Testes Automatizados — Reserva Certa

Este documento descreve os testes automatizados implementados para o projeto "Reserva Certa" e instruções para executá-los.

Resumo

- Framework: pytest (Python)

- Local dos testes: `tests/`

- Total de testes implementados: 5 (todos passando localmente)

Objetivo

- Garantir correção e confiabilidade das principais partes do sistema: camada de persistência (repositórios) e API/rotas principais.

Como rodar os testes

1) Instale dependências (recomendado usar o PowerShell no Windows):

cd "c:\Users\iagof\OneDrive\Documentos\PUC\semestre_5\Engenharia2\reserva_certa_flask_app"

python -m pip install -r requirements.txt

2) Execute a suíte de testes:

python -m pytest -q

Arquivos de teste implementados

- `tests/conftest.py` — fixtures do pytest

- Cria um banco de dados SQLite temporário inicializado com os scripts `sql/create_tables.sql` e `sql/insert_data.sql`.

- Sobrescreve `persistence.connection_factory.DB_PATH` para apontar ao DB de teste, garantindo isolamento.

- Fornece a fixture `client` que expõe o `Flask` test client para testes de integração.

- `tests/test_repositories.py` — testes unitários para repositórios

- `test_space_create_and_get` — cria um espaço via `SpaceRepository.create()` e valida `get()` e `list_all()`.

- `test_user_list_and_get` — recupera um usuário pré-existente (seed) e cria/recupera um usuário de teste via `UserRepository`.

- `test_reservation_create_list_update` — cria uma reserva com `ReservationRepository.create()`, valida listagem por usuário e atualiza o status para `CANCELED`.

- `tests/test_app_routes.py` — testes de integração (endpoints)

- `test_index_and_spaces_routes` — verifica que `GET /` e `GET /spaces` retornam 200 e contém conteúdo esperado.

- `test_create_space_via_post` — submete `POST /spaces/new` (formulário) e valida o redirect/flash indicando sucesso.

Cobertura mínima e mapa para o enunciado

- Camada de domínio: os repositórios exercitam operações essenciais (create/get/update) sobre as entidades (spaces, users, reservations)

- Camada de persistência/API: testes cobrem os métodos dos repositórios e endpoints HTTP principais (`/`, `/spaces`, `/spaces/new`).

- Cobertura mínima requerida (3-5 testes): implementados 5 testes automatizados.

Resultado local (exemplo)

- Comandos executados localmente no ambiente de desenvolvimento:

python -m pip install -r requirements.txt

python -m pytest -q

- Saída observada:

5 passed in 0.24s

Nome do grupo/alunos: Iago Fereguetti, Gabriel Cunha, Lucas Henrique e Felipe Dias

Nome da ferramenta utilizada: pytest (Python)

Capturas de tela ou tabela de métricas:

- Execução local dos testes: saída resumida

5 passed in 0.24s

Você pode anexar capturas de tela da execução local ou incluir uma tabela com métricas (tempo de execução, número de testes, cobertura) conforme necessário. Abaixo há uma sugestão de tabela de métricas que você pode preencher com valores reais:

| Métrica | Valor |

|---|---|

| Total de testes | 5 |

| Tests passados | 5 |

| Tempo de execução | 0.24s |

| Cobertura (se aplicável) | --- |

Interpretação dos resultados e propostas de melhoria:

- Interpretação: a suíte atual contém 5 testes que cobrem operações básicas de repositórios e endpoints principais. Todos os testes passam localmente, indicando que os fluxos principais do aplicativo (criação/listagem/atualização) estão funcionando conforme esperado no ambiente de desenvolvimento.

- Propostas de melhoria:

- Aumentar cobertura com casos de borda (validações, entradas inválidas, concorrência simples).

- Adicionar testes de integração que simulem vários usuários e fluxos de reserva.

- Automatizar execução dos testes em CI para validar cada push/PR.
