

Etapas e Resultados

Influência de boas práticas de

- (i) Arquitetura de software
- (ii) Padrões e Estilos Arquiteturais
- (iii) Padrões de Projeto na Produtividade

Membros da Equipe:

1. Letícia Cavalcanti
2. Maria Diniz

Disciplina:

COMP0439 - Engenharia de Software II (2024.2 - T03)

1. Apresentação do Projeto Escolhido

1.1. Nome do Projeto e URL de Acesso

Projeto: Flutter

URL: <https://github.com/flutter/flutter>

1.2. Finalidade do Projeto

Flutter é um framework de código aberto desenvolvido pelo Google para a criação de aplicativos que funcionam em diversas plataformas com um único código. Com ele, é possível desenvolver aplicativos para Android, iOS, Web, Desktop e até sistemas embarcados. O Flutter se destaca por sua estrutura baseada em widgets, onde cada elemento da interface (como botões, textos e imagens) faz parte de uma árvore de componentes (widget tree). Isso torna a criação de telas mais flexível e organizada.

Outro ponto forte do Flutter é o uso do mecanismo gráfico Skia, que permite a renderização rápida e suave dos elementos visuais, garantindo alto desempenho e animações fluidas. A linguagem utilizada no Flutter é o Dart, criada pelo próprio Google. Ela foi projetada para ser simples, eficiente e otimizada para interfaces gráficas, facilitando o desenvolvimento de aplicativos com uma aparência moderna e responsiva.

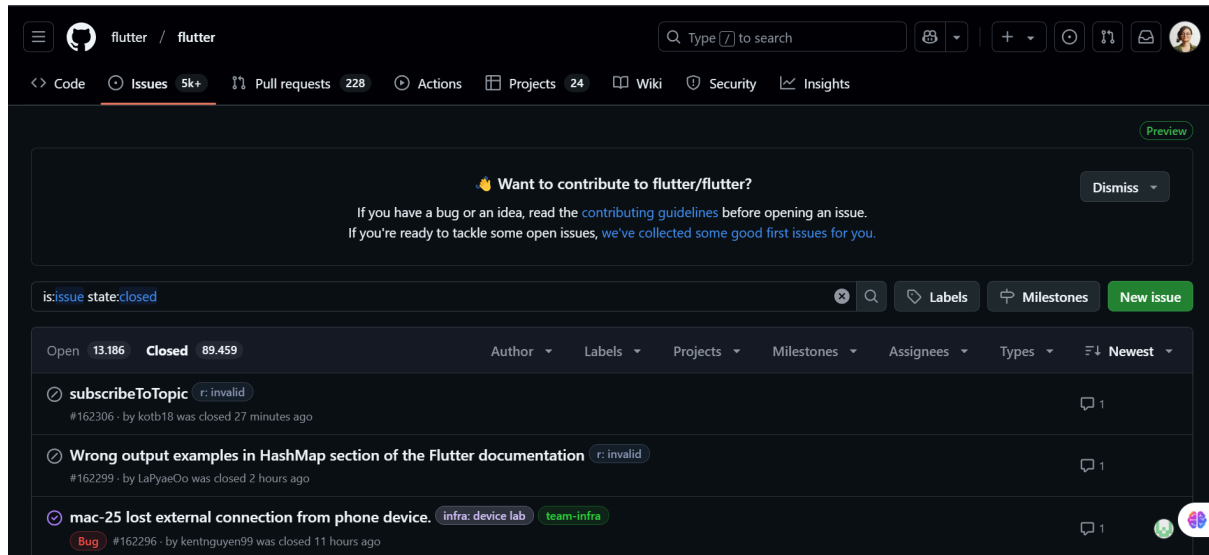
1.3. Características Relevantes do Projeto

- Grande comunidade de desenvolvedores: Possui mais de 1.481 contribuidores no GitHub.
- Elevado número de issues: O projeto apresenta mais de 89.503 issues fechadas
- Quantidade expressiva de pull requests: Conta com mais de 58.125 pull requests fechados registrados.
- Arquitetura modular e flexível: Utiliza conceitos de boas práticas de arquitetura de software, facilitando a extensibilidade e manutenção.
- Uso de padrões e estilos arquiteturais: O Flutter adota padrões como MVU (Model-View-Update) e arquiteturas como Redux e Bloc para gerenciamento de estado.
- Adoção de padrões de projeto: Implementa padrões como Singleton, Factory e Observer em diversas partes do código.

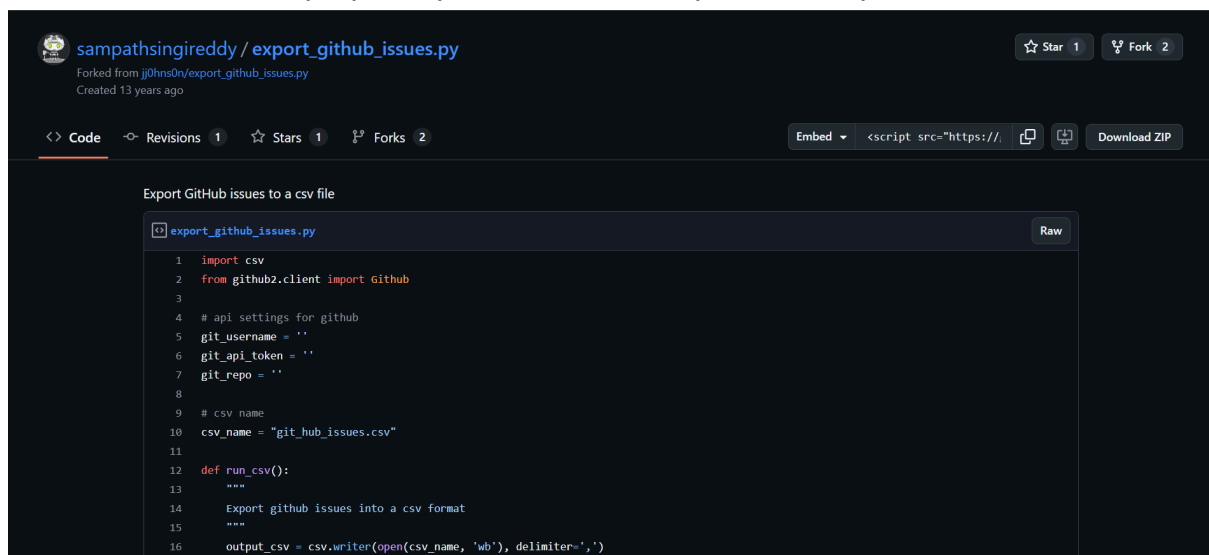
A partir dessas observações, daremos sequência às próximas etapas do estudo, realizando a coleta, classificação e análise das issues do Flutter relacionadas a boas práticas de arquitetura de software, padrões e estilos arquiteturais e padrões de projeto.

2. Etapas

2.1. Optamos pelo Flutter após uma busca por projetos de código aberto que atendessem aos critérios exigidos na atividade ou estivessem próximos desses requisitos.



2.2. Para facilitar a extração das 300 issues fechadas (closed) do repositório do Flutter no GitHub, buscamos um código aberto que automatizasse esse processo. Encontramos um script que exportava as issues para um arquivo CSV.



<https://gist.github.com/sampathsingireddy/2993646>

2.3. Modificamos o código original para que, em vez de exportar os dados para um arquivo CSV, as issues fossem diretamente inseridas em um banco de dados PostgreSQL. Além disso, adicionamos os atributos extras exigidos na atividade,

como data de abertura, data de conclusão, tempo de resolução, prioridade e milestone.

Utilizamos o ChatPT, através do modelo 4o-mini, ajustar para baixar as issues direto no Postgres com o seguinte prompt principal:

“\${código} + como ao inves de converter para csv eu jogar direto no banco postgres?”

E o resultado foi obtido de acordo com nossas máquinas e banco:

```
import requests
import psycopg2
from datetime import datetime

# Configurações do banco
try:
    conn = psycopg2.connect(
        dbname="engenharia-softw-II",
        user="postgres",
        password="1234",
        host="localhost",
        port="5432"
    )
    cursor = conn.cursor()
    print("Conexão com o banco de dados estabelecida com sucesso.")
except Exception as e:
    print(f"Erro ao conectar ao banco de dados: {e}")
    exit()

# Criação da tabela (se não existir)
try:
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS issues (
            id SERIAL PRIMARY KEY,
            issue_id BIGINT UNIQUE,
            title TEXT NOT NULL,
            body TEXT,
            state VARCHAR(20),
            created_at TIMESTAMP,
            closed_at TIMESTAMP,
            resolution_time_days NUMERIC,
            priority TEXT,
            milestone TEXT,
            author TEXT,
```

```

        assignee TEXT,
        updated_at TIMESTAMP
    );
    """
    conn.commit()
    print("Tabela 'issues' criada/verificada com sucesso.")
except Exception as e:
    print(f"Erro ao criar/verificar a tabela: {e}")
    conn.close()
    exit()

# Fazendo a requisição na API com paginação para buscar issues "closed"
url = "https://api.github.com/repos/flutter/flutter/issues"
headers = {"Authorization": "TOKEN: DADO PROTEGIDO"} # Substitua pelo seu
token do GitHub
per_page = 100 # Máximo permitido por página
total_issues = 300 # Quantidade total desejada
issues_closed = 0 # Contador de issues já buscadas
page = 1 # Página inicial

try:
    while issues_closed < total_issues:
        if issues_closed == 301:
            per_page = 1
        else:
            per_page = per_page

        paginated_url = f"{url}?state=closed&per_page={per_page}&page={page}"
        response = requests.get(paginated_url, headers=headers)
        response.raise_for_status()
        issues = response.json()

        if not issues: # Se não houver mais issues, sair do loop
            break

        print(f"Página {page}: {len(issues)} issues fechadas encontradas.")

        for issue in issues:
            try:
                issue_id = issue.get('id')
                title = issue.get('title', 'Sem título')
                body = issue.get('body') if 'body' in issue else ''
                state = issue.get('state', 'unknown')

```

```

        created_at = issue.get('created_at')
        closed_at = issue.get('closed_at')
        updated_at = issue.get('updated_at')

        if created_at and closed_at:
            resolution_time_days = (
                datetime.fromisoformat(closed_at[:-1]) -
                datetime.fromisoformat(created_at[:-1])
            ).days
        else:
            resolution_time_days = None

        # Pegando prioridade com base nos labels (se existir)
        labels = issue.get('labels', [])
        priority = None
        for label in labels:
            label_name = label.get('name', '').lower()
            if "high" in label_name:
                priority = "High"
            elif "medium" in label_name:
                priority = "Medium"
            elif "low" in label_name:
                priority = "Low"

        # Milestone
        milestone_data = issue.get('milestone')
        milestone = milestone_data.get('title') if milestone_data
else None

        # Usuário autor e atribuído
        user_data = issue.get('user')
        author = user_data.get('login') if user_data else None

        assignee_data = issue.get('assignee')
        assignee = assignee_data.get('login') if assignee_data else
None

        cursor.execute("""
            INSERT INTO issues (issue_id, title, body, state,
created_at, closed_at, resolution_time_days, priority, milestone, author,
assignee, updated_at)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            ON CONFLICT (issue_id) DO NOTHING;
        """, (issue_id, title, body, state, created_at, closed_at,
resolution_time_days, priority, milestone, author, assignee, updated_at))

```

```

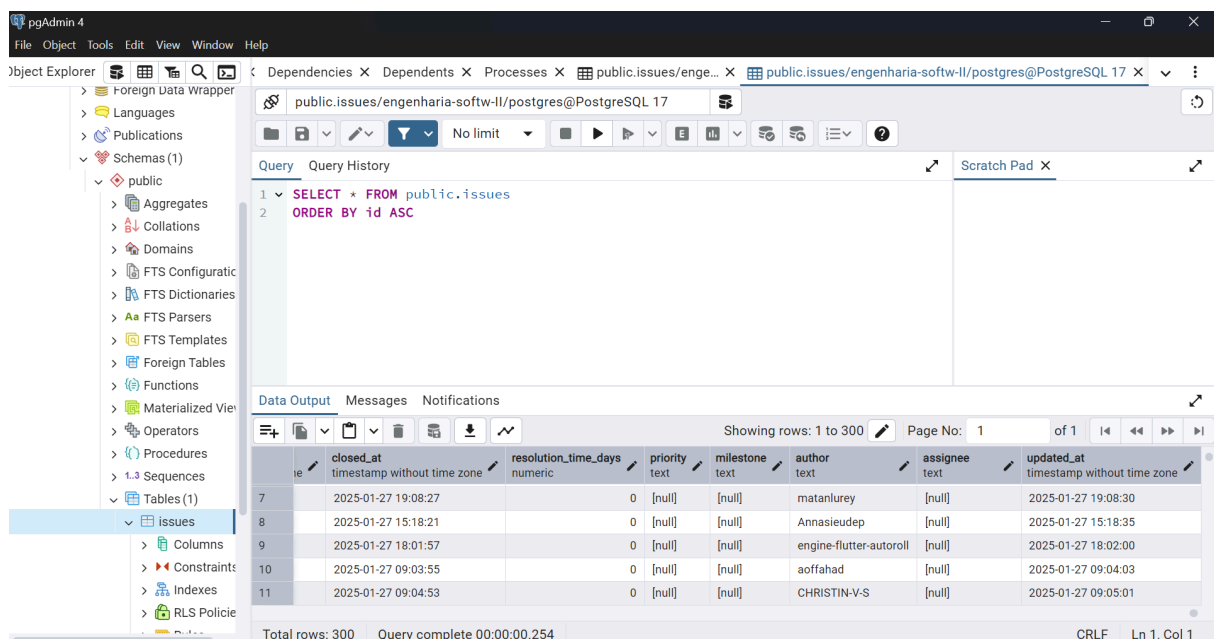
except Exception as issue_error:
    print(f"Erro ao processar a issue {issue_id}: {issue_error}")

conn.commit()
print("Dados salvos no banco.")
issues_closed += len(issues) # Atualizar o contador
page += 1 # Ir para a próxima página

print(f"Total de {issues_closed} issues fechadas inseridas no banco.")
except requests.exceptions.RequestException as e:
    print(f"Erro ao acessar a API do GitHub: {e}")
except Exception as e:
    print(f"Erro ao inserir as issues no banco: {e}")
finally:
    cursor.close()
    conn.close()
    print("Conexão com o banco encerrada.")

```

2.4. Após realizar ajustes no código, executamos uma série de testes para garantir que a exportação estava funcionando corretamente. Após validações bem-sucedidas, conseguimos estabelecer a conexão com o banco de dados e inserir os registros. Confirmamos que os dados estavam devidamente armazenados no banco de dados e realizamos a exportação para um arquivo CSV para facilitar a análise posterior.



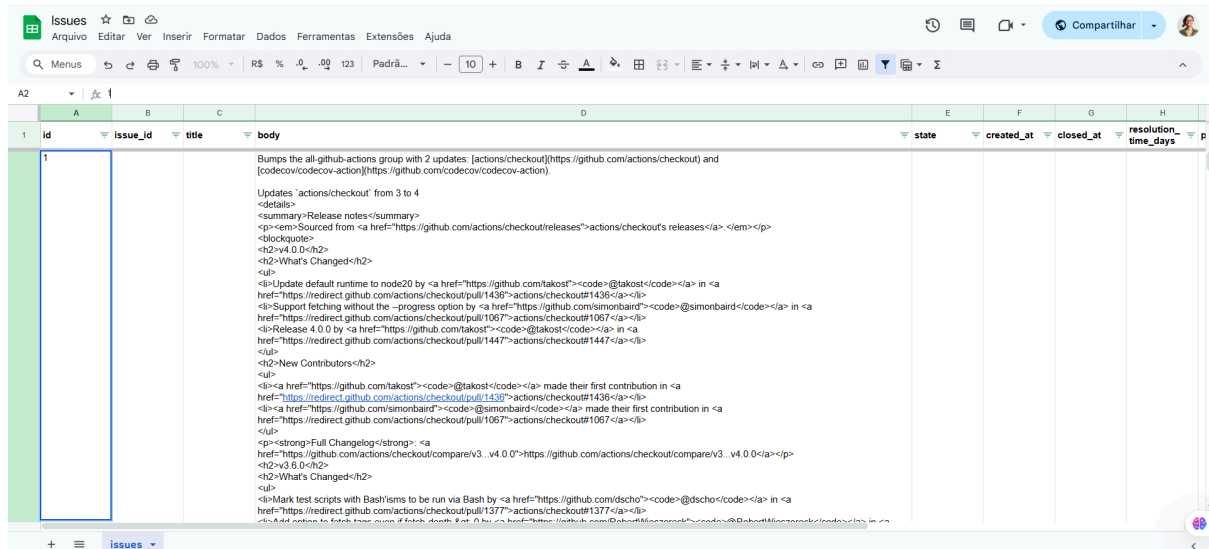
The screenshot shows the pgAdmin 4 interface. On the left, the 'Object Explorer' shows the database structure, with 'public' selected. The 'Query' tab is active, displaying a SQL query: `SELECT * FROM public.issues ORDER BY id ASC`. The 'Data Output' tab shows the results of the query in a table format. The table has 11 columns: `id`, `closed_at`, `resolution_time_days`, `priority`, `milestone`, `author`, `assignee`, and `updated_at`. The results show 11 rows of data.

id	closed_at	resolution_time_days	priority	milestone	author	assignee	updated_at
7	2025-01-27 19:08:27		0	[null]	matanlurey	[null]	2025-01-27 19:08:30
8	2025-01-27 15:18:21		0	[null]	Annasieudep	[null]	2025-01-27 15:18:35
9	2025-01-27 18:01:57		0	[null]	engine-flutter-autoroll	[null]	2025-01-27 18:02:00
10	2025-01-27 09:03:55		0	[null]	aoffahad	[null]	2025-01-27 09:04:03
11	2025-01-27 09:04:53		0	[null]	CHRISTIN-V-S	[null]	2025-01-27 09:05:01

Total rows: 300 Query complete 00:00:00.254 CRLF Ln 1, Col 1

2.5. Importamos o arquivo CSV para o Google Planilhas e adicionamos a coluna "tema relacionado". Em seguida, iniciamos a classificação das issues de acordo com os três temas da atividade:

- Arquitetura de Software
- Padrões e Estilos Arquiteturais0
- Padrões de Projeto



	A	B	C	D	E	F	G	H
1	id	issue_id	title	body	state	created_at	closed_at	resolution_time_days
1	1			<p>Bumps the all-github-actions group with 2 updates: [actions/checkout](https://github.com/actions/checkout) and [codecov/codecov-action](https://github.com/codecov/codecov-action).</p> <p>Updates 'actions/checkout' from 3 to 4</p> <p><details></p> <p><summary>Release notes</summary></p> <p><p>Sourced from actions/checkout's releases. </p></p> <p><blockquote></p> <p><h2>v4.0.0</h2></p> <p><h2>What's Changed</h2></p> <p></p> Update default runtime to node20 by <code>@takost</code> in actions/checkout#1436 Support fetching without the --progress option by <code>@simonbaird</code> in actions/checkout#1067 Release 4.0.0 by <code>@takost</code> in actions/checkout#1447 <p><h2>New Contributors</h2></p> <p></p> <code>@takost</code> made their first contribution in actions/checkout#1436 <code>@simonbaird</code> made their first contribution in actions/checkout#1067 <p><p>Full Changelog https://github.com/actions/checkout/compare/v3..v4.0.0</p></p> <p><h2>v4.0.0</h2></p> <p><h2>What's Changed</h2></p> <p></p> Mark test scripts with Bashisms to be run via Bash by <code>@dscho</code> in actions/checkout#1377 Add action to fetch tree size if fetch depth > 0 by <code>@BobetMoraes</code> in actions/checkout#1377 <p></p>				

2.6. Ao analisar o tempo necessário para classificar manualmente todas as 300 issues, percebemos que esse processo seria demorado e pouco eficiente. Para otimizar a tarefa, decidimos desenvolver uma ferramenta baseada em Inteligência Artificial que pudesse ler e classificar automaticamente cada issue com base nas colunas "title" e "body" do banco de dados.

Para isso, utilizamos o ChatGPT, através do modelo 3.5 Turbo, para auxiliar na geração do código. Fizemos a seguinte solicitação:

"Quer gerar um código usando OpenAI e LangChain para ler os itens das colunas 'title' e 'body' no banco de dados e me retornar a classificação delas numa coluna ou num arquivo TXT?"

Após validar a abordagem inicial, fizemos um novo pedido para armazenar diretamente os resultados no banco de dados:

"Tem como salvar as classificações direto no banco numa coluna nova chamada 'tema_relacionado'?"

Com isso, conseguimos automatizar o processo, reduzindo significativamente o esforço manual necessário para a categorização das issues.

```
import os
from dotenv import load_dotenv
```



```

import psycopg2
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage

load_dotenv()

# Configurações do banco de dados
DB_NAME = os.getenv("DB_NAME")
DB_USER = os.getenv("DB_USER")
DB_PASSWORD = os.getenv("DB_PASSWORD")
DB_HOST = os.getenv("DB_HOST")
DB_PORT = os.getenv("DB_PORT")
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

if not OPENAI_API_KEY:
    raise ValueError("A chave da API OpenAI não foi encontrada. Verifique o
arquivo .env.")

# Função para conectar ao banco de dados
def connect_to_db():
    try:
        conn = psycopg2.connect(
            dbname=DB_NAME,
            user=DB_USER,
            password=DB_PASSWORD,
            host=DB_HOST,
            port=DB_PORT,
        )
        return conn
    except Exception as e:
        print(f"Erro ao conectar ao banco de dados: {e}")
        exit()

# Função para classificar os textos com o modelo GPT-3.5
def classify_texts_with_gpt35(body):
    try:
        # Inicializa o modelo GPT-3.5 com temperatura ajustada
        chat = ChatOpenAI(model="gpt-3.5-turbo", temperature=0.3,
openai_api_key=OPENAI_API_KEY)

        # Prompt para classificar o texto
        message = HumanMessage(
            content=(
                f"Classifique o seguinte texto em um dos seguintes temas:\n"
                f"(i) Arquitetura de Software\n"
                f"(ii) Padrões e Estilos Arquiteturais\n"
                f"(iii) Padrões de Projeto\n"
                f"\nTexto: {body}\n\n"
                f"Responda apenas com o nome do tema correspondente, sem
números ou outros detalhes."
            )
        )

        # Obtém a resposta do modelo
        response = chat([message])
        classification = response.content.strip()

        # Limpeza de possíveis números e parênteses extras na resposta
        if classification.startswith("(i)":
            classification = "Arquitetura de Software"
        elif classification.startswith("(ii)":

```

```

        classification = "Padrões e Estilos Arquiteturais"
    elif classification.startswith("(iii)":
        classification = "Padrões de Projeto"

    # Verifica se a classificação é válida
    if classification not in [
        "Arquitetura de Software",
        "Padrões e Estilos Arquiteturais",
        "Padrões de Projeto"
    ]:
        print(f"Classificação inválida: {classification}. Nenhuma
alteração será feita.")
        return None

    return classification
except Exception as e:
    print(f"Erro ao classificar o texto: {e}")
    return None

# Função principal
def main():
    conn = connect_to_db()
    cursor = conn.cursor()

    try:
        # Adiciona a coluna "tema_relacionado" se ela ainda não existir
        cursor.execute("""
            DO $$
            BEGIN
                IF NOT EXISTS (
                    SELECT 1
                    FROM information_schema.columns
                    WHERE table_name = 'issues'
                    AND column_name = 'tema_relacionado'
                ) THEN
                    ALTER TABLE issues ADD COLUMN tema_relacionado TEXT;
                END IF;
            END$$;
        """)
        conn.commit()
        print("Coluna 'tema_relacionado' verificada/adicionada com sucesso.")

        # Lê os dados do banco de dados
        cursor.execute("SELECT id, body FROM issues WHERE tema_relacionado IS
NULL")
        rows = cursor.fetchall()

        print("Classificando os textos...")

        # Classifica e salva cada texto no banco de dados
        for row in rows:
            issue_id, body = row
            classification = classify_texts_with_gpt35(body)

            # Só atualiza o banco se a classificação for válida
            if classification:
                cursor.execute("""
                    UPDATE issues
                    SET tema_relacionado = %s
                    WHERE id = %s;
                """, (classification, issue_id))

```

```

        print(f"Issue {issue_id} classificada como:
{classification}")
    else:
        print(f"Issue {issue_id} não classificada.")

    conn.commit()
    print("Classificações salvas no banco de dados com sucesso.")

except Exception as e:
    print(f"Erro ao processar os dados: {e}")

finally:
    cursor.close()
    conn.close()
    print("Conexão com o banco encerrada.")

if __name__ == "__main__":
    main()

```

2.7. Para garantir a precisão do modelo comparamos as classificações geradas pela IA com nossa interpretação humana. Esse processo permitiu avaliar a assertividade da ferramenta e ajustar eventuais inconsistências.

2.8. Para facilitar a análise de métricas, modificamos uma coluna do banco de dados para armazenar o tempo de resolução de cada issue em horas. Dessa forma, conseguimos obter um indicador mais preciso do tempo médio necessário para a conclusão das tarefas.

Utilizamos o seguinte prompt no ChatGPT, através do modelo 4-o mini, para gerar o código necessário:

"Gere um código Python que lê as colunas 'created_at' e 'closed_at' e devolve o tempo de fechamento em horas. Considere que meus arquivos vão estar no env."

A seguir, implementamos o código responsável pela conversão do tempo de resolução das issues em horas:

```

import os
import psycopg2
from dotenv import load_dotenv
from datetime import datetime

# Carregar variáveis de ambiente do arquivo .env
load_dotenv()

# Configurações do banco de dados
DB_NAME = os.getenv("DB_NAME")
DB_USER = os.getenv("DB_USER")
DB_PASSWORD = os.getenv("DB_PASSWORD")
DB_HOST = os.getenv("DB_HOST")
DB_PORT = os.getenv("DB_PORT")

```

```

# Função para conectar ao banco de dados
def connect_to_db():
    try:
        conn = psycopg2.connect(
            dbname=DB_NAME,
            user=DB_USER,
            password=DB_PASSWORD,
            host=DB_HOST,
            port=DB_PORT,
        )
        return conn
    except Exception as e:
        print(f"Erro ao conectar ao banco de dados: {e}")
        exit()

# Função para calcular o tempo de fechamento das issues em horas
def calcular_tempo_fechamento():
    conn = connect_to_db()
    cursor = conn.cursor()

    try:
        # Seleciona as datas de criação e fechamento das issues fechadas
        cursor.execute("""
            SELECT id, created_at, closed_at
            FROM issues
            WHERE state = 'closed' AND created_at IS NOT NULL AND closed_at
IS NOT NULL;
            """)

        issues = cursor.fetchall()

        for issue_id, created_at, closed_at in issues:
            # Calcula a diferença em horas
            created_at_dt = datetime.fromisoformat(str(created_at))
            closed_at_dt = datetime.fromisoformat(str(closed_at))
            resolution_time_hours = (closed_at_dt -
created_at_dt).total_seconds() / 3600

            # Atualiza a tabela com o tempo de fechamento
            cursor.execute("""
                UPDATE issues
                SET resolution_time_hours = %s
                WHERE id = %s;
            """, (resolution_time_hours, issue_id))

        conn.commit()
        print(f"{len(issues)} issues atualizadas com tempo de fechamento em
horas.")

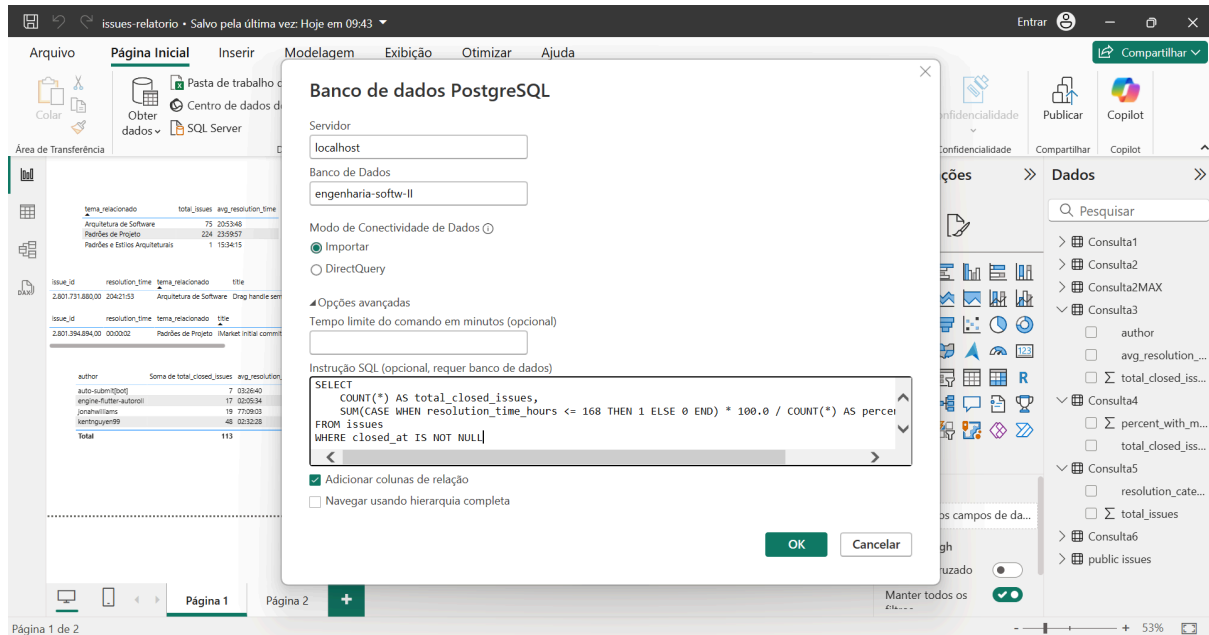
    except Exception as e:
        print(f"Erro ao calcular tempo de fechamento: {e}")

    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    calcular_tempo_fechamento()

```

2.9. Para analisar os dados extraídos, realizamos consultas SQL diretamente no Power BI, extraindo informações relevantes sobre o tempo de resolução das issues.



Consultando as seguintes informações:

1. Nº de issues e tempo médio de resolução em horas por tema

```
SELECT
    tema_relacionado,
    COUNT(*) AS total_issues,
    TO_CHAR(
        INTERVAL '1 second' * ROUND(AVG(resolution_time_hours) *
3600),
        'HH24:MI:SS'
    ) AS avg_resolution_time
FROM issues
WHERE resolution_time_hours IS NOT NULL
GROUP BY tema_relacionado;
```

2. Tempo máximo e mínimo de resolução de issues

-- Issue mais demorada

```
WITH max_issue AS (
    SELECT
        issue_id,
        tema_relacionado,
        title,
        TO_CHAR(
            INTERVAL '1 second' * ROUND(resolution_time_hours * 3600),
            'HH24:MI:SS'
        ) AS resolution_time
```

```

        FROM issues
        WHERE closed_at IS NOT NULL
        ORDER BY resolution_time_hours DESC
        LIMIT 1
    )
    SELECT * FROM max_issue;
-- Issue mais rápida
    WITH min_issue AS (
        SELECT
            issue_id,
            tema_relacionado,
            title,
            TO_CHAR(
                INTERVAL '1 second' * ROUND(resolution_time_hours * 3600),
                'HH24:MI:SS'
            ) AS resolution_time
        FROM issues
        WHERE closed_at IS NOT NULL
        ORDER BY resolution_time_hours ASC
        LIMIT 1
    )
    SELECT * FROM min_issue;

```

3. Top 5 autores que mais fecharam issues e tempo médio de resolução por autor

```

    SELECT
        author,
        COUNT(*) AS total_closed_issues,
        TO_CHAR(
            INTERVAL '1 second' * ROUND(AVG(resolution_time_hours) *
3600),
            'HH24:MI:SS'
        ) AS avg_resolution_time
    FROM issues
    WHERE closed_at IS NOT NULL AND resolution_time_hours IS NOT
NULL
    GROUP BY author
    HAVING COUNT(*) > 5
    ORDER BY total_closed_issues DESC
    LIMIT 5;

```

4. Impacto das issues com base no tempo de fechamento

```

    SELECT
        issue_id,

```

```

        title AS titulo,
        milestone AS marco,
        CASE
        WHEN resolution_time_hours <= 24 THEN 'Curto Prazo'
        WHEN resolution_time_hours BETWEEN 24 AND 168 THEN
'Médio Prazo'
        ELSE 'Longo Prazo'
        END AS impacto_tempo
FROM issues
WHERE state = 'closed';

```

5. Issues fechadas dentro de 24 horas vs. acima disso

Para entender quantas issues são resolvidas rapidamente:

```

SELECT
    CASE WHEN resolution_time_hours <= 24 THEN 'Menos de 1 dia'
    ELSE 'Mais de 1 dia' END AS resolution_category,
    COUNT(*) AS total_issues
FROM issues
WHERE closed_at IS NOT NULL
GROUP BY resolution_category;

```

6. Percentual de issues fechadas dentro de um prazo razoável (exemplo: 7 dias)

```

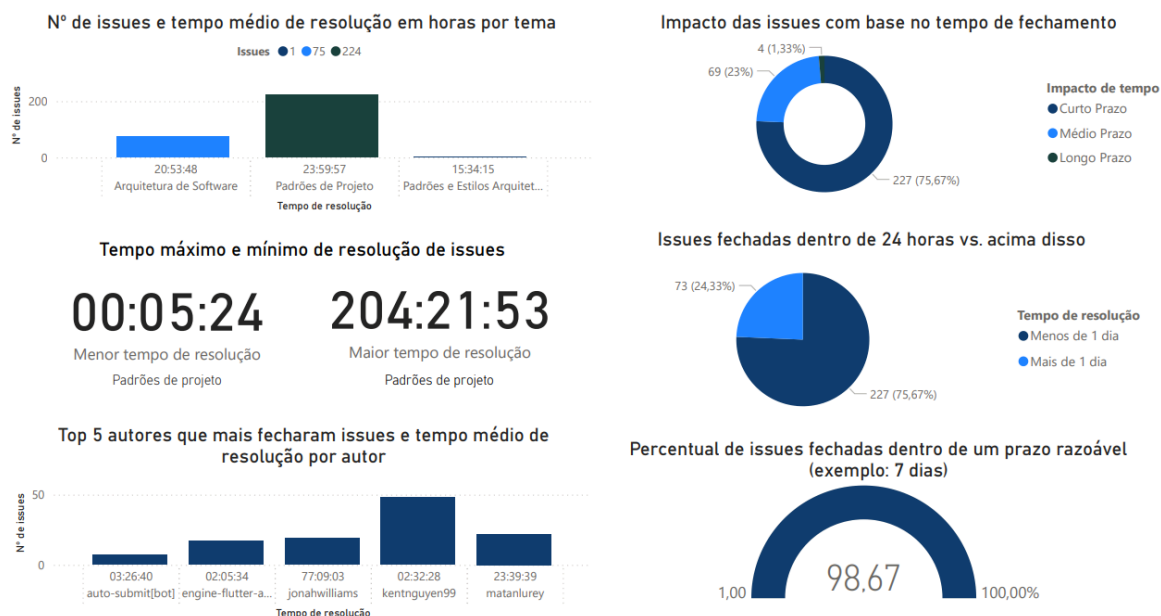
SELECT
    COUNT(*) AS total_closed_issues,
    SUM(CASE WHEN resolution_time_hours <= 168 THEN 1 ELSE 0
    END) * 100.0 / COUNT(*) AS percent_within_7_days
FROM issues
WHERE closed_at IS NOT NULL;

```

2.10. Desenvolvemos um dashboard no PowerBI, utilizando gráficos gerados a partir de consultas SQL executadas diretamente na plataforma.

3. Resultados

Resultados: Análise de 300 issues do Flutter



3.1. Nº de issues e tempo médio de resolução por tema

Análise:

- O tempo médio de resolução das issues mostra uma variação significativa entre os temas.
- Padrões de Projeto tem o maior tempo médio, o que pode indicar maior complexidade ou prioridade menor.
- Arquitetura de Software tem um tempo médio próximo a 21 horas, possivelmente porque envolve decisões técnicas que demandam mais avaliação.

3.2. Tempo máximo e mínimo de resolução de issues

Análise:

- Menor tempo: 5 minutos e 24 segundos
 - Esse tempo sugere que as issues mais rápidas de resolver são aquelas de baixa complexidade, como pequenos ajustes ou correções no código, revisões de documentação ou alterações visuais menores.
 - Pode também ser um reflexo de situações onde a solução já estava preparada antes da abertura da *issue* ou onde ferramentas automatizadas facilitaram o trabalho.
- Maior tempo: 204 horas (8,5 dias)
 - Esse tempo reforça a ideia de que algumas *issues* exigem análise aprofundada, múltiplas etapas de desenvolvimento e testes rigorosos.

- Problemas críticos ou de alta complexidade, como reestruturações de arquitetura, otimizações de desempenho ou decisões estratégicas, frequentemente demandam esse nível de esforço.
- Outro fator pode ser a dependência de terceiros ou o envolvimento de diversas equipes no processo.

3.3. Top 5 autores que mais fecharam issues e tempo médio de resolução por autor

Análise:

- A presença de um bot do Flutter, o “engine-flutter-autoroll”, como o maior responsável pelo fechamento de issues sugere automação no processo de fechamento. Por este motivo, foi um dos selecionados no ranking.
 - O engine-flutter-autoroll é um bot automatizado do GitHub responsável por sincronizar atualizações entre o repositório da engine do Flutter (flutter/engine) e o repositório principal do framework (flutter/flutter).
 - Principais funções:
 - Monitoramento de atualizações: Identifica novas mudanças na engine do Flutter.
 - Criação de Pull Requests automáticos: Gera PRs para integrar as atualizações ao repositório principal do Flutter.
 - Execução de testes: Dispara testes automatizados para validar a estabilidade das atualizações.
 - Mesclagem automatizada: Caso os testes sejam bem-sucedidos, o bot pode mesclar as alterações automaticamente ou aguardar aprovação manual.
 - Importância do engine-flutter-autoroll:
 - A utilização desse bot garante que a engine do Flutter esteja sempre sincronizada com as últimas mudanças, reduzindo a necessidade de intervenções manuais e minimizando riscos de incompatibilidades. Além disso, o processo automatizado contribui para a estabilidade e confiabilidade do framework, assegurando que apenas atualizações testadas sejam incorporadas.
- Autores humanos têm tempos médios muito variados, o que pode indicar que alguns lidam com issues mais complexas do que outros.
- jonahwilliams apresenta o maior tempo médio de resolução, possivelmente porque trata de issues mais complexas.

3.4. Impacto das issues com base no tempo de fechamento

Análise:

- O tempo foi classificado como curto para questões resolvidas em até 24 horas, médio para aquelas resolvidas entre 24 horas e 7 dias, e longo para questões que levaram mais de 7 dias para serem resolvidas.
- A distribuição mostra uma eficiência na resolução da maioria das issues, com quase 76% resolvidas rapidamente.
- Poucas issues permaneceram abertas por um longo período, o que indica uma boa capacidade de resposta e priorização das issues mais críticas.

3.5. Issues fechadas dentro de 24 horas vs. acima disso

Análise:

- Apesar da maioria das issues levarem mais de 24h para serem resolvidas, cerca de 1/4 das issues são resolvidas em menos de um dia, o que é um indicador positivo de eficiência.
- Poderia ser interessante analisar quais tipos de issues são resolvidas rapidamente e quais demoram mais para identificar padrões e possíveis otimizações.

3.6. Percentual de issues fechadas dentro de um prazo razoável (7 dias)

Análise:

- Esse é um indicador altamente positivo. Mostra que a grande maioria das issues não fica aberta por longos períodos, sugerindo que o time tem uma política eficiente de fechamento.
- O 1,33% restante pode representar casos mais críticos ou menos prioritários.

4. Conclusões Gerais

Pontos Positivos

- Alto índice de resolução rápida: 75% das issues são resolvidas rapidamente.
- Baixo percentual de issues de longo prazo: Apenas 1,33% permanecem abertas por um tempo prolongado.
- Automação eficiente: O bot desempenha um papel relevante no fechamento de issues.
- Ótimo prazo médio de resolução: 98,67% das issues fechadas em menos de 7 dias.

Pontos de Melhoria

- Grande variação no tempo médio de resolução por autor: Alguns autores têm tempos de resolução significativamente maiores que outros.
- Padrões de Projeto demora mais para ser resolvido: Poderia ser interessante investigar se esse tempo maior afeta a produtividade.

Sugestões

- Investigar os tipos de issues que demoram mais tempo para serem resolvidas e avaliar formas de otimização.
- Analisar os autores que têm tempos de resolução mais altos e verificar se há gargalos no processo.

Análise das Issues no Processo de Arquitetura de Software

- Issues de Arquitetura de Software:
As questões relacionadas à arquitetura de software apresentam um tempo de resolução moderado, indicando que elas frequentemente exigem uma análise técnica mais aprofundada e criteriosa antes de serem implementadas. Isso sugere que, embora sejam importantes, as decisões podem demandar mais tempo devido à necessidade de um planejamento detalhado, que leve em consideração a escalabilidade, a manutenção e a eficiência do sistema a longo prazo.
- Issues de Padrões e Estilos Arquiteturais:
As questões relacionadas a padrões e estilos arquiteturais seguem um ritmo de resolução similar ao das questões de arquitetura de software, o que indica que as decisões estruturais dentro do projeto são feitas de maneira eficiente e organizada. A adoção de padrões bem estabelecidos contribui para uma solução mais coesa e alinhada com as melhores práticas do setor, facilitando a comunicação e a implementação do projeto.
- Issues de Padrões de Projeto:
As questões ligadas a padrões de projeto tendem a ser mais complexas e demoram mais para serem resolvidas. Isso pode ser um reflexo de uma maior complexidade técnica ou de uma priorização mais baixa no fluxo de trabalho. Quando essas questões não são resolvidas rapidamente, podem afetar diretamente a produtividade do time de desenvolvimento, retardando a evolução do projeto.

Resumo Final

O relatório sugere que o time do Flutter tem um bom desempenho na resolução de issues, com alta eficiência e baixa taxa de atraso. Há oportunidades de melhoria na variação do tempo de resolução entre os autores, mas no geral, os dados indicam um processo bem otimizado.