CogniLink - Progress Report

Xihao Yang

December 4, 2025

Author: Xihao Yang
Date: December 4, 2025

Executive Summary

CogniLink is a robust, client-side bookmark and knowledge management system designed to run entirely within the browser using IndexedDB for data persistence. The application allows users to organize, search, and manage bookmarks through a modern interface without requiring server-side dependencies.

While the core architecture and search mechanisms are functional, the project is currently in the refinement phase, focusing on UI/UX enrichment and data persistence stability.

1 Project Overview

1.1 Original Vision

The goal was to create a “Second Brain” application for browser bookmarks—a tool that goes beyond simple link storage by offering intelligent organization, search capabilities, and local privacy. The system was designed to be offline-first, ensuring zero latency and full user control over data.

1.2 Current Status

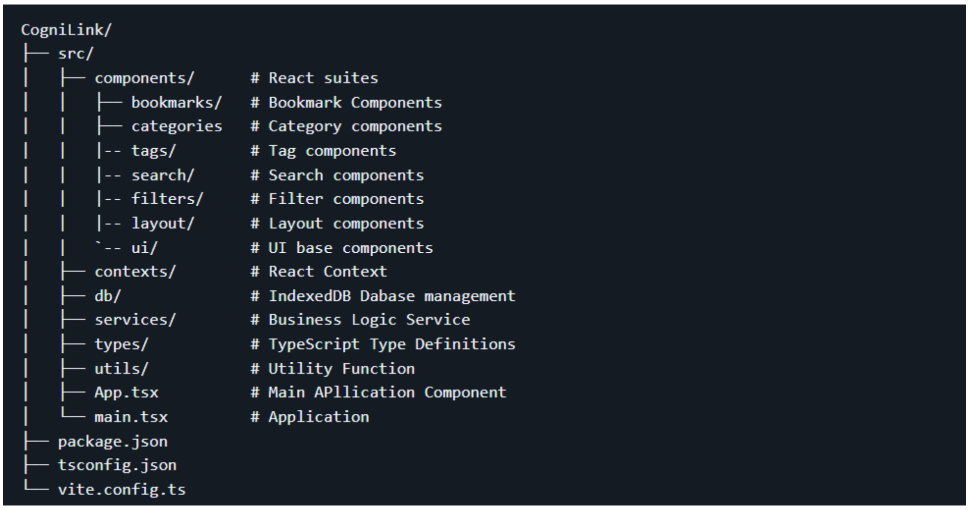
The project has achieved MVP (Minimum Viable Product) status with core modules operational. Current development is focused on ensuring data reliability across sessions and enhancing the visual interface.

- **Data Persistence:** Functional IndexedDB wrapper (Optimization in progress).
- **Search Engine:** Custom client-side search with relevance scoring.
- **Organization:** Hierarchical categories and a dynamic tagging system.
- **User Interface:** Responsive layout implemented; visual polishing ongoing.

2 Technical Implementation

2.1 Architecture

The application follows a clean, component-based architecture with a clear separation of concerns. The project directory structure is outlined below:



```
CogniLink/
├── src/
│   ├── components/      # React suites
│   │   ├── bookmarks/   # Bookmark Components
│   │   ├── categories   # Category components
│   │   |-- tags/        # Tag components
│   │   |-- search/      # Search components
│   │   |-- filters/     # Filter components
│   │   |-- layout/      # Layout components
│   │   `-- ui/          # UI base components
│   ├── contexts/        # React Context
│   ├── db/              # IndexedDB Dabase management
│   ├── services/        # Business Logic Service
│   ├── types/           # TypeScript Type Definitions
│   ├── utils/           # Utility Function
│   ├── App.tsx          # Main APllication Component
│   └── main.tsx         # Application
├── package.json
├── tsconfig.json
└── vite.config.ts
```

Figure 1: CogniLink Project Directory Structure

Key directories include:

- `src/components/`: React UI components (30+) organized by feature (bookmarks, categories, tags).
- `src/contexts/`: State management using React Context API to bridge services and UI.
- `src/services/`: Business logic layer (BookmarkService, CategoryService) handling data manipulation.
- `src/db/`: Low-level database operations interacting directly with the browser's IndexedDB.

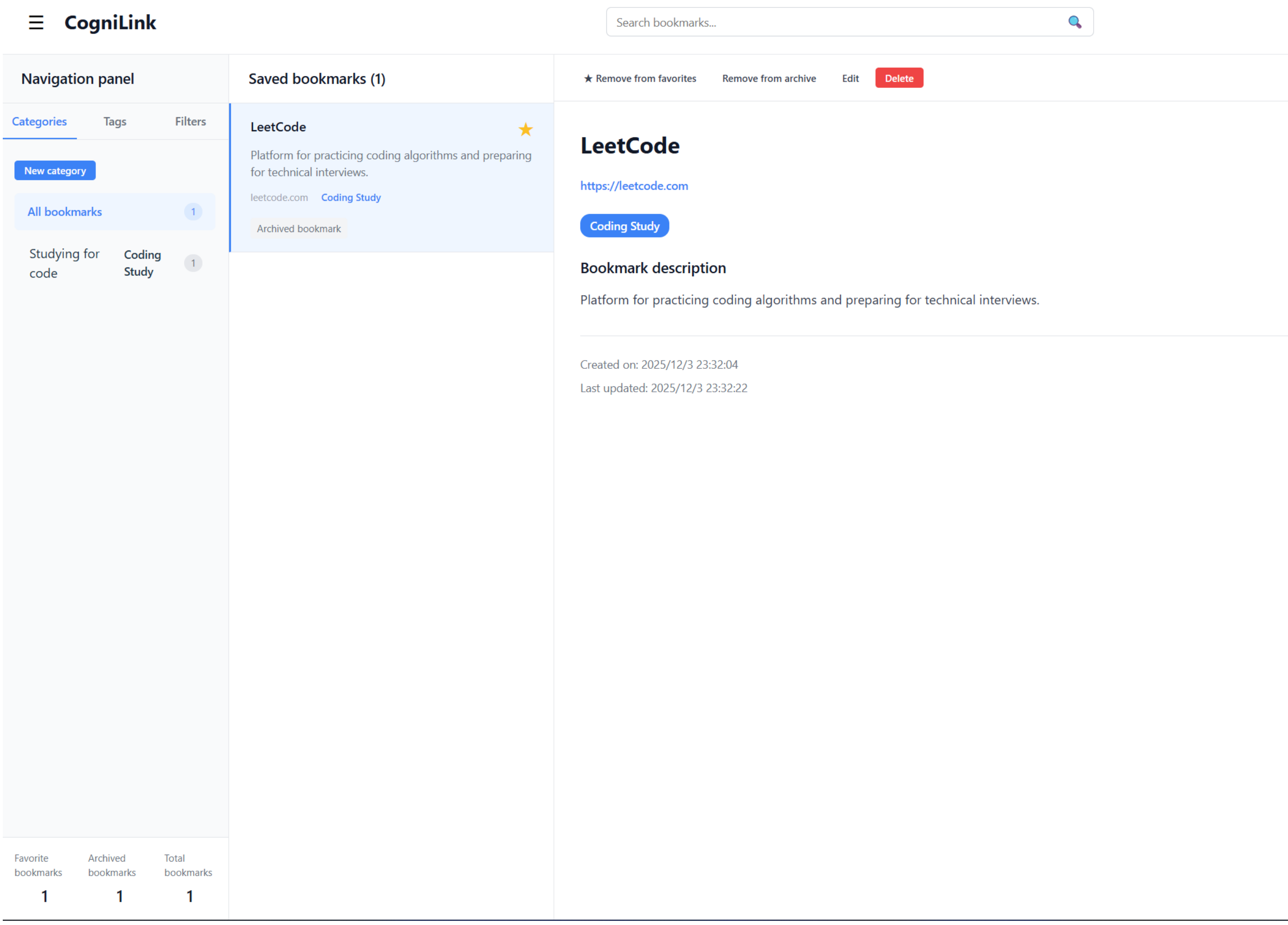
2.2 Technology Stack

Component	Technology	Purpose
Framework	React 18+	User Interface construction
Language	TypeScript	Type safety and application logic
Build Tool	Vite	Fast development server and bundling
Storage	IndexedDB	Local client-side data persistence
Search	Custom Algorithm	Inverted index with TF-IDF scoring

2.3 User Interface Experience

The UI is fully componentized and responsive, designed to maximize information density while maintaining clarity.

Main Dashboard The application features a three-column layout to facilitate quick navigation and management. As shown in Figure 2, the left sidebar provides navigation between Categories, Tags, and Filters. The middle column acts as a list view for selected filters, while the right column displays the detailed view of the selected bookmark, including metadata, descriptions, and action buttons (Edit, Delete, Archive).



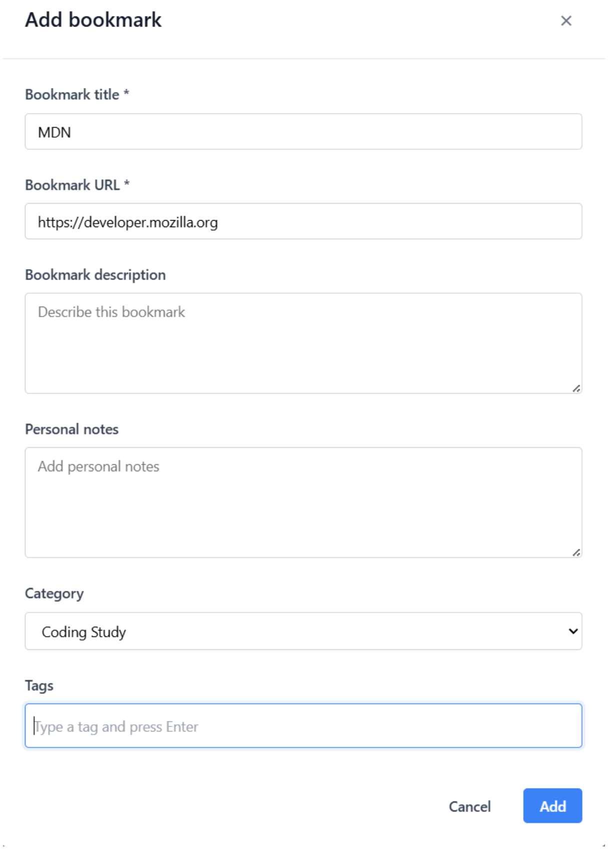
Platform for practicing coding algorithms and preparing for technical interviews.

Created on: 2025/12/3 23:32:04

Last updated: 2025/12/3 23:32:22

Figure 2: CogniLink Main Dashboard. Features a split-view design for efficient browsing.

Data Entry Integration To ensure data quality, the "Add Bookmark" interface utilizes a modal overlay (Figure 3). This form enforces validation on required fields (Title, URL) and provides rich categorization options. Users can assign a primary Category and multiple Tags simultaneously.



Add bookmark

Bookmark title *

MDN

Bookmark URL *

https://developer.mozilla.org

Bookmark description

Personal notes

Category

Coding Study

Tags

Figure 3: Bookmark Creation Modal. Supports categories, tags, and rich text descriptions.

3 Current Metrics

3.1 Code Statistics

Metric	Value
TypeScript Files	30+
Total Lines of Code	~4,500+
React Components	30+
Database Tables	3 (Bookmarks, Categories, Tags)

3.2 Feature Completeness

Feature	Status	Completion
Bookmark CRUD	Complete	100%
Tagging System	Complete	100%
Full-text Search	Complete	100%
Favorites & Archiving	Complete	100%
Persistence Reliability	**In Progress**	**85%** (Stability fixes needed)
UI/UX Polish	**In Progress**	**75%** (Visual enrichment needed)
Data Import/Export	**In Progress**	**60%** (Backend ready, UI missing)

Table 1: *
Source: Internal Project Tracking

4 Next Steps

The focus for the next iteration is to transition from a functional MVP to a polished, stable product.

4.1 System Stability & Functionality

1. **Persistence Reliability:** Fix issues regarding data state rehydration upon application restart. Ensure that the database connection is robustly handled in the `useEffect` hooks to prevent any potential data access errors when reloading the page.

2. **Favicon Fetching:** Implement a utility to automatically fetch and cache icons for bookmarked URLs to improve visual recognition in the list view.

3. **Batch Operations:** Implement the ability to select multiple bookmarks for bulk deletion or moving to a category.

4.2 UI/UX Optimization

- **Visual Enrichment:** The current interface is functional but minimalist. The next phase involves adding decorative illustrations, refined empty states, and clearer iconography to make the application feel more "alive" and professional.
- **Interactive Feedback:** Add toast notifications for success/error states (e.g., "Bookmark Saved Successfully") to provide better system feedback.
- **Responsive Refinements:** Further optimize the sidebar and modal behavior for smaller browser windows.

5 Conclusion

CogniLink has successfully implemented the core logic for a client-side knowledge management system. The search algorithm and database schema are functioning as expected. The immediate next steps are crucial: solving the data persistence reliability on restart and significantly upgrading the user interface with richer visuals and smoother interactions. These improvements will ensure the application is not only functional but also reliable and enjoyable to use.