Relatório de Avaliação de Qualidade de Código

Data: 2025-11-18

Projeto: Reserva Certa

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

Nome da ferramenta utilizada: Flake8, Bandit e Pylint

Como executei (resumo)

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

Flake8

python -m flake8 . --exit-zero > analysis_flake8.txt

Bandit

python -m bandit -r . -f json -o analysis_bandit.json

Pylint

python -m pylint app.py persistence --score=y > analysis_pylint.txt

Sumário executivo

- Flake8: vários avisos de estilo (linhas muito longas, espaços faltando, separação de linhas).

- Bandit: 1 problema de severidade alta (Flask com debug=True), e alguns problemas médios (uso de f-strings em queries SQL, possível risco de injeção).

- Pylint: nota de qualidade 6.49/10 com várias observações (módulos sem docstring, funções sem docstring, nomes de parâmetros que sobrescrevem built-ins, muitos argumentos em métodos de repositório, linhas longas).

Principais problemas identificados (detalhes)

1) Segurança — Flask rodando com debug=True (Bandit B201)

- Arquivo: `app.py`

- Impacto: quando o Flask é executado com debug=True em ambiente acessível, o Werkzeug debug console pode permitir execução remota de código; isso é uma vulnerabilidade crítica em produção.

- Recomendação: assegurar que `app.run(debug=True)` seja usado apenas em desenvolvimento e que, em produção, `debug` esteja `False`. Usar variáveis de ambiente (FLASK_ENV) e servidores WSGI (gunicorn/uWSGI) para produção.

2) Segurança — construção de queries com f-strings (Bandit B608)

- Arquivos: `persistence/space_repository.py`, `persistence/reservation_repository.py`

- Impacto: construir queries SQL dinamicamente por concatenação/f-strings pode introduzir vetores de injeção se campos não forem validados/escapados corretamente.

- Observação: o projeto usa placeholders em vários inserts/executions, porém os métodos `update` usam f-strings para montar `SET a=?, b=?` o que é potencialmente arriscado.

- Recomendação: evitar montar SQL com f-strings; validar os nomes de colunas contra uma lista conhecida antes de interpolar, ou usar uma camada de query builder/ORM (ex.: SQLAlchemy) que trate parâmetros de forma segura.

3) Estilo / Legibilidade (Flake8 + Pylint)

- Problemas recorrentes:

- Linhas muito longas (E501 / line-too-long)

- Espaçamento em expressões e após vírgulas (E231, E225)

- Falta de linhas em branco entre funções/classes (E302)

- Múltiplas importações na mesma linha (C0410)

- Falta de docstrings em módulos, classes e funções (C0114/C0115/C0116)

- Uso de identificador `id` que mascara o builtin (W0622)

- Recomendação: aplicar ferramentas automáticas de formatação (Black) e lint (Flake8/Pylint) como parte do fluxo de desenvolvimento; adicionar docstrings mínimas e seguir PEP8. Renomear parâmetros `id` para `*_id` para evitar sobrescrita do builtin.

4) Design / Simplicidade (Pylint)

- Observado: métodos com muitos parâmetros (ex.: ReservationRepository.create possui 8-9 argumentos) — isso facilita erros e dificulta testes.

- Recomendação: usar dataclasses / objetos de domínio (por exemplo, uma classe Reservation) e passar instâncias, ou reduzir número de parâmetros através de objetos/DTOs.

5) Qualidade geral (Pylint score)

- Pylint calculou nota: 6.49 / 10 (arquivo de saída `analysis_pylint.txt`).

- Principais categorias que afetam a nota: ausência de docstrings, complexidade de métodos, problemas de estilo e import ordering.

Trechos de saída relevantes (resumo)

- Flake8 (exemplos):

.\app.py:1:80: E501 line too long (84 > 79 characters)

.\persistence\reservation_repository.py:5:80: E501 line too long (123 > 79 characters)

.\persistence\reservation_repository.py:8:29: E231 missing whitespace after ','

- Bandit (exemplos):

B201: flask_debug_true -> app.py: app.run(debug=True) (severity: HIGH)

B608: hardcoded_sql_expressions -> persistence/... (severity: MEDIUM)

- Pylint (nota e exemplos):

Your code has been rated at 6.49/10

Common messages: missing-module-docstring, missing-function-docstring, line-too-long, too-many-arguments, redefined-builtin

Ações recomendadas (prioritizadas)

1. Segurança (prioridade alta)

- Remover `debug=True` em produção e documentar workflow de deployment seguro.

- Revisar métodos que constroem SQL dinamicamente; usar parametrização segura e/ou ORM.

2. Integridade e testes (prioridade média)

- Ampliar testes unitários para cobrir casos de borda e validações (ex.: inputs para update/query).

- Executar flake8/bandit/pylint e pytest regularmente (ex.: via CI).

3. Qualidade do código / manutenibilidade (prioridade média)

- Adotar formatação automática (Black) e config de Flake8 para regras do time.

- Adicionar docstrings mínimos para módulos, classes e funções públicas.

- Evitar nomes que sobrescrevem builtins (por exemplo, `id`).

- Refatorar métodos com muitos parâmetros para usar dataclasses/objetos de domínio.

4. Estilo (prioridade baixa)

- Corrigir avisos Flake8 (linhas longas, espaços, blank lines).

- Ajustar ordenação de imports (isort pode ajudar).

Evidências geradas

- analysis_flake8.txt (saída completa do Flake8)

- analysis_bandit.json (relatório JSON do Bandit)

- analysis_pylint.txt (relatório do Pylint)

Capturas de tela ou tabela de métricas

Você pode anexar capturas de tela dos resultados (execução de comandos ou HTML gerado) ou preencher a tabela abaixo com métricas observadas localmente:

| Métrica | Valor |

|---|---|

| Flake8 warnings | (ex.: 34) |

| Bandit findings (HIGH) | 1 |

| Bandit findings (MEDIUM) | 3 |

| Pylint score | 6.49/10 |

Interpretação dos resultados e propostas de melhoria

- Interpretação: a análise estática indica problemas de estilo recorrentes e algumas questões relevantes de segurança e design que merecem prioridade. A nota do Pylint mostra espaço para melhoria em documentação e simplicidade das APIs.

- Propostas de melhoria:

- Prioridade imediata: remover `debug=True` em produção e revisar pontos que constroem SQL dinamicamente.

- Curto prazo: configurar formatação automática (Black) e ajustar regras do Flake8; corrigir avisos críticos e adicionar docstrings.

- Médio prazo: refatorar métodos com muitos parâmetros usando dataclasses/DTOs e ampliar a suíte de testes.
