



INSTITUTO FEDERAL GOIANO – CAMPUS CERES
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

DINOSTAT: IMPLEMENTAÇÃO DE TESTES UNITÁRIOS

CERES – GO
2024

**DAIANNY EVILLIN COSTA DE OLIVEIRA
EMANUEL GONÇALVES MENEZES
GEOVANA SILVA MATUZINHO
LUCAS SANTOS CARVALHO
MAYKO DIOUZEF MENDES DO AMARAL**

DINOSTAT: IMPLEMENTAÇÃO DE TESTES UNITÁRIOS

Trabalho apresentado à disciplina Teste de Software do curso de Bacharelado de sistemas de informação para obtenção de nota parcial.

Orientador(a): Prof. Isaac Mendes de Melo.

SUMÁRIO

1 DOCUMENTAÇÃO DO PROJETO	1
1.1. <i>ESCOPO</i>	1
1.2. <i>REQUISITOS DE SEGURANÇA E PRIVACIDADE</i>	2
2 LINGUAGEM E INFRAESTRUTURA.....	2
3 DER.....	3
4 PLANO DE TESTES	3
5 TESTE UNITÁRIO - CRIATURA	4
1. Teste de Criação de Criatura	4
2. Teste de Busca de Criatura.....	5
3. Teste de Atualização de Criatura.....	5
4. Teste de Deleção de Criatura	6
Resultados da Execução	6
Conclusão	7
6 TESTE UNITÁRIO - ESPÉCIE	7
1. Teste de Criação de Espécie.....	7
2. Teste de Busca de Espécie	7
3. Teste de Atualização de Espécie	8
4. Teste de Deleção de Espécie.....	8
Resultados da Execução	9
Conclusão	9
7 TESTE UNITÁRIO - TOKEN	10
1. Teste de Criação de Token	10
2. Teste de Busca de Token.....	10
3. Teste de Atualização de Token.....	11
4. Teste de Deleção de Token.....	11
Resultados da Execução	12
Conclusão	12
8 TESTE UNITÁRIO - TRIBO.....	12
1. Teste de Criação de Tribo	12
2. Teste de Busca de Tribo	13
3. Teste de Atualização de Tribo	13
4. Teste de Deleção de Tribo	14
Resultados da Execução	14
Conclusão	15
9 TESTE UNITÁRIO - USUÁRIO	15
1. Teste de Criação de Usuário	15

2. Teste de Busca de Usuário.....	15
3. Teste de Atualização de Usuário.....	16
4. Teste de Deleção de Usuário	16
Resultados da Execução	17
Conclusão	17
 10 TESTE UNITÁRIO - VISITANTE	 18
1. Teste de Criação de Visitante	18
2. Teste de Busca de Visitante	18
3. Teste de Atualização de Visitante	19
4. Teste de Deleção de Visitante	19
Resultados da Execução	20
Conclusão	20
 11 CONSIDERAÇÕES FINAIS.....	 21

1 DOCUMENTAÇÃO DO PROJETO

O Bot de Gerenciamento de Criaturas foi idealizado para atender às necessidades dos jogadores do jogo ARK: Survival Ascended, que frequentemente enfrentam desafios para manter as estatísticas de suas criaturas e tribos organizadas. Ao centralizar e facilitar o acesso a essas informações no Discord, o bot visa otimizar a experiência dos jogadores, eliminando a dependência de métodos manuais e dispersos, como planilhas e blocos de notas. Além de fornecer funcionalidades práticas e intuitivas, o projeto prioriza segurança, privacidade e conformidade com legislações como GDPR e LGPD.

1.1. ESCOPO

O bot oferece uma solução organizada e segura para registro e consulta das estatísticas das criaturas e tribos, com dados inseridos manualmente pelos usuários. Ele substitui métodos convencionais por um sistema automatizado no Discord, tornando a gestão de informações mais prática e acessível.

Objetivos principais:

- Fornecer um meio centralizado para registro e consulta das estatísticas de criaturas.
- Promover acessibilidade e organização no gerenciamento de dados de tribos e criaturas.
- Assegurar a segurança e privacidade dos dados dos usuários.

Funcionalidades principais:

1. Termos de Serviço e Política de Privacidade:
 - Exibição e solicitação de aceite antes do uso do bot.
 - Links e botões para consulta às políticas, promovendo transparência e conformidade com as normas.
2. Ajuda ao Usuário:
 - Comando /ajuda com informações sobre comandos disponíveis, recursos e escolha de idioma.
 - Consulta fácil aos Termos de Serviço e Política de Privacidade.
 - Ferramentas para o usuário gerenciar seus dados (como baixar informações ou solicitar exclusão, garantindo o direito ao esquecimento).
3. Gestão de Tribos:
 - Comando /tribo para criar, editar, excluir, transferir propriedade e listar informações.
 - Gerenciamento de membros com diferentes níveis de acesso e permissões.
4. Gestão de Criaturas:
 - Comando /criatura para inserir, editar, excluir e consultar as estatísticas das criaturas.
5. Segurança e Privacidade:
 - Proteção dos dados por meio de criptografia durante a transmissão e o armazenamento.
 - Estrita conformidade com legislações de privacidade.

1.2. REQUISITOS DE SEGURANÇA E PRIVACIDADE

- Criptografia em trânsito: Certificado SSL/TLS para comunicação segura entre o bot e o banco de dados.
- Criptografia em repouso: Proteção dos dados armazenados com chave configurável.
- Conformidade com legislações de privacidade: Respeito ao GDPR, LGPD, CCPA e CPRA, com divulgação clara de termos e políticas.

2 LINGUAGEM E INFRAESTRUTURA

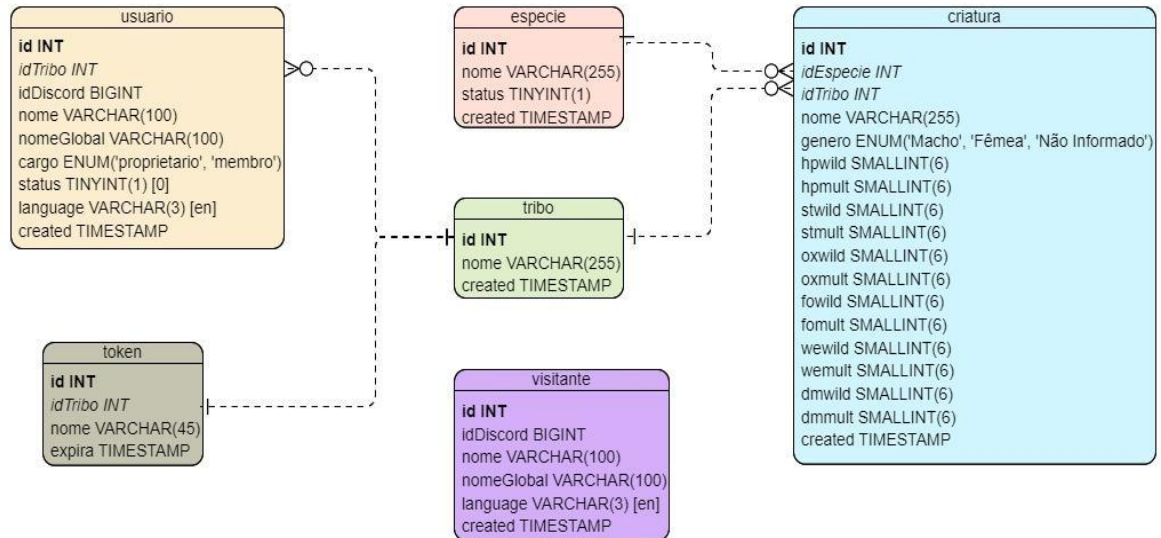
Infraestrutura:

- Banco de Dados: MySQL com conexão segura.
- Linguagem de Programação: Python, utilizando bibliotecas como discord.py e SQLAlchemy para gerenciamento do ORM.

Testes Automatizados:

- Biblioteca PyTest: O framework PyTest foi escolhido para criar e executar testes automatizados, garantindo a qualidade do código do bot DinoStats. Essa ferramenta auxilia na identificação e correção de bugs, além de validar funcionalidades e assegurar o desempenho robusto do bot.

3 DER



4 PLANO DE TESTES

Usuário

- id → INT
- idTribu → INT
- idDiscord → INT
- nome → VARCHAR
- nomeGlobal → VARCHAR
- cango → ENUM
- status → INT
- language → VARCHAR

Ficha

- id → INT
- idTribu → INT
- nome → VARCHAR

Espécie

- id → INT
- nome → VARCHAR
- status → INT

Tribo

- id → INT
- nome → VARCHAR

Visitante

- id → INT
- idDiscord → INT
- nome → VARCHAR

- nomeGlobal→ VARCHAR
- language→ VARCHAR

Criatura

- idEspecie→ INT
- idTribo→ INT
- nome→ VARCHAR
- genero→ ENUM
- hpwild→ INT
- hpmult→ INT
- stwild→ INT
- stmult→ INT
- oxwild→ INT
- oxmult→ INT
- fowild→ INT
- fomult→ INT
- wewild→ INT
- wemult→ INT
- dmwild→ INT
- dmmult→ INT
- created→ INT

5 TESTE UNITÁRIO - CRIATURA

1. Teste de Criação de Criatura

Arquivo: *teste_criatura.py*

Função: *testeCriarCriatura*

Objetivo: Verificar a funcionalidade de criação de uma nova criatura.

Descrição:

- O teste cria uma nova tribo e espécie utilizando os respectivos gerenciadores.
- Em seguida, utiliza o *CriaturaGerenciador* para criar uma criatura com os parâmetros especificados (nome, gênero, atributos de estatísticas como HP, stamina, etc.).
- O teste verifica se a criação retorna um valor não nulo, indicando sucesso.

Código:


```

1 def testeCriarCriatura(criaturaGerenciador, triboGerenciador, especieGerenciador):
2     """Testa a criação de uma criatura."""
3     tribo = triboGerenciador.criar("Arklândia")
4     especie = especieGerenciador.criar("Rex", 1)
5     sucesso = criaturaGerenciador.criar(especie.id, tribo.id, "Banguelo", "Macho", 14, 74, 78, 87, 54, 45, 12, 21, 36, 63, 1, 2)
6     assert sucesso is not None
7

```

2. Teste de Busca de Criatura

Arquivo: *teste_criatura.py*

Função: *testeBuscarCriatura*

Objetivo: Validar a busca de uma criatura existente no banco de dados.

Descrição:

- A função utiliza o *CriaturaGerenciador* para buscar uma criatura com um *ID* específico.
- O teste verifica se a busca retorna um valor não nulo, indicando que a criatura foi encontrada.

Código:

```

9 def testeBuscarCriatura(criaturaGerenciador):
10     """Testa a busca de uma criatura existente."""
11     criatura = criaturaGerenciador.buscar(1)
12     assert criatura is not None

```

3. Teste de Atualização de Criatura

Arquivo: *teste_criatura.py*

Função: *testeAtualizarCriatura*

Objetivo: Confirmar que é possível atualizar os dados de uma criatura existente.

Descrição:

- A função atualiza as informações da criatura (nome, atributos) utilizando o método *atualizar* do *CriaturaGerenciador*.
- Em seguida, realiza uma nova busca para confirmar se o nome foi modificado corretamente.
- O teste verifica se a atualização foi bem-sucedida e se o nome da criatura foi alterado.

Código:

```
15 def testeAtualizarCriatura(criaturaGerenciador):
16     """Testa a atualização de uma criatura existente."""
17     sucesso = criaturaGerenciador.atualizar(1, 1, 1, "Dentuço", 85, 58, 74, 47, 96, 36, 25, 52, 41, 14, 98, 89)
18     assert sucesso is not None
19     criatura = criaturaGerenciador.buscar(1)
20     assert criatura.nome == "Dentuço"
```

4. Teste de Deleção de Criatura

Arquivo: *teste_criatura.py*

Função: *testeDeletarCriatura*

Objetivo: Testar a remoção de uma criatura existente no banco de dados.

Descrição:

- A função utiliza o método *deletar* do *CriaturaGerenciador* para remover uma criatura específica.
- Em seguida, tenta realizar uma busca pela criatura deletada.
- O teste verifica se a deleção foi bem-sucedida.

Código:

```
23 def testeDeletarCriatura(criaturaGerenciador):
24     """Testa a exclusão de uma criatura existente."""
25     sucesso = criaturaGerenciador.deletar(1)
26     assert sucesso is not None
27     criatura = criaturaGerenciador.buscar(1)
28     assert criatura is not None
```

Resultados da Execução

Comando utilizado:

```
pytest -v teste_criatura.py
```

Saída no terminal:

```
PROBLEMAS SAÍDA CONSOLE DE DEPURACÃO TERMINAL PORTAS
(venv) PS C:\TesteUnitario> pytest -v teste_criatura.py
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.3.4, pluggy-1.5.0 -- C:\TesteUnitario\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\TesteUnitario
collected 4 items

teste_criatura.py::testeCriarCriatura PASSED [ 25%]
teste_criatura.py::testeBuscarCriatura PASSED [ 50%]
teste_criatura.py::testeAtualizarCriatura PASSED [ 75%]
teste_criatura.py::testeDeletarCriatura PASSED [100%]

===== 4 passed in 0.19s =====
```

Conclusão

Os testes unitários para a classe *CriaturaGerenciador* validam os principais métodos: criação, busca, atualização e deleção. Todos os testes passaram com sucesso, demonstrando que as funcionalidades implementadas estão operando conforme o esperado.

Essa abordagem garante maior qualidade no código e facilita a identificação de possíveis falhas durante o desenvolvimento do sistema.

6 TESTE UNITÁRIO - ESPÉCIE

1. Teste de Criação de Espécie

Arquivo: *teste_especie.py*

Função: *testeCriarEspecie*

Objetivo: Verificar a funcionalidade de criação de uma nova espécie.

Descrição:

- Utiliza o método *criar* do *EspecieGerenciador* para adicionar uma nova espécie ao banco de dados com os parâmetros fornecidos (nome e status).
- O teste valida se a operação retorna um valor não nulo, indicando sucesso na criação.

Código:

```
1 def testeCriarEspecie(especieGerenciador):
2     """Testa a criação de uma especie."""
3     sucesso = especieGerenciador.criar("Rex", 1)
4     assert sucesso is not None
```

2. Teste de Busca de Espécie

Arquivo: *teste_especie.py*

Função: *testeBuscarEspecie*

Objetivo: Testar a busca de uma espécie existente no banco de dados.

Descrição:

- Utiliza o método *buscar* do *EspecieGerenciador* para localizar uma espécie específica com base no ID.
- O teste valida se a espécie retornada é não nula, indicando que ela foi encontrada.

Código:

```
7 def testeBuscarEspecie(especieGerenciador):
8     """Testa a busca de uma especie existente."""
9     especie = especieGerenciador.buscar(1)
10    assert especie is not None
```

3. Teste de Atualização de Espécie

Arquivo: *teste_especie.py*

Função: *testeAtualizarEspecie*

Objetivo: Confirmar a capacidade de atualizar os dados de uma espécie existente.

Descrição:

- Utiliza o método *atualizar* do *EspecieGerenciador* para alterar o nome de uma espécie com um ID específico.
- Em seguida, realiza uma busca para verificar se o nome foi atualizado corretamente.
- O teste valida se a atualização foi bem-sucedida e se o campo nome reflete a nova alteração.

Código:

```
13 def testeAtualizarEspecie(especieGerenciador):
14     """Testa a atualização de uma especie existente."""
15     sucesso = especieGerenciador.atualizar(1, "Dodô")
16     assert sucesso is not None
17     especie = especieGerenciador.buscar(1)
18     assert especie.nome == "Dodô"
```

4. Teste de Deleção de Espécie

Arquivo: *teste_especie.py*

Função: *testeDeleteEspecie*

Objetivo: Validar a funcionalidade de remoção de uma espécie existente no banco de dados.

Descrição:

- Utiliza o método *deletar* do *EspecieGerenciador* para excluir uma espécie com um ID específico.
- Após a deleção, realiza uma tentativa de busca pela espécie removida.
- O teste verifica se a deleção foi realizada com sucesso.

Código:

```
21 def testeDeleteEspecie(especieGerenciador):
22     """Testa a exclusão de uma especie existente."""
23     sucesso = especieGerenciador.deletar(1)
24     assert sucesso is not None
25     especie = especieGerenciador.buscar(1)
26     assert especie is not None
```

Resultados da Execução

Comando utilizado:

```
pytest -v teste_especie.py
```

Saída no terminal:

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS
(venv) PS C:\TesteUnitario> pytest -v teste_especie.py
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.3.4, pluggy-1.5.0 -- C:\TesteUnitario\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\TesteUnitario
collected 4 items

teste_especie.py::testeCriarTribo PASSED [ 25%]
teste_especie.py::testeBuscarTribo PASSED [ 50%]
teste_especie.py::testeAtualizarTribo PASSED [ 75%]
teste_especie.py::testeDeleteTribo PASSED [100%]

===== 4 passed in 0.15s =====
```

Conclusão

Os testes unitários verificaram com sucesso as operações CRUD (Create, Read, Update, Delete) relacionadas à entidade **Espécie**. Todas as funcionalidades implementadas na classe

EspecieGerenciador passaram nos testes, garantindo que o comportamento esperado foi alcançado e que a lógica do código está correta e estável.

7 TESTE UNITÁRIO - TOKEN

1. Teste de Criação de Token

Arquivo: *teste_token.py*

Função: *testeCriarToken*

Objetivo: Verificar a funcionalidade de criação de um novo token.

Descrição:

- Utiliza o método *criar* do *TokenGerenciador* para adicionar um novo token ao banco de dados.
- O teste depende da criação de uma tribo, necessária para referenciar o *idTribo*.
- Valida se o retorno não é nulo, indicando sucesso na criação.

Código:

```
1 def testeCriarToken(tokenGerenciador, triboGerenciador):
2     """Testa a criação de um token."""
3     tribo = triboGerenciador.criar("Arklândia")
4     sucesso = tokenGerenciador.criar(tribo.id, "qwert")
5     assert sucesso is not None
```

2. Teste de Busca de Token

Arquivo: *teste_token.py*

Função: *testeBuscarToken*

Objetivo: Testar a busca de um token existente no banco de dados.

Descrição:

- Utiliza o método *buscar* do *TokenGerenciador* para localizar um token específico por seu ID.
- Verifica se o retorno não é nulo, confirmando que o token foi encontrado.

Código:

```
8 def testeBuscarToken(tokenGerenciador):
9     """Testa a busca de um token existente."""
10    token = tokenGerenciador.buscar(1)
11    assert token is not None
```

3. Teste de Atualização de Token

Arquivo: *teste_token.py*

Função: *testeAtualizarToken*

Objetivo: Validar a atualização do valor de um token existente.

Descrição:

- Utiliza o método *atualizar* do *TokenGerenciador* para modificar o valor do token.
- Em seguida, realiza uma busca para verificar se o valor foi atualizado corretamente.
- Valida se o campo *token* reflete a nova alteração.

Código:

```
14 def testeAtualizarToken(tokenGerenciador):
15     """Testa a atualização de um token existente."""
16     sucesso = tokenGerenciador.atualizar(1, "asdfg")
17     assert sucesso is not None
18     token = tokenGerenciador.buscar(1)
19     assert token.token == "asdfg"
```

4. Teste de Deleção de Token

Arquivo: *teste_token.py*

Função: *testeDeleteToken*

Objetivo: Testar a funcionalidade de remoção de um token existente.

Descrição:

- Utiliza o método *deletar* do *TokenGerenciador* para remover um token específico pelo ID.
- Após a deleção, tenta buscar o token para verificar se ele foi removido com sucesso.
- Valida se a deleção ocorreu corretamente.

Código:

```

22 def testeDeleteToken(tokenGerenciador):
23     """Testa a exclusão de um token existente."""
24     sucesso = tokenGerenciador.deletar(1)
25     assert sucesso is not None
26     token = tokenGerenciador.buscar(1)
27     assert token is not None

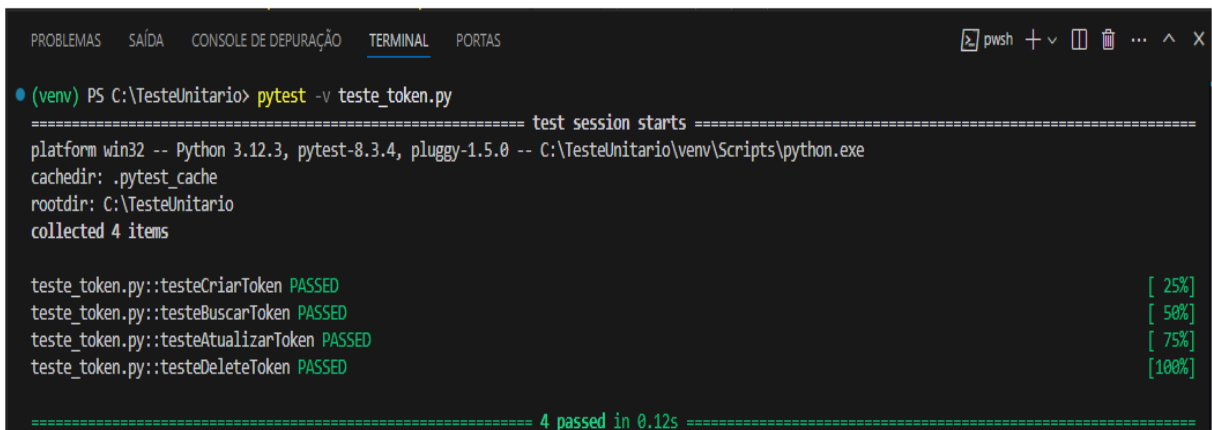
```

Resultados da Execução

Comando utilizado:

```
pytest -v teste_token.py
```

Saída no terminal:



```

PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  PORTAS
(venv) PS C:\TesteUnitario> pytest -v teste_token.py
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.3.4, pluggy-1.5.0 -- C:\TesteUnitario\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\TesteUnitario
collected 4 items

teste_token.py::testeCriarToken PASSED [ 25%]
teste_token.py::testeBuscarToken PASSED [ 50%]
teste_token.py::testeAtualizarToken PASSED [ 75%]
teste_token.py::testeDeleteToken PASSED [100%]

===== 4 passed in 0.12s =====

```

Conclusão

Os testes unitários verificaram com sucesso as operações **CRUD** (Create, Read, Update, Delete) relacionadas à entidade **Token**. Todos os métodos da classe *TokenGerenciador* passaram nos testes, garantindo o correto funcionamento e estabilidade das funcionalidades implementadas.

8 TESTE UNITÁRIO - TRIBO

1. Teste de Criação de Tribo

Arquivo: *teste_tribo.py*

Função: *testeCriarTribo*

Objetivo: Verificar a funcionalidade de criação de uma nova tribo.

Descrição:

- Utiliza o método *criar* do *TriboGerenciador* para adicionar uma nova tribo com o nome fornecido.
- O teste valida se o retorno não é nulo, indicando sucesso na criação da tribo.

Código:

```
1 def testeCriarTribo(triboGerenciador):
2     """Testa a criação de uma tribo."""
3     sucesso = triboGerenciador.criar("Arklândia")
4     assert sucesso is not None
```

2. Teste de Busca de Tribo

Arquivo: *teste_tribo.py*

Função: *testeBuscarTribo*

Objetivo: Testar a busca de uma tribo existente no banco de dados.

Descrição:

- Utiliza o método *buscar* do *TriboGerenciador* para localizar uma tribo com base em seu ID.
- O teste verifica se o retorno não é nulo, indicando que a tribo foi encontrada com sucesso.

Código:

```
7 def testeBuscarTribo(triboGerenciador):
8     """Testa a busca de uma tribo existente."""
9     tribo = triboGerenciador.buscar(1)
10    assert tribo is not None
```

3. Teste de Atualização de Tribo

Arquivo: *teste_tribo.py*

Função: *testeAtualizarTribo*

Objetivo: Validar a atualização do nome de uma tribo existente.

Descrição:

- Utiliza o método *atualizar* do *TriboGerenciador* para modificar o nome de uma tribo com um ID específico.

- Em seguida, realiza uma busca para confirmar se o campo nome foi atualizado corretamente.
- O teste valida se o valor do nome reflete a nova alteração.

Código:

```
13 def testeAtualizarTribo(triboGerenciador):
14     """Testa a atualização de uma tribo existente."""
15     sucesso = triboGerenciador.atualizar(1, "G.A.B")
16     assert sucesso is not None
17     tribo = triboGerenciador.buscar(1)
18     assert tribo.nome == "G.A.B"
```

4. Teste de Deleção de Tribo

Arquivo: *teste_tribo.py*

Função: *testeDeleteTribo*

Objetivo: Testar a remoção de uma tribo existente no banco de dados.

Descrição:

- Utiliza o método *deletar* do *TriboGerenciador* para remover uma tribo pelo ID.
- Após a deleção, tenta buscar a tribo para confirmar que ela foi removida com sucesso.
- O teste valida se a remoção ocorreu corretamente.

Código:

```
21 def testeDeleteTribo(triboGerenciador):
22     """Testa a exclusão de uma tribo existente."""
23     sucesso = triboGerenciador.deletar(1)
24     assert sucesso is not None
25     tribo = triboGerenciador.buscar(1)
26     assert tribo is not None
```

Resultados da Execução

Comando utilizado:

```
pytest -v teste_tribo.py
```

Saída no terminal:

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS
(venv) PS C:\TesteUnitario> pytest -v teste_tribo.py
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.3.4, pluggy-1.5.0 -- C:\TesteUnitario\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\TesteUnitario
collected 4 items

teste_tribo.py::testeCriarTribo PASSED [ 25%]
teste_tribo.py::testeBuscarTribo PASSED [ 50%]
teste_tribo.py::testeAtualizarTribo PASSED [ 75%]
teste_tribo.py::testeDeleteTribo PASSED [100%]

===== 4 passed in 0.13s =====
```

Conclusão

Os testes unitários validaram com sucesso as operações **CRUD** (Create, Read, Update, Delete) para a entidade **Tribo**. Todas as funcionalidades implementadas na classe *TriboGerenciador* foram testadas e aprovadas, garantindo estabilidade e confiabilidade do código.

9 TESTE UNITÁRIO - USUÁRIO

1. Teste de Criação de Usuário

Arquivo: *teste_usuario.py*

Função: *testeCriarUsuario*

Objetivo: Verificar a funcionalidade de criação de um novo usuário.

Descrição:

- Utiliza o método *criar* do *UsuarioGerenciador* para adicionar um novo usuário com atributos específicos, como *idDiscord*, *nome*, *nomeGlobal*, *cargo*, etc.
- O teste depende da criação de uma tribo para associar o *idTribo*.
- Valida se o retorno não é nulo, indicando sucesso na criação do usuário.

Código:

```
1 def testeCriarUsuario(usuarioGerenciador, triboGerenciador):
2     """Testa a criação de um usuário."""
3     tribo = triboGerenciador.criar("Arklândia")
4     sucesso = usuarioGerenciador.criar(tribo.id, "123", "isaac", "Isaac Mendes", "proprietario", 1, "pt-BR")
5     assert sucesso is not None
```

2. Teste de Busca de Usuário

Arquivo: *teste_usuario.py*

Função: *testeBuscarUsuario*

Objetivo: Testar a busca de um usuário existente no banco de dados.

Descrição:

- Utiliza o método *buscar* do *UsuarioGerenciador* para localizar um usuário com base no *idDiscord*.
- Verifica se o retorno não é nulo, indicando que o usuário foi encontrado.

Código:

```
8 def testeBuscarUsuario(usuarioGerenciador):
9     """Testa a busca de um usuário existente."""
10    usuario = usuarioGerenciador.buscar(123)
11    assert usuario is not None
```

3. Teste de Atualização de Usuário

Arquivo: *teste_usuario.py*

Função: *testeAtualizarUsuario*

Objetivo: Validar a atualização dos dados de um usuário existente.

Descrição:

- Utiliza o método *atualizar* do *UsuarioGerenciador* para modificar os campos *nome* e *nomeGlobal* de um usuário com base no *idDiscord*.
- Em seguida, realiza uma busca para verificar se os campos foram atualizados corretamente.
- Valida se o retorno é consistente com as alterações realizadas.

Código:

```
14 def testeAtualizarUsuario(usuarioGerenciador):
15     """Testa a atualização de um usuário existente."""
16     sucesso = usuarioGerenciador.atualizar(123, "isaacmendes", "Isaac")
17     assert sucesso is not None
18     usuario = usuarioGerenciador.buscar(123)
19     assert usuario.nome == "isaacmendes" and usuario.nomeGlobal == "Isaac"
```

4. Teste de Deleção de Usuário

Arquivo: *teste_usuario.py*

Função: *testeDeletarUsuario*

Objetivo: Testar a remoção de um usuário existente no banco de dados.

Descrição:

- Utiliza o método *deletar* do *UsuarioGerenciador* para excluir um usuário com base no *idDiscord*.
- Após a remoção, tenta buscar o usuário para confirmar que ele foi deletado com sucesso.
- Valida se a exclusão foi realizada corretamente.

Código:

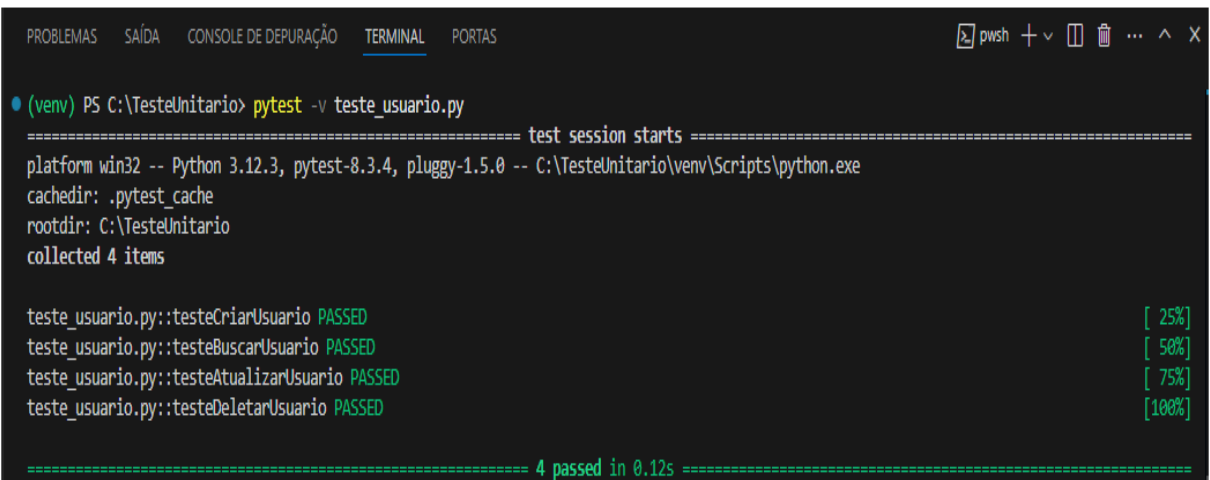
```
22 def testeDeletarUsuario(usuarioGerenciador):
23     """Testa a exclusão de um usuário existente."""
24     sucesso = usuarioGerenciador.deletar(123)
25     assert sucesso is not None
26     usuario = usuarioGerenciador.buscar(123)
27     assert usuario is not None
```

Resultados da Execução

Comando utilizado:

```
pytest -v teste_usuario.py
```

Saída no terminal:



```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  PORTAS
(venv) PS C:\TesteUnitario> pytest -v teste_usuario.py
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.3.4, pluggy-1.5.0 -- C:\TesteUnitario\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\TesteUnitario
collected 4 items

teste_usuario.py::testeCriarUsuario PASSED [ 25%]
teste_usuario.py::testeBuscarUsuario PASSED [ 50%]
teste_usuario.py::testeAtualizarUsuario PASSED [ 75%]
teste_usuario.py::testeDeletarUsuario PASSED [100%]

===== 4 passed in 0.12s =====
```

Conclusão

Os testes unitários validaram com sucesso as operações **CRUD** (Create, Read, Update, Delete) relacionadas à entidade **Usuário**. As funcionalidades implementadas na classe *UsuarioGerenciador* foram testadas e aprovadas, assegurando que o comportamento esperado foi alcançado e a integridade do sistema está garantida.

10 TESTE UNITÁRIO - VISITANTE

1. Teste de Criação de Visitante

Arquivo: *teste_visitante.py*

Função: *testeCriarVisitante*

Objetivo: Verificar a funcionalidade de criação de um novo visitante.

Descrição:

- Utiliza o método *criar* do *VisitanteGerenciador* para adicionar um novo visitante com os atributos *idDiscord*, *nome*, *nomeGlobal* e *language*.
- O teste valida se o retorno não é nulo, indicando sucesso na criação do visitante.

Código:

```
1 def testeCriarVisitante(visitanteGerenciador):
2     """Testa a criação de um visitante."""
3     sucesso = visitanteGerenciador.criar("456", "mendes", "Isaac Mendes", "pt-BR")
4     assert sucesso is not None
```

2. Teste de Busca de Visitante

Arquivo: *teste_visitante.py*

Função: *testeBuscarVisitante*

Objetivo: Testar a busca de um visitante existente no banco de dados.

Descrição:

- Utiliza o método *buscar* do *VisitanteGerenciador* para localizar um visitante com base no *idDiscord*.
- O teste verifica se o retorno não é nulo, indicando que o visitante foi encontrado.

Código:

```
7 def testeBuscarVisitante(visitanteGerenciador):
8     """Testa a busca de um visitante existente."""
9     visitante = visitanteGerenciador.buscar(456)
10    assert visitante is not None
```

3. Teste de Atualização de Visitante

Arquivo: *teste_visitante.py*

Função: *testeAtualizarVisitante*

Objetivo: Validar a atualização dos dados de um visitante existente.

Descrição:

- Utiliza o método *atualizar* do *VisitanteGerenciador* para modificar os campos *nome* e *nomeGlobal* de um visitante com base no *idDiscord*.
- Em seguida, realiza uma busca para verificar se os campos foram atualizados corretamente.
- O teste valida se os valores dos campos *nome* e *nomeGlobal* refletem as alterações.

Código:

```
13 def testeAtualizarVisitante(visitanteGerenciador):
14     """Testa a atualização de um visitante existente."""
15     sucesso = visitanteGerenciador.atualizar(456, "isaacmendes", "Isaac")
16     assert sucesso is not None
17     visitante = visitanteGerenciador.buscar(456)
18     assert visitante.nome == "isaacmendes" and visitante.nomeGlobal == "Isaac"
```

4. Teste de Deleção de Visitante

Arquivo: *teste_visitante.py*

Função: *testeDeletarVisitante*

Objetivo: Testar a remoção de um visitante existente no banco de dados.

Descrição:

- Utiliza o método *deletar* do *VisitanteGerenciador* para excluir um visitante com base no *idDiscord*.
- Após a remoção, tenta buscar o visitante para confirmar que ele foi deletado com sucesso.
- Valida se a remoção foi realizada corretamente.

Código:

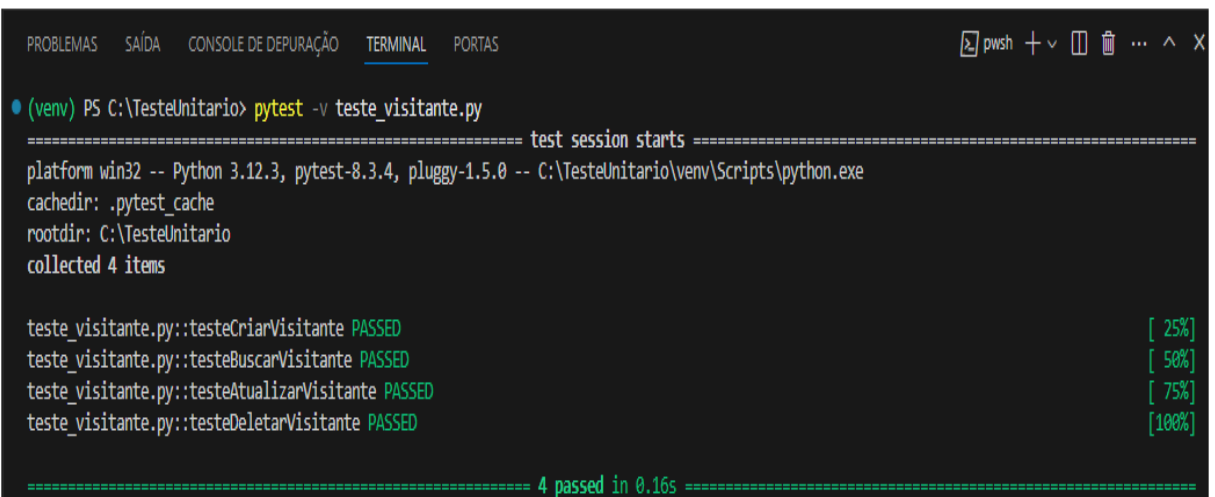
```
21 def testeDeletarVisitante(visitanteGerenciador):
22     """Testa a exclusão de um visitante existente."""
23     sucesso = visitanteGerenciador.deletar(456)
24     assert sucesso is not None
25     visitante = visitanteGerenciador.buscar(456)
26     assert visitante is not None
```

Resultados da Execução

Comando utilizado:

```
pytest -v teste_visitante.py
```

Saída no terminal:



```
PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS
(venv) PS C:\TesteUnitario> pytest -v teste_visitante.py
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.3.4, pluggy-1.5.0 -- C:\TesteUnitario\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\TesteUnitario
collected 4 items

teste_visitante.py::testeCriarVisitante PASSED [ 25%]
teste_visitante.py::testeBuscarVisitante PASSED [ 50%]
teste_visitante.py::testeAtualizarVisitante PASSED [ 75%]
teste_visitante.py::testeDeletarVisitante PASSED [100%]

===== 4 passed in 0.16s =====
```

Conclusão

Os testes unitários validaram com sucesso as operações **CRUD** (Create, Read, Update, Delete) para a entidade **Visitante**. Todas as funcionalidades implementadas na classe *VisitanteGerenciador* foram testadas e aprovadas, garantindo estabilidade, precisão e confiabilidade do código.

11 CONSIDERAÇÕES FINAIS

O software apresentado Dinostat, foi desenvolvido para o gerenciamento de criaturas para o jogo ARK: Survival Ascended, visando atender às necessidades dos jogadores que enfrentam desafios na organização e manutenção das estatísticas de suas criaturas e tribos. A implementação de testes unitários no Bot, visam verificar as funcionalidades individualmente para assegurar que se comportem conforme o esperado, garantindo a qualidade do software.

O framework escolhido para implementação dos testes unitários foi o *Pytest*, uma ferramenta em Python que auxilia tanto em testes simples até mais complexos, atuando na identificação e correção de bugs, além de validar funcionalidades e assegurar o desempenho robusto do bot. Os testes unitários validaram as operações CRUD com as classes *CriaturaGerenciador*, *EspecieGerenciador*, *TokenGerenciador*, *TriboGerenciador*, *UsuarioGerenciador* e *VisitanteGerenciador*, o qual, foram testados e aprovados evidenciando que as funcionalidades estavam operando conforme o definido, garantindo a confiabilidade e eficiência do nosso software, assim, otimizando a experiência dos jogadores do jogo ARK: Survival Ascended.

Para mais detalhes sobre o código e os testes unitários implementados, acesse o repositório do nosso projeto no Github: <https://github.com/MaykoDiouzeff/TesteDeSoftware>.