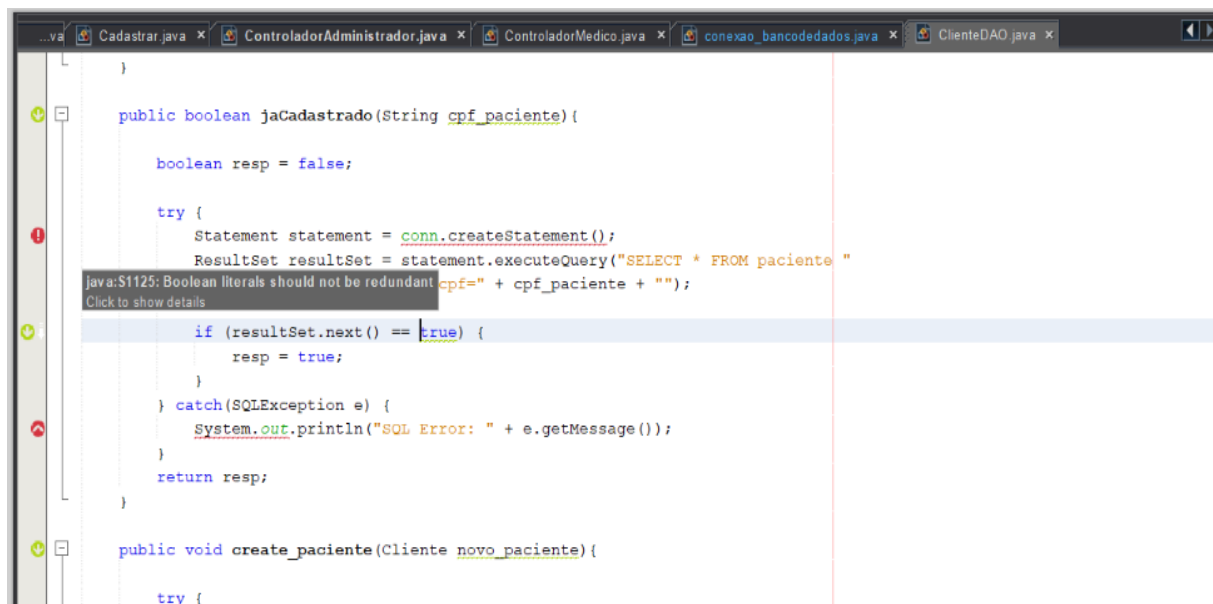
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Adotar um nome que seja complacente com o padrão, como ConexaoBancoDeDados.java.

5.4.16 Problema 16:

- **Descrição:** O erro “S1125” ilustra o uso de uma comparação booleana redundante.

- **Localização:** A questão ocorre na classe ClienteDAO.java.



- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Alterar a condição para apenas `if (resultSet.next())` para acabar com a redundância.

5.4.17 Problema 17:

- **Descrição:** O erro “S1153” relata o problema em concatenar múltiplas Strings. Por se tratar de um objeto imutável, é necessário a criação de um objeto intermediário para que seja feito a junção (append) e depois a conversão novamente para String. A performance diminui à medida que o tamanho da String aumenta.

- **Localização:** Existem 6 acontecimentos desse erro. O exemplo abaixo exhibe o mesmo na classe MedicoDAO.java.



```
System.out.println("Nao foi possivel conectar");

}

}

public Medico login(String cpf, String senha) {

    Medico medico = new Medico();

    try {
        java:S1153: String.valueOf() should not be appended to a String Statement();
        Click to show details
        Statement stmt = stmt.executeQuery("SELECT * FROM medico" +
            " WHERE cpf = '" + String.valueOf(cpf) + "' AND senha = '" + String.valueOf(senha) + "'");

        if (resultSet.next()) {
            medico.setId(resultSet.getInt("id"));
            medico.setNome(resultSet.getString("nome"));
            medico.setCrm(resultSet.getInt("crm"));
            medico.setEstadocrm(resultSet.getString("estadocrm"));
            medico.setCpf(resultSet.getString("cpf"));
            medico.setSenha(resultSet.getString("senha"));
            medico.setAutorizado(resultSet.getString("autorizado").charAt(0));
            medico.setIdespecialidade(resultSet.getInt("idespecialidade"));
        }
    }
}
```

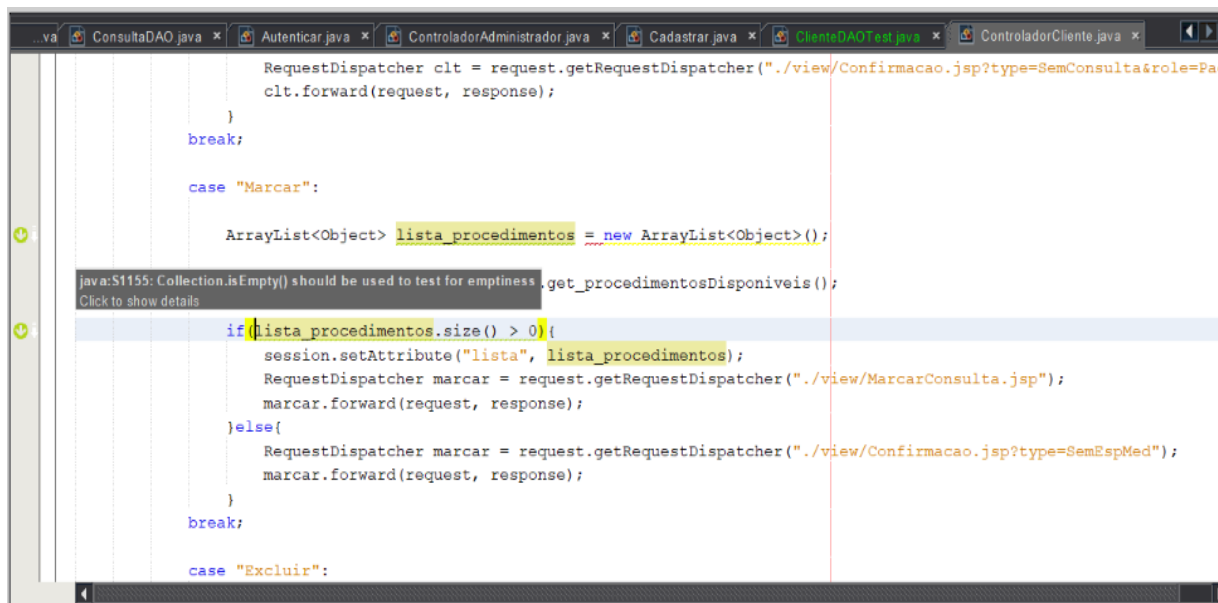
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Usar a classe StringBuilder para fazer a manipulação e concatenação dessas Strings, já que a classe faz o uso de Strings dinâmicas.

5.4.18 Problema 18:

- **Descrição:** O erro “S1155” indica que a verificação se o array está vazio foi feita de forma não otimizada.

- **Localização:** A localização do erro é a classe ControladorCliente.java.



```
RequestDispatcher clt = request.getRequestDispatcher("./view/Confirmacao.jsp?type=SemConsulta&role=Pa");
clt.forward(request, response);

}

break;

case "Marcar":

    ArrayList<Object> lista_procedimentos = new ArrayList<Object>();

    java:S1155: Collection.isEmpty() should be used to test for emptiness;
    Click to show details
    if (lista_procedimentos.size() > 0) {
        session.setAttribute("lista", lista_procedimentos);
        RequestDispatcher marcar = request.getRequestDispatcher("./view/MarcarConsulta.jsp");
        marcar.forward(request, response);
    } else {
        RequestDispatcher marcar = request.getRequestDispatcher("./view/Confirmacao.jsp?type=SemEspMed");
        marcar.forward(request, response);
    }

    break;

case "Excluir":
```

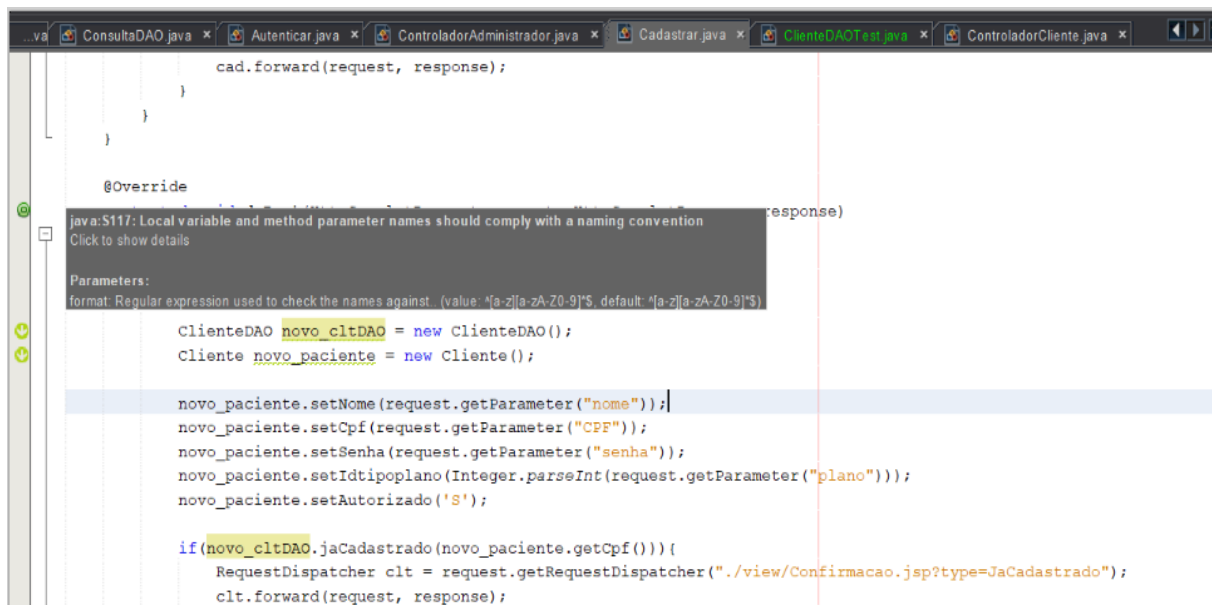
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Substituir a condição que testa o tamanho do array usando o método isEmpty(). Exemplo: lista_procedimentos.isEmpty().

5.4.19 Problema 19:

- **Descrição:** O erro “S117” acusa o uso de nomes para as variáveis e parâmetros de métodos que não seguem a convenção de nomenclatura.

- **Localização:** Esse problema apresenta 133 aparições no código-fonte. Abaixo é mostrado um caso na classe Cadastrar.java.



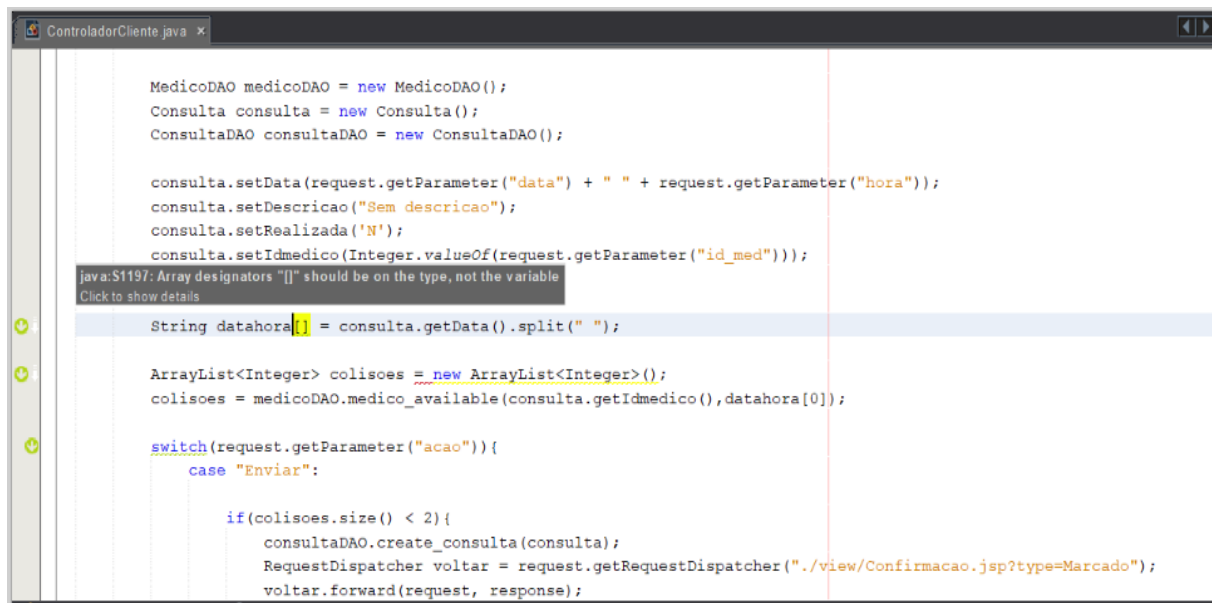
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Adotar um nome que seja complacente com o padrão, como novoCltdao.

5.4.20 Problema 20:

- **Descrição:** O erro “S1197” demonstra que o indicador da declaração de um array “[]” não está no local apropriado.

- **Localização:** O erro acontece na classe ControladorCliente.java.



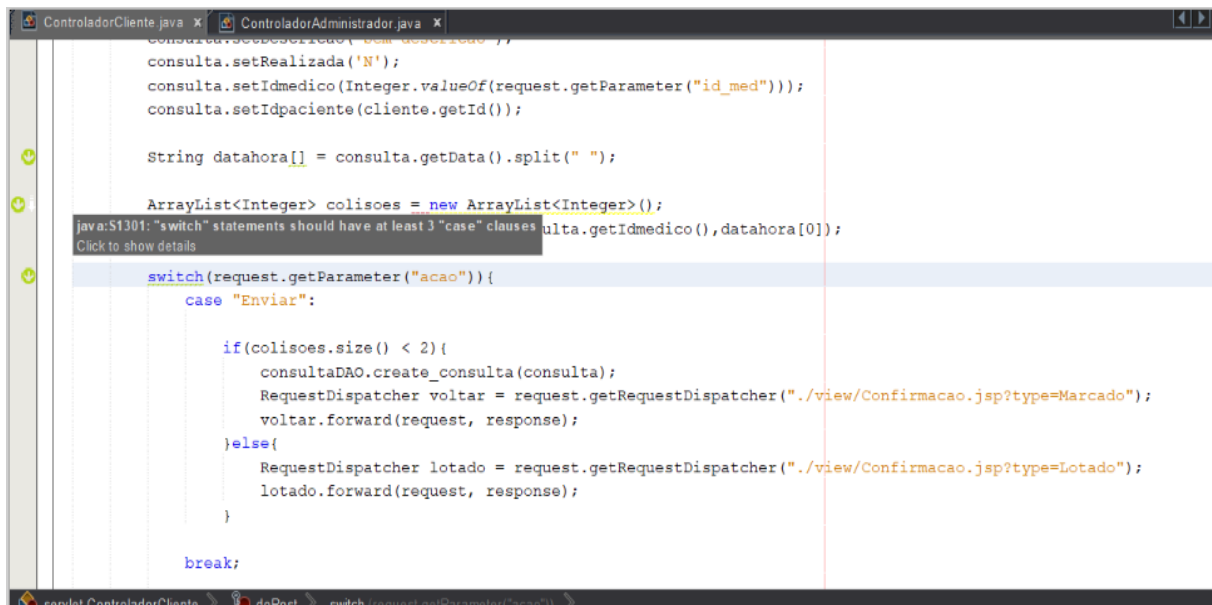
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Declarar o designador “[]” de array junto com o tipo e não com a variável para melhorar a legibilidade.

5.4.21 Problema 21:

- **Descrição:** O erro “S1301” indica o uso não apropriado da estrutura “switch”.

- **Localização:** 8 ocorrências aparecem e estão distribuídas entre as classes `ControladorCiente.java` e `ControladorAdministrador.java`.



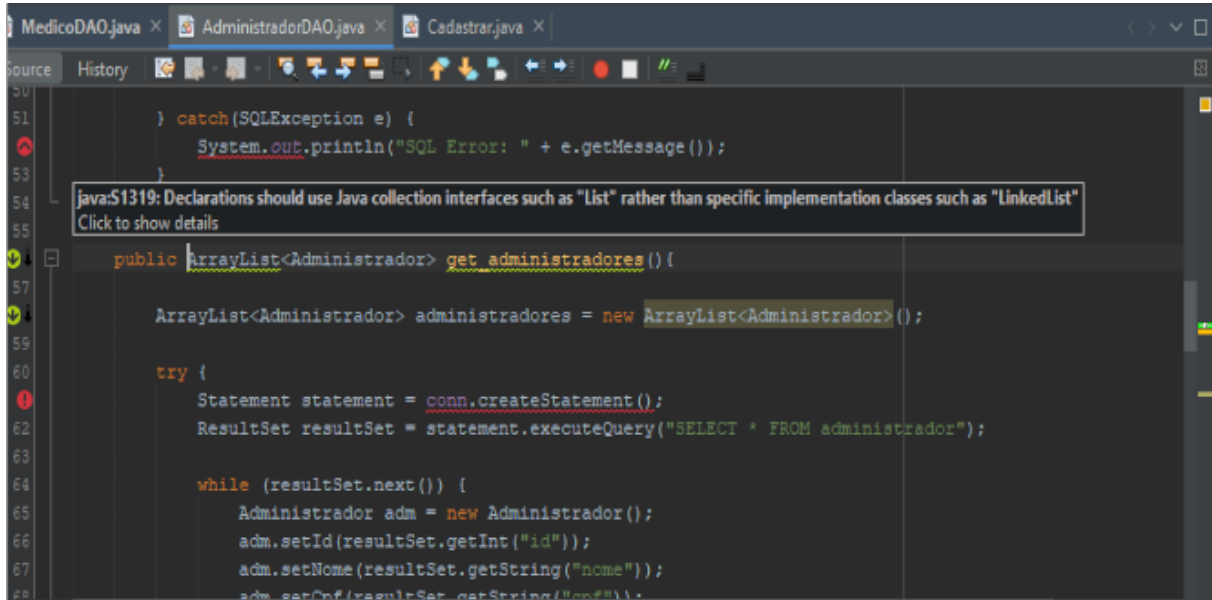
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Para facilitar a leitura do código, é indicado o uso da estrutura “if” se não existem mais de 3 casos condicionais.

5.4.22 Problema 22:

- **Descrição:** O erro “S1319” acusa a declaração de implementações específicas de lista, como o `ArrayList<Administrador>`.

- **Localização:** Existem 20 ocorrências desse erro. Abaixo o mesmo é ilustrado na classe `AdministradorDAO.java`.



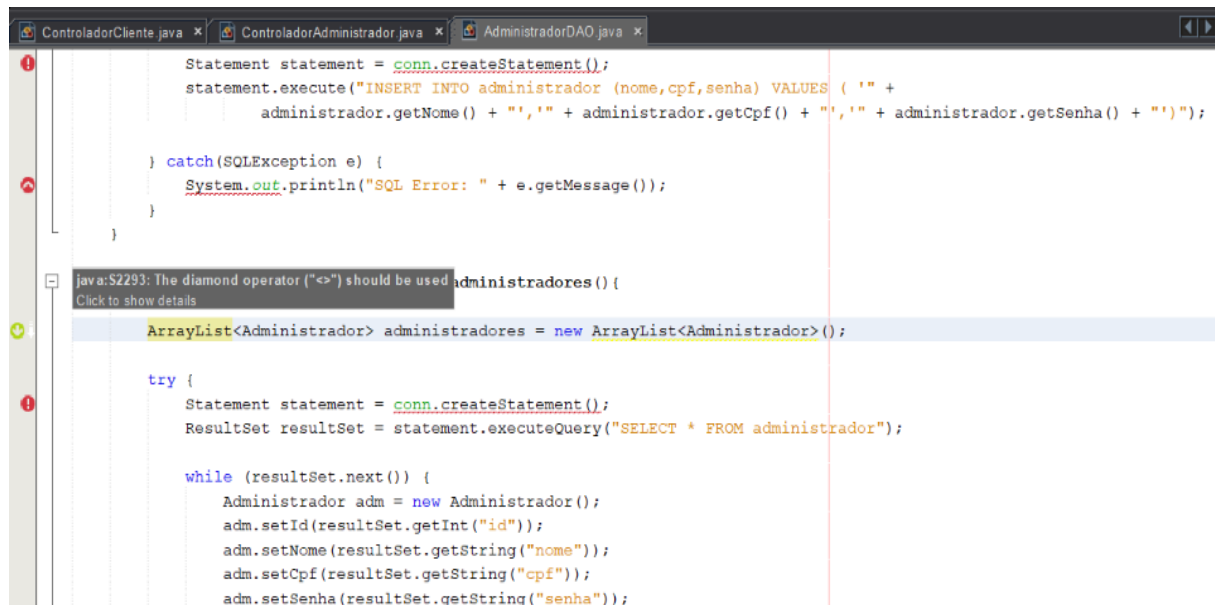
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Usar uma declaração mais genérica de lista, como `List<Administrador>`.

5.4.23 Problema 23:

- **Descrição:** O erro “S2293” apresenta o uso desnecessário da declaração do tipo do array no construtor.

- **Localização:** Existem 60 ocorrências no código-fonte. Abaixo o exemplo é na classe AdministradorDAO.java.



```
Statement statement = conn.createStatement();
statement.execute("INSERT INTO administrador (nome,cpf,senha) VALUES ( ' " +
    administrador.getNome() + "','" + administrador.getCpf() + "','" + administrador.getSenha() + "')");

} catch(SQLException e) {
    System.out.println("SQL Error: " + e.getMessage());
}

}

java:S2293: The diamond operator ("<=>") should be used administradores() {
Click to show details

    ArrayList<Administrador> administradores = new ArrayList<Administrador>();

    try {
        Statement statement = conn.createStatement();
        ResultSet resultSet = statement.executeQuery("SELECT * FROM administrador");

        while (resultSet.next()) {
            Administrador adm = new Administrador();
            adm.setId(resultSet.getInt("id"));
            adm.setNome(resultSet.getString("nome"));
            adm.setCpf(resultSet.getString("cpf"));
            adm.setSenha(resultSet.getString("senha"));
        }
    }
}
```

- **Prioridade:** Minor (Code Smell).

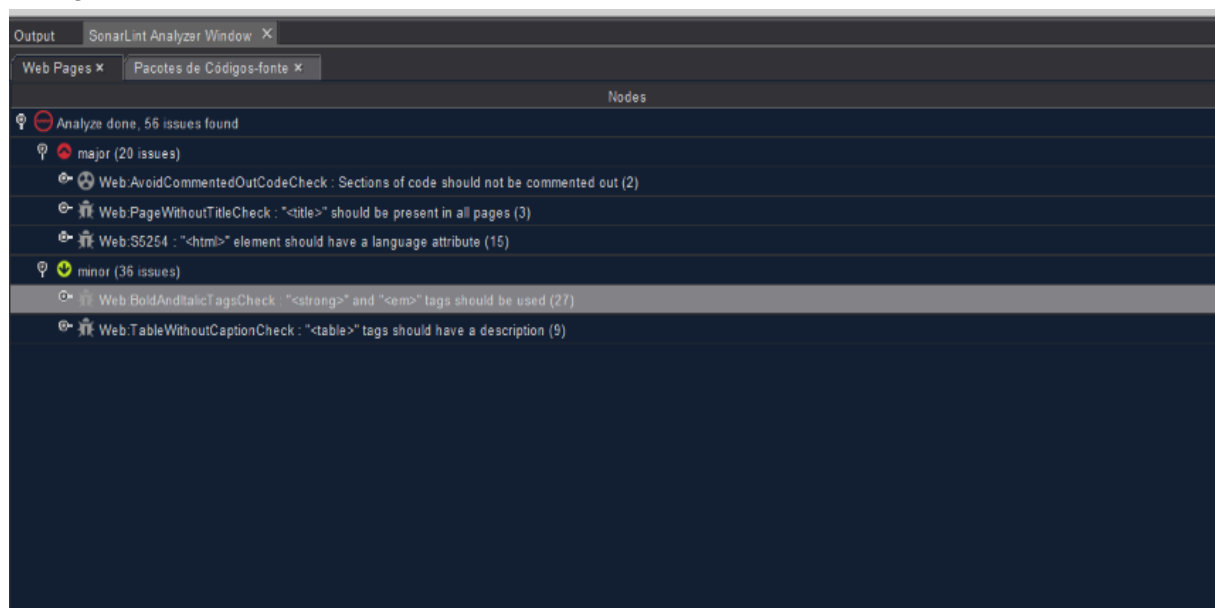
- **Recomendação:** Deixar a declaração de tipo do array apenas no objeto que vai instanciar a classe. A partir do Java 7, o compilador já infere o tipo do construtor sem que seja necessário declará-lo.

5.5. Conclusões:

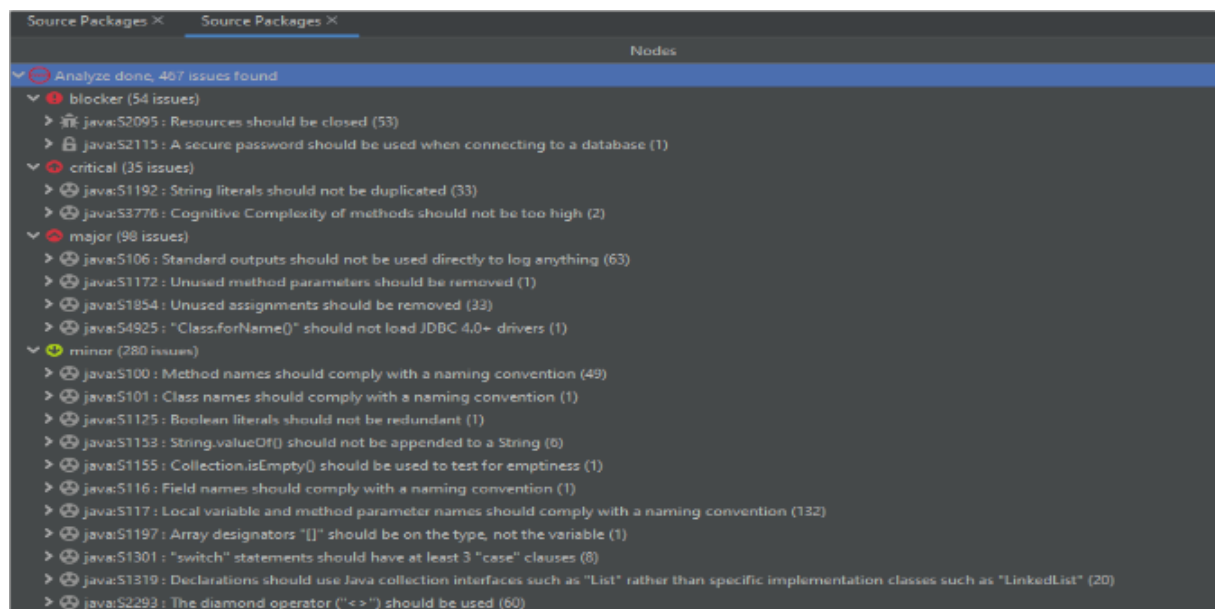
Durante o processo de inspeção de código, com a ajuda do Sonar Lint, pudemos constatar a presença de diversos tipos de erro e diferentes níveis de severidade, mas que a maioria consistia em “code smells”. A partir disso, foi possível tomar conhecimento a respeito dos erros, sobre os diferentes tipos e bem como ao grau de seriedade. Dessa forma, os integrantes do grupo se uniram de forma a estudar e solucionar os erros encontrados no projeto.

5.6. Anexos:

Lista geral com os erros detalhes no item 5.4:



```
Output SonarLint Analyzer Window X
Web Pages x Pacotes de Códigos-fonte x
Nodes
Analyze done, 56 issues found
major (20 issues)
Web.AvoidCommentedOutCodeCheck : Sections of code should not be commented out (2)
Web.PageWithoutTitleCheck : "<title>" should be present in all pages (3)
Web.S6254 : "<html>" element should have a language attribute (15)
minor (36 issues)
Web.BoldAndItalicTagsCheck : "<strong>" and "<em>" tags should be used (27)
Web.TableWithoutCaptionCheck : "<table>" tags should have a description (9)
```

6. Projeto de casos de testes

Teste 1 - História do Usuário

Objetivo: Testar se os botões “home”, “especialidades” e “convênios” estão funcionando Instruções:	
Informações sobre a execução do teste: Nome do testador: Carolina Arêas Data(s) do teste: 17/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: Para a realização desses testes é necessário estar na página home Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11 Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “Especialidades”	A página inicial deve rolar para a sessão de especialidades	A página inicial rolou para a sessão de especialidades		Passou
2.	Clicar no botão “convênios”	A página inicial deve rolar para a sessão de “convênios”	A página inicial rolou para a sessão de “convênios”		Passou
3	Clicar no botão “Home”	A página inicial deve rolar para a sessão de “home” (parte superior da página)	A página inicial rolou para a sessão de “home”		Passou

Teste 2 - Teste de Sistema do fluxo principal completo

Objetivo: Teste para verificar o fluxo de consulta onde o paciente solicita a consulta e o médico registra que a consulta foi realizada

Instruções:

Informações sobre a execução do teste:

Nome do testador: Carolina Arêas

Data(s) do teste: 17/06/2023

Local/servidor em uso: Servidor de teste

Pré-requisitos para este teste: O login e senha do paciente e do médico devem estar registrados no banco de dados

Versões de software:

Navegador: Microsoft Edge Versão 114.0.1823.67

Sistema operacional: Windows 11

Configurações necessárias: Selenium configurado para a versão adequada no navegador

NOTAS E RESULTADOS: O fluxo principal está funcionando de forma correta

ETAPAS/RESULTADOS DO SCRIPT DE TESTE

Passo	Etapa/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/ Falhou
1.	Clicar no botão “login”	A página deve ser alterada para a de login	A página foi alterada para a de login		Passou
2.	Clicar no botão select do tipo de acesso	Deve abrir uma lista com as opções de “cliente”, “administrador” e “médico”	Abriu uma lista com as opções de “cliente”, “administrador” e “médico”		Passou
3.	Selecionar “Cliente”	o botão select deve estar com o nome “Cliente”	o botão select ficou com o nome “Cliente”		Passou
4.	Inserir CPF no campo de entrada do “CPF”	O CPF inserido deve aparecer dentro do campo de entrada	O CPF inserido aparece dentro do campo de entrada	CPF está registrado no banco de dados	Passou
5.	Inserir senha no campo de entrada da “senha”	A senha inserida deve aparecer dentro do campo de entrada de forma censurada	A senha inserida apareceu dentro do campo de entrada de forma censurada	Senha está registrada no banco de dados	Passou
6.	Clicar no botão “enviar”	A página deve ser alterada para a área do cliente	A página foi alterada para a área do cliente		Passou
7.	Clicar no botão “Marcar Consulta”	A página deve ser direcionada para uma página contendo um formulário	A página foi direcionada para uma página contendo um formulário		Passou
8.	Inserir data no campo de entrada da “Data”	A data inserida deve aparecer dentro do campo de entrada	A data inserida apareceu dentro do campo de entrada		Passou
9.	Inserir hora no campo de entrada da “Hora”	A hora inserida deve aparecer dentro do campo de entrada	A hora inserida apareceu dentro do campo de entrada		Passou

10.	Clicar no botão select do tipo de Especialidade Médica	Abriu uma lista com as opções de Especialidades Médicas	Abriu uma lista com as opções de Especialidades Médicas disponíveis		Passou
11.	Clicar no botão select do tipo de Especialidade Médica	o botão select deve estar com o nome do tipo de Especialidade Médica que foi selecionado	o botão select ficou com o nome do tipo de Especialidade Médica que foi selecionado		Passou
12.	Clicar no botão “enviar”	A página deve ser direcionada para um página com aviso que o pedido de consulta foi realizado com sucesso	A página foi direcionada para um página com aviso que o pedido de consulta foi realizado com sucesso		Passou
13.	Clicar no botão “voltar”	A página deve ser direcionada para a área do Paciente	A página foi direcionada para a área do Paciente		Passou
14.	Clicar no botão de logOut	A página deve ser redirecionada para a página inicial, e as opções de Login e Cadastro devem estar novamente disponíveis na página	A página foi redirecionada para a página inicial, e as opções de Login e Cadastro ficaram novamente disponíveis na página		Passou
15.	Clicar no botão “login”	A página deve ser alterada para a de login	A página foi alterada para a de login		Passou
16.	Clicar no botão select do tipo de acesso	Deve abrir uma lista com as opções de “cliente”, “administrador” e “Médico”	Abriu uma lista com as opções de “cliente”, “administrador” e “Médico”		Passou
17.	Selecionar “Médico”	o botão select deve estar com o nome “Médico”	o botão select ficou com o nome “Médico”		Passou
18.	Inserir CPF no campo de entrada do “CPF”	O CPF inserido deve aparecer dentro do campo de entrada	O CPF inserido aparece dentro do campo de entrada	CPF está registrado no banco de dados	Passou
19.	Inserir senha no campo de entrada da “senha”	A senha inserida deve aparecer dentro do campo de entrada de forma censurada	A senha inserida apareceu dentro do campo de entrada de forma censurada	Senha está registrada no banco de dados	Passou
20.	Clicar no botão “enviar”	A página deve ser alterada para a área do Médico	A página foi alterada para a área do Médico		Passou
21.	Clicar no botão “Visualizar Consultas”	A página deve ser direcionada para a página de consultas contendo a tabela	A página foi direcionada para a página de consultas contendo a tabela	O usuário estava logado como a propriedade “Médico”	Passou
22.	Clicar no botão “Marcar como	A página deve ser	A página foi	A consulta	Passou

	Concluída” da consulta desejada	direcionada para uma página contendo um formulário	direcionada para uma página contendo um formulário	estava a com o campo “Realizada” como “Não”	
23.	Preencher o campo de entrada “Descrição “	As informações da descrição devem aparecer no campo de entrada	As informações da descrição apareceram no campo de entrada		Passou
24.	Clicar no botão “enviar”	A página deve ser direcionada para um página com aviso que a consulta foi realizada	A página foi direcionada para um página com aviso que a consulta foi realizada		Passou
25.	Clicar no botão “voltar”	A página deve ser redirecionada para a página com a tabela de consultas, e a consulta em questão deve estar com o campo “Realizada” como “Sim”, e o botão de “Marcar como concluída” deve ser alterado para o botão de “Solicitar exame”	A página foi redirecionada para a página com a tabela de consultas, e a consulta em questão está com o campo “Realizada” como “Sim”, e o botão de “Marca como concluída” foi alterado para o botão de “Solicitar exame”		Passou

Teste 3 - História do usuário: realização de login

Objetivo: Testar a realização do login de um Administrador o qual os dados já estejam registrados Instruções:	
Informações sobre a execução do teste: Nome do testador: Carolina Arêas Data(s) do teste: 17/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: O login e senha devem estar registrados no banco de dados Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11 Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “login”	A página deve ser alterada para a de login	A página foi alterada para a de login		Passou
2.	Clicar no botão select do tipo de acesso	Deve abrir uma lista com as opções de “cliente”, “administrador” e “Médico”	Abriu uma lista com as opções de “cliente”, “administrador” e “Médico”		Passou

3.	Selecionar “Administrador”	o botão select deve estar com o nome “Administrador”	o botão select ficou com o nome “Administrador”		Passou
4.	Inserir CPF no campo de entrada do “CPF”	O CPF inserido deve aparecer dentro do campo de entrada	O CPF inserido aparece dentro do campo de entrada	CPF está registrado no banco de dados	Passou
5.	Inserir senha no campo de entrada da “senha”	A senha inserida deve aparecer dentro do campo de entrada de forma censurada	A senha inserida apareceu dentro do campo de entrada de forma censurada	Senha está registrada no banco de dados	Passou
6.	Clicar no botão “enviar”	A página deve ser alterada para a área do Administrador	A página foi alterada para a área do Administrador		Passou

Teste 4 - História do usuário: realização de login

Objetivo: Testar a realização do login de um paciente o qual os dados não estejam registrados Instruções:	
Informações sobre a execução do teste: Nome do testador: Natália Bruno Rabelo Data(s) do teste: 18/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: O login e senha devem <u>não</u> estar registrados no banco de dados Versões de software: aplicativo: Navegador: Microsoft Edge Banco de dados: Sistema operacional: Configurações necessárias:
NOTAS E RESULTADOS: o teste é equivalente para caso do usuário digitar seu CPF ou senha de forma incorreta	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “login”	A página deve ser alterada para a de login	A página foi alterada para a de login		Passou
2.	Clicar no botão select do tipo de acesso	Deve abrir uma lista com as opções de “cliente”, “administrador” e “médico”	Abriu uma lista com as opções de “cliente”, “administrador” e “médico”		Passou
3.	Selecionar “Cliente”	o botão select deve estar com o nome “Cliente”	o botão select ficou com o nome “Cliente”		Passou
4.	Inserir CPF no campo de entrada do “CPF”	O CPF inserido deve aparecer dentro do campo de entrada	O CPF inserido aparece dentro do campo de entrada	CPF não está registrado no banco de dados	Passou
5.	Inserir senha no campo de entrada da “senha”	A senha inserida deve aparecer dentro do campo de entrada de forma censurada	A senha inserida apareceu dentro do campo de entrada de forma censurada	Senha não está registrada no banco de dados	Passou

6.	Clicar no botão “enviar”	A página deve ser alterada para a mensagem de erro alertando que o login ou senha estão incorretos	A página foi alterada para a mensagem de erro alertando que o login ou senha estão incorretos		Passou
----	--------------------------	--	---	--	--------

Teste 5 - História do usuário: dados não registrados

Objetivo: Testar a realização do login de um médico o qual os dados não estejam registrados	
Instruções:	
Informações sobre a execução do teste: Nome do testador: Carolina Arêas Data(s) do teste: 17/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: O login e senha devem <u>não</u> estar registrados no banco de dados
	Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11
	Configurações necessárias:
NOTAS E RESULTADOS: o teste é equivalente para caso do usuário digitar seu CPF ou senha de forma incorreta	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “login”	A página deve ser alterada para a de login	A página foi alterada para a de login		Passou
2.	Clicar no botão select do tipo de acesso	Deve abrir uma lista com as opções de “cliente”, “administrador” e “Médico”	Abriu uma lista com as opções de “cliente”, “administrador” e “Médico”		Passou
3.	Selecionar “Médico”	o botão select deve estar com o nome “Médico”	o botão select ficou com o nome “Médico”		Passou
4.	Inserir CPF no campo de entrada do “CPF”	O CPF inserido deve aparecer dentro do campo de entrada	O CPF inserido aparece dentro do campo de entrada	CPF não está registrado no banco de dados	Passou
5.	Inserir senha no campo de entrada da “senha”	A senha inserida deve aparecer dentro do campo de entrada de forma censurada	A senha inserida apareceu dentro do campo de entrada de forma censurada	Senha não está registrada no banco de dados	Passou
6.	Clicar no botão “enviar”	A página deve ser alterada para a mensagem de erro alertando que o login ou senha estão incorretos	A página foi alterada para a mensagem de erro alertando que o login ou senha estão incorretos		Passou

Teste 6 - História do usuário: ação de cadastro

Objetivo: Testar a ação de cadastro Instruções:	
Informações sobre a execução do teste: Nome do testador: Carolina Arêas Data(s) do teste: 17/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: É necessário não estar logado e não ter seus dados cadastrados no banco de dados Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11 Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “Cadastre-se”	A página deve ser redirecionada para a página de cadastro	A página foi redirecionada para a página de cadastro	O usuário não estava logado	Passou
2.	Inserir nome no campo de entrada do “nome”	O nome inserido deve aparecer dentro do campo de entrada	O nome inserido aparece dentro do campo de entrada		Passou
3.	Inserir CPF no campo de entrada do “CPF”	O CPF inserido deve aparecer dentro do campo de entrada	O CPF inserido aparece dentro do campo de entrada	CPF não está registrado no banco de dados	Passou
4.	Inserir senha no campo de entrada da “senha”	A senha inserida deve aparecer dentro do campo de entrada de forma censurada	A senha inserida apareceu dentro do campo de entrada de forma censurada		Passou
5.	Clicar no botão select do tipo de Plano de Saúde	Deve abrir uma lista com as opções dos planos de saúde disponíveis	Abriu uma lista com as opções dos planos de saúde disponíveis		Passou
6.	Selecionar o plano de saúde desejado	o botão select deve estar com o plano de saúde desejado	o botão select ficou com o plano de saúde desejado		Passou
7.	Clicar no botão “enviar”	A página deve ser direcionada para um página com aviso que o cadastro foi realizado e as informações do cadastro devem estar registradas no banco de dados	A página foi direcionada para um página com aviso que o cadastro foi realizado e as informações do cadastro foram registradas no banco de dados		Passou

Teste 7- História do usuário: acessar páginas

Objetivo: Testar a ação de abrir páginas de consultas do usuário “Médico” Instruções:
--

Informações sobre a execução do teste: Nome do testador: Natália Bruno Rabelo Data(s) do teste: 18/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: É necessário estar logado como a propriedade “médico”
	Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11
	Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “Visualizar Consultas”	A página deve ser direcionada para a página de consultas contendo a tabela	A página foi direcionada para a página de consultas contendo a tabela	O usuário estava logado como a propriedade “Médico”	Passou

Teste 8 - História do usuário

Objetivo: Testar a ação de marcar uma consulta que estava com o status de “Não” para “Sim” na categoria “Realizada da tabela de consultas” Instruções:	
Informações sobre a execução do teste: Nome do testador: Carolina Arêas Data(s) do teste: 18/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: É necessário estar logado como a propriedade “médico”; é necessário selecionar uma consulta que esteja com o campo “Realizada” como “Não”
	Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11
	Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “Visualizar Consultas”	A página deve ser direcionada para a página de consultas contendo a tabela	A página foi direcionada para a página de consultas contendo a tabela	O usuário estava logado como a propriedade “Médico”	Passou
2.	Clicar no botão “Marcar como Concluída” da consulta desejada	A página deve ser direcionada para uma página contendo um formulário	A página foi direcionada para uma página contendo um formulário	A consulta estava a com o campo “Realizada” como “Não”	Passou
3.	Preencher o campo de entrada	As informações da	As informações da		Passou

	“Descrição “	descrição devem aparecer no campo de entrada	descrição apareceram no campo de entrada		
4.	Clicar no botão “enviar”	A página deve ser direcionada para um página com aviso que a consulta foi realizada	A página foi direcionada para um página com aviso que a consulta foi realizada		Passou
5.	Clicar no botão “voltar”	A página deve ser redirecionada para a página com a tabela de consultas, e a consulta em questão deve estar com o campo “Realizada” como “Sim”, e o botão de “Marcar como concluída” deve ser alterado para o botão de “Solicitar exame”	A página foi redirecionada para a página com a tabela de consultas, e a consulta em questão está com o campo “Realizada” como “Sim”, e o botão de “Marca como concluída” foi alterado para o botão de “Solicitar exame”		Passou

Teste 9 - História do usuário

Objetivo: Testar a ação de solicitar um exame de uma consulta concluída que não há nenhum exame registrado Instruções:	
Informações sobre a execução do teste: Nome do testador: Carolina Arêas Data(s) do teste: 17/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: É necessário estar logado como a propriedade “médico”; é necessário selecionar uma consulta que esteja com o campo “Realizada” como “Sim”; é necessário selecionar uma consulta que esteja com o campo “Exames” como “Não há exames” Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11 Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “Visualizar Consultas”	A página deve ser direcionada para a página de consultas contendo a tabela	A página foi direcionada para a página de consultas contendo a tabela	O usuário estava logado como a propriedade “Médico”	Passou
2.	Clicar no botão “Solicitar Exame” da consulta desejada	A página deve ser direcionada para uma página contendo um formulário	A página foi direcionada para uma página contendo um formulário	A consulta estava a com o campo “Realizada” como “Sim”	Passou
3.	Clicar no botão select do tipo de exame	Abriu uma lista com as opções de exames disponíveis	Abriu uma lista com as opções de exames disponíveis		Passou

4.	Selecionar o tipo de exame desejado	o botão select deve estar com o nome do tipo de exame que foi selecionado	o botão select ficou com o nome do tipo de exame que foi selecionado		Passou
5.	Clicar no botão “enviar”	A página deve ser direcionada para um página com aviso que o exame foi solicitado com sucesso	A página foi direcionada para um página com aviso que o exame foi solicitado com sucesso		Passou
6.	Clicar no botão “voltar”	A página deve ser redirecionada para a página com a tabela de consultas, e a consulta em questão deve estar com o campo “exames” contendo um botão select com tipo de exame solicitado	A página foi redirecionada para a página com a tabela de consultas, e a consulta em questão está com o campo “exames” contendo um botão select com tipo de exame solicitado		Passou

Teste 10 - História do usuário: registro de dados

Objetivo: Testar a ação de solicitar um exame de uma consulta concluída que já tenha pelo menos um exame registrado Instruções:	
Informações sobre a execução do teste: Nome do testador: Carolina Arêas Data(s) do teste: 17/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: É necessário estar logado como a propriedade “médico”; é necessário selecionar uma consulta que esteja com o campo “Realizada” como “Sim”; é necessário selecionar uma consulta que esteja com o campo “Exames” contendo um botão select Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11 Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “Visualizar Consultas”	A página deve ser direcionada para a página de consultas contendo a tabela	A página foi direcionada para a página de consultas contendo a tabela	O usuário estava logado como a propriedade “Médico”	Passou
2.	Clicar no botão “Solicitar Exame” da consulta desejada	A página deve ser direcionada para uma página contendo um formulário	A página foi direcionada para uma página contendo um formulário	A consulta estava com o campo “Realizada” como “Sim”	Passou
3.	Clicar no botão select do tipo de exame	Abriu uma lista com as opções de exames	Abriu uma lista com as opções de exames		Passou

		disponíveis	disponíveis		
4.	Selecionar o tipo de exame desejado	o botão select deve estar com o nome do tipo de exame que foi selecionado	o botão select ficou com o nome do tipo de exame que foi selecionado		Passou
5.	Clicar no botão “enviar”	A página deve ser direcionada para um página com aviso que o exame foi solicitado com sucesso	A página foi direcionada para um página com aviso que o exame foi solicitado com sucesso		Passou
6.	Clicar no botão “voltar”	A página deve ser redirecionada para a página com a tabela de consultas, e a consulta em questão deve estar com o campo “exames” contendo um botão select com um tipo de exame solicitado	A página foi redirecionada para a página com a tabela de consultas, e a consulta em questão está com o campo “exames” contendo um botão select com um tipo de exame solicitado		Passou
7.	Clicar no botão de select da parte de “Exames”	O botão select deve abrir uma lista com o tipo de exame previamente solicitado e com o exame recém solicitado	O botão select abriu uma lista com o tipo de exame previamente solicitado e com o exame recém solicitado		Passou

Teste 11 - História do usuário

Objetivo: Testar a ação de cadastrar “Plano”	
Informações sobre a execução do teste: Nome do testador: Carolina Arêas Data(s) do teste: 17/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: É necessário estar logado como a propriedade “administrador”
	Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11
	Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “Ver Planos” da página do administrador	A página deve ser direcionada para a página de planos contendo a tabela	A página foi direcionada para a página de planos contendo a tabela	O usuário estava logado como a propriedade “Administrador”	Passou
2.	Clicar no botão “Cadastrar Plano”	A página deve ser direcionada para uma página contendo um formulário	A página foi direcionada para uma página contendo um formulário		Passou

	Preencher o campo de entrada “Descrição “	As informações da descrição devem aparecer no campo de entrada	As informações da descrição apareceram no campo de entrada		Passou
3.	Clicar no botão “Cadastrar”	A página deve ser direcionada para um página com aviso que o plano foi cadastrado com sucesso e o plano deve aparecer registrado no banco de dados	A página foi direcionada para um página com aviso que o plano foi cadastrado com sucesso e o plano apareceu registrado no banco de dados		Passou
4.	Clicar no botão “voltar”	A página deve ser redirecionada para a página com a tabela dos planos, e o plano recém registrado deve aparecer no final da lista	A página foi redirecionada para a página com a tabela dos planos, e o plano recém registrado apareceu no final da lista		Passou

Teste 12 - História do usuário: edição

Objetivo: Testar a ação de editar “Plano”	
Informações sobre a execução do teste: Nome do testador: Natália Bruno Rabelo Data(s) do teste: 17/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: É necessário estar logado como a propriedade “administrador”; é necessário que o plano já esteja registrado Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11 : Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/ Falhou
1.	Clicar no botão “Ver Planos” da página do administrador	A página deve ser direcionada para a página de planos contendo a tabela	A página foi direcionada para a página de planos contendo a tabela	O usuário estava logado como a propriedade “Administrador”	Passou
2.	Clicar no botão “Editar” do plano desejado	A página deve ser direcionada para uma página contendo um formulário	A página foi direcionada para uma página contendo um formulário		Passou
3.	Preencher o campo de entrada “Descrição “	As informações da descrição devem aparecer no campo de entrada	As informações da descrição apareceram no campo de entrada		Passou
4.	Clicar no botão “Editar”	A página deve ser direcionada para um	A página foi direcionada para um		Passou

		página com aviso que o plano foi editado com sucesso, a alteração deve aparecer registrado no banco de dados	página com aviso que o plano foi editado com sucesso, a alteração apareceu registrado no banco de dados		
5.	Clicar no botão “voltar”	A página deve ser redirecionada para a página com a tabela dos planos, e o plano editado deve aparecer atualizado	A página foi redirecionada para a página com a tabela dos planos, e o plano editado apareceu atualizado		Passou

Teste 13 - História do usuário: exclusão de dados

Objetivo: Testar a ação de excluir “Plano”	
Informações sobre a execução do teste: Nome do testador: Carolina Arêas Data(s) do teste: 17/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: É necessário estar logado como a propriedade “administrador”; é necessário que o plano já esteja registrado Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11 Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapas/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “Ver Planos” da página do administrador	A página deve ser direcionada para a página de planos contendo a tabela	A página foi direcionada para a página de planos contendo a tabela	O usuário estava logado como a propriedade “Administrador”	Passou
2.	Clicar no botão “excluir” do plano desejado	A página deve ser direcionada para uma página contendo um aviso que o plano foi excluído	A página foi direcionada para uma página contendo um aviso que o plano foi excluído		Passou
3.	Clicar no botão “voltar”	A página deve ser redirecionada para a página com a tabela dos planos, e o plano excluído não deve aparecer na lista, nem no banco de dados	A página foi redirecionada para a página com a tabela dos planos, e o plano excluído não aparece na lista, nem no banco de dados		Passou

Teste 14 - História do usuário

Objetivo: Testar a ação de cadastrar “Médico” pelo administrador

Informações sobre a execução do teste: Nome do testador: Natália Bruno Rabelo Data(s) do teste: 19/06/2023 Local/servidor em uso:	Pré-requisitos para este teste: É necessário estar logado como a propriedade “administrador”;
	Versões de software: Navegador: Microsoft Edge Versão 114.0.1823.67 Sistema operacional: Windows 11
	Configurações necessárias:
NOTAS E RESULTADOS:	

ETAPAS/RESULTADOS DO SCRIPT DE TESTE					
Passo	Etapa/Entrada do teste	Resultados esperados	Resultados reais	Requisitos validados	Passou/Falhou
1.	Clicar no botão “Ver Médicos” da página do administrador	A página deve ser direcionada para a página de médicos contendo a tabela	A página foi direcionada para a página de médicos contendo a tabela	O usuário estava logado como a propriedade “Administrador”	Passou
2.	Clicar no botão “Cadastrar médicos”	A página deve ser direcionada para um formulário	A página foi direcionada para uma página contendo um formulário		Passou
3.	Inserir o nome do médico no campo de entrada do “Nome”	O nome inserido deve aparecer dentro do campo de entrada	O nome inserido aparece dentro do campo de entrada		Passou
4.	Inserir o CRM do médico no campo de entrada do “Nome”	O CRM inserido deve aparecer dentro do campo de entrada	O CRM inserido aparece dentro do campo de entrada		Passou
5.	Clicar no botão select do Estado do CRM	Deve abrir uma lista com as opções dos Estados	Abriu uma lista com as opções dos Estados		Passou
6.	Selecionar o Estado do CRM	o botão select deve estar com o nome Estado selecionado	o botão select ficou com o nome Estado selecionado		Passou
7.	Clicar no botão “enviar”	A página deve ser alterada para um aviso que o Médico foi registrado	A página deve ser alterada para um aviso que o Médico foi registrado		Passou

7. Relatório de cobertura dos testes de mutação

Nosso foco de aumento de cobertura para teste de mutação foram as classes ClienteDAO, ConsultaDAO, EspecialidadeDAO, ExameDAO e PlanoDAO.

Relatório de cobertura antes das alterações para cobrir casos de mutação

Pit Test Coverage Report

Package Summary

model

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
6	91% <div><div></div></div> 557/613	63% <div><div></div></div> 85/134	64% <div><div></div></div> 85/132

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
ClienteDAO.java	95% <div><div></div></div> 117/123	53% <div><div></div></div> 17/32	53% <div><div></div></div> 17/32
ConsultaDAO.java	93% <div><div></div></div> 86/92	70% <div><div></div></div> 14/20	70% <div><div></div></div> 14/20
EspecialidadeDAO.java	91% <div><div></div></div> 74/81	92% <div><div></div></div> 12/13	100% <div><div></div></div> 12/12
ExameDAO.java	93% <div><div></div></div> 78/84	83% <div><div></div></div> 10/12	83% <div><div></div></div> 10/12
MedicoDAO.java	84% <div><div></div></div> 128/153	44% <div><div></div></div> 20/45	45% <div><div></div></div> 20/44
PlanoDAO.java	93% <div><div></div></div> 74/80	100% <div><div></div></div> 12/12	100% <div><div></div></div> 12/12

Report generated by [PIT](#) 1.14.1

Relatório de cobertura após as alterações para cobrir mais casos:

Pit Test Coverage Report

Package Summary

model

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
6	90% <div><div></div></div> 589/653	74% <div><div></div></div> 105/142	75% <div><div></div></div> 105/140

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
ClienteDAO.java	91% <div><div></div></div> 149/163	80% <div><div></div></div> 32/40	80% <div><div></div></div> 32/40
ConsultaDAO.java	93% <div><div></div></div> 86/92	80% <div><div></div></div> 16/20	80% <div><div></div></div> 16/20
EspecialidadeDAO.java	91% <div><div></div></div> 74/81	92% <div><div></div></div> 12/13	100% <div><div></div></div> 12/12
ExameDAO.java	93% <div><div></div></div> 78/84	83% <div><div></div></div> 10/12	83% <div><div></div></div> 10/12
MedicoDAO.java	84% <div><div></div></div> 128/153	51% <div><div></div></div> 23/45	52% <div><div></div></div> 23/44
PlanoDAO.java	93% <div><div></div></div> 74/80	100% <div><div></div></div> 12/12	100% <div><div></div></div> 12/12

Report generated by [PIT](#) 1.14.1

8. Relatório de cobertura dos testes estruturais

Nosso foco de aumento de cobertura para teste estrutural foram as classes ClienteDAO, ConsultaDAO, EspecialidadeDAO, ExameDAO e PlanoDAO.

Relatório de cobertura antes das alterações para cobrir mais casos:

Filename	Coverage	Total	Not Executed
servlet.ControladorAdministrador	0,00 %	345	345
servlet.ControladorCliente	0,00 %	88	88
servlet.ControladorMedico	0,00 %	63	63
servlet.Cadastrar	0,00 %	32	32
servlet.Autenticar	0,00 %	41	41
utils.Constantes	0,00 %	2	2
aplicacao.Administrador	0,00 %	13	13
aplicacao.Medico	0,00 %	25	25
model.AdministradorDAO	0,00 %	75	75
model.MedicoDAO	0,00 %	153	153
model.EspecialidadeDAO	49,38 %	81	41
conexao.ConexaoBancoDeDados	60,00 %	10	4
model.ExameDAO	64,29 %	84	30
model.PlanoDAO	70,00 %	80	24
model.ConsultaDAO	73,91 %	92	24
model.ClienteDAO	78,05 %	123	27
aplicacao.Consulta	94,74 %	19	1
aplicacao.Cliente	94,74 %	19	1
aplicacao.Exame	100,00 %	7	0
aplicacao.Especialidade	100,00 %	7	0
aplicacao.Plano	100,00 %	7	0
Total	27,60 %	1366	989

Relatório de cobertura após as alterações para cobrir mais casos:

Total Coverage: 45,79 %			
Filename	Coverage	Total	Not Executed
servlet.ControladorAdministrador	0,00 %	350	350
servlet.ControladorCliente	0,00 %	91	91
servlet.ControladorMedico	0,00 %	66	66
servlet.Cadastrar	0,00 %	36	36
servlet.Autenticar	0,00 %	41	41
utils.Constantes	0,00 %	2	2
aplicacao.Administrador	0,00 %	13	13
model.AdministradorDAO	0,00 %	76	76
conexao.ConexaoBancoDeDados	58,33 %	12	5
model.MedicoDAO	76,62 %	154	36
model.PlanoDAO	91,36 %	81	7
model.EspecialidadeDAO	91,36 %	81	7
model.ExameDAO	91,76 %	85	7
model.ConsultaDAO	92,47 %	93	7
model.ClienteDAO	94,35 %	124	7
aplicacao.Consulta	94,74 %	19	1
aplicacao.Cliente	94,74 %	19	1
aplicacao.Exame	100,00 %	7	0
aplicacao.Especialidade	100,00 %	7	0
aplicacao.Plano	100,00 %	7	0
aplicacao.Medico	100,00 %	25	0
Total	45,79 %	1389	753

9. Relatório de testes de sistema

Usando a ferramenta Selenium, foram realizados 3 testes automatizados.

9.1 Fluxo principal

Esse teste automatizado teve como objetivo verificar o fluxo principal do sistema, o qual o paciente realiza o login, marca uma consulta com um médico especializado, e esse médico registra que a consulta foi realizada. O teste seguiu os seguintes passos:

1. O teste começa clicando no botão “Log In” para realizar o login do paciente
2. o tipo de acesso é selecionado como “cliente”
3. O CPF e senha do paciente são preenchidos
4. O log In é realizado ao clicar no botão “enviar”
5. O botão “Marcar consulta” é clicado na página da área do paciente
6. A data e a hora da consulta são preenchidos
7. O botão “enviar” é clicado para marcar a consulta oficialmente
8. O botão “voltar” é clicado para voltar a página da área do paciente
9. O botão “Log Out” é clicado para ser realizado o log out como paciente
10. O teste começa clicando no botão “Log In” para realizar o login do médico
11. o tipo de acesso é selecionado como “médico”

12. O CPF e senha do médico são preenchidos
13. O log In é realizado ao clicar no botão “enviar”
14. O botão “visualizar consultas” na área do médico é clicado
15. O botão “concluir consulta” é clicado para a consulta em questão
16. O input da descrição da consulta é preenchido
17. O botão “enviar” é clicado para finalizar o processo

9.2 Fluxo para testar a ocorrência de erros do usuário

Esse teste automatizado teve como objetivo verificar se ao preencher o formulário de forma errada (no caso colocando o CPF correto de acordo com o registro no banco de dados, porém com uma senha errada), o sistema é capaz de funcionar corretamente emitindo a mensagem de erro correta para o cenário em questão. O experimento seria considerado falho se a mensagem de erro não aparecesse ou aparecesse com a mensagem de outro tipo de erro. O teste seguiu os seguintes passos:

1. O teste começa clicando no botão “Log In” para realizar o login do paciente
2. o tipo de acesso é selecionado como “cliente”
3. O CPF do paciente é preenchido de forma correta
4. A senha do paciente é preenchida de forma errada
5. O log In é realizado ao clicar no botão “enviar”

9.3 Fluxo para testar o cadastro do médico feito pelo administrador

Esse teste automatizado teve como objetivo verificar se os privilégios do administrador estão funcionando corretamente, ao realizar o fluxo no qual ele é capaz de cadastrar um médico no banco de dados. O teste seguiu os seguintes passos:

1. O teste começa clicando no botão “Log In” para realizar o login do administrador
2. o tipo de acesso é selecionado como “administrador”
3. O CPF e senha do administrador são preenchidos
4. O log In é realizado ao clicar no botão “enviar”
5. O botão “ver médicos” é clicado para visualizar os médicos cadastrados
6. O botão “cadastrar médico” é clicado para adicionar um novo médico no banco de dados
7. O nome do médico é inserido no input
8. O CRM do médico é inserido no input
9. O Estado do CRM é escolhido por um select
10. O CPF do médico é inserido no input
11. A senha do médico é inserido no input
12. A especialidade do médico é selecionada
13. O botão “cadastrar” é clicado para cadastrar o médico

9.4 Conclusões e observações

O sistema passou em todos os testes automatizados, não apresentando falhas em nenhum dos cenários testados. O cadastro dos dados foi realizado de forma eficiente no banco de dados, no caso do teste do fluxo principal, o qual era cadastrado horário e data de uma nova consulta, cada vez que o teste era efetuado era necessário apagar as informações no banco ou alterar essas informações para realizar o teste novamente, uma vez que ao reservar uma data e um horário o sistema não possibilita uma nova marcação com os mesmos inputs, demonstrando um bom funcionamento do sistema.

10. Relatório de testes de performance

Durante os testes feitos com o JMeter foram testados as requisições para a página index, cadastro e a sua efetuação, e os logins com os diferentes tipos de acesso que temos no sistema, durante esses testes aumentamos o loop para rodar as

mesmas requisições por 100, 1000 e 10000 vezes e no fim todas obtiveram sucesso, mesmo com tantas requisições em paralelo, resultando no máximo em a aplicação ficar mais lenta.

11. Relatório de testes de integração

AutenticarServlet:

Get de logout:

URL: <http://localhost:8080/ClinicaMaven/Autenticar?arg=Logout>

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post da autenticação de administrador:

URL: <http://localhost:8080/ClinicaMaven/Autenticar>

Método: POST

Parâmetros:

papel: Administrador

CPF: 249.252.810-38

senha: 111

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post da autenticação de paciente:

URL: <http://localhost:8080/ClinicaMaven/Autenticar>

Método: POST

Parâmetros:

papel: Cliente

CPF: 937.397.160-37

senha: 111

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post da autenticação de médico:

URL: <http://localhost:8080/ClinicaMaven/Autenticar>

Método: POST

Parâmetros:

papel: Medico

CPF: 381.585.150-53

senha: 111

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post da autenticação de usuário inexistente:

URL: <http://localhost:8080/ClinicaMaven/Autenticar>

Método: POST

Parâmetros:

papel: Medico

CPF: 0000

senha: 111

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

CadastrarServlet:

Get de visualizar página de cadastro:

URL: <http://localhost:8080/ClinicaMaven/Cadastrar?arg=Cadastrar>

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post de cadastro público de cliente:

URL: <http://localhost:8080/ClinicaMaven/Cadastrar>

Método: POST

Parâmetros:

nome: natalia

CPF: 123456

senha: 123

plano: 1

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

ClienteServlet:

Get visualizar página editar consulta:

URL: <http://localhost:8080/ClinicaMaven/ControladorCliente?arg=Editar&id=3>

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Get exibir formulário de marcar consulta:

URL: <http://localhost:8080/ClinicaMaven/ControladorCliente?arg=Marcar>

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Get visualizar consultas:

URL: <http://localhost:8080/ClinicaMaven/ControladorCliente?arg=Visualizar>

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Get visualizar página excluir consulta:

URL: <http://localhost:8080/ClinicaMaven/ControladorCliente?arg=Excluir&id=5>

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post editar consulta:

URL: <http://localhost:8080/ClinicaMaven/ControladorCliente?arg=Editar&id=1>

Método: POST

Parâmetros:

data: 02-07-2023

hora: 00:00:00

id_med: 1

id_cliente: 1

id_consulta: 1

acao: RemarcarConsulta

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post marcar consulta:

URL: <http://localhost:8080/ClinicaMaven/view/Confirmacao.jsp?type=Marcado>

Método: POST

Parâmetros:

data: 02-07-2023

hora: 00:00:00

id_med: 1

acao: Enviar

id_cliente: 1

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post excluir consulta:

URL: <http://localhost:8080/ClinicaMaven/view/Confirmacao.jsp?type=Excluido&=>

Método: POST

Parâmetros:

id_consulta: 5

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

MedicoServlet:

Get visualizar consultas:

URL: <http://localhost:8080/ClinicaMaven/ControladorMedico?arg=Visualizar>

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Get visualizar página de solicitar exame:

URL: <http://localhost:8080/ClinicaMaven/ControladorMedico?arg=SolicitarExame&id=1>

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Get de visualizar página de atender consulta:

URL: http://localhost:8080/ClinicaMaven/view/Confirmacao.jsp?type=Concluido

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post de solicitar exame:

URL: http://localhost:8080/ClinicaMaven/view/Confirmacao.jsp?type=Exame

Método: POST

Parâmetros:

id_exame: 1

id_consulta: 1

acao: Marcar Exame

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Post de atender consulta:

URL: http://localhost:8080/ClinicaMaven/view/Confirmacao.jsp?type=Concluido

Método: POST

Parâmetros:

id_consulta: 1

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Raíz das requisições:

Renderização da Página Principal:

URL: http://localhost:8080/ClinicaMaven/

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Renderização da Página de Login:

URL: http://localhost:8080/ClinicaMaven/view/Login.jsp

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Renderizando aba home:

URL: http://localhost:8080/ClinicaMaven/index.jsp

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Renderizando aba especialidades:

URL: http://localhost:8080/ClinicaMaven/index.jsp#Espec

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Renderizando aba convênios:

URL: <http://localhost:8080/ClinicaMaven/index.jsp#Conv>

Método: GET

Código de Resposta Esperado: 200

HTML de Resposta Esperado: N/A

Observação: Para todas as requisições, espera-se que o código de resposta retorne 200 e o HTML de resposta seja o esperado.

12. Relatório de testes unitários

ConsultaDAO Test:

Teste do Construtor da Classe ConsultaDAO (testConsultaDAO):

Verifica se a instanciação da classe ConsultaDAO não retorna um objeto nulo.

Teste do Método createConsulta da Classe ConsultaDAO (testCreateConsulta):

Cria uma nova consulta com dados válidos.

Obtém o número de consultas antes da inserção.

Chama o método createConsulta para adicionar a nova consulta ao banco de dados.

Obtém o número de consultas após a inserção.

Verifica se o número de consultas aumentou em 1.

Teste do Método getConsultas da Classe ConsultaDAO (testGetConsultas):

Define um ID de paciente válido.

Chama o método getConsultas para obter a lista de consultas do paciente.

Verifica se a lista de consultas retornada não é nula.

Teste do Método getConsulta da Classe ConsultaDAO (testGetConsulta):

Define um ID de consulta válido.

Chama o método getConsulta para obter os dados da consulta.

Verifica se a consulta retornada não é nula.

Teste do Método getMedicoEspecialidade da Classe ConsultaDAO (testGetMedicoEspecialidade):

Define um ID de consulta válido.

Cria uma lista vazia para armazenar os resultados.

Chama o método getMedicoEspecialidade para obter o médico e a especialidade associados à consulta.

Verifica se a lista retornada não é nula.

Teste do Método getProcedimentosDisponiveis da Classe ConsultaDAO (testGetProcedimentosDisponiveis):

Chama o método getProcedimentosDisponiveis para obter a lista de procedimentos disponíveis.

Verifica se a lista retornada não é nula.

Teste do Método updateConsulta da Classe ConsultaDAO (testUpdateConsulta):

Define um ID de consulta válido.

Cria uma nova consulta com dados válidos para atualização.
Chama o método `updateConsulta` para atualizar os dados da consulta no banco de dados.
Obtém a consulta atualizada do banco de dados.
Compara os campos da consulta atualizada com os dados usados para a atualização.

Teste do Método `deleteConsulta` da Classe `ConsultaDAO` (`testDeleteConsulta`):

Cria uma nova consulta para teste.
Obtém o ID da consulta criada.
Chama o método `deleteConsulta` para excluir a consulta do banco de dados.
Obtém a consulta novamente do banco de dados.
Verifica se a consulta retornada é nula, indicando que foi excluída com sucesso.
Testes Utilizando Mockito:
Os testes a partir desse ponto utilizam o framework Mockito para simular comportamentos de objetos e verificar interações.

Teste do Método `createConsulta` utilizando Mockito (`testCreateConsultaMock`):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método `execute`.
Chama o método `createConsulta` utilizando os mocks criados.
Verifica se o método `createStatement` da conexão e o método `execute` do statement foram chamados corretamente.

Teste do Método `getConsultas` utilizando Mockito (`testGetConsultasMock`):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método `executeQuery` e a obtenção de resultados.
Chama o método `getConsultas` utilizando os mocks criados.
Verifica se o método `createStatement` da conexão e o método `executeQuery` do statement foram chamados corretamente.

Teste do Método `getConsulta` utilizando Mockito (`testGetConsultaMock`):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método `executeQuery` e a obtenção de um resultado.
Chama o método `getConsulta` utilizando os mocks criados.
Verifica se o método `createStatement` da conexão e o método `executeQuery` do statement foram chamados corretamente.

Teste do Método `updateConsulta` utilizando Mockito (`testUpdateConsultaMock`):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método `execute`.
Chama o método `updateConsulta` utilizando os mocks criados.
Verifica se o método `createStatement` da conexão e o método `execute` do statement foram chamados corretamente.

Teste do Método `deleteConsulta` utilizando Mockito (`testDeleteConsultaMock`):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método `execute`.
Chama o método `deleteConsulta` utilizando os mocks criados.
Verifica se o método `createStatement` da conexão e o método `execute` do statement foram chamados corretamente.

Teste do Método getMedicoEspecialidade utilizando Mockito (testGetMedicoEspecialidadeMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.

Chama o método getMedicoEspecialidade utilizando os mocks criados- Verifica se o método createStatement da conexão e o método executeQuery do statement foram chamados corretamente.

Teste do Método getProcedimentosDisponiveis utilizando Mockito (testGetProcedimentosDisponiveisMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.

Chama o método getProcedimentosDisponiveis utilizando os mocks criados.

Verifica se o método createStatement da conexão e o método executeQuery do statement foram chamados corretamente.

Testes de Exceção:

Teste do Método getConsultas com SQLException (testGetConsultasSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método getConsultas.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getConsulta com SQLException (testGetConsultaSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método getConsulta.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getMedicoEspecialidade com SQLException (testGetMedicoEspecialidadeSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método getMedicoEspecialidade.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getProcedimentosDisponiveis com SQLException (testGetProcedimentosDisponiveisSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método getProcedimentosDisponiveis.

Verifica se a SQLException foi lançada corretamente.

Teste do Método updateConsulta com SQLException (testUpdateConsultaSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método updateConsulta.
Verifica se a SQLException foi lançada corretamente.

Teste do Método deleteConsulta com SQLException (testDeleteConsultaSQLException):

Cria um mock para a conexão com o banco de dados.
Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.
Chama o método deleteConsulta.
Verifica se a SQLException foi lançada corretamente.
Esses são os testes unitários implementados na classe

ClienteDAOTest:

Teste do Construtor da Classe ClienteDAO (testClienteDAO):

Verifica se a instanciação da classe ClienteDAO não retorna um objeto nulo.

Teste do Método createPaciente da Classe ClienteDAO (testCreatePaciente):

Cria um novo cliente com dados válidos.
Obtém o CPF do cliente antes da inserção.
Chama o método createPaciente para adicionar o novo cliente ao banco de dados.
Verifica se o cliente foi cadastrado corretamente no banco de dados.

Teste do Método getNomePaciente da Classe ClienteDAO (testGetNomePaciente):

Define um ID de cliente válido.
Chama o método getNomePaciente para obter o nome do cliente.
Verifica se o nome retornado corresponde ao nome esperado.

Teste do Método getPacientes da Classe ClienteDAO (testGetPacientes):

Chama o método getPacientes para obter a lista de clientes.
Verifica se a lista de clientes retornada não é nula e não está vazia.

Teste do Método getPaciente da Classe ClienteDAO (testGetPaciente):

Define um ID de cliente válido.
Chama o método getPaciente para obter os dados do cliente.
Verifica se o cliente retornado não é nulo e possui os dados esperados.

Teste do Método updatePaciente da Classe ClienteDAO (testUpdatePaciente):

Define um ID de cliente válido.
Obtém o cliente antes da atualização.
Atualiza os dados do cliente.
Chama o método updatePaciente para atualizar os dados do cliente no banco de dados.
Obtém o cliente atualizado do banco de dados.
Compara os campos do cliente atualizado com os dados usados para a atualização.

Teste do Método deletePaciente da Classe ClienteDAO (testDeletePaciente):

Cria um novo cliente para teste.

Obtém o ID do cliente criado.

Chama o método deletePaciente para excluir o cliente do banco de dados.

Obtém o cliente novamente do banco de dados.

Verifica se o cliente retornado é nulo, indicando que foi excluído com sucesso.

Testes Utilizando Mockito:

Os testes a partir desse ponto utilizam o framework Mockito para simular comportamentos de objetos e verificar interações.

Teste do Método login utilizando Mockito (testLoginMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de um resultado.

Chama o método login utilizando os mocks criados.

Verifica se o cliente retornado não é nulo e possui os dados esperados.

Teste do Método jaCadastrado utilizando Mockito (testJaCadastradoMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de um resultado.

Chama o método jaCadastrado utilizando os mocks criados.

Verifica se o resultado do método é verdadeiro.

Teste do Método getNomePaciente utilizando Mockito (testGetNomePacienteMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de um resultado.

Chama o método getNomePaciente utilizando os mocks criados.

Verifica se o nome retornado corresponde ao nome esperado.

Teste do Método getPacientes utilizando Mockito (testGetPacientesMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.

Chama o método getPacientes utilizando os mocks criados.

Verifica se a lista de clientes retornada não é nula e não está vazia.

Teste do Método getPaciente utilizando Mockito (testGetPacienteMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de um resultado.

Chama o método getPaciente utilizando os mocks criados.

Verifica se o cliente retornado não é nulo e possui os dados esperados.

Teste do Método updatePaciente utilizando Mockito (testUpdatePacienteMock):

Cria mocks para a conexão com o banco de dados e o statement.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.

Cria um cliente com dados válidos e um ID válido.

Chama o método updatePaciente utilizando os mocks criados.
Verifica se o método execute foi chamado corretamente.

Teste do Método getIdDeletePaciente utilizando Mockito (testGetIdDeletePacienteMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.
Chama o método getIdDeletePaciente utilizando os mocks criados.
Verifica se a lista de IDs de exames e consultas não está vazia.

Teste do Método deletePaciente utilizando Mockito (testDeletePacienteMock):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.
Chama o método deletePaciente utilizando os mocks criados.
Verifica se o método execute foi chamado corretamente.

Teste do Método login com SQLException (testLoginSQLException):

Cria um mock para a conexão com o banco de dados.
Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.
Chama o método login.
Verifica se a SQLException foi lançada corretamente.

Teste do Método jaCadastrado com SQLException (testJaCadastradoSQLException):

Cria um mock para a conexão com o banco de dados.
Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.
Chama o método jaCadastrado.
Verifica se a SQLException foi lançada corretamente.

Teste do Método getNomePaciente com SQLException (testGetNomePacienteSQLException):

Cria um mock para a conexão com o banco de dados.
Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.
Chama o método getNomePaciente.
Verifica se a SQLException foi lançada corretamente.

Teste do Método getPacientes com SQLException (testGetPacientesSQLException):

Cria um mock para a conexão com o banco de dados.
Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.
Chama o método getPacientes.
Verifica se a SQLException foi lançada corretamente.

Teste do Método getPaciente com SQLException (testGetPacienteSQLException):

Cria um mock para a conexão com o banco de dados.
Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.
Chama o método getPaciente.
Verifica se a SQLException foi lançada corretamente.

Teste do Método updatePaciente com SQLException (testUpdatePacienteSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Cria um cliente com dados válidos e um ID válido.

Chama o método updatePaciente.

Verifica se a SQLException foi lançada corretamente.

Teste do Método deletePaciente com SQLException (testDeletePacienteSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método deletePaciente.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getIdDeletePaciente com SQLException (testGetIdDeletePacienteSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método getIdDeletePaciente.

Verifica se a SQLException foi lançada corretamente.

EspecialidadeDAOTest:

Teste do Construtor da Classe EspecialidadeDAO (testEspecialidadeDAO):

Verifica se a instanciação da classe EspecialidadeDAO não retorna um objeto nulo.

Teste do Método createEspecialidade da Classe EspecialidadeDAO (testCreateEspecialidade):

Cria uma nova especialidade com uma descrição válida.

Chama o método createEspecialidade para adicionar a nova especialidade ao banco de dados.

Verifica se a especialidade foi cadastrada corretamente no banco de dados.

Teste do Método getEspecialidades da Classe EspecialidadeDAO (testGetEspecialidades):

Chama o método getEspecialidades para obter a lista de especialidades.

Verifica se a lista de especialidades retornada não é nula e não está vazia.

Teste do Método getEspecialidade da Classe EspecialidadeDAO (testGetEspecialidade):

Define um ID de especialidade válido.

Chama o método getEspecialidade para obter a especialidade com base no ID.

Verifica se a especialidade retornada não é nula e possui o ID esperado.

Teste do Método updateEspecialidade da Classe EspecialidadeDAO (testUpdateEspecialidade):

Define um ID de especialidade válido.

Cria uma nova especialidade com uma descrição atualizada.

Chama o método updateEspecialidade para atualizar a especialidade no banco de dados.

Obtém a especialidade atualizada do banco de dados.

Compara a descrição da especialidade atualizada com a descrição esperada.

Teste do Método deleteEspecialidade da Classe EspecialidadeDAO (testDeleteEspecialidade):

Cria uma nova especialidade para teste.
Obtém o ID da especialidade criada.
Chama o método deleteEspecialidade para excluir a especialidade do banco de dados.
Obtém a especialidade novamente do banco de dados.
Verifica se a especialidade retornada é nula, indicando que foi excluída com sucesso.

Teste do Método getIdDeleteEspecialidade da Classe EspecialidadeDAO (testGetIdDeleteEspecialidade):

Cria uma nova especialidade para teste.
Obtém o ID da especialidade criada.
Chama o método getIdDeleteEspecialidade para obter os IDs de exames e consultas associados à especialidade.
Verifica se as listas de IDs de exames e consultas não estão vazias.

Testes Utilizando Mockito:

Os testes a partir desse ponto utilizam o framework Mockito para simular comportamentos de objetos e verificar interações.

Teste do Método createEspecialidade utilizando Mockito (testCreateEspecialidadeMock):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.
Cria uma nova especialidade com uma descrição válida.
Chama o método createEspecialidade utilizando os mocks criados.
Verifica se o método execute foi chamado corretamente.

Teste do Método getEspecialidades utilizando Mockito (testGetEspecialidadesMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.
Chama o método getEspecialidades utilizando os mocks criados.
Verifica se a lista de especialidades retornada não é nula e não está vazia.

Teste do Método getEspecialidade utilizando Mockito (testGetEspecialidadeMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de um resultado.
Chama o método getEspecialidade utilizando os mocks criados.
Verifica se a especialidade retornada não é nula e possui o ID esperado.

Teste do Método updateEspecialidade utilizando Mockito (testUpdateEspecialidadeMock):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.
Cria uma nova especialidade com uma descrição atualizada e um ID válido.
Chama o método updateEspecialidade utilizando os mocks criados.
Verifica se o método execute foi chamado corretamente.

Teste do Método deleteEspecialidade utilizando Mockito (testDeleteEspecialidadeMock):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.
Chama o método deleteEspecialidade utilizando os mocks criados.

Verifica se o método execute foi chamado corretamente.

Teste do Método getIdDeleteEspecialidade utilizando Mockito (testGetIdDeleteEspecialidadeMock):

Cria mocks para a conexão com o banco de dados, o statement e os resultSets.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.

Chama o método getIdDeleteEspecialidade utilizando os mocks criados.

Verifica se as listas de IDs de exames e consultas não estão vazias.

Testes de Exceção:

Os testes a partir desse ponto verificam se as exceções são lançadas corretamente em situações de erro.

Teste do Método createEspecialidade com SQLException (testCreateEspecialidadeSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método createEspecialidade.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getEspecialidades com SQLException (testGetEspecialidadesSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método getEspecialidades.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getEspecialidade com SQLException (testGetEspecialidadeSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Cria uma especialidade com um ID válido.

Chama o método getEspecialidade.

Verifica se a SQLException foi lançada corretamente.

Teste do Método updateEspecialidade com SQLException (testUpdateEspecialidadeSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Cria uma especialidade com uma descrição atualizada e um ID válido.

Chama o método updateEspecialidade.

Verifica se a SQLException foi lançada corretamente.

Teste do Método deleteEspecialidade com SQLException (testDeleteEspecialidadeSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Cria uma especialidade com um ID válido.

Chama o método deleteEspecialidade.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getIdDeleteEspecialidade com SQLException (testGetIdDeleteEspecialidadeSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.
Cria uma especialidade com um ID válido.
Chama o método getIdDeleteEspecialidade.
Verifica se a SQLException foi lançada corretamente.

ExameDAOTest:

Teste do Construtor da Classe ExameDAO (testExameDAO):

Verifica se a instanciação da classe ExameDAO não retorna um objeto nulo.

Teste do Método createExame da Classe ExameDAO (testCreate_exame):

Cria um novo exame com um ID e descrição válidos.
Obtém a lista de exames antes de adicionar o novo exame.
Chama o método createExame para adicionar o novo exame ao banco de dados.
Obtém a lista de exames novamente e verifica se a quantidade de exames aumentou.

Teste do Método getExames da Classe ExameDAO (testGet_exames):

Chama o método getExames para obter a lista de exames.
Verifica se a lista de exames retornada não é nula.

Teste do Método getExame da Classe ExameDAO (testGet_exame):

Define um ID de exame válido.
Chama o método getExame para obter o exame com base no ID.
Verifica se o exame retornado não é nulo.

Teste do Método getExamesDaConsulta da Classe ExameDAO (testGet_examesDaConsulta):

Define um ID de consulta válido.
Cria uma lista de exames vazia.
Chama o método getExamesDaConsulta para obter a lista de exames associados à consulta.
Verifica se a lista de exames retornada não é nula.

Teste do Método updateExame da Classe ExameDAO (testUpdate_exame):

Define um ID de exame válido.
Cria um novo exame com uma descrição atualizada.
Chama o método updateExame para atualizar o exame no banco de dados.
Obtém o exame atualizado do banco de dados.
Compara a descrição do exame atualizado com a descrição esperada.

Teste do Método deleteExame da Classe ExameDAO (testDelete_exame):

Insere um novo exame no banco de dados para teste.
Obtém o ID do exame inserido.
Chama o método deleteExame para excluir o exame do banco de dados.
Obtém o exame novamente do banco de dados.
Verifica se o exame retornado é nulo, indicando que foi excluído com sucesso.

Teste do Método deleteTipoExame da Classe ExameDAO (testDelete_tipoExame):

Insere um novo exame no banco de dados para teste.
Obtém o ID do exame inserido.
Chama o método deleteTipoExame para excluir o tipo de exame do banco de dados.
Obtém o exame novamente do banco de dados.
Verifica se a descrição do exame retornado é nula, indicando que o tipo de exame foi excluído.

Teste do Método getIdDeleteExame da Classe ExameDAO (testGet_idDeleteExame):

Define um ID de exame válido.
Chama o método getIdDeleteExame para obter a lista de IDs de exames e consultas associados ao exame.
Verifica se a lista de IDs de exames não está vazia.
Testes Utilizando Mockito:
Os testes a partir desse ponto utilizam o framework Mockito para simular comportamentos de objetos e verificar interações.

Teste do Método createExame utilizando Mockito (testCreateExameMock):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.
Cria um novo exame com uma descrição válida.
Chama o método createExame utilizando os mocks criados.
Verifica se o método execute foi chamado corretamente.

Teste do Método getExames utilizando Mockito (testGetExamesMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.
Chama o método getExames utilizando os mocks criados.
Verifica se a lista de exames retornada não é nula e não está vazia.

Teste do Método getExame utilizando Mockito (testGetExameMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de um resultado.
Chama o método getExame utilizando os mocks criados.
Verifica se o exame retornado não é nulo.

Teste do Método updateExame utilizando Mockito (testUpdateExameMock):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.
Cria um novo exame com uma descrição atualizada e um ID válido.
Chama o método updateExame utilizando os mocks criados.
Verifica se o método execute foi chamado corretamente.

Teste do Método deleteExame utilizando Mockito (testDeleteExameMock):

Cria mocks para a conexão com o banco de dados e o statement.
Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.
Chama o método deleteExame utilizando os mocks criados.
Verifica se o método execute foi chamado corretamente.

Teste do Método getIdDeleteExame utilizando Mockito (testGetIdDeleteExameMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.

Chama o método getIdDeleteExame utilizando os mocks criados.

Verifica se a lista de IDs de exames não está vazia.

Teste do Método createExame com SQLException (testCreateExameSQLException):

Cria mocks para a conexão com o banco de dados e o statement.

Configura o comportamento do mock da conexão para lançar uma SQLException ao chamar o método createStatement.

Cria um novo exame com uma descrição válida.

Chama o método createExame utilizando os mocks criados.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getExames com SQLException (testGetExamesSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método getExames.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getExame com SQLException (testGetExameSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Define um ID de exame válido.

Chama o método getExame.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getExamesDaConsulta com SQLException (testGetExamesDaConsultaSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Define um ID de consulta válido.

Cria uma lista de exames vazia.

Chama o método getExamesDaConsulta.

Verifica se a SQLException foi lançada corretamente.

Teste do Método updateExame com SQLException (testUpdateExameSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Define um ID de exame válido.

Cria um novo exame com uma descrição atualizada.

Chama o método updateExame.

Verifica se a SQLException foi lançada corretamente.

Teste do Método deleteExame com SQLException (testDeleteExameSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Define um ID de exame válido.
Chama o método deleteExame.
Verifica se a SQLException foi lançada corretamente.

Teste do Método deleteTipoExame com SQLException (testDeleteTipoExameSQLException):

Cria um mock para a conexão com o banco de dados.
Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.
Define um ID de exame válido.
Chama o método deleteTipoExame.
Verifica se a SQLException foi lançada corretamente.

Teste do Método getIdDeleteExame com SQLException (testGetIdDeleteExameSQLException):

Cria um mock para a conexão com o banco de dados.
Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.
Define um ID de exame válido.
Chama o método getIdDeleteExame.
Verifica se a SQLException foi lançada corretamente.

PlanoDAOTest:

Teste do Construtor da Classe PlanoDAO (testPlanoDAO):

Verifica se a instanciação da classe PlanoDAO não retorna um objeto nulo.
Teste do Método createPlano da Classe PlanoDAO (testCreatePlano):

Cria um novo plano com uma descrição válida.
Obtém o número atual de planos antes de adicionar o novo plano.
Chama o método createPlano para adicionar o novo plano ao banco de dados.
Obtém o número atual de planos novamente e verifica se a quantidade de planos aumentou.
Teste do Método getPlano da Classe PlanoDAO (testGetPlano):

Chama o método getPlano para obter um plano com base em um ID válido.
Verifica se o plano retornado não é nulo.
Teste do Método updatePlano da Classe PlanoDAO (testUpdatePlano):

Cria um novo plano com uma descrição atualizada e um ID válido.
Chama o método updatePlano para atualizar o plano no banco de dados.
Obtém o plano atualizado do banco de dados.
Compara a descrição do plano atualizado com a descrição esperada.

Teste do Método deletePlano da Classe PlanoDAO (testDeletePlano):

Cria um novo plano no banco de dados para teste.
Obtém o ID do plano inserido.
Chama o método deletePlano para excluir o plano do banco de dados.
Obtém o plano novamente do banco de dados.
Verifica se o plano retornado é nulo, indicando que foi excluído com sucesso.

Teste do Método getIdDeletePlano da Classe PlanoDAO (testGetIdDeletePlano):

Define um ID de plano válido.
Chama o método getIdDeletePlano para obter a lista de IDs de exames e consultas associados ao plano.

Verifica se a lista de IDs de exames e consultas não está vazia.

Testes Utilizando Mockito:

Os testes a partir desse ponto utilizam o framework Mockito para simular comportamentos de objetos e verificar interações.

Teste do Método createPlano utilizando Mockito (testCreatePlanoMock):

Cria mocks para a conexão com o banco de dados e o statement.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.

Cria um novo plano com uma descrição válida.

Chama o método createPlano utilizando os mocks criados.

Verifica se o método execute foi chamado corretamente.

Teste do Método getPlanos utilizando Mockito (testGetPlanosMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.

Chama o método getPlanos utilizando os mocks criados.

Verifica se a lista de planos retornada não é nula e não está vazia.

Teste do Método getPlano utilizando Mockito (testGetPlanoMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de um resultado.

Chama o método getPlano utilizando os mocks criados.

Verifica se o plano retornado não é nulo.

Teste do Método updatePlano utilizando Mockito (testUpdatePlanoMock):

Cria mocks para a conexão com o banco de dados e o statement.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.

Cria um novo plano com uma descrição atualizada e um ID válido.

Chama o método updatePlano utilizando os mocks criados.

Verifica se o método execute foi chamado corretamente.

Teste do Método deletePlano utilizando Mockito (testDeletePlanoMock):

Cria mocks para a conexão com o banco de dados e o statement.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método execute.

Chama o método deletePlano utilizando os mocks criados.

Verifica se o método execute foi chamado corretamente.

Teste do Método getIdDeletePlano utilizando Mockito (testGetIdDeletePlanoMock):

Cria mocks para a conexão com o banco de dados, o statement e o resultSet.

Configura o comportamento dos mocks para simular a execução bem-sucedida do método executeQuery e a obtenção de resultados.

Chama o método getIdDeletePlano utilizando os mocks criados.

Verifica se a lista de IDs de exames e consultas não está vazia.

Testes de Exceção:

Os testes a partir desse ponto verificam se as exceções são lançadas corretamente em situações de erro.

Teste do Método createPlano com SQLException (testCreatePlanoSQLException):

Cria e configura mocks para a conexão com o banco de dados e o statement.

Configura o comportamento do mock da conexão para lançar uma SQLException ao chamar o método createStatement.

Cria um novo plano com uma descrição válida.

Chama o método createPlano utilizando os mocks criados.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getPlanos com SQLException (testGetPlanosSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Chama o método getPlanos.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getPlano com SQLException (testGetPlanoSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Define um ID de plano válido.

Chama o método getPlano.

Verifica se a SQLException foi lançada corretamente.

Teste do Método updatePlano com SQLException (testUpdatePlanoSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Define um ID de plano válido e um novo plano com uma descrição atualizada.

Chama o método updatePlano.

Verifica se a SQLException foi lançada corretamente.

Teste do Método deletePlano com SQLException (testDeletePlanoSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Define um ID de plano válido.

Chama o método deletePlano.

Verifica se a SQLException foi lançada corretamente.

Teste do Método getIdDeletePlano com SQLException (testGetIdDeletePlanoSQLException):

Cria um mock para a conexão com o banco de dados.

Configura o comportamento do mock para lançar uma SQLException ao chamar o método createStatement.

Define um ID de plano válido.

Chama o método getIdDeletePlano.

Verifica se a SQLException foi lançada corretamente.