Purpose

This campaign manager is a purpose-built narrative operating system for tabletop role-playing games, designed to replace the need for DMs to force general-purpose note tools like Obsidian into workflows they were never designed for. Instead of relying on loosely structured markdown files and manually maintained templates, it provides typed campaign objects (such as NPCs and locations), dynamic views, and a fast inbox for capturing raw session notes without friction. DMs can freely write in markdown while selectively extracting people, places, and ideas into structured entities, enabling reliable queries, relationships, and future automation. The system prioritizes long-term campaign memory, non-destructive evolution, and DM control—supporting improvisation first, structure second—so preparation enhances play rather than becoming a maintenance burden.

Core Principles

- Improvisation First, Structure Second.
  - Raw notes can be captured quickly without forced categorization
- Typed Objects, Not Templates.
  - Campaign elements are structured entities rendered through views. Not copied markdown templates.
- Non-Destructive Evolution.
  - Changing schemas, fields, or views updates all entities automatically without manual rewrites.
- Markdown Where It Matters.
  - Freeform writing remains markdown-based.
- DM Control and Trust.
  - The DM decides what becomes canon and/or visible.
- Long-Term Campaign Memory
  - The system should remain usable across multiple multi-year campaigns without decay.

User-Facing Features

Quick Notes

- Markdown. Allows for fast, low-friction capture of unstructured ideas or notes
- Open / Closed / Archived / Pinned states
- Optional Note Titles

Quick Notes Inbox View

- Centralized inbox for all Quick Notes
- Designed for review, triage, and gradual organization rather than immediate structuring
- Notes can be filtered and sorted by state, creation time, or last activity
- Supports extracting people, places, and ideas directly from note content without altering the original note

Quick Notes Extraction

- Highlight text within Quick Note and extract to Typed Object
- Non-destructive to original Quick Note
- Extracted entity links back to source note for context
- Extraction workflow:
    - User highlights text (e.g., "John the Blacksmith, a human fighter")
    - System suggests entity type (NPC)
    - System pre-populates fields from highlighted text
    - User reviews and completes additional fields
    - Entity created with backref to source note
- Batch extraction:
    - Extract multiple entities from single note
    - System detects wikilink patterns to suggest entities
    - Preview all extractions before confirming

Typed Objects

- Campaign elements (NPCs, locations, plots, etc.) are structured entities with:
    - System-defined core fields - Required fields for consistency and querying
    - User-defined custom fields - DMs can add their own fields specific to their world
    - Markdown body - Long-form narrative content, notes, and improvisation
- Available Typed Object Types
    - NPCs, Locations, Plots, Encounters, Factions, Lore, Items

Views, Not Templates

- Each Typed Object is rendered using configurable views.
- Views control layout and emphasis without modifying stored data
- Future support for multiple views per object (compact vs detailed)

Markdown with First-Class Linking

- Wiki-style links between entities
- Mentions tracked separately from semantic relationships
- Notes remain readable outside the system

Deterministic Queries

- Queries are code-driven and deterministic to ensure reliability and debuggability.
    - Show me locations related to “npc_name”
    - Name all NPCs linked to “faction_name”
    - List all currently active Plots
- Queries operate only on structured data and explicit relationships

Generators

- Various Generators for things like:
    - Magic Shop Inventories

    - Tavern Generators
    - NPC Generators
- Allow for contextual World State Inputs (City, location, NPCs)
- AI utilized for flavor text

Note Sharing

- Allow Players to view notes depending on visibility settings in individual Typed Objects or Notes
- Allow Players or Fellow DMs to add or edit notes
    - Settings to require validation from Campaign Owner, Any DM, or None.

- Exportable data - allow for saving locally.

Timeline System

- Entities Valid from and to dates
    - Archived Entities listed separately

Search

- Ability to do a full-text search across all notes and objects

Import/Export

- Pluggable importer architecture - each source (Obsidian, World Anvil, etc.) gets its own import adapter
- Obsidian Migration
    - Phase 1: Content Import
        - Import all .md files to Quick Notes. Potentially Create an Import Folder in Inbox.
        - Preserve wikilinks as plain text initially
    - Phase 2: Assisted Extraction
        - UI to review imported notes
        - Suggest entities based on wikilink patterns
        - Batch Extraction with preview
        - DM confirms/tweaks field mappings
    - Phase 3: Template Mapping
        - If you detect common templates
        - Offer to map lines to core or user-defined fields
        - Show Preview
- World Anvil
    - Similar Phased Approach
    - Map their types: Article -> Lore, Character -> NPC, Organization -> Faction

Mobile Access

- Access and Create Notes from multiple devices.

Calendar System

- Users can define their own month names, days per month, etc.
- Calendar stored as configuration: JSON definition with month/day structure

Data Model

Typed Objects

- Core System Fields
  - id: GUID-based unique identifier
  - name: Display name
  - created_at: Timestamp of creation
  - updated_at: Timestamp of last modification
  - created_by_actor_id: Which user/DM created this
  - last_modified_by_actor_id: Which user/DM last edited this
  - body: Markdown content for long-form notes
- Campaign Scoping Fields
  - campaign-id : Primary campaign this entity belongs to
  - Is_global: Boolean - if true, visible to all campaigns in world
  - Linked_campaign_ids - List of specific campaigns with explicit access
- Temporal Fields
  - Valid_from and valid_to - nullable dates
- Visibility Fields
  - Visibility - Enum:
    - Private: Only campaign owner can see
    - DMOnly: All DMs in campaign can see
    - Public: Players can see based on campaign setting
- Type Specific Core Fields
  - NPC
    - Name
    - Race
    - Class
  - Location
    - Name
    - Parent_location_id
    - Location_type
  - Plot
    - Name
    - Status
- Custom Field Types. Support the following types:
  - Text
  - LongText
  - Number
  - Boolean
  - Date
  - InGameDate

    - EntityReference
    - EntityReferenceList
    - Currency
    - DiceFormula
    - Measurement

Field Definitions

- Fields are defined per Typed Object not copied via Templates
- Adding or Changing a field automatically updates all existing entities
    - Optional request for a default value to be applied to older entities, if needed
- Entity Fields
    - campaign_id: The primary campaign it belongs to
    - is_global: Boolean - if true, visible to all campaigns
    - linked_campaign_ids: List of specific campaigns with access
    - Valid_from and valid_to - Nullable date

Relationships vs Mentions

- Relationships - Structured, can be filtered by type, used for queries
- Mentions - Generated from markdown content of an entity.
    - Discovery tool “This entity is mentioned in….”

Actor Roles & Permissions

- Three core actor roles:
    - World Owner: Creator the world, ultimate authority, can manage all campaigns
    - DM (Campaign-level): Runs specific campaign(s), full edit access within their campaign
    - Player: View-only by default, can be granted limited edit rights
- Players inherit view permissions from campaign visibility settings
- World Owner can override default permissions per campaign
- Every entity tracks created_by_actor_id and last_modified_by_actor_id
- Every data access requires both campaign_id and actor_id

Relationship Types

- Relationships are typed and directional
- Core relationship types:
    - Location Relationships
        - ParentLocation (City->Nation, Business->City, Cave->Region, etc)
    - NPC Relationships
        - MemberOf (NPC -> Faction)
        - EmployedBy (NPC -> Location/NPC)
        - AlliedWith (NPC -> NPC/Faction)
        - EnemyOF (NPC -> NPC/Faction)
    - Plot Relationships
        - InvolvedIn (NPC -> Plot)
        - OccursIn (Plot -> Location)
    - Generic

        - Related (catch-all for undefined relationships)

Technical Architecture

Storage & Backup

- Storage abstraction layer: All backup operations go through an interface, not directly to filesystem or cloud provider
- Content-addressable media storage. Files named by their hash rather than original filename.
- Relative file paths only. Never store absolute paths
- Use GUIDs instead of auto-incrementing integers for all entity IDs - enables merging campaigns and cross-instance portability
- Backups are complete archives (database + media) that can be restored independently
- All backups encrypted at rest
- Each backup includes schema version number for future migrations
- Support for the Fresh Campaigns (no shared entities) and Linked Campaigns (Global Entities already linked)

API Architecture

- All business logic lives in backend API, not in UI code
- Design auth to support both session cookies (web) and bearer tokens (mobile) from the start
- Each request contains all needed context (campaignId, userId) - no server-side session dependence
- API can return different detail levels (compact/full) based on client needs
- Store thumbnails, medium, and full-size versions of all images
- API versioning from day one (/api/v1/)
- Stateless design: Server stores no session state between requests
- Consistent error handling: Standard error response format across all endpoints

Performance Principles

- Pagination. When searching results only load N number of results at a time. (1-50 of 857)
- Lazy Loading. Entities are not loaded until requested by the user.
- Cache frequently-read, rarely-changed data (view definitions, custom field schemas)
- Cache invalidation: When definitions change, clear related cache
- Database Query Performance
    - Database Indexing
        - Composite indexes for common query combinations
    - Always include campaign_id in WHERE clause
    - Add database query timeouts
- Performance Testing:
    - Create seed scripts for realistic data volumes

Version Control / Audit Trail

- Every update logged automatically, no user action required
- Track which specific fields changed, not just "entity updated"

- Before/after values for each field modification
- Rolling window of detailed history
- Extended audit trail accessible.
- For recent changes, single action to restore previous value
- Every change links to actor who made it
- Multiple field changes in one save operation grouped together
- Audit logs older than 1 year moved to archive table for performance
- Markdown body edits will just show the body is modified. And store snapshots of before and after

Offline First Strategy

- Desktop app maintains local database
- All writes go to local database immediately
- Background sync to server when online
- Offline operations queued and synced when connection restored
- Conflict resolution strategy:
  - Last-write-wins based on timestamp (initial implementation)
  - Manual conflict resolution UI for concurrent edits
  - Only applies when multiple users edit same entity

Technologies

- C#/.NET 10
- EF ORM
- Postgres
- Docker
- Next.js
- TypeScript
- Tailwind CSS

Implementation Roadmap

Slice 1: Quick Notes + Inbox

Phase 1.1: Minimal Quick Note

Goal Create a single Quick Note and save it to a database

1. Set up .NET project structure
   a. Create solution with API and Domain projects
   b. Add EF Core, Postgres NuGet packages
2. Create QuickNote entity
3. Create database context and initial migration
4. Create POST endpoint
5. Create GET endpoint
6. Test with Postman/Swagger
7. Set up Next.js project with TypeScript
8. Create simple form: textarea + “Save” button with tailwind CSS
9. Call API to save note

10. Display Saved Note

Phase 1.2: Note States

Goal: Add Open/Closed/Pinned/Archived states

1. Add State enum to Quicknote
2. Create migration to add state column
3. Update POST endpoint to accept initial state
4. Create PATCH endpoints
5. Add state selector to form
6. Add Change State button to note view
7. Show visual indicator of state (color, icon, badge)

Phase 1.3: Optional Titles

Goal: Add optional Title Field

1. Add nullable Title string to QuickNote
2. Create migration
3. Update endpoints to handle title
4. Add optional title input above body textarea
5. Display title prominently when present
6. Show first line of body as preview when no title

Phase 1.4: Inbox List View

Goal: See all notes in a list

1. Create GET endpoint to return list of all notes
2. Add basic pagination
3. Create Inbox page/component
4. Fetch and display lists of notes
5. Show each note as a card/row
6. Add pagination controls

Phase 1.5: Filtering and Sorting

Goal: Filter by state, sort by date

1. Update GET endpoint to accept filters
2. Implement filtering logic (WHERE clauses)
3. Implement sorting logic (ORDER BY clauses)
4. Add filter dropdown
5. Add sort dropdown
6. Update URL query params when filters change
7. Persist filter/sort preferences in localStorage

Phase 1.6: Full CRUDD Operations

Goal: Complete Create, Read, Update, Delete

1. Create PUT endpoint
2. Create DELETE endpoint
3. Update ‘UpdatedAt’ timestamp automatically on edit
4. Remove Save button

5. Add auto-save with debouncing
6. Add “Saving..” / “Saved” indicator
7. Make title and body editable
8. Add Delete button with confirmation dialog
9. Return to inbox after delete

Phase 1.7: Markdown Rendering

Goal: Render Markdown properly

1. Install Markdown library
2. Add markdown preview/rendering for body
3. Add basic markdown toolbar (bold, italic, lists)
4. Style rendered markdown with Tailwind

Slice 2: Typed Objects + Extraction

Phase 2.1: NPC Entity Foundation

Goal: Create basic NPC entity and CRUD

1. Create NPC entity
2. Create Migration
3. Create NPC endpoints
    a. POST
    b. GET
    c. GET by ID
    d. PUT by ID
    e. DELETE by ID
4. Create NPCs page/route
5. Create NPC list view
6. Create NPC form
7. Create NPC detail view

Phase 2.2: Location Entity

Goal: Add Location entity (without hierarchy yet)

1. Create Location entity
2. Create migration
3. Create Location Endpoints
    a. Same as NPC
4. Create Locations Page
5. Reuse list/form/detail components from NPC

Phase 2.3: Basic Extraction - Manual

Goal: Extract text from Quick Note to create NPC

1. Create Extraction tracking entity
2. Create POST endpoint
3. Create NPC with provided data
4. Add Extract button in Quick Note View
5. Highlight text selection in note body

6. Show extraction dialog:
    a. Radio buttons: Create NPC or Create Location
    b. Name Input
    c. Body Input
7. Call extraction endpoint
8. Show success message with link to new entity

Phase 2.4: Extraction Backlinks

Goal: Link extracted entities back to source note

1. Add SourceEntityId and SourceEntityType to NPC and Location entities
2. Update extraction endpoint to set this field
3. Update GET NPC/Location endpoints to include source note info
4. In NPC/Location detail view, show “Extracted from” section
5. Display source Quick Note Title or preview
6. Add link to jump back to source note
7. In quick note view show extracted entities list

Phase 2.5: Entity Type Abstraction

Goal: Start thinking about shared entity patterns

1. Create BaseEntity abstract class
2. Make NPC and Location inherit from BaseEntity
3. Create Shared repository pattern
4. Create shared EntityList component
5. Create shareEntityDetail component
6. Parameterize by entity type

Slice 3: Search

Slice 4: Custom Fields

Slice 5: Relationships

Slice 6: Mentions/Wikilinks

Slice 7: More Typed Objects

Slice 8: Views System

Slice 9: Media Foundation

Slice 10: Import/Export

Slice 11: Campaign Management

Slice 12: API Hardening

Slice 13: Authentication & User Accounts

Slice 14: Cloud Backup

Slice 15: Actor Roles & Permissions

Slice 16: Visibility & Sharing

Slice 17: Audit Trail UI

Slice 18: Collaborative Editing

Slice 19: Timeline System

Slice 20: Calendar System

Slice 21: Generators

Slice 22: CQL

Slice 23: Offline-First Architecture

Slice 24: Mobile App