
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**.

23 de novembro 2025

Tabela de Conteúdo

1. Introdução	1
2. Modelos de Usuário e Requisitos	1
2.1 Descrição de Atores	1
2.2 Modelos de Usuários (Personas)	1
2.3 Modelo de Casos de Uso e Histórias de Usuários	7
2.4 Diagrama de Sequência do Sistema e Contrato de Operações	12
3. Modelos de Projeto	21
3.1 Diagrama de Classes	21
3.2 Diagramas de Sequência	30
3.3 Diagramas de Comunicação	39
3.4 Arquitetura	40
3.5 Diagramas de Estados	44
3.6 Diagrama de Componentes e Implantação.	45
4. Projeto de Interface com Usuário	47
5. Glossário e Modelos de Dados	53
6. Casos de Teste	60
6.1 Testes de Aceitação	61
6.2 Testes de Integração	73
7. Cronograma e Processo de Implementação	77
7.1 Processo de Implementação	80
8. Post-mortem	80
8.1 Experiência Positivas	80
8.2 Desafios e Soluções	81
8.3 Lições Aprendidas	82
8.4 Melhorias Futuras	82
8.5 Repositório do Trabalho	83

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

A estimativa de esforço em projetos de software representa um desafio crítico no contexto do desenvolvimento ágil moderno. Metodologias como Scrum e práticas como Planning Poker dependem fundamentalmente da capacidade das equipes de prever com precisão o tempo e complexidade necessários para implementar funcionalidades. Contudo, a natureza subjetiva dessas técnicas, aliada à ausência de mecanismos sistemáticos para aproveitamento de dados históricos, resulta em estimativas frequentemente imprecisas que comprometem o planejamento de sprints, a previsibilidade de entregas e a alocação eficiente de recursos. Esta problemática é agravada pelo fato de que, embora as equipes acumulem vasto histórico de issues, *pull requests* e métricas de desenvolvimento em plataformas como GitHub, raramente esses dados são analisados de forma estruturada para calibrar estimativas futuras.

O SET (Software Estimation Tool) é uma ferramenta de linha de comando (CLI) especializada em análise preditiva de esforço, desenvolvida em Go, que utiliza modelos de inteligência artificial para gerar estimativas fundamentadas em dados históricos. Diferentemente de sistemas convencionais de gerenciamento de projetos que focam em rastreamento e organização de tarefas, o SET posiciona-se como uma ferramenta analítica que extrai conhecimento de repositórios GitHub, aplica técnicas de similaridade textual (TF-IDF e cosseno) para identificar padrões em tarefas passadas, e utiliza a API OpenAI para sintetizar estimativas contextualizadas em horas, story points e classificação de tamanho (XS a XL). A arquitetura da solução é fundamentada em princípios de Clean Architecture, garantindo separação clara entre camadas de domínio, aplicação e infraestrutura, com persistência local via BoltDB.

Os desafios técnicos enfrentados no desenvolvimento do SET abrangem múltiplas dimensões da engenharia de software. No âmbito de performance, a ferramenta processa lotes de dezenas de tarefas simultaneamente através de worker pools implementados com goroutines, respeitando rate limits das APIs GitHub e OpenAI. A integração com GitHub Projects para extração de custom fields (Worker Hours, Story Points) via GraphQL requereu tratamento robusto de esquemas heterogêneos e sincronização incremental baseada em timestamps para minimizar consumo de API quota.

Este documento estrutura-se em oito seções principais que detalham todos os aspectos do projeto SET. A Seção 2 apresenta modelos de usuário através de personas detalhadas (Desenvolvedor, Scrum Master, Product Owner), casos de uso com diagramas e contratos de operações. A Seção 3 documenta os modelos de projeto incluindo diagramas de classes organizados por pacotes Go (ai, estimator, github, storage, config), diagramas de sequência para os fluxos críticos (estimativa individual, batch processing, sincronização), diagramas de comunicação e a arquitetura C4 do sistema. A Seção 4 apresenta o projeto de interface CLI com exemplos de output para cada comando. A Seção 5 define o glossário técnico e modelos de dados detalhando as entidades Task, Estimation, Issue e PullRequest com seus relacionamentos. A Seção 6 especifica casos de teste de aceitação e integração cobrindo todas as necessidades identificadas. A Seção 7 apresenta o cronograma de implementação e processo de desenvolvimento adotado. Por fim, a Seção 8 oferece

uma análise post-mortem com experiências positivas, desafios enfrentados e melhorias futuras recomendadas.

2. Modelos de Usuário e Requisitos

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.

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.

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.

A Tabela 2.1 detalha o perfil do Scrum Master, Carlos Mendes, que utilizará o SET principalmente para otimizar o planejamento de *sprints*, aumentar a precisão das estimativas e obter dados concretos para justificar a capacidade da equipe. Seus cenários de uso focam em comandos de sincronização, processamento em lote (*batch*) e exportação de relatórios para melhoria contínua e comunicação com a gestão.

Aspecto	Detalhes
Nome	Carlos Mendes
Idade e Perfil	35 anos, Scrum Master com 8 anos de experiência em metodologias ágeis
Descrição	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.
Contexto Técnico	<ul style="list-style-type: none"> • Utiliza Jira, Azure DevOps e GitHub diariamente • Familiarizado com CLI tools (git, docker, kubectl) • Experiência com ferramentas de métricas (SonarQube, New Relic) • Confortável com terminal Linux/Mac
Dores Principais	<ul style="list-style-type: none"> • Planning Poker Subjetivo: "As estimativas do planning poker variam muito entre os desenvolvedores e nem sempre refletem a realidade" • Falta de Dados Históricos: "Não temos uma base sólida de dados para justificar nossas estimativas para o PO" • Tempo Gasto em Reuniões: "Gastamos 2-3 horas por sprint só em estimativas e nem sempre acertamos" • Inconsistência Entre Times: "Cada time estima de forma diferente, dificultando comparações"
Objetivos	<ul style="list-style-type: none"> • Reduzir tempo de reuniões de Planning em 50% • Aumentar precisão das estimativas para > 80% • Ter dados concretos para apresentar ao management

	<ul style="list-style-type: none"> • Melhorar previsibilidade de entrega dos times • Identificar padrões de super/subestimação
Cenários de Uso	<p>Preparação de Sprint Planning:</p> <ul style="list-style-type: none"> • <code>set sync --custom-fields</code> • <code>set batch --file backlog.json --format csv --output sprint-15.csv</code> • <code>set batch --file backlog.json --format markdown --output planning-report.md</code> • <code>set batch --file backlog.json --workers 10 --progress</code> <p>Revisão de Sprint:</p> <ul style="list-style-type: none"> • <code>set inspect --list --custom --limit 50</code> • <code>set export --format csv --output team-metrics.csv</code> • <code>set export --format markdown --output sprint-summary.md</code>
Métricas de Sucesso	<ul style="list-style-type: none"> • Redução de 50% no tempo de planning • Acurácia > 80% nas estimativas • Velocity mais previsível • Menos re-estimativas durante o <i>sprint</i>

Tabela 2.1 - Persona 1: Scrum Master

A Tabela 2.2 apresenta Ana Rodrigues, Desenvolvedora Sênior, cujo uso da ferramenta visa fundamentar suas próprias estimativas e as do seu time, comparando-as com o histórico e reduzindo a subjetividade do processo. Seus cenários de uso incluem comandos de estimativa individual com sugestões baseadas em dados históricos (`--show-similar`), além da análise e exportação das suas próprias métricas de desempenho.

Aspecto	Detalhes
Nome	Ana Rodrigues
Idade e Perfil	29 anos, Desenvolvedor Full Stack Sênior com 6 anos de experiência

Descrição	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.
Contexto Técnico	<ul style="list-style-type: none">• Expert em Git, GitHub Actions, Docker• Usuária avançada de terminal (Zsh, Vim)• Experiência com ferramentas CLI (make, npm, go build)• Trabalha com múltiplos repositórios simultaneamente
Dores Principais	<ul style="list-style-type: none">• Pressão por Estimativas: "Sempre me pedem para estimar tarefas complexas e a responsabilidade é grande"• Esquecimento de Detalhes: "Às vezes esqueço de considerar testes, refactoring ou edge cases"• Comparação Difícil: "É difícil lembrar quanto tempo levou uma tarefa similar há 3 meses"• Estimativas de Juniores: "Os desenvolvedores juniors sempre subestimam por falta de experiência"
Objetivos	<ul style="list-style-type: none">• Ter base de dados para fundamentar estimativas• Ajudar desenvolvedores juniors com referências• Melhorar a própria auto calibragem• Reduzir ansiedade nas estimativas• Documentar padrões de desenvolvimento

Cenários de Uso	Estimativa Individual: <ul style="list-style-type: none"> • <code>set estimate "Implementar auth JWT" --labels backend,security --show-similar</code> • <code>set estimate "Criar API REST" --labels backend --similar --output json</code> • <code>set estimate "Refatorar módulo" --description "Melhorar arquitetura" --labels refactor</code> Análise de Dados: <ul style="list-style-type: none"> • <code>set inspect --list --custom</code> • <code>set sync --custom-fields</code> • <code>set export --format json --output my-estimates.json</code>
Métricas de Sucesso	<ul style="list-style-type: none"> • Menor variação entre estimado e real • Melhoria na mentoria de juniores • Redução de stress em estimativas

Tabela 2.2 - Persona 2: Desenvolvedor Sênior

O perfil de Roberto Santos, Product Owner, é descrito na Tabela 2.3, destacando seu background técnico que o capacita a usar o SET para planejamento de *releases* mais confiável e tomada de decisão *data-driven*. Seus comandos envolvem o processamento em lote de *epics* ou listas de *features* e exportação de dados em formatos variados para relatórios de *stakeholders* e sistemas de gerenciamento de projetos como Jira e GitHub.

Aspecto	Detalhes
Nome	Roberto Santos
Idade e Perfil	42 anos, Product Owner com background como desenvolvedor (10 anos) e 4 anos em produtos digitais
Descrição	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.

Contexto Técnico	<ul style="list-style-type: none">Ex-desenvolvedor: 10 anos de experiência com Python, JavaScript e GoConfortável com CLI: Usa terminal diariamente (git, docker, npm)GitHub power user: Revisa PRs ocasionalmente, gerencia issuesFerramentas de desenvolvimento: Familiarizado com IDEs, CI/CD, ferramentas DevOps
Dores Principais	<ul style="list-style-type: none">Pressão de Prazos: "O CEO sempre quer saber quando a feature vai estar pronta"Falta de Visibilidade: "Não sei se as estimativas são realistas até ser tarde demais"Comunicação com Stakeholders: "É difícil explicar por que uma 'simples' feature leva 2 sprints"Priorização Sem Dados: "Preciso escolher entre features sem saber o real esforço de cada uma"
Objetivos	<ul style="list-style-type: none">Ter previsões confiáveis para planejamento de releasesDados para justificar priorizaçõesComunicação clara com stakeholdersMelhor ROI nas decisões de produtoReducir riscos de atraso em releases
Cenários de Uso	<p>Planejamento de Release:</p> <ul style="list-style-type: none"><code>set batch --file epic-user-management.json --format markdown --output epic-estimate.md</code><code>set batch --file q1-features.csv --format csv --output roadmap.csv</code><code>set export --format jira --output jira-import.csv</code> <p>Acompanhamento:</p> <ul style="list-style-type: none"><code>set inspect --list</code><code>set export --format github --output github-import.csv</code><code>set export --format excel --output stakeholder-report.csv</code><code>set export --filter feature --format markdown --output features-report.md</code>

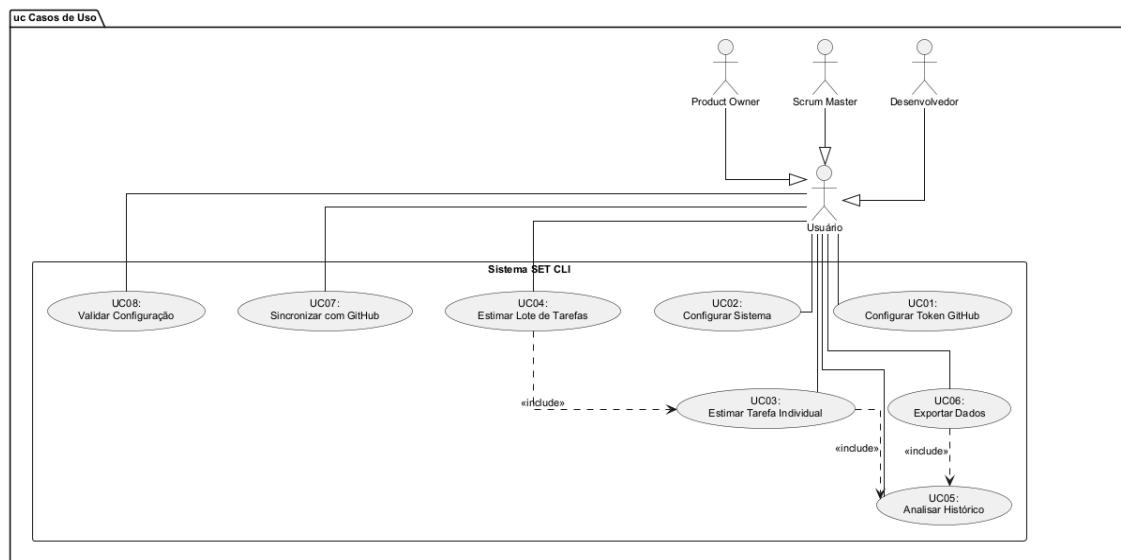
- | | |
|----------------------------|--|
| Métricas de Sucesso | <ul style="list-style-type: none"> • Releases entregues no prazo (+90%) • Melhor comunicação com stakeholders • Decisões de priorização data-driven • Redução de scope creep |
|----------------------------|--|

Tabela 2.3 - Persona 3: Product Owner

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**

A Figura 2.1 apresenta o diagrama de caso de uso do sistema SET CLI implementado. Para fins de organização, utiliza-se identificadores no formato UC#ID, onde UC refere-se a Use Case. Na tabela 2.4 podemos observar todos os casos de uso bem como seu comando na CLI.

Código	Caso de Uso	CLI
--------	-------------	-----

UC01	Configurar Token GitHub	<i>set configure --github-token</i>
UC02	Configurar Sistema	<i>set configure --initial</i>
UC03	Estimar Tarifa Individual	<i>set estimate "<descrição>"</i>
UC04	Estimar Lote de Tarefas	<i>set batch --file <arquivo></i>
UC05	Analizar Histórico	<i>set inspect [--list]</i>
UC06	Exportar Dados	<i>set export --format <csv json markdown jira github excel></i>
UC07	Sincronizar com GitHub	<i>set sync [--custom-fields]</i>
UC08	Validar Configuração	<i>set configure --validate</i>

Tabela 2.4 - Casos de Uso

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.

A Tabela 2.5 - Histórias do Desenvolvedor detalha as necessidades do desenvolvedor ao utilizar a CLI SET, focando na configuração inicial com o token GitHub (US01, US07), na obtenção de estimativas para tarefas específicas com *score* de confiança (US02, US05), na visualização de tarefas similares para validação (US03), na análise de histórico pessoal (US04) e no uso de *labels* para categorização (US06).

ID	Caso de Uso	História
US01	UC01, UC02	Como desenvolvedor , eu quero configurar a CLI SET com meu token GitHub para que o sistema possa acessar dados dos meus repositórios e gerar estimativas personalizadas.
US02	UC03	Como desenvolvedor , eu quero estimar uma tarefa específica via CLI para obter uma estimativa baseada em dados históricos e IA, economizando tempo em planning.
US03	UC03	Como desenvolvedor , eu quero ver tarefas similares encontradas durante a estimativa para validar se a sugestão da IA faz sentido com minha experiência.
US04	UC05	Como desenvolvedor , eu quero analisar meu histórico pessoal de estimativas para identificar padrões e melhorar minha precisão ao estimar.
US05	UC03	Como desenvolvedor , eu quero receber um score de confiança com a estimativa para entender a confiabilidade da sugestão da IA.
US06	UC03, UC05	Como desenvolvedor , eu quero usar labels para categorizar tarefas para melhorar a precisão das estimativas baseadas em similaridade.
US07	UC08	Como desenvolvedor , eu quero validar minha configuração do sistema para garantir que o token GitHub e repositório estão corretos antes de começar a usar a ferramenta.

Tabela 2.5 - Histórias do Desenvolvedor

A Tabela 2.6 - Histórias do Scrum Master apresenta as histórias do Scrum Master, que incluem a configuração da CLI com token e repositório padrão do time (US08), a estimativa em lote de tarefas do backlog (US09, US12), a exportação e inspeção de dados históricos de estimativas (US10, US11) e a sincronização de dados do GitHub, incluindo *custom fields* como *Worker Hours* e *Story Points* (US13).

ID	Caso de Uso	História
US08	UC01, UC02	Como Scrum Master , eu quero configurar a CLI SET com meu token GitHub e repositório padrão para acessar dados históricos do time e automatizar o processo de estimativas.
US09	UC04	Como Scrum Master , eu quero estimar um lote de tarefas do backlog para preparar a reunião de Planning com dados concretos e acelerar as discussões.
US10	UC05, UC06	Como Scrum Master , eu quero exportar dados históricos de estimativas para análise externa e identificação de padrões do time.
US11	UC05	Como Scrum Master , eu quero inspecionar estatísticas do banco de dados local para entender a quantidade e distribuição de tarefas históricas.
US12	UC04, UC06	Como Scrum Master , eu quero processar múltiplas tarefas em paralelo para obter estimativas rapidamente mesmo com grandes backlogs.
US13	UC07	Como Scrum Master , eu quero sincronizar dados do GitHub incluindo custom fields para ter acesso a Worker Hours e Story Points históricos.

Tabela 2.6 - Histórias do Scrum Master

A Tabela 2.7 - Histórias do Product Owner descreve as funcionalidades requeridas pelo Product Owner, tais como a configuração da CLI para consulta e geração de relatórios (US14), a estimativa de épicos completos via *batch processing* para priorização (US16), a geração de relatórios executivos em múltiplos formatos para *stakeholders* (US15), a exportação de dados em formatos compatíveis com Jira e GitHub Projects (US17) e a visualização de estatísticas agregadas de estimativas em lote (US18).

ID	Caso de Uso	História
US14	UC01, UC02	Como Product Owner , eu quero configurar a CLI SET com acesso ao repositório do produto para poder consultar estimativas e gerar relatórios estratégicos.
US15	UC06	Como Product Owner , eu quero gerar relatórios executivos de estimativas em múltiplos formatos (CSV, JSON, Markdown) para comunicar timelines realistas aos stakeholders e management.
US16	UC04	Como Product Owner , eu quero estimar épicos completos via batch processing para comparar esforço entre diferentes features e priorizar com base em ROI.
US17	UC06	Como Product Owner , eu quero exportar dados em formatos compatíveis com Jira e GitHub Projects para integrar as estimativas ao planejamento estratégico do produto.
US18	UC04, UC06	Como Product Owner , eu quero ver estatísticas agregadas de estimativas em lote (total de horas, distribuição por tamanho, nível de confiança) para tomar decisões informadas sobre releases.

Tabela 2.7 -Histórias do Product Owner

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.

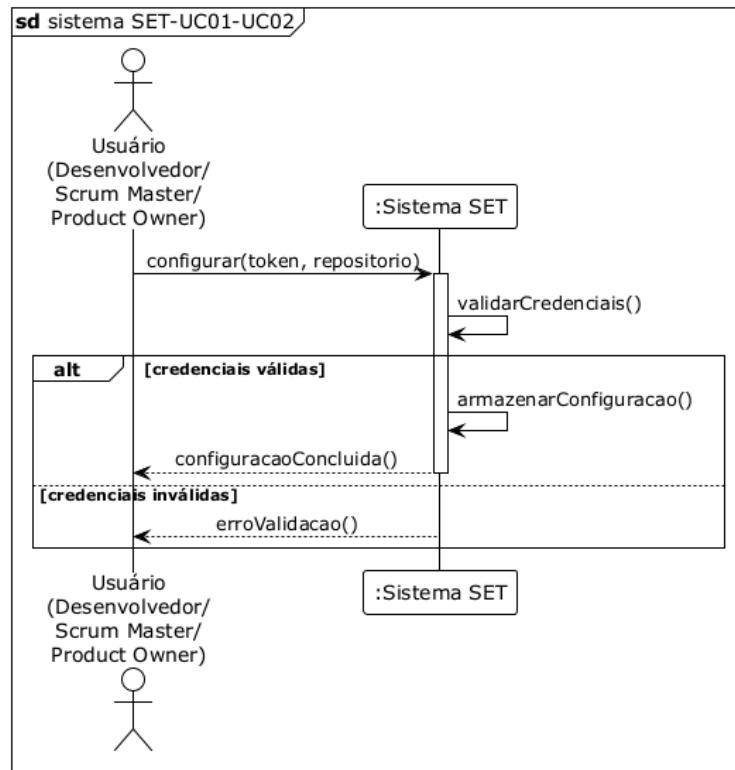
A Tabela 2.8 (Contrato: Configurar Sistema) detalha as etapas e as pré e pós-condições essenciais para a inicialização bem-sucedida da ferramenta. Este contrato assegura que o sistema esteja em um estado operacional e pronto para processar as demais operações, estabelecendo as bases para o uso da aplicação.

Elemento	Descrição
Operação	configurar(token: string, repositorio: string)

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

Tabela 2.8 - Contrato: Configurar Sistema

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

A Tabela 2.9 (Contrato: Validar Configuração) descreve o processo de checagem e verificação das credenciais de acesso e das configurações do sistema. O objetivo é garantir que o usuário ou o processo que está interagindo com o sistema possua as permissões necessárias e que todos os parâmetros de configuração estejam corretos antes da execução de tarefas mais críticas.

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

Tabela 2.9 - Contrato: Validar Configuração

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.

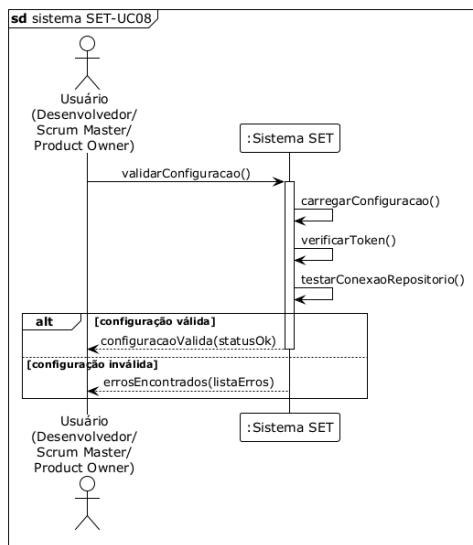


Figura 2.3 - Diagrama de Sequência do Sistema: Validar Configuração

A Tabela 2.10 (Contrato: Estimar Tarefa Individual) define o fluxo de trabalho e as regras de negócio utilizadas para calcular a estimativa de tempo ou esforço para uma única tarefa. Este contrato é fundamental para fornecer previsibilidade e auxiliar no planejamento, detalhando como os dados da tarefa são transformados em uma estimativa quantitativa.

Elemento	Descrição
Operação	<code>estimar(descricao: string, opcoes: OpcoesEstimativa)</code>
Referências	UC03, US02, US03, US05, US06
Pré-condições	<ul style="list-style-type: none"> Sistema configurado Descrição da tarefa fornecida
Pós-condições	<ul style="list-style-type: none"> Estimativa calculada e exibida Nível de confiança determinado Tarefas similares identificadas (se disponíveis)

- Estimativa registrada no histórico

Tabela 2.10 - Contrato: Estimar Tarefa Individual

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.

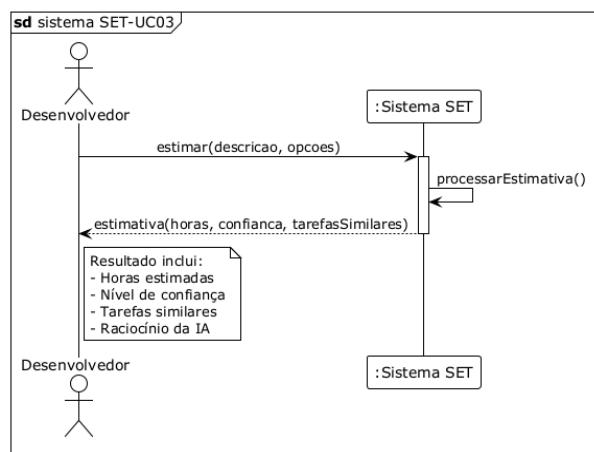


Figura 2.4 - Diagrama de Sequência do Sistema: Estimar Tarefa Individual

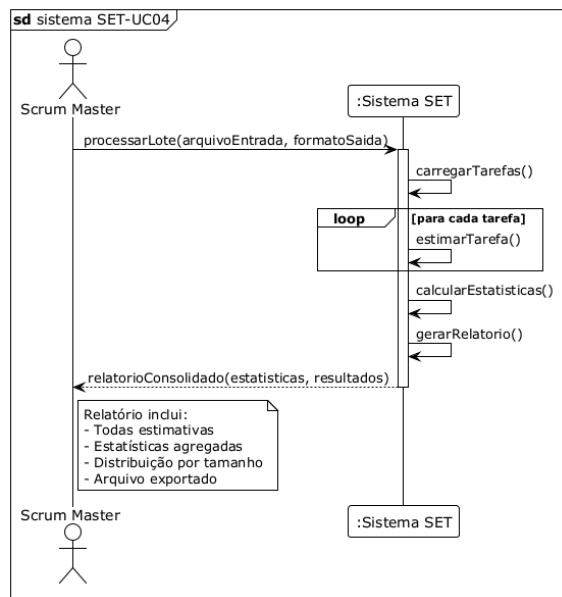
A Tabela 2.11 (Contrato: Processar Lote de Tarefas) especifica o mecanismo de estimativa para múltiplas tarefas simultaneamente. Ao contrário da estimativa individual, este contrato foca na otimização e no gerenciamento do processamento em massa, garantindo a consistência e a eficiência do cálculo de estimativas para um conjunto de tarefas.

Elemento	Descrição
Operação	processarLote(arquivoEntrada: string, formatoSaida: string)
Referências	UC04, US09, US12, US16, US18
Pré-condições	<ul style="list-style-type: none"> • Sistema configurado • Arquivo de entrada válido (JSON/CSV) • Formato de saída suportado

- | | |
|----------------------|---|
| Pós-condições | <ul style="list-style-type: none"> • Todas as tarefas estimadas • Estatísticas agregadas calculadas (total horas, distribuição, confiança média) • Arquivo de saída gerado no formato especificado • Estimativas registradas no histórico |
|----------------------|---|

Tabela 2.11 - Contrato: Processar Lote de Tarefas

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**

A Tabela 2.12 (Contrato: Sincronizar com GitHub) estabelece os procedimentos para a importação e atualização de dados históricos a partir da plataforma GitHub. Este contrato é crucial para a contextualização das análises, assegurando que o sistema possua um repositório atualizado de informações sobre o desenvolvimento e as atividades passadas.

Elemento	Descrição
----------	-----------

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

Tabela 2.12 - Contrato: Sincronizar com GitHub

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.

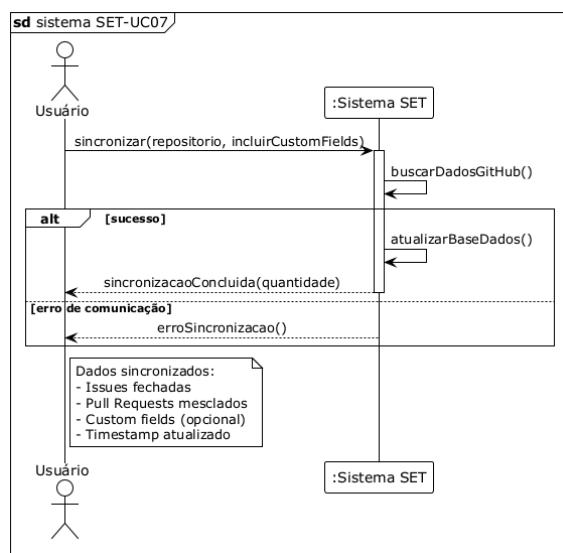


Figura 2.6 - Diagrama de Sequência do Sistema: Sincronizar com GitHub

A Tabela 2.13 (Contrato: Exportar Dados) delineia as operações para a saída de informações do sistema, suportando diversos formatos de arquivo. O foco deste contrato é a interoperabilidade e a capacidade de fornecer os dados processados para outras ferramentas ou para análise externa, detalhando os parâmetros e as regras de formatação.

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

Tabela 2.13 - Contrato: Exportar Dados

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.

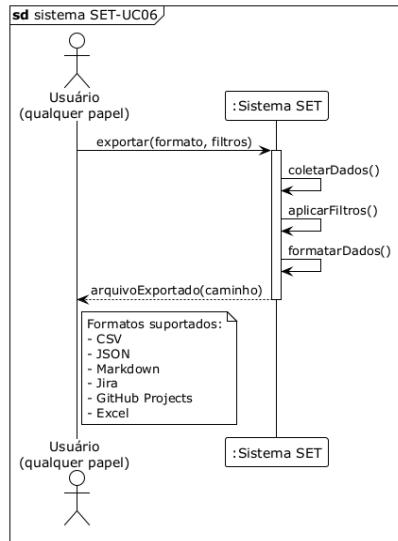


Figura 2.7 - Diagrama de Sequência do Sistema: Exportar Dados

A Tabela 2.14 (Contrato: Analisar Histórico) resume o processo de análise das informações previamente armazenadas e importadas. Este contrato, que se enquadra na área de suporte à decisão (DSS), detalha como o histórico é examinado para extrair insights e gerar relatórios, sendo a base para a inteligência e o aprendizado contínuo do sistema.

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

- Relatório exibido ao usuário

Tabela 2.14 - Contrato: Analisar Histórico

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.

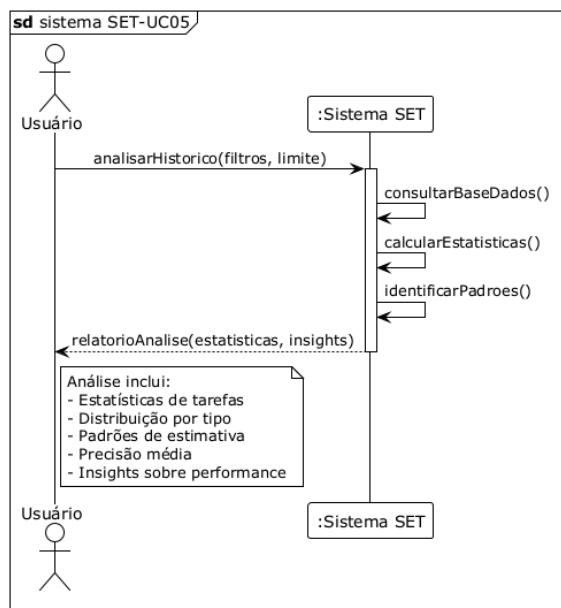


Figura 2.8 - Diagrama de Sequência do Sistema: Analisar Histórico

3. Modelos de Projeto

Nesta seção, é apresentada a modelagem dos objetos e fluxos do sistema proposto. Para isso, são utilizados os seguintes diagramas UML:

- **Diagrama de classes:** Descreve as classes que compõem o sistema e suas relações.
- **Diagrama de sequência:** Foca nos fluxos da aplicação, detalhando as mensagens trocadas entre atores e pacotes.
- **Diagrama de comunicação:** Semelhante ao diagrama de sequência, mapeia os fluxos da aplicação, mas com foco nos atores envolvidos.
- **Diagrama de arquitetura:** Descreve a estrutura da aplicação por meio de seus pacotes e as relações entre eles.
- **Diagrama de estados:** Representa a mudança de estados dos objetos que possuem estados mutáveis.
- **Diagrama de componentes e implantação:** Detalha como os diversos componentes da

aplicação se comunicam e como devem ser implantados na camada física.

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.

O pacote internal/ai, representado na Figura 3.1, 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 AIPublisher. 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. A Figura 3.1.1 ilustra o diagrama de classes que corresponde ao pacote ai.

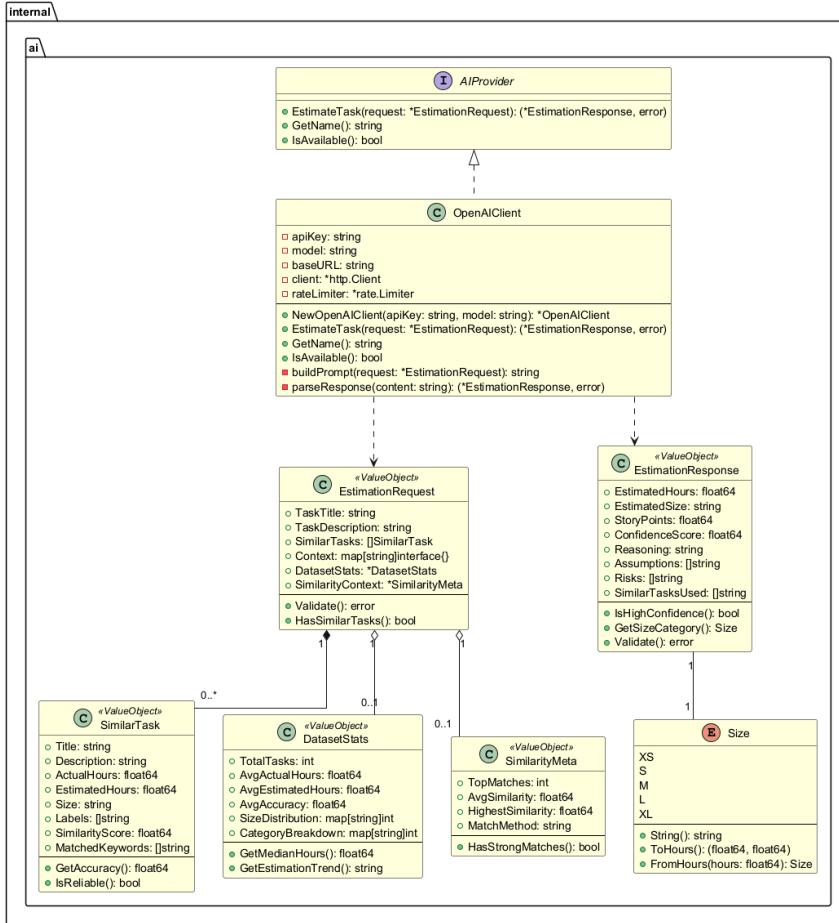


Figura 3.1 - Diagrama de Classes: Pacote AI

Os componentes principais desse diagrama são:

- **AIProvider (Interface):** Abstração que permite diferentes implementações de provedores de IA (OpenAI, Claude, modelos locais). Define o contrato para estimativa 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.

O pacote internal/estimator, representado na Figura 3.2, 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. A Figura 3.1.2 ilustra o diagrama de classes que corresponde ao pacote estimador.

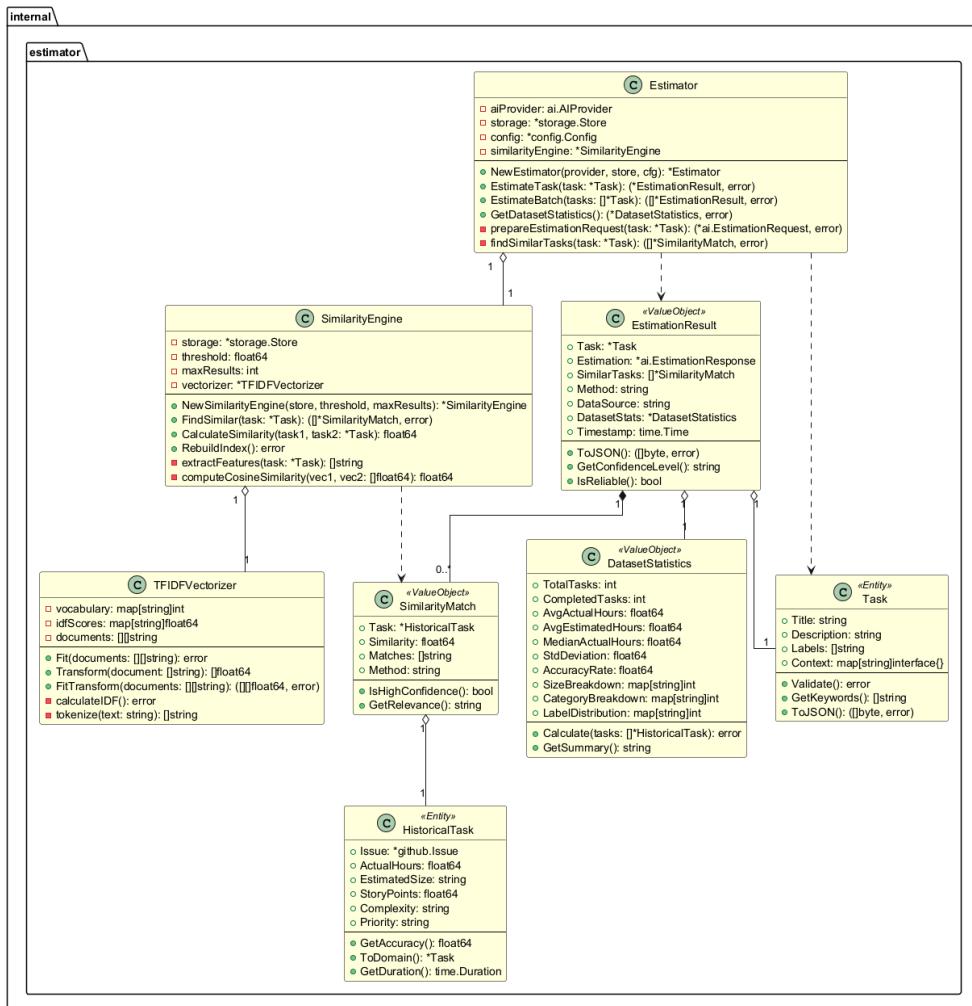


Figura 3.2 - Diagrama de Classes: Pacote Estimator

Os componentes principais desse diagrama são:

- **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).

O pacote internal/github , representado na Figura 3.3, 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 permite sincronização de custom fields (story points, horas estimadas) que enriquecem o dataset histórico e melhoram a precisão das estimativas. A Figura 3.1.3 ilustra o diagrama de classes que corresponde ao pacote GitHub.

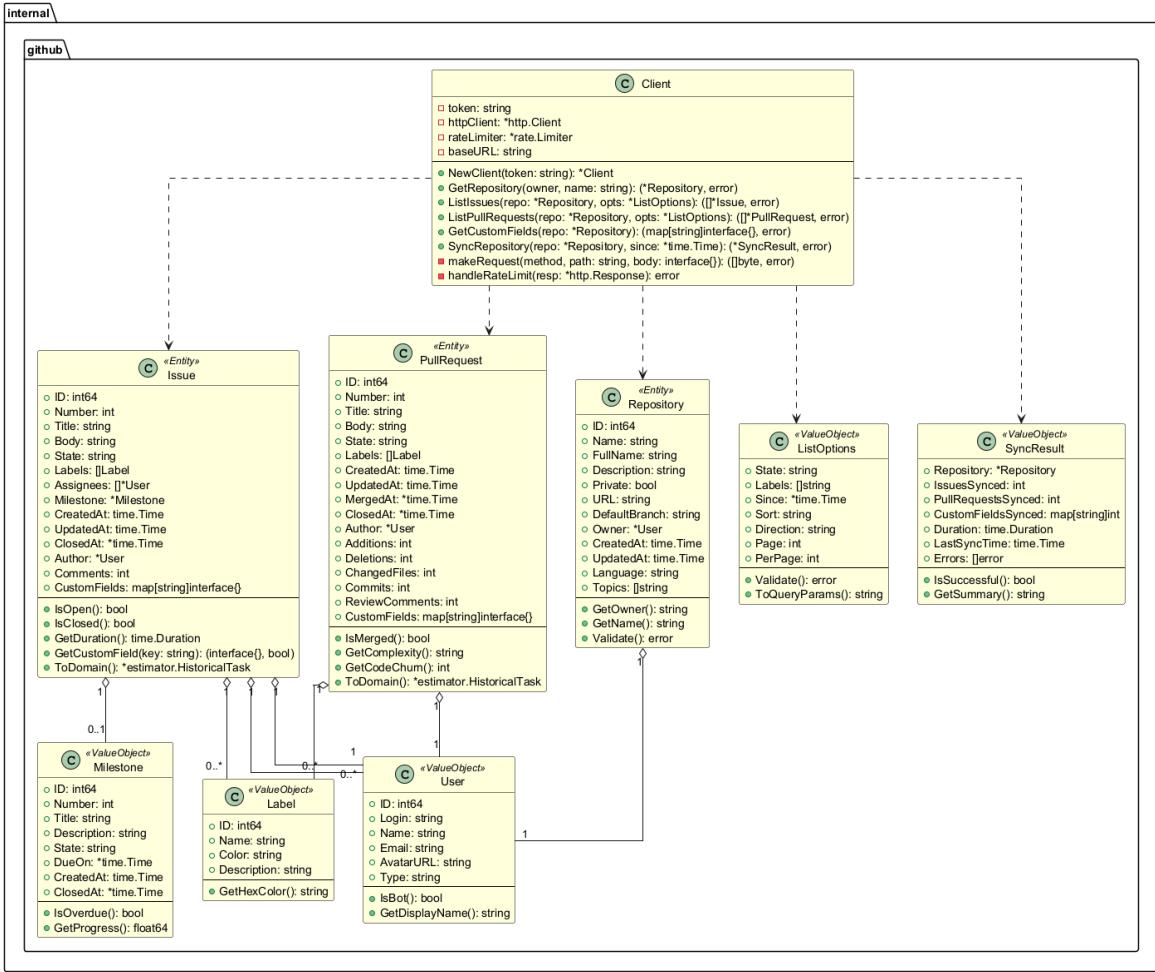


Figura 3.3 - Diagrama de Classes: Pacote GitHub

Os componentes principais desse diagrama são:

- **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. 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).

O pacote internal/storage, representado na Figura 3.4, 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. A Figura 3.1.4 ilustra o diagrama de classes que corresponde ao pacote storage.

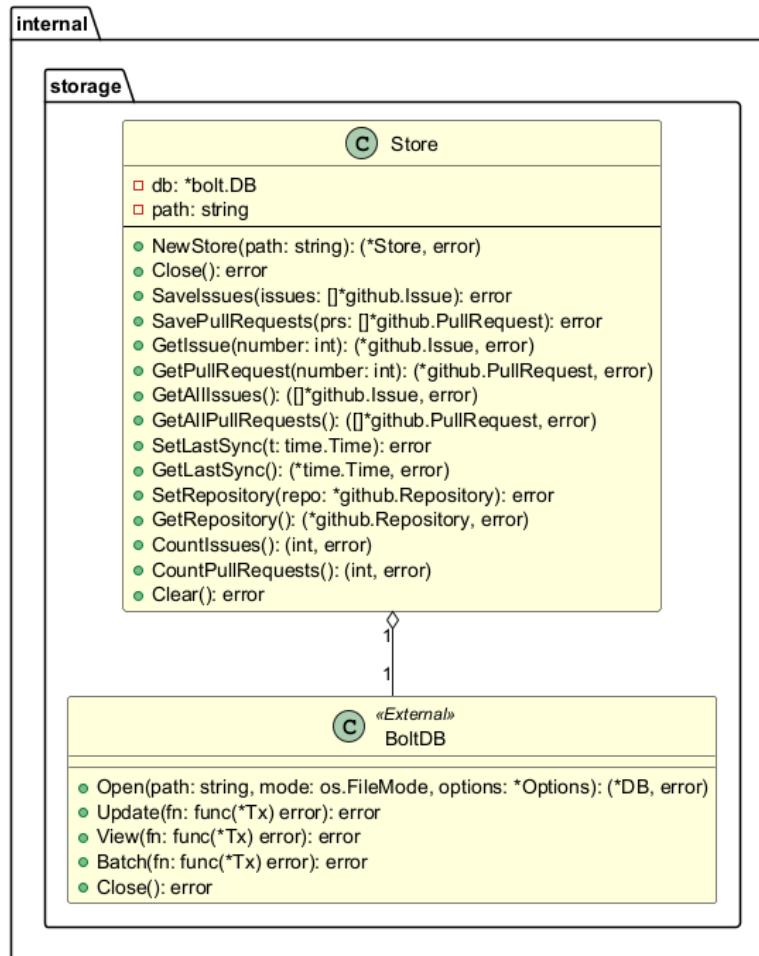


Figura 3.4 - Diagrama de Classes: Pacote Storage

Os componentes principais desse diagrama são:

- 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).
- 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

O pacote internal/config, representado na Figura 3.5, 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). A Figura 3.1.5 ilustra o diagrama de classes que corresponde ao pacote config.

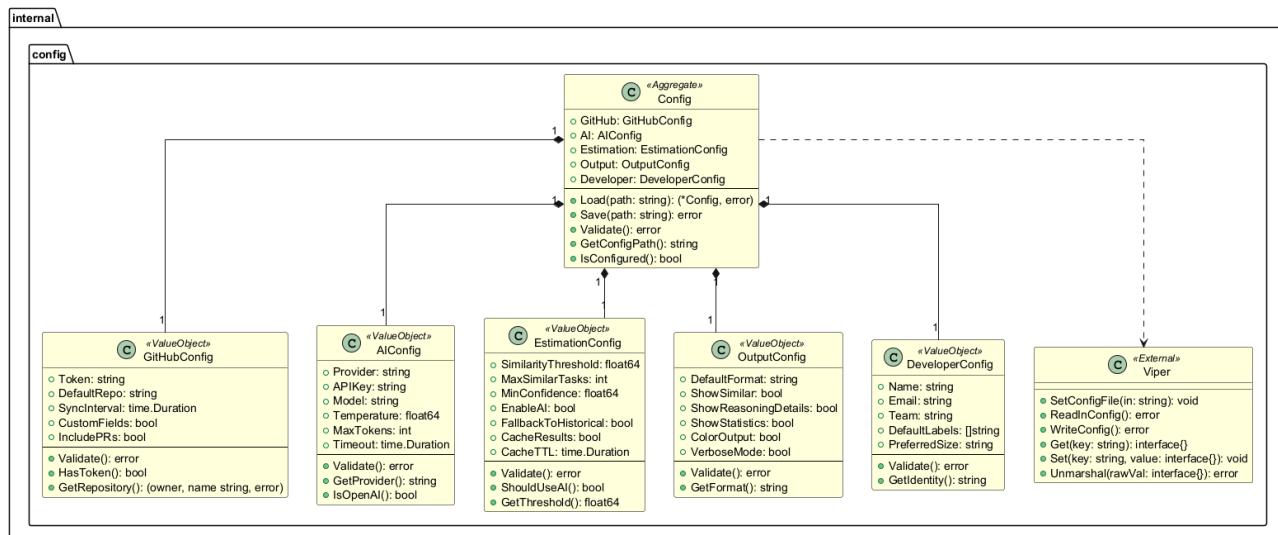


Figura 3.5 - Diagrama de Classes: Pacote Config

Os componentes principais desse diagrama são:

- **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

3.1.1 Padrões de Design Aplicados

A modelagem do sistema aplica diversos padrões consolidados:

Foi utilizado o padrão Domain-Driven Design (DDD), o qual foi subdividido em:

- **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)

Foi utilizado o padrão Clean Architecture, conforme abaixo:

- 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 (AIPublisher, Repository implícito)

Em adição, também utilizou-se o Padrão GoF da seguinte maneira:

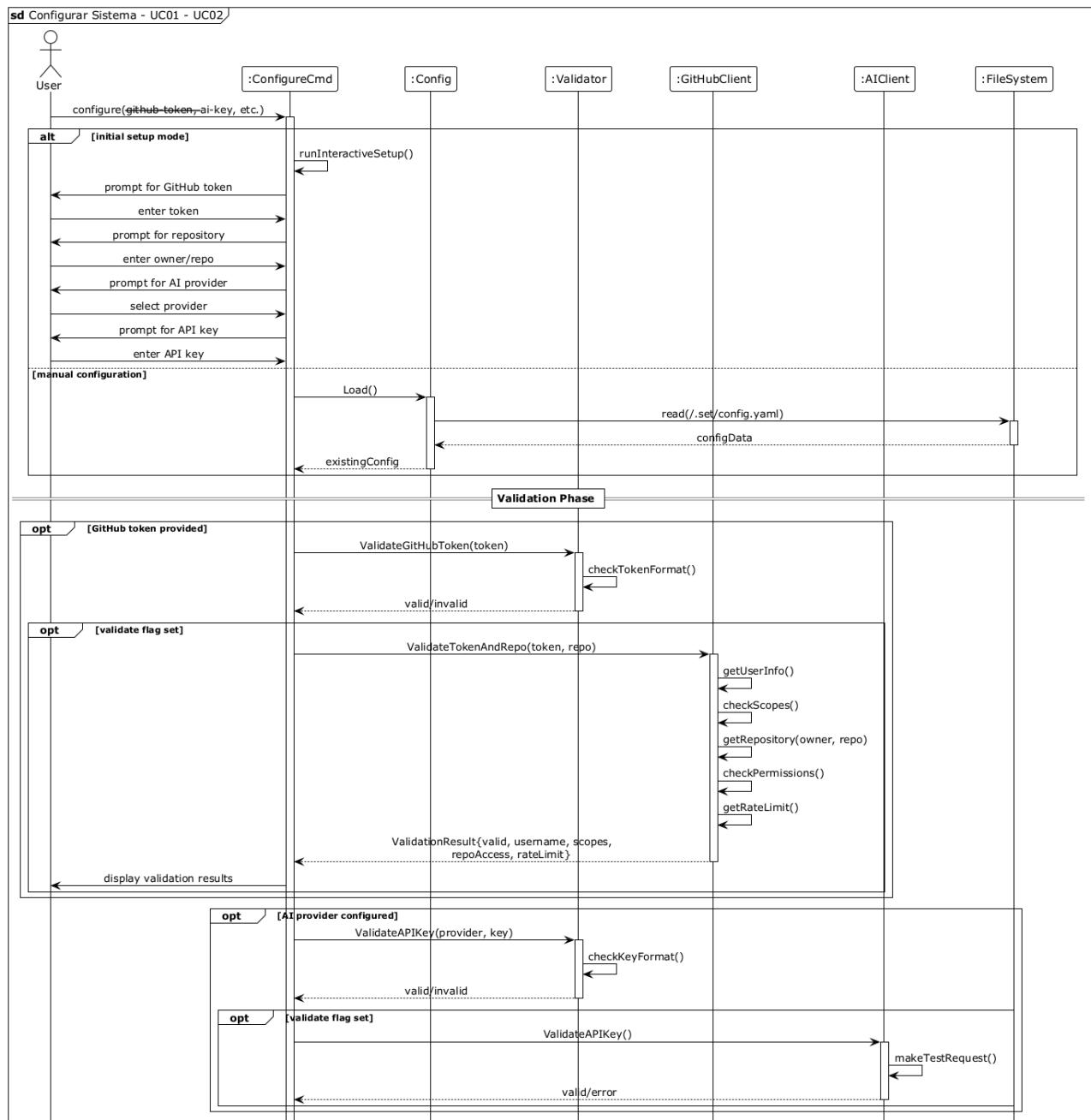
Strategy: AIPublisher permite trocar algoritmo de estimativa

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.



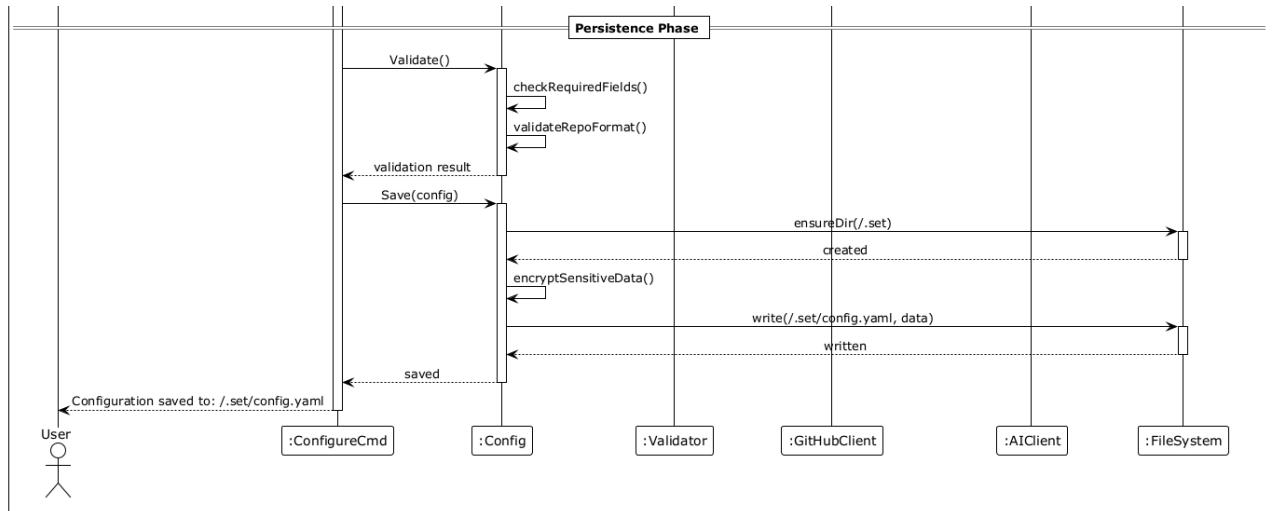


Figura 3.6 - Diagrama de Sequência: Configurar Sistema

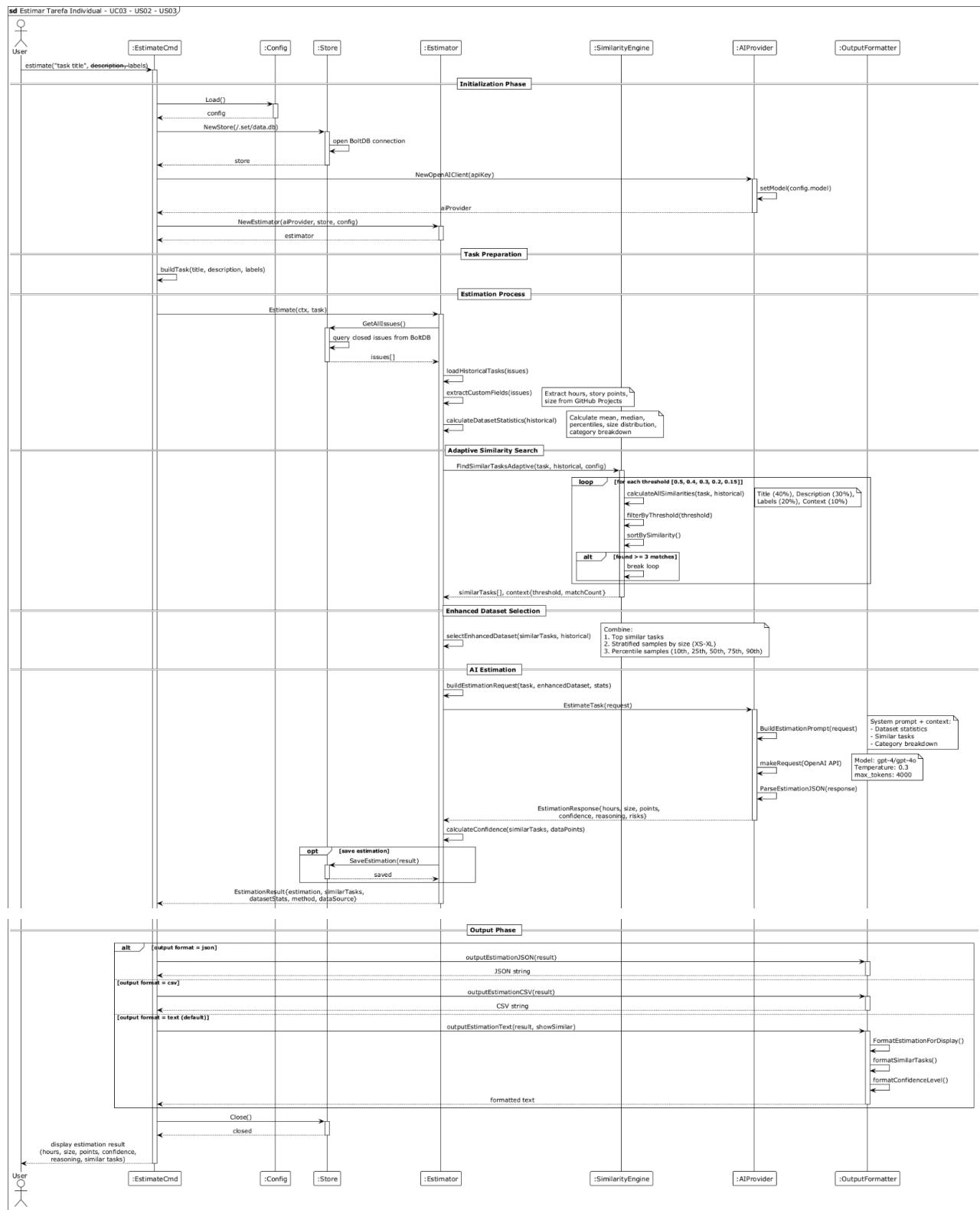
O diagrama apresentado na Figura 3.6 ilustra o processo integral de configuração do sistema SET CLI, que engloba a validação de credenciais e a persistência de dados. Ele implementa os casos de uso UC01 - Configurar Token GitHub e UC02 - Configurar Sistema, atendendo às histórias de usuário US01 (Desenvolvedor), US08 (Scrum Master) e US14 (Product Owner). O fluxo é estruturado em três estágios principais, começando pela Fase de Inicialização.

A Fase de Inicialização permite a configuração de modo interativo (com a flag `--initial`) ou manual, carregando as configurações do arquivo `~/.set/config.yaml` com a biblioteca Viper.

Em seguida, na Fase de Validação, são realizadas verificações de integridade. A validação do Token GitHub confere o formato e, opcionalmente, o estado online do token. Se a flag `--validate` for usada, ocorre uma Validação Completa de Repositório, que verifica escopos, permissões (como acesso a issues e pull requests) e o rate limit da API. Por fim, a Validação do Provedor de IA checa o formato da API key e testa a conexão com o provedor de IA configurado (OpenAI).

Concluídas as verificações, a Fase de Persistência se inicia, garantindo a consistência dos campos obrigatórios e a existência do diretório `~/.set`. Para segurança, dados sensíveis são criptografados antes de serem salvos, e a configuração final é persistida no arquivo `~/.set/config.yaml` em formato YAML.

Este fluxo segue rigorosos padrões de design, como o Validation Pattern e o princípio Fail-Fast, além da Separation of Concerns com validadores dedicados. A segurança é reforçada pela Encryption at Rest (Criptografia em Repouso) para proteger dados armazenados.

**Figura 3.7 - Diagrama de Sequência: Estimar Tarefa Individual**

O algoritmo central de estimativa do sistema, detalhado na Figura 3.7, implementa o caso de uso UC03 - Estimar Tarefa Individual, atendendo às histórias US02 (estimativa via CLI), US03 (visualização de tarefas similares), US05 (score de confiança) e US06 (uso de labels para categorização). Ele opera através de um processo estruturado em sete fases para garantir a precisão das estimativas.

O processo de estimativa de projeto compreende as seguintes etapas. A primeira é a Inicialização, que consiste na configuração do ambiente (Viper), no estabelecimento de conexão com o BoltDB e na inicialização das interfaces do cliente OpenAI (ex: GPT-4) e do módulo Estimator. Em seguida, ocorre a Preparação da Tarefa, com a criação e validação do objeto Task, a partir dos dados fornecidos pelo usuário, como título, descrição, rótulos (*labels*) e contexto.

A terceira etapa é o Processamento da Estimativa, que envolve o carregamento e a filtragem de dados históricos (registros de *issues* encerradas do BoltDB e campos customizados do GitHub). Posteriormente, são calculadas métricas estatísticas do conjunto de dados (*dataset*), incluindo média, mediana, percentis e detalhamento por categoria (*breakdown*).

O quarto passo é a Busca Adaptativa por Similaridade. Esta etapa emprega um algoritmo dinâmico que itera sobre diferentes limiares (*thresholds*) (de 50% a 15%) até identificar um mínimo de três correspondências (*matches*). O Cálculo de Similaridade Multifatorial atribui pesos ponderados: 40% para o Título (Jaccard), 30% para a Descrição (Jaccard), 20% para Rótulos (*Labels*) e 10% para o Contexto.

Na Seleção de Dataset Otimizado, que é a quinta etapa, combinam-se as Top Tarefas Semelhantes (*Top Similar Tasks*), Amostragem Estratificada (por dimensão: XS-XL) e Amostragem por Percentil, garantindo um mínimo de dez tarefas para assegurar elevada confiabilidade.

A sexta etapa é a Estimativa Baseada em Inteligência Artificial. Nesta fase, ocorre a construção da requisição com os dados da tarefa e os *datasets* selecionados, a geração do *prompt* (definição da função e contexto histórico/similar) e o acionamento da API OpenAI (modelo configurável, temperatura 0.3, json_object). A análise (*Parsing*) extrai a *EstimationResponse* (horas, *story points*, *ConfidenceScore* e *Reasoning*). O Nível de Confiança é categorizado como Elevado ($\geq 70\%$), Médio (50-69%) ou Baixo ($< 50\%$), com base na qualidade e na quantidade das correspondências (*matches*).

Por fim, a etapa de Saída compreende a geração dos resultados nos formatos Text, JSON e CSV. O design arquitetural incorpora os padrões *Adaptive Algorithm*, *Statistical Sampling*, *Strategy Pattern* (para formatação) e *Template Method* (para o fluxo de trabalho).

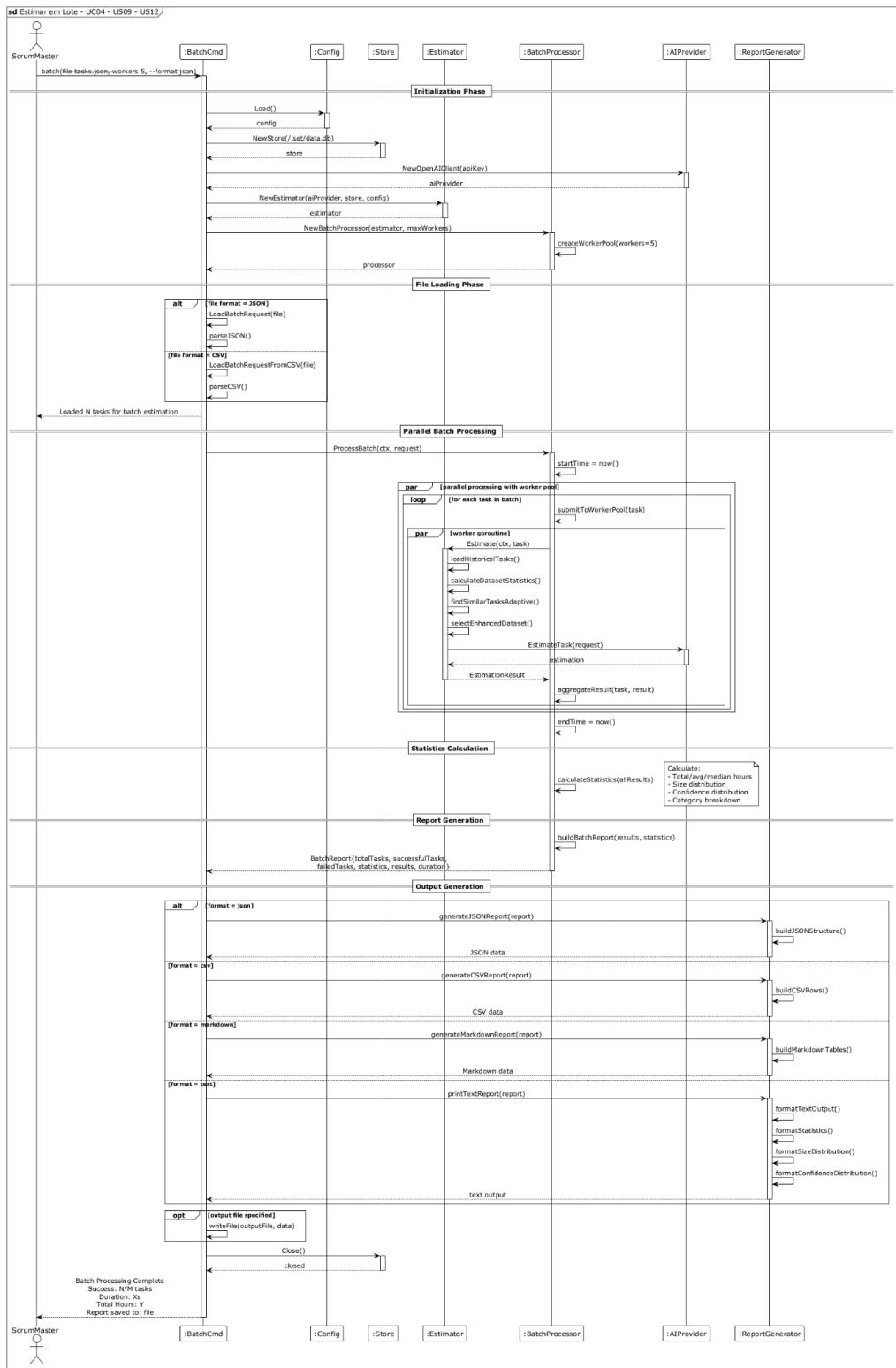


Figura 3.8 - Diagrama de Sequência: Estimar em Lote

O processamento paralelo de múltiplas estimativas, representado na figura 3.8, implementa o caso de uso UC04 - Estimar Lote de Tarefas, atendendo às histórias US09 (preparação de Planning pelo Scrum Master), US12 (processamento paralelo), US16 (estimativa de épicos pelo Product Owner) e US18 (estatísticas agregadas). Ele é estruturado em seis fases sequenciais.

A Fase de Inicialização estabelece o ambiente, carregando configurações essenciais e instanciando componentes primários como Store, AIProvider e Estimator. Nesta fase, é criado um BatchProcessor que emprega um pool de *workers* configurável, implementado por meio de *goroutines* da linguagem Go para garantir um paralelismo efetivo.

A Fase de Carregamento de Arquivo é responsável por consumir os dados de entrada, suportando dois formatos principais: JSON, que segue a estrutura BatchRequest com metadados e tarefas, e CSV, um formato simplificado de colunas predefinidas. Um *parser* específico é utilizado para cada formato, garantindo-se a validação de *schema* antes do início do processamento.

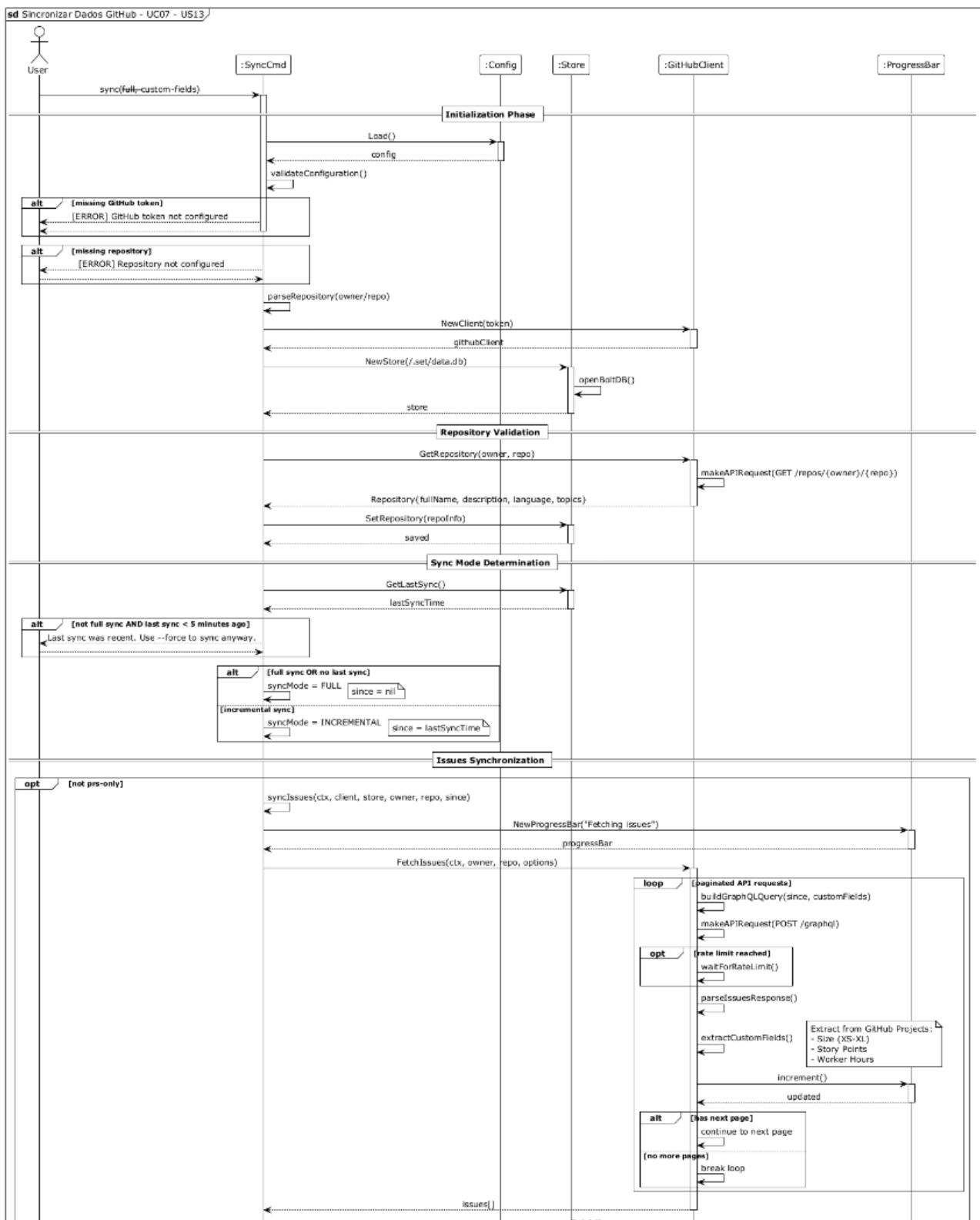
No cerne do sistema, a Fase de Processamento Paralelo em Lote adota o padrão *Worker Pool* para execução concorrente. Um canal de tarefas é criado para distribuir as unidades de trabalho entre N *goroutines workers* independentes. Cada *worker* executa o algoritmo completo de estimativa, o qual inclui carregamento de histórico, cálculo de estatísticas, busca adaptativa por similaridade, seleção de *dataset* aprimorado e, por fim, a estimativa via Inteligência Artificial. Os resultados são subsequentemente agregados de forma *thread-safe*, contabilizando-se sucessos e falhas e registrando o tempo de processamento de cada estimativa.

Em seguida, a Fase de Cálculo de Estatísticas gera métricas agregadas, como o total, a média, a mediana e o intervalo (min-max) das horas estimadas, além da confiança média. Esta fase também calcula distribuições detalhadas: por tamanho (XS-XL), por nível de confiança (High/Medium/Low) e por categoria, baseada nos rótulos das tarefas.

A Fase de Geração de Relatório constrói o BatchReport, que consolida os metadados do processamento (total, sucesso, falha e duração das tarefas), as estatísticas agregadas e os resultados individuais de cada tarefa.

Finalmente, a Fase de Geração de Saída permite a apresentação do relatório em quatro formatos distintos: Text (formatado para terminal), JSON (estrutura completa para integração), CSV (tabela de resultados por tarefa) e Markdown (para documentação). O resultado pode ser salvo em arquivo ou exibido no *stdout*.

O projeto incorpora diversos Elementos de Design importantes, como o *Worker Pool Pattern* e *Producer-Consumer* para gerenciamento de concorrência, *Map-Reduce* para o processamento e agregação, o *Strategy Pattern* para a troca de geradores de relatório, e o *Observer Pattern* na coleta de resultados. Otimizações de Performance são incluídas, como o paralelismo configurável via *flag --workers*, o processamento assíncrono, a utilização de operações em lote no BoltDB para otimização de I/O, o reuso de conexão HTTP com a OpenAI API e a inclusão de um indicador de progresso opcional.



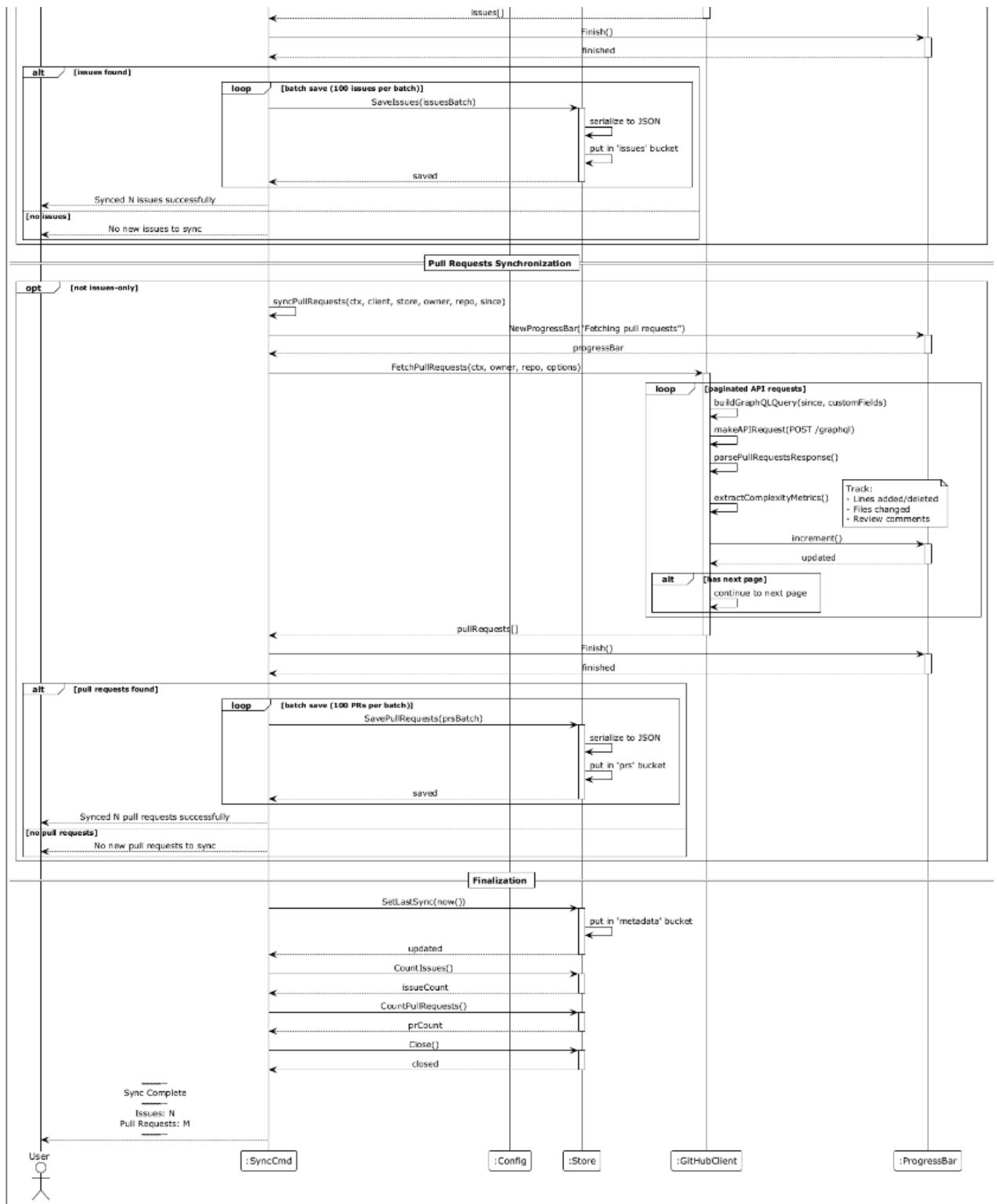


Figura 3.9 - Diagrama de Sequência: Sincronizar Dados GitHub

O diagrama ilustrado na figura 3.9 descreve o processo de sincronização incremental de dados do GitHub para um armazenamento local, projetado para otimizar a eficiência e garantir a resiliência.

Ele implementa o caso de uso UC07 - Sincronizar com GitHub, atendendo especialmente à história US13 (sincronização de custom fields). Este processo está estruturado em seis fases distintas.

A Fase de Inicialização é responsável por carregar a configuração, validar as credenciais necessárias, e verificar a presença do *GitHub token* e do repositório configurado. Em caso de configuração inválida, o processo é encerrado prematuramente com uma mensagem de erro. Após a validação, o *GitHubClient* é inicializado e o banco de dados BoltDB é aberto ou criado no caminho `~/.set/data.db`.

Na sequência, a Fase de Validação de Repositório utiliza a *GitHub REST API* para buscar informações do repositório. O objetivo é validar a existência do repositório e confirmar o acesso por meio do *token* fornecido. Metadados importantes, como nome completo, descrição, linguagem e tópicos, são armazenados para fins de rastreamento.

A Fase de Determinação do Modo de Sincronização aplica o *Sync Throttling*, verificando o *timestamp* da última sincronização. Se a última sincronização ocorreu há menos de 5 minutos e a opção `--force` não foi utilizada, o processo é interrompido para evitar requisições desnecessárias, respeitando os limites de taxa (*rate limits*) da API do GitHub. O modo de sincronização é então selecionado: a Sincronização Completa (*Full Sync*), ativada pela flag `--full` ou na primeira execução, busca todos os dados. A Sincronização Incremental (*Incremental Sync*), que é o padrão, utiliza o *lastSyncTime* para buscar apenas os itens que foram atualizados.

A Fase de Sincronização de Issues inicia com a criação de uma barra de progresso visual para *feedback* ao usuário. Realizam-se chamadas paginadas à *GitHub GraphQL API*, utilizando *cursors* para navegar em grandes volumes de dados. Mecanismos de resiliência incluem a detecção e espera de *reset* quando o *rate limit* é atingido, e a implementação de lógica de repetição com *backoff* exponencial para lidar com erros temporários. Esta fase é crucial para a extração de campos customizados do *GitHub Projects*, como *Size*, *Story Points* e *Worker Hours*, essenciais para a qualidade das estimativas. A persistência dos dados é otimizada por meio de *Batch Persistence*, onde *issues* são serializadas em JSON e salvas em lotes de 100 no *bucket* 'issues' do BoltDB, utilizando transações para garantir a atomicidade das operações.

Em seguida, a Fase de Sincronização de Pull Requests segue um processo similar ao das *issues*, mas com a adição da coleta de métricas de complexidade, como linhas adicionadas/deletadas, número de arquivos alterados e contagem de comentários de revisão. Essas métricas são valiosas para correlacionar a complexidade do código com o tempo de desenvolvimento, e os dados são salvos em um *bucket* separado, 'prs', no BoltDB.

Por fim, a Fase de Finalização atualiza o *timestamp* da última sincronização no *bucket* 'metadata'. São obtidos os contadores atualizados de *issues* e *Pull Requests*. A conexão com o BoltDB é fechada e um resumo formatado, exibindo os totais sincronizados, é apresentado ao usuário.

O *design* do sistema incorpora elementos fundamentais para sua robustez, como o *Incremental Sync Pattern*, que minimiza a carga de rede e API; o *Retry with Exponential Backoff*, que confere

resiliência; e a conformidade com o *Rate Limiting* do GitHub. Otimizações de performance incluem o uso de GraphQL para reduzir o *payload*, índices B+tree no BoltDB para *queries* rápidas, serialização JSON eficiente e operações em lote para minimizar chamadas ao sistema (*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. A Figura 3.10 ilustra a colaboração estrutural durante o processo de estimativa de uma tarefa. Mostra como múltiplos componentes trabalham em conjunto para gerar estimativas precisas baseadas em IA e dados históricos.

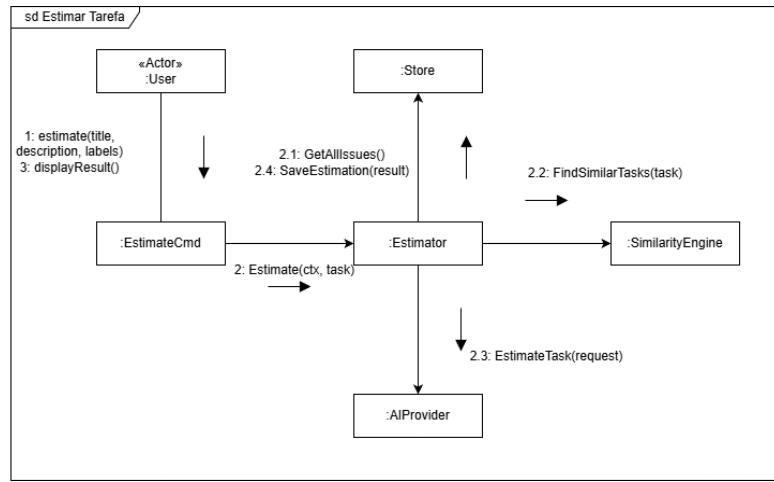


Figura 3.10 - Diagrama de Comunicação: Estimando uma tarefa

A figura 3.11 mostra as interações de comunicação durante a configuração do sistema SET CLI. Representa como os componentes colaboram para validar credenciais (GitHub token, AI API key) e persistir configurações.

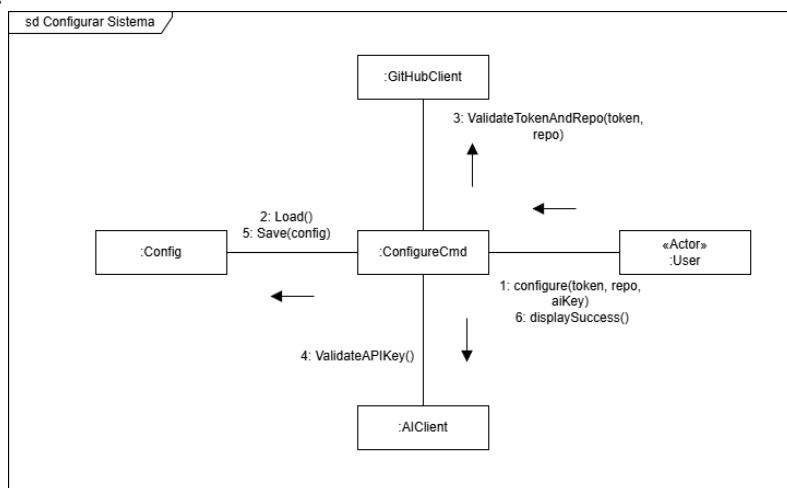


Figura 3.11 - Diagrama de Comunicação: Configurar sistema

O diagrama da Figura 3.12 mostra a comunicação de processamento paralelo para estimativas em lote. Destaca como o sistema utiliza workers concorrentes para processar múltiplas tarefas simultaneamente.

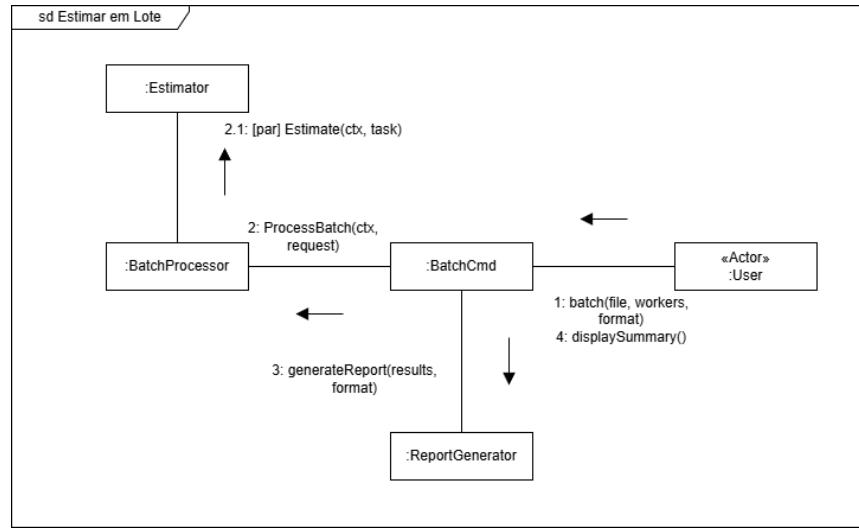


Figura 3.12 - Diagrama de Comunicação: Estimar em Lote

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 e sempre seguindo boas práticas de desenvolvimento de Software. 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)

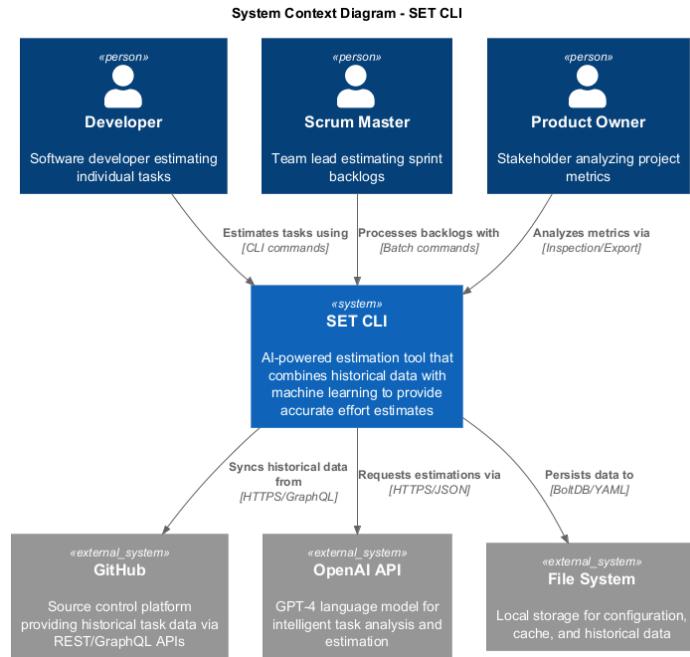


Figura 3.13 - Diagrama de Contexto C4

O diagrama de contexto da Figura 3.13 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)

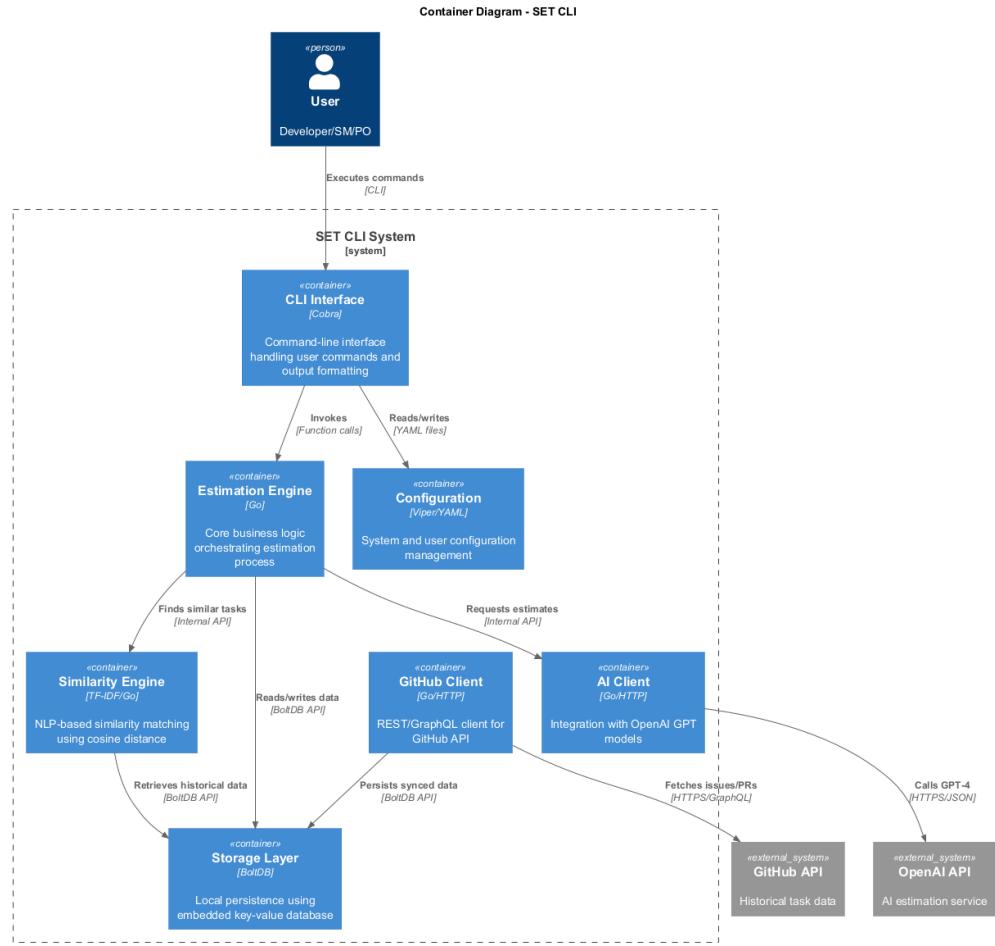


Figura 3.14 - Diagrama de Contêineres C4

O diagrama de contêineres da Figura 3.14 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 Tecnologias e Frameworks

O projeto será desenvolvido utilizando a linguagem de programação Go, na versão 1.21 ou superior, devido às suas vantagens em termos de performance, concorrência nativa (goroutines) e facilidade de distribuição (geração de um único binário). Para otimizar o desenvolvimento, serão empregados frameworks específicos como Cobra para a interface de linha de comando (CLI), Viper para a gestão eficiente de configurações e BoltDB para persistência de dados.

A estratégia de persistência local foca em soluções de alta performance e portabilidade. O BoltDB será utilizado como um banco de dados embarcado, ideal para o armazenamento de dados históricos.

e estimativas. Além disso, arquivos nos formatos JSON e YAML serão empregados para a gestão de configurações e cache estruturado, aproveitando a natureza *zero-configuração* e a performance local que estas abordagens oferecem.

No que tange à integração externa, o projeto fará uso da GitHub API v4, utilizando tanto GraphQL quanto REST, para a coleta de dados detalhados de repositórios. Adicionalmente, a OpenAI API será integrada para viabilizar funcionalidades avançadas de análise de linguagem natural e para a geração de estimativas. É fundamental destacar que será implementada uma política de Rate Limiting rigorosa para assegurar o respeito aos limites de uso impostos por todas as APIs externas.

3.4.4 Fluxo de Dados e Processamento

O projeto é estruturado em torno de três fluxos de processamento distintos:

O processo é iniciado quando a Interface de Linha de Comando (CLI) recebe o comando e os parâmetros fornecidos pelo usuário. Em seguida, o Estimation Service é responsável por processar a descrição detalhada da tarefa. O Similarity Engine entra em ação para buscar tarefas similares que já foram registradas no histórico local, fornecendo um contexto para a análise. Concomitantemente, o AI Client envia o contexto da tarefa para a análise da Inteligência Artificial. Por fim, o resultado consolidado da estimativa é retornado e formatado de maneira adequada para a exibição ao usuário.

A funcionalidade começa com a CLI carregando o arquivo contendo a lista de tarefas, que pode estar nos formatos JSON ou CSV. O Core Engine então processa cada uma das tarefas de forma eficiente, fazendo uso de goroutines para garantir a paralelização e otimizar o tempo de processamento. Após a conclusão do processamento, os resultados são agregados e as estatísticas relevantes são calculadas. A etapa final é executada pelo Export Engine, que gera os arquivos de saída nos formatos especificados e solicitados.

O processo é conduzido pelo GitHub Client, que realiza a busca por dados incrementais do repositório. O Cache Manager atua para otimizar as requisições repetitivas, melhorando a performance geral. Posteriormente, o Repository persiste com os novos dados coletados no banco de dados local BoltDB. Concluindo a sincronização, o Similarity Engine reindexa os dados recém-adicionados, preparando-os para futuras e rápidas buscas de similaridade.

3.4.5 Estratégias de Performance

A arquitetura do projeto incorpora diversas estratégias para garantir alta performance, escalabilidade e eficiência no processamento de dados e requisições.

Para tal, aproveitamos a natureza eficiente do Go, implementando concorrência robusta. Utilizamos *goroutines* para habilitar o processamento paralelo de estimativas em lote e gerenciar múltiplas requisições de API simultaneamente, maximizando a utilização dos recursos do sistema.

Além disso, o sistema adota uma abordagem de múltiplas camadas para caching, otimizando o

acesso a dados:

- Implementamos um cache em memória para armazenar dados frequentemente acessados, garantindo tempos de resposta ultrarrápidos.
- Utilizamos cache de arquivos para persistir e reutilizar resultados de APIs externas, reduzindo a latência e o custo de chamadas repetidas.
- O *Time-to-Live* (TTL) de ambos os caches é configurável, permitindo um ajuste fino entre a *freshness* dos dados e o desempenho do sistema.

Finalmente, para acelerar as operações de busca e recuperação de dados, o BoltDB é utilizado com índices otimizados. Esses índices são cruciais para a busca rápida de tarefas similares, utilizando *keywords* e metadados como critérios de pesquisa.

3.4.6 Segurança e Configuração

Para uma maior segurança a autenticação do sistema é realizada através de Tokens do GitHub e da OpenAI. Esses tokens são armazenados localmente e possuem as permissões adequadas para o funcionamento do projeto.

A configuração do sistema segue um modelo hierárquico bem definido, garantindo flexibilidade e facilidade de override. A ordem de precedência, da menor para a maior, é: valores *default* (padrões do sistema), seguido pelo conteúdo do arquivo de configuração (*config file*), depois pelas variáveis de ambiente e, por fim, pelos parâmetros passados via linha de comando (CLI), que possuem a precedência máxima.

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.

A Figura 3.15 ilustra através do diagrama os estados pelos quais uma solicitação de estimativa passa, desde a submissão até a conclusão ou erro.

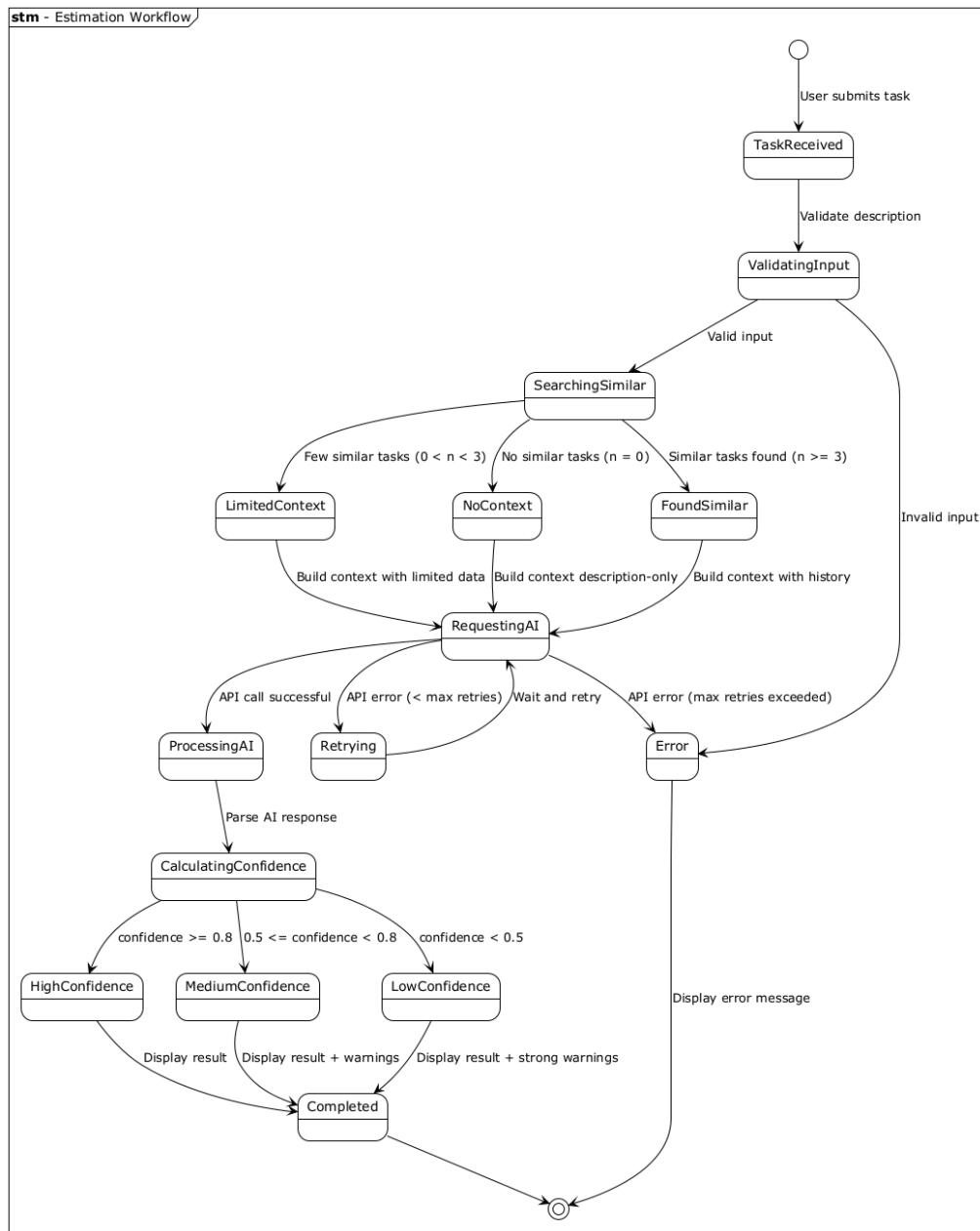
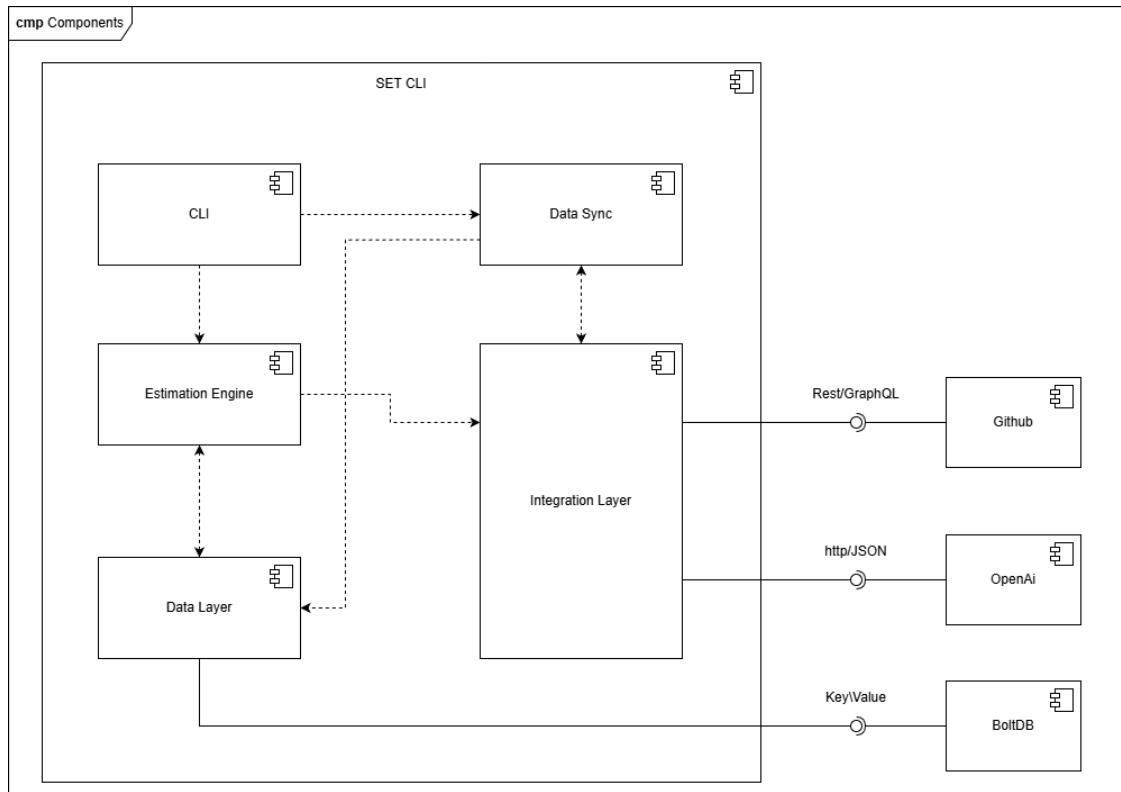
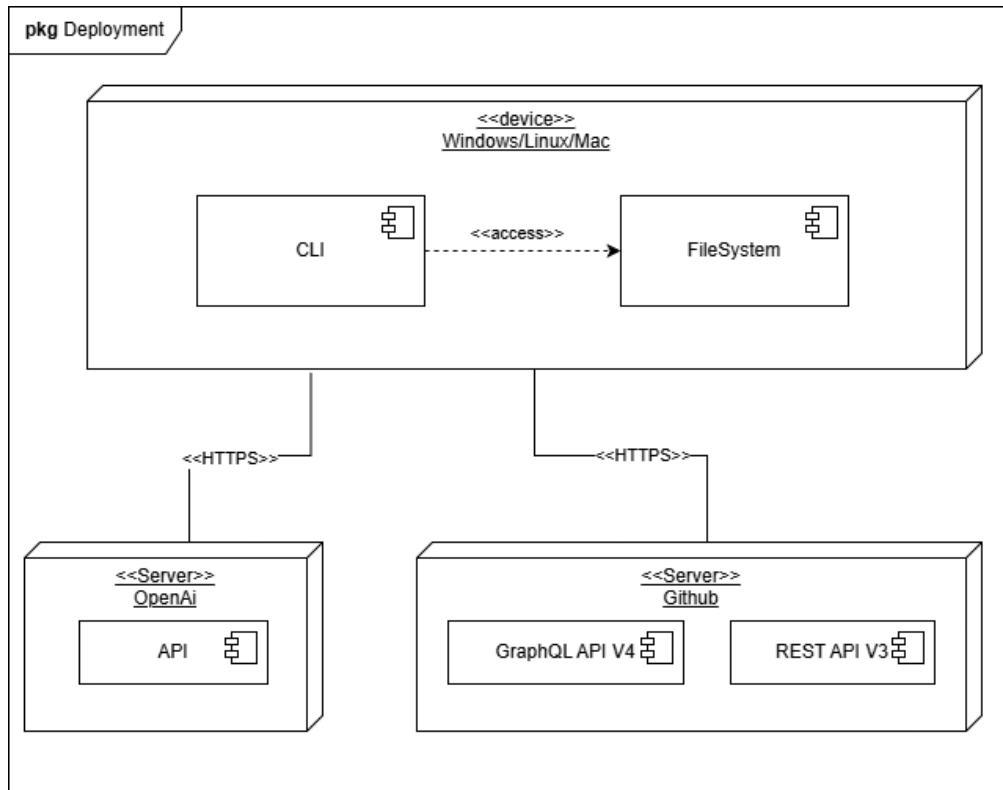


Figura 3.15 - Diagrama de Estados: Workflow de Estimativa

3.6 Diagrama de Componentes e Implantação.

A Figura 3.16 mostra o diagrama de componentes, o qual ilustra a organização modular do sistema em componentes de alto nível, suas interfaces fornecidas e requeridas, e as dependências entre eles.

**Figura 3.16 - Diagrama de Componentes****Figura 3.17 - Diagrama de Implantação da aplicação**

O diagrama de implantação (Figura 3.17) mostra a arquitetura física do sistema, incluindo os nós de hardware, artefatos de software e seus protocolos de comunicação.

O ambiente de implantação do projeto é composto pelos seguintes nós, cada um com uma função específica na execução e processamento das tarefas: o Device Windows/Linux/Mac, que representa a máquina cliente onde o binário *set* é executado localmente; os GitHub Servers, responsáveis por hospedar os repositórios do código-fonte e fornecer acesso através de APIs; e os OpenAI Servers, dedicados ao processamento das requisições que envolvem inteligência artificial.

A comunicação entre os diferentes nós e o acesso a recursos externos são realizados através dos seguintes protocolos e mecanismos de autenticação: HTTPS com autenticação por token (GitHub), HTTPS com autenticação por chave de API (OpenAI) e acesso local ao filesystem para utilização do 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.

A Figura 4.1 ilustra a saída do comando *set help*, que exibe a lista completa de comandos disponíveis no sistema, suas descrições e flags globais. Esta interface fornece ao usuário uma visão geral rápida de todas as funcionalidades disponíveis e serve como ponto de entrada para exploração do sistema.

The screenshot shows the command-line interface for the SET tool. At the top, it displays the title "SET CLI" and the subtitle "Software Estimation Tool AI-Powered Effort Estimation for Agile Teams". Below this, there is a brief description: "A powerful CLI tool that combines AI intelligence with historical data to provide accurate software development effort estimates." The main content is organized into sections: "Core Features" and "Perfect For".

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

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-tcc1-0393100-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*

A Figura 4.2 apresenta a interface do comando *set configure --initial*, que realiza a configuração inicial do sistema. Este comando solicita ao usuário o token de acesso pessoal do GitHub e OpenAI. Também permite configurar o repositório padrão para sincronização de dados históricos e outros valores de preferência. A interface guia o usuário através de um fluxo interativo com validação em tempo real e feedback imediato sobre o sucesso de cada etapa.

```
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/claudie) [openai]:
Enter your OPENAI API key: [REDACTED]

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

A Figura 4.3 apresenta a interface do comando `set configure --initial`, que verifica a configuração inicial do sistema. Garantindo que a configuração do usuário está correta. A interface mostra ao usuário detalhes de sua configuração de forma clara e objetiva.

```
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 4.3 - Interface ao executar o comando de validar configuração: `set configure --validate`

A Figura 4.4 demonstra a execução do comando `set estimate` para estimativa de uma tarefa individual. O comando analisa a descrição da tarefa, busca tarefas similares no histórico utilizando algoritmos de similaridade textual, consulta a API de IA configurada, e apresenta uma estimativa detalhada incluindo horas de esforço, story points, nível de confiança e raciocínio fundamentado. A interface exibe também as tarefas similares encontradas com seus respectivos esforços reais, permitindo ao usuário validar a relevância das comparações.

Task
Implement OAuth login with Google

Estimation

Time	16.0h
Size	L
Points	8
Conf	70% 

Analysis

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.

Assumptions

- 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

Risks

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

Recommendation

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.

ai · openai

Next Steps

- Review the assumptions and risks above
- Consider adding custom fields to GitHub Projects

Figura 4.4 - Interface ao executar o comando de estimar: *set estimate "Implement OAuth login with Google"*

A Figura 4.5 apresenta a interface do comando *set batch --file*, que processa múltiplas tarefas simultaneamente a partir de um arquivo de entrada. O comando utiliza processamento paralelo com worker pools para maximizar eficiência, exibe progresso em tempo real, e gera um relatório consolidado com estatísticas agregadas incluindo distribuição de tamanhos, níveis de confiança e tarefas que requerem revisão adicional. A saída inclui múltiplos formatos (CSV, PDF, JSON) para facilitar integração com outras ferramentas de planejamento.

```

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  Confidence
---  ---  ---  ---  ---  ---
Implementar autenticação ...  40.0  L    13    80%
Corrigir bug no formulário...  8.0   M    5     70%
Adicionar paginação na li...  16.0  L    13    75%
Escrever testes unitários ...  16.0  L    8     70%
Refatorar componente de car...  16.0  L    13    70%

```

Figura 4.5 - Interface ao executar o comando de estimar em batch: `set batch --file examples\sprint-backlog.json`

A Figura 4.6 exemplifica o uso do comando `set inspect --list --custom`, que permite visualizar e explorar os dados históricos armazenados localmente no BoltDB. O comando apresenta uma tabela formatada com as issues sincronizadas, incluindo número, título e labels. Esta funcionalidade é útil para validar a qualidade dos dados históricos e identificar tarefas específicas que podem servir como referência para novas estimativas.

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 SOMS 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-Standard 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 November...	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 4.6 - Interface ao executar o comando de listar dados: *set inspect --list --custom*

A Figura 4.7 mostra a execução do comando *set sync --custom-fields*, responsável por sincronizar dados históricos do repositório GitHub configurado. O comando utiliza tanto a REST API quanto a GraphQL API do GitHub para coletar issues, pull requests e custom fields definidos em GitHub Projects. A interface apresenta estatísticas sobre os dados coletados, incluindo quantidades de issues e Pull Requests sincronizados.

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

Figura 4.7 - Interface ao executar o comando de sincronizar: *set sync --custom-fields*

5. Glossário e Modelos de Dados

Esta seção se dedica à descrição dos modelos de dados da aplicação e à definição de um glossário para a interpretação dos conceitos específicos deste projeto. A compreensão destes termos e estruturas de dados é fundamental para o entendimento da arquitetura e funcionamento do sistema SET CLI.

O glossário apresentado na Tabela 5.1 reúne os termos técnicos utilizados ao longo da documentação e no desenvolvimento do sistema. Estes termos abrangem conceitos de metodologias ágeis, tecnologias específicas utilizadas na implementação, métricas de estimativa de software e componentes de infraestrutura. A familiaridade com esta terminologia é essencial para compreender as decisões de design do sistema e os algoritmos de estimativa implementados.

Termo	Definição
CLI	Command Line Interface - Interface de linha de comando
IA	Inteligência Artificial
API	Application Programming Interface - Interface de Programação de Aplicações
Scrum	Framework ágil para desenvolvimento de software
GitHub	Plataforma de hospedagem de código e versionamento
Sprint	Período fixo de desenvolvimento no Scrum (geralmente 1-4 semanas)
Story Points	Unidade de medida para estimar o esforço relativo de tarefas
Velocity	Quantidade de story points completados por sprint
Planning Poker	Técnica de estimativa colaborativa usada em metodologias ágeis
Accuracy	Precisão das estimativas (quão próximas estão do tempo real)
Confidence Score	Nível de confiança da estimativa (0.0 a 1.0)
TF-IDF	Term Frequency-Inverse Document Frequency - algoritmo de processamento de texto

Cosine Similarity	Medida de similaridade entre vetores de texto
BoltDB	Banco de dados embarcado key-value em Go
GraphQL	Linguagem de consulta para APIs
Token	Chave de autenticação para APIs
Rate Limit	Limite de requisições por período de tempo

Tabela 5.1 - Glossário de Termos

Na tabela 5.2 a entidade Task representa a estrutura de dados fundamental do sistema, armazenando informações sobre tarefas de desenvolvimento de software. Esta entidade captura tanto dados descritivos da tarefa quanto métricas de estimativa e execução. O identificador único permite rastreamento individual de cada tarefa, enquanto os campos de título e descrição fornecem o contexto necessário para os algoritmos de similaridade textual. Os atributos de tempo (created_at e completed_at) possibilitam análises de ciclo de vida e velocidade de conclusão. As métricas de esforço (estimated_hours, actual_hours, story_points) são cruciais para o funcionamento dos algoritmos de estimativa e para avaliação da precisão do sistema ao longo do tempo.

Atributo	Tipo	Descrição
id	string	Identificador único da tarefa
title	string	Título descritivo da tarefa
description	string	Descrição detalhada da tarefa
labels	[]string	Lista de labels para categorização
estimated_hours	float64	Horas estimadas pela IA
actual_hours	float64	Horas realmente gastas (histórico)
story_points	int	Story points atribuídos

size	string	Tamanho da tarefa (XS, S, M, L, XL)
created_at	timestamp	Data de criação
completed_at	timestamp	Data de conclusão
developer	string	Desenvolvedor responsável
repository	string	Repositório de origem

Tabela 5.2 - Entidade: Task

Na Tabela 5.3 a entidade Estimation armazena o resultado completo de uma operação de estimativa realizada pelo sistema. Esta estrutura mantém não apenas os valores numéricos calculados, mas também informações contextuais que permitem rastreabilidade e auditoria das estimativas geradas. O campo *reasoning* é particularmente importante pois contém a explicação gerada pelo modelo de linguagem sobre como a estimativa foi calculada, proporcionando transparência ao processo. A lista de *similar_tasks* referência as tarefas históricas que foram utilizadas como base para a estimativa, enquanto o campo *confidence* quantifica o nível de certeza do sistema sobre a precisão da estimativa. O atributo *model* registra qual versão do modelo de IA foi utilizada, permitindo comparações de desempenho entre diferentes versões ao longo do tempo.

Atributo	Tipo	Descrição
id	string	Identificador único da estimativa
task_id	string	Referência para a tarefa
estimated_hours	float64	Horas estimadas
story_points	int	Story points calculados
size	string	Classificação de tamanho
confidence	float64	Nível de confiança (0.0-1.0)
reasoning	string	Explicação da IA sobre a estimativa
similar_tasks	[]SimilarTask	Lista de tarefas similares encontradas
warnings	[]string	Avisos sobre limitações

created_at	timestamp	Data da estimativa
model	string	Modelo de IA usado (ex: gpt-4)

Tabela 5.3 - Entidade: Estimation

A entidade *SimilarTask* na Tabela 5.4 representa tarefas históricas identificadas como similares durante o processo de estimativa. Estas tarefas são descobertas através dos algoritmos de similaridade textual (TF-IDF e coseno) que comparam a nova tarefa com o histórico armazenado localmente. Cada tarefa similar contribui para a estimativa final através de uma média ponderada baseada no *similarity_score*, onde tarefas mais similares têm maior peso na recomendação. O atributo *accuracy_rate* preserva a taxa de acerto da estimativa original daquela tarefa histórica, calculada pela razão entre *estimated_hours* e *actual_hours*, permitindo que o sistema identifique padrões de estimativas consistentemente precisas ou imprecisas para determinados tipos de tarefas. Esta informação é crucial para calibração contínua do sistema e para geração de warnings quando tarefas similares historicamente apresentaram estimativas muito imprecisas.

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

Tabela 5.4 - Entidade: SimilarTask

A entidade Issue representa na Tabela 5.5 apresenta as issues do GitHub armazenadas localmente após o processo de sincronização via comando `set sync`. Esta entidade combina dados obtidos tanto da REST API quanto da GraphQL API do GitHub, sendo enriquecida com custom fields definidos em GitHub Projects. A estratégia de armazenamento local em BoltDB permite que o sistema funcione offline após a sincronização inicial e elimina limitações de rate limiting da API do GitHub durante operações de estimativa. Os campos *actual_hours*, *story_points* e *estimated_size* são extraídos de custom fields configurados no GitHub Projects, representando métricas registradas pela equipe ao longo do ciclo de vida da tarefa. O campo *custom_fields* mantém outros campos personalizados em formato JSON, permitindo flexibilidade para diferentes configurações de projetos. A combinação de dados descritivos (title, description, labels) com métricas de execução

real (actual_hours, story_points) torna esta entidade a base fundamental para os algoritmos de similaridade e estimativa do sistema.

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

Tabela 5.5 - Entidade: Issue (GitHub)

A entidade PullRequest da Tabela 5.6 armazena informações sobre pull requests sincronizados do GitHub, complementando os dados de issues com métricas objetivas sobre a implementação real das tarefas. Esta entidade captura dados quantitativos como número de commits, linhas adicionadas e removidas, e quantidade de arquivos alterados, fornecendo indicadores concretos sobre a complexidade e escopo de mudanças realizadas. O relacionamento com issues através do campo issue_number permite correlacionar estimativas iniciais com métricas de implementação efetiva, possibilitando análises futuras de precisão das estimativas. As timestamps created_at e merged_at permitem calcular o tempo de ciclo de desenvolvimento, métrica valiosa para calibração do sistema. A diferença entre additions e deletions pode indicar se uma tarefa envolveu predominantemente código novo ou refatoração de código existente, padrões que podem influenciar futuras estimativas.

Esta entidade enriquece significativamente o dataset histórico ao adicionar dimensões objetivas e mensuráveis que complementam as descrições textuais das issues.

Atributo	Tipo	Descrição
id	string	ID único do GitHub
number	int	Número do PR no repositório
repository_id	string	Referência ao repositório
issue_number	int	Issue relacionada (se houver)
title	string	Título do PR
state	string	Estado (open, closed, merged)
created_at	timestamp	Data de criação
merged_at	timestamp	Data de merge
commits	int	Número de commits
additions	int	Linhas adicionadas
deletions	int	Linhas removidas
changed_files	int	Arquivos alterados
author	string	Autor do PR

Tabela 5.6 - Entidade: PullRequest

A Figura 5.1 mostra a estrutura do banco de dados BoltDB, organizada em buckets (equivalentes a tabelas) e os relacionamentos entre as entidades.

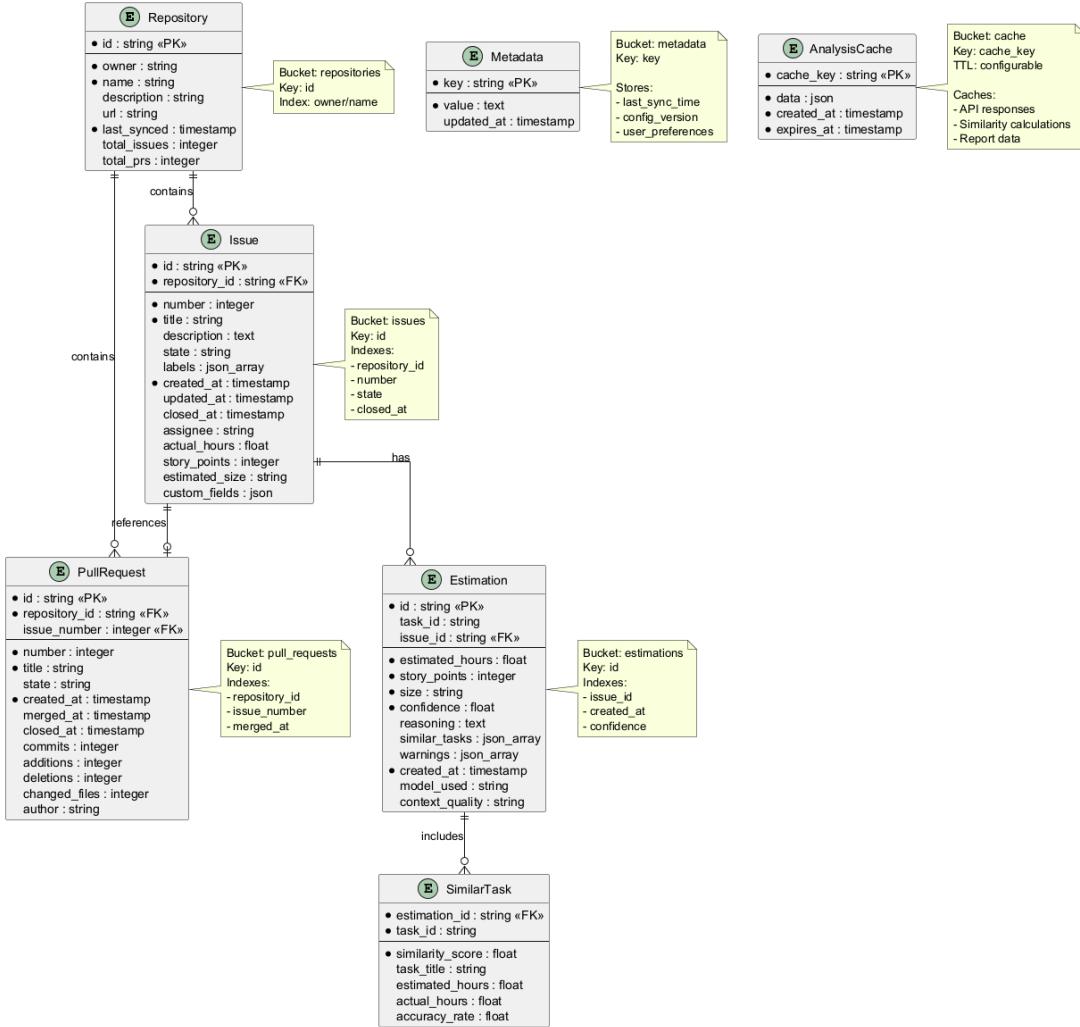


Figura 5.1 - Diagrama Entidade-Relacionamento

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.

A Tabela 6.1 demonstra a Configuração inicial iterativa. 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
Identificador	TA01
Casos de Uso	UC01, UC02
Necessidade	Integração com Ferramentas Existentes (N4)
Caso de teste	Configurar sistema pela primeira vez no modo interativo
Pré-condições	<ul style="list-style-type: none">• Sistema não configurado previamente• GitHub token válido disponível• OpenAI API key válida disponível

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

Tabela 6.1 - TA01 - Configuração Inicial Interativa**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
Identificador	TA02
Casos de Uso	UC08
Necessidade	Integração com Ferramentas Existentes (N4)
Caso de teste	Validar tokens e conectividade com APIs externas
Pré-condições	<ul style="list-style-type: none"> Sistema configurado Conexão com internet disponível
Dados de entrada	Comando: set configure --validate

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

Tabela 6.2 -TA02 - Validação de Configuração

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
Identificador	TA03
Casos de Uso	UC03

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

Tabela 6.3 -TA03 - Estimativa com Alta Similaridade

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

- Sugestão de quebrar em tarefas menores se XL

Tabela 6.4 - TA04 - Estimativa com Baixa Similaridade**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
Identificador	TA05
Casos de Uso	UC03
Necessidade	Precisão em Estimativas de Esforço (N1)
Caso de teste	Estimativa com descrição detalhada e contexto
Pré-condições	<ul style="list-style-type: none"> • Sistema configurado e sincronizado
Dados de entrada	Comando: set estimate "Migração de banco PostgreSQL" --description "Migrar 50GB de dados, 30 tabelas, com zero downtime" --labels database,migration --priority high
Ações	<ol style="list-style-type: none"> 1. Sistema processa título + descrição detalhada 2. Labels adicionam peso à busca de similaridade 3. Priority pode influenciar complexidade assumida 4. AI analisa contexto completo
Resultado esperado	<ul style="list-style-type: none"> • Estimativa mais precisa devido ao contexto • Reasoning menciona "50GB", "30 tabelas", "zero downtime" • Riscos identificados pela IA listados • Recomendações técnicas incluídas

Critérios de aceitação	<ul style="list-style-type: none"> Descrição detalhada aumenta precisão em ~20% Riscos relevantes identificados (ex: downtime, data loss) Próximos passos sugeridos pela IA
-------------------------------	--

Tabela 6.5 - TA05 - Estimativa com Contexto Adicional

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
Identificador	TA06
Casos de Uso	UC04
Necessidade	Automatização do Processo de Estimativa (N2)
Caso de teste	Processar lote de tarefas para sprint planning
Pré-condições	<ul style="list-style-type: none"> Sistema configurado Arquivo sprint-backlog.json com 15 tarefas Base histórica disponível
Dados de entrada	Arquivo JSON com 15 tarefas
Ações	<ol style="list-style-type: none"> Executar set batch --file sprint-backlog.json --workers 5 --format text Sistema cria worker pool com 5 goroutines Tarefas distribuídas entre workers Processamento paralelo com agregação thread-safe

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

Tabela 6.6 - TA06 - Processamento em Lote - Planning Sprint**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
Identificador	TA07
Casos de Uso	UC04, UC06
Necessidade	Automatização (N2), Integração (N4)
Caso de teste	Exportar resultados de batch em diferentes formatos
Pré-condições	<ul style="list-style-type: none"> Estimativas geradas via batch

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

Tabela 6.7 - TA07 - Exportação em Múltiplos Formatos

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

Tabela 6.8 - TA08 - Sincronização Incremental

TA09 - Sincronização Full com Custom Fields

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

-
- Accuracy calculável (estimado vs real)
-

Tabela 6.9 - TA09 - Sincronização Full com Custom Fields**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
Identificador	TA10
Casos de Uso	UC05
Necessidade	Utilização de Contexto Histórico (N3)
Caso de teste	Consultar e filtrar dados históricos
Pré-condições	<ul style="list-style-type: none"> • Base histórica sincronizada • ≥50 issues com dados completos
Dados de entrada	Comandos: <ul style="list-style-type: none"> • set inspect --list • set inspect --list --custom • set inspect --filter backend --limit 20
Ações	<ol style="list-style-type: none"> 1. Listar todas as issues armazenadas 2. Listar apenas issues com custom fields 3. Filtrar por label e limitar resultados
Resultado esperado	<ul style="list-style-type: none"> • Lista paginada de issues exibida • Custom fields destacados quando disponíveis • Filtros funcionando corretamente • Performance: <2s para consulta de 1000+ issues

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

Tabela 6.10 - TA10 - Análise de Histórico**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
Identificador	TA11
Casos de Uso	UC06
Necessidade	Integração com Ferramentas Existentes (N4)
Caso de teste	Exportar dados históricos para análise externa
Pré-condições	<ul style="list-style-type: none"> • Base histórica com estimativas • Dados de accuracy disponíveis
Dados de entrada	Comandos: <ul style="list-style-type: none"> • set export --format csv --output metrics.csv • set export --format json --output data.json • set export --format markdown --output report.md

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

Tabela 6.11 -TA11 - Exportação para Ferramentas Externas

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, 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
Identificador	TI01

Componentes	GitHubClient → GitHub REST API → BoltRepository → BoltDB
Caso de teste	Fluxo completo de sincronização e persistência
Pré-condições	<ul style="list-style-type: none"> • Token GitHub válido com scopes: repo, read:org • Repositório público ou acessível • BoltDB em <code>~/set/data.db</code>
Dados de entrada	Comando: <code>set sync --repository user/project</code>
Ações	<ol style="list-style-type: none"> 1. GitHubClient: Autenticação via token 2. GitHubClient: GET <code>/repos/user/project</code> (validação) 3. GitHubClient: GET <code>/repos/user/project/issues</code> (paginado) 4. GitHubClient: GET <code>/repos/user/project/pulls</code> (paginado) 5. BoltRepository: Serialização para JSON 6. BoltDB: Batch insert em bucket "issues" 7. BoltDB: Batch insert em bucket "prs" 8. Metadata: Atualização de <code>lastSyncTime</code>
Resultado esperado	<ul style="list-style-type: none"> • Issues: Todas persistidas (verificar count) • Pull Requests: Todas persistidas • Relacionamentos: <code>issue.number ↔ pr.number</code> preservados • Integridade: Dados recuperáveis via <code>set inspect</code> • Transações: Atomicidade garantida • Índices: B+tree criado para buscas rápidas
Validação	<ul style="list-style-type: none"> • Comparar count GitHub vs BoltDB • Verificar schema dos objetos persistidos • Testar query de similaridade após sync • Validar relacionamentos (foreign keys)
Tratamento de erros	<ul style="list-style-type: none"> • Rate limit 403: Retry com exponential backoff • Network timeout: Retry até 3 vezes • Corrupted data: Rollback de transação

Tabela 6.12 - Integração com GitHub API

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

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

Tabela 6.13 - Integração com OpenAI API

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.

A Tabela 7.1 ilustra o cronograma, o qual foi estruturado considerando as datas marco estabelecidas e organizando as atividades em sprints quinzenais para facilitar o acompanhamento e entregas incrementais.

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

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

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

Tabela 7.1 - Cronograma de Implementação

7.1 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:
 - Validação de entradas
 - Tratamento de erros
 - Fallbacks e recuperação
 - Logs e observabilidade
 - 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

Esta seção contém a análise crítica do processo de desenvolvimento realizada após a entrega do projeto. O objetivo é identificar os pontos fortes, os desafios encontrados e as lições aprendidas durante todo o processo de desenvolvimento.

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 arquiteturais foram enfrentados, conforme pode ser observado na Tabela 8.1.

Desafio	Desafio
Tratamento de Dados Inconsistentes do GitHub	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.
Balanceamento entre Custo de API e Precisão	Sistema de cache inteligente para evitar chamadas repetitivas. Batch processing para reduzir overhead por requisição.
Definição de Thresholds para Similaridade	Implementação de threshold adaptativo que se ajusta baseado na quantidade e qualidade de dados históricos disponíveis. Múltiplos níveis de confiança informados ao usuário.
Sincronização com GitHub Rate Limits	Implementação de rate limiting awareness com backoff exponencial. Cache agressivo de dados já sincronizados. Sync incremental baseado em timestamps.

Tabela 8.1 - Desafios e Soluções

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 descriptivas 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.

Supporte 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-tcci-0393100-dev-inacio-moraes>