Blinker Reader

Blink. Open. Read.

Blinker Reader is a fast, secure, no-noise document reader designed to combine the instant startup and light footprint of SumatraPDF with the structured library management of Calibre — without their drawbacks.
It is built for users who want a truly lightweight, privacy-respecting, keyboard-friendly reader that works the same across Windows, macOS and Linux, without telemetry, bloat, or outdated UI patterns.

Key value points

- Instant startup, minimal RAM usage, no background daemons
- Secure by design: sandboxed rendering, no scripting in documents, no network by default
- Modern UI/UX: clean, keyboard-first, dark/light themes, no clutter
- Local library indexing with tags, metadata and fast search
- Offline-first, file-system-first, no lock-in, no cloud requirement
- Plugin system based on sandboxed WASM, not unsafe Python/Java plugins

Core features (MVP)

- Open and read PDF, EPUB, CBZ/CBR, TXT, Markdown
- Library indexing with metadata and tags
- Full-text search on titles/authors/tags + intra-document search
- Portable builds for Windows, macOS, Linux
- Zero telemetry, no outbound connections, no auto-updates

Tech stack

- **Core:** Rust (secure, memory-safe, no GC)
- **Rendering:** PDFium (PDF), Rust EPUB pipeline + HTML flow, image decode for comic formats
- **UI:** Tauri + WebView (React + TypeScript frontend)
- **Database:** SQLite + FTS5
- **Extensions:** WebAssembly (WASI sandbox, capability-based)

License & compliance

- Project license: Apache-2.0 or MPL-2.0 (to be decided)
- Fully GDPR-compliant: no data collection, no tracking
- All third-party lib licensing documented in `/docs/licenses.md`

Technical Plan

Table of Contents

---

1. Scope, non-scope, target and performance budget

Scope (MVP v0.1)

- Document reader for PDF, EPUB, CBZ/CBR, TXT, Markdown
- Local library index with metadata and tag support
- Full-text search for titles/authors/tags + in-document search
- Minimal, keyboard-first UI with dark/light themes
- Runs as a portable executable on all three desktop platforms
- No telemetry, no network usage by default

Non-scope (MVP)

- Format conversion (Calibre-style)
- Built-in store, remote catalogues, DRM
- Cloud sync (will be optional in v1.x)

Supported platforms

- Windows 10+ x64
- macOS 13+ (Intel & Apple Silicon)
- Linux x64 (glibc)
  Mobile may be considered later.

Performance targets

Metric	Target
Cold startup	< 300 ms (NVMe) / < 700 ms (SATA)
Idle memory usage	< 120 MB library of ±1k documents
Render PDF page 1080p	< 16 ms avg after warm-up
Library indexing 1k items	< 90 s incremental

---

2. High-level architecture

Layer	Technology	Notes
UI layer	Tauri + React/TS + WebView	Native shell, WebView rendering, no Electron bloat
Core logic	Rust crates (`blinker-core-*`)	Fully native, safe, testable
PDF rendering	PDFium via Rust bindings	Sandboxed, JS disabled
EPUB rendering	Rust HTML/CSS flow + optional epub.js fallback	Layout control via CSS
Database	SQLite + FTS5	Embedded, robust, no server
Security	OS sandbox + content sanitization	No network, no exec
Plugins	WASM/WASI	Capability-restricted extensions

---

3. Security model and threat mitigation

Threats addressed

- Malicious PDFs (JavaScript, embedded files, launch actions)

- Path traversal in archived comic formats
- EPUB with remote resources or scripts
- Arbitrary code execution through plugins
- Supply chain attacks

Mitigations

- PDFium built **without JavaScript**, with risky actions disabled
- No automatic extraction of embedded files
- EPUB content sanitized; remote URLs blocked
- CBZ/CBR unpacking done in-memory with strict path normalization
- WASM-only plugin system (no Python/Java exec)
- OS-level sandbox: AppContainer (Win), AppSandbox (macOS), seccomp-bpf (Linux)
- No outbound network traffic unless explicitly enabled
- `cargo-audit`, `cargo-deny`, locked dependencies

---

4. Data model and persistence

- SQLite schema with:
  - `library_item` (file, hash, metadata, timestamps)
  - `tag` & `item_tag`
  - `reading_state`
  - `annotation`
- FTS5 used for metadata search (full-text extraction of book contents comes later)
- File-system-first: files are not moved or duplicated; DB only indexes them
- Hash-based deduplication (BLAKE3)

---

5. Functional requirements

Reading

- Open document directly or via library entry
- Table of contents navigation
- Search inside document
- Page layouts: single, dual, continuous scroll
- Keyboard shortcuts (Vim mode optional)

Library

- Import by folder, drag & drop
- Automatic metadata parsing
- Editable metadata and cover
- Filter by type, tag, author, full-text

Annotations

- Highlights, notes, bookmarks
- Export to JSON/Markdown
- Future: write-back to PDF annotations where possible

File-system behaviour

- No lock-in: files stay where they are
- Library updates tracked via filesystem watcher

---

6. Non-functional requirements

- Startup time and memory budget enforced in CI
- 60 fps UI interaction, GPU-accelerated scrolling
- Accessibility: contrast, keyboard nav, ARIA roles
- Internationalization: English + Italian at launch

---

7. Plugin system (post-MVP)

- Extension format: `.blinker-plugin`
- Containing WASM module + TOML manifest
- Capability-based API (read-only annotations, write sandbox file, etc.)
- No filesystem or network access unless granted

---

8. Build, packaging, CI/CD

- Rust stable + Node 20
- Vendored PDFium binaries or CI-built artefacts
- Packaging via Tauri Bundler:
  - Windows `.msi` with Authenticode signature
  - macOS `.dmg` + notarization
  - Linux AppImage + `.deb`
- GitHub Actions:
  - Build matrix (Win/macOS/Linux)
  - Security audit jobs
  - Release artifacts on semantic tags

---

9. UX requirements and flows

- **Home:** search bar, filters, grid/list toggle
- **Book detail:** metadata, tags, “Open” button, recent activity
- **Reader view:** minimal header, optional TOC sidebar, inline search, progress bar, command palette (`Ctrl+K`)
- Zero onboarding: defaults should be sensible

---

10. Licensing & compliance

- Project: Apache-2.0 or MPL-2.0
- PDFium: BSD, NOTICE retained
- All third-party licenses tracked in `docs/licenses.md`
- No telemetry → auto-compliant with GDPR

---

11. Roadmap

Milestone	Focus	Estimate
0.1 – Reader Base	PDF rendering, basic UI, library index, packaging	2–4 weeks
0.2 – EPUB & Library	EPUB flow, metadata editing, file watcher	+3 weeks
0.3 – Annotations	Highlights, notes, export, theming	+3 weeks
0.4 – Polish & Security	Sandbox, accessibility, i18n, fuzzing	+2 weeks
1.0 – Stable	WASM plugins, notarized apps, crash reporter	+2 weeks

---

12. Repository structure

```
blinker/
├─ apps/
│  └─ desktop/
│     ├─ src-tauri/
│     └─ ui/
├─ crates/
│  ├─ blinker-core-render/
│  ├─ blinker-core-library/
│  ├─ blinker-core-annot/
│  ├─ blinker-core-security/
│  └─ blinker-core-common/
├─ sql/
├─ scripts/
├─ .github/workflows/
└─ docs/
```

---

13. IPC API overview (no code)

- `library.scan(paths[]) -> ScanReport`
- `library.query(filters) -> Item[]`
- `library.updateMeta(id, fields)`
- `reader.open(id) -> Session`
- `reader.search(session, query) -> Matches[]`
- `annot.add(itemId, range, kind, text, color)`
- `annot.list(itemId) -> Annotation[]`

---

14. Test strategy

- Unit tests: EPUB parsing, DB operations, metadata
- Integration tests: import flow, deduplication, annotation round-trip
- Rendering snapshot tests for deterministic PDF page bitmaps
- Fuzzing: malformed PDF/EPUB/CBZ input
- Performance: startup gate in CI
- Security: crafted archive path traversal, malicious PDF actions

---

15. Risks & mitigation

Risk	Mitigation
PDFium multi-platform binaries	Cached builds, automated fetching
EPUB layout edge cases	Fallback to web-based flow render
UI startup regressions	CI perf gate
OS sandbox differences	Per-platform tests + feature flags

---

16. Initial development tasks

1. Bootstrap Rust + Tauri workspace
2. Integrate PDFium + validate first page render
3. Implement SQLite schema and migration scripts
4. Library scanner with hashing + dedup
5. Basic UI shell (Home + Reader placeholder)
6. Packaging for 3 OS targets

17. Definition of Done (per milestone)

- ✅ Green tests
- ✅ Build artifacts for Windows/macOS/Linux
- ✅ Startup time within budget
- ✅ PDF + EPUB baseline working
- ✅ No network calls detected
- ✅ Licenses documented

---

18. Next steps

- Initialize repository with folder layout above
- Commit base workspace + placeholder crates
- Add GitHub Actions with CI matrix
- Add issue board with milestones 0.1 → 1.0
- Begin with **Milestone 0.1 – Reader Base**

End of document.