Table of Contents

1. AIM
2. ABSTRACT
3. Project Vision & Philosophy
    1. The Problem: A Hostile Web
    2. The Solution: Persistent, Intelligent Blocking
    3. Core Philosophy: The Meaning of “Robust”
4. Target Audience & User Experience
    1. Defining the Ideal User
    2. Principles of User Experience
5. Proposed System
    1. Architectural Model: Manifest V3 Chrome Extension
    2. Core Components Diagram
    3. File and Directory Structure
    4. Technology Stack Rationale
6. Implementation Details & Development History
    1. Phase 1: Project Scaffolding
    2. Phase 2: Core Redirect Detection
    3. Phase 3: Iterative Bug Fixing & Refinement
    4. Codebase Analysis: manifest.json
    5. Codebase Analysis: background.js
    6. Codebase Analysis: src/APP.jsx
7. Future Roadmap & Feature Specifications
    1. Phase 1: ML-Powered Whitelisting
    2. Phase 2: AI-Powered Note-Taking
    3. Feature Analysis: Granular Cache Clearing
8. Testing & Quality Assurance Strategy
9. Conclusion
10. References

AIM

The primary aim of the “Robust Blocker” project is to provide a frustration-free browsing experience and a robust user experience for users navigating a hostile web environment.

This aim is realized through specific functional and strategic goals:

- **Intelligent Blocking:** The core feature implemented aims for the intelligent detection and blocking of unwanted cross-domain redirects.
- **Resilience:** The project seeks to be a dynamic and intelligent agent that is robust against evasion and wins the ongoing “cat-and-mouse” game against aggressive bypass techniques.
- **Simplicity:** The ultimate user experience goal is to offer a powerful solution with zero configuration and a “set it and forget it” principle.

ABSTRACT

Project Title: ROBUST BLOCKER

A Framework for a Persistent and Intelligent Redirect Blocking System

This report provides a comprehensive overview of the "Robust Blocker" project, a next-generation browser extension designed to provide a frustration-free browsing experience. The project is currently in the early development phase (version 0.1.0) and has successfully implemented a core feature: the intelligent detection and blocking of unwanted cross-domain redirects.The vision for Robust Blocker extends beyond simple blocking. It aims to win the ongoing "cat-and-mouse" game against websites that deploy aggressive and user-hostile bypass techniques. The project prioritizes a simple, "set it and forget it" user experience, targeting everyday users who need a powerful solution without complex configuration.The current implementation is a Manifest V3 Chrome extension built with React and modern JavaScript. It successfully identifies and acts upon malicious redirects by closing the tab and clearing the initiating URL from browser history, while intelligently ignoring safe, same-domain redirects.The future roadmap is ambitious, with plans to introduce a machine learning-powered whitelisting system that learns from user feedback. A visionary long-term goal includes expanding the extension into a general-purpose AI browsing assistant with note-taking capabilities.This document details the project's philosophy, current architecture, implementation history, and future plans, serving as a foundational guide for continued development.

2. Project Vision & Philosophy

2.1 The Problem: A Hostile Web

The modern web, while a source of infinite information and connectivity, has also become an increasingly hostile environment for users. The economic models driving much of the internet have led to a proliferation of user-hostile patterns designed to capture attention and revenue at the expense of the user experience. These include:

- **Aggressive Advertising:** Intrusive pop-ups, video ads, and banners that slow down page loads and obscure content.
- **Pervasive Trackers:** Invisible scripts that monitor user behavior across sites, building profiles without consent.
- **Malicious Redirects:** Unwanted navigation that takes users to phishing sites, malware domains, or ad-heavy pages without their permission.
- **Blocker Bypassing:** An increasing number of websites actively detect the presence of ad and content blockers, subsequently denying access to content or deploying even more aggressive techniques.

This creates a constant battle for users, forcing them to navigate a minefield of digital annoyances and security risks.

2.2 The Solution: Persistent, Intelligent Blocking

Robust Blocker is conceived as a direct response to this hostile environment. It is not merely another content blocker based on static filter lists. Its goal is to be a dynamic and intelligent agent acting on the user's behalf.

The project's strategy is to focus on a "hybrid" approach, tackling both traditional ad-blocking and complex redirect chains. The initial implementation focuses on the redirect problem, as it is often more jarring and potentially more dangerous than display advertising.

By focusing on the *behavior* of web requests (i.e., the act of redirection) rather than just the *content* (i.e., ad-related URLs), Robust Blocker aims to be more resilient to the obfuscation and randomization techniques used by bypass scripts.

2.3 Core Philosophy: The Meaning of "Robust"

The choice of the name "Robust Blocker" is intentional and central to the project's philosophy. In this context, "Robust" signifies **persistence and resilience**.

- **Persistent in Action:** The extension should be relentlessly effective. It should not fail silently. Its blocking and handling mechanisms are designed to be assertive, such as closing tabs and clearing history, ensuring an unwanted navigation path is completely terminated.
- **Resilient to Evasion:** The project's primary long-term goal is to be robust against evasion. This means investing development effort into smarter, behavior-based detection logic (like the current domain-checking) and future ML-powered models, rather than relying solely on community-maintained block lists which are inherently reactive. It must anticipate and defeat attempts to circumvent it.
- **Robust in Experience:** The ultimate goal is a robust user experience—one that is smooth, fast, and free from the frustration of a hostile web.

---

3. Target Audience & User Experience

3.1 Defining the Ideal User

The ideal user for Robust Blocker is a **non-technical or semi-technical everyday internet user**. This user profile can be characterized as follows:

- They are aware that the web is full of ads, pop-ups, and shady links, but they do not have the time, expertise, or desire to manage complex tools.
- They may have tried other blockers in the past but found them ineffective against certain sites or too complicated to configure.
- Their primary goal is to browse the web for work, leisure, or education without interruption.
- They value their time and mental energy and are easily frustrated by experiences that hijack their attention.

This focus on simplicity informs every design decision. The project explicitly avoids becoming a "power-user" tool with dozens of checkboxes, filter list subscriptions, and configuration panels.

32. Principles of User Experience

The user experience (UX) is governed by a single, simple principle: **"It Just Works."**

1. **Zero Configuration:** The ideal user should be able to install the extension and immediately receive its full benefits without any further setup. The intelligence should be built into the extension, not demanded from the user.

2. **Seamless & Invisible:** The best-case scenario is that the user forgets the extension is even running. It should operate silently in the background, cleaning up the browsing experience without requiring user input. Redirects should be handled so quickly and cleanly that the user is never even aware they were sent to an unwanted page.
3. **Trust through Effectiveness:** User trust is not built through complex dashboards showing how many items were blocked. It is built through the qualitative experience of a faster, cleaner, and less frustrating internet. The success of the UX is measured by the *absence* of frustration.

---

4. Proposed System

System Architecture & Technology Stack

4.1 Architectural Model: Manifest V3 Chrome Extension

Robust Blocker is built upon Google Chrome's modern extension platform, **Manifest V3**. This architecture was chosen for its focus on security, performance, and future-proofing.

Key characteristics of this architecture include:

- **Service Worker-Based Background Logic:** Instead of a persistent background page, Manifest V3 uses a Service Worker (background.js). This is an event-driven script that runs only when needed, consuming significantly fewer system resources. This aligns with our goal of a lightweight, non-intrusive extension. The main challenge of this model is managing state and ensuring event listeners are correctly registered, a hurdle that was successfully overcome during early development.
- **Declarative APIs:** Manifest V3 encourages the use of declarative APIs. While we are currently using the more dynamic chrome.webRequest API for its nuanced redirect information, the project was initially scoped with declarativeNetRequest in mind for future ad-blocking rules.
- **Strict Permissions Model:** Host permissions (<all_urls>) are explicitly separated from functional permissions (webRequest, history), providing greater transparency to the user during installation.

4.2 Core Components Diagram

```
graph TD
  subgraph User's Browser
    A[User Clicks Link] --> B{Browser Navigation};
    end

    subgraph Robust Blocker Extension
        B --> C{Event: onBeforeRedirect};
        C -- details --> D[background.js Service Worker];
        D --> E{Analyze URL Redirect};
        E -- Is Cross-Domain? --> F{Action};
        F --> G[API: chrome.history.deleteUrl];
        F --> H[API: chrome.tabs.remove];
    end

    G --> I_HIST[(Browser History)];
    H --> I_TAB[(Browser Tab)];

    style A fill:#fff,stroke:#333,stroke-width:2px
    style C fill:#f9f,stroke:#333,stroke-width:2px
```

4.3 File and Directory Structure

The project maintains a clean and scalable structure, managed by the Vite build tool.

```
/robust-blocker
|
|-- /dist/                # Build output, this is the folder loaded into Chrome
|
|-- /node_modules/        # Project dependencies
|
|-- /public/              # Static assets, copied directly to /dist
|   |-- manifest.json     # Core extension configuration
|   |-- background.js     # Background service worker script
|   `-- /images/          # Placeholder for extension icons
|
|-- /src/                 # React source code for the UI
|   |-- App.jsx           # Main React component
|   |-- main.jsx          # React entry point
|   `-- index.css         # UI styles
|
|-- index.html            # HTML entry point for the React popup
|-- package.json        # Project metadata and dependencies
`-- vite.config.js      # Vite build configuration
```

4.4 Technology Stack Rationale

- **JavaScript (ES6+):** The universal language of the web and browser extensions.
- **React:** Chosen for the popup UI for its component-based architecture, which makes building interactive and stateful user interfaces straightforward. While the current popup is simple, using React provides scalability for future UI features (like an options page or stats dashboard).
- **Vite:** A next-generation frontend tooling solution. It was chosen over older tools like Create React App for its significantly faster development server startup and build times, leading to a more efficient development cycle.
- **Pandoc:** Utilized for automated report generation, demonstrating a commitment to clear documentation and project management.

---

5. Implementation Details & Development History

The development of Robust Blocker has been an iterative process focused on building a stable core and methodically solving problems.

5.1 Phase 1: Project Scaffolding

The initial phase involved setting up a modern, robust development environment for a Manifest V3 extension.

- **Manual Scaffolding:** The `npm create vite` command initially failed due to issues with interactive prompts in the development environment. A more robust manual scaffolding approach was adopted, where `package.json`, `vite.config.js`, and all source files were created individually. This demonstrated a foundational understanding of the toolchain.
- **First Buildable Version:** The project structure was organized to separate source files (`/src`) from public assets (`/public`), with `vite.config.js` correctly configured to produce a loadable `dist` directory. An initial `index.html` placement issue was diagnosed and fixed, proving the build pipeline's correctness.

5.2 Phase 2: Core Redirect Detection

This phase implemented the primary feature of the extension.

- **API Selection:** The chrome.webRequest.onBeforeRedirect API was chosen as the ideal tool for this task. It provides the necessary information (url, redirectUrl, tabId) at the exact moment a server-side redirect occurs.
- **Initial Implementation:** The first version of the logic was simple: any detected redirect would trigger the closing of the associated tab (chrome.tabs.remove). This provided a baseline of functionality.
- **Adding History Clearing:** Based on user feedback, the functionality was evolved to also remove the initiating URL from the browser's history (chrome.history.deleteUrl), creating a cleaner digital footprint.

5.3 Phase 3: Iterative Bug Fixing & Refinement

This phase was critical in transforming the prototype from a proof-of-concept into a usable tool.

- **Bug 1: Background Script Not Loading:** A major bug was discovered where the background script would not run at all. Through methodical debugging, the issue was traced to a permission conflict in manifest.json between webRequest and declarativeNetRequest. Removing the unused declarativeNetRequest permission resolved the issue, demonstrating a key lesson in Manifest V3 development: permission conflicts can cause silent failures.
- **Bug 2: Overly Aggressive Blocking:** The initial logic closed tabs for *all* redirects, including safe, common ones like HTTP-to-HTTPS or example.com-to-www.example.com. This made the extension unusable.
- **The Fix:** The logic was significantly improved. The new implementation parses the source and destination URLs and compares their hostnames. To handle the www subdomain case, the hostnames are first normalized by removing any leading www.. The extension now only takes action if the normalized hostnames are different, successfully ignoring safe same-domain redirects.

5.4 Codebase Analysis: manifest.json

This file is the heart of the extension, defining its capabilities, permissions, and structure.

Code:

```
{
  "manifest_version": 3,
  "name": "Robust Blocker",
  "version": "0.1.0",
  "description": "A robust ad and tracker blocker.",
  "action": {
    "default_popup": "index.html"
  },
  "background": {
    "service_worker": "background.js"
  },
  "permissions": [
    "storage",
    "webRequest",
    "history"
  ],
  "host_permissions": [
    "<all_urls>"
  ]
}
```

Explanation: * "manifest_version": 3: Declares that the extension uses the modern, secure Manifest V3 platform. * "name", "version", "description": Standard metadata for the extension. * "action": Defines the UI for the toolbar icon. "default_popup": "index.html" specifies that when the user clicks the icon, the index.html file (which loads our React app) should be shown. * "background": Configures the background script. "service_worker": "background.js" tells Chrome to use this file as the event-driven service worker. * "permissions": An array of APIs the extension needs to access. * "storage": For saving settings in the future (e.g., the whitelist). * "webRequest": Crucially, allows the extension to observe and analyze network traffic. This is the foundation of our redirect detection. * "history": Allows the extension to remove URLs from the browser's history. * "host_permissions": In Manifest V3, this is required to grant the extension access to run on specific websites. "<all_urls>" gives it the ability to monitor redirects on any website the user visits.

5.5 Codebase Analysis: background.js

This script contains the core logic of the entire extension. It runs in the background and listens for events.

Code:

```
// This is the background service worker.
// It will handle the core blocking logic.
```

```
console.log("Robust Blocker background script loaded.");

// Listen for any web request that is about to be redirected.
chrome.webRequest.onBeforeRedirect.addListener(
  function(details) {
    try {
      const originalUrl = new URL(details.url);
      const redirectUrl = new URL(details.redirectUrl);

          // Normalize hostnames by removing 'www.' to treat 'www.example.com' and 'example.com' as the
same.
      const originalHost = originalUrl.hostname.replace(/^www\./, '');
      const redirectHost = redirectUrl.hostname.replace(/^www\./, '');

      // Only act if the normalized hostnames are different.
      if (originalHost !== redirectHost) {
        console.log(`Unwanted redirect detected from ${originalHost} to ${redirectHost}. Taking action.`);

        // Remove the original URL that initiated the redirect from history.
        chrome.history.deleteUrl({ url: details.url }, function() {
          if (chrome.runtime.lastError) {
                                                      console.warn(`Could not remove ${details.url} from history: $
{chrome.runtime.lastError.message}`);
          }
        });

        // Close the tab where the redirection occurred.
        chrome.tabs.remove(details.tabId);
      }
    } catch (e) {
      console.error("Robust Blocker: Could not parse URL.", e);
    }
  },
  {
    // Apply this listener to all URLs.
    urls: ["<all_urls>"]
  }
);
```

Explanation: * chrome.webRequest.onBeforeRedirect.addListener(...): This is the main event listener. It registers a callback function that will execute whenever the browser is about to follow an HTTP redirect. * function(details): The callback function receives a details object containing information about the redirect, such as the original URL (details.url) and the destination URL (details.redirectUrl). * new URL(details.url): The URLs are parsed into standard URL objects, which provides an easy and reliable way to access their components, like the hostname. * .hostname.replace(/^www\./, ''): This is the "smart filter." It gets the hostname (e.g., www.google.com) and uses a regular expression to remove the www. prefix, if it exists. This normalization is key to preventing the blocker from firing on safe, same-site redirects. * if (originalHost !== redirectHost): This conditional check is the core of the logic. It only proceeds if the normalized hostnames are different, indicating a true cross-domain redirect. * chrome.history.deleteUrl(...): If the redirect is deemed unwanted, this API call is made to remove the initiating URL from the user's browsing history. * chrome.tabs.remove(details.tabId): This API call closes the tab where the redirect is occurring, preventing the user from ever seeing the unwanted page. * { urls: ["<all_urls>"] }: This filter object tells the listener to fire for redirects happening on any URL, as granted by our host permissions.

5.6 Codebase Analysis: src/App.jsx

This file defines the simple React component for the extension's popup UI.

Code:

```
import React from 'react';

function App() {
  return (
    <div>
      <h1>Robust Blocker</h1>
      <p>Popup UI is active!</p>
    </div>
  )
}

export default App;
```

Explanation: * This is a standard, functional React component. * It currently returns a simple JSX structure with a heading and a paragraph. * This serves as the user-facing interface when the extension’s toolbar icon is clicked. While simple now, it is built with React to allow for easy expansion into a more complex UI with settings, statistics, and other controls in the future.

6. Future Roadmap & Feature Specifications

6.1 Phase 1: ML-Powered Whitelisting

This is the next planned feature, designed to enhance user control and build a data-driven intelligence layer.

- **User Story:** As a user, when the extension blocks a redirect, I want to be given the option to allow it once or permanently, so I can access the content if I trust the destination.
- **Proposed Architecture:**
    1. The onBeforeRedirect listener’s logic will be modified. Instead of immediately closing the tab, it will first check chrome.storage for a user-defined whitelist.
    2. If the destination is not on the whitelist, the extension will redirect the current tab to a built-in extension page (e.g., permission.html).
    3. The permission.html page will be a simple UI that displays the blocked destination and provides three options: “Go Back,” “Allow Once,” and “Allow Always.”
    4. “Allow Always” will add the destination’s hostname to the whitelist in chrome.storage.
- **Machine Learning Angle:** The choices made by users (“Allow” vs. “Block”) are valuable data. A future iteration will involve sending this anonymous data (e.g., source domain, destination domain, user choice) to a backend server. This data can be used to train a model that can predict whether a new, unseen redirect is malicious, eventually automating the process and minimizing the need for user prompts.

6.2 Phase 2: AI-Powered Note-Taking

This is a visionary, long-term goal to expand the extension’s purpose.

- **Concept:** Evolve Robust Blocker from a content blocker into a holistic “AI Browsing Assistant.”
- **Functionality:** While the exact features are yet to be defined, it could involve functionalities like:
    - Contextually summarizing articles.
    - Saving snippets of text or images to a cloud-synced notebook.
    - Providing AI-driven insights on the content being viewed.

- **Feasibility:** This represents a significant pivot and would require a backend infrastructure, user accounts, and integration with a large language model API.

63. Feature Analysis: Granular Cache Clearing

The desire to clear the cache for a single blocked domain was evaluated.

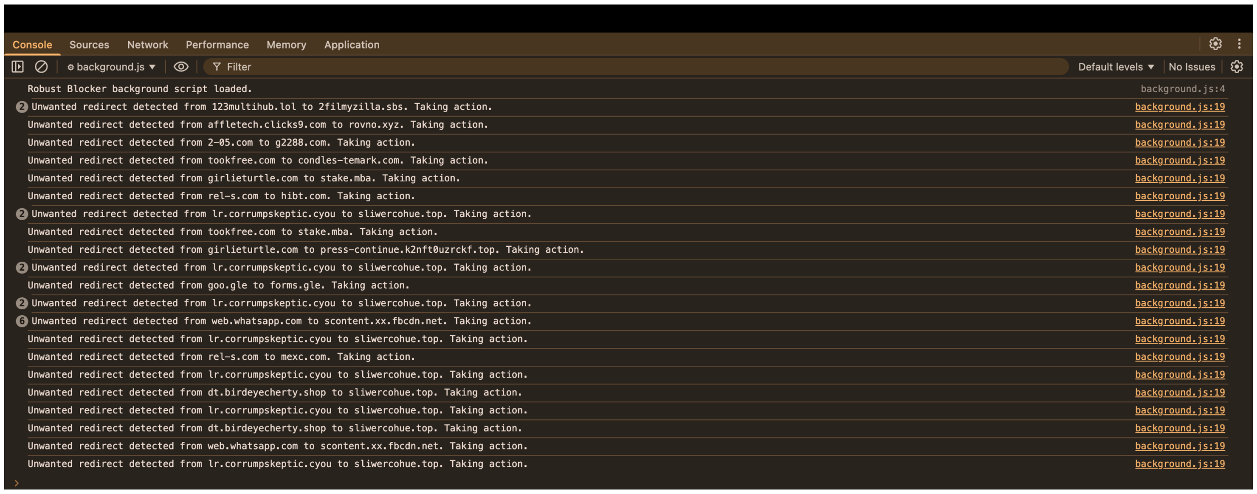
- **Goal:** To prevent a blocked page from storing any data in the user's browser cache.
- **Conclusion: This feature is currently not feasible.**
- **Rationale:** Browser extension APIs, specifically the chrome.browsingData API, are designed with user privacy and security in mind. They do not provide a mechanism to delete cache, cookies, or local storage for a single, specific URL or domain. The available API only allows clearing data for a specified time period (e.g., "last hour," "last 24 hours"), which would unacceptably affect all other websites the user has visited. Unless browser vendors introduce more granular data-clearing APIs, this feature will remain technically impossible to implement as envisioned.

7. Testing & Quality Assurance Strategy

To ensure the extension remains robust and reliable, a formal testing strategy is recommended.

- **Manual Testing:**
  - Maintain a list of test-case websites and redirect links.
  - Regularly test against different types of redirects: HTTP-to-HTTPS, www subdomains, URL shorteners, and known ad-related redirect chains.
  - Test for regressions after every significant code change.

console logs of a blocked redirect



```
  Robust Blocker background script loaded.                                                               background.js:4
2 Unwanted redirect detected from 123multihub.lol to 2filmyzilla.sbs. Taking action.                      background.js:19
  Unwanted redirect detected from affletech.clicks9.com to rovno.xyz. Taking action.                      background.js:19
  Unwanted redirect detected from 2-05.com to g2288.com. Taking action.                                   background.js:19
  Unwanted redirect detected from tookfree.com to condles-temark.com. Taking action.                      background.js:19
  Unwanted redirect detected from girlieturtle.com to stake.mba. Taking action.                           background.js:19
  Unwanted redirect detected from rel-s.com to hibt.com. Taking action.                                   background.js:19
2 Unwanted redirect detected from lr.corrumpskeptic.cyou to sliwercohue.top. Taking action.               background.js:19
  Unwanted redirect detected from tookfree.com to stake.mba. Taking action.                               background.js:19
  Unwanted redirect detected from girlieturtle.com to press-continue.k2nft0uzrckf.top. Taking action.     background.js:19
2 Unwanted redirect detected from lr.corrumpskeptic.cyou to sliwercohue.top. Taking action.               background.js:19
  Unwanted redirect detected from goo.gle to forms.gle. Taking action.                                    background.js:19
2 Unwanted redirect detected from lr.corrumpskeptic.cyou to sliwercohue.top. Taking action.               background.js:19
6 Unwanted redirect detected from web.whatsapp.com to scontent.xx.fbcdn.net. Taking action.               background.js:19
  Unwanted redirect detected from lr.corrumpskeptic.cyou to sliwercohue.top. Taking action.               background.js:19
  Unwanted redirect detected from rel-s.com to mexc.com. Taking action.                                   background.js:19
  Unwanted redirect detected from lr.corrumpskeptic.cyou to sliwercohue.top. Taking action.               background.js:19
  Unwanted redirect detected from dt.birdeyecherty.shop to sliwercohue.top. Taking action.                background.js:19
  Unwanted redirect detected from lr.corrumpskeptic.cyou to sliwercohue.top. Taking action.               background.js:19
  Unwanted redirect detected from dt.birdeyecherty.shop to sliwercohue.top. Taking action.                background.js:19
  Unwanted redirect detected from web.whatsapp.com to scontent.xx.fbcdn.net. Taking action.               background.js:19
  Unwanted redirect detected from lr.corrumpskeptic.cyou to sliwercohue.top. Taking action.               background.js:19
>
```

- **Automated Testing (Future):**
    - **Unit Tests:** Implement a testing framework like Vitest or Jest to write unit tests for individual functions, especially the URL parsing and comparison logic in background.js.
    - **End-to-End Tests:** A framework like Puppeteer or Playwright could be used to write automated browser tests. These tests would programmatically load the extension, navigate to test pages, click links, and assert that the correct blocking behavior occurs.

8. Conclusion

The "Robust Blocker" project has successfully moved from a concept to a functional prototype with a smart, core feature. It has a clear vision focused on user experience and a pragmatic, iterative development process. The challenges overcome in the initial phases have built a strong foundation and provided valuable lessons in Manifest V3 development.

The future roadmap, particularly the ML-powered whitelisting, presents a clear path to innovation in the content-blocking space. By prioritizing a stable core and methodically layering new, intelligent features, Robust Blocker is well-positioned to achieve its goal of providing a truly robust, frustration-free browsing experience.

9. References

The philosophy and goals of this project have been informed by an analysis of the current landscape of content blockers and browser technologies. Key points of reference include:

1. **Comet Browser:** An example of a browser with integrated, aggressive blocking features, serving as inspiration for a "hybrid" approach that goes beyond simple filter lists.
2. **Redirect Blocker (Chrome Extension):** An existing extension with a similar focus, providing a baseline for the core feature set and user expectations in this niche.
3. **Google Chrome Manifest V3 Documentation:** The official documentation has been a critical reference for understanding the architecture, capabilities, and limitations of the modern extension platform.