ChatGPT

Midday-to-Zeke Feature Mapping

This document maps key features of the Midday open-source project to the Zeke platform's core primitives – **Stories**, **Sources**, **Insights**, **Briefs**, and **Playbooks** – and suggests how to adapt or rename them. It also highlights architectural patterns from Midday (TRPC structure, Supabase usage, modal flows, sync jobs) that Zeke can emulate. All mappings cover both UI components and backend/data alignment.

Feature Mapping Table

Midday Feature	Mapped Zeke Primitive(s)	Description / Justification	Implementation Notes
Import Modal (CSV Import for Transactions)	**Sources** (Data Ingestion)	Midday's **ImportModal** UI allows users to upload a CSV of bank transactions, acting as an entry point for external data [1]. This aligns with **Sources** in Zeke, which represent incoming raw information. The multi-step import flow (file selection, field mapping, confirmation) mirrors adding a new source of data.	Reuse Midday's multi-stage modal design for "Add Source." For example, implement a **Source Import Modal** with steps: select file/type → map fields → confirm import [1]. In backend, create a server action (e.g. `importSourceAction`) similar to Midday's `importTransactionsAction` to handle file upload and processing [2]. This action should store the file in Supabase (Zeke's storage) and trigger an ingestion job. Leverage URL query params to control modal state (e.g. `?step=import`) as Midday does [3], enabling deep links and state persistence.

Midday Feature	Mapped Zeke Primitive(s)	Description / Justification	Implementation Notes
Magic Inbox (Automated Receipt Inbox)	**Sources** (Ingestion Pipeline)	Midday's "Magic Inbox" automatically ingests incoming invoices/receipts and matches them to transactions [4]. In Zeke, this concept maps to **Sources** handling automated input feeds. It represents an inbox of raw source materials (emails, files, etc.) that are processed into the system. The Magic Inbox performs OCR and classification on documents [5], akin to Zeke ingesting and analyzing source content upon arrival.	Introduce a **Source Inbox** in Zeke to capture incoming source documents. For example, allow users to forward emails or upload files to a unique "inbox" address/folder (similar to Midday's email ingestion). Utilize background jobs for processing: e.g. extract text, classify content, and link sources to relevant insights/stories. Midday's pipeline (upload → OCR → match) can serve as a playbook for automation [5]. Zeke might rename "Magic Inbox" to **Source Inbox** or **Imports**. Ensure the backend supports this via a dedicated **inbox** table and TRPC router (as Midday's does [6] [7]) to manage incoming source records and their processing status.

Midday Feature	Mapped Zeke Primitive(s)	Description / Justification	Implementation Notes
Vault (Documents Storage & UI)	**Sources** (Document Library)	Midday's **Vault** is a secure file repository for important documents (contracts, receipts, etc.) [8]. This corresponds to Zeke's **Sources** library – a central place to store and browse all source documents. In Midday, uploaded documents are stored in a Supabase "vault" bucket and categorized (e.g. `teamId/inbox/…`) [9]. The Vault UI (grid/list, tags, search) lets users manage files. Similarly, Zeke needs a document library for source materials.	Adapt Midday's Vault module as Zeke's **Source Library**. Repurpose the UI components: e.g. a **VaultView** page in Zeke's app to list source documents with filtering and tagging (Midday's vault components like `vault-item`, `vault-header`, filters can be reused with new naming) [10] [11]. On the backend, mirror Midday's `documents` schema and TRPC router [12] for CRUD operations on sources. Zeke might keep the name "Vault" or use **Library**/**Files** for clarity. Implement tagging and search for sources as Midday does (leveraging a `document_tags` table and search integration). Also preserve linking capabilities – Midday links documents to transactions via attachments [13], and Zeke should allow linking source files to **Insights** or **Stories** for traceability.

Midday Feature	Mapped Zeke Primitive(s)	Description / Justification	Implementation Notes
Transactions (Financial Records & Matching)	**Insights** (Discrete Findings) *(or Internal Data Model)*	Midday's **Transactions** are granular records of financial events (date, amount, category, etc.), which get enriched by matching receipts (documents) [14] [15]. Zeke may not deal with bank data, but an **Insight** in Zeke (a specific fact or finding derived from sources) can be treated similarly to a transaction record: an atomic piece of information with metadata, linked to source evidence. Each Midday transaction also carries status (unmatched, matched) which parallels an insight's validation or review status.	If Zeke requires tracking many small pieces of information, adopt a **Transaction-like model** for insights. This means using a structured table for insights with fields for content, source reference, tags, status, etc., akin to Midday's transactions schema. Midday's TRPC router for `transactions` [16] and matching logic can guide how to implement insight linking (e.g., similar to how receipts attach to transactions via `transaction_attachments`). If direct mapping isn't needed, Zeke can skip a "transactions" module, but still leverage Midday's patterns: e.g. use Zod schemas and routers to handle insight creation, and possibly an automated matching mechanism (Midday's matching engine) to connect new sources to existing insights/ stories. Note that terminology should shift – Midday's "transactions" would be internal, whereas Zeke would call them **Insights** or **Findings** in the UI.

Midday Feature	Mapped Zeke Primitive(s)	Description / Justification	Implementation Notes
Reports (Financial Overview & Analytics)	**Briefs** (Compiled Reports)	Midday's **Reports** module generates summaries of financial data (e.g. spending by category, cash flow over time). This equates to Zeke's **Briefs**, which are compiled reports or summaries of insights. Both represent higher-level aggregation of data for end-users. Midday's philosophy is that once data is reconciled, it enables powerful reporting and insights [17]. Zeke's briefs similarly would present synthesized information (e.g. a summary of findings or a narrative report).	Use Midday's Reports as a template for **Brief** generation in Zeke. On the UI side, Midday likely has pages or components for reports (charts, tables) – Zeke can create a **Brief Viewer/Editor** that might combine text and data visualization. The backend should support assembling briefs from underlying data: for example, queries that aggregate source-derived insights (mirroring how Midday's reports query transactions and documents). If Midday uses pre-defined report types (e.g., monthly summaries), Zeke can introduce templates for briefs. We might rename "Reports" to **Briefs** in the code, and repurpose Midday's reports TRPC router [18] for retrieving compiled brief data. Ensure briefs can be saved and perhaps edited by users (Midday's upcoming **Invoicing** feature – creating collaborative web-based invoices [19] – can inspire how to make briefs editable and shareable).

Midday Feature	Mapped Zeke Primitive(s)	Description / Justification	Implementation Notes
AI Assistant & Insights (Tailored Analysis)	**Insights** (AI-generated or Query)	Midday includes an AI **Assistant** that provides insights into the user's financial situation (answering questions like "How much tax did I pay?") [20] [21]. This corresponds to Zeke's **Insights** primitive – especially those generated on the fly or via AI. Essentially, both platforms use AI to derive meaning from data and surface it as insights. In Zeke, the assistant might analyze sources to produce key findings or answer queries, which would be captured as insights for the user.	Integrate an **AI Insight Assistant** in Zeke, following Midday's approach. Midday's code uses OpenAI (and possibly vector search with pgvector) to generate and retrieve insights [22]. Zeke can implement a similar TRPC endpoint (e.g., `search` or `ask`) to query embeddings of source content and return answers or insight snippets. These AI answers can be logged as **Insights** in Zeke's system for reference. From a UI perspective, consider a chat or query interface like Midday's assistant. Any patterns from Midday's AI filters (e.g. `generate-transactions-filters.ts`, `generate-vault-filters.ts` for intelligent filtering [23]) can be repurposed to help users sift through sources. The assistant's insights might not be persistent by default, but Zeke could allow saving an AI result as a confirmed insight (similar to saving a report or adding to a brief).

Midday Feature	Mapped Zeke Primitive(s)	Description / Justification	Implementation Notes
Ingestion Workflows (e.g. Matching Engine)	**Playbooks** (Automated Sequences)	Midday's back-end workflows – such as the document matching engine and background jobs – are essentially orchestrated sequences of steps (upload → OCR → match → notify) [24] [15]. These can be viewed as **Playbooks** in Zeke: repeatable, automated procedures that operate on sources to produce outcomes. While Midday doesn't call them "playbooks," the concept is reflected in how Trigger.dev jobs coordinate multi-step processes. For instance, importing transactions or syncing bank connections triggers a sequence of tasks on the server [2] [25]. Zeke's playbooks can similarly automate complex analyses or data collection tasks.	Implement **Playbook workflows** in Zeke by drawing on Midday's job orchestration. Midday utilizes Trigger.dev (`tasks.trigger(...)`) to run multi-step jobs asynchronously [2], and a realtime status is tracked to update the UI [26]. Zeke can adopt this pattern for long-running processes (e.g., ingesting a large document corpus or performing an extensive analysis). Define playbooks as sequences of actions, possibly stored as configurable workflows. In practice, this means creating background job handlers (using a job queue or serverless functions) for each playbook, and front-end hooks to monitor progress (like Midday's `useSyncStatus` hook updating sync state [26]). Though Midday's domain is different, the architectural pattern of **URL-driven flows** + **background tasks** + **real-time updates** is directly applicable. Zeke might expose "Playbooks" to users as templates (e.g., a "Source Ingestion Playbook" that automates importing a source and extracting insights, modeled after Midday's Magic Inbox pipeline).

(Notes: Some Midday features like Time Tracking or Team Management do not have direct analogs in Zeke's primitives and are omitted. However, their underlying patterns – e.g., project entries could inform story timelines, team multi-tenancy for user orgs – can be considered separately.)

Architectural Patterns & Best Practices from Midday

Beyond individual features, Midday's architecture offers patterns that Zeke should replicate for a robust platform:

- **Modular TRPC Router Structure:** Midday organizes its API with a clear separation by domain – e.g. `transactions`, `documents`, `inbox`, `reports` each have their own tRPC router [27] and Zod schema definitions. Zeke should mirror this approach by creating routers for each primitive (Stories, Sources, Insights, etc.), which promotes clean encapsulation of logic. This end-to-end type safety (from database to client) will make Zeke's development more maintainable. The repository shows a dedicated **types package** and use of Zod schemas (even integrating with OpenAPI in Midday's case) to ensure consistent contracts across services.

- **Supabase for Database and Storage:** Midday uses Supabase (Postgres + storage) as its backbone [28] [29]. Zeke can leverage the same stack – a Postgres database (with Row-Level Security for multi-team data isolation) and Supabase Storage for files. In Midday, all user documents are stored in a single `vault` bucket with organized paths (team-scoped directories) [9], and they retrieve signed URLs for access [30]. Adopting this, Zeke should store source files in a secure bucket, perhaps reusing Midday's strategy of path tokens and expiration for downloads. Supabase's real-time capabilities can also be utilized: Midday likely listens to changes (for example, new transactions or match status) to update the UI instantly – Zeke can do the same for new insights or story updates.

- **URL-State Driven Modal Flows:** Midday's UI heavily uses URL query parameters to manage modal navigation and state. For instance, opening the import workflow sets `?step=import` which the component reads to determine if the **ImportModal** is open [3]. Each sub-step in the flow (connect account vs upload CSV) is a state in the URL, making the UI refresh-/share-friendly. Zeke should adopt this pattern for its workflows – e.g., creating a new Story or adding a Source could push a `step=` or route state so that modals (or sidebars) are driven by URL. This makes complex flows (like the proposed Source Import or multi-step Playbooks) easier to manage and debug. Midday also uses this for back-navigation in modals (e.g., a back button simply changes the URL param to go to a previous step [31]). Implementing a small hook like Midday's `useQueryStates` in Zeke will simplify state management for modals.

- **Sync Status and Background Jobs:** Midday offloads heavy tasks (CSV parsing, bank sync, AI processing) to background jobs using Trigger.dev, and uses a consistent pattern to report status back to the UI. After kicking off a job via an action (e.g., `importTransactionsAction` triggers a job and returns a `runId` [2]), the front-end monitors this via a custom hook `useSyncStatus` which subscribes to the job's status (e.g., via WebSocket or polling) [32] [26]. When the job completes or fails, the UI is updated and relevant data is re-fetched [33]. Zeke should employ a similar mechanism for any long-running processes – for example, a "source ingestion" playbook that might take time. Use a background worker (Trigger.dev, Celery, or Supabase Functions) to run the job and have the client listen for a completion event. On completion, auto-refresh the affected views (Midday invalidates the cache for transactions, accounts, reports, etc., then shows a success toast [34] – Zeke can do likewise, e.g., refresh the Sources list or new Insights and notify the user). This pattern ensures a responsive UI without blocking the user during processing.

- **Modal & Inline Actions (Server Actions):** Midday's Next.js 13 app makes use of React Server Actions for mutations – for example, the ImportModal calls an action `importTransactionsAction` on form submit instead of a traditional API call [35]. This action lives in the frontend code (`apps/dashboard/actions/...`) but securely executes on the server with authentication [36]. Zod is used to validate inputs, and side-effects (like updating DB via Supabase client and triggering jobs) occur server-side [37] [2]. Zeke should follow this modern pattern: define server actions for creating stories, uploading sources, generating briefs, etc., which can be imported and used directly in React components. It simplifies API calls and maintains type safety. Additionally, Midday wraps sensitive actions in an `authActionClient` (ensuring the user context is provided) [38] – Zeke can implement a similar wrapper for any state-changing operation to enforce auth and team scoping.

- **Supabase Realtime & Sync:** Although not explicitly detailed in the code snippets, Midday likely uses Supabase's realtime features (or listens via Trigger.dev events) to keep data in sync (for example, reflecting new transactions as they arrive from a bank feed). Zeke can benefit from this by subscribing to changes on important tables (Sources, Insights, etc.) so that collaborators see updates live. This complements the manual cache invalidation approach.

- **Design System and UI Components:** Midday uses ShadCN UI (Radix-based components) and Tailwind for a consistent design [39]. Zeke can accelerate development by reusing these UI components for modals, tables, buttons, etc. For instance, the **Dialog** component and pattern used in Midday's modals [40] can be replicated to ensure accessible, stylized modals in Zeke. Maintaining parity in UI structure (sidebar layout, page organization under a Next.js `app` directory) will make it easier to integrate Midday's code. The Vault and Inbox UIs in Midday already provide a lot of the functionality Zeke needs (file listing, empty states, loading skeletons, etc. as seen in vault components [41] [42]), so adapting them preserves a polished UX.

By leveraging these patterns, Zeke's platform will not only gain the functionality mapped in the table above but also a strong foundation modeled on a proven open-source architecture.

"Import Modal – Vision & Flow" for Zeke Sources

To illustrate how Zeke can integrate the above ideas, consider the **Source Import Modal** flow inspired by Midday's import process. This will cover adding a new Source (e.g., uploading a document or connecting an external source) in a guided, multi-step manner:

1. **Launch Import Flow:** The user clicks "Add Source" in Zeke's UI. Instead of immediately navigating away, this action updates the URL (for example, adding `?step=import`) and opens the **Import Source Modal** in front of the current page [3]. Zeke's app reads the URL and determines that the import modal should be shown (using a hook similar to Midday's that sets `isOpen = (params.step === "import")`).

2. **Step 1 – Choose Source Type:** The modal first presents options to bring in a source. This could include **Upload File**, **Enter URL**, or **Connect Account** (if Zeke supports integrations like APIs or email). This stage is analogous to Midday's "connect" step in the import flow (where a user might choose a bank connection or manual CSV). If the user selects a direct integration (for example, an RSS feed or database), Zeke could initiate an OAuth/connect flow in a similar pop-up or inline frame,

then skip to confirmation. If the user chooses **Upload File**, the modal will prompt them to select a file from their computer. (In Midday, clicking a back arrow sets `step=connect` vs `step=import` [31]; in Zeke, we could manage these sub-steps under the import modal umbrella).

3. **Step 2 – Upload & Preview:** Once a file is chosen, the modal automatically advances to the next stage (e.g., `step=import&stage=mapping`). Zeke should then display a preview of the uploaded source. Midday accomplishes this by reading the CSV's columns and first rows, then moving to a "Confirm import" page [43] [1]. For Zeke, if the source is a document, this preview might show the text extracted or simply the file name and size. If it's a CSV/structured data, Zeke can mimic Midday's field mapping UI: list detected columns and allow the user to map them to Zeke's standard fields (if applicable). This step ensures the user can verify that the source data is interpreted correctly before ingestion. Under the hood, once the file is selected, Zeke should upload it to Supabase storage immediately (to a `sources` bucket, similar to Midday's `vault` bucket [44]). The file path returned can be stored temporarily while the user confirms.

4. **Step 3 – Confirmation & Ingestion:** The user confirms the import settings and submits. At this point, Zeke invokes a server-side action to actually ingest the source. For example, calling an `importSourceAction` (which we create akin to Midday's importTransactionsAction). This action would take parameters like the file path (from the previous upload), any mapping info, and perhaps the source type, then perform the following on the server:

5. Save a new Source record in the database (status = "processing").
6. If textual, start an extraction job (e.g., read PDF, perform OCR, generate embeddings).
7. If structured data, parse rows and store them or link them to new Insight records.
8. Trigger any follow-up Playbook workflows (similar to how Midday triggers an import task for transactions [2]).

In code, this could utilize a background jobs system; Midday uses `tasks.trigger("import-transactions", payload)` to delegate heavy work to the background [2]. Zeke can do the same (e.g., `tasks.trigger("ingest-source", { sourceId, ... })`). The action immediately returns an identifier for the job (or some `runId`), which the front-end will use to track progress.

1. **Step 4 – Progress Tracking:** After submission, the modal can show a progress indicator (e.g. "Importing… please wait"), or even allow the user to close it while work continues. Midday's UI sets an `isImporting` state and uses `useSyncStatus(runId)` to listen for completion [45] [26]. Zeke's implementation would subscribe to the job status via WebSocket or polling. For example, using Supabase Realtime on a `jobs` table or a Trigger.dev listener to update when done. While syncing, the modal might show a spinner and messages like "Parsing document…" for transparency.

2. **Completion & Feedback:** Once the background process finishes, the status hook will return "COMPLETED" or "FAILED". Midday's pattern is to then reset the modal state and show a toast notification [46] [47]. Similarly, Zeke should close the Import modal (clearing the `step=import` param), possibly navigate the user to the new Source detail page or highlight it in the Sources list. A success message like "Source imported successfully" is displayed, as well as any immediate insight (e.g., "3 Insights extracted from the document") if applicable. Meanwhile, the front-end should refresh relevant queries so the new data appears without a full page reload. Midday does this by invalidating caches for transactions, accounts, reports [33] – Zeke can invalidate or refetch the Source

list and any dashboard counts. If the job failed, show an error alert (Midday provides a generic error toast on failure [48]). The user can always re-trigger the flow if something went wrong.

3. **Post-Import Linking:** (Optional) If the source is related to an existing Story or if an Insight was auto-generated, Zeke might prompt the user to confirm linking it. For instance, "Source imported. Create a Story from this source?" – This can be a small follow-up modal or toast action. This extends the flow but can be part of the playbook: Midday's equivalent is the Magic Inbox auto-matching receipts to transactions behind the scenes, whereas Zeke could auto-match new sources to open research questions or story threads if intelligence is applied.

Throughout this flow, the **UX is modeled after Midday's proven design**: using a modal overlay for focus, breaking the process into two or three concise steps, and giving feedback at each stage. By following this vision, Zeke ensures that adding Sources – a foundational activity – is as smooth as Midday's import experience, setting the stage for users to seamlessly bring in data and immediately start deriving value (Insights, Briefs, Stories) from it.

---

[1] [2] [3] [6] [7] [9] [10] [11] [12] [13] [16] [18] [23] [25] [26] [27] [30] [31] [32] [33] [34] [35] [36] [37] [38] [40] [41] [42] [43] [44] [45] [46] [47] [48] repomix-output.txt
file://file-Pbs4utuNJfVP1Biz71qzzG

[4] [8] [19] [20] [28] [29] [39] GitHub - midday-ai/midday: Invoicing, Time tracking, File reconciliation, Storage, Financial Overview & your own Assistant made for Freelancers
https://github.com/midday-ai/midday

[5] [14] [15] [17] [21] [22] [24] Updates | Midday
https://midday.ai/updates/