

---

# **Documentação de Projeto**

**para o sistema**

## **SET (Software Estimation Tool)**

**Versão 1.0**

Projeto de sistema elaborado pelo aluno **Inácio Moraes da Silva** e apresentado ao curso de **Engenharia de Software** da **PUC Minas** como parte do Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo e orientação acadêmica do professor **Cleiton Silva Tavares**.

**1 de agosto 2025**

# Tabela de Conteúdo

<b>1. Introdução</b>	<b>1</b>
1.1 Público-Alvo e Natureza CLI	1
<b>2. Modelos de Usuário e Requisitos</b>	<b>1</b>
2.1 Descrição de Atores	1
2.2 Modelos de Usuários (Personas)	2
2.2.1 Persona 1: Scrum Master	2
2.2.2 Persona 2: Desenvolvedor Sênior	4
2.2.3 Persona 3: Product Owner	6
2.3 Modelo de Casos de Uso e Histórias de Usuários	7
2.3.1 Diagrama de Casos de Uso	8
2.3.2 Histórias de Usuários	9
2.4 Diagrama de Sequência do Sistema e Contrato de Operações	12
2.4.1 DSS 01: Configurar Sistema	12
2.4.2 DSS 02: Validar Configuração	13
2.4.3 DSS 03: Estimar Tarefa Individual	15
2.4.4 DSS 04: Processar Lote de Tarefas	16
2.4.5 DSS 05: Sincronizar com GitHub	18
2.4.6 DSS 06: Exportar Dados	20
2.4.7 DSS 07: Analisar Histórico	21
<b>3. Modelos de Projeto</b>	<b>23</b>
3.1 Diagrama de Classes	23
3.1.1 Pacote AI (internal/ai)	23
3.1.2 Pacote Estimator (internal/estimator)	25
3.1.3 Pacote GitHub (internal/github)	27
3.1.4 Pacote Storage (internal/storage)	28
3.1.5 Pacote Config (internal/config)	30
3.1.6 Padrões de Design Aplicados	31
3.2 Diagramas de Sequência	32
3.2.1 Configurar Sistema	32
3.2.2 Estimar Tarefa Individual	34
3.2.3 Estimar em Lote	37
3.2.4 Sincronizar Dados GitHub	40
3.3 Diagramas de Comunicação	44
3.3.1 Diagrama de Comunicação - Estimar Tarefa	44
3.4 Arquitetura	45

3.4.1 Diagrama de Contexto (C4 Level 1)	45
3.4.2 Diagrama de Contêineres (C4 Level 2)	46
3.4.3 Padrões Arquiteturais Adotados	46
3.4.4 Tecnologias e Frameworks	47
3.4.5 Fluxo de Dados e Processamento	47
3.4.6 Estratégias de Performance	48
3.4.7 Segurança e Configuração	48
3.5 Diagramas de Estados	48
3.5.1 Diagrama de Estados - Estimativa	49
3.6 Diagrama de Componentes e Implantação.	50
<b>4. Projeto de Interface com Usuário</b>	<b>52</b>
4.1 Ajuda	52
4.2 Configuração Inicial	54
4.3 Validando a Configuração	54
4.4 Estimativa de Tarefa Única	55
4.5 Processamento em Batch	55
4.6 Inspeção de Dados Históricos	56
4.7 Sincronização GitHub	57
<b>5. Glossário e Modelos de Dados</b>	<b>57</b>
5.1 Glossário de Termos	57
5.2 Modelos de Dados	59
5.3 Diagrama Entidade-Relacionamento	63
<b>6. Casos de Teste</b>	<b>63</b>
6.1 Testes de Aceitação	64
6.1.1 Configuração do Sistema (UC01, UC02, UC08)	64
6.1.2 Estimativa Individual (UC03)	66
6.1.3 Estimativa em Lote (UC 04)	70
6.1.4 Sincronização e Histórico (UC05, UC07)	72
6.1.5 Exportação de Dados (UC06)	75
6.2 Testes de Integração	77
6.2.1 Integração com GitHub API	77
6.2.2 Integração com OpenAI API	78
<b>7. Cronograma e Processo de Implementação</b>	<b>80</b>
7.1 Cronograma de Implementação	80
7.2 Processo de Implementação	83
<b>8. Post-mortem</b>	<b>84</b>
8.1 Experiência Positivas	84
8.2 Desafios e Soluções	84

8.3 Lições Aprendidas	85
8.4 Melhorias Futuras	85
8.5 Repositório do Trabalho	86

## Histórico de Revisões

Nome	Data	Razões para Mudança	Versão
Inácio Moraes	10/09/2025	Documentação inicial	0.1
Inácio Moraes	15/10/2025	Inclusão de diagramas	0.2
Inácio Moraes	04/11/2025	Atualização do Doc com base no que foi desenvolvido	0.3
Inácio Moraes	23/11/2025	Postmortem + Revisão final	1.0

# 1. Introdução

Este documento agrega: 1) a elaboração e revisão de modelos de domínio e 2) modelos de projeto para o sistema **SET** (Software Estimation Tool). A referência principal para a descrição geral do problema, domínio e requisitos do sistema é o Documento de Visão que descreve a visão de domínio do sistema.

O SET é uma ferramenta **CLI** (Command Line Interface) inteligente desenvolvida em **Go** que utiliza Inteligência Artificial para revolucionar o processo de estimativa de esforço em projetos de software. A escolha da linguagem Go proporciona alta performance, facilidade de distribuição multiplataforma através de binários únicos, e excelente suporte para integração com APIs REST.

## 1.1 PÚBLICO-ALVO e Natureza CLI

**Importante:** O SET é uma ferramenta de linha de comando (CLI) projetada especificamente para profissionais com background técnico em desenvolvimento de software. O público-alvo inclui:

- Desenvolvedores que trabalham diariamente com terminal e ferramentas CLI
- Scrum Masters com formação técnica e experiência em desenvolvimento
- Product Owners com background como desenvolvedores

Todos os usuários do sistema possuem familiaridade com:

- Terminal/console (Linux, macOS, Windows PowerShell)
- Git e versionamento de código
- Ferramentas CLI comuns (npm, docker, kubectl, etc.)
- Conceitos de desenvolvimento de software

Esta escolha deliberada por CLI permite integração natural no workflow de desenvolvedores, automação via scripts, e uso em pipelines CI/CD, mantendo a ferramenta enxuta e eficiente.

# 2. Modelos de Usuário e Requisitos

## 2.1 Descrição de Atores

**Scrum Master:** Facilitador de metodologias ágeis com formação técnica em desenvolvimento de software, responsável por utilizar a ferramenta para obter estimativas mais precisas durante o planejamento de sprints e auxiliar na tomada de decisões sobre a capacidade da equipe. Confortável com terminal e ferramentas CLI.

**Desenvolvedor:** Membro da equipe de desenvolvimento que utilizará a ferramenta diariamente para validar estimativas próprias e do time, comparando com dados históricos e obtendo sugestões baseadas em IA. Usuário avançado de terminal e ferramentas de linha de comando.

**Product Owner:** Responsável pelo backlog do produto com experiência prévia como desenvolvedor, que usará as estimativas geradas para priorização de features e planejamento de releases. Mantém familiaridade com ferramentas técnicas e CLI para entender melhor a complexidade das tarefas.

## 2.2 Modelos de Usuários (Personas)

Esta seção apresenta os modelos de usuários desenvolvidos por meio da implementação de *personas* que representam os principais usuários da ferramenta CLI SET. As *personas* foram criadas baseando-se nas necessidades identificadas no Documento de Visão e nos padrões observados em equipes ágeis de desenvolvimento.

**Perfil Técnico das Personas:** Todas as personas possuem formação técnica e experiência com desenvolvimento de software, refletindo a natureza CLI da ferramenta. O SET é projetado para profissionais que trabalham diariamente com terminal, versionamento de código (Git), e outras ferramentas de linha de comando. A diferença entre as personas está nos seus papéis atuais dentro da equipe ágil, não no background técnico.

### 2.2.1 Persona 1: Scrum Master

Aspecto	Detalhes
<b>Nome</b>	Carlos Mendes
<b>Idade e Perfil</b>	35 anos, Scrum Master com 8 anos de experiência em metodologias ágeis
<b>Descrição</b>	Carlos é Scrum Master em uma startup de tecnologia com 15 desenvolvedores. Formado em Ciência da Computação, possui certificação PSM I e PSM II. Trabalha com 3 times de Scrum simultaneamente e é apaixonado por métricas e melhoria contínua. Valoriza a transparência e busca constantemente formas de otimizar o processo de desenvolvimento.
<b>Contexto Técnico</b>	<ul style="list-style-type: none"><li>Utiliza Jira, Azure DevOps e GitHub diariamente</li></ul>

- Familiarizado com CLI tools (git, docker, kubectl)
- Experiência com ferramentas de métricas (SonarQube, New Relic)
- Confortável com terminal Linux/Mac

<b>Dores Principais</b>	<ul style="list-style-type: none"><li>• <b>Planning Poker Subjetivo:</b> "As estimativas do planning poker variam muito entre os desenvolvedores e nem sempre refletem a realidade"</li><li>• <b>Falta de Dados Históricos:</b> "Não temos uma base sólida de dados para justificar nossas estimativas para o PO"</li><li>• <b>Tempo Gasto em Reuniões:</b> "Gastamos 2-3 horas por sprint só em estimativas e nem sempre acertamos"</li><li>• <b>Inconsistência Entre Times:</b> "Cada time estima de forma diferente, dificultando comparações"</li></ul>
-------------------------	---

<b>Objetivos</b>	<ul style="list-style-type: none"><li>• Reduzir tempo de reuniões de Planning em 50%</li><li>• Aumentar precisão das estimativas para &gt; 80%</li><li>• Ter dados concretos para apresentar ao management</li><li>• Melhorar previsibilidade de entrega dos times</li><li>• Identificar padrões de super/subestimação</li></ul>
------------------	--

<b>Cenários de Uso</b>	<p><b>Preparação de Sprint Planning:</b></p> <ul style="list-style-type: none"><li>• <code>set sync --custom-fields</code></li><li>• <code>set batch --file backlog.json --format csv --output sprint-15.csv</code></li><li>• <code>set batch --file backlog.json --format markdown --output planning-report.md</code></li><li>• <code>set batch --file backlog.json --workers 10 --progress</code></li></ul>
------------------------	---

	<p><b>Revisão de Sprint:</b></p> <ul style="list-style-type: none"><li>• <code>set inspect --list --custom --limit 50</code></li><li>• <code>set export --format csv --output team-metrics.csv</code></li><li>• <code>set export --format markdown --output sprint-summary.md</code></li></ul>
--	--

- |                            |   |
|----------------------------|---|
| <b>Métricas de Sucesso</b> | <ul style="list-style-type: none"><li>• Redução de 50% no tempo de planning</li><li>• Acurácia &gt; 80% nas estimativas</li><li>• Velocity mais previsível</li><li>• Menos re-estimativas durante o <i>sprint</i></li></ul> |
|----------------------------|---|

### 2.2.2 Persona 2: Desenvolvedor Sênior

Aspecto	Detalhes
<b>Nome</b>	Ana Rodrigues
<b>Idade e Perfil</b>	29 anos, Desenvolvedor Full Stack Sênior com 6 anos de experiência
<b>Descrição</b>	Ana é desenvolvedora sênior em uma consultoria de software que atende múltiplos clientes. Especialista em React, Node.js e Go. Lidera tecnicamente 2 desenvolvedores juniors e frequentemente é consultada para estimativas de tarefas complexas. Preocupa-se com a qualidade técnica e a precisão das entregas.
<b>Contexto Técnico</b>	<ul style="list-style-type: none"><li>• Expert em Git, GitHub Actions, Docker</li><li>• Usuária avançada de terminal (Zsh, Vim)</li><li>• Experiência com ferramentas CLI (make, npm, go build)</li><li>• Trabalha com múltiplos repositórios simultaneamente</li></ul>

<b>Dores Principais</b>	<ul style="list-style-type: none"><li>• <b>Pressão por Estimativas:</b> "Sempre me pedem para estimar tarefas complexas e a responsabilidade é grande"</li><li>• <b>Esquecimento de Detalhes:</b> "Às vezes esqueço de considerar testes, refactoring ou edge cases"</li><li>• <b>Comparação Difícil:</b> "É difícil lembrar quanto tempo levou uma tarefa similar há 3 meses"</li><li>• <b>Estimativas de Juniores:</b> "Os desenvolvedores juniors sempre subestimam por falta de experiência"</li></ul>
<b>Objetivos</b>	<ul style="list-style-type: none"><li>• Ter base de dados para fundamentar estimativas</li><li>• Ajudar desenvolvedores juniors com referências</li><li>• Melhorar a própria auto calibragem</li><li>• Reduzir ansiedade nas estimativas</li><li>• Documentar padrões de desenvolvimento</li></ul>
<b>Cenários de Uso</b>	<p><b>Estimativa Individual:</b></p> <ul style="list-style-type: none"><li>• <code>set estimate "Implementar auth JWT" --labels backend,security --show-similar</code></li><li>• <code>set estimate "Criar API REST" --labels backend --similar --output json</code></li><li>• <code>set estimate "Refatorar módulo" --description "Melhorar arquitetura" --labels refactor</code></li></ul> <p><b>Análise de Dados:</b></p> <ul style="list-style-type: none"><li>• <code>set inspect --list --custom</code></li><li>• <code>set sync --custom-fields</code></li><li>• <code>set export --format json --output my-estimates.json</code></li></ul>
<b>Métricas de Sucesso</b>	<ul style="list-style-type: none"><li>• Menor variação entre estimado e real</li><li>• Melhoria na mentoria de juniores</li><li>• Redução de stress em estimativas</li></ul>

### 2.2.3 Persona 3: Product Owner

Aspecto	Detalhes
<b>Nome</b>	Roberto Santos
<b>Idade e Perfil</b>	42 anos, Product Owner com background como desenvolvedor (10 anos) e 4 anos em produtos digitais
<b>Descrição</b>	Roberto é PO de um produto SaaS B2B com 50k+ usuários. Começou a carreira como desenvolvedor full-stack, trabalhou 10 anos com código antes de migrar para o time de produtos. Usa sua experiência técnica para fazer estimativas mais realistas e se comunicar melhor com a equipe de engenharia. Mantém contato com código ocasionalmente para entender a complexidade técnica.
<b>Contexto Técnico</b>	<ul style="list-style-type: none"> <li><b>Ex-desenvolvedor:</b> 10 anos de experiência com Python, JavaScript e Go</li> <li><b>Confortável com CLI:</b> Usa terminal diariamente (git, docker, npm)</li> <li><b>GitHub power user:</b> Revisa PRs ocasionalmente, gerencia issues</li> <li><b>Ferramentas de desenvolvimento:</b> Familiarizado com IDEs, CI/CD, ferramentas DevOps</li> </ul>
<b>Dores Principais</b>	<ul style="list-style-type: none"> <li><b>Pressão de Prazos:</b> "O CEO sempre quer saber quando a feature vai estar pronta"</li> <li><b>Falta de Visibilidade:</b> "Não sei se as estimativas são realistas até ser tarde demais"</li> <li><b>Comunicação com Stakeholders:</b> "É difícil explicar por que uma 'simples' feature leva 2 sprints"</li> <li><b>Priorização Sem Dados:</b> "Preciso escolher entre features sem saber o real esforço de cada uma"</li> </ul>
<b>Objetivos</b>	<ul style="list-style-type: none"> <li>Ter previsões confiáveis para planejamento de releases</li> <li>Dados para justificar priorizações</li> </ul>

- Comunicação clara com stakeholders
- Melhor ROI nas decisões de produto
- Reduzir riscos de atraso em releases

**Cenários de Uso****Planejamento de Release:**

- `set batch --file epic-user-management.json --format markdown --output epic-estimate.md`
- `set batch --file q1-features.csv --format csv --output roadmap.csv`
- `set export --format jira --output jira-import.csv`

**Acompanhamento:**

- `set inspect --list`
- `set export --format github --output github-import.csv`
- `set export --format excel --output stakeholder-report.csv`
- `set export --filter feature --format markdown --output features-report.md`

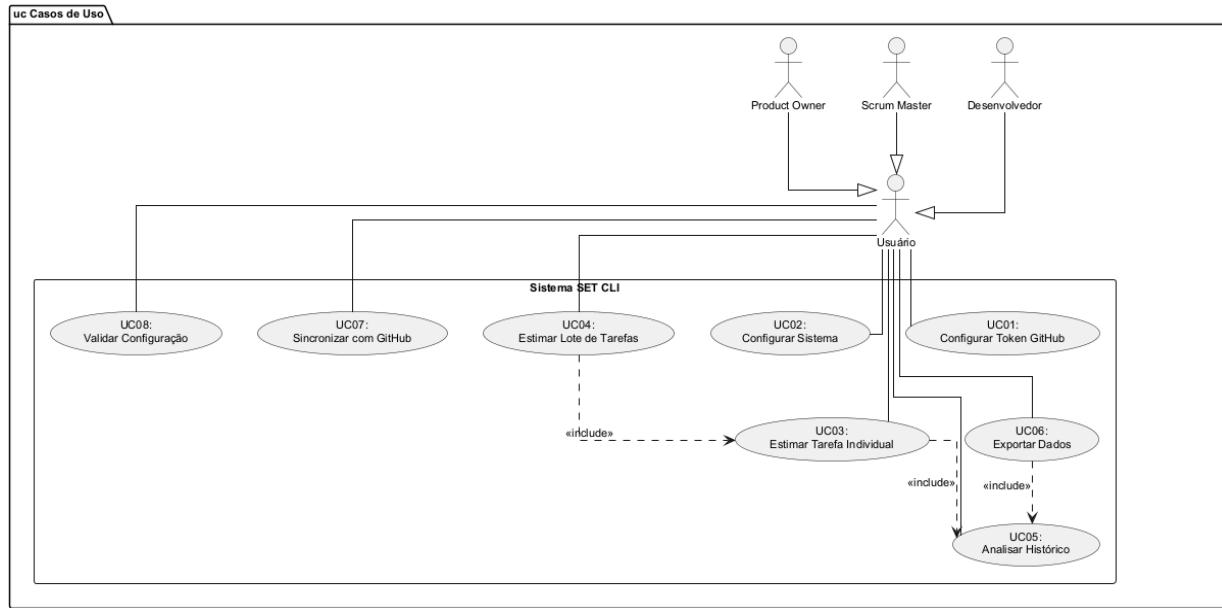
**Métricas de Sucesso**

- Releases entregues no prazo (+90%)
- Melhor comunicação com stakeholders
- Decisões de priorização data-driven
- Redução de scope creep

## 2.3 Modelo de Casos de Uso e Histórias de Usuários

Esta seção apresenta o diagrama de casos de uso e as histórias de usuários que descrevem as principais interações dos atores com o sistema SET CLI. O diagrama ilustra as funcionalidades essenciais e suas relações, enquanto as histórias de usuários detalham os requisitos funcionais na perspectiva de cada tipo de usuário.

### 2.3.1 Diagrama de Casos de Uso



**Figura 2.1 - Casos de Uso**

O diagrama (Figura 2.1) acima apresenta os principais casos de uso do sistema SET CLI implementados. Para fins de organização, utiliza-se identificadores no formato UC#ID, onde UC refere-se a Use Case.

#### *Casos de Uso:*

Código	Caso de Uso	CLI
<b>UC01</b>	Configurar Token GitHub	<i>set configure --github-token</i>
<b>UC02</b>	Configurar Sistema	<i>set configure --initial</i>
<b>UC03</b>	Estimar Tarifa Individual	<i>set estimate "&lt;descrição&gt;"</i>

<b>UC04</b>	Estimar Lote de Tarefas	<code>set batch --file &lt;arquivo&gt;</code>
<b>UC05</b>	Analizar Histórico	<code>set inspect [--list]</code>
<b>UC06</b>	Exportar Dados	<code>set export --format &lt;csv json markdown jira github excel&gt;</code>
<b>UC07</b>	Sincronizar com GitHub	<code>set sync [--custom-fields]</code>
<b>UC08</b>	Validar Configuração	<code>set configure --validate</code>

### 2.3.2 Histórias de Usuários

As histórias de usuário apresentadas a seguir estão organizadas por ator e relacionadas aos casos de uso identificados. Para organização, utiliza-se o formato **US#ID**, onde US refere-se a User Story.

#### *Histórias do Desenvolvedor:*

ID	Caso de Uso	História
<b>US01</b>	UC01, UC02	Como <b>desenvolvedor</b> , eu quero <b>configurar a CLI SET com meu token GitHub</b> para que o sistema possa acessar dados dos meus repositórios e gerar estimativas personalizadas.
<b>US02</b>	UC03	Como <b>desenvolvedor</b> , eu quero <b>estimar uma tarefa específica via CLI</b> para obter uma estimativa baseada em dados históricos e IA, economizando tempo em planning.

<b>US03</b>	UC03	Como <b>desenvolvedor</b> , eu quero <b>ver tarefas similares encontradas durante a estimativa</b> para validar se a sugestão da IA faz sentido com minha experiência.
<b>US04</b>	UC05	Como <b>desenvolvedor</b> , eu quero <b>analisar meu histórico pessoal de estimativas</b> para identificar padrões e melhorar minha precisão ao estimar.
<b>US05</b>	UC03	Como <b>desenvolvedor</b> , eu quero <b>receber um score de confiança com a estimativa</b> para entender a confiabilidade da sugestão da IA.
<b>US06</b>	UC03, UC05	Como <b>desenvolvedor</b> , eu quero <b>usar labels para categorizar tarefas</b> para melhorar a precisão das estimativas baseadas em similaridade.
<b>US07</b>	UC08	Como <b>desenvolvedor</b> , eu quero <b>validar minha configuração do sistema</b> para garantir que o token GitHub e repositório estão corretos antes de começar a usar a ferramenta.

### ***Histórias do Scrum Master:***

ID	Caso de Uso	História
<b>US08</b>	UC01, UC02	Como <b>Scrum Master</b> , eu quero <b>configurar a CLI SET com meu token GitHub e repositório padrão</b> para acessar dados históricos do time e automatizar o processo de estimativas.
<b>US09</b>	UC04	Como <b>Scrum Master</b> , eu quero <b>estimar um lote de tarefas do backlog</b> para preparar a reunião de Planning com dados concretos e acelerar as discussões.

<b>US10</b>	UC05, UC06	Como Scrum Master, eu quero <b>exportar dados históricos de estimativas</b> para análise externa e identificação de padrões do time.
<b>US11</b>	UC05	Como Scrum Master, eu quero <b>inspecionar estatísticas do banco de dados local</b> para entender a quantidade e distribuição de tarefas históricas.
<b>US12</b>	UC04, UC06	Como Scrum Master, eu quero <b>processar múltiplas tarefas em paralelo</b> para obter estimativas rapidamente mesmo com grandes backlogs.
<b>US13</b>	UC07	Como Scrum Master, eu quero <b>sincronizar dados do GitHub incluindo custom fields</b> para ter acesso a Worker Hours e Story Points históricos.

### ***Histórias do Product Owner:***

ID	Caso de Uso	História
<b>US14</b>	UC01, UC02	Como Product Owner, eu quero <b>configurar a CLI SET com acesso ao repositório do produto</b> para poder consultar estimativas e gerar relatórios estratégicos.
<b>US15</b>	UC06	Como Product Owner, eu quero <b>gerar relatórios executivos de estimativas em múltiplos formatos</b> (CSV, JSON, Markdown) para comunicar timelines realistas aos stakeholders e management.
<b>US16</b>	UC04	Como Product Owner, eu quero <b>estimar épicos completos via batch processing</b> para comparar esforço entre diferentes features e priorizar com base em ROI.
<b>US17</b>	UC06	Como Product Owner, eu quero <b>exportar dados em formatos compatíveis com Jira e GitHub Projects</b> para integrar as estimativas ao planejamento estratégico do produto.

<b>US18</b>	UC04, UC06	Como <b>Product Owner</b> , eu quero <b>ver estatísticas agregadas de estimativas em lote</b> (total de horas, distribuição por tamanho, nível de confiança) para tomar decisões informadas sobre releases.
-------------	---------------	---

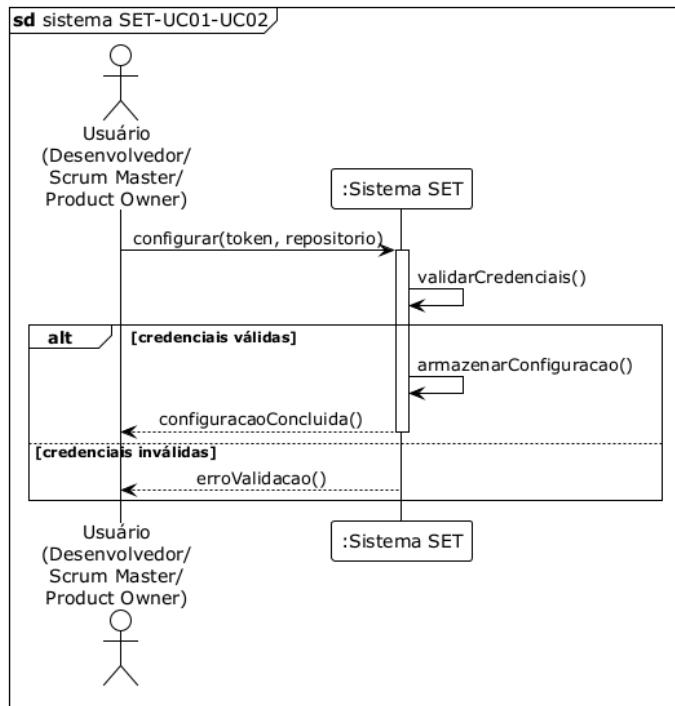
## 2.4 Diagrama de Sequência do Sistema e Contrato de Operações

Esta seção apresenta os Diagramas de Sequência do Sistema (DSS) que descrevem as principais interações entre os atores e o sistema SET CLI em uma visão de caixa-preta. Os DSS mostram apenas as mensagens trocadas entre o ator externo e o sistema, sem revelar a estrutura interna dos componentes.

### 2.4.1 DSS 01: Configurar Sistema

Elemento	Descrição
<b>Operação</b>	configurar(token: string, repositorio: string)
<b>Referências</b>	UC01, UC02, US01, US08, US14
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Usuário possui token GitHub válido</li> <li>• Usuário tem permissões no repositório especificado</li> </ul>
<b>Pós-condições</b>	<ul style="list-style-type: none"> <li>• Token armazenado de forma segura no sistema</li> <li>• Repositório padrão configurado</li> <li>• Sistema pronto para operação</li> </ul>

A Figura 2.2 representa o primeiro contato do usuário com o sistema SET CLI, onde é necessário realizar a configuração inicial para que a ferramenta possa funcionar adequadamente. Este fluxo é fundamental pois estabelece a conexão com o GitHub API e define os parâmetros básicos de funcionamento.



**Figura 2.2** - Diagrama de Sequência do Sistema: Configurar Sistema

### **Descrição do Fluxo:**

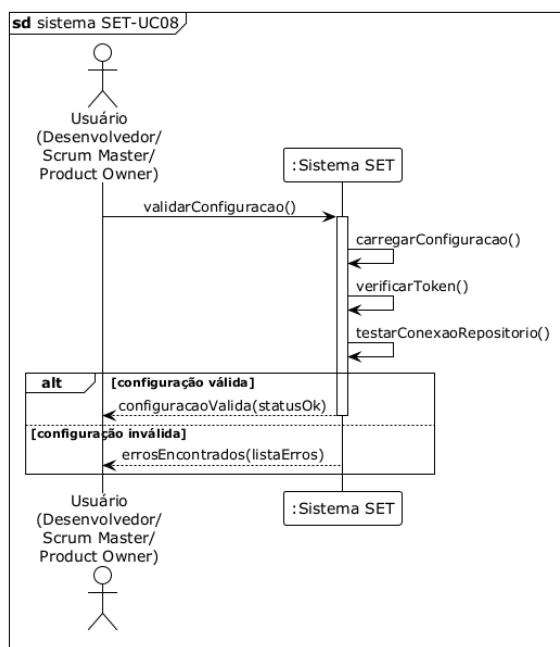
1. O ator (Desenvolvedor/Scrum Master/Product Owner) inicia o comando de configuração através de um wizard interativo (`--initial`) ou via flags individuais (`--github-token`, `--default-repo`, `--ai-provider`, `--ai-key`)
2. O sistema valida o formato do token GitHub fornecido (formato `ghp_...`) e opcionalmente o repositório (padrão `owner/repo`)
3. O sistema armazena a configuração de forma segura no arquivo `~/.set/config.yaml`, incluindo token GitHub, repositório padrão, provedor de IA (OpenAI ou Claude) e suas respectivas chaves de API
4. O sistema confirma que a configuração foi bem-sucedida exibindo um resumo dos valores configurados (com o token mascarado por segurança)

#### **2.4.2 DSS 02: Validar Configuração**

Elemento	Descrição
Operação	<code>validarConfiguracao()</code>

<b>Referências</b>	UC08, US07
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>Configuração previamente realizada</li> <li>Arquivo de configuração existe</li> </ul>
<b>Pós-condições</b>	<ul style="list-style-type: none"> <li>Status de validação retornado ao usuário</li> <li>Erros identificados reportados (se existirem)</li> <li>Conectividade com GitHub verificada</li> </ul>

A Figura 2.3 apresenta a interação para validação da configuração existente do SET CLI, permitindo ao usuário verificar se as credenciais e permissões estão corretas antes de iniciar operações com o GitHub.



**Figura 2.3 - Diagrama de Sequência do Sistema: Validar Configuração**

### **Descrição do Fluxo:**

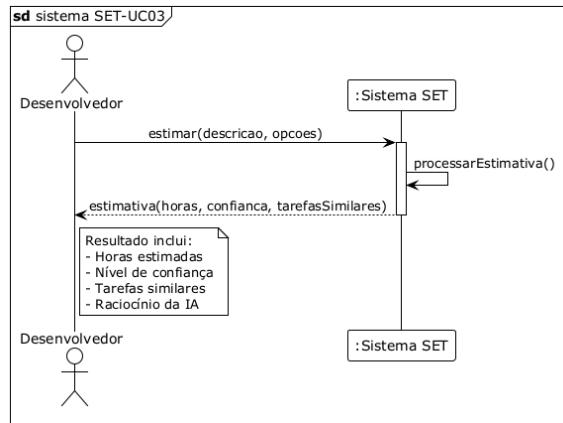
1. O ator executa o comando `set configure --validate` para verificar a integridade da configuração atual

2. O sistema carrega a configuração armazenada em `~/.set/config.yaml` e verifica se o token GitHub está presente
3. O sistema realiza uma chamada à API do GitHub para validar o token, verificando sua validade, scopes (permissões) disponíveis e nome de usuário autenticado
4. O sistema testa a conexão com o repositório padrão configurado (se houver), verificando se o token tem permissões de leitura sobre o repositório
5. O sistema retorna um relatório detalhado incluindo: status de validação (válido ou inválido), nome de usuário autenticado, scopes disponíveis (repo, read:org, etc.), status de acesso ao repositório, e lista de erros específicos caso a validação falhe

#### 2.4.3 DSS 03: Estimar Tarefa Individual

Elemento	Descrição
<b>Operação</b>	estimar(descricao: string, opcoes: OpcoesEstimativa)
<b>Referências</b>	UC03, US02, US03, US05, US06
<b>Pré-condições</b>	<ul style="list-style-type: none"><li>• Sistema configurado</li><li>• Descrição da tarefa fornecida</li></ul>
<b>Pós-condições</b>	<ul style="list-style-type: none"><li>• Estimativa calculada e exibida</li><li>• Nível de confiança determinado</li><li>• Tarefas similares identificadas (se disponíveis)</li><li>• Estimativa registrada no histórico</li></ul>

A Figura 2.4 ilustra a interação para estimativa de uma tarefa individual utilizando IA e dados históricos, representando o caso de uso mais frequente do sistema.



**Figura 2.4** - Diagrama de Sequência do Sistema: Estimar Tarefa Individual

### Descrição do Fluxo:

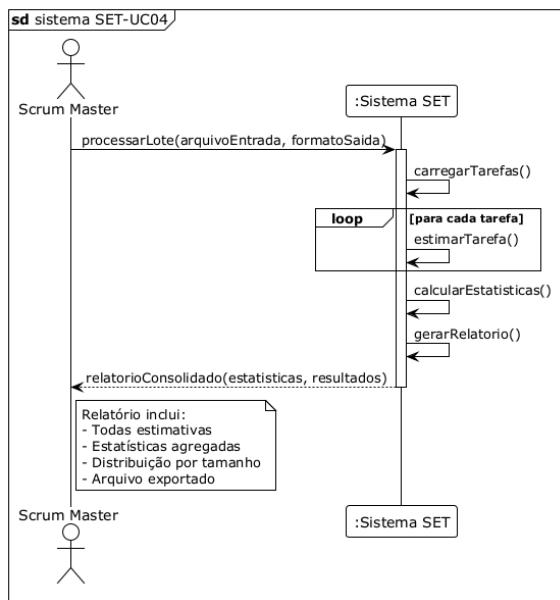
1. O ator (principalmente Desenvolvedor) fornece o título da tarefa e opcionalmente: descrição detalhada (*--description*), labels para categorização (*--labels*), contexto adicional (*--context*), e flags de controle como *--similar* para exibir tarefas similares ou *--no-ai* para usar apenas dados históricos
2. O sistema processa a solicitação através de múltiplas estratégias: busca por tarefas similares no histórico local (BoltDB) usando algoritmos de similaridade textual, consulta ao provedor de IA configurado (OpenAI GPT ou Claude) para análise semântica da tarefa, análise de custom fields do GitHub Projects (size, story points, worker hours), e cálculo estatístico baseado em performance histórica
3. O sistema retorna uma estimativa consolidada contendo: número de horas estimadas (média ponderada entre IA e dados históricos), nível de confiança da estimativa (0-100%), lista de tarefas similares encontradas (título, labels, tempo real de conclusão), raciocínio da IA explicando a estimativa, além da opção de exportar o resultado em múltiplos formatos (text, JSON, CSV)

#### 2.4.4 DSS 04: Processar Lote de Tarefas

Elemento	Descrição
Operação	processarLote(arquivoEntrada: string, formatoSaida: string)
Referências	UC04, US09, US12, US16, US18

<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Sistema configurado</li> <li>• Arquivo de entrada válido (JSON/CSV)</li> <li>• Formato de saída suportado</li> </ul>
<b>Pós-condições</b>	<ul style="list-style-type: none"> <li>• Todas as tarefas estimadas</li> <li>• Estatísticas agregadas calculadas (total horas, distribuição, confiança média)</li> <li>• Arquivo de saída gerado no formato especificado</li> <li>• Estimativas registradas no histórico</li> </ul>

A Figura 2.5 mostra a interação para processamento em lote de estimativas, funcionalidade estratégica utilizada por Scrum Masters durante Sprint Planning e Product Owners no planejamento de releases.



**Figura 2.5** - Diagrama de Sequência do Sistema: Processar Lote de Tarefas

### **Descrição do Fluxo:**

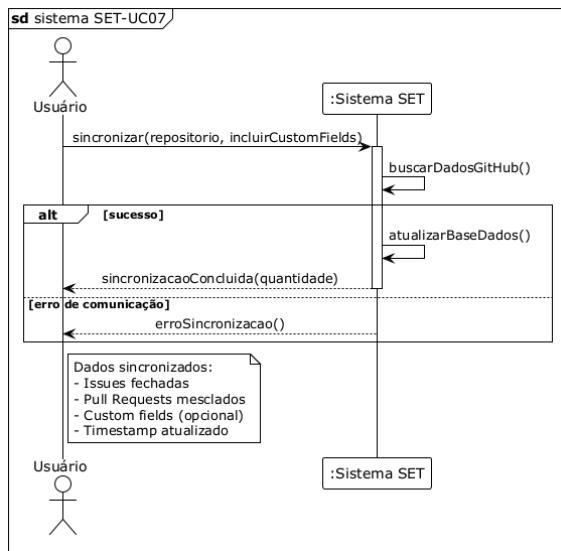
1. O ator (principalmente Scrum Master ou Product Owner) fornece arquivo de entrada em formato JSON ou CSV (`-file tasks.json`) contendo múltiplas tarefas com seus respectivos metadados (id, título, descrição, labels, prioridade, assignee), especifica o formato de saída

- desejado (*--format text|json|csv|markdown*), define o número de workers para processamento paralelo (*--workers 10*), e opcionalmente ativa indicador de progresso (*--progress*)
2. O sistema carrega e valida o arquivo de entrada, processa todas as tarefas em paralelo utilizando pool de workers configurável (padrão 5 workers), aplica o mesmo algoritmo de estimativa usado no comando individual (IA + similaridade + dados históricos) para cada tarefa, coleta métricas de performance durante o processamento, e salva cada estimativa no histórico local (BoltDB)
  3. O sistema gera um relatório consolidado contendo: estatísticas agregadas (total de horas, média, mediana, desvio padrão), distribuição de tarefas por tamanho (pequenas/médias/grandes), nível de confiança médio do lote, breakdown por label/categoria, tempo total de processamento, e lista completa de estimativas individuais
  4. O sistema exporta o relatório no formato especificado: texto formatado para terminal, JSON estruturado para integração com APIs, CSV para análise em planilhas, ou Markdown para documentação, salvando em arquivo (*--output report.md*) ou exibindo no stdout

#### 2.4.5 DSS 05: Sincronizar com GitHub

Elemento	Descrição
<b>Operação</b>	sincronizar(repository: string, incluirCustomFields: boolean)
<b>Referências</b>	UC07, US13
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Sistema configurado com token válido</li> <li>• Repositório acessível</li> <li>• Conectividade com GitHub API</li> </ul>
<b>Pós-condições</b>	<ul style="list-style-type: none"> <li>• Issues fechadas importadas para histórico</li> <li>• Pull Requests mesclados importados</li> <li>• Custom fields (Worker Hours, Story Points) sincronizados se solicitado</li> <li>• Timestamp de última sincronização atualizado</li> <li>• Base de dados local atualizada</li> </ul>

A Figura 2.6 apresenta a interação para sincronização de dados históricos do GitHub com o banco de dados local, permitindo análise offline e consultas rápidas sem consumir rate limit da API.



**Figura 2.6 - Diagrama de Sequência do Sistema: Sincronizar com GitHub**

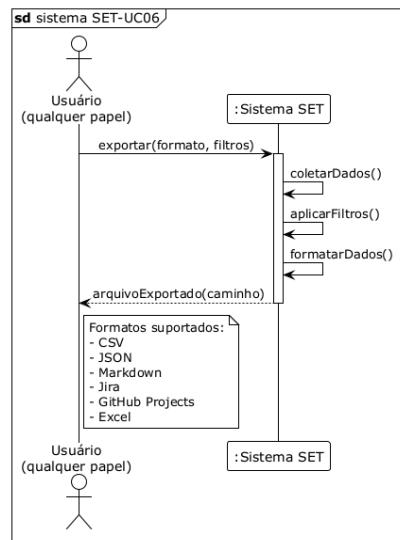
### **Descrição do Fluxo:**

1. O ator executa o comando set sync com opções de controle: sincronização completa (*--full*) ignorando timestamp da última sincronização, sincronização forçada (*--force*) mesmo se executada recentemente, filtros específicos (*--issues-only* ou *--prs-only*), e inclusão de custom fields do GitHub Projects V2 (*--custom-fields*) para obter size, story points, e estimativas
2. O sistema valida a configuração (token e repositório), determina o modo de sincronização (incremental usando timestamp da última sync ou completo desde o início), e estabelece conexão autenticada com a API REST do GitHub v3 e GraphQL API (para custom fields)
3. O sistema busca dados do repositório configurado através de chamadas paginadas à API: issues fechadas com seus metadados (título, descrição, labels, assignee, milestone, created\_at, closed\_at), pull requests mesclados (merged PRs), custom fields do GitHub Projects V2 se solicitado (requer permissões adicionais), e calcula duração real de cada tarefa (closed\_at - created\_at)
4. O sistema persiste os dados no BoltDB local (*~/.set/data.db*) organizados em buckets (issues, prs, custom\_fields), atualiza o timestamp da última sincronização, exibe barra de progresso durante o download, e retorna um relatório de sincronização contendo: quantidade de issues importadas, quantidade de PRs importados, quantidade de custom fields sincronizados, duração total da operação, e próximo horário recomendado para sync (respeitando rate limit)

### 2.4.6 DSS 06: Exportar Dados

Elemento	Descrição
<b>Operação</b>	exportar(formato: FormatoExportacao, filtros: FiltrosExportacao)
<b>Referências</b>	UC06, US10, US12, US15, US17, US18
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Dados disponíveis no sistema</li> <li>• Formato de exportação suportado (CSV, JSON, Markdown, Jira, GitHub, Excel)</li> </ul>
<b>Pós-condições</b>	<ul style="list-style-type: none"> <li>• Arquivo gerado no formato especificado</li> <li>• Dados filtrados conforme critérios</li> <li>• Estrutura compatível com ferramenta de destino (Jira/GitHub)</li> </ul>

A Figura 2.7 ilustra o processo de exportação de dados históricos e estimativas em diferentes formatos para integração com ferramentas de gestão de projetos e análise de dados.



**Figura 2.7** - Diagrama de Sequência do Sistema: Exportar Dados**Descrição do Fluxo:**

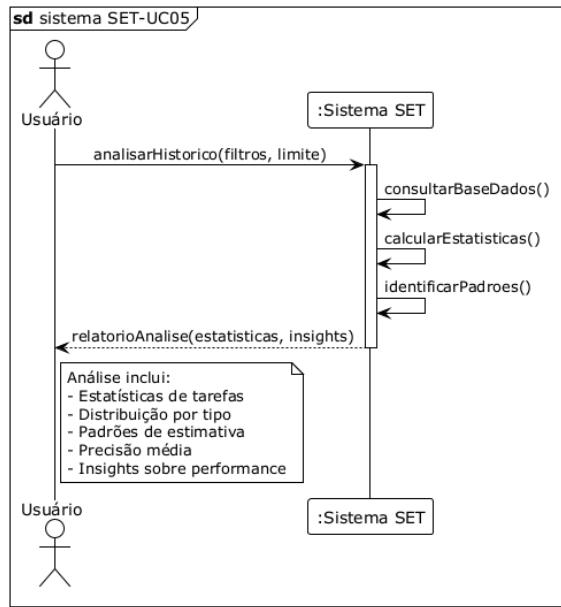
1. O ator (qualquer papel) executa set export especificando formato de saída (*--format csv|json|jira|github|excel|markdown*), arquivo de destino (*--output issues.csv*), e filtros opcionais: filtro por label (*--filter bug*), intervalo de datas (*--date-from 2025-09-19 --date-to 2025-11-16*), e inclusão/exclusão de issues fechadas (*--include-closed*)
2. O sistema consulta o BoltDB local aplicando os filtros especificados: busca issues/PRs no intervalo de datas, filtra por labels quando especificado, inclui ou exclui items fechados conforme configurado, e carrega custom fields associados quando disponíveis
3. O sistema transforma os dados para o formato solicitado aplicando mapeamentos específicos: CSV genérico com todas as colunas disponíveis, JSON estruturado com objetos aninhados completos, formato Jira-compatível seguindo schema de importação do Jira Cloud (Summary, Description, Issue Type, Priority, Labels), formato GitHub Projects seguindo CSV import template do GitHub, formato Excel com colunas enriquecidas incluindo fórmulas e custom fields, ou Markdown table para documentação legível
4. O sistema persiste o arquivo exportado no caminho especificado ou exibe no stdout se *--output* não for fornecido, valida a integridade dos dados exportados, e retorna confirmação contendo: quantidade de registros exportados, tamanho do arquivo gerado, path absoluto do arquivo, e preview das primeiras linhas quando formato é texto

**2.4.7 DSS 07: Analisar Histórico**

Elemento	Descrição
<b>Operação</b>	analisarHistorico(filtros: FiltrosAnalise, limite: number)
<b>Referências</b>	UC05, US04, US06, US10, US11
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Dados históricos disponíveis</li> <li>• Banco de dados local acessível</li> </ul>
<b>Pós-condições</b>	<ul style="list-style-type: none"> <li>• Estatísticas calculadas (total de tarefas, distribuição por tipo, precisão média)</li> <li>• Padrões identificados (tendências de super/subestimação)</li> </ul>

- Insights gerados sobre performance de estimativas
- Relatório exibido ao usuário

A Figura 2.8 mostra a interação para análise e inspeção de dados históricos armazenados localmente, permitindo visualização detalhada de issues, PRs e custom fields sincronizados.



**Figura 2.8** - Diagrama de Sequência do Sistema: Analisar Histórico

### Descrição do Fluxo:

1. O ator executa `set inspect` com diferentes modos de operação: visualizar issue específica (`--issue 123`), visualizar pull request específico (`--pr 45`), listar todos os itens armazenados (`--list`), filtrar apenas itens com custom fields (`--custom`), limitar quantidade de resultados (`--limit 50`), ou exportar em JSON (`--json`)
2. O sistema verifica a existência do banco de dados local (`~/.set/data.db`) e retorna erro caso não exista (instruindo executar `set sync` primeiro), abre conexão read-only com o BoltDB, e executa queries nos buckets apropriados (issues, prs, custom\_fields) conforme o modo solicitado
3. O sistema processa e enriquece os dados recuperados: calcula estatísticas agregadas quando em modo list (total de itens, distribuição por status, breakdown por label), formata datas em formato human-readable, máscara informações sensíveis se houver, organiza custom fields em estrutura hierárquica, e calcula métricas derivadas (duração média, velocidade do time, precisão de estimativas)
4. O sistema exibe os resultados formatados: tabela colorida no terminal para modo list, detalhes completos para inspeção individual (todos os campos, timeline de eventos, custom

fields, links relacionados), JSON estruturado para modo `--json`, estatísticas resumidas (total de tarefas por tipo, distribuição temporal, padrões identificados), e insights sobre performance de estimativas comparando tempo estimado vs. tempo real

## 3. Modelos de Projeto

### 3.1 Diagrama de Classes

Esta seção apresenta os diagramas de classes do sistema SET CLI organizados por pacotes Go (packages), seguindo a estrutura modular da implementação. Cada diagrama mostra as classes, interfaces e relacionamentos específicos de um pacote, refletindo o código efetivamente implementado.

A arquitetura segue os princípios de Clean Architecture e Domain-Driven Design (DDD), com clara separação de responsabilidades entre as camadas de domínio, aplicação, infraestrutura e interface.

#### 3.1.1 Pacote AI (internal/ai)

O pacote internal/ai define a abstração para provedores de inteligência artificial e os tipos relacionados ao processo de estimativa assistida por IA. Este pacote encapsula toda a lógica de integração com modelos de linguagem de grande escala (LLMs), permitindo que diferentes provedores (OpenAI, Claude, modelos locais) sejam utilizados de forma intercambiável através da interface AIProvider. O design orientado a interfaces facilita a extensibilidade e testabilidade do sistema, permitindo mock objects durante testes unitários e a adição de novos provedores de IA sem modificar o código cliente. As estruturas de dados foram projetadas como Value Objects imutáveis para garantir thread-safety durante processamento concorrente de múltiplas estimativas.

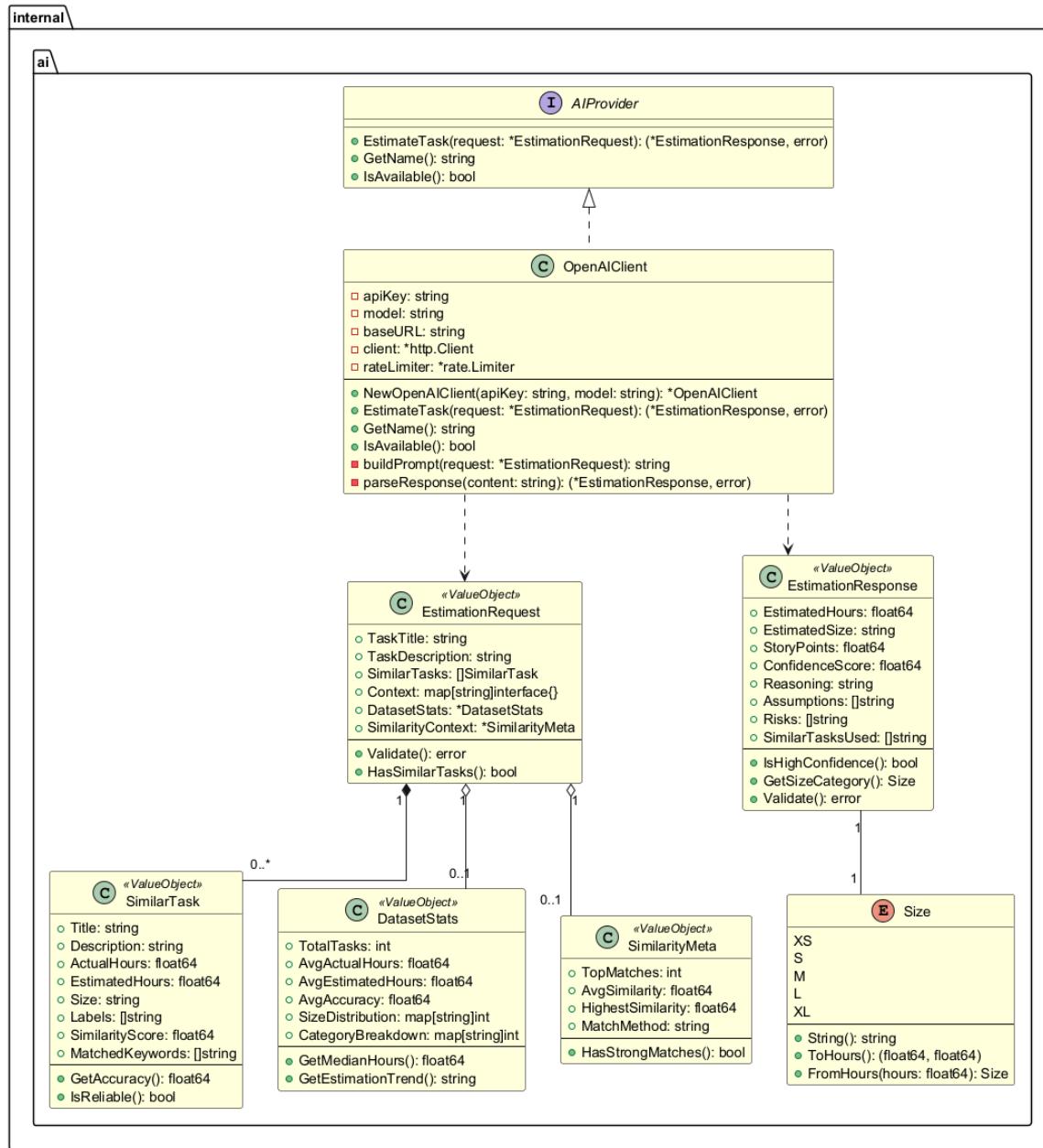


Figura 3.1.1 - Diagrama de Classes: Pacote AI

### Componentes Principais:

- **AIProvider (Interface):** Abstração que permite diferentes implementações de provedores de IA (OpenAI, Claude, modelos locais). Define o contrato para estimação de tarefas.
- **OpenAIclient:** Implementação concreta do AIProvider que integra com a API da OpenAI. Gerencia rate limiting, constrói prompts contextualizados e processa respostas do modelo GPT.

- **EstimationRequest:** Value Object que encapsula todo o contexto necessário para gerar uma estimativa: descrição da tarefa, tarefas similares do histórico, estatísticas do dataset e metadados de similaridade.
- **EstimationResponse:** Value Object que representa o resultado da estimativa gerada pela IA, incluindo horas estimadas, story points, tamanho (XS-XL), nível de confiança, raciocínio detalhado e riscos identificados.
- **Size (Enum):** Enumerações de tamanhos no formato t-shirt sizing (XS, S, M, L, XL), com conversão para faixas de horas.

### 3.1.2 Pacote Estimator (internal/estimator)

O pacote internal/estimator contém a lógica central de estimativa, incluindo o motor de similaridade baseado em TF-IDF e o orquestrador que combina dados históricos com IA. Este é o coração do domínio da aplicação, onde reside a regra de negócio principal: combinar análise histórica com inteligência artificial para gerar estimativas precisas. O SimilarityEngine implementa algoritmos de processamento de linguagem natural (NLP) usando vetorização TF-IDF e similaridade de cosseno para encontrar padrões em tarefas passadas. A classe Estimator atua como um orquestrador (facade), coordenando múltiplos componentes (storage, similarity engine, AI provider) para produzir estimativas calibradas. O design suporta processamento em lote com execução paralela via goroutines, aproveitando as capacidades de concorrência nativa do Go para alta performance.

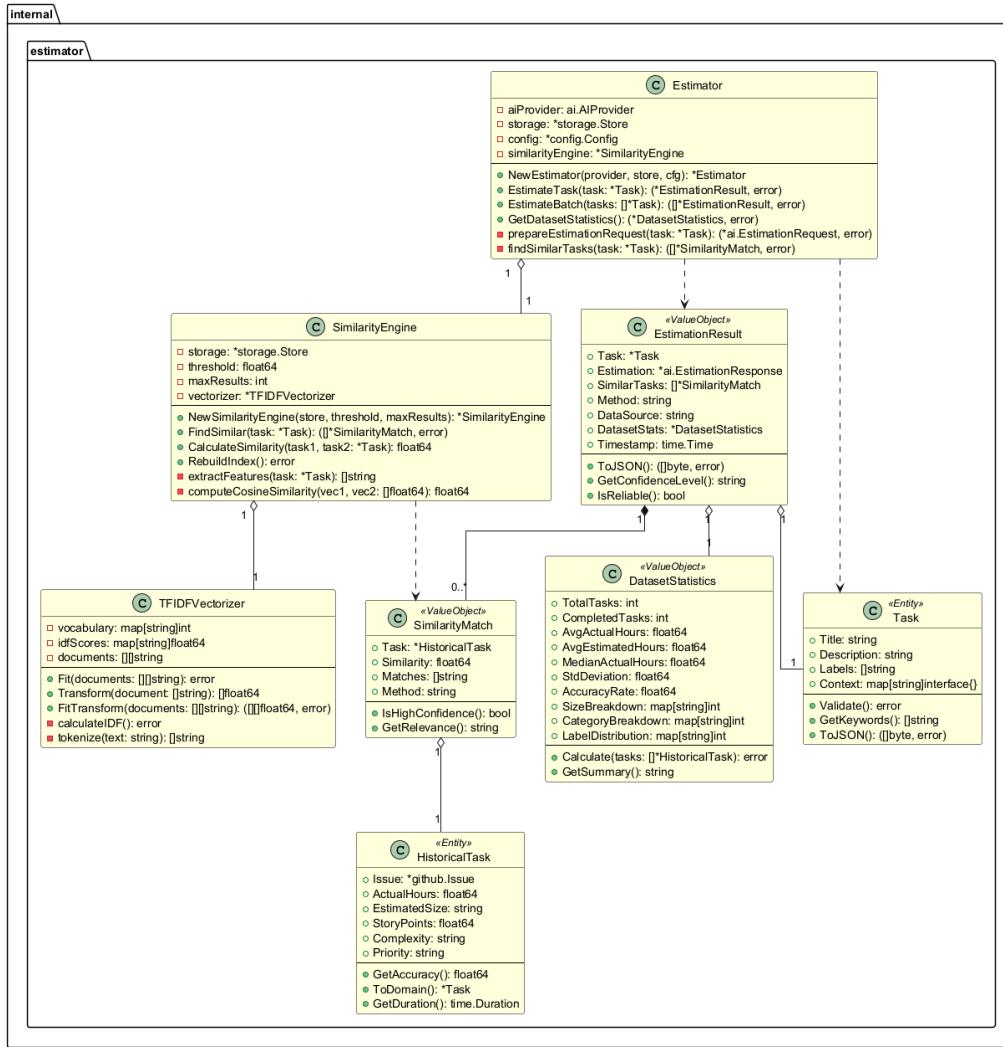


Figura 3.1.2 - Diagrama de Classes: Pacote Estimator

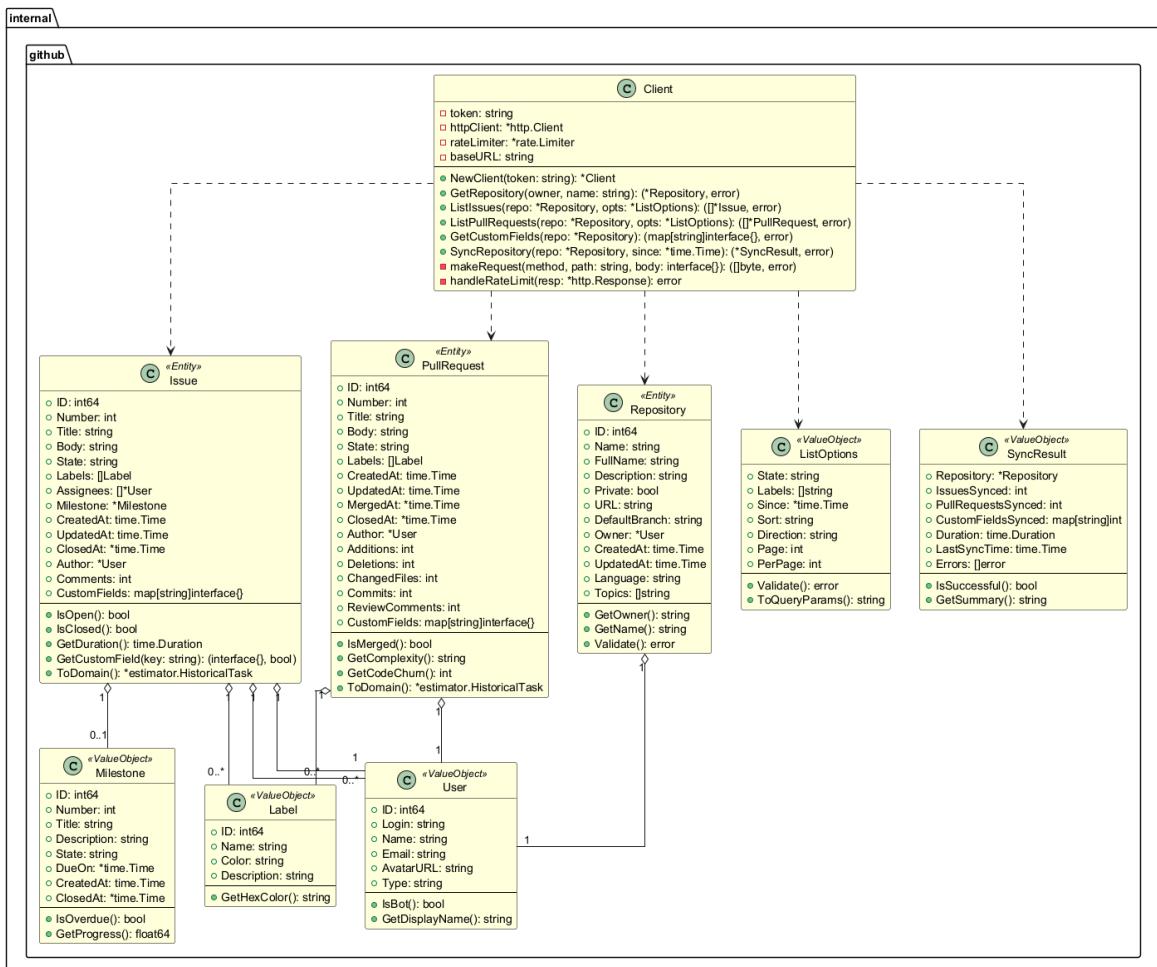
### Componentes Principais:

- **Estimator:** Serviço principal que orquestra o processo de estimativa. Busca tarefas similares no histórico, calcula estatísticas do dataset, solicita estimativa à IA e consolida os resultados.
- **SimilarityEngine:** Motor de busca por similaridade que utiliza vetorização TF-IDF e similaridade de cosseno para encontrar tarefas históricas semanticamente similares à tarefa sendo estimada.
- **TFIDFVectorizer:** Componente de NLP (Natural Language Processing) que implementa o algoritmo Term Frequency-Inverse Document Frequency para transformar texto em vetores numéricos comparáveis.
- **Task:** Entidade que representa uma tarefa a ser estimada, com título, descrição, labels e contexto adicional.
- **HistoricalTask:** Entidade que representa uma tarefa concluída do histórico, com dados reais de esforço (horas, story points, tamanho) para calibração das estimativas.

- **EstimationResult:** Value Object que encapsula o resultado completo do processo de estimativa, incluindo a estimativa da IA, tarefas similares encontradas e estatísticas do dataset.
- **DatasetStatistics:** Value Object com métricas estatísticas agregadas do histórico (média, mediana, desvio padrão, distribuição por tamanho/categoria).

### 3.1.3 Pacote GitHub (internal/github)

O pacote internal/github implementa a integração com a API do GitHub para sincronização de dados históricos de repositórios. Este pacote atua como uma camada de adaptação (Adapter Pattern) entre o domínio interno da aplicação e a API externa do GitHub, isolando detalhes de implementação da API REST/GraphQL e transformando entidades do GitHub (Issues, Pull Requests) em objetos de domínio (HistoricalTask). O cliente HTTP implementa estratégias de resiliência incluindo rate limiting para respeitar os limites da API do GitHub (5000 requisições/hora), retry logic com backoff exponencial, e tratamento robusto de erros de rede. O suporte a GitHub Projects V2 permite sincronização de custom fields (story points, horas estimadas) que enriquecem o dataset histórico e melhoram a precisão das estimativas.



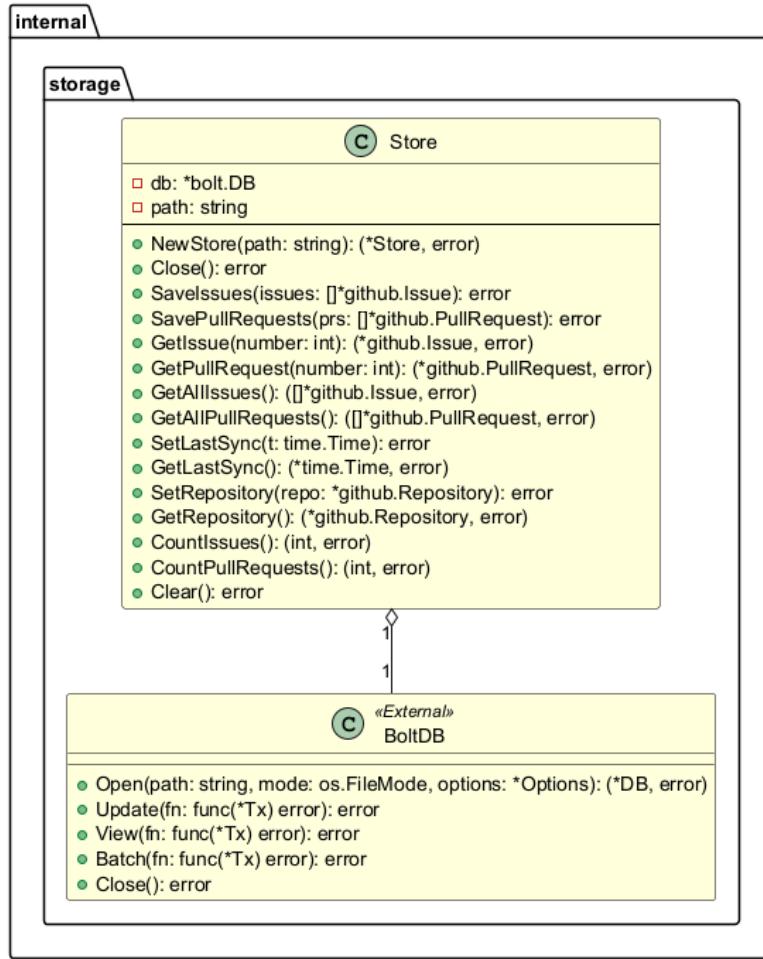
**Figura 3.1.3 - Diagrama de Classes: Pacote GitHub**

## **Componentes Principais:**

- **Client:** Cliente HTTP que gerencia comunicação com as APIs REST e GraphQL do GitHub. Implementa rate limiting, retry logic e tratamento de erros para operações de sincronização.
- **Repository:** Entidade que representa um repositório GitHub com metadados como nome, descrição, linguagem, tópicos e owner.
- **Issue:** Entidade que representa uma issue do GitHub. Contém informações completas incluindo labels, assignees, milestone, datas de ciclo de vida e custom fields do GitHub Projects V2. Pode ser convertida para HistoricalTask.
- **PullRequest:** Entidade que representa um pull request do GitHub. Inclui métricas de complexidade (linhas adicionadas/deletadas, arquivos alterados) que auxiliam na estimativa. Pode ser convertida para HistoricalTask.
- **User:** Value Object representando um usuário do GitHub (autor, assignee).
- **Label, Milestone:** Value Objects com metadados adicionais de organização.
- **ListOptions:** Value Object para parametrizar consultas à API (filtros, ordenação, paginação).
- **SyncResult:** Value Object com resultado de operação de sincronização (quantidade sincronizada, duração, erros).

### **3.1.4 Pacote Storage (internal/storage)**

O pacote internal/storage implementa o padrão Repository para persistência local usando BoltDB. A escolha do BoltDB como banco de dados embarcado foi estratégica para uma aplicação CLI: não requer instalação ou configuração externa, armazena todos os dados em um único arquivo portável, oferece transações ACID completas, e proporciona excelente performance de leitura através de índices B+tree otimizados. O padrão Repository abstrai completamente os detalhes de persistência, permitindo que a camada de domínio trabalhe com interfaces de alto nível sem conhecimento de SQL ou estruturas de armazenamento. A implementação utiliza buckets separados para issues, pull requests e metadados, com serialização JSON para flexibilidade no schema. As operações em lote (Batch) minimizam I/O de disco durante sincronizações grandes de repositórios.



**Figura 3.1.4 - Diagrama de Classes: Pacote Storage**

### **Componentes Principais:**

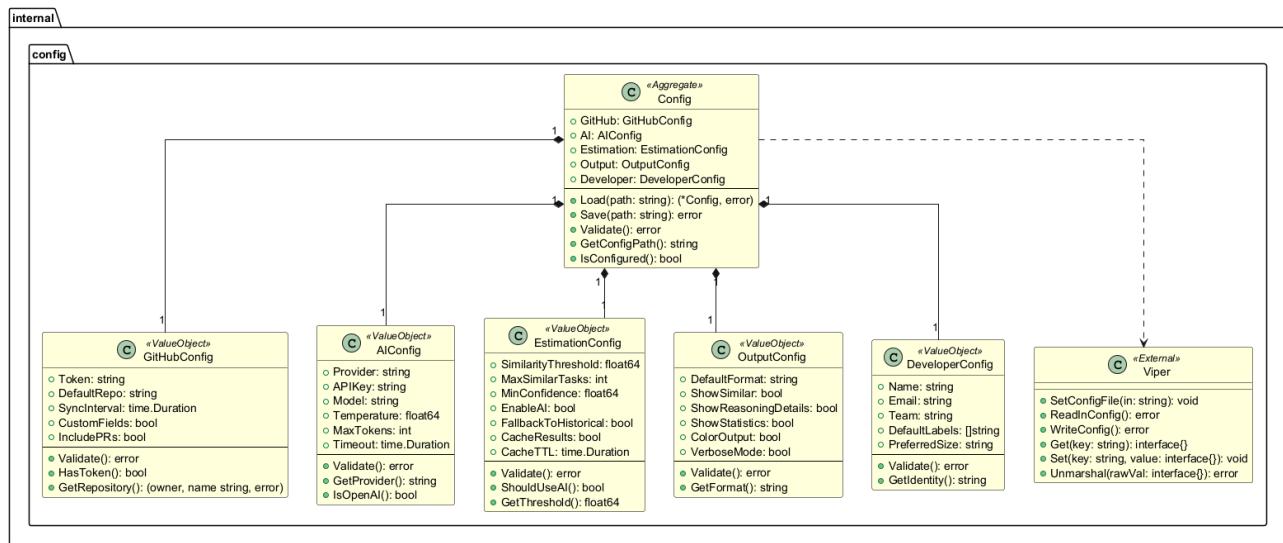
1. **Store:** Implementação do padrão Repository que abstrai o acesso a dados. Gerencia três buckets no BoltDB: issues (issues do GitHub), prs (pull requests) e metadata (informações de sincronização).
2. **BoltDB:** Banco de dados embarcado key-value utilizado para persistência local. Características: zero configuração, ACID compliant, performance otimizada para leitura via B+tree, ideal para aplicações CLI.

### **Operações Principais:**

- Persistência de issues e pull requests com serialização JSON
- Consultas por número ou listagem completa
- Controle de timestamp da última sincronização
- Operações em lote para performance
- Contadores e estatísticas rápidas

### 3.1.5 Pacote Config (internal/config)

O pacote internal/config gerencia todas as configurações do sistema utilizando Viper para suportar múltiplas fontes (arquivo, variáveis de ambiente, flags CLI). A configuração segue uma hierarquia bem definida onde valores mais específicos sobrescrevem valores mais genéricos: defaults do código < arquivo YAML (~/.set.yaml) < variáveis de ambiente < flags de linha de comando. Esta abordagem flexível permite diferentes estratégias de configuração conforme o contexto: desenvolvedores usam arquivos locais, ambientes de CI/CD usam variáveis de ambiente, e usuários finais podem sobrescrever valores via flags sem editar arquivos. A classe Config atua como um Aggregate Root no contexto de DDD, garantindo validação consistente de todas as configurações relacionadas e evitando estados inválidos (por exemplo, API key vazia quando AI está habilitada).



**Figura 3.1.5 - Diagrama de Classes: Pacote Config**

#### Componentes Principais:

- **Config:** Aggregate Root que encapsula todas as configurações do sistema. Arquivo de configuração localizado em `~/.set.yaml`.
- **GitHubConfig:** Configurações de integração com GitHub (token, repositório padrão, intervalo de sincronização, custom fields).
- **AIConfig:** Configurações do provedor de IA (OpenAI), incluindo API key, modelo (GPT-4/GPT-3.5), temperatura e limites.
- **EstimationConfig:** Parâmetros para controle do processo de estimativa (threshold de similaridade, máximo de tarefas similares, confiança mínima, cache).
- **OutputConfig:** Preferências de formatação de saída (formato padrão, verbosidade, cores, estatísticas).
- **DeveloperConfig:** Informações do desenvolvedor (nome, e-mail, equipe, labels padrão).

**Hierarquia de Configuração:** defaults <- arquivo <- variáveis de ambiente <- flags CLI

### 3.1.6 Padrões de Design Aplicados

A modelagem do sistema aplica diversos padrões consolidados:

#### **Domain-Driven Design (DDD):**

- **Entities com identidade:** Task, HistoricalTask, Issue, PullRequest, Repository
- **Value Objects imutáveis:** EstimationRequest, EstimationResponse, SimilarTask, Config, etc.
- **Aggregate Roots:** Config (configurações), SeedData (dados de exemplo)
- **Repositories:** Store (padrão Repository para abstração de persistência)

#### **Clean Architecture:**

- Separação em camadas: Domain (estimator, ai) → Application → Infrastructure (github, storage) → Interface (cmd)
- Dependências apontando para dentro (regras de negócio independentes de frameworks)
- Uso de interfaces para inversão de dependências (AIProvider, Repository implícito)

#### **Padrões GoF:**

**Strategy:** AIProvider permite trocar algoritmo de estimativa

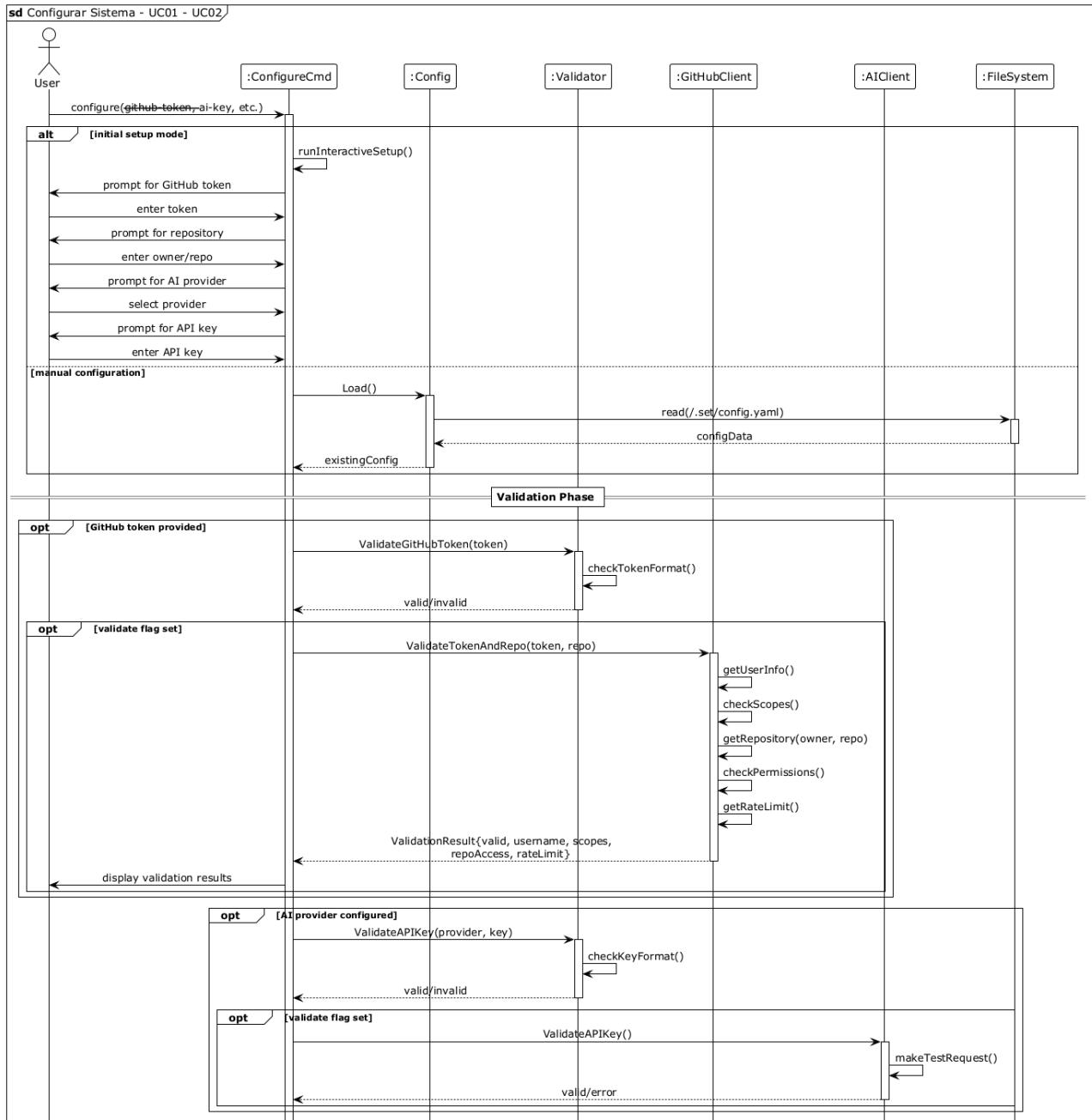
**Factory:** Construtores como NewEstimator, NewClient, NewStore

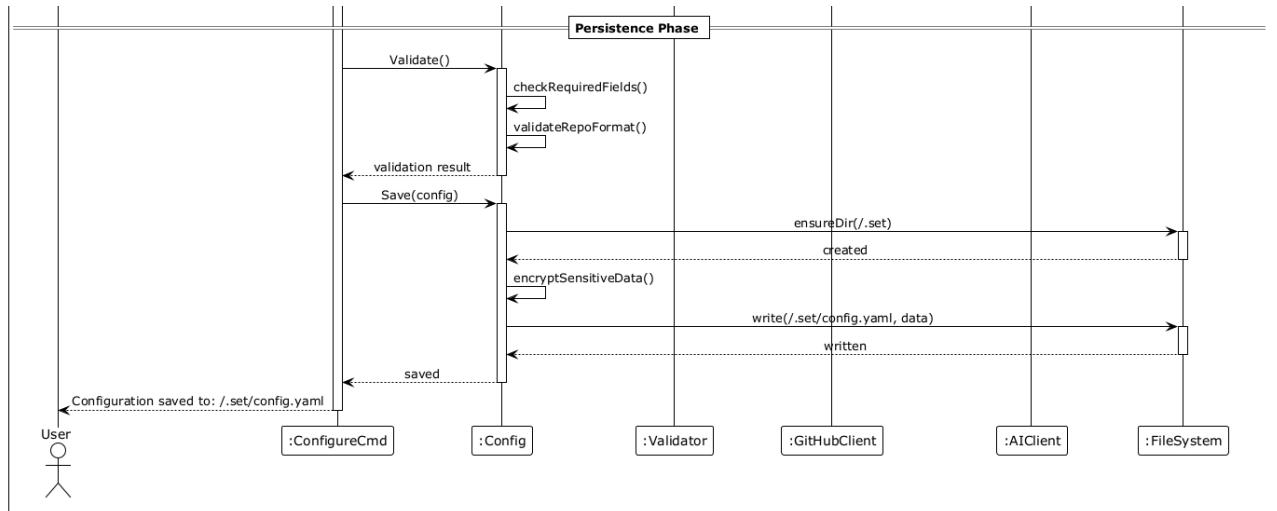
**Adapter:** Conversões ToDomain() adaptam entidades externas (GitHub) para domínio interno

## 3.2 Diagramas de Sequência

Esta seção apresenta os Diagramas de Sequência detalhados que descrevem as interações entre os componentes internos do sistema SET CLI. Diferentemente dos Diagramas de Sequência do Sistema (DSS) apresentados na Seção 2.4 do documento de requisitos, estes diagramas mostram a visão interna (caixa-branca) do sistema, revelando como os componentes colaboram para realizar cada operação.

### 3.2.1 Configurar Sistema





**Figura 3.2 - Diagrama de Sequência: Configurar Sistema**

Este diagrama representa o **processo completo de configuração** do sistema SET CLI, incluindo validação de credenciais e persistência de dados. O fluxo está organizado em três fases principais:

#### Fase de Inicialização:

- Suporta dois modos de configuração: interativo (*--initial*) e manual via flags CLI
- No modo interativo, o sistema guia o usuário através de prompts sequenciais para coletar todas as configurações necessárias
- No modo manual, carrega a configuração existente do arquivo *~/.set/config.yaml* usando a biblioteca Viper

#### Fase de Validação:

- **Validação de Token GitHub:** Verifica formato do token (prefixo *ghp\_*, *gho\_*, etc.) e realiza validação online opcional
- **Validação Completa de Repositório:** Quando a flag *--validate* é utilizada:
  - Obtém informações do usuário autenticado via API do GitHub
  - Verifica escopos do token (repo, public\_repo, read:org, etc.)
  - Testa acesso ao repositório especificado
  - Verifica permissões específicas (leitura de issues, pull requests, projects)
  - Obtém informações de rate limit da API
- **Validação de Provedor de IA:** Verifica formato da API key e realiza teste de conexão com o provedor (OpenAI/Claude)

#### Fase de Persistência:

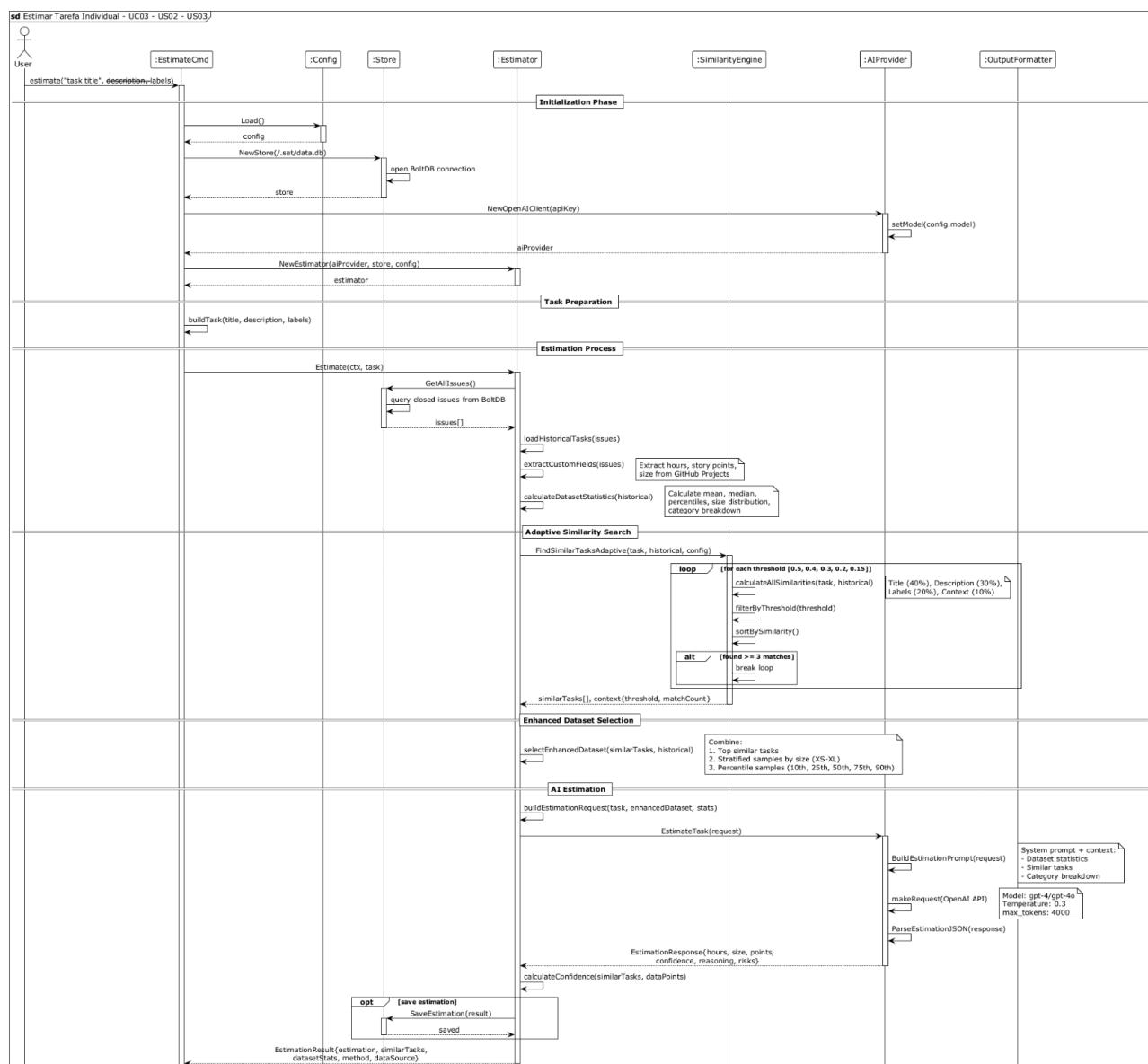
- Válida consistência de todos os campos obrigatórios
- Garante que o diretório *~/.set* existe (criando se necessário)

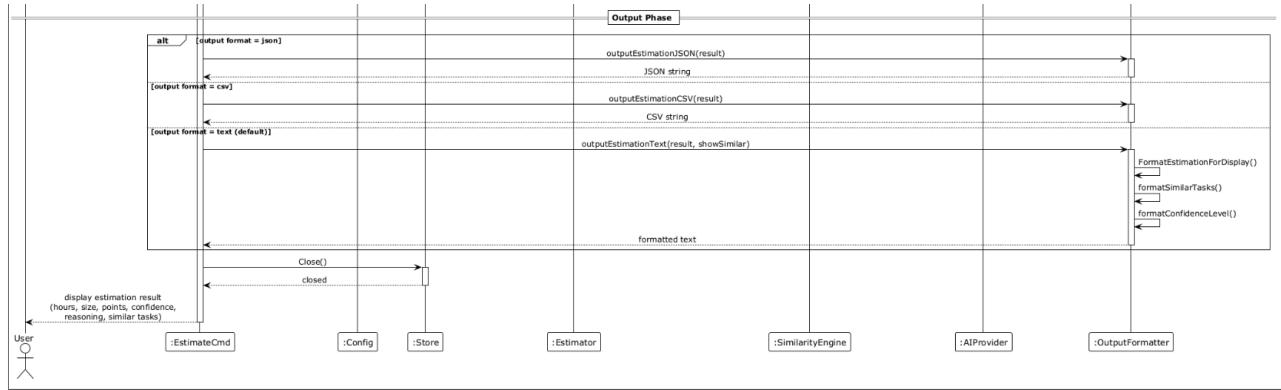
- Criptografa dados sensíveis (tokens, API keys) antes de salvar
- Persiste a configuração em formato YAML no arquivo `~/.set/config.yaml`

### Elementos de Design:

- Utiliza **Validation Pattern** para garantir integridade dos dados antes da persistência
- Implementa **Fail-Fast** ao detectar configurações inválidas
- Segue **Separation of Concerns** com validadores dedicados para cada tipo de credencial
- Aplica **Encryption at Rest** para proteção de dados sensíveis

### 3.2.2 Estimar Tarefa Individual





**Figura 3.3 - Diagrama de Sequência: Estimar Tarefa Individual**

Este diagrama detalha o **algoritmo central de estimativa** do sistema, mostrando como múltiplos componentes colaboram para produzir estimativas precisas. O processo é organizado em cinco fases principais:

#### Fase de Inicialização:

- Carrega configuração do arquivo `~/.set/config.yaml` usando Viper
- Abre conexão com BoltDB (`~/.set/data.db`) para acesso ao histórico
- Inicializa cliente OpenAI com configuração de modelo (GPT-4, GPT-4o, etc.)
- Cria instância do Estimator com configurações de similaridade e limites

#### Fase de Preparação da Tarefa:

- Constrói objeto Task com título, descrição, labels e contexto adicional fornecido pelo usuário
- Validação de entrada para garantir campos obrigatórios

#### Fase de Processamento de Estimativa:

- **Carregamento de Dados Históricos:**
  - Busca todas as issues fechadas do BoltDB
  - Filtra apenas issues com dados de estimativa (horas, story points, ou tamanho)
  - Extrai custom fields de GitHub Projects (Size, Story Points, Worker Hours)
- **Cálculo de Estatísticas do Dataset:**
  - Métricas agregadas: média, mediana de horas
  - Percentis (10°, 25°, 50°, 75°, 90°) para entender distribuição
  - Distribuição por tamanho (XS, S, M, L, XL)
  - Breakdown por categoria (labels) com métricas específicas

#### Fase de Busca Adaptativa por Similaridade:

- Implementa **algoritmo adaptativo de busca** com múltiplos thresholds:

- Tenta threshold 50% (muito similar)
- Tenta threshold 40% (bastante similar)
- Tenta threshold 30% (moderadamente similar - padrão)
- Tenta threshold 20% (algo similar)
- Tenta threshold 15% (pouco similar)
- Para em cada nível quando encontra pelo menos 3 matches (estatisticamente relevante)
- **Cálculo de Similaridade Multifatorial:**
  - Título: 40% do peso (Jaccard similarity)
  - Descrição: 30% do peso (Jaccard similarity)
  - Labels: 20% do peso
  - Contexto/Custom Fields: 10% do peso
- Ordena resultados por score de similaridade (descendente)
- Retorna contexto com threshold usado, número de matches, e similaridade máxima encontrada

### **Fase de Seleção de Dataset Aprimorado:**

- Combina três estratégias de amostragem para dataset robusto:
  - i. **Top Similar Tasks:** Tarefas mais similares encontradas na busca adaptativa
  - ii. **Stratified Sampling:** Amostras de cada bucket de tamanho (XS-XL) para garantir diversidade
  - iii. **Percentile Sampling:** Amostras nos percentis 10°, 25°, 50°, 75°, 90° para capturar toda a distribuição
- Garante mínimo de 10 tarefas no dataset (quando possível) para confiabilidade estatística

### **Fase de Estimativa com IA:**

- **Construção da Requisição:**
  - Inclui título e descrição da tarefa
  - Adiciona estatísticas do dataset completo
  - Inclui dataset aprimorado com tarefas similares e amostras
  - Adiciona contexto de similaridade (threshold usado, matches encontrados)
- **Geração do Prompt:**
  - System prompt define papel do assistente como especialista em estimativas
  - User prompt estruturado com seções:
    - Tarefa a ser estimada
    - Contexto histórico (estatísticas agregadas)
    - Tarefas similares com detalhes (horas reais, tamanho, pontos)
    - Breakdown por categoria/label
- **Chamada à API OpenAI:**
  - Modelo configurável (GPT-4, GPT-4o, GPT-3.5)
  - Temperatura 0.3 para respostas consistentes
  - max\_tokens: 4000 para respostas detalhadas
  - response\_format: json\_object (quando suportado pelo modelo)

- **Parsing da Resposta:**

- Extrai EstimationResponse com:
  - EstimatedHours (float)
  - EstimatedSize (XS/S/M/L/XL)
  - StoryPoints (float)
  - ConfidenceScore (0-1)
  - Reasoning (texto explicativo)
  - Risks (lista de riscos identificados)

- **Cálculo de Confiança:**

- Baseado em: número de tarefas similares, qualidade dos matches, quantidade de dados históricos
- Classificado em: High ( $\geq 70\%$ ), Medium (50-69%), Low (<50%)

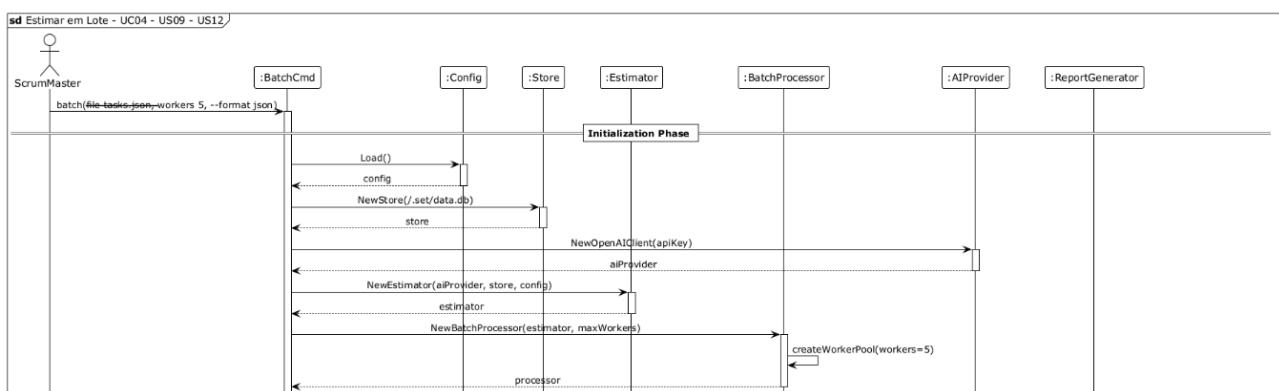
### Fase de Saída:

- Suporta múltiplos formatos:
  - **Text:** Formatação visual com cores, tabelas de tarefas similares, próximos passos
  - **JSON:** Estrutura completa para integração programática
  - **CSV:** Formato tabular para análise em planilhas

### Elementos de Design:

- **Adaptive Algorithm:** Busca por similaridade com ajuste dinâmico de threshold
- **Statistical Sampling:** Estratificação e amostragem por percentis para dataset balanceado
- **Strategy Pattern:** Múltiplos formatadores de saída intercambiáveis
- **Template Method:** Fluxo de estimativa padronizado com variações em cada etapa
- **Chain of Responsibility:** Processamento sequencial de etapas de estimativa

### 3.2.3 Estimar em Lote



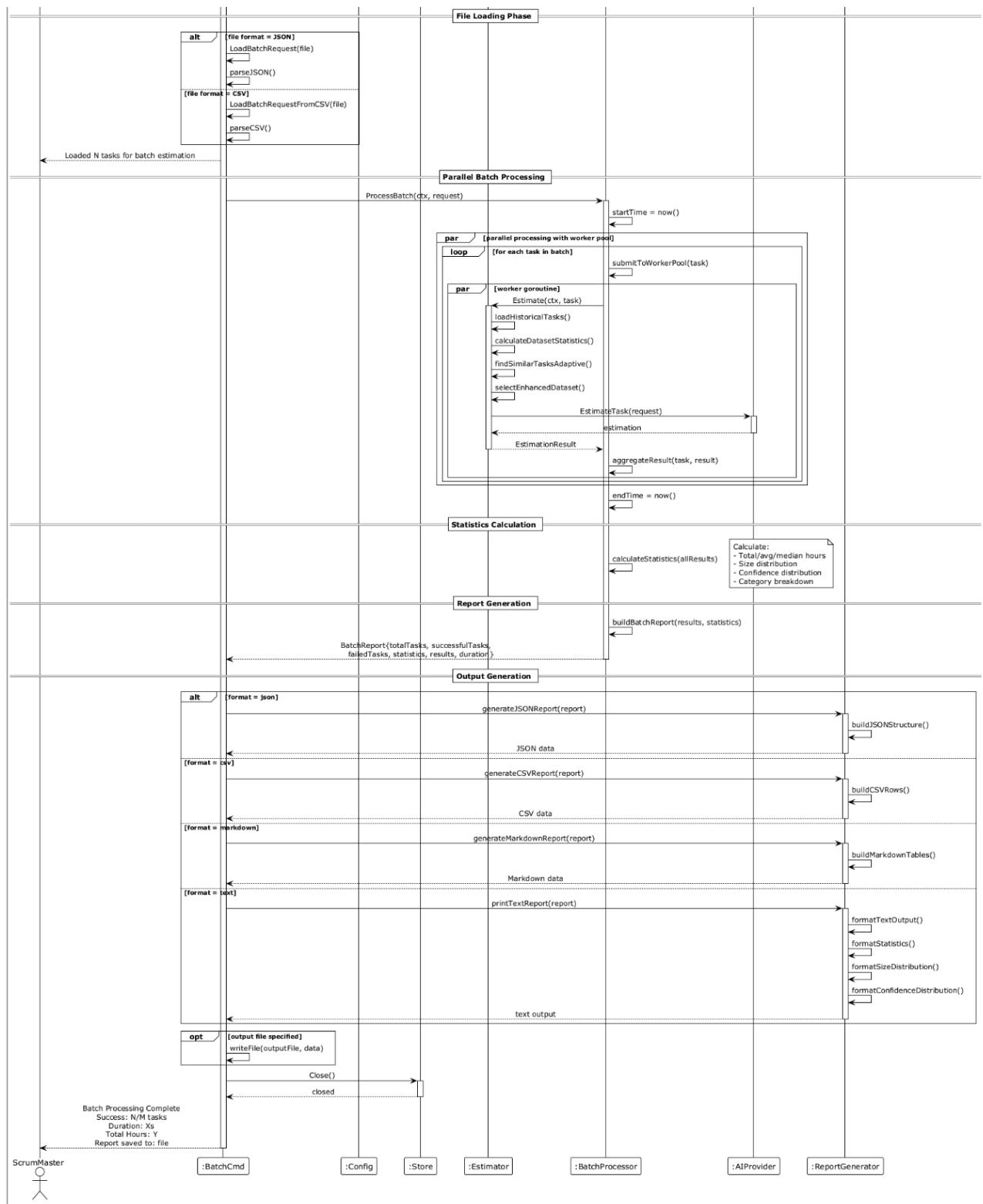


Figura 3.4 - Diagrama de Sequência: Estimar em Lote

Este diagrama ilustra o **processamento paralelo de múltiplas estimativas**, otimizado para alta performance e throughput. O processo é organizado em seis fases:

#### Fase de Inicialização:

- Carrega configuração e inicializa componentes (Store, AIProvider, Estimator)
- Cria BatchProcessor com pool de workers configurável (padrão: 5 workers)
- Pool de workers implementado usando goroutines do Go para paralelismo real

#### Fase de Carregamento de Arquivo:

- Suporta dois formatos de entrada:
  - **JSON**: Estrutura BatchRequest com metadados e array de tasks
  - **CSV**: Formato simples com colunas: id, title, description, labels, priority, assignee
- *Parser* específico para cada formato
- Validação de schema antes do processamento

#### Fase de Processamento Paralelo em Lote:

- Utiliza **Worker Pool Pattern** para processamento concorrente:
  - Cria canal de tarefas (buffered channel)
  - Spawning N goroutines workers
  - Distribui tarefas entre workers via channel
  - Cada worker processa tarefas independentemente
- Cada worker executa o **algoritmo completo de estimativa**:
  - Carregamento de histórico
  - Cálculo de estatísticas
  - Busca adaptativa por similaridade
  - Seleção de dataset aprimorado
  - Estimativa via IA
- **Agregação de Resultados**:
  - Coleta resultados de forma thread-safe
  - Contabiliza sucessos e falhas
  - Armazena timing de cada estimativa

#### Fase de Cálculo de Estatísticas:

- **Métricas Agregadas**:
  - Total de horas estimadas (soma)
  - Média, mediana de horas
  - Range (min-max)
  - Confiança média
- **Distribuições**:
  - Por tamanho (XS-XL): contagem, total de horas, média por tamanho

- Por confiança: High/Medium/Low counts
- Por categoria: baseado em labels das tarefas

### Fase de Geração de Relatório:

- Constrói BatchReport com:
  - Metadados (total tasks, successful, failed, duration)
  - Estatísticas agregadas
  - Resultados individuais de cada tarefa

### Fase de Geração de Saída:

- Suporta quatro formatos de relatório:
  - **Text**: Relatório formatado para terminal com tabelas e seções
  - **JSON**: Estrutura completa com todos os dados para integração
  - **CSV**: Tabela com resultados de cada tarefa
  - **Markdown**: Relatório em markdown com tabelas para documentação
- Permite salvar em arquivo ou exibir no stdout

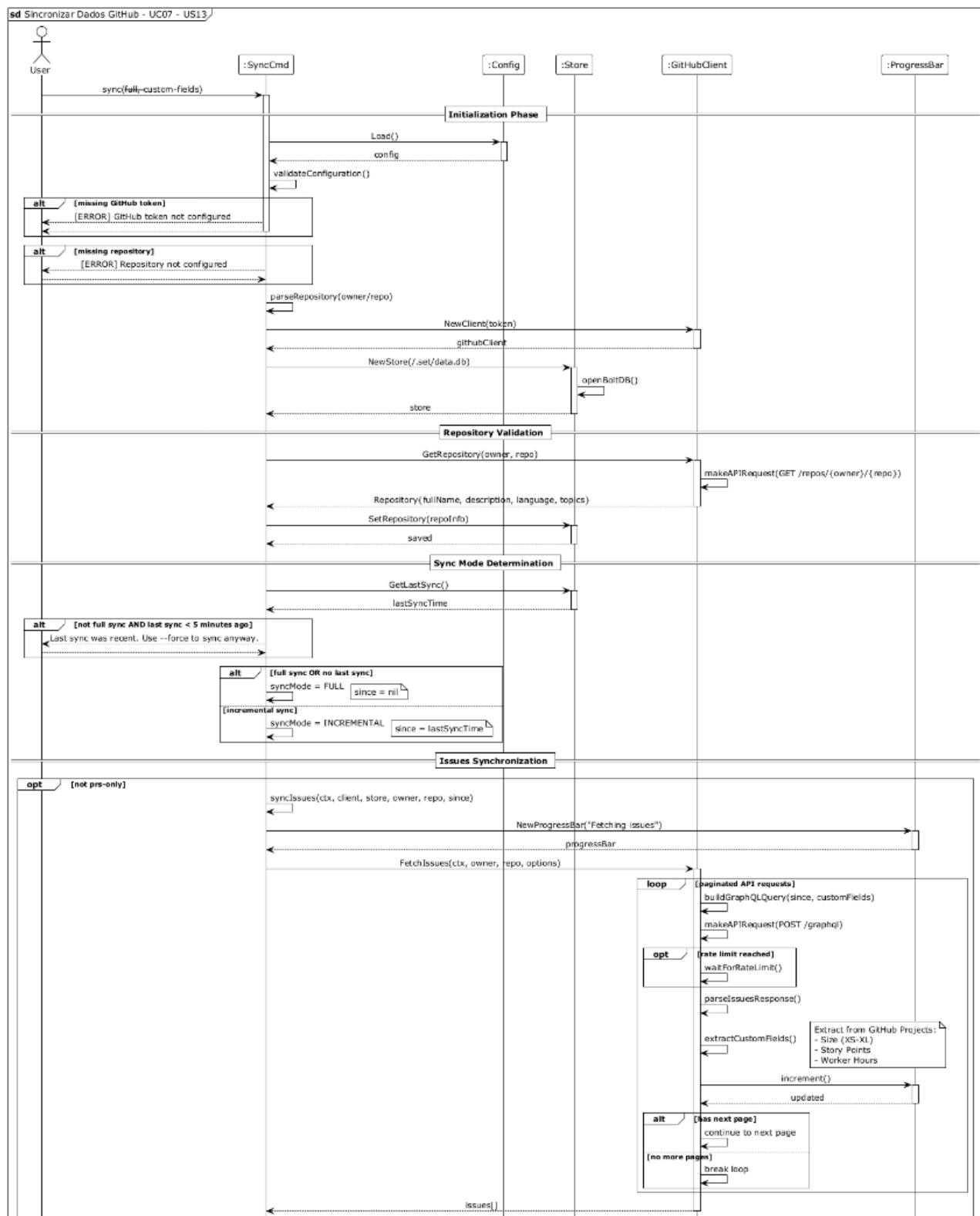
### Elementos de Design:

- **Worker Pool Pattern**: Processamento paralelo eficiente com controle de concorrência
- **Producer-Consumer**: Canal de tarefas alimentado por producer, consumido por workers
- **Map-Reduce**: Processamento paralelo (map) seguido de agregação (reduce)
- **Strategy Pattern**: Múltiplos geradores de relatório intercambiáveis
- **Observer Pattern**: Agregação de resultados conforme workers completam tarefas

### Otimizações de Performance:

- Paralelismo configurável (--workers flag)
- Processamento assíncrono para maximizar throughput
- Batch operations no BoltDB para reduzir I/O
- Reuso de conexão HTTP com OpenAI API
- Progress indicator opcional para feedback visual

#### 3.2.4 Sincronizar Dados GitHub



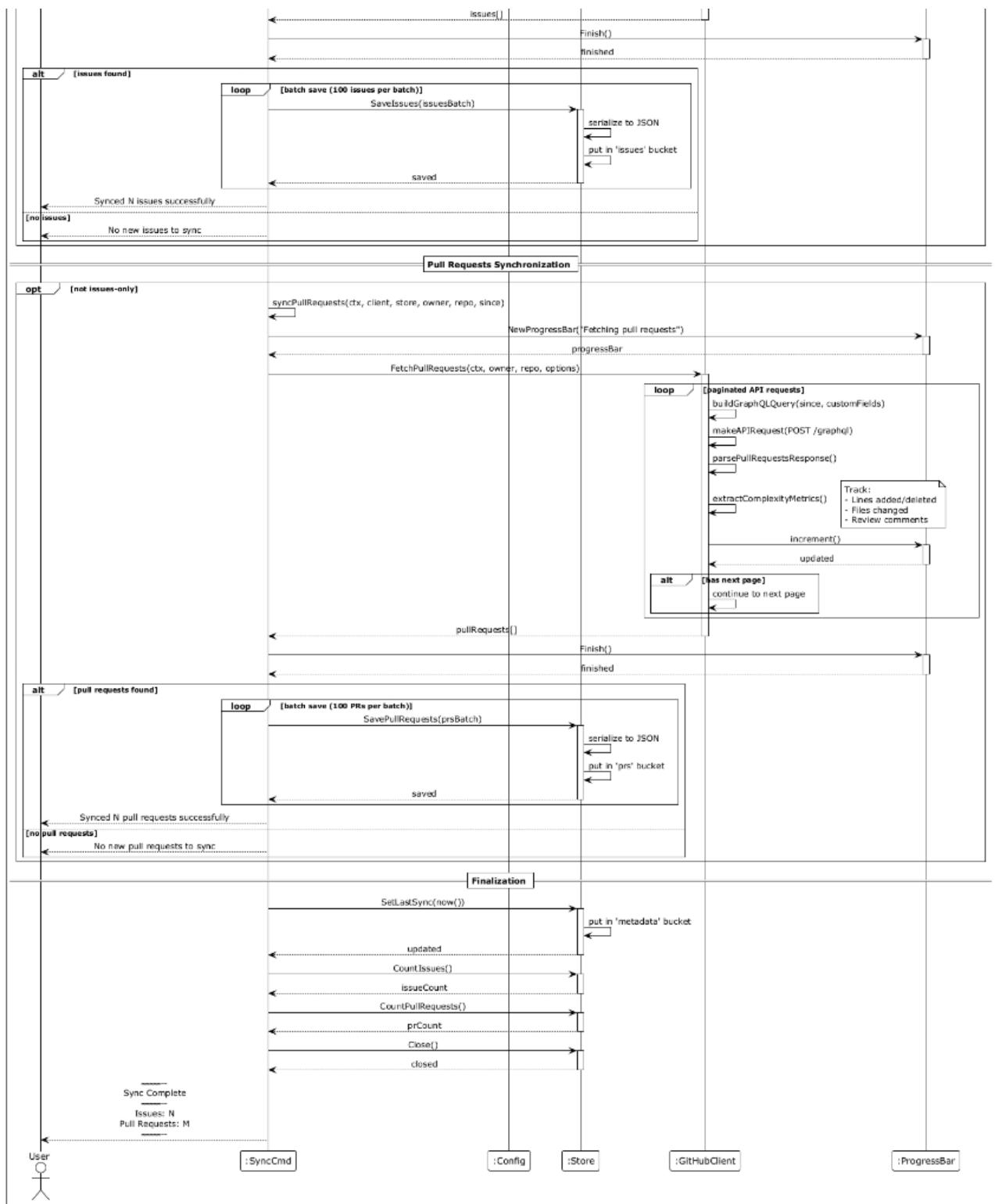


Figura 3.5 - Diagrama de Sequência: Sincronizar Dados GitHub

Este diagrama detalha o **processo de sincronização incremental** de dados do GitHub para armazenamento local, otimizado para eficiência e resiliência. O processo é organizado em seis fases:

### Fase de Inicialização:

- Carrega configuração e valida credenciais necessárias
- Verifica presença de GitHub token e repositório configurado
- Retorna early com mensagem de erro se configuração inválida
- Inicializa GitHubClient com token de autenticação
- Abre/cria banco de dados BoltDB em `~/set/data.db`

### Fase de Validação de Repositório:

- Busca informações do repositório via GitHub REST API
- Valida que repositório existe e token tem acesso
- Armazena metadados do repositório (nome completo, descrição, linguagem, topics)
- Útil para tracking

### Fase de Determinação do Modo de Sincronização:

- **Sync Throttling:** Verifica timestamp da última sincronização
  - Se última sync < 5 minutos atrás e não --force: retorna early (evita requisições desnecessárias)
  - Respeita rate limits da API do GitHub
- **Sync Mode Selection:**
  - **Full Sync** (--full ou primeira sync): *since = nil*, busca todos os dados
  - **Incremental Sync** (padrão): *since = lastSyncTime*, busca apenas items atualizados

### Fase de Sincronização de Issues:

- Cria progress bar visual para feedback ao usuário
- Realiza chamadas paginadas à GitHub GraphQL API:
  - **Paginação:** Utiliza cursors para navegar grandes datasets
  - **Rate Limiting:** Detecta quando rate limit é atingido e espera reset
  - **Retry Logic:** Implementa backoff exponencial em caso de erros temporários
- **Extração de Custom Fields:**
  - Busca campos customizados de GitHub Projects v2
  - Extrai: Size (XS-XL), Story Points, Worker Hours
  - Crucial para qualidade das estimativas
- **Batch Persistence:**
  - Salva issues em lotes de 100 para otimizar I/O do BoltDB
  - Serializa cada issue para JSON
  - Armazena no bucket 'issues' do BoltDB
  - Utiliza transações para garantir atomicidade

### Fase de Sincronização de Pull Requests:

- Processo similar ao de issues, com adições:

- **Métricas de Complexidade:**
  - Linhas adicionadas/deletadas
  - Número de arquivos alterados
  - Contagem de comentários de revisão
- Útil para correlacionar complexidade de código com tempo de desenvolvimento
- Salva em bucket separado 'prs' no BoltDB

### Fase de Finalização:

- Atualiza timestamp de última sincronização no bucket 'metadata'
- Obtém contadores atualizados de issues e PRs
- Fecha conexão com BoltDB
- Exibe resumo formatado com totais sincronizados

### Elementos de Design:

- **Incremental Sync Pattern:** Reduz carga de rede e API buscando apenas deltas
- **Retry with Exponential Backoff:** Resiliência contra falhas temporárias de rede/API
- **Rate Limiting:** Respeita limites da API do GitHub (5000 req/hora autenticado)
- **Batch Processing:** Otimiza operações de I/O agrupando persistências
- **Transaction Pattern:** Garante atomicidade de operações de persistência
- **Pagination Pattern:** Lida com datasets grandes de forma eficiente

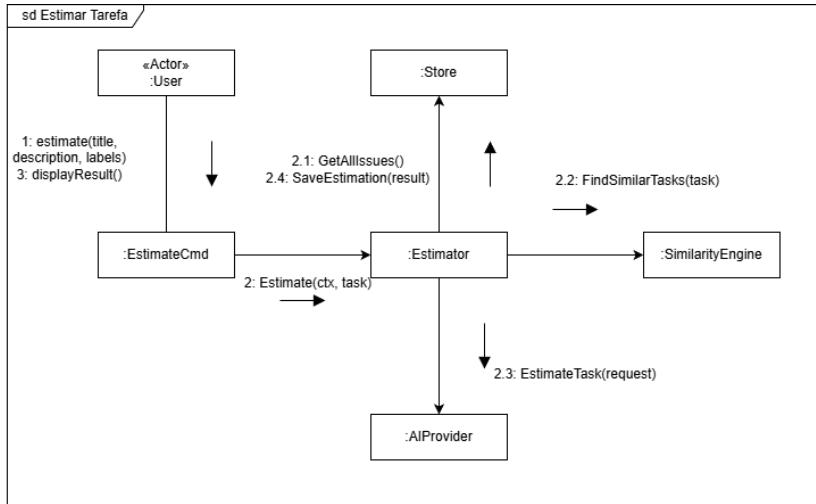
### Otimizações de Performance:

- GraphQL para buscar apenas campos necessários (reduz payload)
- Índices B+tree do BoltDB para queries rápidas
- Serialização JSON eficiente
- Bulk operations para minimizar syscalls

## 3.3 Diagramas de Comunicação

Os diagramas de comunicação complementam os diagramas de sequência, focando na estrutura das relações entre objetos e a ordem das mensagens trocadas.

### 3.3.1 Diagrama de Comunicação - Estimar Tarefa



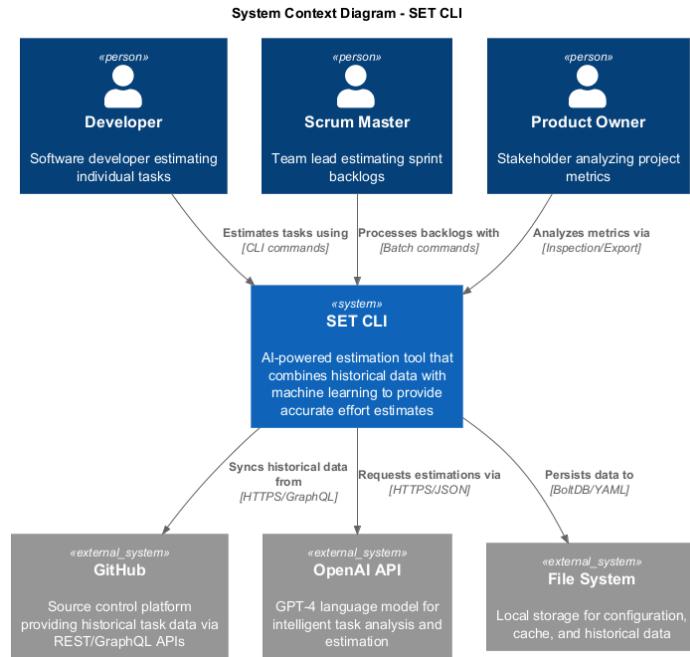
**Figura 3.5 - Diagrama de Comunicação: Estimando uma tarefa**

### 3.4 Arquitetura

Esta seção apresenta a modelagem da arquitetura do sistema SET CLI utilizando o modelo C4 (Context, Containers, Components, Code), que fornece uma abordagem hierárquica para visualizar a arquitetura de software em diferentes níveis de abstração.

A arquitetura do sistema SET CLI foi projetada seguindo os princípios de Clean Architecture, garantindo separação de responsabilidades, baixo acoplamento e alta coesão. O sistema utiliza Go como linguagem principal, aproveitando suas características de performance, concorrência e facilidade de distribuição multiplataforma.

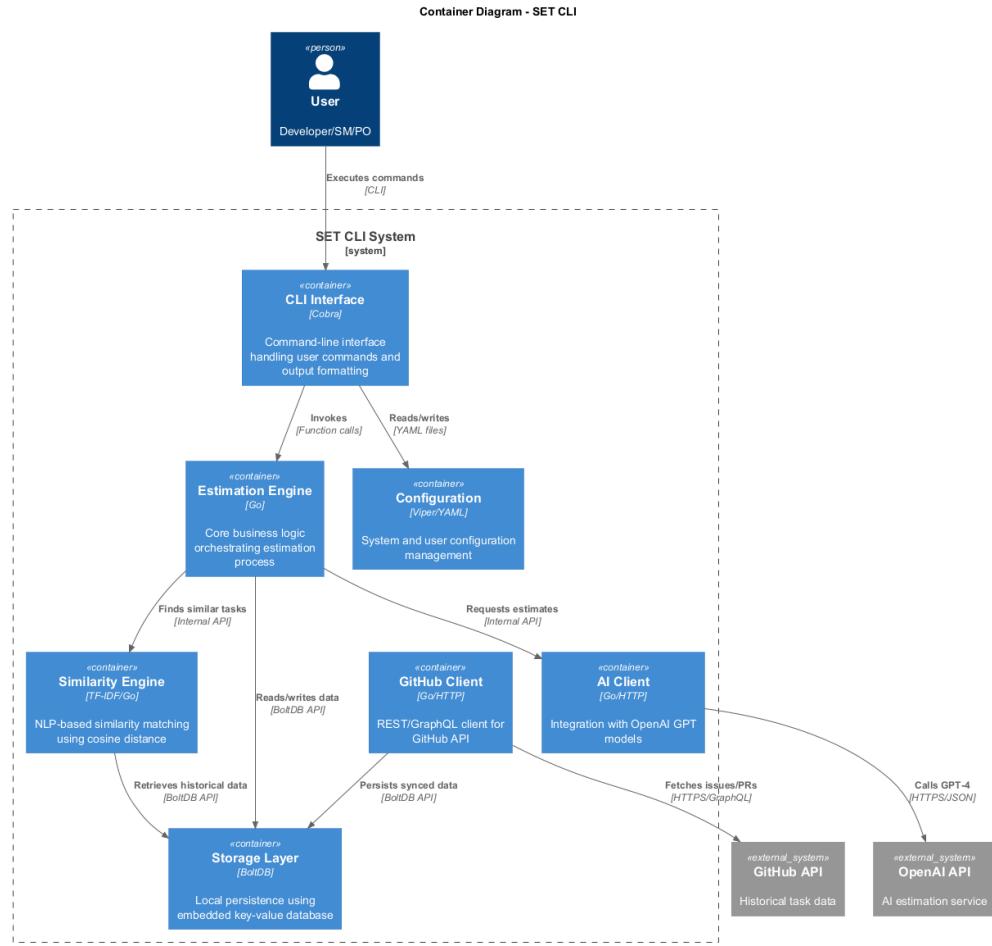
### 3.4.1 Diagrama de Contexto (C4 Level 1)



**Figura 3.10 - Diagrama de Contexto C4**

O diagrama de contexto mostra a visão de mais alto nível do sistema SET CLI, identificando os atores principais (Developer, Scrum Master, Product Owner) e os sistemas externos com os quais interage (GitHub API, OpenAI API, File System).

### 3.4.2 Diagrama de Contêineres (C4 Level 2)



**Figura 3.11 - Diagrama de Contêineres C4**

O diagrama de contêineres decompõe o sistema SET CLI em seus principais componentes executáveis, mostrando como a CLI Interface, Estimation Engine, Similarity Engine, AI Client, GitHub Client, Storage Layer e Configuration trabalham em conjunto para fornecer funcionalidade de estimativa.

### 3.4.3 Padrões Arquiteturais Adotados

**Clean Architecture:** O sistema segue os princípios da Clean Architecture, organizando o código em camadas concêntricas onde as dependências apontam sempre para dentro. As regras de negócio estão isoladas das implementações específicas de infraestrutura.

**Repository Pattern:** Utilizado para abstrair o acesso a dados, permitindo diferentes implementações de persistência (BoltDB para dados locais, potencial extensão para bancos remotos).

**Dependency Injection:** Implementado através de interfaces Go, facilitando testes unitários e flexibilidade na troca de implementações.

**Strategy Pattern:** Aplicado nos algoritmos de estimativa, permitindo diferentes abordagens de cálculo conforme o contexto.

### 3.4.4 Tecnologias e Frameworks

**Linguagem Principal:** Go 1.21+

- **Vantagens:** Performance superior, concorrência nativa com goroutines, facilidade de distribuição (binário único), excelente suporte para CLI
- **Frameworks:** Cobra (CLI), Viper (configuração), BoltDB (persistência)

**Persistência Local:**

- **BoltDB:** Banco de dados embarcado para dados históricos e estimativas
- **JSON/YAML:** Arquivos de configuração e cache estruturado
- **Vantagens:** Zero configuração, performance local, portabilidade

**Integração Externa:**

- **GitHub API v4:** Para coleta de dados de repositórios via GraphQL/REST
- **OpenAI API:** Para análise de linguagem natural e geração de estimativas
- **Rate Limiting:** Implementado para respeitar limites das APIs externas

### 3.4.5 Fluxo de Dados e Processamento

**Estimativa Individual:**

1. CLI recebe comando e parâmetros do usuário
2. Estimation Service processa a descrição da tarefa
3. Similarity Engine busca tarefas similares no histórico local
4. AI Client envia contexto para análise de IA
5. Resultado consolidado é retornado e formatado para exibição

**Análise em Lote:**

1. CLI carrega arquivo de tarefas (JSON/CSV)
2. Core Engine processa cada tarefa usando goroutines para paralelização
3. Resultados são agregados e estatísticas calculadas
4. Export Engine gera arquivos de saída nos formatos solicitados

**Sincronização de Dados:**

1. GitHub Client busca dados incrementais do repositório
2. Cache Manager otimiza requisições repetitivas
3. Repository persiste novos dados no BoltDB local

4. Similarity Engine reindexiza dados para buscas futuras

### 3.4.6 Estratégias de Performance

**Concorrência:** Utilização de goroutines para processamento paralelo de estimativas em lote e requisições de API.

**Caching:**

- Cache em memória para dados frequentemente acessados
- Cache de arquivos para resultados de APIs externas
- TTL configurável para balancear freshness e performance

**Indexação:** Índices otimizados no BoltDB para buscas rápidas de tarefas similares baseadas em keywords e metadados.

### 3.4.7 Segurança e Configuração

**Autenticação:** Tokens GitHub armazenados com criptografia AES-256 local.

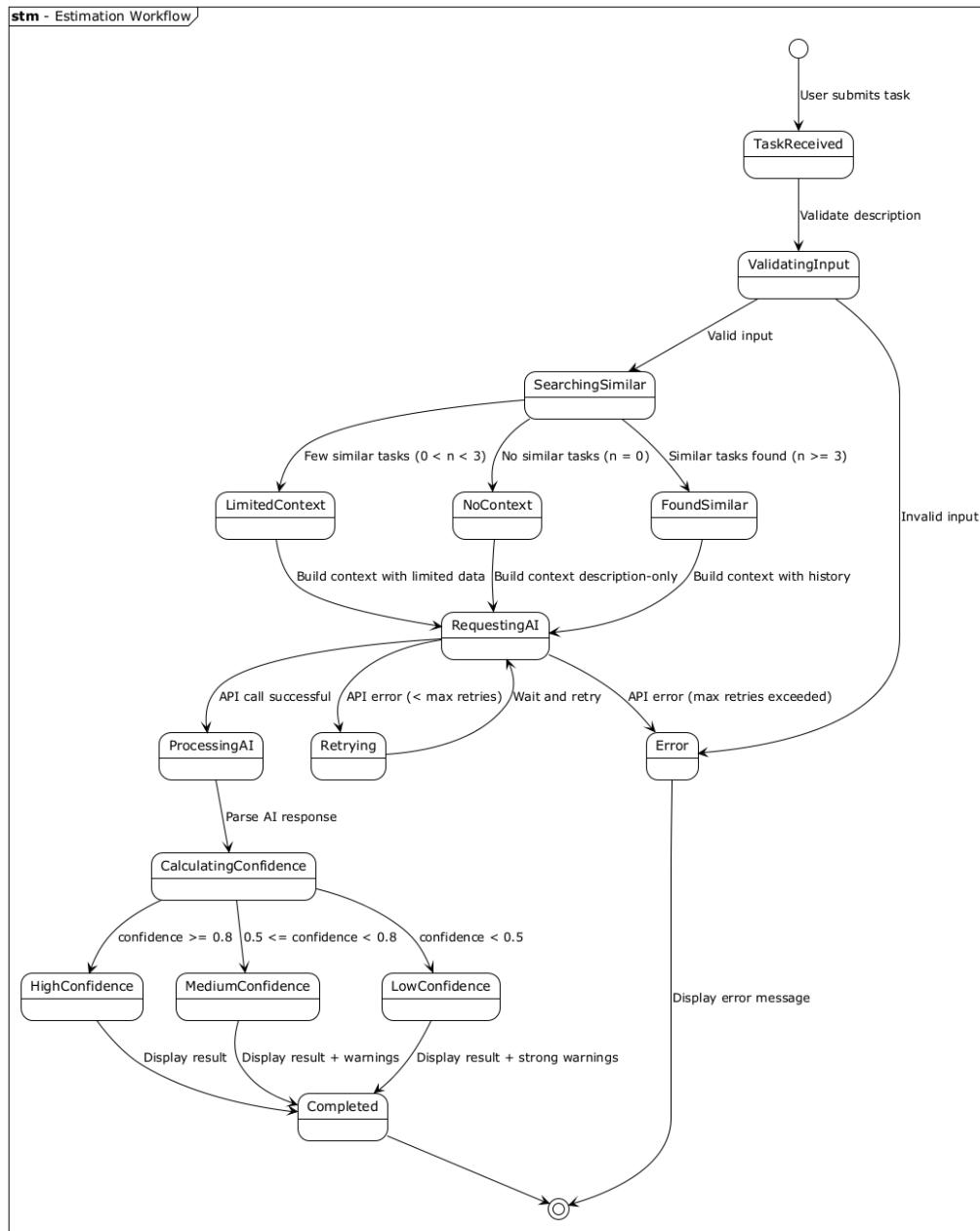
**Configuração:** Sistema hierárquico de configuração (defaults < arquivo config < variáveis ambiente < parâmetros CLI).

**Validação:** Validação rigorosa de entrada para prevenir injection attacks e garantir integridade dos dados.

## 3.5 Diagramas de Estados

Os diagramas de máquina de estados descrevem os diferentes estados pelos quais as entidades principais do sistema passam durante sua execução.

### 3.5.1 Diagrama de Estados - Estimativa

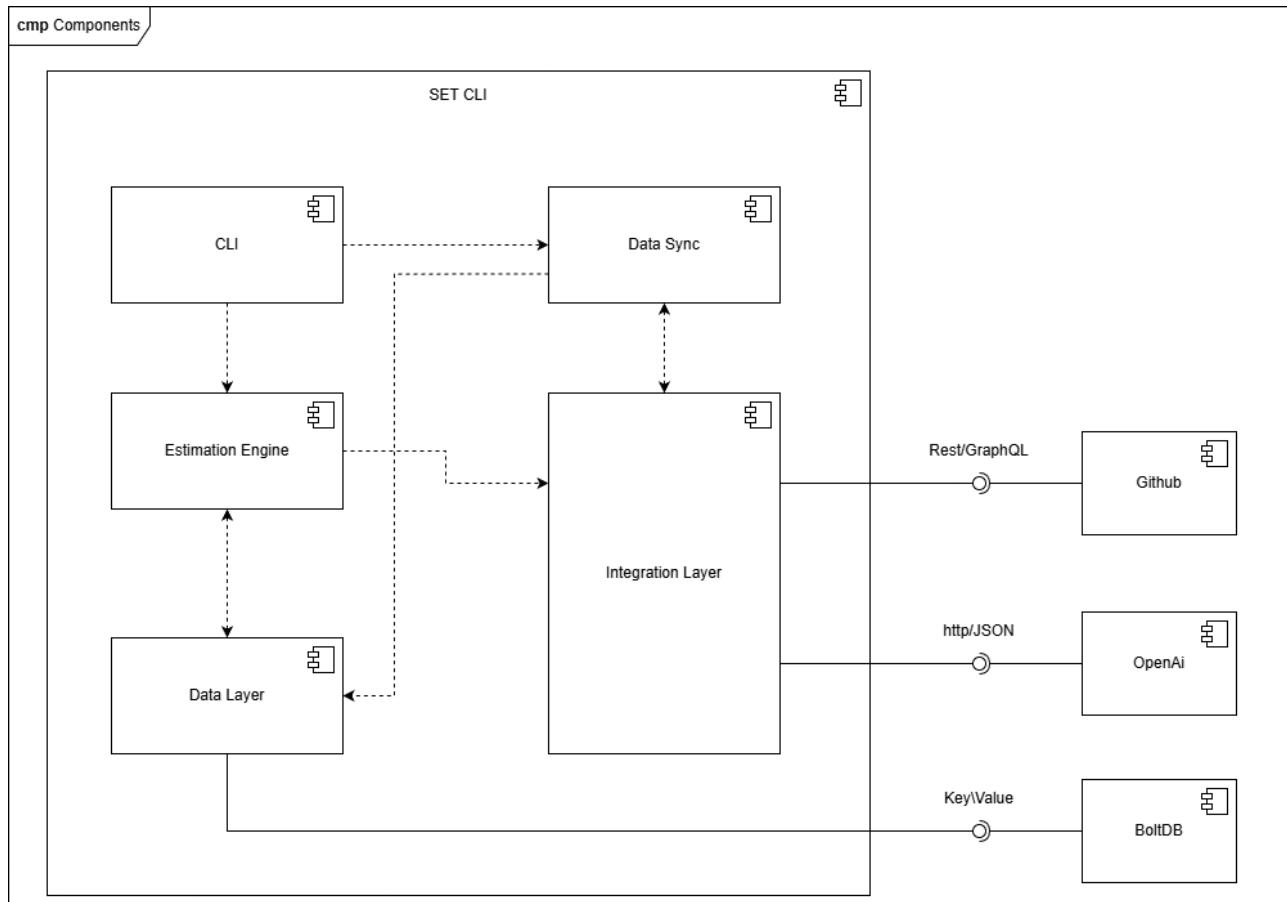


**Figura 3.5 - Diagrama de Estados: Workflow de Estimativa**

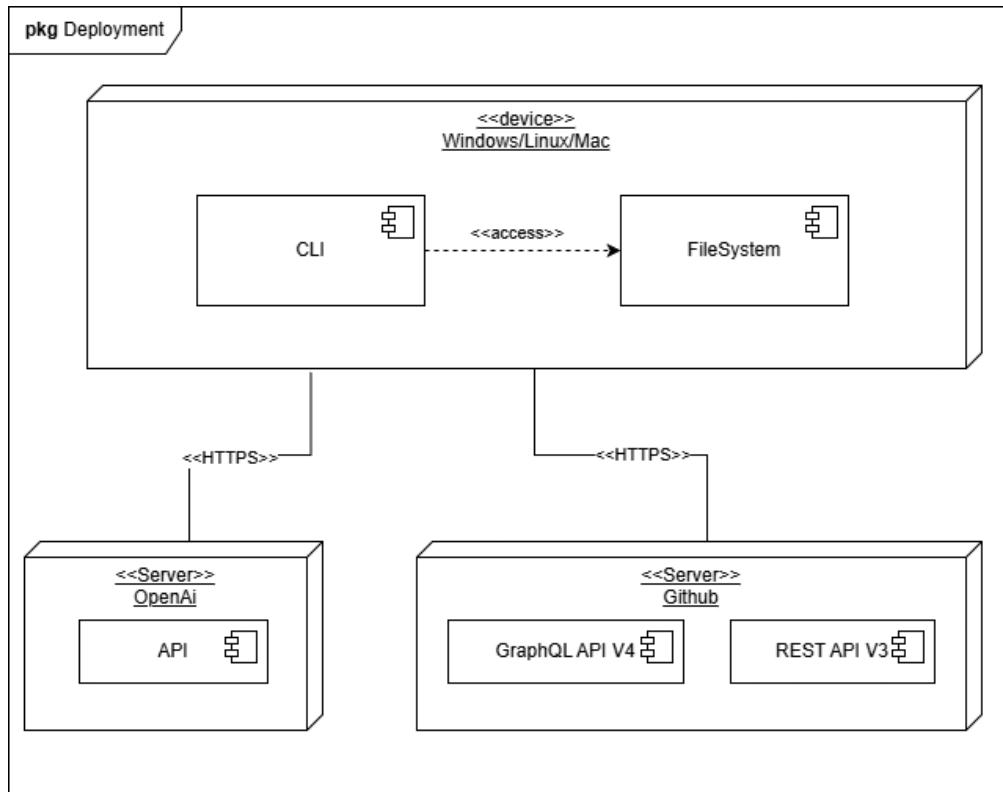
Este diagrama mostra os estados pelos quais uma solicitação de estimativa passa, desde a submissão até a conclusão ou erro.

### 3.6 Diagrama de Componentes e Implantação.

O diagrama de componentes(Figura 3.6.1) mostra a organização modular do sistema em componentes de alto nível, suas interfaces fornecidas e requeridas, e as dependências entre eles.



**Figura 3.6.1 - Diagrama de Componentes**



**Figura 3.6.2 - Diagrama de Implantação da aplicação**

O diagrama de implantação (Figura 3.6.2) mostra a arquitetura física do sistema, incluindo os nós de hardware, artefatos de software e seus protocolos de comunicação.

#### Nós de Implantação:

- **Device Windows/Linux/Mac:** Máquina local onde o binário *set* é executado
- **GitHub Servers:** Servidores que hospedam os repositórios e APIs
- **OpenAI Servers:** Servidores que processam as requisições de IA

#### Protocolos de Comunicação:

- HTTPS com autenticação por token (GitHub)
- HTTPS com autenticação por API key (OpenAI)
- Acesso local ao filesystem para BoltDB

## 4. Projeto de Interface com Usuário

Como o SET é uma ferramenta CLI (Command Line Interface), esta seção apresenta exemplos de interface de terminal para os principais comandos do sistema.

### 4.1 Ajuda

SET CLI  
Software Estimation Tool  
AI-Powered Effort Estimation for Agile Teams

A powerful CLI tool that combines AI intelligence with historical data to provide accurate software development effort estimates.

#### Core Features

- AI-Powered Estimations  
Uses OpenAI GPT models to analyze task complexity, dependencies, and risks. Provides estimates in hours, story points, and t-shirt sizes (XS, S, M, L, XL).
- Historical Data Analysis  
Learns from your GitHub repository history using advanced similarity algorithms (TF-IDF + cosine similarity). Matches new tasks with similar past work for data-driven estimates.
- Batch Processing  
Estimate entire sprint backlogs in seconds with parallel processing. Supports JSON and CSV input formats. Perfect for sprint planning.
- Multiple Export Formats  
Export results as JSON, CSV, Markdown, JIRA, or GitHub format for seamless integration with your existing tools.
- Team Performance Analytics  
Track estimation accuracy, analyze trends by category, and optimize team velocity with detailed statistics.
- GitHub Projects Integration  
Sync custom fields like story points, estimated hours, and priority from GitHub Projects V2 for enhanced accuracy.

#### Perfect For

• Developers	Accurate task estimates for better planning
• Scrum Masters	Sprint capacity planning and backlog refinement
• Product Owners	Roadmap planning and release forecasting
• Engineering Leads	Resource allocation and timeline estimation

#### Quick Start

1. Initial Setup (interactive configuration)  
\$ set configure --initial
2. Load Historical Data (from GitHub or sample dataset)  
\$ set sync --custom-fields  
\$ set dev seed --count 100 --with-custom-fields

```

| Quick Start
  1. Initial setup (interactive configuration)
    $ set configure --initial

  2. Load Historical Data (from GitHub or sample dataset)
    $ set sync --custom-fields
    $ set dev seed --count 100 --with-custom-fields

  3. Estimate a Single Task
    $ set estimate "Add user authentication" --labels backend,security

  4. Batch Process Sprint Backlog
    $ set batch --file sprint-backlog.json

  5. Export Results
    $ set export --format csv --output estimates.csv

| Common Workflows
  Sprint Planning:
    $ set sync --custom-fields
    $ set batch --file sprint-15.json --output sprint-15-report.md

  Single Task Estimation:
    $ set estimate "Fix memory leak" --description "Cache not releasing" --show-similar

  Data Analysis:
    $ set inspect
    $ set inspect --list --custom
    $ set export --format json --output data.json

| Configuration
  Config file: ~/.set.yaml (auto-created on first run)

  Required Settings:
    - AI Provider (OpenAI)
    - API Key (OpenAI API key)
    - Model (gpt-4, gpt-4-turbo, gpt-3.5-turbo)

  Optional Settings:
    - GitHub Token (for syncing repositories)
    - Default Repository (owner/repo)
    - Similarity Threshold (0.0-1.0, default: 0.3)
    - Max Similar Tasks (default: 10)

| Need Help?
  set [command] --help      Detailed help for any command
  set version               Show version information
  set configure list        View current configuration

For documentation, examples, and troubleshooting:
  https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2025-2-tcci-0393100-dev-inacio-moraes/tree/main/Codigo



---


Usage:
  set [command]

Available Commands:
  batch      Process batch estimation requests
  completion Generate the autocompletion script for the specified shell
  configure  Configure SET CLI settings
  dev        Developer utilities and testing tools
  estimate   Estimate task effort using AI and historical data
  export     Export historical data to various formats
  help       Help about any command
  inspect   Inspect locally stored GitHub data
  sync       Synchronize GitHub repository data
  version   Print the version number of SET CLI

Flags:
  --config string    config file (default is ${HOME}/.set.yaml)
  -h, --help          help for set
  --log-level string log level (debug, info, warn, error) (default "info")
  --verbose          enable verbose logging

Use "set [command] --help" for more information about a command.

```

**Figura 4.1** - Interface ao executar o comando de ajuda: *set help*

## 4.2 Configuração Inicial

```
Welcome to SET CLI - Initial Setup Wizard

GitHub Configuration
Enter your GitHub personal access token: [REDACTED]
Enter default repository (e.g., facebook/react) [optional]: 0-Inacio-0/SET_Projet_TEST

AI Provider Configuration
Choose AI provider (openai/claude) [openai]:
Enter your OPENAI API key: [REDACTED]

Preferences
Output format (table/json/csv) [table]:
Enable colored output? (y/n) [y]: y

Saving configuration...
✓ Configuration saved to: C:\Users\Inácio Moraes\.set.yaml

Setup complete! You can now start using SET CLI.
Try: set estimate --task "Your task description"
```

**Figura 4.2** - Interface ao executar o comando de configuração inicial: `set configure --initial`

## 4.3 Validando a Configuração

```
Validating SET CLI Configuration

Validating GitHub token...
Validation Results:
✓ Token is valid
Authenticated as: 0-Inacio-0

Token scopes:
- project
- repo

Required Permissions:
✓ Repository access
✓ Issues access
✓ Pull requests access
✓ GitHub Projects access (custom fields)

Repository: 0-Inacio-0/SET_Projet_TEST
✓ Repository accessible
[Private repository]

Testing specific permissions...
✓ Can read issues
✓ Can read pull requests

API Rate Limit:
Limit: 5000 requests/hour
Remaining: 4994 requests

Validating AI Configuration...
Provider: openai
✓ AI API key is valid
✓ Model available: gpt-4

You can now use AI-powered estimation:
set estimate "Add user authentication"

✓ Configuration is valid and ready to use!

You can now run:
set sync      # Sync repository data
set estimate  # Estimate tasks
```

**Figura 4.2** - Interface ao executar o comando de validar configuração: `set configure --validate`

## 4.4 Estimativa de Tarefa Única

The screenshot shows the GitHub Project interface for estimating a task. The task title is "Implement OAuth login with Google". The estimation details are as follows:

- Time: 16.0h
- Size: L
- Points: 8
- Conf: 70% [progress bar]

The analysis section states: "The task of implementing OAuth login with Google is a development task that involves integration with an external service (Google). This typically involves understanding the external service's API, coding the integration, handling different scenarios (success, failure, cancellation), and testing. The task is more complex than a simple feature but less complex than creating new components from scratch. The average duration for development tasks is 16 hours, which aligns with this estimate."

The assumptions section lists:

- The development environment is set up and ready for development
- The developer is familiar with OAuth and Google APIs
- There are no significant architectural changes required to support this feature

The risks section lists:

- ⚠ Potential delays if unfamiliar with OAuth or Google APIs
- ⚠ Possible complications or limitations with Google's API
- ⚠ Potential security implications of handling user authentication

The recommendation section advises: "Before starting the task, ensure the developer is familiar with OAuth and Google APIs. Consider a spike to investigate Google's API and any potential challenges. Also, ensure to follow best practices for handling user authentication to mitigate security risks."

At the bottom, it shows "ai · openai" and "Next Steps" with items: "Review the assumptions and risks above" and "Consider adding custom fields to GitHub Projects".

**Figura 4.2** - Interface ao executar o comando de estimar: *set estimate "Implement OAuth login with Google"*

## 4.5 Processamento em Batch

```
Batch Estimation
=====
Input File: .\examples\sprint-backlog.json
Tasks: 5
Workers: 1
Model: gpt-4
=====

Processing tasks...

=====
[✓] Batch Processing Complete
Duration: 56s
Success: 5/5 tasks
=====
```

Batch Estimation Report				
Overall Statistics				
Total Hours	96.0			
Average	19.2			
Median	19.2			
Range	8.0 - 40.0			
Avg Confidence	73 %			
Size Distribution				
Size	Count	Total Hours	Avg Hours	
M	1	8.0	8.0	
L	4	88.0	22.0	
Confidence Distribution				
High (>70%)	5 tasks			
Medium (50-69%)	0 tasks			
Low (<50%)	0 tasks			
Category Distribution				
Category	Count	Total Hours		
backend	3	72.0		
security	1	40.0		
frontend	3	40.0		
critical	1	8.0		
test	1	16.0		
feature	2	56.0		
bug	1	8.0		
refactor	1	16.0		
performance	1	16.0		
Task Results				
ID	Title	Hours	Size	Points
	Implementar autenticação ...	40.0	L	13
	Corrigir bug no formulário...	8.0	M	5
	Adicionar paginação na li...	16.0	L	13
	Escrever testes unitários ...	16.0	L	8
	Refatorar componente de car...	16.0	L	13

**Figura 4.2 - Interface ao executar o comando de estimar em batch:** `set batch --file examples\sprint-backlog.json`

## 4.6 Inspeção de Dados Históricos

Showing first 20 of 471 issues						
#	State	Title	Author	Labels	Custom Fields	
#1001	closed	Daily Management Tasks - April 2013	user0	enhancement +1	3 fields	
#1004	closed	Upgrade cube PC CD-Writer	user7	support +2	3 fields	
#1012	closed	Creating new SiP testing Components	user8	feature +2	3 fields	
#1017	closed	Investigate website errors	user7	support +2	3 fields	
#1020	closed	Merging database to latest Technical ...	user9	feature +1	3 fields	
#1021	closed	Risk History Not Showing USM Claim Fi...	user2	bug +2	3 fields	
#1027	closed	CCC Documentation of Any SMS Changes...	user8	enhancement +2	3 fields	
#1030	closed	Knowledge Base Need An Attached Temp...	user1	feature +1	3 fields	
#1032	closed	BNI meeting and setting up for Visit...	user3	enhancement	3 fields	
#1034	closed	Weekly Technical Developer Meeting 25...	user3	enhancement +2	3 fields	
#1035	closed	Moving Diary Event Changes to DEV, Co...	user6	feature +1	3 fields	
#1038	closed	YouLink HR: Non-Standart Working Patt...	user8	bug +2	3 fields	
#1039	closed	Daily server checks 2007/38	user9	support +2	3 fields	
#1040	closed	YYY ZZZ Meeting	user0	enhancement +1	3 fields	
#1041	closed	Test and Deploy Latest Security Patches	user7	support +2	3 fields	
#1042	closed	Weekly Developer Meeting 08th Novembe...	user1	enhancement +2	3 fields	
#1043	closed	Cross Market Services IT Progress Mee...	user3	enhancement	3 fields	
#1044	closed	Daily Server Health Checks - 2009/14	user3	support +2	3 fields	
#1048	closed	Build Affiliate Business Object	user7	feature +1	3 fields	
#1051	closed	Weekly Developer Meeting 27th Feb 200...	user0	enhancement +2	3 fields	

Next Steps  
• Use 'set inspect --issue <number>' to view details

**Figura 4.2 - Interface ao executar o comando de listar dados:** `set inspect --list --custom`

#### 4.7 Sincronização GitHub

```
! Fetching pull requests [0s]
-----
Sync Complete ✓
-----
Issues:      1201
Pull Requests: 0
-----
```

**Figura 4.2** - Interface ao executar o comando de listar dados: *set inspect --list --custom*

## 5. Glossário e Modelos de Dados

### 5.1 Glossário de Termos

Termo	Definição
<b>CLI</b>	Command Line Interface - Interface de linha de comando
<b>IA</b>	Inteligência Artificial
<b>API</b>	Application Programming Interface - Interface de Programação de Aplicações
<b>Scrum</b>	Framework ágil para desenvolvimento de software
<b>GitHub</b>	Plataforma de hospedagem de código e versionamento
<b>Sprint</b>	Período fixo de desenvolvimento no Scrum (geralmente 1-4 semanas)

<b>Story Points</b>	Unidade de medida para estimar o esforço relativo de tarefas
<b>Velocity</b>	Quantidade de story points completados por sprint
<b>Planning Poker</b>	Técnica de estimativa colaborativa usada em metodologias ágeis
<b>Accuracy</b>	Precisão das estimativas (quão próximas estão do tempo real)
<b>Confidence Score</b>	Nível de confiança da estimativa (0.0 a 1.0)
<b>TF-IDF</b>	Term Frequency-Inverse Document Frequency - algoritmo de processamento de texto
<b>Cosine Similarity</b>	Medida de similaridade entre vetores de texto
<b>BoltDB</b>	Banco de dados embarcado key-value em Go
<b>GraphQL</b>	Linguagem de consulta para APIs
<b>Token</b>	Chave de autenticação para APIs
<b>Rate Limit</b>	Limite de requisições por período de tempo

## 5.2 Modelos de Dados

### 5.2.1 Entidade: Task

Atributo	Tipo	Descrição
<b>id</b>	string	Identificador único da tarefa
<b>title</b>	string	Título descritivo da tarefa
<b>description</b>	string	Descrição detalhada da tarefa
<b>labels</b>	[]string	Lista de labels para categorização
<b>estimated_hours</b>	float64	Horas estimadas pela IA
<b>actual_hours</b>	float64	Horas realmente gastas (histórico)
<b>story_points</b>	int	Story points atribuídos
<b>size</b>	string	Tamanho da tarefa (XS, S, M, L, XL)
<b>created_at</b>	timestamp	Data de criação
<b>completed_at</b>	timestamp	Data de conclusão
<b>developer</b>	string	Desenvolvedor responsável
<b>repository</b>	string	Repositório de origem

### 5.2.2 Entidade: Estimation

Atributo	Tipo	Descrição
<b>id</b>	string	Identificador único da estimativa

<b>task_id</b>	string	Referência para a tarefa
<b>estimated_hours</b>	float64	Horas estimadas
<b>story_points</b>	int	Story points calculados
<b>size</b>	string	Classificação de tamanho
<b>confidence</b>	float64	Nível de confiança (0.0-1.0)
<b>reasoning</b>	string	Explicação da IA sobre a estimativa
<b>similar_tasks</b>	[]SimilarTask	Lista de tarefas similares encontradas
<b>warnings</b>	[]string	Avisos sobre limitações
<b>created_at</b>	timestamp	Data da estimativa
<b>model</b>	string	Modelo de IA usado (ex: gpt-4)

### 5.2.3 Entidade: SimilarTask

Atributo	Tipo	Descrição
<b>task_id</b>	string	ID da tarefa similar
<b>title</b>	string	Título da tarefa similar
<b>similarity_score</b>	float64	Score de similaridade (0.0-1.0)
<b>estimated_hours</b>	float64	Horas que foram estimadas
<b>actual_hours</b>	float64	Horas realmente gastas
<b>accuracy_rate</b>	float64	Taxa de acerto da estimativa original

### 5.2.4 Entidade: Issue (GitHub)

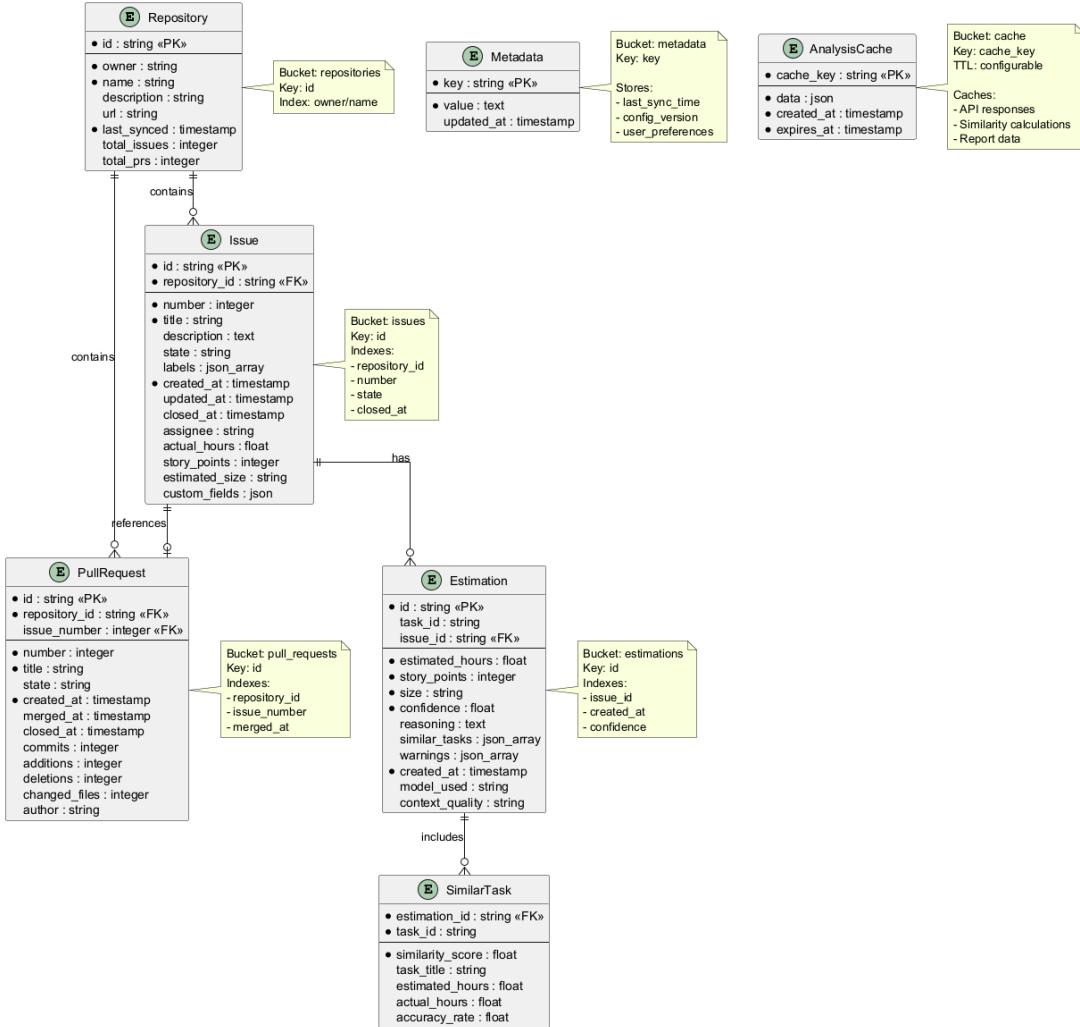
Atributo	Tipo	Descrição
<b>id</b>	string	ID único do GitHub
<b>number</b>	int	Número da issue no repositório
<b>repository_id</b>	string	Referência ao repositório
<b>title</b>	string	Título da issue
<b>description</b>	text	Corpo da issue
<b>state</b>	string	Estado (open, closed)
<b>labels</b>	json_array	Labels da issue
<b>created_at</b>	timestamp	Data de criação
<b>closed_at</b>	timestamp	Data de fechamento
<b>actual_hours</b>	float64	Horas gastas (de custom field)
<b>story_points</b>	int	Story points (de custom field)
<b>estimated_size</b>	string	Tamanho estimado (de custom field)
<b>custom_fields</b>	json	Outros campos customizados

### 5.2.5 Entidade: PullRequest

Atributo	Tipo	Descrição
<b>id</b>	string	ID único do GitHub
<b>number</b>	int	Número do PR no repositório

<b>repository_id</b>	string	Referência ao repositório
<b>issue_number</b>	int	Issue relacionada (se houver)
<b>title</b>	string	Título do PR
<b>state</b>	string	Estado (open, closed, merged)
<b>created_at</b>	timestamp	Data de criação
<b>merged_at</b>	timestamp	Data de merge
<b>commits</b>	int	Número de commits
<b>additions</b>	int	Linhas adicionadas
<b>deletions</b>	int	Linhas removidas
<b>changed_files</b>	int	Arquivos alterados
<b>author</b>	string	Autor do PR

### 5.3 Diagrama Entidade-Relacionamento



O diagrama acima mostra a estrutura do banco de dados BoltDB, organizada em buckets (equivalentes a tabelas) e os relacionamentos entre as entidades.

## 6. Casos de Teste

Esta seção apresenta os casos de teste elaborados para validar o sistema SET CLI. Na Seção 6.1, são detalhados os testes de aceitação, projetados para assegurar que as funcionalidades desenvolvidas atendam às necessidades dos usuários conforme especificado no documento de visão. A Seção 6.2 descreve os testes de integração, focando em verificar a interação entre o sistema e os componentes externos, como GitHub API e serviços de IA.

Os testes foram organizados para cobrir as quatro necessidades principais identificadas no documento de visão: precisão em estimativas de esforço, automatização do processo de estimativa, utilização de contexto histórico e integração com ferramentas existentes. Cada caso de teste inclui

pré-condições, dados de entrada, ações a serem executadas e resultados esperados, garantindo cobertura completa das funcionalidades críticas.

## 6.1 Testes de Aceitação

Os testes de aceitação (TA) foram elaborados com base nas necessidades identificadas no documento de visão e nos casos de uso apresentados. As necessidades testadas são:

- **N1:** Precisão em Estimativas de Esforço
- **N2:** Automatização do Processo de Estimativa
- **N3:** Utilização de Contexto Histórico
- **N4:** Integração com Ferramentas Existentes

Para a escrita dos testes, utiliza-se uma tabela com seis campos: Identificador (padrão TAXX), Necessidade, Caso de teste, Pré-condições, Dados de entrada, Ações e Resultado esperado.

### 6.1.1 Configuração do Sistema (UC01, UC02, UC08)

Esta seção valida o processo de configuração inicial do sistema SET CLI, incluindo a configuração interativa de credenciais (GitHub token e OpenAI API key), validação de conectividade com APIs externas, e persistência segura de dados sensíveis. Os testes garantem que o sistema pode ser configurado corretamente tanto em modo interativo quanto através de flags CLI, e que todas as validações necessárias são realizadas antes da persistência.

#### TA01 - Configuração Inicial Interativa

Este teste valida o fluxo de configuração inicial do sistema através do modo interativo, onde o usuário é guiado por prompts sequenciais para fornecer todas as credenciais necessárias. Garante que tokens sejam armazenados com permissões seguras.

Campo	Descrição
<b>Identificador</b>	TA01
<b>Casos de Uso</b>	UC01, UC02
<b>Necessidade</b>	Integração com Ferramentas Existentes (N4)
<b>Caso de teste</b>	Configurar sistema pela primeira vez no modo interativo

<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Sistema não configurado previamente</li> <li>• GitHub token válido disponível</li> <li>• OpenAI API key válida disponível</li> </ul>
<b>Dados de entrada</b>	<ul style="list-style-type: none"> <li>• GitHub Token: ghp_xxxxxxxxxxxx</li> <li>• Repositório: user/project</li> <li>• AI Provider: openai</li> <li>• AI API Key: sk-xxxxxxxxxxxx</li> <li>• AI Model: gpt-4</li> </ul>
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Executar set configure --initial</li> <li>2. Seguir prompts interativos fornecendo dados</li> <li>3. Confirmar salvamento da configuração</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• Arquivo <code>~/.set/config.yaml</code> criado</li> <li>• Permissões de arquivo: 600 (somente leitura/escrita pelo dono)</li> <li>• Mensagem de sucesso exibida</li> <li>• Sistema pronto para uso</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• Validação de formato de tokens realizada</li> <li>• Tokens nunca expostos em logs</li> <li>• Configuração recuperável em próximas execuções</li> <li>• Rollback em caso de erro durante configuração</li> </ul>

## TA02 - Validação de Configuração

Este teste verifica a funcionalidade de validação de configuração, assegurando que o sistema pode verificar a validade de tokens GitHub e OpenAI API keys através de chamadas de teste às APIs externas. Valida que o sistema fornece feedback claro sobre o status das credenciais e conectividade.

Campo	Descrição
<b>Identificador</b>	TA02
<b>Casos de Uso</b>	UC08
<b>Necessidade</b>	Integração com Ferramentas Existentes (N4)
<b>Caso de teste</b>	Validar tokens e conectividade com APIs externas
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Sistema configurado</li> <li>• Conexão com internet disponível</li> </ul>

<b>Dados de entrada</b>	Comando: set configure --validate
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Executar validação de configuração</li> <li>2. Sistema valida GitHub token</li> <li>3. Sistema valida OpenAI API key</li> <li>4. Sistema verifica conectividade</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• Validação de GitHub token: ✓ Válido, scopes corretos</li> <li>• Validação de OpenAI: ✓ API key válida, créditos disponíveis</li> <li>• Informações de rate limit exibidas</li> <li>• Recomendações de segurança se houver</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• Timeout de validação: ≤ 10 segundos</li> <li>• Mensagens de erro claras em caso de falha</li> <li>• Validação não expõe tokens completos</li> <li>• Sistema funciona offline após validação inicial</li> </ul>

### 6.1.2 Estimativa Individual (UC03)

Esta seção testa a funcionalidade central do sistema: a geração de estimativas individuais para tarefas. Os testes cobrem diferentes cenários de similaridade de dados históricos, desde casos com alta confiança (muitas tarefas similares disponíveis) até casos com baixa confiança (tarefas inéditas sem histórico similar). Também valida a capacidade do sistema de utilizar contexto adicional fornecido pelo usuário para melhorar a precisão das estimativas.

#### TA03 - Estimativa com Alta Similaridade

Este teste valida o cenário ideal onde o sistema possui dados históricos abundantes sobre tarefas similares à tarefa sendo estimada. Verifica que o algoritmo de busca adaptativa encontra matches de alta qualidade e que a IA gera estimativas precisas baseadas nesse contexto rico.

Campo	Descrição
<b>Identificador</b>	TA03
<b>Casos de Uso</b>	UC03

<b>Necessidade</b>	Precisão em Estimativas de Esforço (N1), Utilização de Contexto Histórico (N3)
<b>Caso de teste</b>	Gerar estimativa com alta confiança para tarefa similar
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Sistema configurado e sincronizado</li> <li>• Base histórica contém <math>\geq 5</math> tarefas de autenticação</li> <li>• Custom fields disponíveis (hours, story points)</li> </ul>
<b>Dados de entrada</b>	Comando: set estimate "Implementar autenticação OAuth com Google" --labels authentication,backend --show-similar
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Sistema analisa descrição e labels</li> <li>2. Busca adaptativa por tarefas similares (thresholds: 50%, 40%, 30%)</li> <li>3. SimilarityEngine calcula Jaccard similarity</li> <li>4. Dataset aprimorado selecionado (similar + stratified + percentile)</li> <li>5. AI gera estimativa com contexto</li> <li>6. Sistema calcula confidence score</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• <b>Estimativa:</b> 6-10 horas</li> <li>• <b>Tamanho:</b> M (Medium)</li> <li>• <b>Story Points:</b> 5</li> <li>• <b>Confiança:</b> <math>\geq 75\%</math></li> <li>• <b>Tarefas similares:</b> <math>\geq 3</math> listadas com scores</li> <li>• <b>Tempo de resposta:</b> <math>\leq 15</math> segundos</li> <li>• <b>Reasoning:</b> Explicação detalhada da IA</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• Similaridade <math>&gt; 40\%</math> para top 3 matches</li> <li>• Dataset inclui min 10 tarefas (se disponível)</li> <li>• Confiança baseada em: número de matches, qualidade de dados, variância</li> <li>• Histórico de tarefas similares exibido com horas reais</li> </ul>

## TA04 - Estimativa com Baixa Similaridade

Este teste verifica o comportamento do sistema quando não há dados históricos similares disponíveis. Garante que o sistema ainda consegue gerar estimativas razoáveis baseadas em padrões gerais, mas sinaliza claramente a baixa confiança e recomenda revisão manual.

Campo	Descrição
<b>Identificador</b>	TA04
<b>Casos de Uso</b>	UC03
<b>Necessidade</b>	Precisão em Estimativas de Esforço (N1)
<b>Caso de teste</b>	Gerar estimativa com confiança reduzida para tarefa inédita
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Sistema configurado</li> <li>• Base histórica não contém tarefas similares a blockchain</li> </ul>
<b>Dados de entrada</b>	Comando: set estimate "Implementar blockchain para rastreamento de transações" --labels blockchain,experimental
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Sistema busca tarefas similares em múltiplos thresholds</li> <li>2. Nenhum match &gt; 30% encontrado</li> <li>3. AI faz estimativa baseada apenas em descrição e contexto geral</li> <li>4. Sistema calcula confiança baixa</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• <b>Estimativa:</b> Range amplo (ex: 20-40 horas)</li> <li>• <b>Tamanho:</b> L ou XL</li> <li>• <b>Confiança:</b> ≤ 40%</li> <li>• <b>Aviso:</b> "⚠ Baixa confiança - Dados históricos limitados"</li> <li>• <b>Recomendação:</b> "Revisar estimativa com equipe técnica"</li> <li>• <b>Threshold usado:</b> 15% (mais permissivo)</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• Sistema não falha mesmo sem dados similares</li> <li>• Aviso claro sobre limitações</li> <li>• Estimativa ainda é gerada (baseada em padrões gerais)</li> </ul>

- Sugestão de quebrar em tarefas menores se XL

## TA05 - Estimativa com Contexto Adicional

Este teste valida a capacidade do sistema de utilizar descrições detalhadas e contexto adicional (labels, priority, descrição expandida) para gerar estimativas mais precisas. Verifica que a IA analisa todos os detalhes fornecidos e identifica riscos específicos baseados no contexto.

Campo	Descrição
<b>Identificador</b>	TA05
<b>Casos de Uso</b>	UC03
<b>Necessidade</b>	Precisão em Estimativas de Esforço (N1)
<b>Caso de teste</b>	Estimativa com descrição detalhada e contexto
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Sistema configurado e sincronizado</li> </ul>
<b>Dados de entrada</b>	Comando: set estimate "Migração de banco PostgreSQL" --description "Migrar 50GB de dados, 30 tabelas, com zero downtime" --labels database,migration --priority high
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Sistema processa título + descrição detalhada</li> <li>2. Labels adicionam peso à busca de similaridade</li> <li>3. Priority pode influenciar complexidade assumida</li> <li>4. AI analisa contexto completo</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• Estimativa mais precisa devido ao contexto</li> <li>• Reasoning menciona "50GB", "30 tabelas", "zero downtime"</li> <li>• Riscos identificados pela IA listados</li> <li>• Recomendações técnicas incluídas</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• Descrição detalhada aumenta precisão em ~20%</li> <li>• Riscos relevantes identificados (ex: downtime, data loss)</li> <li>• Próximos passos sugeridos pela IA</li> </ul>

### 6.1.3 Estimativa em Lote (UC 04)

Este teste valida a capacidade do sistema de utilizar descrições detalhadas e contexto adicional (labels, priority, descrição expandida) para gerar estimativas mais precisas. Verifica que a IA analisa todos os detalhes fornecidos e identifica riscos específicos baseados no contexto.

#### TA06 - Processamento em Lote - Planning Sprint

Este teste valida o processamento paralelo de múltiplas tarefas usando worker pools (goroutines), garantindo alta performance e throughput. Verifica que todas as tarefas são processadas com sucesso, estatísticas são calculadas corretamente, e o sistema fornece feedback visual.

Campo	Descrição
<b>Identificador</b>	TA06
<b>Casos de Uso</b>	UC04
<b>Necessidade</b>	Automatização do Processo de Estimativa (N2)
<b>Caso de teste</b>	Processar lote de tarefas para sprint planning
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Sistema configurado</li> <li>• Arquivo sprint-backlog.json com 15 tarefas</li> <li>• Base histórica disponível</li> </ul>
<b>Dados de entrada</b>	Arquivo JSON com 15 tarefas: <pre>json&lt;br&gt;{&lt;br&gt; "metadata": {"sprint": 15, "team": "backend"},&lt;br&gt; "tasks": [&lt;br&gt; {"id": "T-001", "title": "...", "labels": [...]},&lt;br&gt; ...&lt;br&gt; ]&lt;br&gt;}&lt;br&gt;</pre>
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Executar <code>set batch --file sprint-backlog.json --workers 5 --format text --progress</code></li> <li>2. Sistema cria worker pool com 5 goroutines</li> <li>3. Tarefas distribuídas entre workers</li> <li>4. Processamento paralelo com agregação thread-safe</li> <li>5. Estatísticas calculadas</li> </ol>

<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• <b>Taxa de sucesso:</b> 100% (15/15 tarefas)</li> <li>• <b>Tempo total:</b> ≤ 90 segundos</li> <li>• <b>Total de horas:</b> Soma agregada exibida</li> <li>• <b>Distribuição por tamanho:</b> XS/S/M/L/XL contabilizada</li> <li>• <b>Confiança média:</b> Exibida</li> <li>• <b>Progress bar:</b> Atualizado em tempo real</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• Processamento paralelo real (goroutines)</li> <li>• Progress indicator funcional</li> <li>• Estatísticas precisas (média, mediana, range)</li> <li>• Relatório consolidado gerado</li> <li>• Nenhuma task perdida ou duplicada</li> </ul>

## TA07 - Exportação em Múltiplos Formatos

Este teste verifica a capacidade do sistema de exportar resultados de estimativas em lote em diferentes formatos (JSON, CSV, Markdown), garantindo que os dados sejam consistentes entre formatos e compatíveis com ferramentas externas como Excel, Google Sheets, e sistemas de documentação.

Campo	Descrição
<b>Identificador</b>	TA07
<b>Casos de Uso</b>	UC04, UC06
<b>Necessidade</b>	Automatização (N2), Integração (N4)
<b>Caso de teste</b>	Exportar resultados de batch em diferentes formatos
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Estimativas geradas via batch</li> </ul>

<b>Dados de entrada</b>	Comandos:
	<ul style="list-style-type: none"> <li>• set batch --file tasks.json --format json --output results.json</li> <li>• set batch --file tasks.json --format csv --output results.csv</li> <li>• set batch --file tasks.json --format markdown --output report.md</li> </ul>
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Executar batch com formato JSON</li> <li>2. Executar batch com formato CSV</li> <li>3. Executar batch com formato Markdown</li> <li>4. Validar cada arquivo gerado</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• <b>JSON</b>: Estrutura completa, válida, parseável</li> <li>• <b>CSV</b>: Headers corretos, importável em Excel</li> <li>• <b>Markdown</b>: Tabelas formatadas, legível</li> <li>• Todos os formatos contêm mesmos dados</li> <li>• Arquivos salvos nos caminhos especificados</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• JSON válido (validação com schema)</li> <li>• CSV com encoding UTF-8</li> <li>• Markdown renderiza corretamente no GitHub</li> <li>• Dados consistentes entre formatos</li> </ul>

#### 6.1.4 Sincronização e Histórico (UC05, UC07)

Esta seção testa a sincronização de dados do GitHub (issues e pull requests) e a análise de dados históricos. Os testes validam tanto a sincronização incremental (apenas deltas desde última execução) quanto a sincronização completa com extração de custom fields do GitHub Projects V2. Também verifica a capacidade de consultar e filtrar dados históricos armazenados localmente.

#### TA08 - Sincronização Incremental

Este teste valida o modo de sincronização incremental, que busca apenas dados novos ou atualizados desde a última execução. Verifica que o sistema respeita o throttling para evitar sincronizações desnecessárias, utiliza paginação eficiente, e preserva dados antigos enquanto adiciona novos.

Campo	Descrição
<b>Identificador</b>	TA08
<b>Casos de Uso</b>	UC07
<b>Necessidade</b>	Utilização de Contexto Histórico (N3), Integração (N4)
<b>Caso de teste</b>	Sincronizar dados do GitHub incrementalmente
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Sistema configurado com GitHub</li> <li>• Sincronização prévia realizada há 1 semana</li> <li>• Novas issues e PRs criadas desde última sync</li> </ul>
<b>Dados de entrada</b>	Comando: set sync (sem flags, modo incremental)
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Sistema verifica timestamp da última sincronização</li> <li>2. Modo incremental ativado (since = lastSyncTime)</li> <li>3. GraphQL query com filtro temporal</li> <li>4. Paginação automática de resultados</li> <li>5. Batch save de issues/PRs (100 por lote)</li> <li>6. Atualização de lastSyncTime</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• Apenas <b>deltas</b> sincronizados (ex: 15 issues novas, 8 PRs)</li> <li>• <b>Tempo de sync</b>: Proporcional ao delta (ex: &lt;30s para 25 items)</li> <li>• <b>Rate limit</b>: Respeitado (exibir restante)</li> <li>• <b>Progress bar</b>: Feedback visual</li> <li>• <b>Resumo</b>: "Synced 15 issues, 8 PRs in 22s"</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• Sincronização incremental funcional</li> <li>• Throttling: não sync se última &lt; 5min (exceto --force)</li> <li>• Dados antigos preservados</li> <li>• Relacionamentos issue-PR mantidos</li> </ul>

## TA09 - Sincronização Full com Custom Fields

Este teste verifica a sincronização completa do repositório com extração de custom fields do GitHub Projects V2 (Size, Story Points, Worker Hours). Garante que campos personalizados sejam corretamente extraídos via GraphQL e associados às issues correspondentes, permitindo análise de accuracy posteriormente.

Campo	Descrição
<b>Identificador</b>	TA09
<b>Casos de Uso</b>	UC07
<b>Necessidade</b>	Utilização de Contexto Histórico (N3)
<b>Caso de teste</b>	Sincronização completa com extração de custom fields do GitHub Projects V2
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>Repositório usa GitHub Projects V2</li> <li>Custom fields configurados: Size (XS-XL), Story Points, Worker Hours</li> </ul>
<b>Dados de entrada</b>	Comando: set sync --full --custom-fields
<b>Ações</b>	<ol style="list-style-type: none"> <li>Modo full sync ativado (since = nil)</li> <li>GraphQL query incluindo Projects API</li> <li>Sistema extrai custom fields de cada issue</li> <li>Dados enriquecidos persistidos no BoltDB</li> <li>Índice atualizado</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li><b>Custom fields</b> extraídos corretamente</li> <li>Issues com Size (XS/S/M/L/XL) identificadas</li> <li>Story Points numerados capturados</li> <li>Worker Hours real vinculado</li> <li>Dados disponíveis para cálculo de accuracy</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>100% das issues com custom fields processadas</li> <li>Tipos de dados validados (Size enum, Points int, Hours float)</li> <li>Dados consultáveis via set inspect --custom</li> </ul>

- 
- Accuracy calculável (estimado vs real)
- 

## TA10 - Análise de Histórico

Este teste valida a funcionalidade de consulta e filtragem de dados históricos armazenados localmente no BoltDB. Verifica que o sistema pode listar issues com e sem custom fields, aplicar filtros por labels, e fornecer resultados paginados com boa performance mesmo com grandes volumes de dados.

Campo	Descrição
<b>Identificador</b>	TA10
<b>Casos de Uso</b>	UC05
<b>Necessidade</b>	Utilização de Contexto Histórico (N3)
<b>Caso de teste</b>	Consultar e filtrar dados históricos
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Base histórica sincronizada</li> <li>• ≥50 issues com dados completos</li> </ul>
<b>Dados de entrada</b>	Comandos: <ul style="list-style-type: none"> <li>• set inspect --list</li> <li>• set inspect --list --custom</li> <li>• set inspect --filter backend --limit 20</li> </ul>
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Listar todas as issues armazenadas</li> <li>2. Listar apenas issues com custom fields</li> <li>3. Filtrar por label e limitar resultados</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• Lista paginada de issues exibida</li> <li>• Custom fields destacados quando disponíveis</li> <li>• Filtros funcionando corretamente</li> <li>• Performance: &lt;2s para consulta de 1000+ issues</li> </ul>

---

---

<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• Índice B+tree do BoltDB utilizado</li> <li>• Paginação eficiente</li> <li>• Ordenação por data (mais recentes primeiro)</li> <li>• Estatísticas agregadas opcionais</li> </ul>
-------------------------------	---

---

### 6.1.5 Exportação de Dados (UC06)

Esta seção testa a capacidade do sistema de exportar dados históricos e métricas de estimativas para ferramentas externas. Os testes verificam que os dados podem ser exportados em formatos compatíveis com planilhas (CSV), sistemas de documentação (Markdown), e processamento programático (JSON), facilitando análise externa e integração com outras ferramentas.

#### TA11 - Exportação para Ferramentas Externas

Este teste valida a exportação de dados históricos, estimativas e métricas de accuracy em múltiplos formatos. Garante que os arquivos gerados sejam válidos, importáveis em ferramentas externas, e contenha todas as métricas relevantes para análise de eficácia do sistema.

Campo	Descrição
<b>Identificador</b>	TA11
<b>Casos de Uso</b>	UC06
<b>Necessidade</b>	Integração com Ferramentas Existentes (N4)
<b>Caso de teste</b>	Exportar dados históricos para análise externa
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• Base histórica com estimativas</li> <li>• Dados de accuracy disponíveis</li> </ul>
<b>Dados de entrada</b>	Comandos: <ul style="list-style-type: none"> <li>• set export --format csv --output metrics.csv</li> <li>• set export --format json --output data.json</li> <li>• set export --format markdown --output report.md</li> </ul>

---

<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Coletar todas as estimativas geradas</li> <li>2. Calcular métricas (accuracy, confiança média, etc.)</li> <li>3. Formatar conforme especificação</li> <li>4. Salvar em arquivo</li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• <b>CSV:</b> Importável em Excel/Google Sheets</li> <li>• <b>JSON:</b> Schema completo, parseável</li> <li>• <b>Markdown:</b> Relatório formatado com tabelas e gráficos ASCII</li> <li>• Métricas incluídas: accuracy, confidence distribution, size breakdown</li> </ul>
<b>Critérios de aceitação</b>	<ul style="list-style-type: none"> <li>• Headers descritivos no CSV</li> <li>• UTF-8 encoding sem BOM</li> <li>• JSON pretty-printed (indentação)</li> <li>• Markdown com TOC e seções organizadas</li> </ul>

## 6.2 Testes de Integração

Esta seção valida a integração completa com a GitHub API (REST e GraphQL), incluindo autenticação, paginação, extração de custom fields do GitHub Projects V2, e persistência de dados no BoltDB. Também testa a integração com a OpenAI API para geração de estimativas usando modelos de linguagem (GPT-4, GPT-4o). Os testes validam a construção correta de prompts, chamadas à API com parâmetros apropriados, parsing de respostas JSON.

### 6.2.1 Integração com GitHub API

Este teste valida o fluxo end-to-end de sincronização de dados do GitHub, incluindo autenticação via token, chamadas paginadas à API REST, serialização de dados para JSON, e persistência atômica no BoltDB. Verifica que relacionamentos entre issues e PRs são preservados e que transações garantem integridade dos dados.

Campo	Descrição
<b>Identificador</b>	TI01
<b>Componentes</b>	GitHubClient → GitHub REST API → BoltRepository → BoltDB

<b>Caso de teste</b>	Fluxo completo de sincronização e persistência
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>Token GitHub válido com scopes: repo, read:org</li> <li>Repositório público ou acessível</li> <li>BoltDB em <code>~/.set/data.db</code></li> </ul>
<b>Dados de entrada</b>	Comando: <code>set sync --repository user/project</code>
<b>Ações</b>	<ol style="list-style-type: none"> <li><b>GitHubClient:</b> Autenticação via token</li> <li><b>GitHubClient:</b> GET <code>/repos/user/project</code> (validação)</li> <li><b>GitHubClient:</b> GET <code>/repos/user/project/issues</code> (paginado)</li> <li><b>GitHubClient:</b> GET <code>/repos/user/project/pulls</code> (paginado)</li> <li><b>BoltRepository:</b> Serialização para JSON</li> <li><b>BoltDB:</b> Batch insert em bucket "issues"</li> <li><b>BoltDB:</b> Batch insert em bucket "prs"</li> <li><b>Metadata:</b> Atualização de <code>lastSyncTime</code></li> </ol>
<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li><b>Issues:</b> Todas persistidas (verificar count)</li> <li><b>Pull Requests:</b> Todas persistidas</li> <li><b>Relacionamentos:</b> <code>issue.number ↔ pr.number</code> preservados</li> <li><b>Integridade:</b> Dados recuperáveis via <code>set inspect</code></li> <li><b>Transações:</b> Atomicidade garantida</li> <li><b>Índices:</b> B+tree criado para buscas rápidas</li> </ul>
<b>Validação</b>	<ul style="list-style-type: none"> <li>Comparar count GitHub vs BoltDB</li> <li>Verificar schema dos objetos persistidos</li> <li>Testar query de similaridade após sync</li> <li>Validar relacionamentos (foreign keys)</li> </ul>
<b>Tratamento de erros</b>	<ul style="list-style-type: none"> <li>Rate limit 403: Retry com exponential backoff</li> <li>Network timeout: Retry até 3 vezes</li> <li>Corrupted data: Rollback de transação</li> </ul>

### 6.2.2 Integração com OpenAI API

Este teste valida o fluxo completo de geração de estimativa via OpenAI API, desde a construção do prompt com contexto histórico até o parsing da resposta JSON. Verifica que o sistema envia headers corretos (Authorization, Content-Type), utiliza parâmetros apropriados (temperature, max\_tokens), e trata erros da API adequadamente.

Campo	Descrição
<b>Identificador</b>	TI02
<b>Componentes</b>	EstimationService → PromptBuilder → AIClient → OpenAI API → ResponseParser
<b>Caso de teste</b>	Fluxo completo de estimativa com IA
<b>Pré-condições</b>	<ul style="list-style-type: none"> <li>• API key OpenAI válida</li> <li>• Créditos disponíveis</li> <li>• Modelo configurado: gpt-4 ou gpt-4o</li> </ul>
<b>Dados de entrada</b>	<p>Task: "Implementar cache Redis para sessões de usuário com TTL configurável"</p> <p>Similar tasks: 3 tarefas de cache (5h, 8h, 6h)</p>
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. <b>PromptBuilder:</b> Construir system prompt (papel da IA)</li> <li>2. <b>PromptBuilder:</b> Construir user prompt com contexto:             <ul style="list-style-type: none"> <li>- Descrição da tarefa</li> <li>- Dataset statistics (média: 6.3h, mediana: 6h)</li> <li>- Similar tasks com detalhes</li> <li>- Category breakdown</li> </ul> </li> <li>3. <b>AIClient:</b> POST <a href="https://api.openai.com/v1/chat/completions">https://api.openai.com/v1/chat/completions</a></li> <li>4. <b>Request:</b> Model: gpt-4, temperature: 0.3, max_tokens: 4000</li> <li>5. <b>OpenAI:</b> Processar e retornar estimativa</li> <li>6. <b>ResponseParser:</b> Parse JSON response</li> <li>7. <b>Validation:</b> Validar schema da resposta</li> </ol>

<b>Resultado esperado</b>	<ul style="list-style-type: none"> <li>• <b>HTTP 200:</b> Sucesso da requisição</li> <li>• <b>Response Schema:</b>  <pre>json&lt;br&gt;{&lt;br&gt; "estimated_hours": 7.5,&lt;br&gt; "estimated_size": "M",&lt;br&gt; "story_points": 5,&lt;br&gt; "confidence_score": 0.85,&lt;br&gt; "reasoning": "...",&lt;br&gt; "risks": ["cache invalidation", "redis connection"]&lt;br&gt; }&lt;br&gt;</pre> </li> <li>• <b>Token Usage:</b> Logged (prompt: ~800, completion: ~300)</li> <li>• <b>Response time:</b> ≤ 15 segundos</li> <li>• <b>Confidence:</b> Calculado com base em similaridade</li> </ul>
<b>Validação</b>	<ul style="list-style-type: none"> <li>• Verificar headers da requisição (Authorization, Content-Type)</li> <li>• Validar JSON response contra schema</li> <li>• Confirmar tipos de dados corretos</li> <li>• Log de token usage para monitoramento</li> </ul>
<b>Tratamento de erros</b>	<ul style="list-style-type: none"> <li>• 401 Unauthorized: Erro de API key</li> <li>• 429 Rate Limit: Backoff e retry</li> <li>• 500 Server Error: Fallback para método baseado em similaridade</li> </ul>

## 7. Cronograma e Processo de Implementação

Esta seção apresenta o cronograma detalhado para implementação do sistema SET CLI e descreve o processo que será seguido durante o desenvolvimento. O cronograma foi estruturado considerando as datas marco estabelecidas e organizando as atividades em sprints quinzenais para facilitar o acompanhamento e entregas incrementais.

### 7.1 Cronograma de Implementação

Período	Marco	Atividades Principais	Entregáveis

		<ul style="list-style-type: none"> <li>• Setup do Projeto e Estrutura Inicial</li> <li>• Configuração do repositório Git</li> <li>• Setup do ambiente Go com módulos</li> <li>• Implementação da estrutura CLI com Cobra</li> <li>• Projeto Go inicializado</li> <li>• Comandos CLI básicos funcionais</li> <li>• README com instruções de setup</li> </ul>
16/09 - 29/09	<b>A2 - Boilerplate</b>	<ul style="list-style-type: none"> <li>• Sistema de Configuração e GitHub Integration</li> <li>• Implementação do Configuration Service</li> <li>• Desenvolvimento do GitHub Client</li> <li>• Sistema de autenticação com tokens</li> <li>• Comandos de configuração inicial</li> <li>• Validação de conectividade</li> <li>• Comando set configure funcional</li> <li>• Integração GitHub API implementada</li> <li>• Validação de tokens funcionando</li> <li>• Testes unitários das integrações</li> <li>• Documentação da configuração</li> </ul>
30/09 - 13/10		<ul style="list-style-type: none"> <li>• Core de Estimativas e Persistência Local</li> <li>• Implementação do EstimationService</li> <li>• Desenvolvimento do BoltRepository</li> <li>• Sistema de busca de similaridade</li> <li>• Comando básico de estimativa individual</li> <li>• Estrutura de dados históricos</li> <li>• Comando set estimate funcional</li> <li>• Persistência local operacional</li> <li>• Algoritmo de similaridade básico</li> <li>• Testes de integração com BoltDB</li> <li>• Documentação das APIs internas</li> </ul>
14/10 - 20/10	<b>A3 - Core Funcional</b>	

		<ul style="list-style-type: none"> <li>• Integração com IA e Análise Avançada</li> <li>• Implementação do AI Client (OpenAI)</li> <li>• Desenvolvimento do Similarity Engine</li> <li>• Integração IA + dados históricos</li> <li>• Sistema de confiança das estimativas</li> <li>• Tratamento de erros e fallbacks</li> <li>• IA integrada às estimativas</li> <li>• Sistema de confiança funcional</li> <li>• Tratamento robusto de erros</li> <li>• Testes com mocks da API OpenAI</li> <li>• Métricas de accuracy implementadas</li> </ul>
21/10 - 03/11		<ul style="list-style-type: none"> <li>• Funcionalidades Avançadas e Otimização</li> <li>• Implementação do comando batch</li> <li>• Desenvolvimento do AnalysisService</li> <li>• Sistema de relatórios básico</li> <li>• Cache e otimização de performance</li> <li>• Comando de calibração</li> <li>• Comando set estimate --batch funcional</li> <li>• Análise histórica implementada</li> <li>• Sistema de cache operacional</li> <li>• Relatórios básicos funcionando</li> <li>• Performance otimizada</li> </ul>
04/11 - 10/11	<b>A4 - Implementação Finalizada</b>	<ul style="list-style-type: none"> <li>• Relatórios Executivos e Export</li> <li>• Implementação do ReportService completo</li> <li>• Sistema de exportação (CSV, PDF)</li> <li>• Dashboards e visualizações</li> <li>• Comandos de análise avançada</li> <li>• Testes de integração completos</li> <li>• Sistema completo de relatórios</li> <li>• Exportação funcionando</li> <li>• Todos os comandos implementados</li> <li>• Suite completa de testes</li> <li>• Cobertura de testes &gt;85%</li> </ul>
11/11 - 24/11		

18/11 - 23/11	<b>A5 - Revisão Final</b>	<ul style="list-style-type: none"> <li>• Testes Finais e Documentação</li> <li>• Execução de todos os casos de teste</li> <li>• Correção de bugs identificados</li> <li>• Documentação técnica completa</li> <li>• Guias de usuário e instalação</li> <li>• Revisão de código finalizada</li> </ul>	<ul style="list-style-type: none"> <li>• Todos os testes passando</li> <li>• Guias de instalação e uso</li> <li>• Documentação final entregue</li> <li>• Projeto pronto para uso</li> </ul>
24/11 - 30/11		<ul style="list-style-type: none"> <li>• Preparação da apresentação</li> </ul>	<ul style="list-style-type: none"> <li>• Apresentação preparada</li> </ul>

## 7.2 Processo de Implementação

Esta seção descreve o processo de desenvolvimento real utilizado durante a implementação do sistema SET CLI. Por se tratar de um projeto individual desenvolvido por um único desenvolvedor, optou-se por uma abordagem pragmática e incremental, focada em entregar valor funcional rapidamente e evoluir a robustez do sistema de forma iterativa. O cronograma para implementação não foi estritamente seguido e o desenvolvimento se deu pela disponibilidade do desenvolvedor.

**Desenvolvimento Incremental Orientado a Requisitos:** O processo de implementação priorizou a entrega de funcionalidades completas seguindo o fluxo de valor dos requisitos, ao invés de seguir um processo ágil rígido com cerimônias formais. A estratégia foi:

1. **Requisitos como guia:** Cada ciclo de desenvolvimento focou em implementar um ou mais casos de uso completos (UC01, UC02, etc.), garantindo que os requisitos funcionais fossem atendidos de forma incremental

2. **Caminho Feliz:** A primeira implementação de cada funcionalidade focou no caminho feliz - cenários onde tudo funciona conforme esperado, sem tratamento extensivo de erros ou casos extremos
3. **Evolução da Resiliência:** Após o caminho feliz funcional, o sistema evoluiu para adicionar:
  - o Validação de entradas
  - o Tratamento de erros
  - o Fallbacks e recuperação
  - o Logs e observabilidade
  - o Testes de casos extremos
4. **Entregas Funcionais:** Cada incremento resultava em software executável e testável, permitindo validação contínua do progresso

O software foi majoritariamente desenvolvido aos finais de semana devido a limitação de tempo. Mesmo assim ele foi finalizado na entrega A3 permitindo o foco na documentação para o restante das entregas.

## 8. Post-mortem

### 8.1 Experiência Positivas

Durante o desenvolvimento do sistema SET CLI, diversos aspectos positivos podem ser destacados:

**Escolha da Linguagem Go:** A decisão de utilizar Go como linguagem principal demonstrou-se extremamente acertada. A facilidade de criar um binário único multiplataforma simplificou significativamente a distribuição, enquanto a performance nativa e o excelente suporte para concorrência permitiram processamento paralelo eficiente em operações de batch. Além disso, toda experiência prévia dos envolvidos com a linguagem contribuiu com a velocidade de desenvolvimento.

**Arquitetura Clean Architecture:** A aplicação dos princípios de Clean Architecture desde o início do projeto facilitou a manutenção, testabilidade e evolução do código. A separação clara entre camadas de domínio, casos de uso e infraestrutura permitiu mudanças e extensões sem impacto em outras partes do sistema.

**Integração com IA (OpenAI):** O uso da API OpenAI para geração de estimativas baseadas em linguagem natural superou as expectativas. A capacidade da IA de compreender contexto e gerar raciocínios explicativos agregou valor significativo ao produto.

**BoltDB como Storage Local:** A escolha de BoltDB como banco de dados embarcado eliminou dependências externas e configurações complexas. O desempenho para as operações de leitura e escrita mostrou-se mais que adequado para o volume de dados típico.

**Framework Cobra para CLI:** O uso do Cobra como framework para a interface CLI proporcionou uma experiência de usuário consistente e profissional, com parsing de argumentos robusto, sistema de help integrado e subcomandos bem organizados.

## 8.2 Desafios e Soluções

Durante o desenvolvimento, diversos desafios técnicos e arquitetrais foram enfrentados:

Desafio	Desafio
<b>Tratamento de Dados Inconsistentes do GitHub</b>	Validação rigorosa de dados na camada de integração. Implementação de normalização e sanitização de campos antes de persistir. Logs detalhados para debug de dados problemáticos.
<b>Balanceamento entre Custo de API e Precisão</b>	Sistema de cache inteligente para evitar chamadas repetitivas. Batch processing para reduzir overhead por requisição.
<b>Definição de Thresholds para Similaridade</b>	Implementação de threshold adaptativo que se ajusta baseado na quantidade e qualidade de dados históricos disponíveis. Múltiplos níveis de confiança informados ao usuário.
<b>Sincronização com GitHub Rate Limits</b>	Implementação de rate limiting awareness com backoff exponencial. Cache agressivo de dados já sincronizados. Sync incremental baseado em timestamps.

## 8.3 Lições Aprendidas

**Importância de dataset de qualidade:** A qualidade das estimativas está diretamente relacionada à qualidade e quantidade dos dados históricos. Projetos com menos de 50 issues têm estimativas significativamente menos confiáveis. Recomenda-se aguardar o acúmulo de histórico antes de confiar cegamente nas estimativas.

**Validação de inputs é essencial:** Diversos bugs em produção foram evitados graças a validações rigorosas de entrada. Especialmente importante para dados vindos de APIs externas que podem ter formatos inconsistentes.

**CLI UX Requer Feedback Claro:** Para ferramentas CLI, feedback visual claro é crucial. Implementação de progress bars, colorização de output e mensagens de erro descritivas melhorou significativamente a interpretação dos dados.

**Trade-offs de Performance vs Precisão:** Nem sempre vale a pena buscar a precisão máxima se isso compromete muito a performance. Encontrar o ponto de equilíbrio requer experimentação e métricas claras.

## 8.4 Melhorias Futuras

Baseado na experiência de desenvolvimento, as seguintes melhorias são recomendadas para versões futuras:

**Suporte a Múltiplos Provedores de IA:** Implementar interfaces agnósticas para permitir uso de Anthropic Claude, Google Gemini, ou outros modelos além de OpenAI. Isso reduziria vendor lock-in e permitiria comparação de resultados.

**Interface Web Complementar:** Embora a CLI seja o foco, uma interface web leve para visualização de métricas e dashboards agregaria valor para Product Owners e stakeholders não-técnicos.

**Integração com Jira/Azure DevOps:** Expandir além do GitHub para suportar outras ferramentas de gestão de projetos amplamente usadas.

**Machine Learning Local:** Implementar um modelo de ML local (treinado nos dados históricos do time) como fallback quando a API está indisponível ou para reduzir custos.

**Suporte a Múltiplos Repositórios:** Permitir análise consolidada de múltiplos repositórios para empresas que trabalham com micro serviços.

**Notificações e Alertas:** Sistema de alertas quando accuracy começa a degradar ou quando padrões suspeitos são detectados.

## 8.5 Repositório do Trabalho

O código-fonte completo do projeto SET CLI, incluindo toda a documentação e casos de teste e histórico de desenvolvimento, está disponível publicamente no GitHub:

**Repositório Oficial:**

<https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2025-2-tcc-0393100-dev-inacio-moraes>