BMaestro

Cross-Browser Bookmark Management System

Technical Specification Document

Version 1.0 | December 2025

1. Executive Summary

BMaestro is a comprehensive bookmark management system that enables centralized, programmatic control of bookmarks across Chrome, Brave, and Edge browsers. The system provides natural language control via Claude AI, a web-based dashboard for monitoring and manual operations, and robust synchronization capabilities with full audit trails and rollback functionality.

Key Capabilities:

- Unified bookmark management across Chrome, Brave, and Edge browsers
- Natural language control through Claude AI integration (MCP Server)
- Real-time synchronization with conflict resolution
- Web dashboard for monitoring, manual operations, and sync oversight
- Graveyard system for backup and rollback of original bookmarks
- PocketBase database for persistent state, history, and audit logging

2. System Architecture

2.1 High-Level Architecture

The system consists of six primary components that communicate through well-defined interfaces:

Component Stack (Top to Bottom):

- User Interface Layer: Voice/text input via Claude, Web Dashboard (SvelteKit)
- AI Layer: Claude with MCP Server integration
- Application Layer: BMaestro Core Server (Node.js)
- Data Layer: PocketBase database, Graveyard storage
- Communication Layer: Native Messaging Host
- Browser Layer: Chrome, Brave, Edge extensions using chrome.bookmarks API

2.2 Data Flow

User commands flow through the system as follows: User input (voice/text/dashboard) reaches Claude or the web interface. Claude interprets natural language and calls MCP tools. The MCP Server validates commands and writes to PocketBase. The Core Server routes commands to the Native Messaging Host. The Native Messaging Host forwards to the appropriate browser extension. The extension executes chrome.bookmarks API calls. Results propagate back through the chain with status updates stored in PocketBase.

2.3 Technology Stack

Component	Technology
MCP Server	Node.js 20+, TypeScript, @anthropic-ai/sdk
Core Server	Node.js 20+, Express.js, TypeScript
Web Dashboard	SvelteKit 2.x, Tailwind CSS 3.x, TypeScript
Database	PocketBase 0.22+ (SQLite-backed)
Native Host	Node.js compiled to .exe via pkg
Browser Extension	Manifest V3, Chrome APIs, TypeScript
IPC	Named pipes (Windows), Unix sockets (macOS/Linux)

3. Component Specifications

3.1 Browser Extension

A single codebase produces three extension builds (Chrome, Brave, Edge) with browser-specific manifest files. The extension acts as a thin wrapper around the chrome.bookmarks API, receiving commands via Native Messaging and returning results.

Manifest Permissions:

```
{ "permissions": ["bookmarks", "nativeMessaging"], "background": {
"service_worker": "background.js" } }
```

Supported Operations:

Operation	API Method	Description
search	chrome.bookmarks.search()	Find by title/URL pattern
getTree	chrome.bookmarks.getTree()	Full bookmark hierarchy
getChildren	chrome.bookmarks.getChildren()	List folder contents
get	chrome.bookmarks.get()	Single bookmark by ID
create	chrome.bookmarks.create()	Add bookmark with position
update	chrome.bookmarks.update()	Modify title/URL
move	chrome.bookmarks.move()	Change parent/position
remove	chrome.bookmarks.remove()	Delete single bookmark
removeTree	chrome.bookmarks.removeTree()	Delete folder recursively

Message Protocol:
All messages use JSON with a standard envelope format. Requests include action, params, requestId, and timestamp. Responses include success, data/error, requestId, duration, and browser identifier.

3.2 Native Messaging Host

The Native Messaging Host bridges the Core Server and browser extensions. It is compiled to a standalone executable using pkg (no Node.js runtime required on user machines).

Installation Locations:

- Windows: %LOCALAPPDATA%\BMaestro\bmaestro-host.exe
- macOS: ~/Library/Application Support/BMaestro/bmaestro-host
- Linux: ~/.local/share/BMaestro/bmaestro-host

Registry/Manifest Registration:
Each browser requires a native messaging manifest registered in its configuration. On Windows, this involves registry keys under HKCU\Software\<Browser>\NativeMessagingHosts\com.bmaestro.host pointing to a JSON manifest file that specifies the executable path and allowed extension IDs.

3.3 Core Server

The Core Server is the central orchestrator. It manages communication with browsers, maintains state in PocketBase, exposes REST APIs for the dashboard, and coordinates synchronization operations.

Server Configuration:

- Default port: 3847 (configurable via BMAESTRO_PORT environment variable)
- IPC socket: Named pipe \\.\pipe\bmaestro (Windows) or /tmp/bmaestro.sock (Unix)

- PocketBase URL: http://127.0.0.1:8090 (configurable)

3.4 MCP Server

The MCP (Model Context Protocol) Server exposes BMaestro functionality as Claude-callable tools. It communicates with the Core Server via HTTP and translates natural language-oriented tool calls into specific bookmark operations.

Available Tools:

Tool	Description
bookmark_search	Search bookmarks by title/URL across specified browsers
bookmark_add	Create bookmark with folder path, position, optional sync to other browsers
bookmark_move	Relocate bookmark to different folder and/or position
bookmark_rename	Update bookmark title
bookmark_delete	Remove bookmark (requires confirmation flow)
bookmark_list_folder	List all bookmarks in a folder with metadata
bookmark_sort_folder	Sort folder contents by title/URL/date
bookmark_bulk_rename	Apply transformation (titleCase, lowercase, etc.) to folder
bookmark_sync	Synchronize from source browser to targets
bookmark_get_tree	Get full or partial bookmark tree structure
sync_status	Get current sync state, last sync times, pending operations
graveyard_list	List archived bookmark snapshots
graveyard_restore	Restore bookmarks from a graveyard snapshot

4. Database Schema (PocketBase)

PocketBase provides an embedded SQLite database with a REST API, real-time subscriptions, and built-in authentication. All BMaestro state is persisted here for durability and queryability.

4.1 Collections

browsers

Tracks registered browser instances and their connection state.

Field	Type	Description
id	string	PocketBase auto-generated ID
name	string	Browser identifier: chrome, brave, edge
profile	string	Profile name (default: 'Default')
extension_id	string	Installed extension ID
is_connected	boolean	Current connection status
is_canonical	boolean	Whether this is the source-of-truth browser
last_seen	datetime	Last successful communication timestamp
last_sync	datetime	Last successful sync completion
bookmark_count	number	Cached count of bookmarks
version	string	Browser version string

bookmarks

Cached representation of bookmarks across all browsers. Used for quick lookups and diff calculations.

Field	Type	Description
id	string	PocketBase auto-generated ID
browser_id	relation	FK to browsers collection
native_id	string	Browser's internal bookmark ID
parent_native_id	string	Parent folder's native ID
title	string	Bookmark display title
url	string	Bookmark URL (null for folders)
is_folder	boolean	True if this is a folder
position	number	0-indexed position in parent
path	string	Full folder path: 'Bookmarks Bar/Dev/Projects'
date_added	datetime	Original creation timestamp
checksum	string	Hash of title+url for change detection

sync_operations

Log of all synchronization operations for audit and debugging.

Field	Type	Description
id	string	PocketBase auto-generated ID
operation_type	string	full_sync, incremental, merge, restore
source_browser	relation	FK to source browser
target_browsers	json	Array of target browser IDs
status	string	pending, running, completed, failed, partial

started_at	datetime	Operation start timestamp
completed_at	datetime	Operation completion timestamp
duration_ms	number	Total operation duration
items_processed	number	Count of bookmarks processed
items_created	number	Bookmarks created in targets
items_updated	number	Bookmarks updated in targets
items_deleted	number	Bookmarks removed from targets
errors	json	Array of error objects
graveyard_snapshot_id	relation	FK to pre-sync graveyard snapshot

graveyard_snapshots

Complete bookmark tree snapshots for rollback capability.

Field	Type	Description
id	string	PocketBase auto-generated ID
browser_id	relation	FK to browsers collection
snapshot_type	string	initial, pre_sync, manual, scheduled
created_at	datetime	Snapshot creation timestamp
bookmark_count	number	Number of bookmarks in snapshot
tree_data	json	Complete bookmark tree structure
checksum	string	Hash of tree_data for integrity
notes	text	Optional description/reason for snapshot
is_restorable	boolean	Whether this snapshot can be restored

operation_log

Granular log of every bookmark operation for complete audit trail.

Field	Type	Description
id	string	PocketBase auto-generated ID
browser_id	relation	FK to browsers collection
sync_operation_id	relation	FK to parent sync operation (if any)
action	string	create, update, move, delete
bookmark_native_id	string	Browser's bookmark ID
before_state	json	Bookmark state before operation
after_state	json	Bookmark state after operation
source	string	mcp, dashboard, sync, api
timestamp	datetime	Operation timestamp
duration_ms	number	Operation duration
success	boolean	Whether operation succeeded
error_message	text	Error details if failed

settings

System configuration and user preferences.

Field	Type	Description
id	string	PocketBase auto-generated ID
key	string	Setting key (unique)
value	json	Setting value
updated_at	datetime	Last modification timestamp

5. Graveyard System

The Graveyard is a critical safety feature that preserves original bookmark states before any destructive or bulk operations. It enables full rollback capability and provides peace of mind during initial setup and ongoing synchronization.

5.1 Snapshot Types

Type	Description & Trigger
initial	Captured when a browser is first registered with BMaestro. This is the pristine state before any BMaestro operations. Always preserved and marked as restorable.
pre_sync	Automatically created before any sync operation. Allows rollback if sync produces unexpected results. Retained for 30 days by default.
manual	User-initiated via dashboard or MCP tool. Created on demand when user wants to save current state. No automatic expiration.
scheduled	Created by scheduled backup job. Default: daily at 3 AM. Retention: 7 daily, 4 weekly, 3 monthly.

5.2 Storage Strategy

Snapshots store the complete bookmark tree as JSON within PocketBase. For typical bookmark collections (500-2000 bookmarks), this results in 50KB-200KB per snapshot. With compression and the retention policy, storage requirements are minimal.

Retention Policy (Configurable):

- Initial snapshots: Never deleted automatically
- Pre-sync snapshots: 30 days retention
- Manual snapshots: No automatic deletion
- Scheduled snapshots: 7 daily + 4 weekly + 3 monthly (grandfather-father-son rotation)

5.3 Restore Process

Restoring from a graveyard snapshot is a multi-step process designed to be safe and reversible:

1. Create a new pre_sync snapshot of current state (safety backup)
2. Validate snapshot integrity via checksum verification
3. Generate diff between current state and snapshot
4. Present diff summary to user for confirmation
5. Execute restore operations in batches with progress reporting
6. Verify final state matches snapshot
7. Log complete operation history

5.4 Per-Browser Preservation

Each browser maintains its own graveyard snapshots independently. This is critical because different browsers may have different bookmark histories, and users may want to restore one browser without affecting others. The initial snapshot for each browser captures its unique state before BMaestro integration.

6. Synchronization System

6.1 Canonical Browser Concept

One browser is designated as the 'canonical' source of truth. During initial setup, users select their primary browser (typically the one with the most complete/curated bookmark collection). This browser's bookmarks become the authoritative set that gets propagated to other browsers.

Canonical Browser Selection Criteria:

- Contains the most complete bookmark collection
- Has the most organized folder structure
- Is the user's primary daily browser
- Has the most recent/curated bookmarks

6.2 Initial Merge Strategy

When BMaestro is first installed and configured with multiple browsers, the initial merge requires careful handling to avoid data loss while achieving a unified bookmark set.

Phase 1: Discovery & Snapshot

1. Connect to all registered browsers
2. Pull complete bookmark trees from each browser
3. Create 'initial' graveyard snapshot for each browser
4. Calculate statistics: total bookmarks, unique URLs, folder structures

Phase 2: Analysis & Diffing

1. Identify bookmarks unique to each browser (by URL)
2. Identify bookmarks common across browsers (URL match, possibly different titles)
3. Identify structural differences (same bookmarks in different folders)
4. Flag potential conflicts for user review

Phase 3: Merge Decision

User selects merge strategy via dashboard:

Strategy	Behavior
Canonical Only	Use only canonical browser's bookmarks. Others are replaced entirely.
Canonical + Unique	Start with canonical, then add bookmarks unique to other browsers into a designated folder (e.g., 'Imported from Brave').
Full Merge	Combine all bookmarks, canonical browser structure takes precedence for duplicates.
Manual Review	Generate full diff report, user manually approves each change category.

Phase 4: Execution

1. Create pre_sync snapshot of each target browser
2. Execute merge operations in batches (see Performance section)
3. Update PocketBase bookmark cache
4. Report results with detailed statistics

6.3 Ongoing Synchronization

After initial merge, BMaestro maintains sync through two mechanisms:

Push Sync (Immediate):
When a bookmark operation is performed via MCP or dashboard, BMaestro immediately propagates the change to all registered browsers. This ensures instant consistency for user-initiated actions.

Pull Sync (Periodic):
Every 5 minutes (configurable), BMaestro polls each browser for changes made outside BMaestro (e.g., user manually added a bookmark in Chrome). Detected changes from the canonical browser are propagated to others. Changes in non-canonical browsers trigger a conflict review queue.

6.4 Conflict Resolution

Conflict Type	Resolution Strategy
Same URL, different title	Use canonical browser's title, log discrepancy
Same title, different URL	Keep both as separate bookmarks, flag for review
Bookmark exists in non-canonical only	Add to canonical (if auto-merge enabled) or queue for review
Bookmark deleted in canonical	Delete from all browsers
Bookmark deleted in non-canonical	Queue for review (was it intentional?)
Position/folder changed in both	Canonical wins, log change

7. Performance & Speed

7.1 Performance Targets

Operation	Target Latency
Single bookmark create/update/delete	< 100ms per browser
Bookmark search (by title/URL)	< 50ms from cache, < 200ms live
Folder listing (< 100 items)	< 100ms
Full tree retrieval	< 500ms per browser
Bulk rename (100 bookmarks)	< 5 seconds
Full sync (1000 bookmarks)	< 30 seconds
Initial merge (3 browsers, 1000 each)	< 2 minutes

7.2 Optimization Strategies

Batching:
Multiple operations are batched to reduce round-trips. For bulk operations (sort, rename all, sync), commands are grouped into batches of 50 operations maximum. Each batch is sent as a single message to the extension, which processes them sequentially and returns a batch result.

Caching:
PocketBase maintains a complete cache of all browser bookmarks. Search operations query the cache first. Cache invalidation occurs on any write operation or periodic sync. Cache TTL: 5 minutes for folder listings, 1 hour for full tree (unless invalidated).

Parallel Execution:
When propagating to multiple browsers, operations are executed in parallel (Promise.all). Each browser processes independently. Failure in one browser doesn't block others. Results are aggregated with per-browser status.

Incremental Sync:
Rather than syncing entire trees, BMaestro tracks change timestamps and checksums. Only modified bookmarks since last sync are processed. This reduces typical sync operations from O(n) to O(changes).

7.3 Rate Limiting

To prevent overwhelming browsers, BMaestro implements rate limiting:

- Maximum 100 operations per second per browser
- Batch size limit: 50 operations per batch
- Concurrent browser connections: 3 maximum
- Retry backoff: exponential (100ms, 200ms, 400ms, 800ms, 1600ms)

8. Error Handling

8.1 Error Categories

Category	Code Range	Examples
Connection	1xxx	Browser not running, extension not installed, native host unreachable
Validation	2xxx	Invalid URL, empty title, folder path not found
Operation	3xxx	Bookmark not found, permission denied, duplicate exists
Sync	4xxx	Conflict detected, timeout, partial failure
System	5xxx	Database error, disk full, memory exhausted

8.2 Error Response Format

All errors follow a consistent JSON structure with the following fields: code (numeric error code), category (error category string), message (human-readable description), details (additional context object), recoverable (boolean indicating if retry might succeed), suggested_action (guidance for resolution), and timestamp.

8.3 Common Errors & Resolutions

Code	Error	Resolution
1001	Browser not connected	Ensure browser is running; reinstall extension if persists
1002	Extension not responding	Restart browser; check extension is enabled
1003	Native host timeout	Restart BMaestro service; check system resources
2001	Invalid bookmark URL	Verify URL format; must include protocol
2002	Folder path not found	Check folder exists; use exact path spelling
3001	Bookmark not found	Bookmark may have been deleted; refresh and retry
3002	Duplicate bookmark	Bookmark with same URL exists in folder
4001	Sync conflict	Review conflict in dashboard; choose resolution
4002	Partial sync failure	Some browsers failed; check individual statuses
5001	Database unreachable	Restart PocketBase; check disk space

8.4 Retry Strategy

For recoverable errors, BMaestro implements automatic retry with exponential backoff. Maximum 5 retry attempts are made. The initial delay is 100ms with a multiplier of 2x.

Maximum delay caps at 5 seconds. For operations marked critical (sync, bulk operations), failed attempts are queued for manual review rather than silently failing.

9. Web Dashboard

9.1 Technology Stack

The dashboard is built with SvelteKit 2.x for the application framework, providing SSR, routing, and API endpoints. Tailwind CSS 3.x handles styling with utility-first CSS. TypeScript ensures type safety throughout the codebase. PocketBase SDK enables real-time data subscriptions. The application is served on localhost:3848 by default.

9.2 Dashboard Pages

Route	Description
/	Overview dashboard: browser status cards, recent activity, quick actions
/browsers	Browser management: connection status, set canonical, reconnect
/bookmarks	Bookmark explorer: tree view, search, filter by browser, bulk operations
/sync	Sync control center: trigger sync, view history, resolve conflicts
/graveyard	Backup management: view snapshots, compare, restore
/logs	Operation logs: filterable activity stream, export capability
/settings	Configuration: sync intervals, retention policies, notifications

9.3 Real-Time Updates

The dashboard uses PocketBase's real-time subscriptions to receive instant updates. Browser connection status changes appear immediately. Sync progress updates in real-time with progress bars. Operation log streams as operations occur. Bookmark changes reflect across all open dashboard instances.

9.4 Key UI Components

Browser Status Cards:
Visual cards for each registered browser showing connection status (green/yellow/red indicator), last sync timestamp, bookmark count, and quick action buttons (sync now, view bookmarks, disconnect).

Bookmark Tree Explorer:
Collapsible tree view of bookmarks with drag-and-drop reorganization, inline editing of titles, multi-select for bulk operations, and browser filter tabs.

Sync Timeline:
Visual timeline of sync operations with expandable details, success/failure indicators, and rollback buttons for recent syncs.

Conflict Resolution Modal:
Side-by-side comparison of conflicting bookmarks with action buttons: keep canonical, keep other, keep both, merge titles.

10. REST API Endpoints

The Core Server exposes a REST API for dashboard and external integrations. Base URL: http://localhost:3847/api/v1

10.1 Browser Endpoints

Method	Endpoint	Description
GET	/browsers	List all registered browsers with status
GET	/browsers/:id	Get single browser details
POST	/browsers/:id/ping	Test connection to browser
PUT	/browsers/:id/canonical	Set browser as canonical
DELETE	/browsers/:id	Unregister browser

10.2 Bookmark Endpoints

Method	Endpoint	Description
GET	/bookmarks	Search/list bookmarks (query params: q, browser, folder)
GET	/bookmarks/tree	Get full bookmark tree for browser
POST	/bookmarks	Create new bookmark
PUT	/bookmarks/:id	Update bookmark
DELETE	/bookmarks/:id	Delete bookmark
POST	/bookmarks/:id/move	Move bookmark to new location
POST	/bookmarks/bulk	Bulk operations (rename, move, delete)

10.3 Sync Endpoints

Method	Endpoint	Description
POST	/sync	Trigger sync operation
GET	/sync/status	Get current sync status
GET	/sync/history	List past sync operations
GET	/sync/conflicts	List unresolved conflicts
POST	/sync/conflicts/:id/resolve	Resolve a conflict

10.4 Graveyard Endpoints

Method	Endpoint	Description
GET	/graveyard	List snapshots (query: browser, type, date range)
GET	/graveyard/:id	Get snapshot details
POST	/graveyard	Create manual snapshot
POST	/graveyard/:id/restore	Restore from snapshot
GET	/graveyard/:id/diff	Diff snapshot against current state
DELETE	/graveyard/:id	Delete snapshot (if allowed)

11. Installation & Setup

11.1 Prerequisites

- Node.js 20.x or later
- One or more Chromium browsers: Chrome, Brave, and/or Edge
- Windows 10/11, macOS 12+, or Linux (Ubuntu 20.04+)
- Administrator/sudo access for native host registration

11.2 Installation Steps

1. Run installer: npx @bmaestro/install
2. Follow prompts to select browsers for integration
3. Load extensions in each browser when prompted (extensions page opened automatically)
4. Grant bookmark permissions when prompted by each browser
5. Complete initial setup wizard in dashboard (opens automatically)
6. Select canonical browser and initial merge strategy
7. Review and confirm initial sync

11.3 Claude Desktop Integration

To enable Claude control, add the following to your Claude Desktop configuration file (claude_desktop_config.json):

```
{ "mcpServers": { "bmaestro": { "command": "node", "args": ["<install-path>/mcp-server.js"] } } }
```

11.4 File Locations

Component	Windows Path
Installation	%LOCALAPPDATA%\BMaestro\
PocketBase Data	%LOCALAPPDATA%\BMaestro\data\pb_data\
Logs	%LOCALAPPDATA%\BMaestro\logs\
Native Host	%LOCALAPPDATA%\BMaestro\bmaestro-host.exe
Config	%LOCALAPPDATA%\BMaestro\config.json

12. Project Structure

```
bmaestro/ ├── package.json ├── tsconfig.json ├── pocketbase/                    #
PocketBase binary and data │   ├── pocketbase.exe │   └── pb_data/ ├── src/ │
├── core-server/              # Main orchestration server │   │   ├── index.ts │
│   ├── routes/ │   │   │   ├── browsers.ts │   │   │   ├── bookmarks.ts │   │   │
├── sync.ts │   │   │   └── graveyard.ts │   │   ├── services/ │   │   │   ├──
browser-manager.ts │   │   │   ├── sync-engine.ts │   │   │   ├── graveyard.ts │
│   │   └── conflict-resolver.ts │   │   └── lib/ │   │       ├── pocketbase.ts │
│       └── native-messaging.ts │   │ │   ├── mcp-server/               # Claude
MCP integration │   │   ├── index.ts │   │   ├── tools/ │   │   │   ├── bookmark-
search.ts │   │   │   ├── bookmark-add.ts │   │   │   ├── bookmark-move.ts │   │
│   ├── bookmark-rename.ts │   │   │   ├── bookmark-delete.ts │   │   │   ├──
bookmark-list.ts │   │   │   ├── bookmark-sort.ts │   │   │   ├── bookmark-bulk-
rename.ts │   │   │   ├── bookmark-sync.ts │   │   │   ├── sync-status.ts │   │
│   ├── graveyard-list.ts │   │   │   └── graveyard-restore.ts │   │   └── core-
client.ts │   │ │   ├── native-host/               # Native messaging bridge │   │
├── index.ts │   │   ├── message-protocol.ts │   │   └── browser-router.ts │   │ │
├── extension/                # Browser extension │   │   ├──
manifest.chrome.json │   │   ├── manifest.brave.json │   │   ├──
manifest.edge.json │   │   ├── background.ts │   │   └── bookmark-api.ts │   │ │
└── dashboard/                # SvelteKit dashboard │       ├── src/ │       │
├── routes/ │       │   │   ├── +page.svelte │       │   │   ├── browsers/ │
│   │   ├── bookmarks/ │       │   │   ├── sync/ │       │   │   ├── graveyard/ │
│   │   ├── logs/ │       │   │   └── settings/ │       │   ├── lib/ │       │   │
├── components/ │       │   │   ├── stores/ │       │   │   └── api.ts │       │
└── app.css │       ├── tailwind.config.js │       └── svelte.config.js │ ├──
scripts/ │   ├── install.ts │   ├── register-native-host.ts │   └── build-
extension.ts │ └── dist/
```

13. Future Enhancements

13.1 Planned Features

- Multi-profile support: Manage bookmarks across multiple browser profiles
- Firefox support: Extend to Firefox via its WebExtensions API
- Safari support: macOS Safari integration via Safari App Extensions
- Mobile sync: Sync to Chrome/Edge mobile via signed-in profiles
- Bookmark health check: Detect and report dead/broken links
- Smart folders: Virtual folders based on rules (e.g., 'Recently Added', 'News Sites')
- Tagging system: Add tags to bookmarks for cross-folder organization
- Import/Export: Support for Netscape HTML format, JSON export
- Chrome Web Store publication: Signed extension distribution
- Cloud backup: Optional encrypted backup to cloud storage

13.2 Known Limitations

- Browser must be running for real-time operations
- Favicons are not synced (browser-managed separately)
- Only 'Default' profile supported in v1.0
- Extension requires manual installation (not via store in v1.0)
- No mobile browser support in v1.0

14. Appendix

14.1 Glossary

Term	Definition
Canonical Browser	The designated source-of-truth browser whose bookmarks take precedence in sync conflicts
Graveyard	The backup system storing snapshots of bookmark states for rollback
MCP	Model Context Protocol - Anthropic's standard for Claude tool integration
Native Messaging	Chrome's mechanism for extensions to communicate with local executables
Snapshot	A complete capture of a browser's bookmark tree at a point in time
Sync Operation	The process of reconciling bookmarks between browsers

14.2 References

- Chrome Bookmarks API: https://developer.chrome.com/docs/extensions/reference/bookmarks/
- Chrome Native Messaging: https://developer.chrome.com/docs/apps/nativeMessaging/
- PocketBase Documentation: https://pocketbase.io/docs/
- SvelteKit Documentation: https://kit.svelte.dev/docs
- MCP Specification: https://github.com/anthropics/anthropic-cookbook

— End of Specification —