# LazyNote 项目架构蓝图与战略实施报告

## 1. 执行摘要：无摩擦认知的哲学重构

在当代知识工作者的数字生态系统中，生产力工具的选择往往陷入一种两难的境地：我们不得不在功能强大但操作繁琐的工具（如 Obsidian、Jira）与设计精美但逻辑僵化、资源占用高昂的工具（如 Notion、Apple Notes）之间做出妥协。现有的解决方案往往要求用户去适应软件的逻辑，而非让软件适应用户的思维流速。"LazyNote" 项目的构想并非仅仅是为了开发一款新的应用程序，而是对当前生产力软件工业体系的一种反驳。它主张效率的提升不应依赖于功能的堆砌，而应源于对交互摩擦的极致消除。

"LazyNote" 的核心理念在于“惰性”（Lazy），这并非指用户的懒惰，而是指系统的主动性——系统应当预判意图，自动处理繁杂的元数据，让用户仅需专注于核心的创造性工作。该应用旨在构建一个跨越时间（日程管理）与空间（知识库）的统一体，通过一个极简的统一入口，实现对这两类数据的无缝操作。本报告详细阐述了 LazyNote 的架构设计、技术选型及实施路径，旨在构建一个跨平台（Windows, iOS, Android）、本地优先（Local-First）、隐私至上的生产力生态系统，它将融合 Obsidian 的双向链接深度、Notion 的现代化审美以及类似 Raycast 的即时交互体验，并在底层架构上彻底解决资源占用与即时同步的工程难题。

LazyNote 的架构设计摒弃了传统应用将“笔记”与“任务”视为异构数据的做法，转而采用一种基于“原子化数据”（Atomic Data）的统一模型。在这个模型中，笔记、任务、日程事件不再是割裂的实体，而是同一数据的不同视图投影。这种设计哲学直接响应了用户对于“笔记 section 与 task calendar 足够联动”的核心需求。同时，为了满足“Apple 设计语言”的审美要求，系统将在 Windows 和 Android 等非原生平台上，通过底层的渲染引擎优化，复刻出具有磨砂玻璃质感（Liquid Glass）、流畅物理动画及极简排版的高保真 UI，确保三端体验的高度一致性与原生感。

## 2. 核心架构：本地优先的数据范式与同步策略

当前主流生产力工具（如 Notion, Trello）的一个根本性架构缺陷在于过度依赖云端作为单一事实来源（Source of Truth）。这种架构不仅引入了网络延迟——这是“即时记录”的天敌——而且在离线环境下功能大打折扣 1。为了实现 LazyNote 所追求的极致响应速度与数据主权，必须采用“本地优先”（Local-First）的架构范式 2。

### 2.1 统一数据模型：原子化设计

为了打破笔记与任务之间的壁垒，LazyNote 在数据库层面摒弃了传统的异构表设计。在我们的架构中，所有内容的基本单位被称为“原子”（Atom）。一个原子本质上是 SQLite 数据库中的一行记录，它通过元数据的多态性（Polymorphism）来决定其表现形式。一个原子可以同时是一条笔记、一个待办事项、和一个日历事件。这种多态性允许数据在不同状态间流转而无需进行昂贵的迁移操作：会议期间记录的一段纯文本笔记，可以通过添加一个时间戳属性，瞬间“变身”为日程表上的一个事件，或者通过添加状态标记转化为待办清单中的一项 3。

#### 2.1.1 数据库模式设计与 SQLite 优化

SQLite 被选定为 LazyNote 的存储引擎，这不仅因为其在所有目标平台（Windows, Android, iOS）上的原生支持，更因为其单文件特性极大地简化了数据的备份与迁移 4。为了支撑数万条笔记的毫秒级查询与双向链接，数据库模式必须经过精心设计。

atoms 表是系统的核心，其设计如下：

| **字段名** | **类型** | **用途与说明** |
| --- | --- | --- |
| uuid | TEXT (UUIDv4) | 唯一标识符，用于在分布式系统中确保引用的稳定性，避免自增 ID 在多端同步时产生冲突 5。 |
| type | TEXT | 区分原子类型：'note', 'task', 'event', 'journal'。支持复合类型查询。 |
| content | TEXT | 存储 Markdown 格式的原始内容。支持内联元数据解析。 |
| content\_vector | BLOB | (预留) 用于 sqlite-vec 向量搜索的嵌入数据，支持未来的语义搜索功能 6。 |
| task\_status | TEXT | 任务状态机：'todo', 'in\_progress', 'done', 'archived'。仅当 type 含 task 时有效。 |
| event\_start | INTEGER | UNIX 时间戳，日程开始时间。支持精确到秒的调度。 |
| event\_end | INTEGER | UNIX 时间戳，日程结束时间。 |
| hlc\_timestamp | TEXT | 混合逻辑时钟（Hybrid Logical Clock）时间戳，用于 CRDT 冲突解决 7。 |
| is\_deleted | INTEGER | 软删除标记（Tombstone），确保同步时删除操作能正确传播。 |

为了实现类似 Obsidian 的双向链接功能，且避免每次启动时全量解析 Markdown 文件的性能开销，我们设计了一个独立的 links 邻接表。该表通过数据库触发器（Triggers）自动维护：每当 atoms 表中的 content 字段发生更新，触发器会调用自定义的 Tokenizer 函数解析文本中的 ] 语法，并实时更新 links 表 5。这种设计将链接解析的计算成本分摊到了写入时刻，从而保证了读取和图形视图（Graph View）渲染时的绝对高性能。

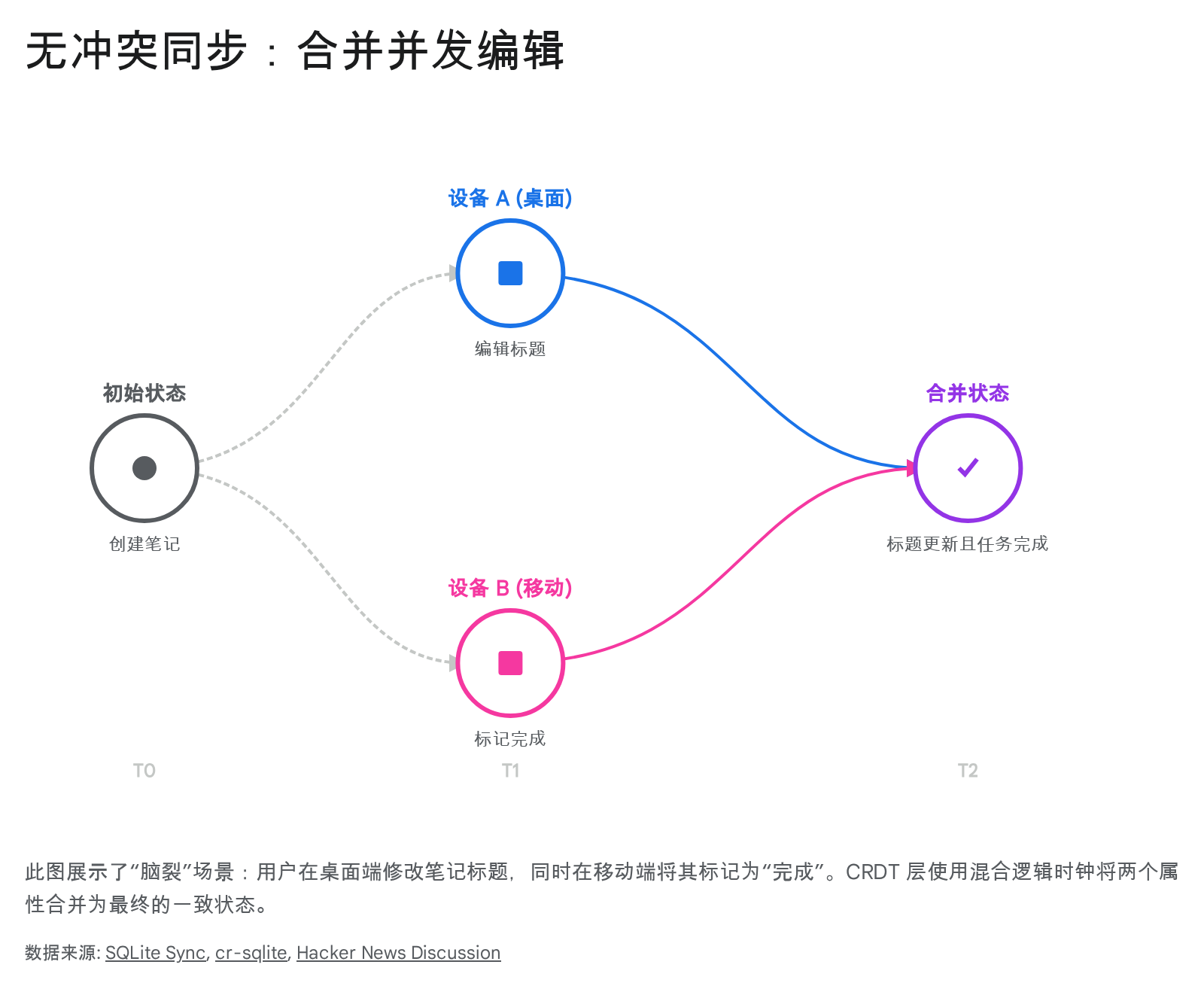
### 2.2 无冲突同步引擎：CRDT 与端到端加密

用户对“Apple 风格”的追求隐含了对体验流畅性的极高要求。这意味着同步必须是隐形的，用户绝不应看到“版本冲突，请选择保留版本”的技术性对话框。为了实现这一目标，LazyNote 将集成 CRDT（无冲突复制数据类型）层，具体方案倾向于采用 cr-sqlite 或基于混合逻辑时钟（HLC）的自定义实现 7。

#### 2.2.1 同步机制详解

LazyNote 的同步架构设计为“哑管道”（Dumb Pipe）模型，这是实现隐私至上（Privacy-First）的关键。

1. **客户端逻辑（Flutter/Rust FFI）**：应用程序内部运行一个同步守护进程。该进程监听 SQLite 的预写日志（WAL）或利用数据库钩子捕获变更。一旦检测到数据修改，系统将生成一个包含 HLC 时间戳、列 ID 和新值的二进制增量包（Delta Packet）9。
2. **传输层**：这些增量包在发送前会在本地进行 AES-256 加密。这意味着同步服务器（Relay Server）只负责转发加密的二进制流，无法解密查看用户数据，从而实现了真正的端到端加密（E2EE）10。
3. **冲突解决**：当客户端接收到来自其他设备的增量包时，CRDT 算法会根据 HLC 时间戳自动合并数据。由于合并发生在字段级别（Column-level），即使用户 A 在 Windows 上修改了笔记的标题，而用户 B 同时在 iOS 上修改了该笔记的正文，两个修改也能完美合并，互不覆盖。



通过这种机制，LazyNote 实现了数据的“最终一致性”（Eventual Consistency），无论设备是在线还是离线，用户都可以随时进行写入操作，系统会在网络恢复后自动完成状态收敛。

## 3. 模块一：极简笔记引擎（认知层）

用户明确指出笔记模块需“比 Obsidian 更符合使用习惯，比 Notion 更好看，同时避免 Obsidian 的繁杂及 Notion 的高占用”。这要求我们在工程实现上在“纯文本的轻量”与“富文本的表现力”之间找到完美的平衡点。

### 3.1 编辑器工程：混合渲染架构

Obsidian 的核心优势在于 Markdown 的通用性与本地存储，但其编辑体验往往带有“代码感”；Notion 提供了所见即所得的块（Block）编辑体验，但其基于 Web 技术栈（Electron）的架构导致了内存占用过高（通常 500MB+）和启动缓慢 1。LazyNote 将采用**基于 Flutter 的原生渲染引擎**来构建编辑器。

**技术选型与实现细节：** 我们不使用 WebView 来渲染编辑器，而是利用 Flutter 的 Canvas 直接绘制文本。这种方案（类似 super\_editor 或 flutter\_quill 的底层实现）具有显著的性能优势：内存占用可控制在 100MB 以内，且渲染帧率可稳定在 60fps 以上 13。

编辑器将采用“隐形 Markdown”模式。用户输入 Markdown 语法（如 \*\*加粗\*\* 或 # 标题）后，编辑器会即时将其渲染为富文本样式，并隐藏语法标记，仅在光标置于该文本段落时才显露标记。这种设计保留了 Markdown 的高效输入流，同时提供了 Notion 级的视觉整洁度。为了满足“比 Notion 更好看”的需求，排版引擎将内置 Apple 风格的排版规则：严格的行高控制（Line-height 1.6）、字间距微调以及对 San Francisco (iOS) / Segoe UI Variable (Windows) 字体特性的深度支持 14。

### 3.2 组织结构：反文件夹系统

文件夹结构强制用户在创作前进行分类决策（“这个笔记该放哪里？”），这增加了认知摩擦。为了贯彻“Lazy”的理念，LazyNote 采用**时序与关联**主导的组织方式。

1. **每日流（Daily Stream）**：这是应用的主视图。默认情况下，所有新创建的原子都归属于“今天”。这模仿了人类记忆的时序特性，使得记录变得毫无压力。
2. **双向链接图谱**：通过 ] 语法，笔记之间建立有机的语义关联。与 Obsidian 不同，LazyNote 不会默认展示复杂的全局关系图，而是通过侧边栏的“相关原子”面板，智能推荐与当前笔记存在强关联的内容。
3. **智能集合（Smart Collections）**：取代传统文件夹的是基于 SQL 查询的动态视图。例如，用户可以创建一个名为“Project Alpha”的视图，其定义为 SELECT \* FROM atoms WHERE content LIKE '%#project-alpha%' AND status!= 'archived'。这种动态分类确保了笔记永远处于最新的上下文中。

## 4. 模块二：时空融合的任务引擎（行动层）

此模块旨在成为“升级版 Notion Calendar”并整合“滴答清单”式的任务管理。其核心创新在于将**时间**视为笔记的一个原生维度，而非外挂属性。

### 4.1 时间阻塞（Time-Blocking）交互界面

传统的任务列表往往缺乏时间感，而日历应用又缺乏任务的上下文。LazyNote 将两者融合在同一时间轴上。界面设计放弃了传统的月视图网格，转而采用类似 Notion Calendar 或 Cron 的**无限垂直时间轴**设计。

**交互逻辑：**

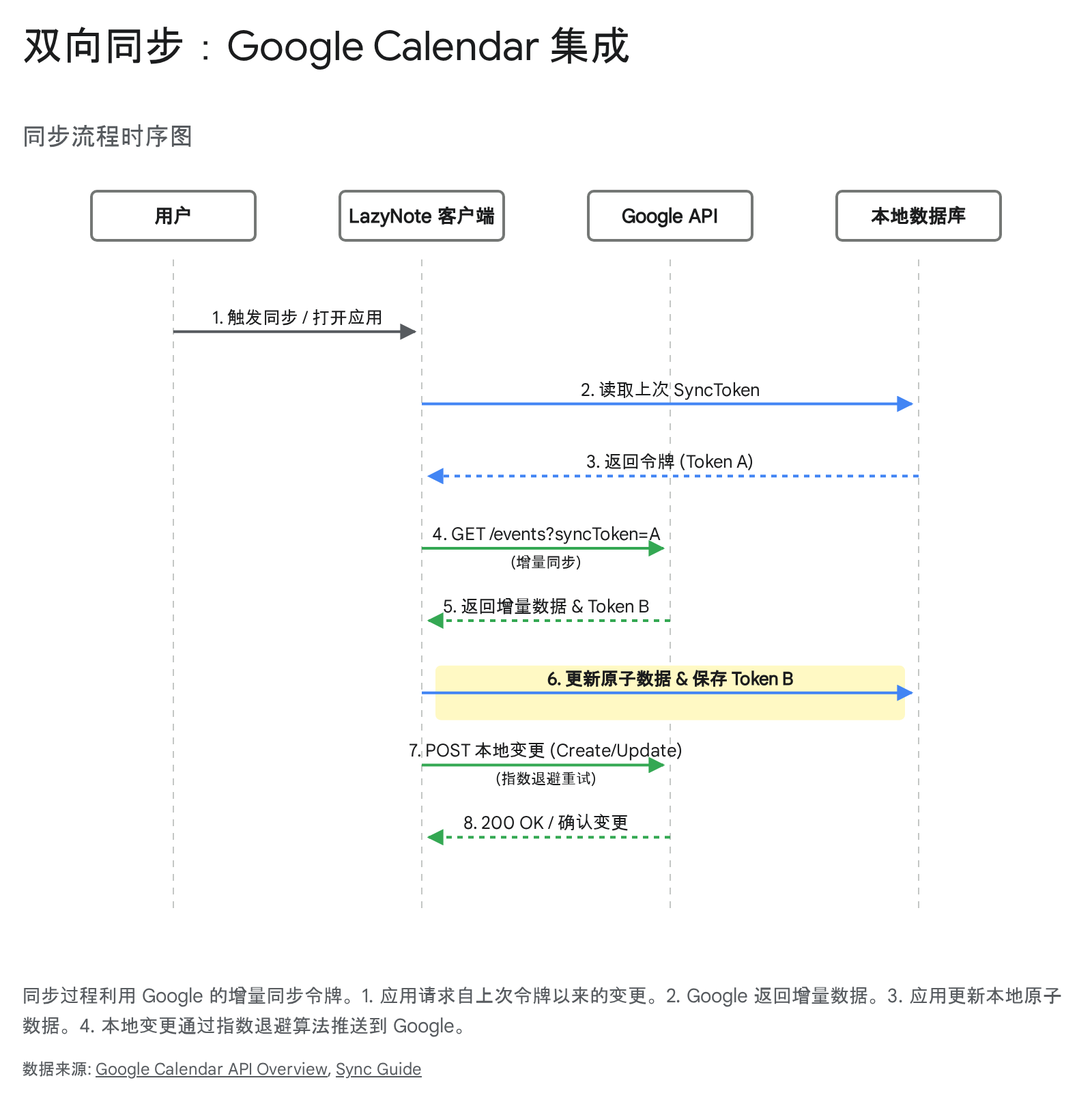
界面左侧为时间刻度，中间为日程泳道，右侧为“停靠区”（Dock）。停靠区存放着所有未排期的任务原子（即那些仅有 TODO 状态但无 event\_start 时间戳的原子）。用户可以通过拖拽操作，将停靠区的任务直接拉入时间轴的特定时间段。这一动作在数据库层面原子性地更新了该记录的 event\_start 和 event\_end 字段，瞬间将“任务”转化为“日程”。

### 4.2 Google Calendar 双向同步机制

为了成为用户日程管理的“第一入口”，与 Google Calendar 的完美同步是必须跨越的技术门槛。这不仅是数据的展示，更是状态的深度绑定 16。

**同步架构设计：**

1. **认证与鉴权**：通过 Flutter 的 google\_sign\_in 插件实现 OAuth 2.0 流程，获取 calendar.events 读写权限 18。
2. **增量同步策略**：利用 Google Calendar API 的 syncToken 机制。首次全量同步后，系统会保存服务器返回的 syncToken。后续同步请求仅需携带此 token，Google 服务器即返回自上次同步以来的增量变更（Created, Updated, Deleted）。这极大地减少了网络流量和 API 调用次数，规避了配额限制 18。
3. **影子原子（Shadow Atoms）**：系统不会直接“显示”远程数据，而是将 GCal 事件映射为本地数据库中的“影子原子”。这些原子具有特殊的 source: 'gcal' 标记。这种设计允许用户对 GCal 事件进行本地化的扩展——例如，用户可以在 LazyNote 中为一个 GCal 会议事件添加详细的会议纪要（作为该原子的 content），这些富文本内容存储在本地，而时间、标题等核心属性则与 GCal 保持双向同步。



通过这种机制，用户可以在 LazyNote 中直接修改会议时间，变更会实时推送到 Google Calendar；反之，同事在 GCal 上的修改也会即时反映在 LazyNote 中。

## 5. 模块三：统一接入窗口（交互层）

这是 LazyNote 的灵魂功能——一个无处不在的“全知之眼”。它借鉴了 Raycast 和 macOS Spotlight 的设计模式，旨在成为用户与数字大脑交互的第一触点 19。

### 5.1 界面设计与交互流

**Windows 端实现：** 为了在 Windows 上复刻 Apple Spotlight 的视觉体验，我们需要突破原生窗口的限制。通过使用 Flutter 的 window\_manager 和 flutter\_acrylic 库，我们可以创建一个无边框、背景模糊（Acrylic/Mica 效果）、始终置顶的浮动窗口 20。该窗口通过全局热键（如 Alt + Space）瞬间唤醒。设计上，它将摒弃所有多余的 UI 元素，只保留一个带有微妙光标动画的输入框，背景采用高斯模糊处理，以显示下方桌面内容的轮廓，营造出一种“悬浮于系统之上”的轻盈感 21。

**移动端实现（iOS/Android）：** 移动端的“统一入口”体现为锁屏小组件（Lock Screen Widget）和主屏快速捕获（Quick Capture）入口。在 iOS 18+ 上，我们将利用 WidgetKit 开发交互式小组件，允许用户在不解锁手机的情况下查看今日首要任务或快速录入灵感 22。Android 端则利用 App Shortcuts 和桌面微件，提供一键直达输入框的能力。点击入口后，应用将以“底单模式”（Bottom Sheet）弹出键盘，背景保持半透明，给用户一种未离开当前环境的心理暗示。

### 5.2 NLP 驱动的意图识别引擎

“统一入口”不仅仅是一个搜索框，它是一个基于自然语言处理（NLP）的命令中枢。用户输入的不仅是关键词，更是结构化的指令。

为了在本地实现低延迟（<50ms）的意图识别，我们将在客户端集成轻量级的 NLP 库（如 Rust 的 lingua-rs 或针对 Dart 优化的正则表达式引擎），而非依赖云端 API 24。

**解析流水线（Parsing Pipeline）：**

1. **输入**：“明天下午三点和 Sarah 开会 #work”
2. **分词与实体提取**：引擎识别出时间实体“明天下午三点”（解析为 CurrentDate + 1 day, 15:00），人名实体“Sarah”（关联到通讯录或已有笔记链接 ]），以及标签实体“#work”。
3. **意图分类**：根据关键词（“开会”）和语法结构，系统判定此输入大概率为“日程事件”（Event）。
4. **实时反馈**：在用户输入的同时，界面下方会实时渲染出预览卡片：“创建日程：和 Sarah 开会，时间：10月28日 15:00，标签：工作”。
5. **执行**：用户按下回车，数据直接写入 SQLite 数据库，并触发后台的 GCal 同步任务。

这一过程无需任何鼠标点击或模式切换，完美诠释了“Lazy”的核心——让软件承担认知的重负。

## 6. 跨平台工程战略：三端同构

用户要求架构从一开始就做到 Windows、iOS、Android 三端兼容。这不仅是代码复用的问题，更是用户体验一致性的挑战。

### 6.1 Flutter 框架的战略选择

虽然 React Native 在跨平台领域也占据一席之地，但对于 LazyNote 这种对 UI 渲染精度和性能有极高要求的应用，**Flutter** 是唯一可行的选择 26。

1. **像素级控制**：Flutter 使用自绘引擎（Skia/Impeller），这意味着我们可以完全绕过系统原生的 UI 组件限制，在 Windows 和 Android 上 1:1 地绘制出 Apple 风格的模糊、阴影和圆角。这对于实现“比 Notion 更好看”的视觉目标至关重要。
2. **FFI（外部函数接口）性能**：LazyNote 的核心逻辑（数据库操作、CRDT 合并、加密）是计算密集型的。通过 Flutter 的 FFI 机制，我们可以将这部分逻辑用 **Rust** 编写（构建为动态库），并在 Dart 层直接调用。这种“UI (Dart) + Core (Rust)”的架构既保证了界面的灵活性，又确保了底层逻辑在所有平台上的高性能与安全性 28。

### 6.2 平台特性的适配

尽管追求 UI 一致性，但必须尊重平台的交互习惯，以避免“恐怖谷”效应。

* **Windows**：利用 flutter\_windows 插件集成系统托盘（System Tray），支持多窗口（Multi-window）操作——用户可以将某条笔记“弹出”为一个独立的小窗口，钉在桌面上参考 30。
* **iOS**：严格遵循人机交互指南（HIG），支持“滑动返回”（Swipe Back）手势，适配动态岛（Dynamic Island）以显示后台同步状态或计时器任务 32。
* **Android**：深度集成通知系统。对于待办事项，提供带有“完成”和“推迟”操作按钮的富媒体通知。

## 7. 隐私与安全：数据主权

在本地优先的架构下，隐私不仅是承诺，更是代码层面的保证。

### 7.1 本地存储加密

虽然 SQLite 数据库存储在本地，但设备丢失风险依然存在。LazyNote 将使用 SQLCipher 对 SQLite 数据库进行全盘加密。加密密钥由用户的主密码（Master Password）经 Argon2id 算法派生，并存储在操作系统的安全存储区（Windows DPAPI, iOS Keychain, Android Keystore）中 11。

### 7.2 隐私政策设计

作为一款“隐私至上”的应用，LazyNote 的隐私政策将基于“零知情”（Zero-Knowledge）原则：

* **零遥测**：默认关闭所有使用统计和崩溃报告。
* **无账号强制**：用户无需注册账号即可完整使用本地功能。
* **数据所有权**：用户随时可以将所有数据导出为标准的 Markdown 和 JSON 格式，确保持久的可移植性，杜绝厂商锁定（Vendor Lock-in）34。

## 8. 实施路线图：从 MVP 到 V1

为了确保项目的可交付性，我们将开发过程划分为三个阶段。

### 第一阶段：数字便签簿（The Digital Jotter）- 核心验证期

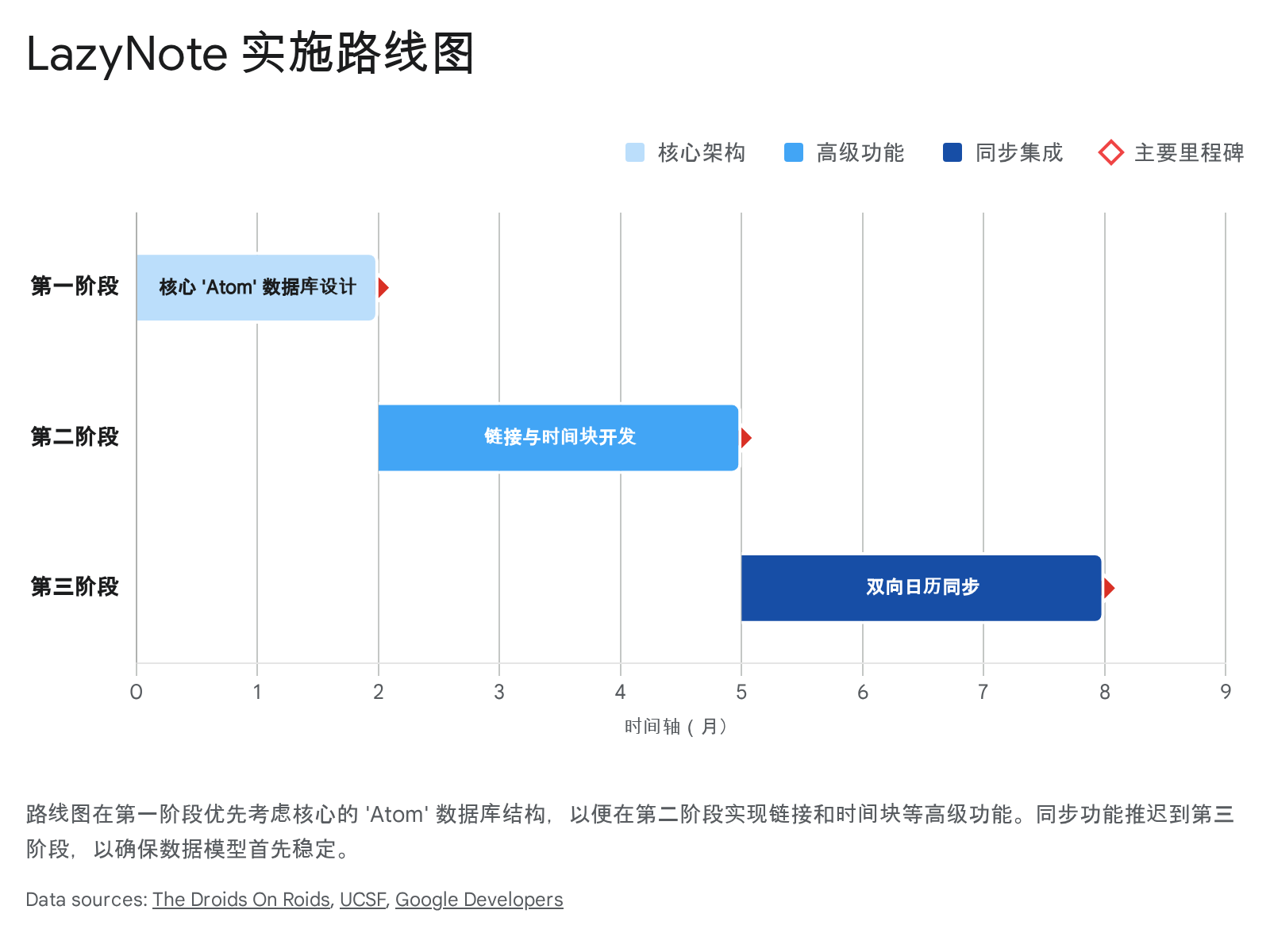
* **目标**：构建一个超越 Windows Notepad 的极简笔记应用，验证 SQLite 数据层和 Flutter 渲染性能。
* **交付物**：
  + 基于 Flutter 的 Markdown 实时渲染编辑器。
  + Windows 端无边框悬浮窗口及全局热键唤醒。
  + 基础 SQLite CRUD 功能，支持原子化数据存储。
  + 简单的标签系统。

### 第二阶段：时空链接更新（The Chrono-Link Update）- 结构成型期

* **目标**：引入时间维度和关联能力，完成“笔记”向“知识库”的进化。
* **交付物**：
  + 实现 ] 解析器与双向链接侧边栏。
  + 构建无限垂直滚动的时间轴视图（Calendar View）。
  + 集成 NLP 引擎，实现从输入框直接创建带时间的任务。
  + 实现任务与笔记的相互转化逻辑。

### 第三阶段：同步与移动端扩展（The Sync & Mobile Expansion）- 生态完善期

* **目标**：打通三端数据，实现 Google Calendar 同步，发布移动端应用。
* **交付物**：
  + 部署 Go/Rust 编写的同步中继服务器。
  + 在客户端集成 CRDT 逻辑，实现多端无冲突同步。
  + 完成 iOS 与 Android 端的 Widget 开发与发布。
  + 全功能的 Google Calendar 双向同步。



## 9. 结论

LazyNote 项目不仅是对现有生产力工具痛点的一次精准打击，更是对未来个人知识管理形态的一次探索。通过整合本地优先的 SQLite 架构、Flutter 的高性能渲染以及 Rust 驱动的智能核心，我们有能力构建出一个既具备 Apple 级精致美学，又拥有黑客级高效内核的生产力平台。它将复杂的认知过程封装在极简的界面之下，真正实现“所想即所得”的无摩擦体验。

#### 引用的著作

1. Notion vs Obsidian – All Features Compared (2026) - Productive.io, 访问时间为 二月 11, 2026， <https://productive.io/blog/notion-vs-obsidian/>
2. Local-first architecture with Expo, 访问时间为 二月 11, 2026， <https://docs.expo.dev/guides/local-first/>
3. Building a Todo App with Flutter and OpenAI's ChatGPT API | by ..., 访问时间为 二月 11, 2026， <https://medium.com/@justin.guckes/building-a-todo-app-with-flutter-and-openais-chatgpt-api-da30dc7237bd>
4. Flutter databases overview - updated 2025 - Greenrobot, 访问时间为 二月 11, 2026， <https://greenrobot.org/database/flutter-databases-overview/>
5. Using sqlite3 as a notekeeping document graph with automatic ..., 访问时间为 二月 11, 2026， <https://epilys.github.io/bibliothecula/notekeeping.html>
6. How sqlite-vec Works for Storing and Querying Vector Embeddings | by Stephen Collins, 访问时间为 二月 11, 2026， <https://medium.com/@stephenc211/how-sqlite-vec-works-for-storing-and-querying-vector-embeddings-165adeeeceea>
7. vlcn-io/cr-sqlite: Convergent, Replicated SQLite. Multi-writer and CRDT support for SQLite, 访问时间为 二月 11, 2026， <https://github.com/vlcn-io/cr-sqlite>
8. sqliteai/sqlite-sync: SQLiteSync is a local-first SQLite extension using CRDTs for seamless, conflict-free data sync and real-time collaboration across devices. - GitHub, 访问时间为 二月 11, 2026， <https://github.com/sqliteai/sqlite-sync>
9. ElectricSQL (Legacy) Vs PowerSync, 访问时间为 二月 11, 2026， <https://www.powersync.com/blog/electricsql-vs-powersync>
10. FR: End-to-End Encryption (E2EE) for sync data (Notes & Progress) · Issue #2795 - GitHub, 访问时间为 二月 11, 2026， <https://github.com/readest/readest/issues/2795>
11. 5 Steps to Implement End-to-End Encryption in Apps | Sidekick Interactive, 访问时间为 二月 11, 2026， <https://www.sidekickinteractive.com/uncategorized/5-steps-to-implement-end-to-end-encryption-in-apps/>
12. I've used Notion and Obsidian for years, which one is better? - YouTube, 访问时间为 二月 11, 2026， <https://www.youtube.com/watch?v=izMEwR8Irsc>
13. Create Android and iOS Apps with Flutter: A Complete Developer Guide - Medium, 访问时间为 二月 11, 2026， <https://medium.com/@info_80576/create-android-and-ios-apps-with-flutter-a-complete-developer-guide-29488b6a1555>
14. Comparison: Material UI vs. Fluent UI vs. SwiftUI | by Louis Marc Leonard | Medium, 访问时间为 二月 11, 2026， <https://medium.com/@llouismrc2/comparison-material-ui-vs-fluent-ui-vs-swiftui-9d2450f17102>
15. Materials | Apple Developer Documentation, 访问时间为 二月 11, 2026， <https://developer.apple.com/design/human-interface-guidelines/materials>
16. Thoughts on Ticktick vs. Notion Calendar? - Reddit, 访问时间为 二月 11, 2026， <https://www.reddit.com/r/ticktick/comments/19dga14/thoughts_on_ticktick_vs_notion_calendar/>
17. Google Calendar API overview, 访问时间为 二月 11, 2026， <https://developers.google.com/workspace/calendar/api/guides/overview>
18. Synchronize resources efficiently | Google Calendar | Google for ..., 访问时间为 二月 11, 2026， <https://developers.google.com/workspace/calendar/api/guides/sync>
19. Command Palette Interfaces - Philip Davis, 访问时间为 二月 11, 2026， <https://philipcdavis.com/writing/command-palette-interfaces>
20. How to minimize my desktop Flutter app to system tray? - Stack Overflow, 访问时间为 二月 11, 2026， <https://stackoverflow.com/questions/56619797/how-to-minimize-my-desktop-flutter-app-to-system-tray>
21. Multi Blog – Nailing the Activation Behavior of a Spotlight / Raycast ..., 访问时间为 二月 11, 2026， <https://multi.app/blog/nailing-the-activation-behavior-of-a-spotlight-raycast-like-command-palette>
22. How to Build Home Screen Widgets for iOS and Android with Flutter - Walturn, 访问时间为 二月 11, 2026， <https://www.walturn.com/insights/how-to-build-home-screen-widgets-for-ios-and-android-with-flutter>
23. iOS Lockscreen Widgets with Flutter and home\_widget | by Anton Borries | Medium, 访问时间为 二月 11, 2026， <https://medium.com/@ABausG/ios-lockscreen-widgets-with-flutter-and-home-widget-0dfecc18cfa0>
24. treebender - Rust - Docs.rs, 访问时间为 二月 11, 2026， <https://docs.rs/treebender>
25. pemistahl/lingua-rs: The most accurate natural language detection library for Rust, suitable for short text and mixed-language text - GitHub, 访问时间为 二月 11, 2026， <https://github.com/pemistahl/lingua-rs>
26. Flutter vs React Native: Complete 2025 Framework Comparison Guide | Blog, 访问时间为 二月 11, 2026， <https://www.thedroidsonroids.com/blog/flutter-vs-react-native-comparison>
27. Flutter vs. React Native in 2025 — Detailed Analysis - Nomtek, 访问时间为 二月 11, 2026， <https://www.nomtek.com/blog/flutter-vs-react-native>
28. Tauri 2.0 | Tauri, 访问时间为 二月 11, 2026， <https://v2.tauri.app/>
29. we are rewriting SQLite! - Glauber Costa (Small Data SF 2025) - YouTube, 访问时间为 二月 11, 2026， <https://www.youtube.com/watch?v=vRIUbvKmoCg>
30. Flutter Desktop: System Tray & Menus - Vibe Studio, 访问时间为 二月 11, 2026， <https://vibe-studio.ai/insights/flutter-desktop-system-tray-menus>
31. Added support for Flutter Desktop Multi Window - window\_manager\_plus : r/FlutterDev, 访问时间为 二月 11, 2026， <https://www.reddit.com/r/FlutterDev/comments/1fzlvti/added_support_for_flutter_desktop_multi_window/>
32. iOS vs. Android App UI Design: The Complete Guide, 访问时间为 二月 11, 2026， <https://www.learnui.design/blog/ios-vs-android-app-ui-design-complete-guide.html>
33. I Built a Privacy-First Note-Taking App with Flutter — Here's What I Learned, 访问时间为 二月 11, 2026， <https://dev.to/pranta/i-built-a-privacy-first-note-taking-app-with-flutter-heres-what-i-learned-3nd1>
34. Local-First Development Tools - Privacy & Productivity, 访问时间为 二月 11, 2026， <https://super-productivity.com/guides/privacy-productivity/>