Plano de Desenvolvimento de Software: KeyAI Desktop v1.0

1. Objetivo do Documento

Este Plano de Desenvolvimento de Software (PDS) estabelece a referência técnica e operacional para o projeto KeyAI Desktop v1.0. O documento detalha a arquitetura, o escopo, o cronograma e os recursos necessários para o desenvolvimento, teste, lançamento e suporte inicial do produto.

2. Escopo do Projeto

Premissa: O escopo definido refere-se exclusivamente à versão 1.0 do produto.

Funcionalidades Incluídas (v1.0)

- Captura de digitação em background (Windows, macOS, Linux).
- Mascaramento local e em tempo real de PII (e.g., CPF, e-mail).
- Armazenamento local criptografado com SQLite e SQLCipher.
- Busca de texto completo (Full-Text Search) via FTS5.
- Busca por similaridade semântica (Vector Search) via sqlite-vec.
- GUI minimalista para busca, visualização e configurações básicas.
- Extração de texto de screenshots (OCR) de forma offline.

Funcionalidades Excluídas (Fora do Escopo)

- Sincronização de dados com a nuvem ou serviços externos.
- Suporte para plataformas móveis (iOS, Android).
- Recursos de colaboração ou compartilhamento de dados entre usuários.
- Suporte a plugins ou extensões de terceiros.
- Análise de sentimento ou categorização avançada de texto.

3. Arquitetura de Alto Nível

A arquitetura do KeyAI Desktop adota uma filosofia "local-first", garantindo que todos os dados sensíveis e operações de processamento permaneçam no dispositivo do usuário. Esta abordagem prioriza a privacidade e a funcionalidade offline, eliminando a dependência de serviços externos e reduzindo os custos operacionais.

Snippet de código

```
@startuml
skinparam componentStyle uml2

package "KeyAI Desktop Application" {
  -down->

  package "Background Services" {
    [Input Capture Agent] --> [PII Masker]
    --> [PII Masker]
    [PII Masker] -->
  }

  <-->
}

database "SQLite (SQLCipher + FTS5 + sqlite-vec)" as DB {

}
@enduml
```

Descrição dos Componentes

- **Input Capture Agent:** Módulo de baixo nível que utiliza a crate rdev para capturar eventos de teclado de forma multiplataforma.[1]
- **OCR Service:** Serviço offline que usa a crate ocrs para extrair texto de imagens, operando sem dependências de rede.[3]
- **PII Masker:** Componente que recebe texto, aplica regras de regex para ofuscar PII e prepara os dados para persistência.
- **Database Access Layer:** Abstração em Rust que gerencia a conexão, migrações

(rusqlite_migration) e as APIs de inserção/busca.[5]

- **Encrypted Database:** Arquivo SQLite criptografado com SQLCipher e estendido com FTS5 e sqlite-vec para buscas avançadas.[7]
- **GUI (Graphical User Interface):** Frontend construído com Tauri, comunicando-se com o backend Rust para renderizar a interface e executar comandos.[10]

A escolha estratégica por dependências como ocrs e rusqlite com a feature bundled-sqlcipher [12] visa minimizar a complexidade de compilação entre plataformas. Ao evitar dependências externas que requerem toolchains C/C++ complexas (como Tesseract [13]), o processo de build se torna mais previsível e robusto, um fator crítico para a manutenção de um produto multiplataforma.

4. Pilha Tecnológica (Tech-Stack)

A seleção tecnológica prioriza crates em Rust nativas ou com dependências estaticamente vinculadas para garantir performance e simplificar a distribuição multiplataforma.

Categoria	Tecnologia	Versão Mínima	Justificativa / Notas
Linguagem	Rust	1.78.0	Performance, segurança de memória e ecossistema robusto.
UI Framework	Tauri	1.7.0	Backend Rust nativo, leve, seguro e multiplataforma.[14]
Captura de Eventos	rdev	0.5.3	Crate consolidada para escuta de eventos de teclado/mouse.[1]
OCR	ocrs	0.10.3	Motor de OCR moderno, offline, em Rust puro, evitando FFI complexo.[4]
Banco de Dados	SQLite	3.42.0	Padrão de mercado para bancos de dados embarcados.
Criptografia DB	SQLCipher	bundled	Criptografia AES-256 via feature bundled-sqlcipher do rusqlite.[12]
Busca Vetorial	sqlite-vec	0.2.0	Sucessor do sqlite-vss, com integração simplificada para Rust.[9]

Interação com DB	rusqlite	0.31.0	Wrapper idiomático e seguro para SQLite em Rust.[12]
Migrações de DB	rusqlite_migration	1.1.0	Biblioteca leve para gerenciar evoluções de schema do DB.[5]

5. Plano de Iterações / Sprints

O desenvolvimento principal está planejado para 7 sprints de duas semanas (14 semanas no total), focando em entregar valor incremental e mitigar os riscos mais altos no início do projeto.

Sprint	Objetivo	Entregáveis Chave
Sprint 0	Configuração e Prova de Conceito (PoC)	Repositório, CI/CD básico (build, lint), App Tauri "Hello World", PoC de rdev no Windows.
Sprint 1	Captura e Armazenamento Bruto	Módulo Agent funcional no Windows/macOS. Integração rusqlite + SQLCipher.
Sprint 2	Camada de Banco de Dados Avançada	Integração de FTS5 e sqlite-vec. Schema inicial do DB com migrações.
Sprint 3	Motor de Mascaramento de PII (v1)	Implementação do PII Masker com regras de regex. Testes unitários de mascaramento.
Sprint 4	GUI e Busca Básica	Tela de busca na GUI (Tauri). Integração da busca FTS5 com o backend.
Sprint 5	Integração de OCR e Busca Vetorial	Módulo OCR Service com ocrs. GUI para acionar OCR. Integração da busca vetorial.
Sprint 6	Polimento, Testes e Captura no Linux	Implementação do Agent no Linux (X11). Testes E2E. Otimização de performance.

6. WBS (Work Breakdown Structure)

A estrutura de trabalho está dividida em épicos que correspondem aos componentes arquiteturais.

- **Epic 1: Core de Captura (Agent)**
  - 1.1: Implementar hook de teclado Windows via rdev.
  - 1.2: Implementar Accessibility API no macOS para rdev (Dep: Permissões TCC).
  - 1.3: Implementar captura em Linux (X11) e pesquisar suporte a Wayland.
  - 1.4: Criar abstração multiplataforma para eventos de teclado.
  - 1.5: Implementar captura de screenshots para o serviço de OCR.
- **Epic 2: Persistência e Busca (Database)**
  - 2.1: Configurar rusqlite com a feature bundled-sqlcipher.[12]
  - 2.2: Definir schema v1 e implementar migrações com rusqlite_migration.[5]
  - 2.3: Habilitar e configurar a extensão FTS5 para busca de texto.[8]
  - 2.4: Integrar e configurar a extensão sqlite-vec via sqlite3_auto_extension.[9]
  - 2.5: Desenvolver DAL com APIs para escrita e leitura.
- **Epic 3: Análise e Mascaramento de Conteúdo (Masker & OCR)**
  - 3.1: Desenvolver motor de regex para PIIs (CPF, Email, Fone).
  - 3.2: Integrar ocrs para extração de texto de imagens.[4]
  - 3.3: Empacotar modelos do ocrs para distribuição offline.[3]
  - 3.4: Criar pipeline de processamento: Input -> OCR/Text -> Masker -> DB.
  - 3.5: Gerar embeddings de texto para alimentar a busca vetorial.
- **Epic 4: Interface do Usuário (GUI)**
  - 4.1: Estruturar projeto Tauri com frontend (React).
  - 4.2: Desenvolver componente de barra de busca e exibição de resultados.
  - 4.3: Criar tela de configurações (habilitar/desabilitar, limpar dados).
  - 4.4: Implementar comunicação segura entre frontend e backend via comandos Tauri.
- **Epic 5: CI/CD e Distribuição**
  - 5.1: Criar workflow de CI no GitHub Actions (test, lint, build).
  - 5.2: Configurar tauri-action para builds multiplataforma.[17]
  - 5.3: Implementar signing e notarização no macOS (Dep: Certificado Apple).[18]
  - 5.4: Implementar signing no Windows (Dep: Certificado EV).[20]
  - 5.5: Automatizar criação de releases no GitHub com artefatos assinados.

A dependência crítica identificada é a obtenção dos certificados de assinatura de código (tarefas 5.3 e 5.4), que é um processo administrativo e deve ser iniciado no Sprint 0 para não bloquear os lançamentos.

7. Cronograma e Recursos (FTE)

Premissa: A equipe é enxuta e composta por especialistas para maximizar a eficiência.

Linha do Tempo Simplificada (6 Meses)

-> -> -> [GA v1.0]

Alocação de Recursos (FTE)

Recurso	Carga (FTE)	Período	Responsabilidades Principais
Arquiteto de Software	0.5 FTE	Meses 1-6	Arquitetura, revisão de código, gestão de riscos.
Desenvolvedor Rust Sênior	1.0 FTE	Meses 1-6	Implementação do backend (Agent, Masker, DB).
Desenvolvedor Rust/Frontend	1.0 FTE	Meses 1-6	Implementação da GUI (Tauri) e integrações.
QA / Testes	0.5 FTE	Meses 3-6	Testes manuais multiplataforma, automação E2E.

8. CI/CD Detalhado

O pipeline de CI/CD será implementado no GitHub Actions para automatizar a verificação, construção e distribuição do aplicativo. A automação de assinatura de código é um pilar para a confiança do usuário, evitando alertas de segurança severos no Windows e permitindo a instalação no macOS.

Workflows do GitHub Actions

1. **pull_request.yml:** Acionado em cada PR, executa cargo clippy, cargo test e cargo build para garantir a qualidade e integridade do código antes do merge. Incluirá um passo para medição de cobertura de código.
2. **release.yml:** Acionado pela criação de uma tag v*, utiliza o tauri-apps/tauri-action [17] com uma matriz de build para

macos-latest, windows-latest e ubuntu-latest. Este workflow será responsável por compilar, assinar, notarizar (no macOS) e anexar os binários à release do GitHub.

Secrets Necessários no GitHub Actions

Secret Name	Plataforma	Propósito	Fonte / Referência
APPLE_CERTIFICATE	macOS	Certificado.p12 em base64	Exportado do Keychain Access [18]
APPLE_CERTIFICATE_PASSWORD	macOS	Senha do certificado.p12	Definida durante a exportação [18]
APPLE_SIGNING_IDENTITY	macOS	Identidade do signatário	Encontrada no Keychain Access [18]
APPLE_ID	macOS	Apple ID para notarização	Credencial da conta de dev Apple [18]
APPLE_PASSWORD	macOS	Senha específica de app	Gerada na conta de dev Apple [18]
WINDOWS_CERTIFICATE	Windows	Certificado.pfx em base64	Gerado a partir do certificado EV [20]
WINDOWS_CERTIFICATE_PASSWORD	Windows	Senha do certificado.pfx	Definida durante a criação do.pfx [20]

9. Qualidade e Testes

As metas de qualidade refletem a proposta de valor do produto: ser uma ferramenta de fundo leve, rápida e discreta.

Metas Quantitativas

- **Cobertura de Código (Backend Rust):** ≥85% (medido com cargo-llvm-cov [22]).
- **Latência de Busca (p95):** ≤150ms (FTS5 em base de 1M de registros).
- **Uso de CPU (Idle):** <2% (processo em background).
- **Uso de Memória (Idle):** <50 MB.

Estratégia de Testes

- **Unitários:** Lógica de negócio no PII Masker e Database Access Layer.
- **Integração:** Interação entre os componentes Agent -> Masker -> DB.
- **End-to-End (E2E):** Automação da GUI para simular fluxos de usuário completos.
- **Performance:** Benchmarks (criterion) para a camada de DB e mascaramento.
- **Compatibilidade:** Testes manuais obrigatórios em Windows 11, macOS Sonoma e Ubuntu 22.04 LTS antes de cada release.

10. Gestão de Riscos

A matriz de riscos prioriza ameaças técnicas e de projeto que podem impactar o cronograma e a qualidade do produto.

Risco	Probabilidade	Impacto	Mitigação
Incompatibilidade com Wayland no Linux	Alta	Alto	Priorizar suporte a X11; oferecer fallback/instruções claras para Wayland.
Baixa acurácia do PII Masker	Média	Alto	Desenvolver suíte de testes extensa; permitir regras customizadas pelo usuário.
Instabilidade em ocrs ou sqlite-vec	Média	Alto	Isolar a funcionalidade em um módulo com API estável (Façade) para facilitar a substituição.
Atraso na obtenção de certificados	Média	Alto	Iniciar processo de compra no Sprint 0; tratar como tarefa de projeto com acompanhamento.
Vulnerabilidade na criptografia do DB	Baixa	Crítico	Usar rusqlite com bundled-sqlcipher[12]; manter dependências atualizadas com cargo-audit.

11. Plano de Lançamento & Suporte

O lançamento será faseado para coletar feedback e garantir a estabilidade.

Fases de Lançamento

1. **Alpha (Interna):** Final do Sprint 6. Distribuída para a equipe interna.
2. **Beta Fechado:** Mês 5. Distribuída para 50-100 usuários técnicos convidados.
3. **Beta Aberto:** Mês 6. Disponível publicamente no site do produto para download.
4. **GA (General Availability) v1.0:** Final do Mês 6. Lançamento oficial.

Critérios de Passagem de Fase

- **Alpha -> Beta Fechado:** Cobertura de testes > 80%; ausência de bugs críticos (crash, perda de dados).
- **Beta Fechado -> Beta Aberto:** Resolução de 90% do feedback crítico; performance dentro das metas.
- **Beta Aberto -> GA:** Nenhum bug crítico novo reportado em 2 semanas; documentação finalizada.

12. Orçamento Detalhado

Premissas: Custos para o primeiro ano. Equipe baseada em Toronto, Canadá (salários em USD para referência). Os custos de pessoal (CAPEX) cobrem 6 meses de desenvolvimento.
O orçamento reflete a estratégia arquitetural. Mais de 98% do custo é CAPEX (desenvolvimento), enquanto o OPEX é mínimo devido à ausência de infraestrutura de nuvem, um benefício direto da abordagem "local-first".

Categoria	Item	Custo Anual (USD)	Justificativa / Fonte
CAPEX	**Desenvolvimento (6 meses)**		
	2.5 FTE Devs @ ~$110k/ano	$137,500	Custo de pessoal para o período de desenvolvimento.[23]
OPEX	**Infraestrutura e Licenças**		

	Apple Developer Program	$99	Taxa anual obrigatória para signing no macOS.[26]
	Windows EV Code Signing Cert.	~$400	Custo médio anual para assinatura no Windows.[28]
	Domínio e Hospedagem (Site)	~$200	Custo para o site do produto e downloads.
	Subtotal OPEX	**~$699**	
Contingência	Fundo para Riscos (15% do CAPEX)	$20,625	Para cobrir riscos não planejados ou atrasos.
TOTAL ESTIMADO (ANO 1)		**~$158,824**	

Referências citadas

1. Crate rdev - Rust - Docs.rs, acessado em junho 27, 2025, https://docs.rs/rdev/
2. rdev - crates.io: Rust Package Registry, acessado em junho 27, 2025, https://crates.io/crates/rdev/0.3.3
3. ocrs - A new open source OCR engine, written in Rust : r/rust - Reddit, acessado em junho 27, 2025, https://www.reddit.com/r/rust/comments/18xhds9/ocrs_a_new_open_source_ocr_engine_written_in_rust/
4. robertknight/ocrs: Rust library and CLI tool for OCR (extracting text from images) - GitHub, acessado em junho 27, 2025, https://github.com/robertknight/ocrs
5. rusqlite_migration - Rust - Docs.rs, acessado em junho 27, 2025, https://docs.rs/rusqlite_migration
6. cljoly/rusqlite_migration: ↕️ Simple database schema migration library for rusqlite, written with performance in mind. - GitHub, acessado em junho 27, 2025, https://github.com/cljoly/rusqlite_migration
7. Create your encrypted database with SQLCipher and sqlx in Rust (for Windows) - Medium, acessado em junho 27, 2025, https://medium.com/@lemalcs/create-your-encrypted-database-with-sqlcipher-and-sqlx-in-rust-for-windows-4d25a7e9f5b4
8. SQLite FTS5 Extension, acessado em junho 27, 2025, https://www.sqlite.org/fts5.html
9. Using sqlite-vec in Rust - Alex Garcia, acessado em junho 27, 2025, https://alexgarcia.xyz/sqlite-vec/rust.html
10. Tauri (software framework) - Wikipedia, acessado em junho 27, 2025, https://en.wikipedia.org/wiki/Tauri_(software_framework)
11. Tauri 2.0 | Tauri, acessado em junho 27, 2025, https://v2.tauri.app/
12. rusqlite/rusqlite: Ergonomic bindings to SQLite for Rust - GitHub, acessado em junho 27, 2025, https://github.com/rusqlite/rusqlite
13. Building a Rust-Powered OCR Tool for Image Text Extraction and Link Detection -

Medium, acessado em junho 27, 2025, https://medium.com/@siddheshmhatrecodes1808/building-a-rust-powered-ocr-tool-for-image-text-extraction-and-link-detection-504bd705d76e
14. Releases | Tauri v1, acessado em junho 27, 2025, https://v1.tauri.app/releases/
15. Key in rdev - Rust - Docs.rs, acessado em junho 27, 2025, https://docs.rs/rdev/latest/rdev/enum.Key.html
16. asg017/sqlite-vss: A SQLite extension for efficient vector search, based on Faiss! - GitHub, acessado em junho 27, 2025, https://github.com/asg017/sqlite-vss
17. tauri-apps/tauri-action: Build your Web application as a Tauri binary for macOS, Linux and Windows - GitHub, acessado em junho 27, 2025, https://github.com/tauri-apps/tauri-action
18. Code Signing macOS Applications | Tauri v1, acessado em junho 27, 2025, https://tauri.app/v1/guides/distribution/sign-macos
19. tauri-docs-wip/src/distributing/macos.md at main - GitHub, acessado em junho 27, 2025, https://github.com/JonasKruckenberg/tauri-docs-wip/blob/main/src/distributing/macos.md
20. Windows Code Signing - Tauri 2.0, acessado em junho 27, 2025, https://tauri.app/distribute/sign/windows/
21. Windows - The Tauri Documentation WIP, acessado em junho 27, 2025, https://jonaskruckenberg.github.io/tauri-docs-wip/distributing/windows.html
22. Rust code coverage without 3rd party utilities - Eugene Babichenko, acessado em junho 27, 2025, https://eugene-babichenko.github.io/blog/rust-code-coverage-without-3rd-party-utilities/
23. Salary: Rust Developer in Toronto, Ontario (June, 2025) - ZipRecruiter, acessado em junho 27, 2025, https://www.ziprecruiter.com/Salaries/Rust-Developer-Salary-in-Toronto,ON
24. Rust Developer Salary in Canada 2025 - Jobicy, acessado em junho 27, 2025, https://jobicy.com/salaries/ca/rust-developer
25. Salary: Rust Developer in Ontario (June, 2025) - ZipRecruiter, acessado em junho 27, 2025, https://www.ziprecruiter.com/Salaries/Rust-Developer-Salary--in-Ontario
26. developer.apple.com, acessado em junho 27, 2025, https://developer.apple.com/help/account/membership/program-enrollment/
27. Membership Details - Apple Developer Program, acessado em junho 27, 2025, https://developer.apple.com/programs/whats-included/
28. EV Code Signing Certificates - Sign Windows Drivers & Packages - SignMyCode, acessado em junho 27, 2025, https://signmycode.com/ev-code-signing
29. EV Code Signing Certificates - SSL Shopper, acessado em junho 27, 2025, https://www.sslshopper.com/ev-code-signing-certificates.html