"Cmd-Vault" (Gerenciador de Comandos TUI Retro)

O objetivo é criar um utilitário de linha de comando (cmd-vault) leve e autônomo em **Go (Golang)**, focado em gerenciamento e execução de comandos salvos.

I. Restrições e Tecnologias Fundamentais (Não Negociáveis)

Requisito	Tecnologia/Plataforma	Detalhe Crucial
Linguagem Principal	Go (Golang)	A aplicação deve ser escrita integralmente em Go.
Persistência	SQLite	Banco de dados em arquivo único (vault.db) para portabilidade.
Interface Principal	TUI (Terminal User Interface)	Deve usar **Bubbletea** para o *framework* e **Lipgloss** para o estilo.
Alvo de Distribuição	Windows	O foco principal é a geração de um **executável único (.exe)** e *autônomo* para o sistema operacional Windows.
Estética	Vintage "Anos 80"	Cores **rigorosamente monocromáticas** (e.g., verde neon/âmbar em fundo preto), fontes mono-espaçadas e bordas de caracteres.

II. Estrutura e Modos de Operação

1. **Modo CLI (cmd-vault run <name>):**
    - Deve usar **Cobra** para *parsing* de comandos.
    - Permite a execução direta de um comando salvo pelo seu Name.
2. **Modo TUI (Interativo) (cmd-vault sem argumentos):**
    - Inicia a interface de painéis interativa, sendo o modo principal.

III. Layout TUI e Estética (Inspiração: lazydocker)

A TUI deve ser composta por três painéis principais e um rodapé, seguindo a estética "Anos 80":

Painel	Conteúdo	Interatividade	Estilo Lipgloss
Painel de Comandos (Esquerda)	Lista de comandos (Nome, UsageCount).	Navegável (setas). O item selecionado dita o conteúdo dos painéis à direita.	Fundo escuro, cor de destaque monocromática (e.g., verde neon) para o item selecionado.
Painel de Detalhes (Superior Direita)	Exibe o Name e o Command completo do item selecionado.	**Editável** no Modo Edição.	Borda de caracteres grossa, cor monocromática.
Painel de Notas (Inferior Direita)	Exibe a Note completa (campo **obrigatório** e não vazio).	**Editável** no Modo Edição.	Borda de caracteres fina, texto justificado.
Rodapé / Barra de Status	Atalhos (Ex: [A] Adicionar, [E] Editar, [R] Rodar, [D] Apagar, [Q] Sair, [?] Ajuda).	Indica o status atual (Modo Normal / Modo Edição).	Fundo escuro, texto de alto contraste.

IV. Funcionalidades e Interatividade (CRUD)

Ação	Atalho	Requisitos
Adicionar	[A]	Abre um **modal/formulário** para inserir Name, Command e Note. Deve validar: **Nome Único** e **Note Não Vazia**.
Editar	[E]	Alterna para o Modo Edição, tornando os Painéis de Detalhes e Notas **diretamente editáveis**. Salva as alterações ao sair do modo.
Apagar	[D]	Exige uma **confirmação na tela** (modal de alerta/pergunta, **NUNCA** prompt nativo) antes do DELETE.

Ação	Atalho	Requisitos
Rodar/Executar	[R]	**Execução de Comando (Crítica):** Usar **os/exec** configurado especificamente para o **Shell Nativo do Windows** (cmd.exe com /C ou powershell.exe com -c). O TUI deve **pausar ou fechar temporariamente** durante a execução e **retornar** ao TUI após o comando. Deve incrementar o UsageCount.
Ajuda	[?]	Exibe um *overlay* ou modal com a descrição de todos os atalhos.

V. Persistência de Dados (SQLite Schema)

A tabela principal (commands) deve ter a seguinte estrutura:

- id (INTEGER PRIMARY KEY)
- name (TEXT UNIQUE NOT NULL)
- command_str (TEXT NOT NULL)
- note (TEXT NOT NULL)
- usage_count (INTEGER DEFAULT 0)
- created_at (TEXT/TIMESTAMP NOT NULL)

Lista de Objetivos Verificáveis (Checklist da IA)

Esta lista divide o projeto em fases, permitindo que a IA construa e teste incrementalmente.

Fase 1: Setup e Persistência (Backend)

Objetivo	Status	Verificação
1.0 Inicialização do Projeto Go (go mod init).	[ ]	Estrutura básica de pastas criada.
1.1 Implementação da Conexão e Inicialização do SQLite.	[ ]	Função de inicialização de DB que cria o arquivo e a tabela commands se não existirem.

Objetivo	Status	Verificação
1.2 Implementação do CRUD Básico (Modelo de Dados).	[ ]	Funções Go para Insert, Select All, Select by Name, Update, e Delete funcionando via CLI de teste.
1.3 Implementação do Modo CLI com Cobra.	[ ]	Comando cmd-vault run <name> que consulta o DB e executa o comando via os/exec no Windows (configuração cmd /C).

Fase 2: Construção e Estilização do TUI

Objetivo	Status	Verificação
2.1 Configuração do TUI com Bubbletea e Lipgloss.	[ ]	Modelo Bubbletea básico (model.go) que carrega dados do DB na inicialização.
2.2 Estilização "Anos 80" (Lipgloss).	[ ]	Esquema de cores monocromático definido e aplicado globalmente.
2.3 Construção do Layout de Painéis.	[ ]	Implementação da divisão principal (Painel Esquerdo, Painéis Direitos Superior/Inferior, Rodapé).
2.4 Implementação da Navegação na Lista (Painel Esquerdo).	[ ]	Navegação vertical (setas) funcionando e atualizando os painéis de Detalhes/Notas.

Fase 3: Interatividade e Execução

Objetivo	Status	Verificação
3.1 Implementação do Modal de Confirmação para Apagar ([D]).	[ ]	Modal customizado exibido antes do DELETE do comando.
3.2 Implementação do Fluxo de Adicionar ([A]).	[ ]	Modal/Formulário de entrada de dados funcionando, com validação de unicidade (Name) e não-vazio (Note).

Objetivo	Status	Verificação
3.3 Implementação do Modo Edição ([E]).	[ ]	Os painéis Detalhes e Notas se tornam campos de entrada editáveis ao pressionar [E], salvando no DB ao sair.
3.4 Implementação da Execução de Comandos ([R]).	[ ]	Comando selecionado é executado no shell do Windows (cmd /C) e o UsageCount é incrementado. O TUI retorna à interface após a execução.
3.5 Implementação do Rodapé e Ajuda ([?]).	[ ]	O Rodapé exibe as teclas de atalho e o modal de ajuda funciona.

Fase 4: Finalização e Distribuição

Objetivo	Status	Verificação
4.1 Tratamento de Erros no DB e TUI.	[ ]	Mensagens de erro visíveis e tratadas dentro do TUI (e.g., erro ao salvar, DB corrompido).
4.2 Geração do Executável Único para Windows.	[ ]	Comando de go build (GOOS=windows GOARCH=amd64) configurado para gerar um único .exe sem dependências externas.

Este formato fornece à IA um guia claro e testável, com ênfase nas restrições de *design* e *cross-platform* (Windows).

Este é um excelente ponto. Embora o prompt anterior fosse muito detalhado, podemos adicionar especificações de baixo nível (como estruturas e cores) para **eliminar a ambiguidade** e facilitar a codificação imediata pela IA.

Abaixo estão as informações adicionais focadas em **estruturas de dados**, **dependências exatas** e **especificações de estilo concretas**.

---

1. Estruturas de Dados e Dependências Exatas

Para começar a codificar imediatamente, a IA deve usar estas definições exatas:

A. Estrutura Principal em Go (Modelo de Comando)

A estrutura Go (struct) que interage diretamente com o banco de dados e o modelo Bubbletea (model) deve ser definida como:

```
Go

type Command struct {
  ID        int
  Name      string
  CommandStr  string // O comando real a ser executado (e.g., "git status")
  Note      string // Descrição/contexto (Obrigatório, NOT NULL)
  UsageCount  int
  CreatedAt   time.Time
}
```

B. Dependências Go Mandatórias

A IA deve garantir que apenas estas bibliotecas principais (e suas dependências transitivas) sejam usadas, minimizando a superfície de ataque e garantindo a leveza:

1. **Interface TUI:** github.com/charmbracelet/bubbletea
2. **Estilo TUI:** github.com/charmbracelet/lipgloss
3. **CLI/Estrutura:** github.com/spf13/cobra
4. **Driver SQLite:** github.com/mattn/go-sqlite3

2. Especificação Concreta de Estética (Lipgloss)

Para impor o estilo **Vintage "Anos 80" Monocromático**, vamos definir o esquema de cores exato:

Paleta de Cores (Estilo Âmbar em Fundo Escuro):

- **Fundo (Background):** lipgloss.Color("#000000") ou simplesmente a cor padrão do terminal.
- **Cor Principal (Neon/Âmbar):** lipgloss.Color("#FFBF00") (Âmbar/Laranja Neon).
- **Cor Secundária (Destaque Quente):** lipgloss.Color("#E5AA00") (Âmbar Ligeiramente mais Escuro, para bordas).
- **Cor de Seleção (Active Item):** Inverter a cor, ou usar **negrito** e **sublinhado** com a cor principal.

Requisito de Estilo Adicional:

- **Bordas:** Usar as constantes de borda de caracteres do Lipgloss (e.g., lipgloss.Border{Top: "=", Bottom: "=", Left: "|", Right: "|", ...}). As bordas de painel nunca devem ser arredondadas ou usar glifos complexos, apenas caracteres simples (-, |, =, etc.).

3. Lógica Crítica de Execução no Windows

O ponto de maior falha potencial é a execução de comandos, que **deve** respeitar a sintaxe e as variáveis de ambiente do Windows.

Instrução para os/exec (Função Rodar/Executar):

Ao receber uma *string* de comando do banco de dados (ex: echo %PATH%), a função Go **não deve** tentar parsear ou executar o comando diretamente.

A implementação correta para Windows deve ser:

Go

// commandString é o valor de CommandStr do banco de dados

cmd := exec.Command("cmd", "/C", commandString)

// cmd := exec.Command("powershell", "-Command", commandString) // Alternativa

// O output deve ser retransmitido ao terminal do usuário

cmd.Stdout = os.Stdout

cmd.Stderr = os.Stderr

// Executar e tratar o erro de execução

err := cmd.Run()

// ... Lógica para verificar 'err' e retornar ao TUI ...

A IA deve **priorizar o uso de cmd /C** por ser universalmente disponível e mais direto para comandos *legacy* do Windows.

4. Tratamento de Erros e Edge Cases

A IA deve garantir que os seguintes **erros críticos** sejam tratados de forma elegante, retornando uma mensagem de erro na TUI (no rodapé ou em um modal) em vez de travar:

1. **Erro de Execução (R):** Se o comando rodado falhar (ex: c:\nonexistent.exe), o TUI deve capturar o código de saída e exibir "Comando falhou com código de saída X" antes de voltar à lista.
2. **Duplicidade (A):** Se o usuário tentar adicionar um Name que já existe, deve aparecer uma mensagem de erro clara no formulário de adição, sem perder os dados já inseridos.
3. **Persistência (Geral):** Se houver falha ao ler ou escrever no SQLite, o TUI deve exibir uma mensagem de erro fatal, possivelmente registrando o erro em um arquivo de log simples, antes de sair.