JioInstitute

Browser Tab Manager

PGP in Artificial Intelligence & Data Science

Rishita Merchant
Sanskriti Dhar
Sneha Shinde

I. Introduction

With increasing multitasking needs, users often open dozens of tabs while working, studying, or browsing. Over time, this leads to a cluttered browser, consuming more memory and slowing down performance. Users rarely go back and manually close inactive tabs, leading to distractions and inefficient workflows.

Browser Tab Manager is a smart system that automatically keeps the **five most recently used tabs** open and closes all others. It works silently in the background using a **Chrome Extension** for tab tracking and a **FastAPI backend** that applies the **Least Recently Used (LRU)** algorithm with heapq.

This tool is designed to reduce memory load, improve focus, and ensure that only actively used tabs remain open - all without requiring user input or manual rules.

II. Objectives

- **Automated Tab Cleanup**
  Inactive tabs are automatically closed every 15 seconds, ensuring the browser remains clutter-free without requiring manual intervention. This periodic action is based on actual user activity, not static timers.
- **Real-Time Usage Tracking**
  The system listens for tab switches and focus changes, updating each tab's timestamp instantly. This allows the extension to reflect real usage patterns and ensures accurate tracking of tab activity.
- **True LRU Implementation**
  The backend uses a Least Recently Used (LRU) algorithm implemented with a min-heap (heapq) that prioritizes actual usage over tab open time. This ensures that the tabs most relevant to the user are retained.

- **Top 5 Tabs Always Preserved**
  Only the five most recently accessed tabs are kept active at any time. All others are considered for closure if inactive, minimizing memory usage while preserving current workflows.
- **No Manual Configuration Required**
  The entire process is fully automatic. Users are not required to manually mark important tabs, configure keywords, or set exceptions — ensuring a seamless and intelligent experience.

III. System Architecture

The project is divided into two main components:

1. Frontend: Chrome Extension

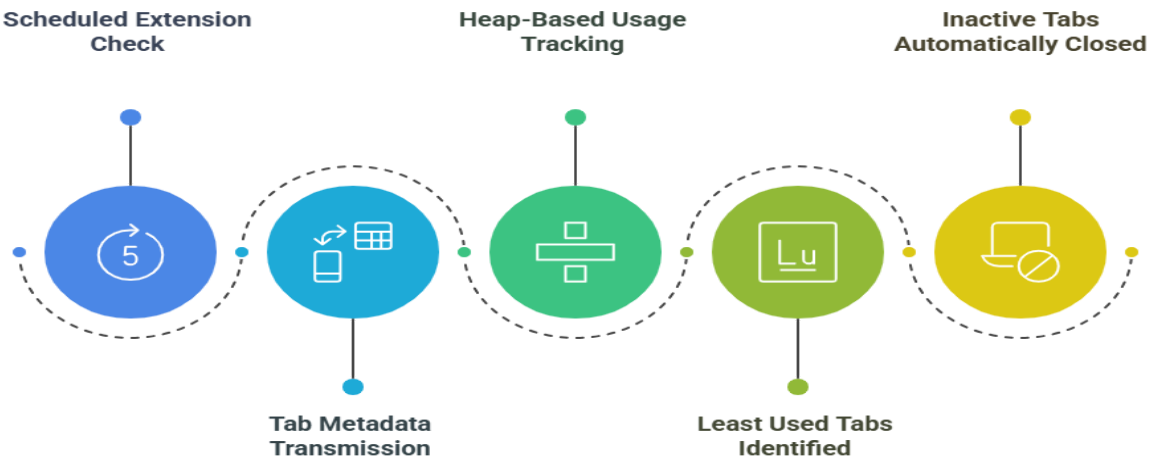
- **Tab Activity Monitoring**: Injects a content script into tabs that listens to activity such as focus or navigation.
- **Timestamp Update**: Updates the last accessed time for each tab using background scripts.
- **Data Transmission**: Periodically (e.g., every 5 seconds), this data is sent to the backend for analysis via HTTP requests.

2. Backend: FastAPI Server

- **Metadata Storage**: Receives real-time tab metadata and stores it in:
  - An in-memory dictionary (tab_cache) mapping tab IDs to last_seen timestamps.
  - A heapq-based min-heap that helps efficiently identify the least recently used tab.

- **LRU Eviction Logic**: Every 15 seconds, the backend:
  - Pops from the heap to find the oldest tab (least recently used).
  - Validates the timestamp with tab_cache to avoid evicting recently updated tabs.
  - Sends back a list of inactive tab IDs that should be closed.

- **Frontend Sync**: The extension receives these IDs and closes them using chrome.tabs.remove().



IV. LRU Algorithm Using heapq

1. **Timestamp Tracking**
   - Each tab update is captured with a last_seen timestamp, reflecting the most recent access time.
   - This timestamp is stored in an in-memory dictionary (tab_cache) using the format {tab_id: last_seen}.
     Simultaneously, each update is pushed into a min-heap (heapq) as a tuple: (last_seen, tab_id).
   - This setup ensures both fast lookup (via dictionary) and efficient ordering by recency (via heap).

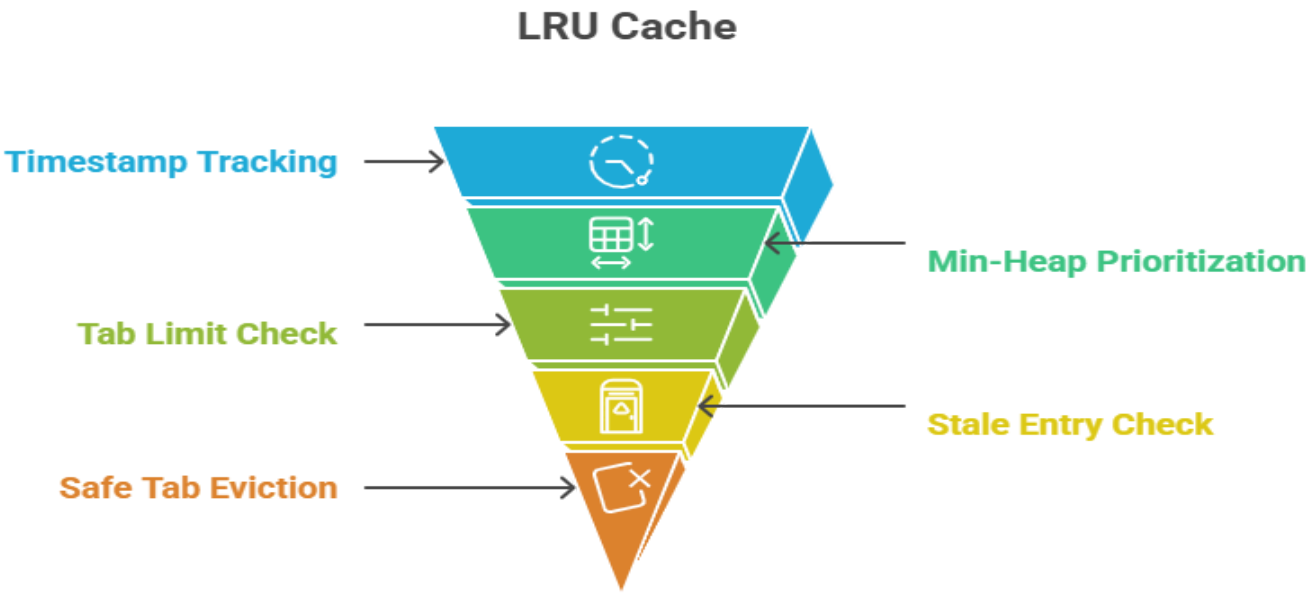
2. **Min-Heap Prioritization**
   - The heapq min-heap maintains tab usage order based on the last_seen value.
   - Tabs are stored as tuples: (last_access_time, tab_id) to prioritize the least recently used.
   - The min-heap ensures the oldest (least recently used) tab is always at the top for quick eviction.
   - On every tab update, the heap is rebuilt with fresh timestamps to reflect the most current usage state.

3. **Tab Limit Check & Trigger**
   - A predefined constant, MAX_CACHE_SIZE, limits the number of active tabs retained.
   - When this threshold is exceeded, the backend initiates the eviction cycle.

- The backend begins popping entries from the min-heap, starting from the least recently used tabs.
- Only the **top 5 most recently used** tabs are retained and protected from eviction.

4. **Stale Entry Check**
   - Because the heap may contain outdated entries (due to multiple updates for the same tab), validation is essential before removal.
   - For each popped tab, its last_seen timestamp is cross-verified with the latest value in tab_cache.
   - If the timestamps match, the tab is truly inactive; otherwise, it's skipped as a stale heap entry.

5. **Safe Tab Eviction**
   - Verified inactive tabs are safely deleted from the tab_cache.
   - Their tab_id is sent back to the frontend for closure.
   - The Chrome Extension uses chrome.tabs.remove() to automatically close these inactive tabs.
   - This ensures only genuinely unused tabs are closed, without affecting the user's current activity.



V. Key Features

- **Smart Tab Retention**
  Keeps only the 5 most recently used tabs open by continuously tracking tab activity. This helps maintain an uncluttered workspace without the need for manual tab management.
- **Fully Automatic Operation**
  The extension works without any manual configuration or user interaction. Tabs are tracked, evaluated, and closed automatically at regular intervals, ensuring minimal user effort.
- **Real-Time Activity Monitoring**
  Unlike traditional methods that rely solely on creation time, this solution uses actual usage patterns such as tab focus and interaction to determine recency.
- **Efficient LRU Management**
  Implements a memory-efficient Least Recently Used (LRU) logic using Python's heapq module, ensuring fast and optimized operations even with many tabs.
- **Enhanced Focus & Performance**
  By closing unused tabs automatically, the extension reduces distractions and system memory consumption, leading to better productivity and performance.

VI. Technologies Used

1. **Backend (FastAPI)**
   - Python 3
   - FastAPI
   - `heapq` (for priority-based LRU logic)
2. **Frontend**
   - JavaScript (background script)
   - Chrome Extension Manifest V3
3. **Communication**
   - REST API
   - JSON for data transfer

VII. Challenges Faced

- **Accurate Event Tracking**
  Precisely capturing tab switches, focus, and blur events in real-time was critical. It required fine-tuning the content and background scripts to ensure every user action was logged without delay or duplication.
- **Efficient Resource Management**
  Preventing memory leaks and avoiding redundant or unnecessary API calls was essential to keep the extension lightweight. Debouncing update triggers and cleaning up unused event listeners played a key role in this optimization.
- **Scalable Backend Logic**
  Ensuring the backend remained efficient and scalable, even with frequent tab updates, involved using in-memory data structures (like dictionaries and heaps) with constant or logarithmic time operations, reducing CPU and memory overhead.
- **Robust Testing with Tab Dynamics**
  The system was stress-tested with scenarios involving rapid tab creation and closure. The goal was to guarantee no tab update was missed, and the LRU logic continued functioning correctly under pressure.

VIII. Results

1. Tab Management Operations Overview

Operation	Data Structure	Time Complexity	Notes
Add/update tab entry	Dictionary (dict)	O(1)	Fast key-based update
Push to heap	Min-Heap (heapq)	O(log n)	Maintains LRU order
Pop LRU tab	Min-Heap (heapq)	O(log n)	Removes oldest tab
Validate timestamp	Dictionary (dict)	O(1)	Avoids stale eviction
Remove from cache	Dictionary (dict)	O(1)	Constant-time deletion

2. Observed Outcomes & Benefits

- Only the 5 most recently used tabs were successfully maintained at any time, ensuring optimal relevance and minimizing distractions.
- Automatic closure of inactive tabs every 15 seconds eliminated the need for manual tab management.

- Improved focus and reduced browser memory usage during long working sessions were noticeable.
- The FastAPI backend efficiently handled rapid tab switches, demonstrating the scalability of the architecture even under high interaction rates.

IX. Future Improvements

- **Tab Whitelisting**:
  Option to exclude essential tabs (e.g., YouTube, Gmail) from automatic closure.
- **Activity Dashboard**:
  Display currently open and recently closed tabs through a simple UI.
- **Custom Tab Limit**:
  Let users define how many tabs to retain (e.g., 3, 5, 10).
- **Multi-Profile & Mobile Support**:
  Extend compatibility to multiple browser profiles and mobile browsers.

X. Conclusion

The **Browser Tab Manager** addresses a prevalent challenge faced by modern users — managing an overwhelming number of open tabs — with a clean, efficient, and automated solution. It smartly integrates a **Chrome Extension** and a lightweight **FastAPI backend**, leveraging **Least Recently Used (LRU) logic** to automatically track and close inactive tabs in real time.

This architecture ensures that only the most relevant tabs remain open, reducing memory consumption and improving overall browser performance. With minimal user interaction, it enhances **focus**, **reduces cognitive load**, and promotes a **smoother, distraction-free browsing experience**. By optimizing both frontend activity tracking and backend decision-making, the system offers a seamless, intelligent tab management solution tailored for productivity in the digital age.