Z

Zettelkasten Inbox のしくみ

— 15人の担当者に分けて読む —

zettelkasten-inbox-ak.akuramochi.chatgpt.site

Notionに溜まった読みかけのメモを、迷わず「使える知識」の棚に仕舞うためのWebアプリ

この資料の読み方

プログラムを書かない人が、「中で何が、どの順番で、どう動いているか」を自分の言葉で説明できるようになることを目標にしています。用語は3章の「用語ミニ辞典」で先にすべて説明してから使います。本文中で このように 点線の付いた言葉は、その辞典に載っている言葉です。

4章では、このアプリを**15人の担当者が並んだ工場のライン**と見立てて、1人1ページで説明します。各ページには**流れの図**（誰から何が来て、どこへ何が出るか）と、**左＝やさしい言い方／右＝同じことを指す技術用語**を並べた表が入っています。右側は、あとで自分で調べるための手がかりです。そして——**この15人の中に、AIを使う人は1人もいません。**その意味は6章で説明します。

対象システム: Zettelkasten Inbox（個人用 知識整理Webアプリ）

作成日: 2026年8月10日 ／ 全28ページ ／ 想定読者: 非エンジニア（大学1年生程度の予備知識）

これは何をするアプリか

ツェッテルカステン（Zettelkasten）は、ドイツの社会学者ルーマンが使っていた紙のカード式ノート術です。読んだものをそのまま溜めるのではなく、**自分の言葉に直して、既にある考えのどれかに結びつけて**初めて保管する、という決まりが核にあります。このアプリは、その決まりを守る作業を画面で助ける道具です。

3種類のノートの関係

Literature Note（文献ノート）	読んだ記事・本・Xの投稿などの控え。**入り口**。Notionに自動でどんどん溜まっていく。
Tag（観点）	「どういう切り口の話か」を表す見出し。棚そのもの。親子関係を持てる。
ABC Note（永久保存ノート）	Tagという棚の中に置かれた、**自分の考えを育てていく1枚**。ここに文献ノートを追記していくことで、考えが太っていく。

問題は、入り口の Literature Note だけが溜まり続けることです。行き先を決める作業は面倒で、後回しになります。このアプリは**「行き先が決まっていないノート」だけを取り出して1件ずつ目の前に置き、Tag → ABC Note の順に決めさせ、決まったらNotionに書き戻す**という、その1点のためだけに作られています。

使う人の流れ（受信箱の画面）

1. 左側に、**まだ行き先が決まっていないノートの列**が並ぶ。上から1件選ぶ。
2. 中央にそのノートの本文が出る。アプリが4文だけ抜き出した**要点**も一緒に出る。Xの投稿ならその投稿がそのまま表示される。
3. **ステップ01**：観点となるTagを選ぶ。おすすめが上位10件、最初から並んでいる。
4. **ステップ02**：選んだTagの下にあるABC Noteを選ぶ（最大9件が並ぶ）。適当なものが無ければ、その場で新しいABC Noteを作ることもできる。
5. **自分の言葉でコメントを書く。**ここは省略できない。空欄では保存ボタンが押せない。
6. 保存すると、Notionの3か所が同時に書き換わる。列から1件消え、次のノートが出てくる。

> **この設計から読み取れること**
>
> - **コメント欄が必須**なのは、ツェッテルカステンの「自分の言葉に直してから仕舞う」という決まりを、アプリの側から強制しているためです。
> - **Tagを先、ABC Noteを後**という順番も固定です。先に棚を決めれば、次に選ぶ候補が数十件から数件に減る。人が迷う量を減らすための順番です。
> - もう1つの画面「**全体マップ**」は、書き込みが一切できない読むだけの画面で、「棚はあるのに中身が無い」「関係は付いているのに本文には一度も出てこない」といった**取りこぼし**を数えて見せます。

全体像を1枚の図で

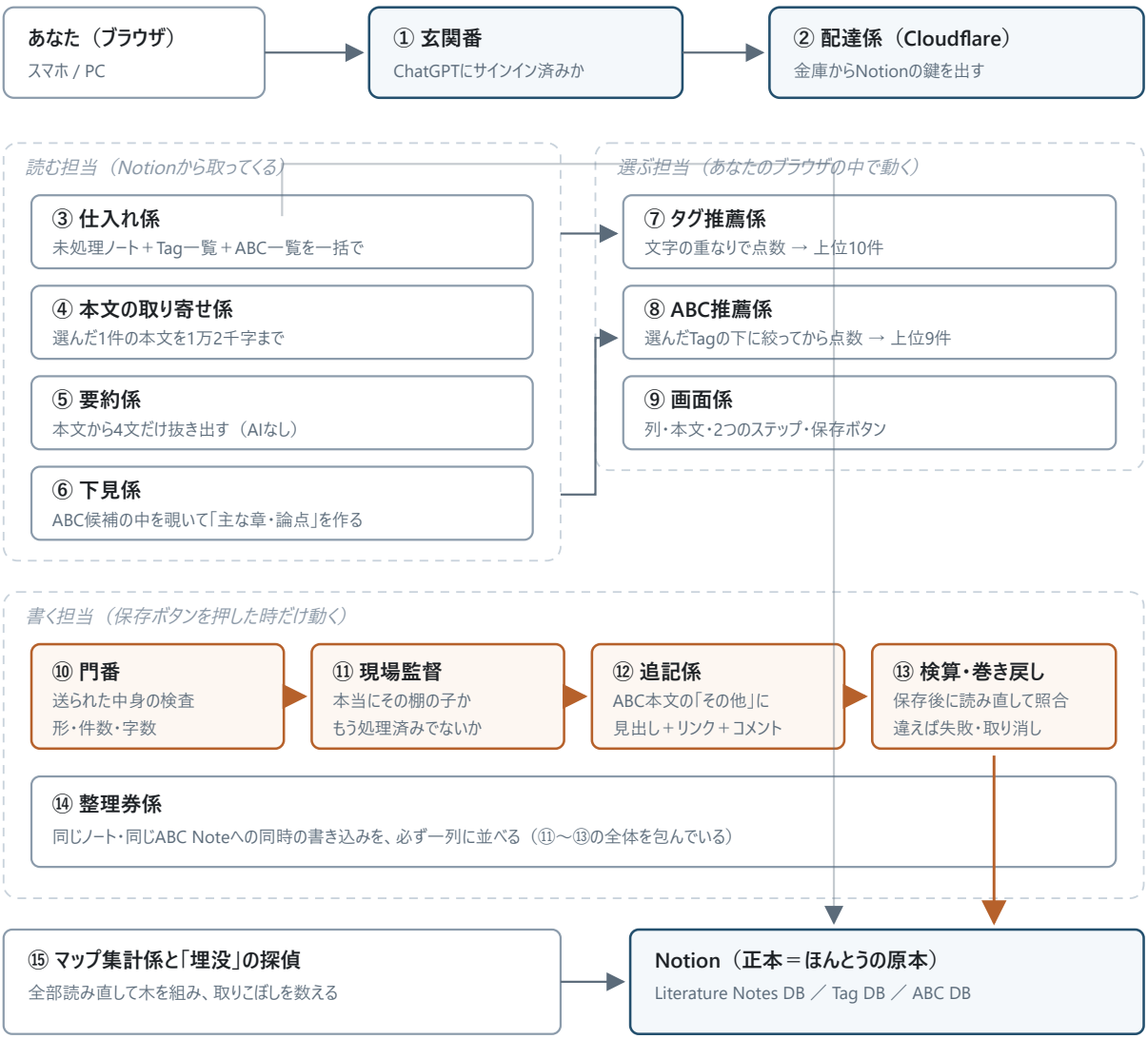

図1　全体像。**色の濃い枠**は出入口とNotion、**オレンジの枠**は書き込みに関わる担当。
①玄関番は、この図のすべての矢印の手前に立っています。そして**この図の中に、AIを呼びに行く矢印は1本もありません。**

用語ミニ辞典（この先で使う言葉）

この先の説明は、なるべくこの18語だけで書いています。分からなくなったらこのページに戻ってください。

サーバー	頼まれた仕事をこなすために、ずっと待ち構えているコンピュータ。このアプリのサーバーは自宅のPCではなく、**インターネット上の貸倉庫**にあります。
ブラウザ	スマホやPCでWebを見る画面のこと（Safari、Chromeなど）。**お客さん側**。このアプリでは、おすすめを計算する仕事もここでやっています。
API	プログラム同士の**受付窓口**。「決まった形で頼むと、決まった形で返ってくる」。人間向けの画面ではなく機械向けの窓口です。Notionにもこの窓口があります。
JSON	データの書き方の一種。**ラベルの付いた箱**にデータを入れた形。例：「title は Sky、count は 7」。人も機械も読めます。
データベース（DB）	表の形でデータを溜めておく箱。ここではNotionの中の3つの表（Literature Notes / Tag / ABC）を指します。
正本（せいほん）	**どれが本物か、迷った時に見るべき原本**。このアプリの正本はすべてNotion側にあり、アプリは自分ではデータを持ちません。
リレーション	Notionの表と表を**ひもで結ぶ機能**。「このメモは、あのABC Noteに属する」という線。このアプリの仕事の大半は、この線を引くことです。
ブロック	Notionのページ本文を作る**1行1行の部品**。見出し、段落、箇条書きなどが、それぞれ1個の部品として番号を持っています。
キャッシュ	一度取ってきたものを**手元に置いておいて、しばらくは使い回すこと**。取り直さない分、速くて、相手にも優しい。
バリデーション	送られてきたものが**形式と中身の条件を満たしているか、機械が点検すること**。人の目ではなくプログラムが行います。
レート制限	相手のサービスが決めた**「1秒あたり何回まで」の上限**。超えると一時的に断られます。だから間隔を空けたり、断られたら待って出し直したりします。
Nグラム	文章を**2文字ずつずらして刻んだ細かい破片**。「知識管理」→「知識」「識管」「管理」。破片がどれだけ重なるかで、文章同士の似ぐあいを機械的に測れます。
スコアリング	候補に**点数を付けて並べ替えること**。このアプリの「おすすめ」は、AIの判断ではなく、この点数の足し算引き算です。
ロック（排他制御）	**整理券**。同じものを2人が同時にいじると壊れるので、順番に1人ずつ通す仕組み。
ロールバック	**巻き戻し**。途中で失敗した時に、それまでに作ってしまったものを消して、無かったことにする後始末。
環境変数	プログラム本体とは別のところに置く**設定と鍵のメモ**。鍵をプログラムに直接書かないための仕組みです。
サーバーレス	自分でサーバー機を持たず、**頼まれた時だけ他社の設備の上で数百ミリ秒だけ動く**やり方。使わない間は何も動いていません。
自動テスト	プログラムが正しく動くかを**別のプログラムに確かめさせる仕掛け**。人が毎回手で確認しなくて済みます。

① 玄関番

AIを使わない　入口

`requireApiAccess` ／ `app/api/_auth.ts`（7行）

窓口に仕事が持ち込まれるたびに、「**この人はサインインしているか**」だけを確認する担当。していなければ、その先の担当者は誰も動きません。

```
あなたのブラウザ ──[依頼：住所＋本文]──▶ 貸倉庫の入口《ここで身分の札を貼る》
        └──[依頼＋札]──▶ ① 玄関番 ──[通過]──▶ ③〜⑱の担当 ──▶ Notion
                                  └──[401＋日本語の断り]──▶ 画面に赤字
```

受け取るもの	依頼＋身分の札（札を貼るのは貸倉庫であって、アプリではない）
出すもの	通過、または401番＋日本語の断り
AI	使わない
失敗した時	その場で終了。**Notionには一切触れない**

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
依頼が窓口に届いた瞬間。7つの窓口すべてで、毎回	依頼の**送り状の1行だけ**を読む。中身の箱は開けない。**なぜ**：本人確認が済む前に、Notionに触る処理を1ミリも進めないため	**HTTPヘッダー。**依頼は「本文」と「ヘッダー（送り状）」でできている。見ているのは `oai-authenticated-user-id` の1行だけ
札を読む、その直前	札を貼ったのはアプリではなく**貸倉庫の入口**。外から同じ札を書いて送っても、最も外側の門番がそこで剥がす前提	**信頼境界。**安全を担保しているのはこのコードではなく置き場所。**引っ越すと同じコードでも守りが消える**
札が無かった時	その場で終了し、**401番＋日本語の断り文**を返す。「権利がない」ではなく「あなたが誰か分からない」と言っている	**ステータスコード。**401＝認証の失敗（誰か不明）、403＝認可の失敗（誰かは分かるが権利なし）
札があった時（通した後）	誰であるかは**もう見ない**。サインインした人は全員、すべての窓口を使える。一人用の道具だから成立する割り切り	**認証と認可。**認証＝誰か、認可＝何をしてよいか。このアプリは**認証だけで認可が無い**
断ると決めた瞬間	断り文は自分で作らず「断りの札」を投げるだけ。**日本語の返事に整えるのは別の係**（`_response.ts`）	**ガード節と例外処理の集約。**気付く人と伝える人を分けると、7つの窓口で返事の形が揃う
手元のPCで開発している間	この確認を**丸ごと飛ばす**。毎回サインインしていては開発が進まないため	**環境による分岐（`NODE_ENV`）。**同じコードが本番と開発で違う顔をする

たとえるなら　会員証の有無だけを見るラウンジのスタッフ。名前も座席も覚えません。そして信用しているのは会員証ではなく、**1階でそれを発行している受付**のほうです。

② 配達係 AIを使わない 入口

worker ／ worker/index.ts（61行）

インターネットから届いた依頼を**最初に受け取り**、金庫から鍵を出して中の担当者に渡し、返事を送り返す担当。このアプリで「サーバー」と呼べるものは、実質この61行です。

```
インターネット ──[依頼]──▶ ② 配達係
    《1.道具箱を受け取る → 2.鍵を詰め替える → 3.行き先で仕分ける》
      ├─[住所が /_vinext/image]──▶ 自分で画像を縮める ──▶ ブラウザ
      └─[それ以外]──▶ ① 玄関番 → 各担当 ──[画面 or データ]──▶ ブラウザ
```

受け取るもの	ブラウザからの依頼すべて＋実行環境が渡す道具箱
出すもの	画面（HTML）／データ（JSON）／縮めた画像
AI	使わない
失敗した時	依頼ごとに独立して失敗する。他には波及しない

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
依頼が届いた瞬間（この関数が呼ばれた時）	「依頼を受け取り、返事を返す」だけの係。削ぎ落とすとそれしか残らない	**fetchハンドラ。**Request（依頼）を受けResponse（返事）を返す関数1個＝サーバーの最小単位
同じ瞬間、依頼と一緒に	道具箱が**外から手渡される。**中身は静的ファイル置き場／画像変換／データベース（空のまま）／**Notionの鍵**	**バインディング（依存を外から渡す）。**渡す側を差し替えれば接続先が変わる。**テストで偽物のNotionを渡せる**のも同じ理屈
中の担当が動き出す、その前	鍵を別の棚に**詰め替える3行。**むだに見えて、実は古い流儀と新しい流儀の翻訳。**なぜ：**Notion担当が `process.env` から鍵を取る前提で書かれているため	**環境変数・シークレットと互換レイヤー。**鍵はコードに1文字も無い。**無駄に見える行は、たいてい2つの世界の継ぎ目**
詰め替えた直後	依頼の**住所を見て仕分ける。**画像を縮める特別な住所だけ自分で処理し、残りは中の担当へ丸ごと渡す	**ルーティング（`url.pathname` による振り分け）。**玄関で行き先を決める発想
返事を返した後	小部屋ごと片付けられて消える。しかも**次はどの拠点で動くか分からない。**前回覚えたことは残っていないかもしれない	**サーバーレス／アイソレート／ステートレス／エッジ実行。**8章「キャッシュは無くても正しい」と7章「保存後に読み直す」が、ここから導かれる
呼ばれていない間	何も動いていないので、費用も発生しない	**従量課金。**個人の道具を、常時起動のサーバー代なしで公開し続けられる理由

たとえるなら　呼ばれた時だけ現れる出前の配達員。エプロン（鍵）は毎回金庫から借り、仕事が済めば返して帰ります。次も同じ人が来るとは限らず、**どの店舗から来るかも毎回違う**ので、頭の中のメモを当てにした働き方は最初からできません。

③ 仕入れ係

AIを使わない　読む

`getBootstrapData ／ lib/notion.ts・app/api/bootstrap/route.ts`

画面を開いた瞬間に走り出し、**その日の仕事に必要なものを3つまとめて運んでくる**担当。「未処理ノートの列」「Tagの全一覧」「ABC Noteの全一覧」です。

```
画面を開く ──▶ ③ 仕入れ係 ──[3セットの問い合わせ]──▶ Notion
    《1.未処理ノート　2.Tag全件　3.ABC Note全件／各100件ずつ手繰る》
    ──[3つを1つにまとめたJSON]──▶ ⑨画面係・⑦⑧の点数計算へ
    （取ったものは30秒だけ手元に保存。保存が成功したら即破棄）
```

受け取るもの	「画面を開いた」という合図だけ
出すもの	3種類の一覧を1つにまとめたJSON
AI	使わない
失敗した時	「Notionを読み込めませんでした」と再試行ボタンだけが残る

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
画面を開いた直後、1回だけ	3種類を**まとめて**取りに行く。**なぜ**：往復のたびに待ち時間が積み上がるため	**初期データの一括取得（bootstrap）。**小さい問い合わせを何度も出すより、まとめて1回のほうが速い
Notionに条件を出す時	**「Tagの線もABCの線も1本も無い」**を"未処理"の定義にしている。未処理フラグを人が立てる必要がない	**状態を持たず、データの形から導く。**フラグを別に持つと必ずズレる。**Notion側で手作業しても矛盾しない**のはこのため
返事が100件で頭打ちになった時	「続きのしおり」を添えてもう一度頼む。しおりが返らなくなるまで繰り返す	**ページネーション（カーソル方式）。**ここを手抜きすると、多い人ほど数え落とすという厄介な間違いが出る
一覧を組み立てる時	Tagには説明が無いので、**配下のABC Note名を3つ並べて**説明文をその場で作る	**導出フィールド。**保存せず、表示のたびに計算する項目。増えても古くならない
並べる時	Tagは**使われた回数の多い順**、ABC Noteは**最近いじった順**。よく使う棚と、いま考え中のノートが上に来る	**ソートキーの選択とロールアップ集計。**「何を上に出すか」は設計判断であり、思想がにじむ場所
取得直後〜30秒／保存成功時	30秒は使い回す。ただし**保存に成功した瞬間に破り捨てる。**古い列を見せないため	**TTLキャッシュとキャッシュ無効化。**難しいのは貯めることではなく、**いつ捨てるか**を決めること

たとえるなら　朝いちばんに倉庫へ行き、その日使う材料を台車1台で運んでくる係。「未処理の箱」を探すのではなく、**「行き先ラベルが貼られていない箱」**を全部持ってくる、という探し方をします。

④ 本文の取り寄せ係

AIを使わない　読む

`getNoteDetail` ／ `app/api/notes/[id]/route.ts`

列から1件選んだ瞬間だけ動き、**そのノートの本文を取ってきて、ただの平たい文章に均す**担当。1件ずつしか運ばないのは、全部を先に運ぶと遅くて重いからです。

```
1件を選ぶ ──[ノートID]──▶ ④ 取り寄せ係 ──[本文の部品を100個ずつ]──▶ Notion
    《飾りを捨てて文字だけにする／1万2千字で打ち切る》
    ──[平たい文章＋「切りました」の印]──▶ ⑤要約係・⑦⑧の点数計算・画面
```

受け取るもの	選ばれたノートの番号1つ
出すもの	本文の文章（最大1万2千字）と「途中で切りました」の印
AI	使わない
失敗した時	本文欄だけエラー。**Tag選びと保存は続けられる**

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
列から1件選んだ瞬間	その1件だけを取りに行く。**なぜ**：数十件の本文を先に全部運ぶと、開くだけで長く待たされるため	**遅延読み込み（オンデマンド取得）**。「必要になった時に、必要な分だけ」
Notionへ頼む前	番号の形（32文字の英数字）を点検し、変ならその場で断る	**入力検証**。おかしな依頼を外部サービスへ**転送しない**のも守りのうち
本文の部品が届いた時	見出しも段落も区別せず**文字だけ**抜く。文字が無ければURLや画像の説明文を拾う	**木構造の平坦化**。Notionの本文は入れ子の部品の集まり。用途に応じて構造を捨てる判断
1万2千字に達した時	そこで打ち切り、**「切りました」の印を付けて**返す。画面にもそう出る。**黙って切らない**	**上限設計と切り詰めの明示**。制限そのものより、**制限したことを伝えるか**が品質を分ける
100件ごと	しおりを添えて続きを取る	**ページネーション**。③と同じ作法をここでも使う
本文が取れなかった時	本文欄だけ赤くなる。Tag選びも保存も動き続ける。おすすめの精度が落ちるだけ	**機能の縮退（グレースフルデグラデーション）**。一部が壊れても全部は倒れない作り

たとえるなら　書庫から1冊だけ出してもらい、**中身をコピーして持ってくる**係。装丁や挿絵は写さず、文字だけ。分厚すぎる本には「ここまでで打ち切りました」と付箋を貼って渡します。

⑤ 要約係

AIを使わない　読む

`summarizeNoteText` ／ `lib/note-summary.mjs`

本文から**4文だけを選んで並べる**担当。文章を新しく書くことはしません。**もともとあった文の中から選ぶだけ**です。ここがAIの要約との決定的な違いです。

④の本文 ──▶ ⑤ 要約係《あなたのブラウザの中。どこにも送らない》
1.ゴミ行を捨てる → 2.文に切る → 3.点数を付ける → 4.4文選ぶ → 5.元の順に戻す
──[最大4文・560字。すべて元の文そのまま]──▶ 画面の「要点」欄

受け取るもの	④が取ってきた本文
出すもの	最大4文・560字までの要点（元の文そのまま）
AI	使わない
失敗した時	要点欄が空になるだけ。他には何も影響しない

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
本文が届いた直後	「Copyright」「ログイン」「関連記事」「広告」「続きを読む」で始まる行、URL、日付だけの行を捨てる	**ノイズ除去とパターンマッチ（正規表現）。**Webページ丸ごと保存には、本文でない部品が大量に混ざる
掃除のすぐ後	表記を揃えてから見比べ、**同じ内容の重複**を落とす	**正規化（NFKC）と重複排除。**比べる前に形を揃えるのは、この後の⑦⑧でも繰り返し出てくる原則
点数を付ける時	**他の文にも出てくる2文字の破片を多く持つ文**ほど高い点。文章全体が繰り返し語っている話題の中心を、機械的に見つける	**Nグラムと中心性。**意味を理解せず「よそでも使われている言葉か」だけで重要度を測る、抽出型要約の考え方
同時に	長すぎる文は減点、**冒頭の1文は無条件で採用。**記事の第1文はたいてい主題だから	**特徴量の重み付け。**位置・長さ・頻度という素朴な手がかりの足し算
4文を選び終えた後	**本文に出てきた順に並べ直す。**これをしないと話の流れが逆立ちして読みにくい	**順序の復元。**選ぶ処理と見せる処理を分けて考える
全体を通して	新しい文は1文字も作らない。だから**元の記事に無い話が混ざりようがない**	**抽出型と生成型の違い。**生成型（LLM）に付きまとう「もっともらしい嘘」が、原理的に起きない

たとえるなら　本文に**蛍光ペンで4か所だけ線を引く**人。自分の言葉は1文字も足しません。「何度も出てくる言葉が入っている文はどれか」を数えて線を引いているだけです。

⑥ 下見係

AIを使わない　読む

`extractCandidateContent` ／ `app/api/candidate-content/route.ts`

ABC Noteの候補が画面に並んだら、**その中身をこっそり覗きに行って「主な章・論点」の一文を作る**担当。名前だけでは「どっちの棚だったか」を思い出せない、という問題を解くためにいます。

候補9件が画面に並ぶ ―[IDを最大9件]―▶ ⑥ 下見係
―[3件ずつ・間に0.9秒]―▶ Notion ―[各ノートの見出しと段落]―▶
《大見出し＝章、中見出し＝論点として一文に組み立てる》
―[220字の紹介文＋点数用の本文6千字]―▶ 画面の説明文／⑧の材料（6時間保存）

受け取るもの	画面に並んでいるABC Noteの番号（1回に1〜9件）
出すもの	候補ごとの短い紹介文（220字）と、点数計算用の本文（6千字）
AI	使わない
失敗した時	紹介文が出ないだけ。候補の並びは名前だけで機能し続ける

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
候補が画面に並んだ直後	並んだ**9件ぶんだけ**中を覗きに行く。**なぜ**：全ABC Noteの中身を先に取ると重すぎるため	**必要な分だけ取る設計。**「候補を絞ってから詳細を取る」は、あらゆる一覧画面の定石
中身を読む時	**大見出しを3つ、中見出しを2〜3つ**拾って「主な章: ○○／論点:「□□」」という一文を組み立てる	**文書構造の利用。**見出しの階層は、書き手が付けた意味の目印。**構造がそのまま説明文になる**
拾う時	「その他」だけの見出し、記号だけの行、1文字の行は捨てる。**なぜ**：どのノートも「主な章: その他」になってしまうため	**フィルタ規則。**実データを見て初めて分かる例外を、規則として書き足す作業
Notionへ頼む時	**3件ずつ、間に0.9秒待つ。**一気に9件投げると断られる可能性がある。急がないことで結果的に速く終わる	**レート制限とスロットリング。**相手のあるシステムでは、速度より**断られないこと**が優先される
結果を手元に置く時	6時間・最大500件。ただし**そのノートの最終更新時刻を目印に一緒に保存**し、変わっていれば取り直す	**内容の印による検証。**時間だけに頼ると6時間ずっと古い中身を見せる。**時間＋印の二段構え**
同じ候補が同時に求められた時	取得中であることも覚えていて、**先に走っている1回の結果を二人で分け合う**	**重複リクエストの合流。**同じ問い合わせを二重に投げない、地味だが効く工夫

たとえるなら　候補の本を9冊選んだあと、店員が**1冊ずつ目次だけを見せに来てくれる**ようなもの。全文は読みません。目次を読み上げるだけで、たいていは探している本か分かります。

⑦ タグ推薦係

AIを使わない　選ぶ

`tagRecommendationScore` ／ `lib/recommendation.mjs`

数十件のTagから、**いま開いているノートに合いそうな10件を選んで上に出す**担当。このアプリの「賢さ」の半分は、この30行ほどの計算式です。

題名＋書きかけのコメント＋本文 ──▶ ⑦ タグ推薦係《ブラウザの中で計算・外に出ない》
　全Tagに点数（名前の一致・破片の重なり・使用回数・整理用タグは減点）
　──[高い順に上位10件]──▶ 画面のステップ01

受け取るもの	ノートの題名＋あなたが書きかけのコメント＋本文と、Tagの全一覧
出すもの	点数順に並べ替えたTag（上位10件を表示）
AI	使わない（**ブラウザの中で計算が終わる。どこにも送られない**）
失敗した時	失敗しようがない。材料が少なければ順番が平凡になるだけ

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
文字を1文字打つたび	そのつど全Tagを計算し直す。**あなたの言葉が、そのままヒントとして効く**	**再計算とメモ化。**材料が変わった時だけ計算し直す仕掛けがあるので、打っても重くならない
比べる前	全角と半角、大小文字を統一し、空白と記号を落とす。「**ＧｅｎＡＩ・推論 モデル**」→「**genai 推論モデル**」	**正規化。**これをしないと同じ言葉が別物として扱われる。日本語では特に効く
点数を出す時	Tag名が本文にそのまま出れば＋**140点**、破片の重なりで＋85点、親名と説明文で＋65点、使用回数で＋5〜15点	**スコアリング関数と重み付け。**「賢さ」の正体は、この足し算の設計そのもの
重なりを測る時	「候補の何割が本文に含まれるか」72％＋「互いの共通度」28％で混ぜる。1つの尺度の癖を、別の尺度で打ち消すため	**包含率とDice係数の合成。**短い名前が有利すぎる／長い名前が不利すぎる、を互いに補正する
整理用のTagが候補に入った時	「あとで読む」「本」などは－**45点**。どんなノートにも当てはまるのに、知識としては何も分類していないから	**設計者の意思を数値に埋め込む。**コードの中で、方針がいちばんはっきり見える場所
検索欄に文字を入れた時	おすすめ順をやめ、ふつうの絞り込み（最大24件）に切り替える。**人が答えを知っている時、機械のおすすめは邪魔**	**明示的な意図が推薦に優先する。**推薦機能を設計するうえでの基本的な作法

たとえるなら　原稿をぱらぱらめくって**「この単語が何回出てくるか」だけを数え**、棚の名札と見比べて候補を10個挙げる司書。意味は分かっていません。それでも、たいてい当たります。

⑧ ABC推薦係

AIを使わない　選ぶ

permanentRecommendationScore ／ lib/recommendation.mjs

Tagが決まってから動く担当。**まず選ばれたTagの下だけに候補を絞り、それから点数を付ける**という二段構えが肝です。この順番のおかげで、数百件が数件になります。

Tagが決まる ──▶ ⑧ ABC推薦係
1. そのTag直下だけに絞る（親Tagがちょうど1つ、かつ今のTag）
2. 名前で点数 ──[上位9件]──▶ ⑥下見係へ「中身を見てきて」
3. 中身が届いたら もう一度点数 ──[並べ替え]──▶ 画面のステップ02

受け取るもの	選ばれたTag、ノートの文章、⑥が取ってきた候補の中身
出すもの	上位9件のABC Note（点数順）
AI	使わない（これもブラウザの中で完結）
失敗した時	候補が0件なら「新しく作る」側へ誘導される

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
Tagが決まった瞬間	点数より先に**絞る**。数百件が数件になる。**なぜ**：候補が少ないほど、人も機械も間違えないから	**候補生成と絞り込みの二段構え。**検索・推薦の世界で最も効く、ありふれた、そして強力な型
絞る条件を当てはめる時	候補になれるのは「**親Tagがちょうど1つで、それが今選んだTag**」だけ。複数の棚にまたがるノートは外れる	**整合性の規律をアプリ側で守らせる。**外れたものは⑮が「要確認」として拾い、取りこぼさない
点数を出す時	選んだTagと一致＝＋120点、名前が本文に出れば＋120点、**ABC本文との重なりは最大＋115点**、名前の重なりは＋70点	**重み設計。**名前(70)より本文(115)を重くしている＝**「名前が似た棚」より「同じことを書いている棚」を選ぶ**という判断
下見の結果が届いた時	名前だけで選んだ9件を、**中身を材料に加えて計算し直す**。順番が入れ替わることがある	**段階的な絞り込み（粗く→細かく）。**「中身を見てから決め直す」を機械がやっている形
実績を見る時	これまで何件ぶら下がっているかで、わずかに加点（＋2〜6点）	**対数での加点。**件数が多いほど効きを鈍らせ、大きな棚が独占しないようにする
候補が0件だった時	「新しいABC Noteを作る」側へ自然に誘導する	**空の状態の設計。**何も無い時に何を見せるかで、道具の使いやすさが決まる

たとえるなら　まず「経済」の棚の前まで歩いていって、**その棚の中だけで**背表紙を見比べる人。館内全部から探すより速く、間違いも少ない。しかも9冊抜き出したあと、中を開いて並べ直します。

⑨ 画面係

AIを使わない　選ぶ

InboxApp ／ app/InboxApp.tsx（約1,150行・アプリ最大）

あなたの目の前にあるもの全部の担当。列、本文、要点、2つのステップ、コメント欄、保存ボタン。**そして⑦⑧の点数計算も、この中（＝あなたの端末の中）で走っています。**

③〜⑧のデータ ──▶ ⑨ 画面係《あなたのスマホ／PCの中》
　列 ／ 本文 ／ 要点 ／ ステップ01・02 ／ コメント欄 ／ 転記イメージ
　──[題名・コメント・Tag・ABCが全部そろった時だけ]──▶ 保存ボタンが押せる
　──[1回のまとまった依頼]──▶ ⑩門番へ

受け取るもの	③〜⑧が用意したデータと、あなたの操作
出すもの	画面の表示と、保存ボタンを押した時の1回の依頼
AI	使わない
失敗した時	その場所にだけ赤い字が出る。画面全体は生き残る

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
常に（操作のたび）	題名・**コメントが空でない**・Tagが1つ以上・ABCが決まっている、が全部そろうまでボタンは灰色	**事前検証と状態管理。**間違いを後で直させるのではなく、**最初からできなくする**という考え方
ABCの指定方法を切り替えた時	「既存を選ぶ」と「新しく作る」は同時に選べない。ただし**同じ点検を⑩門番も行う**	**二重の検証。**画面の検証は親切のため、窓口の検証は安全のため。**役割が違うので両方要る**
ノートのURLがXの投稿だった時	投稿そのものを表示する。読み込めなければ静かにリンク表示に戻す	**外部埋め込みとフォールバック。**他社の部品は落ちる前提で組む
日経・NewsPicksなどの記事の時	**購読しないと本文が取れない出典**だと先に知らせる。「本文が空＝故障」ではないと分かるように	**期待値の管理。**直せない制約は、隠さず伝えるほうが道具の信頼が上がる
保存ボタンを押す前	コメント欄の下に**ABC Noteへの転記イメージ**が出る。何がどこに入るか、押す前に見える	**副作用の可視化（プレビュー）。**取り消しにくい操作ほど、事前に見せる価値が高い
1件終わるごと	列から消し、「今回○件完了」を数える。作業が進んでいる感覚を残す	**進捗のフィードバック。**機能ではなく、続けられるかどうかを左右する部分

たとえるなら　仕分け台そのもの。手が届く範囲に道具が並び、**条件がそろわないと判子が押せない**作業台です。しかも計算尺（点数計算）は台の上にあり、本社に問い合わせずその場で答えが出ます。

⑩ 門番

AIを使わない　書く

`POST /api/process ／ app/api/process/route.ts`

保存ボタンが押されて**最初に会う人**。届いた中身を1項目ずつ点検し、1つでもおかしければ**Notionに触る前**に突き返します。

```
⑨から ──[JSON：ノートID／Tag／ABC／題名／コメント]──▶ ⑩ 門番
    形・件数・字数・「同時指定の禁止」を点検
      ├─[NG]──▶ 400番＋日本語の断り ──▶ 画面（Notionには触れていない）
      └─[OK]──▶ ⑪ 現場監督へ
```

受け取るもの	ノート番号、Tag番号、ABC番号（または新規名と親Tag）、題名、コメント
出すもの	通す、または**日本語のエラー文**（画面にそのまま出る）
AI	使わない
失敗した時	400番で即座に返す。Notionには一切触れていない

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
依頼が届いた瞬間	まず**中身が読めるか**を確かめる。壊れていたら「送信内容を読み取れませんでした」	**入力の解析と例外処理**。外から来るものは壊れている前提で扱う
番号を受け取った時	形を検査し、ハイフンの有無を**1つの書き方に揃える**。同じものが別物と扱われる事故を入口で潰す	**検証と正規化（正準形）**。「受け付けたら、まず1つの形に直す」は堅い作りの基本
件数と字数を見る時	Tagは10件、ABCは10件、題名500字、**コメント1万2千字**まで。上限が無いと事故で巨大なデータが来る	**上限設計**。悪意だけでなく**うっかり**への備え。上限の無い入力欄は、いつか必ず溢れる
既存選択と新規作成を見る時	同時指定をはっきり断る。画面でも防いでいるが、**窓口は画面を信用しない**	**サーバー側での再検証**。窓口は誰からでも直接叩ける。**画面の検証は破れる前提**で組む
新規ABCの親Tagを見る時	「今回選んだTagの中の1つ」でなければ断る。関係のない棚の下に新しいノートが生まれるのを防ぐ	**参照整合性の事前確認**。壊れたつながりは、作らせない段階で止めるのが最も安い
断ると決めた時	すべて日本語で、**次に何をすればよいか**が書かれた文を返す	**エラーメッセージ設計**。番号だけ返す窓口は、使う人の時間を奪う

たとえるなら　役所の窓口。記入漏れやはみ出しをその場で指摘して突き返します。**いったん受け付けてから奥で差し戻すより、はるかに安全で速い**という発想です。

⑪ 現場監督

AIを使わない　書く

`processInboxNote / archiveInboxNote ／ lib/notion.ts`

実際にNotionを書き換える一連の作業を、**順番どおりに、辻褄が合っているかを確かめながら**進める担当。このアプリで一番神経を使っている場所です。

⑩から ──▶ ⑩整理券を取る ──▶ ⑪ 現場監督
1.本人確認（本当にその表のページか）　2.まだ未処理か　3.選んだTagの子か
4.（新規なら）ABC Noteを作る ──▶ ⑫追記係 ──▶ ノート本体を更新 ──▶ ⑬検算係
（途中で失敗したら、その回に作ったものは⑬がゴミ箱へ戻す）

受け取るもの	⑩門番を通った、形の正しい依頼
出すもの	書き換えの結果（新規ABCの情報、本文をどう更新したか）
AI	使わない
失敗した時	**止まった段階によって後始末が変わる。**作ったものがあれば⑬が消す

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
いちばん最初	そのページを取り直し、**本当にLiterature Notes DBのものか**を確かめる。番号の形が正しくても、正しい表のページとは限らない	**形式の検証と所属の検証は別物。**⑩が形を、⑪が中身と出どころを見る、という分業
次に	Tagの線かABCの線が**1本でも付いていたら止める**（409）。別の端末やNotionアプリで先に処理されていた場合の備え	**競合の検出。**「取ってから書くまでの間に、他人が変えたかもしれない」を前提に作る考え方
その次に	選んだABC Noteが、選んだTagの子であることを確かめる。画面表示中にNotion側で棚が付け替えられていたら弾く	**参照整合性の再確認。**画面のデータは、送信の瞬間にはもう古いかもしれない
新規作成の時	アイコン・親Tag・目次・「その他」見出し・今回の記入を**最初から入れて**作る	**初期状態は作る側が責任を持つ。**後から整えるより、生まれた瞬間に正しい形にする
書き込む順番	先にABC Noteの本文、**最後にノート本体。**本体の更新が成功して初めて、そのノートは列から消える	**手順の固定。**順番を決めることが、途中で落ちた時の被害を小さくする最も簡単な方法
削除を頼まれた時	同じ作法で、**まだ未処理のものだけ**をゴミ箱へ。しかも**移った印を確認してから**成功と答える	**論理削除と事後確認。**完全消去ではないのでNotion側で戻せる。「消したつもり」を作らない

たとえるなら　工事現場の監督。資材が図面どおりか、既に他の業者が手を付けていないかを**着工前に一つずつ確認**し、決めた順番でしか作業させません。

⑫ 追記係

AIを使わない　書く

`upsertAbcOtherEntry` ／ `lib/notion.ts`

ABC Noteの本文の「**その他**」という見出しの下に、**今回の1件を差し込む**担当。差し込む形は決まっていて、**見出し（題名）＋元ノートへのリンク＋あなたのコメント**の3点セットです。

```
⑪から ──[ABC ID／ノートID／題名／コメント]──▶ ⑫ 追記係
    「その他」を探す ──2つ以上あった──▶ 中止して人に相談（409）
          ├─[同じリンクが既にある]──▶ 足さずに書き換える
          ├─[本文の奥に隠れていた]──▶ 何もしない
          └─[無い]──▶ 「その他」の最後に 見出し＋リンク＋コメント を差し込む
```

受け取るもの	ABC Noteの番号、ノートの番号・題名・コメント
出すもの	「差し込んだ／書き換えた／すでにある／見出しごと作った」のいずれか
AI	使わない
失敗した時	ノート本体の更新へ進まずに中断する

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
最初	本文の見出しを上から見て、**全角半角や空白のゆれを吸収して**「その他」を探す	**文字列の正規化。**人が手で書いた見出しは、必ずゆれる
「その他」が2つ以上あった時	**作業を止めて人に相談する**（「Notion側で1つに整理してください」）。どちらに書くべきか機械には決められないから	**曖昧な状態では自動判断しない。**推測で進めた自動化は、後から追跡できない被害を出す
差し込む前	同じノートへのリンクが既にあれば、**新しく足さずに書き換える。**2回押しても結果は1つ	**冪等性（べきとうせい）。**何度実行しても同じ結果になる性質。自動化の設計で最も重要な考え方のひとつ
見つからない時	「その他」の外や折りたたみの中に隠れていないか、**入れ子の奥まで潜って**探す（最大5,000か所）	**再帰的な探索と打ち切り。**探索には必ず上限を付ける。無限に潜らせない
入れる場所を決める時	見出しの階層を読み、**次の同じ強さの見出しが始まる直前**を区切りとして、その手前に入れる	**木構造の区間特定。**見出しは並んでいるだけに見えて、実は入れ子の範囲を持つ
コメントが長い時	1,900字ずつに機械的に割って入れる。見た目は1つの文章のまま	**相手の制約に合わせた分割。**Notion側の1かたまりの上限に合わせている

たとえるなら　バインダーの「未分類」の仕切りの**一番後ろ**に用紙を綴じる人。綴じる前に全ページをめくって同じ用紙が無いか確かめ、あれば**その1枚を差し替えます。**仕切りが2枚ある変なバインダーには、手を出しません。

⑬ 検算係と巻き戻し係

AIを使わない　書く

`processInboxNote 後半（検証と後始末）／ lib/notion.ts`

書き終わった**後**でもう一度Notionから読み直し、**頼んだとおりになっているかを1項目ずつ突き合わせる**担当。合っていなければ、成功とは言いません。

```
書き込み完了 ──▶ ⑬ 検算係 ──[もう一度読む]──▶ Notion
    照合：題名／コメント／Tagの本数／ABCの本数／選んだ番号がすべて含まれるか
      ├─[全部合う]──▶ 成功 ──▶ ③のキャッシュを破棄 ──▶ 列から1件消える
      └─[1つでも違う]──▶ 502で失敗（列から消さない）＋新規作成分はゴミ箱へ
```

受け取るもの	書き込みが一通り終わった状態
出すもの	「成功」、または「保存結果を確認できませんでした」
AI	使わない
失敗した時	502番。画面に赤字が出て、**列からノートは消えない**

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
書き終えた直後	「保存できました」という**返事**ではなく、**実際にNotionに残っている中身**を見て成否を決める	**書き込み後の読み取り検証。**ネットの途中で切れることも、一部だけ通ることもある、という前提に立つ
照合する時	題名が一致するか、コメントが一字一句一致するか、本数が合うか、**選んだ番号がすべて含まれるか。**1つでも欠ければ失敗	**不変条件の検査。**「こうなっているはず」を言葉ではなくコードで書き、毎回確かめる
新規ABCを作った時	正しい表にあるか、親Tagがちょうど1つで指定どおりか、**今回のノートがぶら下がっているか**まで見る	**事後条件の確認。**作った直後がいちばん確かめやすい
途中で失敗した時	その回に**新しく作ったABC Noteをゴミ箱へ移してから**エラーを返す。「棚だけできて中身が空」を残さない	**補償トランザクション（巻き戻し）。**まとめて取り消せない相手には、**打ち消す操作**を自分で書く
後始末にも失敗した時	**最初のエラーを優先して**人に伝える。掃除でつまずいたことより、本来何が起きたかのほうが大事だから	**エラーを握り潰さない。**コードのコメントにも、その理由が日本語で書かれている
成功した時	③の作り置きを**即座に破り捨てる。**次に開いた時、確実に新しい列が出る	**書き込みに連動したキャッシュ無効化。**「速さ」と「正しさ」の境目はここにある

たとえるなら　振込のあと必ず通帳を記帳し、金額と相手先を指差し確認する人。**「送信しました」の画面ではなく、記帳された結果を信じます。**

⑭ 整理券係

AIを使わない　書く

`withLock / withNoteLock / withAbcBodyLock ／ lib/notion.ts`

姿は見えませんが、⑪〜⑬の全体を外側から包んでいる担当。**同じノート、同じABC Noteへの作業が、絶対に同時に走らないよう一列に並べます。**

```
⑪〜⑬ の全体を、外から包む
    同じノートID への作業 ──▶ [列1] ──▶ 1件ずつ通す
    同じABC ID の本文書き換え ──▶ [列2] ──▶ 1件ずつ通す
    （前の人が失敗しても、列は必ず前に進む／効くのは"いま動いている小部屋の中"だけ）
```

受け取るもの	「これからこのノート（ABC Note）をいじります」という申告
出すもの	順番が来たら通す。前の人が終わるまで待たせる
AI	使わない
失敗した時	前の人が失敗しても**列は必ず前に進む**（詰まらせない）

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
書き込みが始まる直前	同じ相手に対する作業を**一列に並べる**。同じものを2人が同時にいじると壊れるから	**ロック（排他制御）**。「同時に触らせない」という、並行処理でいちばん基本の道具
列を選ぶ時	列は2種類。**ノート用**と**ABC本文用**。守りたい対象が違うので列も分ける	**ロックの粒度**。粗いと遅くなり、細かいと守り漏れる。どこで区切るかが設計判断
なぜ必要か（⑫を思い出す）	⑫は「探す→無ければ足す」の2段階。**2件が同時に走ると、両方が「無い」と判断してから両方が足す**	**競合状態（レースコンディション）**。確認と実行の**あいだ**に他人が割り込む、という古典的な不具合
作業が終わった時	成功でも失敗でも**必ず**次に譲る。誰も待っていなければ列そのものを片付ける	**確実な解放**。失敗した人が列を塞いだままになると、全体が止まる（デッドロック）
別の端末から同時に保存された時	この列では**防げない**。効くのは、いま動いている小部屋の中だけ	**メモリ内ロックの限界**。②の「毎回別の場所で動く」が、ここで効いてくる
だから、その先で	⑪の「もう処理済みか」の確認と、⑬の読み直しが**最後の砦**になる	**多層防御**。1つの守りに頼らず、破られた時に次で受ける

たとえるなら　試着室の前の整理券。**同じ部屋に2人を入れない**ためだけに存在します。ただし整理券は**その店舗の中でしか通じない**ので、別店舗と同時に在庫をいじられるのは防げません。だから最後にレジで確認します。

⑮ マップ集計係と「埋没」の探偵

AIを使わない　見える化

`getKnowledgeMapData / analyzeSubBuriedAbcNotes ／ lib/notion.ts`

もう1つの画面「全体マップ」の担当。**3つの表を全部読み直して木の形に組み立て、「取りこぼし」を数えます。**書き込みは一切しません。

```
/map を開く ──▶ ⑮ 集計係 ──[3つの表を全件]──▶ Notion
    木を組む：Tag → ABC Note → Literature Note
    数える：埋没（棚止まり）／サブ埋没（線だけ）／要確認（親が0か2つ以上）
    ──[木＋13種類の集計値]──▶ 画面（5分保存）
    「サブ埋没」ボタン ──[最大3件]──▶ 探偵が本文の奥を潜る（30分保存）
```

受け取るもの	Notionの3つの表の全件
出すもの	Tag→ABC→Literatureの木と、13種類の集計値
AI	使わない
失敗した時	マップが出ないだけ。受信箱の作業には影響しない

いつ動くか	誰が・何を・なぜ（やさしく言うと）	技術のことば（入り口）
マップを開いた時	3つの表を**全件**読む。重い処理なので5分は使い回す	**全件走査。**件数に比例して遅くなる（＝ノートが2倍なら時間も2倍）。11章の改良案はここに効く
木を組み立てる時	関係が25件を超えると一度に返ってこないので、**「続きがある」印が立っていたら追加で全部取りに行く**	**ページネーション漏れ＝数え落とし。**手抜きすると**多くぶら下がっている棚ほど間違える**という、たちの悪い不具合になる
数える時	**埋没**＝Tagはあるが行き先なし。**サブ埋没**＝線はあるが本文に一度も出てこない。**要確認**＝親Tagが0か2つ以上	**指標の定義。**「何を問題とみなすか」を決める作業そのものが、この画面の価値
「サブ埋没」を調べる時	そのABC Noteの本文を**入れ子の奥まで全部めくり**、リンクとして登場する番号を集める。線はあるのに出てこないものが該当	**グラフの探索。**関係（線）と本文中の言及は別物、という気付きがこの機能の出発点
結果を持っておく時	30分。ただし**更新時刻と、ぶら下がる顔ぶれが変われば捨てる。**1回に3件までしか調べない	**内容の印による検証と負荷の制限。**重い処理ほど、範囲を先に区切る
画面に並べる時	Tagを**埋没が多い順**に並べる。開くと、放置が多い棚が上に来る	**見える化を作業指示に変える。**数字を出すだけで終わらせない、という設計判断

たとえるなら　棚卸し担当。「箱はあるが中身が空の棚」と「台帳には載っているのに実物が見当たらない品」を分けて数え、**問題の多い棚から順に**報告書を並べて持ってきます。

1回の流れを時間軸で追う

「1件を仕分けて保存する」あいだに、誰がいつ動いているか。左から右へ時間が流れます。

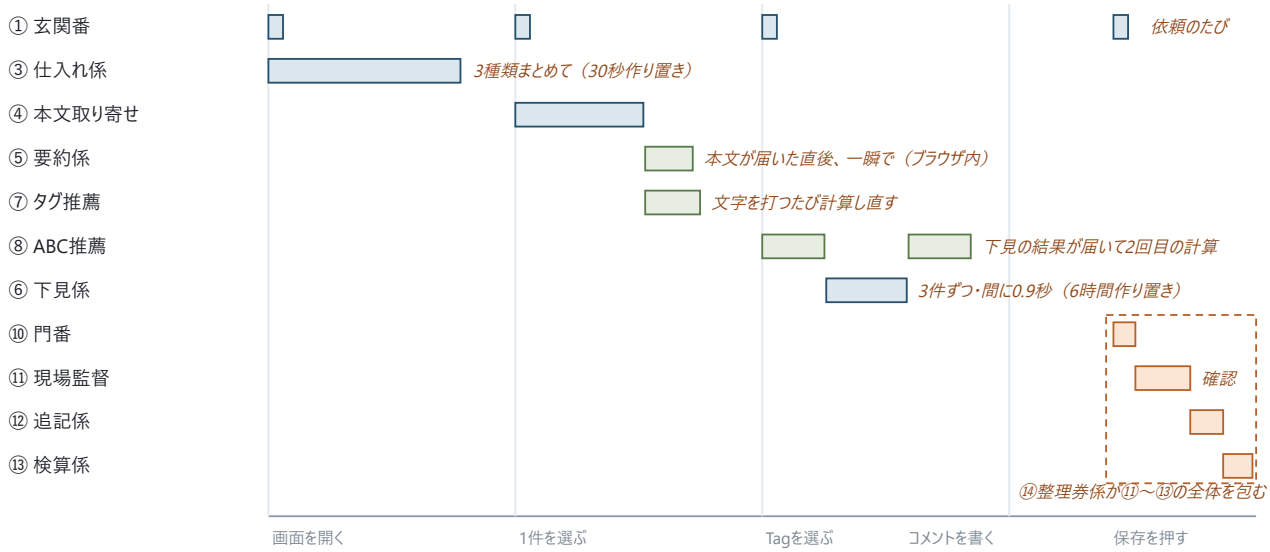

図2　1件を処理する間の時間の使われ方。**薄緑**はあなたのブラウザの中だけで完結する仕事。

同じ流れを、言葉で追う

1. **画面を開く。**①が身分を確かめ、③が3種類の一覧をまとめて取ってきます。ここが一番待つ場面（Notionへの問い合わせが3セット）。
2. **1件選ぶ。**④が本文を取りに行き、届いた瞬間に⑤が4文を抜き、⑦がタグの点数を計算して10件を並べます。**⑤と⑦はネットに出ないので一瞬です。**
3. **Tagを選ぶ。**⑧が候補を9件に絞り、⑥がその9件の中身を見に行きます。戻ってきたら⑧が**もう一度**計算して並べ替えます。
4. **コメントを書く。**1文字打つたびに⑦⑧が計算し直すので、書いているうちにおすすめの順番が変わります。**あなたの言葉が、そのままヒントとして効いています。**
5. **保存を押す。**⑩が形を点検→⑭が整理券を発行→⑪が本人確認と処理済みチェック→（新規なら作成）→⑫が本文に追記→ノート本体を更新→⑬が読み直して照合。**ここまで全部成功して初めて**「保存しました」と出て、列から1件消えます。

> **この流れから読み取れること**
>
> ネットの向こう（Notion）に出かける回数を、できるだけ**後ろに、少なく**しています。考えている間はブラウザの中だけで完結し、外に出るのは「読む時の数回」と「保存の一回」だけ。だから、迷っている時間が長くても通信は増えません。

AIを使う所と、使わない所

> **結論：このアプリは、動いている間にAIを1回も呼びません**
>
> 15人の担当者を全部調べても、AIに問い合わせる行は1行もありません。設定にAIの鍵もありません。「**おすすめ**」も「**要約**」も、**すべて文字を数える計算です。**

では、何がこのアプリを賢く見せているのか

絞ってから選ぶ	Tagを先に決めさせ、**その棚の下だけ**を候補にする。数百件が数件になる。これはAIではなく**手順の設計**による賢さです。
文字の重なりを測る	2文字ずつの破片がどれだけ共通かを測る（Nグラム）。意味は分かっていないのに、実用的にはよく当たります。しかも名前だけでなく**ABC Noteの中身**とも比べ、中身の一致を名前の一致より重く数えます。
意思を数字にする	「あとで読む」のような整理用タグに－**45点**。作った人の方針が、点数として書き込まれています。
人に決めさせる	最終的にどのTag・どのABC Noteかは、必ず人が押します。機械は**並べ替えるところまで**。

AIはどこにいたのか

1か所だけあります。**このアプリ自体が、ChatGPTを使って作られ、ChatGPTのサイト置き場に公開されている**ことです（住所が `...chatgpt.site` なのはそのため）。つまり**AIは「作る時」と「置き場所」には関わっているが、「動いている時の判断」には一切関わっていない。**この線引きが、このアプリの性格を決めています。

この設計の良いところと、限界

良いところ

- **お金がかからない**（AIの利用料が発生しない）
- **速い。**文字を打つたびに計算し直しても待たされない
- **結果がぶれない。**同じ材料なら必ず同じ順番になる
- **ノートの中身が外に出ない。**計算はブラウザの中だけ
- **もっともらしい嘘が出ない。**要約は元の文を抜くだけ

限界

- **言い換えに弱い。**「生成AI」と「LLM」は文字が重ならないので、別の話に見えてしまう
- **英語の記事と日本語のTagは、まず結びつかない**
- 「言葉は違うが同じことを言っている」を見抜けない。**ここだけはAIが得意な領域**

よくある誤解

「AIアプリではないなら、単純な道具なのか」――逆です。**AIを使わずに実用的な精度を出すために、絞り込みの順番・点数の重み・二段階の計算という工夫が必要**でした。AIを呼べば1行で済んだ部分を、意図して手で組んであると読むのが正確です。

間違いを起こさない5つの工夫

相手（Notion）は別のサービスで、途中で通信が切れることも、他の端末から同時にいじられることもあります。その前提で置かれている備えが5つあります。

工夫1　同じ規則を、二重に守らせる

「既存を選ぶ」と「新しく作る」を同時にできない、という規則は**画面**（⑨）**でも、窓口**（⑩）**でも、現場**（⑪）**でも**点検されます。画面だけで守ると、窓口を直接叩かれた時に破られます。**窓口は画面を信用しない**――これはWebアプリ全般の鉄則です。

工夫2　書いた後に、読み直して突き合わせる

⑬の仕事です。「保存できました」という**返事**ではなく、**実際にNotionに残っている中身**を見て成否を決めます。合わなければ失敗として扱い、ノートを列から消しません。**取りこぼしよりも、取りこぼしに気付かないことのほうが怖い**という判断です。

工夫3　途中で失敗したら、作ったものを消して戻す

新しいABC Noteを作った直後にエラーが起きた場合、そのABC Noteをゴミ箱へ移してからエラーを返します（**ロールバック**）。中途半端な残骸は、後から見ると原因が分からなくなるためです。

工夫4　「もう処理済みのもの」には手を出さない

保存も削除も、**最初に必ず「まだTagもABC Noteも付いていないか」を確認**します。付いていれば、そこで止めて「画面を更新してください」と伝えます。Notionアプリ側で先に手を付けていた時に、上書きしてしまう事故を防ぎます。

工夫5　決めていいことと、そうでないことを分ける

⑫追記係は、「その他」の見出しが2つあると**作業をやめて人に相談します**。機械が推測で片方に書き込めば、後から探せなくなるかもしれない。**迷ったら止まる**ほうが、直す手間が小さいという判断です。

> **おまけ：断られた時は、待って出し直す**
>
> Notionから「混んでいます」と断られた場合（**レート制限**）、**相手が指定した秒数（1〜5秒に丸めて）待ってから、最大2回まで自動でやり直します**。それでも駄目なら「少し待って再実行してください」と日本語で伝えます。**Notion側のエラー番号は、すべて日本語の説明に翻訳されてから画面に出ます。**

データはどこに、どう置かれているか

> **いちばん大事な一文**
>
> **このアプリは、自分のデータを1件も持っていません。**正しいデータ（**正本**）はすべてNotionの中にあり、アプリは「読んで、見せて、書き戻す」だけの窓口です。画面の右上に「正本 Notion」と表示されているのは、その宣言です。

置き場所の一覧

Literature Notes DB	読んだものの控え。**入り口であり、作業対象。**題名・コメント・Tagの線・ABC Noteの線・URL・媒体を持つ
Tag DB	観点の一覧。親子関係と、「何件のノートで使われたか」の集計を持つ
ABC DB（Zettelkasten DB）	永久保存ノート。親Tagへの線と、ぶら下がるノートへの線、そして**本文**を持つ
アプリ側のデータベース	**空。**使える枠は用意されているが、中身は意図的に何も入っていない

なぜ自前でデータを持たないのか

- **二重管理をしないため。**アプリにも控えを持つと、Notion側で直した時にどちらが正しいか分からなくなります。
- **Notionで直接いじっても壊れないため。**Notionアプリ側で手作業でTagを付ければ、このアプリの列からも自然に消えます（③のとおり「線が無い＝未処理」だから）。
- **このアプリが無くなっても、知識はNotionのページとして残るため。**

そのかわり必要になるのが「作り置き」

毎回すべてをNotionに取りに行くと遅く、相手にも負担がかかります。そこで4種類の**キャッシュ**を使い分けています。

対象	保つ時間	捨てるきっかけ
未処理ノート・Tag・ABCの一覧（③）	30秒	**保存に成功した瞬間に即破棄**
全体マップの集計（⑮）	5分	同上（保存成功時）
ABC候補の中身（⑥）	6時間	**そのノートの最終更新時刻が変わったら破棄**（最大500件まで保持）
サブ埋没の調査結果（⑮）	30分	更新時刻と、ぶら下がるノートの顔ぶれが変わったら破棄

> **作り置きの正しい扱い方**
>
> 4つとも、**時間で切れるだけでなく「元が変わったら捨てる」という条件**を必ず持っています。時間だけに頼ると、6時間ずっと古い中身を見せる危険があるためです。そして②で触れたとおり、**作り置きは消えてもアプリは正しく動きます。**「あれば速い、無くても正しい」――これが安全な作り置きの条件です。

どうやって動き、外からどう使えるのか

置き場所は、ChatGPTが用意したWebサイト置き場

住所は `zettelkasten-inbox-ak.akuramochi.chatgpt.site`。前半がアプリ名、後半があなたの作業場の名前です。**ChatGPTが提供しているサイト公開の仕組み**に置かれ、実際の実行はCloudflareという会社の設備の上で行われます（②配達係が動く場所）。**自宅のPCの電源が落ちていても動きます。**

> **StudyGuideとの決定的な違い**
>
> StudyGuideはご自宅のPCの中で動き、外から使うために専用の通路（トンネル）を通していました。こちらは最初から外の設備の上にあります。**その代わり、鍵の管理と身分確認を、置き場所の仕組みに任せることになります。**

鍵の置き場所

- **Notionのキー**は、プログラムの中には書かれていません。置き場所側に預けた**環境変数**として保管され、②配達係が依頼のたびに金庫から出します。**このプログラムを丸ごと他人に見せても、Notionは開けません。**
- **3つの表の番号**も同じ仕組みで差し替えられます（置き場所側の設定が優先）。
- **入口の鍵はChatGPTのサインイン**。サインインすると、置き場所が依頼に身分の札を貼ります。①玄関番はその札だけを見ています。

外から使える窓口は7つだけ

窓口	向き	用途
`GET /api/bootstrap`	読む	未処理ノート・Tag・ABCの一覧をまとめて
`GET /api/notes/{番号}`	読む	1件の本文
`POST /api/candidate-content`	読む	ABC候補の中身（1〜9件）
`GET /api/map`	読む	全体マップの集計
`POST /api/map/sub-buried`	読む	サブ埋没の調査（1〜3件）
`POST /api/process`	**書く**	1件の仕分けを確定する（このアプリ唯一の書き込み窓口）
`DELETE /api/notes/{番号}`	**書く**	ノートをNotionのゴミ箱へ

書き込みができる窓口は**2つだけ**。窓口を絞ってあること自体が、安全の一部です。

使っている道具

React / Next.js（vinext）	画面を組み立てる道具。ボタンを押すと画面の一部だけが変わる、という作りを支えている
Cloudflare Workers	依頼が来た時だけ動く実行場所（**サーバーレス**）
Notion API	Notionの受付窓口。バージョン2026-03-11を指定して使っている

品質を保つ仕組み

自動テスト8本

「直したつもりが別のところを壊す」を防ぐため、**プログラムを別のプログラムに検査させる**仕掛けが8本あります。しかもその多くが、**Notionの偽物を用意して**本物に触れずに検査しています。

テスト	何を確かめているか
`process-validation`	おかしな依頼が**全部ちゃんと断られるか**（番号の形、件数超過、同時指定、字数超過）
`process-write`	正しい依頼で、**Notionへ送られる中身が意図どおりか**。新規作成と追記の両方
`note-delete`	処理済みのノートを**間違って捨てないか**。ゴミ箱に入ったことを確認しているか
`sub-buried-validation`	調査依頼の件数制限（1〜3件）が効いているか
`candidate-content`	候補の中身取得で、**Notionを叩く回数が増えすぎていないか**（作り置きが効いているか）
`recommendation`	表記ゆれの統一と点数計算。「ＧｅｎＡＩ・推論モデル」→「genai推論モデル」など
`note-summary`	**実際の日本語の記事**を材料に、広告やコピーライト行が落ち、要点が残るか
`rendered-html`	画面が最後まで組み立たるか（起動そのものが壊れていないか）

> **ここが実務的に効いている考え方**
>
> - **「断れるか」のテストが、「できるか」のテストと同じ数だけある。**正しく動くことより、**間違ったことをしないこと**を重く見ています。
> - **本物のNotionを使わずに検査する。**偽のNotionを立てて、そこに何が送られたかを見る。だから**何度検査しても本物のデータは1文字も変わりません。**
> - **検査は組み立て直してから走る。**公開されるのと同じ形に組み立ててから検査するので、「手元では動くのに公開したら動かない」が起きにくくなります。

もう1つの品質：エラー文が全部日本語

Notionから返る英語のエラーは、すべて**その場で日本語の説明に置き換えられて**から画面に出ます。「401」ではなく「Notion APIキーが無効です」、「404」ではなく「対象のNotionデータベースが連携に共有されていません」。**自分ひとりで使う道具でも、半年後の自分は他人だ**という考え方の表れです。

弱いところと、これから選べる道

欠陥という意味ではなく、「**今はこう割り切ってある**」という設計上の選択と、その代償を並べます。

外部への依存

- **Notionが止まれば、何もできません。**読むことも書くこともできず、手元に控えもありません（8章のとおり意図的にそうしています）。
- **Notionの受付窓口の仕様が変われば、影響を受けます。**版数を指定して使っているので急には壊れませんが、いつかは追随が要ります。
- **混雑時の待ちは、実際に起こりえます。**特に全体マップは全件を読むので重く、ノートが増えるほど時間が延びます。

作りの上での割り切り

- **作り置きは、その瞬間動いている場所の中だけ。**②のとおり動く場所は毎回変わりえるので、同じ操作でも速さが揺れます。
- **整理券（⑭）も、その場所の中だけ。**2つの端末から同時に同じノートを保存した場合、整理券では防げません。「**もう処理済みか**」の確認と保存後の読み直しが最後の砦です。守りはあるが一本ではない、という状態。
- **本文の追記先が「その他」で固定。**内容にふさわしい章に振り分けることはしません。人が後で移す前提です。
- 「**埋没**」「**サブ埋没**」を見つけても、**直すのは手作業。**マップは指摘するだけです。
- **入口の身分確認は「サインイン済みか」だけ。**置き場所の公開設定が変わると意味が変わります（①）。
- **名前が3通りある。**Notion上は「🧠 Zettelkasten DB」、プログラム内では「permanent（永久）」、画面では「ABC Note」。同じものです。

もし次に手を入れるなら（優先度順の一案）

順	やること	効き目と手間
1	**言い換えに強くする**（同義語の対応表を持つ、または推薦の最後の数件だけAIに並べ替えさせる）	6章の唯一の弱点への直球。対応表なら手間も費用もほぼ増えない
2	**マップから直接「埋もれたノート」を処理できるようにする**	指摘して終わりを、片付けまでつなげる。既存の窓口をつなぐだけで済む
3	**全体マップの集計を軽くする**（更新されたぶんだけ数え直す）	ノートが増えるほど効いてくる。作りは少し複雑になる
4	**追記先の章を選べるようにする**	「その他」に溜まり続ける問題への対処。画面の追加が必要

> **あえてやらないほうがよいこと**
>
> **コメント欄をAIに書かせること。**「自分の言葉に直す」がツェッテルカステンの核であり、このアプリが存在する理由そのものです。ここを自動化すると、道具は速くなりますが、**知識は育たなくなります。**AIを足すなら「候補の並べ替え」まで、というのが筋の通った線引きです。

この資料の言葉と、実際のファイルの対応

実物を開いて確かめたい時のための対応表です。すべて `C:\Users\aky_m\Scripts\zettelkasten-inbox\` の下にあります。

この資料での呼び名	実際の名前	場所
① 玄関番	`requireApiAccess`	`app/api/_auth.ts`
② 配達係	`worker.fetch`	`worker/index.ts`
③ 仕入れ係	`getBootstrapData`	`lib/notion.ts` ／ `app/api/bootstrap/route.ts`
④ 本文の取り寄せ係	`getNoteDetail`	`lib/notion.ts` ／ `app/api/notes/[id]/route.ts`
⑤ 要約係	`summarizeNoteText`	`lib/note-summary.mjs`
⑥ 下見係	`extractCandidateContent / getCandidateContents`	`lib/notion.ts` ／ `app/api/candidate-content/route.ts`
⑦ タグ推薦係	`tagRecommendationScore`	`lib/recommendation.mjs`
⑧ ABC推薦係	`permanentRecommendationScore`	`lib/recommendation.mjs`
⑨ 画面係	`InboxApp`	`app/InboxApp.tsx`
⑩ 門番	`POST` ハンドラ	`app/api/process/route.ts`
⑪ 現場監督／捨てる係	`processInboxNote / archiveInboxNote`	`lib/notion.ts`
⑫ 追記係	`upsertAbcOtherEntry / appendBlocks`	`lib/notion.ts`
⑬ 検算係・巻き戻し係	`processInboxNote` 後半（`verifiedOk` の判定）	`lib/notion.ts`
⑭ 整理券係	`withLock / withNoteLock / withAbcBodyLock`	`lib/notion.ts`
⑮ マップ集計係	`getKnowledgeMapData`	`lib/notion.ts` ／ `app/map/KnowledgeMap.tsx`
⑮ 埋没の探偵	`analyzeSubBuriedAbcNotes / collectBodyReferencedPageIds`	`lib/notion.ts` ／ `app/api/map/sub-buried/route.ts`
データの形の定義	型定義	`lib/types.ts`
見た目（配色・レイアウト）	スタイル	`app/globals.css`
自動テスト8本	`*.test.mjs`	`tests/`
置き場所の設定	`hosting.json`	`.openai/hosting.json`

Notion側の名前の対応

この資料での呼び名	Notionでの表示名	プログラム内での呼び名
Literature Note（文献ノート）	Literature Notes DB／プロパティ「名前」「コメント」「タグ」「URL」「媒体」「ステータス」	`literature`
Tag（観点）	Tag DB／プロパティ「名前」「親アイテム」「Out_Permanent」	`tags`
ABC Note（永久保存ノート）	🧠 Zettelkasten DB／プロパティ「名前」「タグ」「Literature Notes DB」	`permanent`
整理用のTag（減点対象）	「あとで読む」「本」「読みたい本、ブログ」ほか	`UTILITY_TAGS`

決め打ちされている上限の一覧

本文の取得	1万2千字	コメント	1万2千字	題名	500字
Tagの同時指定	10件	ABC Noteの同時指定	10件	候補の中身取得	1回9件
サブ埋没の調査	1回3件	点数計算に使う本文	6千字	本文めくりの上限	5,000か所

3つだけ持ち帰るなら

1　このアプリの賢さは、AIではなく「順番」でできている

Tagを先に決めさせ、その棚の下だけを候補にする。この一手で、選択肢が数百件から数件になります。同じことをAIに一発でやらせることもできますが、**手順を工夫すれば、AIなしでも実用に足りる**という実例になっています。判断を機械に渡すかどうかは、能力の問題ではなく**設計の選択**だ、ということです。

2　外部サービスを使う時は、「書いた後に読み直す」

Notionのような他社のサービスとやり取りする以上、「OKと返ってきた＝保存された」とは限りません。このアプリは保存のたびに読み直して1項目ずつ突き合わせ、合わなければ失敗として扱います。地味ですが、**他のどんな自動化にもそのまま応用できる考え方**です。今後、別の道具を作るとき・頼むときの、確認事項の筆頭にできます。

3　道具が守らせているのは、「自分の言葉で書く」という規律

コメント欄が空だと保存ボタンが押せない。「その他」の見出しが2つあると機械は手を止めて人に聞く。全体マップは「棚に入れただけで考えに結びついていないノート」を数え上げて突きつける。**このアプリは、作業を楽にする道具であると同時に、楽をしすぎないための道具**でもあります。そこが、この15人の並びから読み取れる、いちばん人間くさい設計思想です。

※ この資料は、2026年8月10日時点の `zettelkasten-inbox` のプログラムを1行ずつ読んで作成しました。
※「担当者」という言い方はこの資料の説明のための呼び名で、プログラム内にその名前が付いているわけではありません。対応は付録の表のとおりです。
※ 数値（点数の重み、上限、作り置きの時間）はすべて実際のコードから取った値です。プログラムを直せば、これらの数値も変わります。