SET - Software Estimation Tool

Aluno: Inácio Moraes
Orientador:Cleiton Silva

Objetivos do Projeto

01

Redução da subjetividade nas estimativas de esforço

02

Otimização do tempo e precisão no planejamento.

03

Aumento da precisão utilizando dados históricos do GitHub.

04

Estimativa gerada bem contextualizada.

Escopo

01

Estimar tarefas individualmente utilizando inteligência artificial

02

Processar todo o backlog em lote e de forma eficiente

03

Sincronizar e utilizar dados históricos reais do GitHub

Escopo

04

Gerar estimativas em horas/pontos e com nível de confiança

05

Oferecer raciocínio detalhado para cada resultado obtido

Fora do Escopo

Interface

Interface web ou mobile não será desenvolvida; apenas CLI

Integrações

Não há integração com Jira ou Azure DevOps, somente GitHub

AI

Não utiliza modelo de Machine Learning próprio; depende de API OpenAI

Fora do Escopo

Dados

Não oferece autenticação multi-usuário ou análise de desempenho

Integrações

Estimativas dependem da qualidade dos dados históricos e API Keys

API

Existem limitações de rate limits nas APIs utilizadas (GitHub e OpenAI)

Atores

Ana Rodrigues
Desenvolvedor Full Stack Sênior

Ana é desenvolvedora sênior em uma consultoria de software 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.

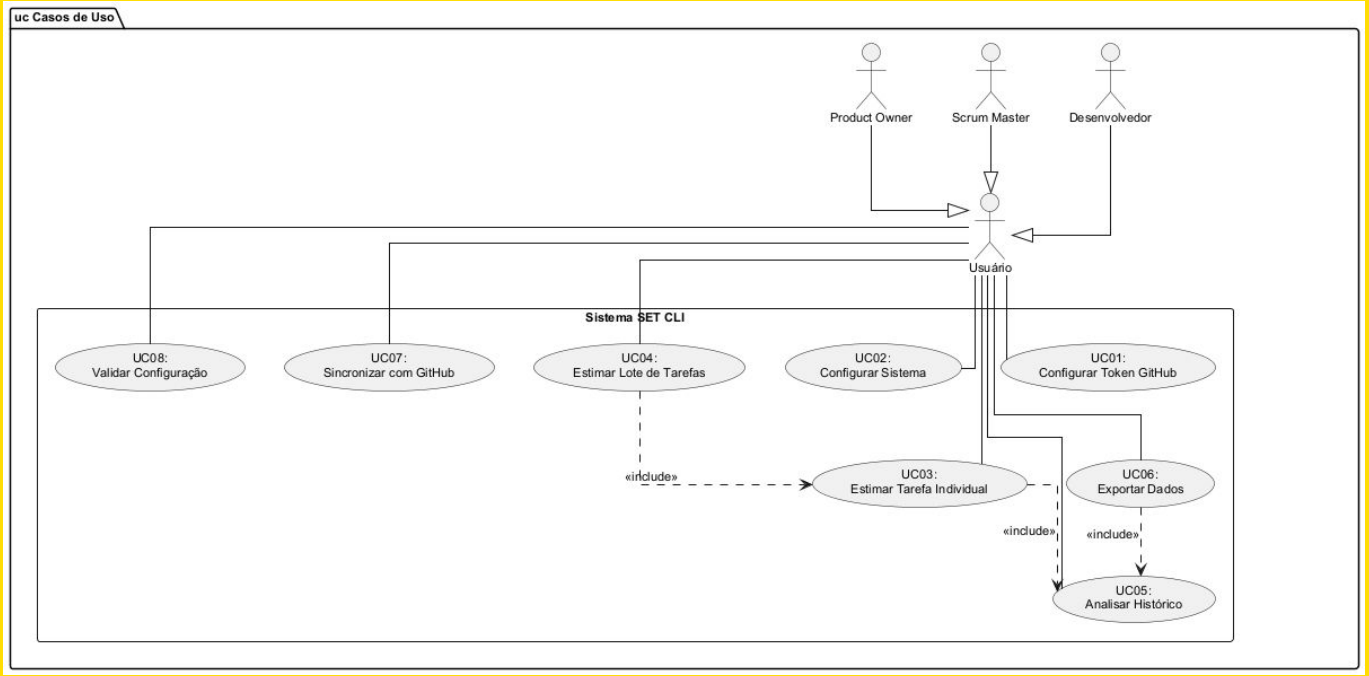
Carlos Mendes
Scrum Master

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.

Roberto Santos
Product Owner

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.

Casos de Uso


uc Casos de Uso
Product Owner
Scrum Master
Desenvolvedor
Usuário
Sistema SET CLI
UC08:
Validar Configuração
UC07:
Sincronizar com GitHub
UC04:
Estimar Lote de Tarefas
UC02:
Configurar Sistema
UC01:
Configurar Token GitHub
«include»
UC03:
Estimar Tarefa Individual
UC06:
Exportar Dados
«include»
«include»
UC05:
Analisar Histórico

Funcionalidades

	Comando na CLI
Configurar Token GitHub	*set configure --github-token*
Configurar Sistema	*set configure --initial*
Estimar Tarifa Individual	*set estimate "<descrição>"*
Estimar Lote de Tarefas	*set batch --file <arquivo>*

Funcionalidades

	Comando na CLI
Analisar Histórico	*set configure --github-token*
Exportar Dados	*set configure --initial*
Sincronizar com GitHub	*set estimate "<descrição>"*
Validar Configuração	*set batch --file <arquivo>*

Necessidades e Funcionalidades

01

Configurar Token GitHub - *set configure --github-token*

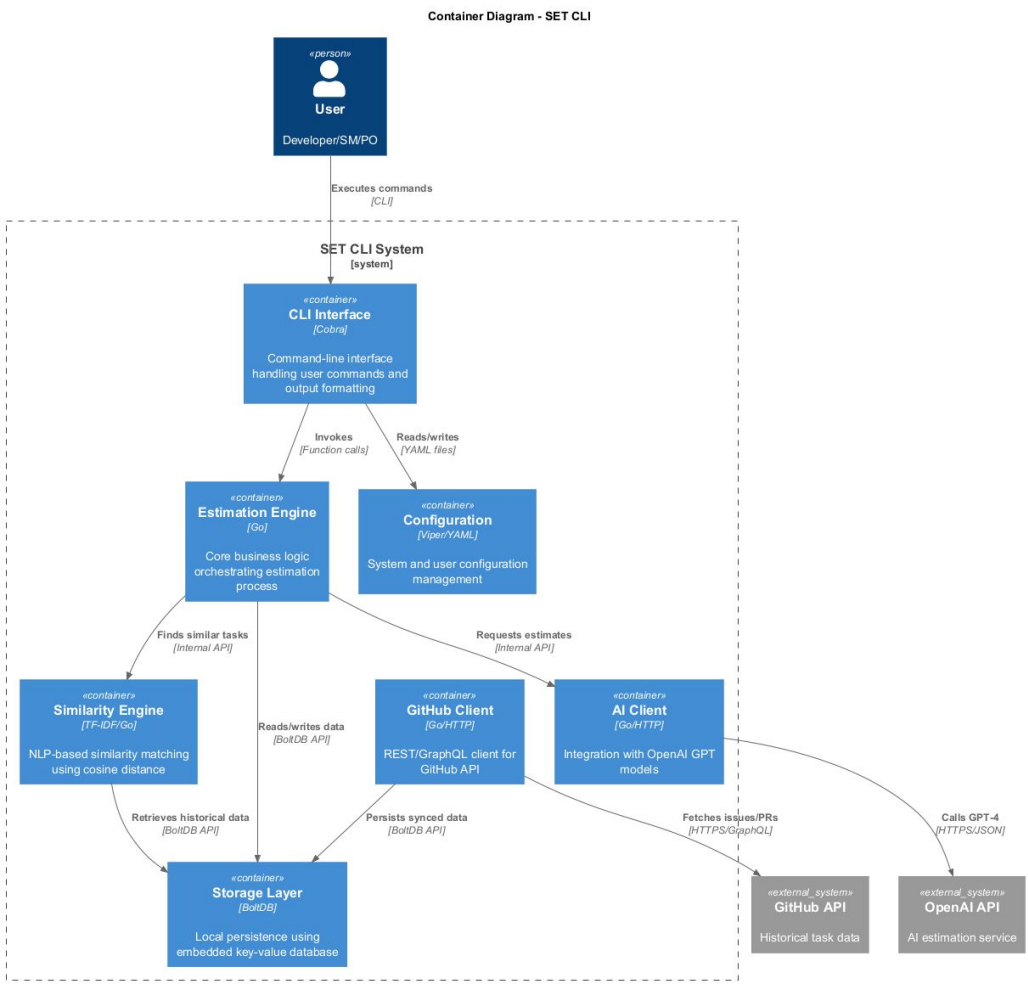
02

Processar todo o backlog em lote e de forma eficiente

03

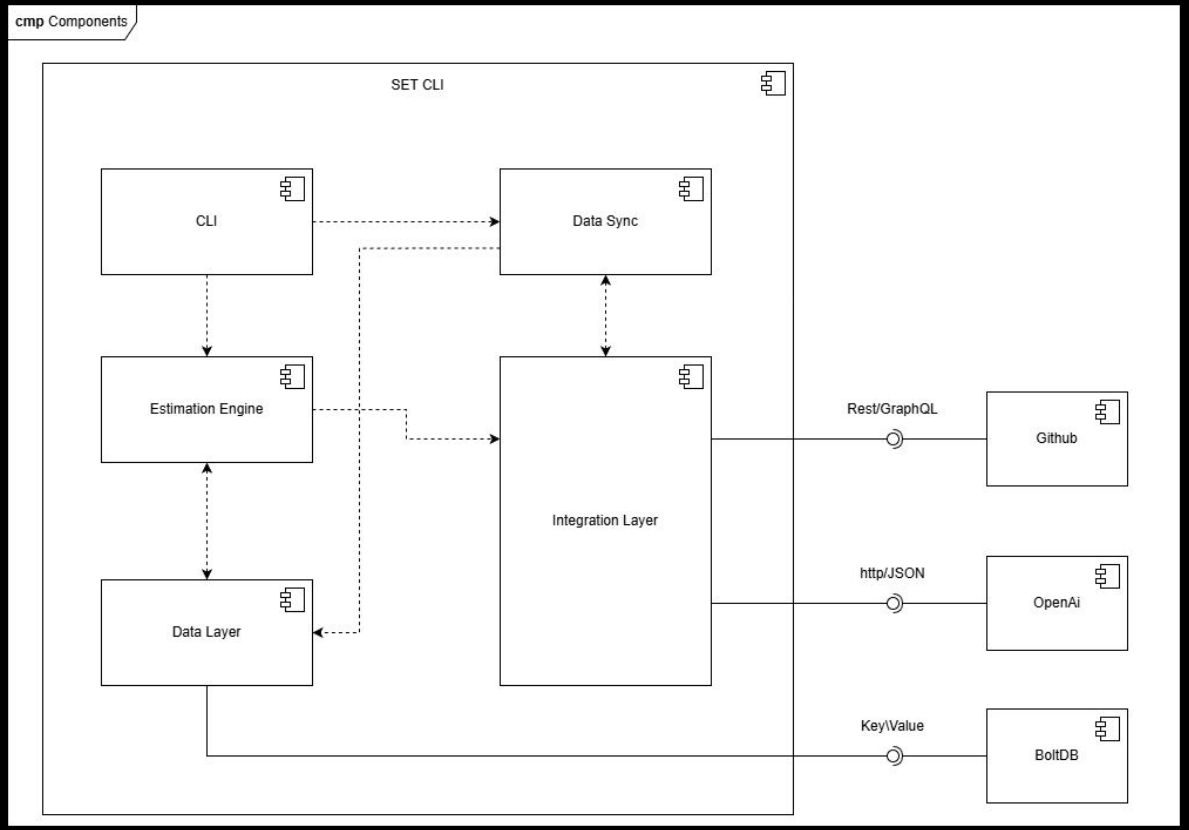
Sincronizar e utilizar dados históricos reais do GitHub

Arquitetura



O diagrama de contêineres da 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.

Principais Componentes

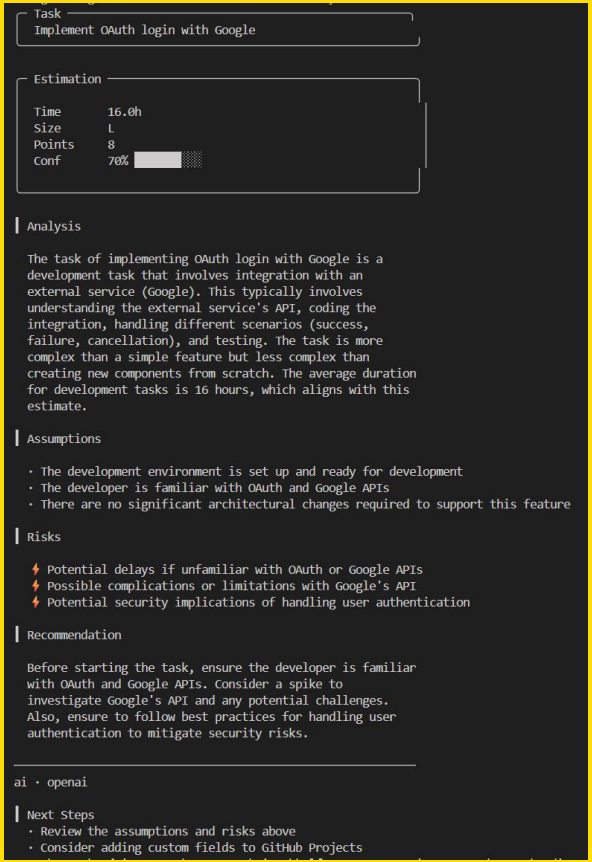


O diagrama de componentes, demonstra a organização modular do sistema em componentes de alto nível, suas interfaces fornecidas e requeridas, e as dependências entre eles

Interface

Comando Principal de estimar.

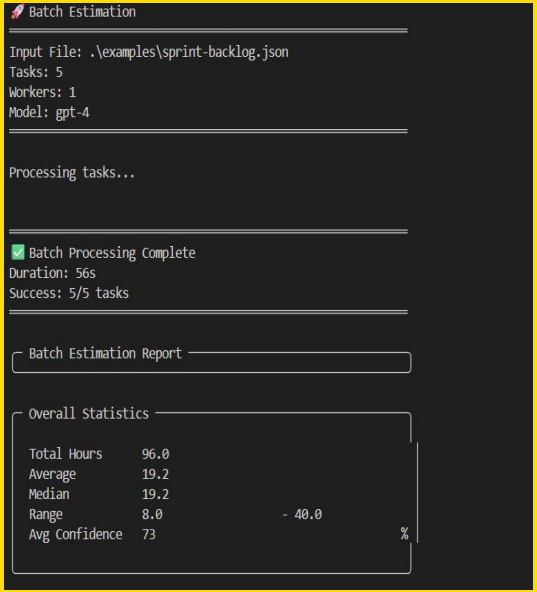
set estimate "Implement OAuth login with Google"

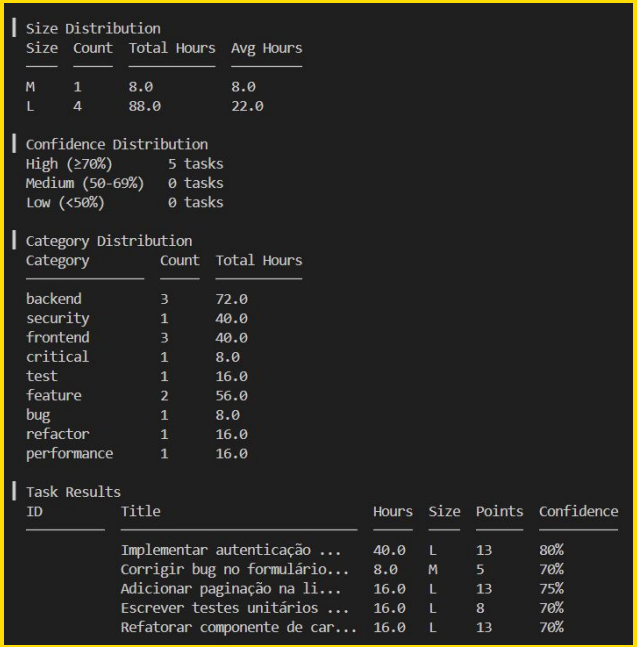


Interface

Comando Principal de estimar.

set batch --file .\examples\sprint-backlog.json

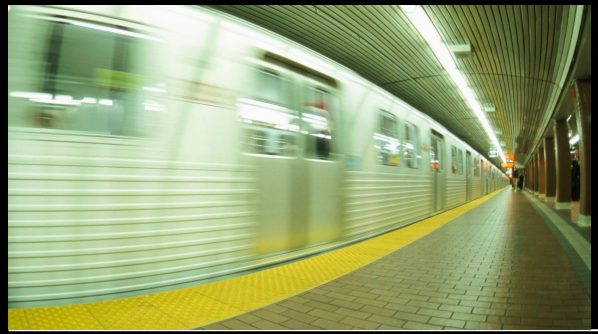




Demo

Post-mortem

01 Experiência Positivas



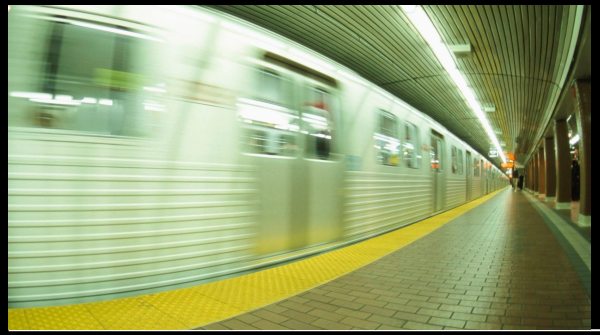
Escolha da Linguagem Go: O uso de Go foi acertado devido à facilidade de criar binário único multiplataforma, excelente performance nativa, suporte eficiente a concorrência para processamento paralelo e a familiaridade da equipe com a linguagem, acelerando o desenvolvimento.

Integração com IA (OpenAI): A API OpenAI superou expectativas na geração de estimativas baseadas em linguagem natural, agregando valor ao produto pela capacidade da IA de compreender contexto e gerar raciocínios explicativos.

Arquitetura Clean Architecture: A adoção dos princípios de Clean Architecture facilitou a manutenção, testabilidade e evolução do código, com uma separação clara entre as camadas de domínio, casos de uso e infraestrutura.

Post-mortem

02 Desafios e Soluções



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.

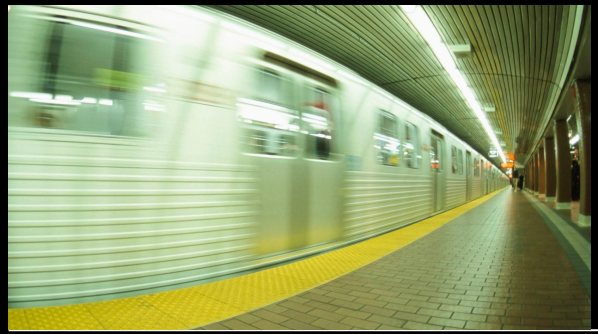
Balanceamento entre Custo de API e Precisão:
Sistema de cache inteligente para evitar chamadas repetitivas. Batch processing para reduzir overhead por requisição.

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.

Post-mortem

03 Lições Aprendidas



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

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

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

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

Post-mortem

04 Melhorias Futuras

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

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

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

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

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

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

Obrigado!