UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO – UFERSA
CENTRO MULTIDISCIPLINAR DE PAU DOS FERROS – CMPF
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

TÍTULO DO PROJETO: SISTEMA DE GERENCIAMENTO DE AGENDAMENTOS PARA CLÍNICAS

NOME DOS ALUNOS: ARTHUR KELLYSON PINHEIRO DE NEGREIROS
LUIS DAVI DA SILVA SARMENTO
PEDRO MAKSON FONTES DA COSTA
NOME DO PROFESSOR: ALYSSON FILGUEIRA MILANEZ

1. INTRODUÇÃO

O "Sistema de Gerenciamento de Consultas para Clínicas" surgiu das dificuldades enfrentadas por um cliente real, que gerencia informações de sua clínica por meio de planilhas Excel.

Diante disso, via-se a necessidade de automatizar tarefas administrativas e otimizar o agendamento de consultas, serviços, realização de prescrições médicas, registros de prontuários, registros de pacientes e funcionários, como também, ter um controle mais preciso de dados financeiros.

A implementação ocorreu com o objetivo de oferecer uma solução abrangente que traga benefícios tanto para os pacientes quanto para os profissionais de saúde. Dessa forma, o sistema foi desenvolvido na linguagem Java, seguindo uma abordagem orientada a objetos e adotando o modelo MVC (Model-View-Controller).

Por fim, o projeto visa oferecer uma solução personalizada para uma administração mais eficiente da clínica, buscando simplificar processos e melhorar a qualidade de gestão dos seus serviços.

2. DETALHAMENTO DE TESTES

Para a realização dos testes foi utilizada a ferramenta JUnit, um framework para a execução automatizada de testes unitários em programas desenvolvidos na linguagem Java. A atividade dos testes no Sistema de Gerenciamento de Consultas

para Clínicas ocorreu sobre as classes do arquivo *DAO*, responsáveis pelo gerenciamento das informações com o banco de dados.

Os testes foram conduzidos para as funcionalidades de gerenciamento de funcionários, pacientes, consultas, serviços, prontuários e prescrições. Dessa forma, o JUnit serviu para verificar se as operações de acesso e manipulação dos dados estariam funcionando conforme o esperado.

A partir disso, foi aplicado o Particionamento de Equivalência e a Análise de Valor Limite como critérios de abordagem para geração de casos de teste válidos e inválidos, bem como também na análise dos limites dos dados de entrada no sistema, procurando avaliar a conformidade com as restrições estabelecidas.

2.1. TESTES FUNCIONAIS

- **Particionamento de Equivalência e Análise de Valor Limite**

Como não há uma interface gráfica para o Sistema de Gerenciamento de Consultas para Clínicas, os testes de caixa-preta também foram realizados em cima da execução do código pelo próprio terminal de forma manual.

Com isso, no que diz respeito aos testes de particionamento de equivalência e de análise de valor limite, a execução ocorreu conforme as funcionalidades pré-definidas na etapa de levantamento de requisitos, com base na documentação realizada. Em outras palavras, foram aplicados testes em cima da entrada de dados, visando analisar as classes válidas, inválidas e as extremidades do sistema.

Para fácil compreensão, podemos tomar esse exemplo real feito em cima do método:

- ***cadastrarFuncionario():***
    - Caso de testes 01:

➔ cpf: “12345678900”
➔ nome: “Pedro Makson”
➔ dataNascimento: “2002-08-23”
➔ sexo: “Masculino”
➔ cargo: “Analista de Suporte”
➔ salario: 500.00
➔ telefone: “84912345678”
➔ email: “makson@gmail.com”
➔ cep: “59980000”
➔ rua: “Rua André Leite da Costa”
➔ numero: 7
➔ bairro: “Centro”
➔ cidade: “Jose da Penha”
➔ uf: “RN”

Esse é um cenário em que o caso de teste foi executado com sucesso. As entradas repassadas estão de acordo com o que foi documentado nas especificações do sistema.

Adentrando no segundo caso de teste, é exposto apenas alguns atributos presentes no momento de cadastrar o funcionário.

- Caso de testes 02:

➔ cpf: “”, “123456789001”, “12345”, “cpf45678910”

➔ salario: ””, 500.00, “quinhentos reais”, -500.00

➔ email: “”, “makson.com”, “pedro@gmail”

➔ cep: “”, “12345”, “12345@78”

Esse é um cenário em que o caso de teste foi executado com fracasso. Visto que, as entradas de dados não obedecem às restrições estabelecidas na documentação do sistema. Ressaltando que, no momento do desenvolvimento do código foram atribuídas validações (sendo elas de tamanhos, caracteres especiais, não nulos e vazios, formatação de datas, números negativos, etc.) levando em conta o modelo lógico do banco de dados.

2.2. TESTES ESTRUTURAIS

- **Cobertura**

Dentro do conjunto de casos de teste efetuados, a análise da cobertura foi realizada no intuito de verificar quanto do sistema foi testado.

Através da cobertura obtida com os testes no JUnit, é possível visualizar a porcentagem em relação às linhas de código testadas.

A cobertura para a maioria das classes testes foi menor que 50%, o que significa afirmar que mais casos de testes poderiam ser implementados para cobrir as partes não executadas.

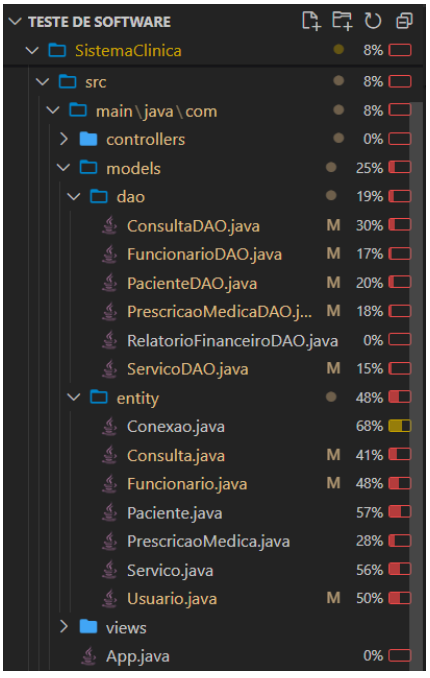

Figura 01: nível de cobertura

Na classe PacienteDAO.java, a cobertura obtida foi de 30%, o que corresponde a passagem dos casos de teste efetuados sobre os métodos inserirPaciente() e atualizarAtributoPaciente(). Abaixo, pode ser observado um exemplo do cenário de uso testado para o método inserirPaciente(). A partir do caso de teste realizado, é possível confirmar o cadastro de um paciente no banco de dados, uma vez que o retorno foi *true* no *assert* do teste.

```java
@Test
public void testCadastrarPacienteSucessoNominal() throws ParseException {

    Date dataNascimento = dateFormat.parse(source:"2002-08-23");

    paciente = new Paciente(
            cpf:"32145678946",
            nome:"Pedro Makson",
            dataNascimento,
            sexo:"Masculino",
            peso:62,
            altura:1.76,
            tipoSanguineo:"A+",
            telefone:"12345678911",
            email:"pedr485o@example.com",
            senha:"senha123",
            cep:"12345678",
            rua:"Rua dos Pacientes",
            numeroDaCasa:123,
            bairro:"Centro",
            cidade:"Cidade",
            uf:"UF");

    assertTrue(pacienteDAO.inserirPaciente(paciente));
}
```

Figura 02: método testCadastrarPacienteSucessoNominal()

2.3. TESTES DE UNIDADE E INTEGRAÇÃO

No teste de unidade, unidades individuais do código foram testadas de forma isolada para garantir que cada componente funcione conforme o esperado. Isso geralmente envolve a execução de funções ou métodos específicos e a verificação de seus resultados em relação às expectativas definidas. Os testes foram executados manualmente para garantir que cada unidade atenda aos requisitos e expectativas de qualidade.

Ademais, o teste de integração serviu para verificar se as unidades individuais de código funcionam corretamente quando combinadas e interagem umas com as outras. Durante o processo, diferentes partes do sistema são integradas e testadas em conjunto para garantir que elas se comuniquem e cooperem adequadamente.