UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO

CAMPUS PAU DOS FERROS

DISCIPLINA: TESTE DE SOFTWARE

DOCENTE: JOÃO BATISTA DE SOUZA NETO

GUSTAVO KESLEY DE FONTES NUNES

JOÃO GUSTAVO SOUZA LIMA

THYAGO FABRICIO MELO COSTA

PLANO DE TESTE

PAU DOS FERROS – RN

2025

1. INTRODUÇÃO:

Este relatório apresenta a execução de testes unitários aplicados a um sistema de fila de atendimento para um pronto-socorro. O objetivo principal desses testes é validar o correto funcionamento das funcionalidades implementadas, garantindo que o sistema opere conforme esperado e atenda aos requisitos estabelecidos. Além disso, os testes desempenham um papel fundamental na identificação precoce de falhas, contribuindo para a confiabilidade e robustez do software.

Para isso, foi elaborado um plano de testes que define uma abordagem sistemática, estruturando estratégias que asseguram a qualidade do sistema. O foco desses testes está na detecção de inconsistências, erros lógicos e possíveis comportamentos inesperados antes da implantação em um ambiente real. Dessa forma, busca-se não apenas a correção de defeitos, mas também a prevenção de problemas futuros, garantindo um software mais seguro e estável para os usuários.

Os testes unitários foram desenvolvidos utilizando a biblioteca **unittest** do Python, permitindo a automação e padronização da verificação de cada componente do código de forma isolada. Com isso, é possível avaliar a integridade dos módulos individuais, facilitar a manutenção do sistema e promover um desenvolvimento contínuo mais eficiente. Além de reduzir custos com correções tardias, essa abordagem também contribui para a escalabilidade do software, assegurando que futuras implementações possam ser integradas com menor risco de impacto sobre as funcionalidades já existentes.

2. DESENVOLVIMENTO

Para iniciar o desenvolvimento, realizamos primeiramente a etapa de testes, adotando o critério funcional como base. Para isso, utilizamos a documentação do sistema como referência principal, garantindo que os testes estivessem alinhados aos requisitos estabelecidos.

Além disso, aplicamos a técnica de particionamento do espaço de entrada para estruturar os casos de teste, assegurando uma cobertura eficiente das diferentes combinações de entradas possíveis. Essa abordagem permitiu identificar cenários representativos e validar corretamente o comportamento esperado do sistema antes de avançarmos para a implementação.

2.1 Plano de Teste

característica: Registrar Paciente

ID	Entrada	Saída esperada
CT1	Nome: João Gustavo	Registro bem - Sucedido
CT2	CPF: "12345678901"	Registro bem-sucedido
CT2 (inválido)	CPF: "123456789"	Erro: CPF inválido
CT3	Email: "joao@email.com"	Registro bem-sucedido
CT3(inválido)	Email: "joao.com"	Erro: Formato de e-mail inválido
CT4	Data: "15/08/1995"	Registro bem-sucedido
CT4 (inválido)	Data: "32/12/1995"	Erro: Formato de e-mail inválido

Referência: A1 ,A2, A3, A4, A5

Característica: Realizar Triagem

ID	Entrada	Saída esperada
CT1	risco_morte=True	Retorna Risco.VERMELHO (Prioridade Máxima)
CT2	risco_morte=False e gravidade_alta=True	Retorna Risco.Laranja
CT3 (inválido)	risco_morte=False, gravidade_alta=False,	Retorna Risco.Amarelo

	Gravidade_moderada=True	
CT4	risco_morte=False, gravidade_alta=False, gravidade_moderada=False, Gravidade_baixa=True	Retorna Risco.Verde

Referência: B1, B2, B3, B4

Característica: Gerenciamento Fila de Atendimento

ID	Entrada	Saída esperada
CT1	Fila Vazia	Erro Nenhum Paciente na fila
CT2	Fila com Pacientes	Fila processada corretamente

Referência: C1, C2

Característica: Chama o próximo da fila

ID	Entrada	Saída esperada
CT1	Paciente com faixa vermelha	Paciente chamado primeiro
CT2	Ordem correta (azul < verde < amarelo < laranja < vermelho)	Fila atendida corretamente
CT3	fila vazia	Erro: Nenhum paciente na fila

Referência: D1, D2, D3

Característica: Pesquisar histórico de Atendimento

ID	Entrada	Saída esperada
CT1	CPF Válido	Histórico retornado
CT2	CPF não cadastrado	Erro: CPF não encontrado

Referência: E1, E2

Requisitos de teste (Cobertura de bloco Base):

Bloco base: A1,B2,C2,D3,E1

RT	Requisitos Cobertos (Blocos Base)	Descrição do Teste
RT1	**A1, B2, C2, D3, E1**	Paciente registrado corretamente, triagem com gravidade alta, fila com pacientes, fila vazia ao chamar, histórico salvo.
RT2	A2, B2, C2, D2, E1	Paciente não registrado (CPF inválido), triagem com gravidade alta, fila com pacientes, chamado corretamente e histórico salvo.
RT3	A3, B3, C2, D2, E1	Paciente não registrado (Erro no e-mail), triagem com gravidade moderada, fila com pacientes, chamado corretamente.
RT4	A4, B4, C1, D3, E1	Paciente não registrado (Data inválida), triagem com gravidade baixa, fila vazia, erro ao chamar e histórico salvo.
RT5	A1, B2, C2, D1, E2	Paciente registrado corretamente, triagem com gravidade alta, fila com pacientes, chamado corretamente, mas CPF não cadastrado no histórico.

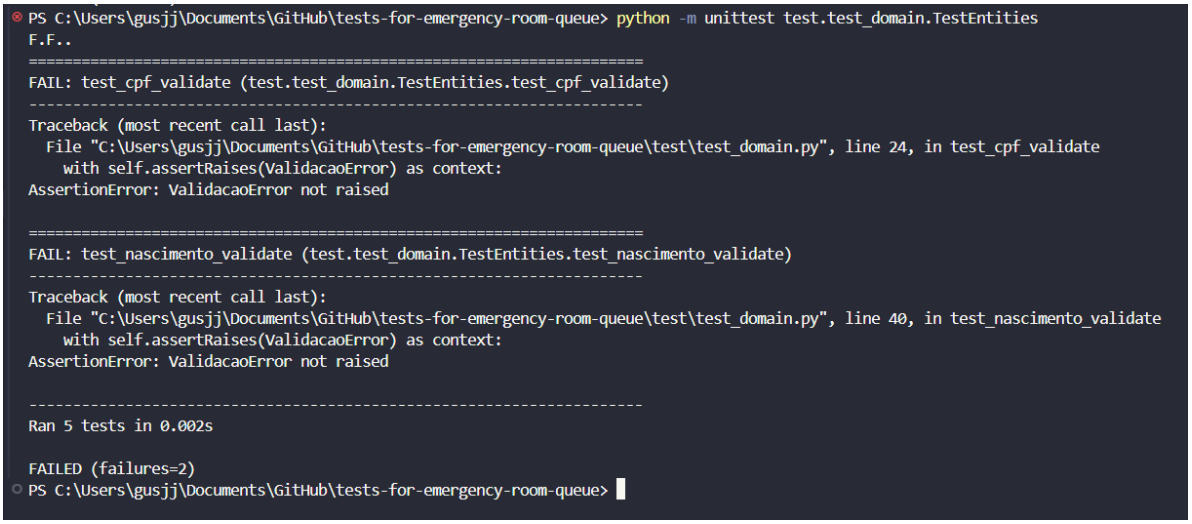
RT6	A1, B3, C1, D3, E2	Paciente registrado corretamente, triagem com gravidade moderada, fila vazia, erro ao chamar e CPF não cadastrado no histórico.
RT7	A1, B4, C2, D2, E1	Paciente registrado corretamente, triagem com gravidade baixa, fila com pacientes, chamado corretamente e histórico salvo.
RT8	A2, B3, C1, D3, E2	Paciente não registrado (CPF inválido), triagem com erro, fila vazia, erro ao chamar e CPF não cadastrado no histórico.

2.2 Automação de Teste e Execução de Testes:

Para a realização dos testes unitários, foi utilizada a biblioteca ***unittest***, que permitiu a validação das funcionalidades implementadas de acordo com os requisitos do sistema. Nesta seção, são discutidos os principais resultados obtidos, bem como a cobertura dos testes em relação aos requisitos estabelecidos, destacando possíveis falhas e melhorias identificadas durante a execução dos testes.

2.2.1 Registrar Paciente:

Testamos primeiro as funcionalidades de registro de pacientes, elaboramos testes para todos os blocos **A1, A2, A3, A4, A5.** Observamos que já existia tratamento de exceções para o método, então focamos em desenvolver teste que verifica se as exceções estão sendo realmente disparadas, usando o **'*assertRaises*'**



Anexo 1. Fonte: Própria

No teste do cpf foi analisado que o no código atual, se o CPF não tiver 11 dígitos, a exceção não será levantada, porque a função apenas verifica se o CPF contém apenas números (***isdigit()***), mas não verifica a quantidade de dígitos, assim fazendo o teste falhar.

Já o erro "***AssertionError: ValidacaoError not raised***" indica que a exceção esperada não foi levantada. Isso significa que seu método ***validar_nascimento*** está aceitando uma data inválida em algum caso que deveria gerar um erro.

O método não valida datas impossíveis

"31/02/2020" (fevereiro nunca tem 31 dias) deve gerar um erro, mas talvez não esteja.
"30/02/2021" também é inválido.

O erro só será capturado se ***datetime.strptime()*** falhar, na função da linha 26 do domain.py podemos ver um engano gerado por humano, onde n foi chamado o metodo ***validar_nascimento***, assim não sendo possivel ser disparado a execeção. Os demais testes funcionaram, mostrando que o código está cobrindo eventuais falhas na execução.

2.2.2 Realizar Triagem:

Anexo 2. Fonte: Própria

A execução dos testes unitários na classe ***TestCLI***, utilizando a biblioteca unittest, confirmou que a implementação das funcionalidades de triagem de pacientes e histórico de atendimento atende parcialmente aos requisitos estabelecidos. Foram realizados nove testes distintos, cobrindo os seguintes cenários:

Triagem dos pacientes:

- Verificação da correta classificação dos pacientes conforme a gravidade informada, garantindo que os níveis de risco (**Vermelho, Laranja, Amarelo, Verde e Azul**) sejam atribuídos corretamente com base nas respostas do paciente.
- Tratamento adequado de pacientes não cadastrados no sistema ao tentar registrar um atendimento.
- Manutenção da prioridade correta na fila de atendimento.

Histórico de Atendimento:

- Validação do retorno correto do histórico para pacientes registrados.
- Identificação de CPF inválido e tratamento de erro apropriado quando um paciente não possui histórico cadastrado.

Os testes revelaram três falhas em um total de nove testes realizados. A primeira falha ocorreu no teste ***test_buscar_historico_cpf_nao_cadastrado***, cujo objetivo é verificar se, ao

tentar buscar o histórico de um CPF não cadastrado, o sistema exibe corretamente a mensagem "Paciente não encontrado." No entanto, o erro apresentado foi um ***AssertionError,*** indicando que a chamada **print("\nPaciente não encontrado.")** não foi encontrada. Isso sugere que a mensagem de retorno não está sendo exibida corretamente, possivelmente porque o código está retornando **None** sem exibir a mensagem esperada.

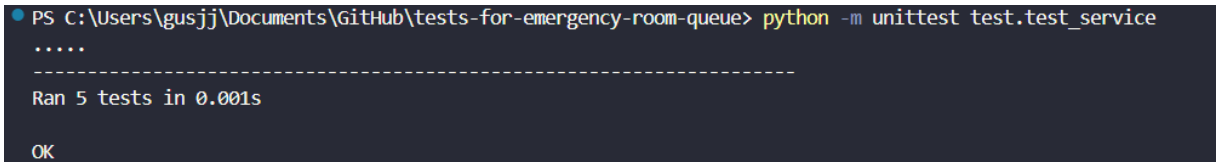
A segunda falha foi identificada no teste ***test_erro_ao_registrar_atendimento***, que tem como objetivo testar se um erro ao buscar um paciente durante o registro de atendimento é tratado corretamente, exibindo a mensagem "Erro ao registrar atendimento: Erro ao buscar paciente." O erro apresentado foi novamente um ***AssertionError***, informando que a chamada **print("\nErro ao registrar atendimento: Erro ao buscar paciente.")** não foi encontrada. Isso indica que, embora a exceção tenha sido levantada, a mensagem esperada não foi impressa, possivelmente porque a exceção está sendo silenciada ou retornando um texto diferente.

Por fim, a terceira falha ocorreu no teste ***test_paciente_nao_cadastrado***, que verifica se, ao tentar registrar um atendimento para um paciente não cadastrado, o sistema exibe a mensagem "Paciente não encontrado." O erro apresentado foi o mesmo***: AssertionError***, com a ausência da chamada **print("\nPaciente não encontrado.").** Isso indica que a execução dessa etapa não disparou a mensagem esperada e, consequentemente, não interrompeu o fluxo de execução quando o paciente não foi encontrado.

Então, no que se refere a esse arquivo de teste da classe TestCLI, que contém os testes unitários para validar os requisitos de teste das funcionalidades de triagem dos pacientes e de histórico do atendimento, temos o seguinte relatório de conclusão.

TESTE	ERRO IDENTIFICADO	SOLUÇÃO PROPOSTA
test_buscar_historico_cpf_nao_cadastrado	Mensagem de erro esperada não foi exibida	Garantir que o print("\nPaciente não encontrado.") seja chamado corretamente no código.
test_erro_ao_registrar_atendimento	Exceção PSBaseError não gerou a saída correta	Capturar corretamente a exceção e imprimir a mensagem esperada.
test_paciente_nao_cadastrado	O sistema não parou corretamente ao não encontrar paciente	Adicionar uma verificação para interromper a execução quando o paciente não for encontrado.

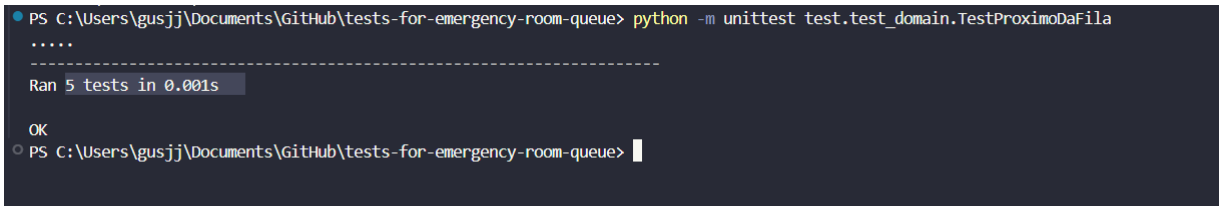
2.2.3 Gerenciamento Fila de Atendimento:



Anexo 3. Fonte: Própria

O principal objetivo desses testes é garantir que a estrutura de dados utilizada para armazenar e processar a fila de atendimento funcione corretamente, respeitando a ordem de prioridade definida pelos níveis de risco dos pacientes. A classe '***TestFilaAtendimento***' implementa três testes principais para validar o funcionamento da fila de atendimento.

Cada teste avalia um comportamento específico do sistema, garantindo que os casos de uso principais sejam cobertos.

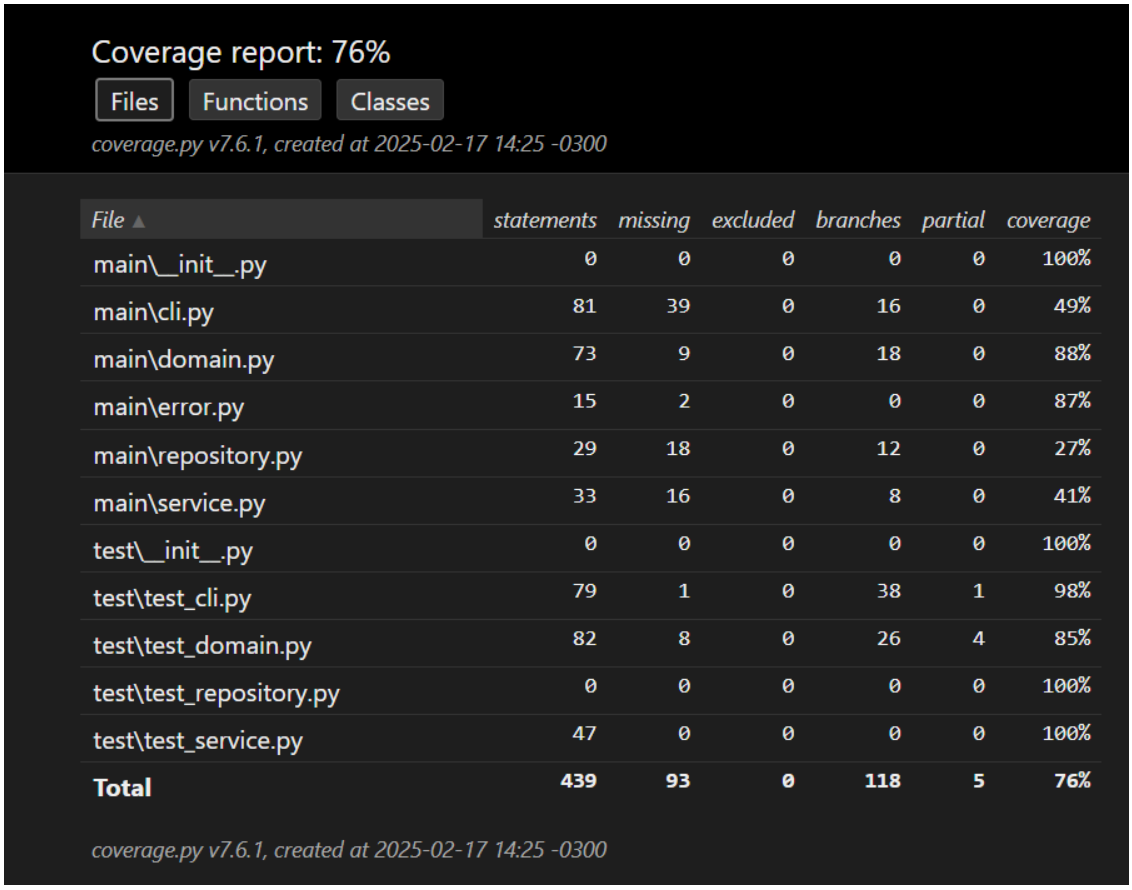


Anexo 4. Fonte: Própria

A execução dos testes unitários na classe ***TestProximoDaFila***, utilizando a biblioteca **unittest**, confirmou que todas as funcionalidades testadas na FilaAtendimento estão operando conforme esperado. Foram realizados cinco testes distintos, abrangendo cenários como a verificação da exceção **FilaVaziaErro**r ao chamar **proximo()** em uma fila vazia, a validação de erro ao tentar registrar um paciente com e-mail inválido, a correta priorização de pacientes conforme o nível de gravidade, a remoção de pacientes com alta prioridade sem que seu CPF fosse registrado no histórico e a garantia de que um paciente de gravidade extrema fosse atendido, deixando a fila vazia e sendo armazenado no histórico.

O resultado exibiu OK, indicando que todos os testes passaram com sucesso, validando que a implementação cumpre os requisitos RT1, RT3 e RT5 Isso assegura que a lógica de atendimento está correta, garantindo que o sistema lida adequadamente com os casos testados.

2.2.4 Cobertura:



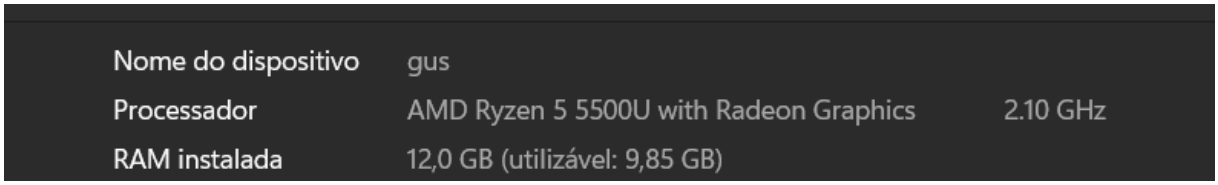
File ▲	statements	missing	excluded	branches	partial	coverage
main__init__.py	0	0	0	0	0	100%
main\cli.py	81	39	0	16	0	49%
main\domain.py	73	9	0	18	0	88%
main\error.py	15	2	0	0	0	87%
main\repository.py	29	18	0	12	0	27%
main\service.py	33	16	0	8	0	41%
test__init__.py	0	0	0	0	0	100%
test\test_cli.py	79	1	0	38	1	98%
test\test_domain.py	82	8	0	26	4	85%
test\test_repository.py	0	0	0	0	0	100%
test\test_service.py	47	0	0	0	0	100%
Total	**439**	**93**	**0**	**118**	**5**	**76%**

Anexo 5. Fonte: Própria

O relatório de cobertura de testes indica que a suíte de testes implementada já cobre **76%** do código do projeto, mostrando um bom nível de validação do sistema.

2.3 Configurações da máquina:

Os testes foram realizados nessa máquina com essas seguintes configurações:



2.4 Repositório:

O repositório com todos os teste e automatização deles.

https://github.com/gusjjpv/tests-for-emergency-room-queue

3. Conclusão:

A realização dos testes unitários no sistema de fila de atendimento para um pronto-socorro demonstrou a importância da validação contínua do software, garantindo que as funcionalidades implementadas atendam aos requisitos especificados. Através do uso da biblioteca unittest do Python, foi possível identificar falhas e aprimorar a robustez do código, assegurando um funcionamento mais estável e confiável.

Os testes abordaram aspectos essenciais do sistema, como o registro de pacientes, a triagem, o gerenciamento da fila de atendimento e a recuperação do histórico de atendimentos. Durante a execução, foram detectadas inconsistências em alguns módulos, como a validação de CPF, o tratamento de exceções na triagem e a exibição de mensagens de erro. Essas falhas foram analisadas detalhadamente, permitindo a proposição de soluções para corrigir e aprimorar o comportamento do sistema.

Além disso, a cobertura de testes alcançada foi de 76%, um indicador positivo de que a maioria das funcionalidades críticas foi testada. Contudo, há espaço para melhorias, especialmente na ampliação dos testes para cobrir cenários mais complexos e aprimorar a detecção de possíveis falhas antes da implantação em ambiente real.

Dessa forma, o processo de testes contribuiu significativamente para a qualidade do software, reduzindo riscos de falhas e proporcionando maior segurança para os usuários. A continuidade da prática de testes e a implementação de novas estratégias, como testes automatizados mais abrangentes, serão fundamentais para manter a confiabilidade e a escalabilidade do sistema no futuro.