CHROMIUM - GOOGLE SUMMER OF CODE

# Debug WebUI For Tabstrip states


## Summary

Project Proposal Link

webui_for_tabstrip_states

*Tabstrip state and session states for browsers are complicated and rely on correct ordering of tabs in the tabstrip model, groups and sessions to restore tabs in the right position and selection state. As a result this has resulted in numerous bugs and it is often a pain point to figure out if it is a tabstrip model issue or a client issue. A webUI that captures the live state of the backend of the browser and tabstrip models will be really helpful in finding issues and edge cases.*

Owner: Koushik
Contributors: Koushik Kumar Bug

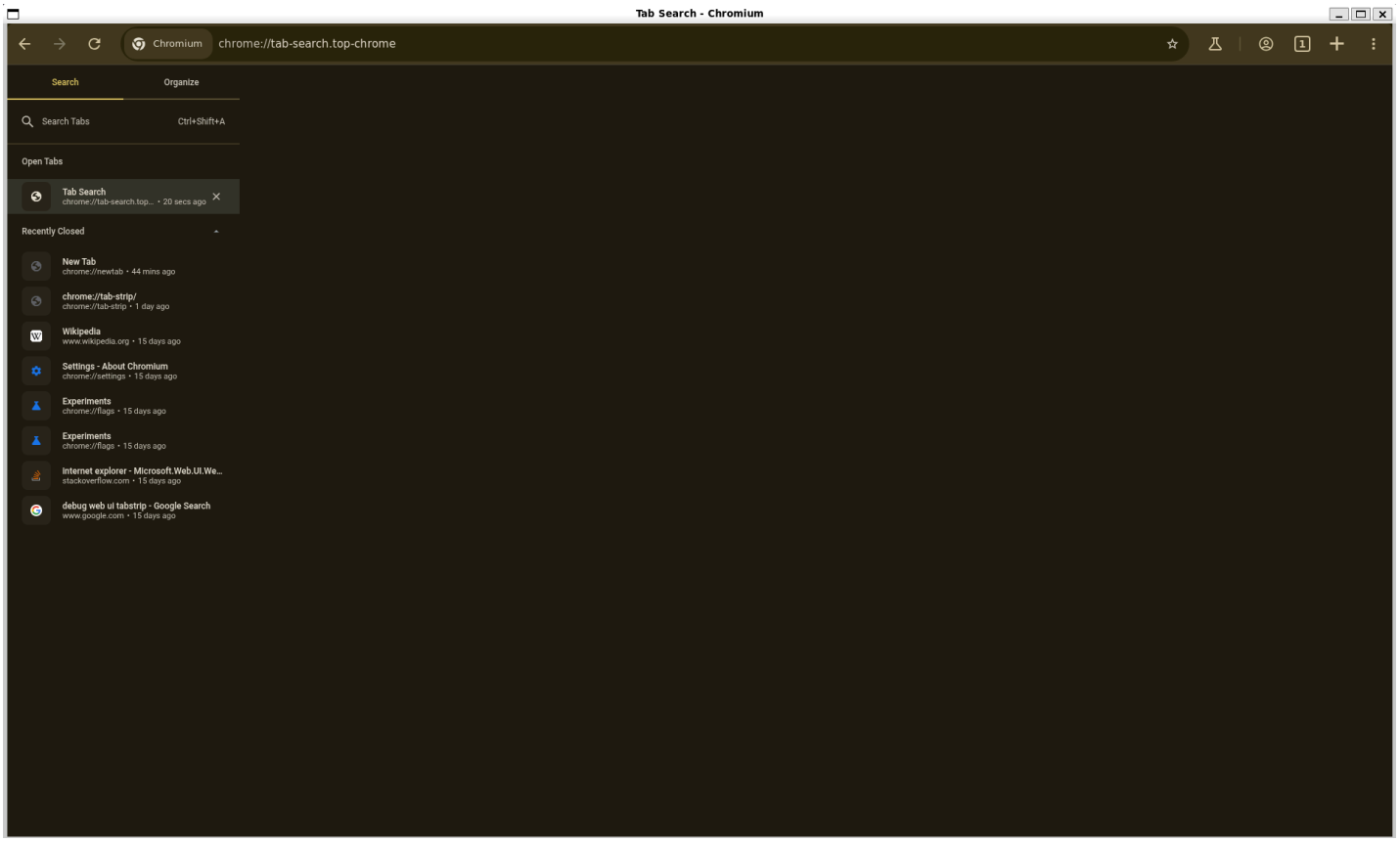
Approver: shibalik@google.com
dljames@google.com

Status: Final
Created: 06-04-2025

# Project Abstract

*The "Debug WebUI For Tabstrip States" project aims to enhance the debugging experience for Chromium developers by building a read-only WebUI page (similar to chrome://tab-search.top-chrome/) that captures and visually represents the live state of the browser's tabstrip model, group metadata, selection model, and session restore information. Tabstrip and session states in browsers are inherently complex, relying on the correct ordering of tabs, groups, and session data to restore tabs accurately. This complexity has led to numerous bugs, and distinguishing between tabstrip model issues and client-side problems remains a challenge. This project seeks to provide a clear, visual tool to streamline debugging and improve developer productivity.*

*The focus is solely on creating a WebUI page for debugging purposes, without modifying the underlying tabstrip model.*



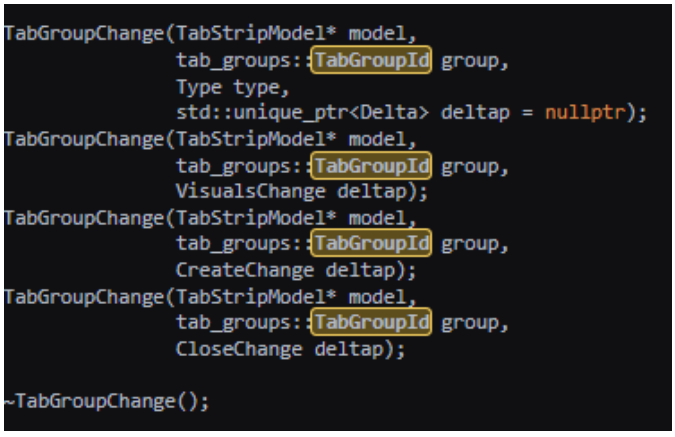
# Background

---

*TabStrip is the UI component at the top of the browser window that displays and manages all open tabs. It is the 'row of tabs' at the top of the browser -> Chromium has built the tabstrip using native UI code (C++, HTML, CSS, TS and the Views framework). The tabstrip in Chromium manages the arrangement, grouping, and selection of tabs, while session restore ensures that tabs are reloaded in their correct state after a browser restart.*
*What is meant by 'correct ordering of tabs'?*
*Normally, in chromium tabs appear from left to right. This order isn't just about visual display; it has functional aspects.*

- *Pinned tabs -> Pinned tabs must be at the beginning of the TabStripModel.*
- *Tab Groups -> Tabs within a group must be contiguous (next to each other) in the TabStripModel.*
- *Active tab -> The TabStripModel maintains a single "active" tab, which is the one whose contents are currently displayed in the browser window.*
- *Selection -> The TabStripModel has a 'selection_model_' that tracks which tabs are selected.*
- *Session Restore -> When we restore a browser session (after a crash or restart), the tabs must be recreated in the exact same order and with the same pinned/grouped/active/selected state as before. This is where the ordering becomes critical. Any deviation can lead to a bug.*

*Errors in these systems—whether due to incorrect tab ordering, group metadata, or session data—can lead to bugs that are difficult to fix. Currently, developers rely on logs, manual inspection, or custom debugging tools, which are time-consuming and inefficient. A dedicated WebUI page that mirrors the backend state of the tabstrip model, would provide a live, visual representation of these states, making it easier to pinpoint issues and edge cases.*
*This project will use Chromium's existing infrastructure to fetch and display data from the tabstrip model, group collections, selection model, and session restore system, offering a centralized debugging resource for developers.*



```
TabGroupChange(TabStripModel* model,
               tab_groups::TabGroupId group,
               Type type,
               std::unique_ptr<Delta> deltap = nullptr);
TabGroupChange(TabStripModel* model,
               tab_groups::TabGroupId group,
               VisualsChange deltap);
TabGroupChange(TabStripModel* model,
               tab_groups::TabGroupId group,
               CreateChange deltap);
TabGroupChange(TabStripModel* model,
               tab_groups::TabGroupId group,
               CloseChange deltap);

~TabGroupChange();
```

```
// Register for this callback to detect when the group changes.
using GroupChangedCallback = base::RepeatingCallback<
    void(TabInterface*, std::optional<tab_groups::TabGroupId> new_group)>;
virtual base::CallbackListSubscription RegisterGroupChanged(
    GroupChangedCallback callback) = 0;

// Features that want to show tab-modal UI are mutually exclusive. Before
```

```
// This must be kept in sync with |contents_data_|.
ui::ListSelectionModel selection_model_;
```

## Mission

---

*Fast, seamless, safe, and reliable debugging experiences:*

- *Speed: Quickly visualize tabstrip and session states.*
- *Simplicity: Provide an intuitive, easy to use interface.*
- *Security: Ensure the WebUI is read-only and does not expose sensitive data.*
- *Stability: Deliver a robust tool that accurately reflects backend states.*

## Design Ideas and Implementation

---

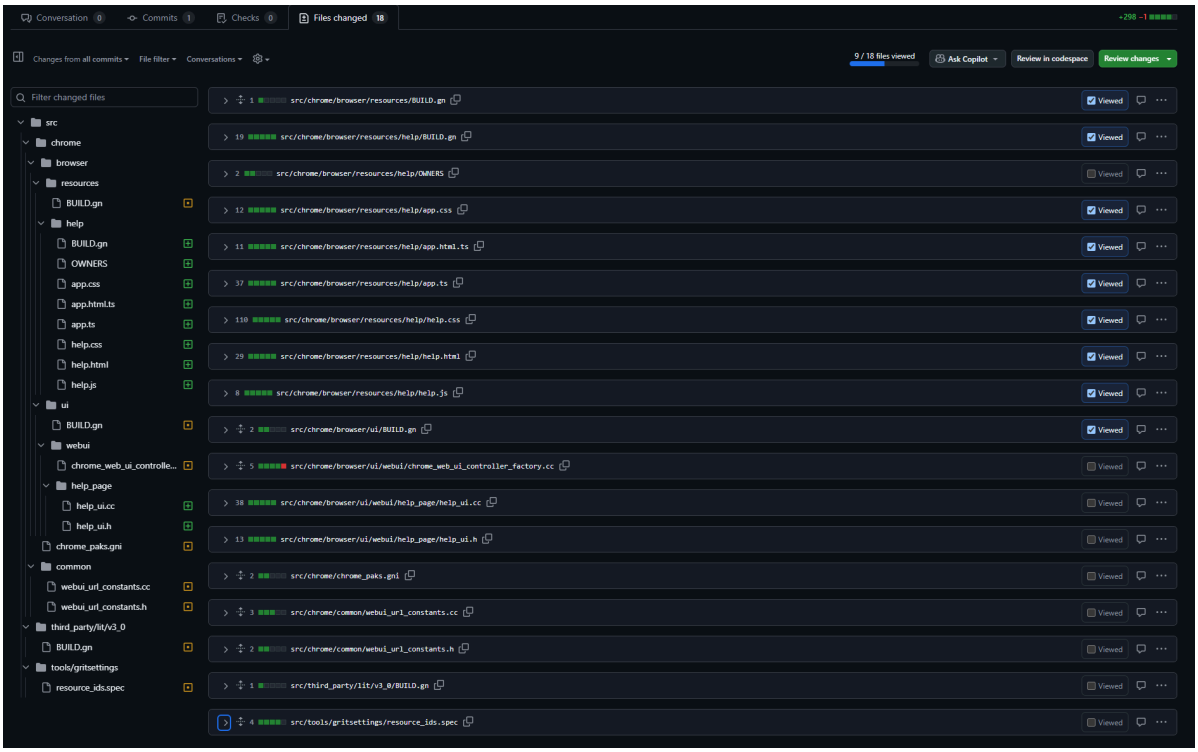
*The read-only WebUI page that will visualize the live state of the TabStripModel from all windows under current user profile, its tab group data, selection model, and session restore information will provide the following key features:*

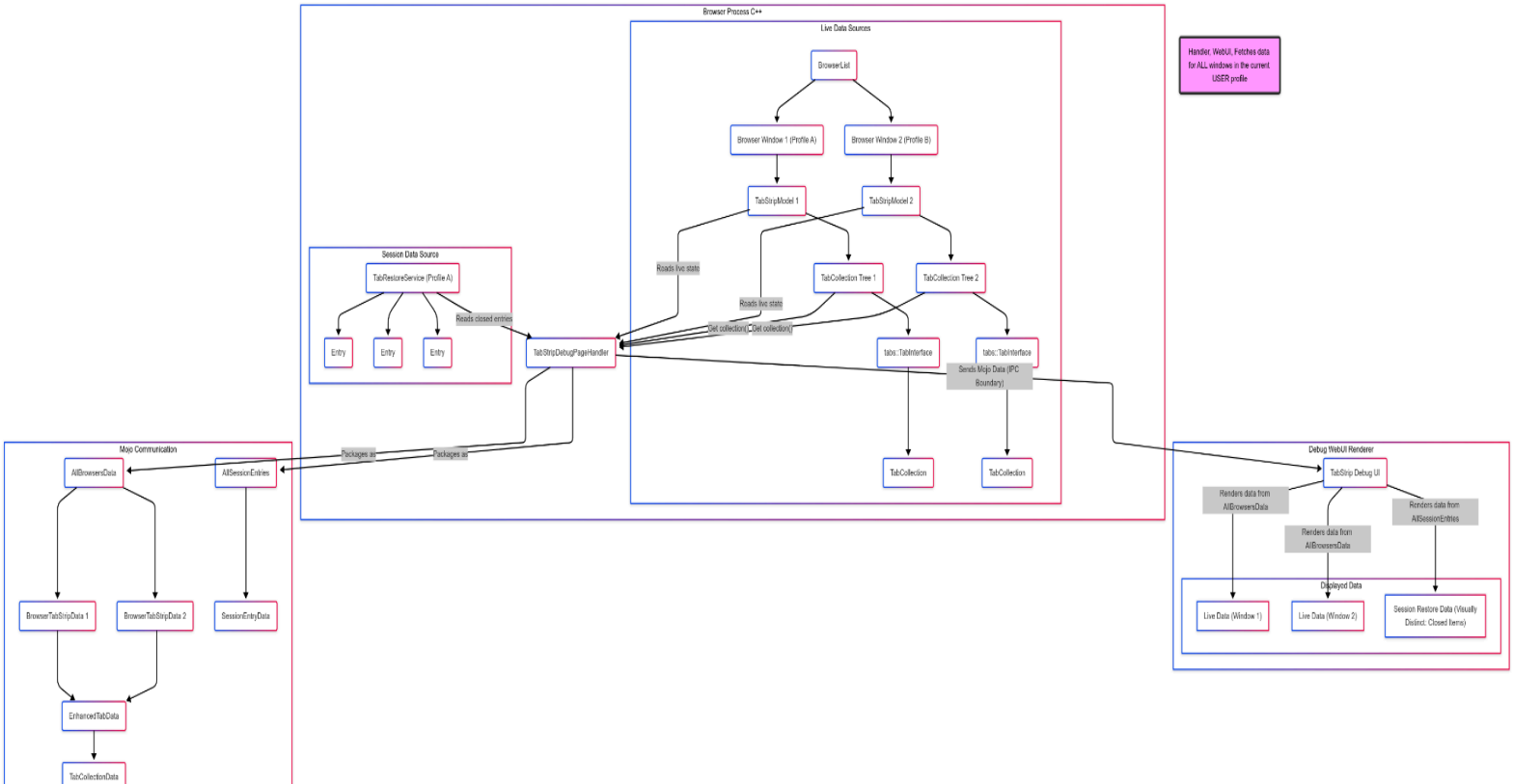
- *Create a new WebUI page (e.g: chrome://tabstrip-debug) that involves setting up the required HTML, CSS, and JavaScript files, along with a C++ WebUIController to handle communication between the browser backend and the frontend.*
    - *I have experience in creating WebUI Page. So, I think I won't need much time there.*
    - *File structure:*
        - *chrome/browser/ui/webui/tabstrip_debug/tabstrip_debug_ui.h and .cc (Controller)*
        - *chrome/browser/resources/tabstrip_debug/ directory with:*

- *tabstrip_debug.html*
- *tabstrip_debug.css*
- *tabstrip_debug.js*

■ *Updating build files (BUILD.gn)*

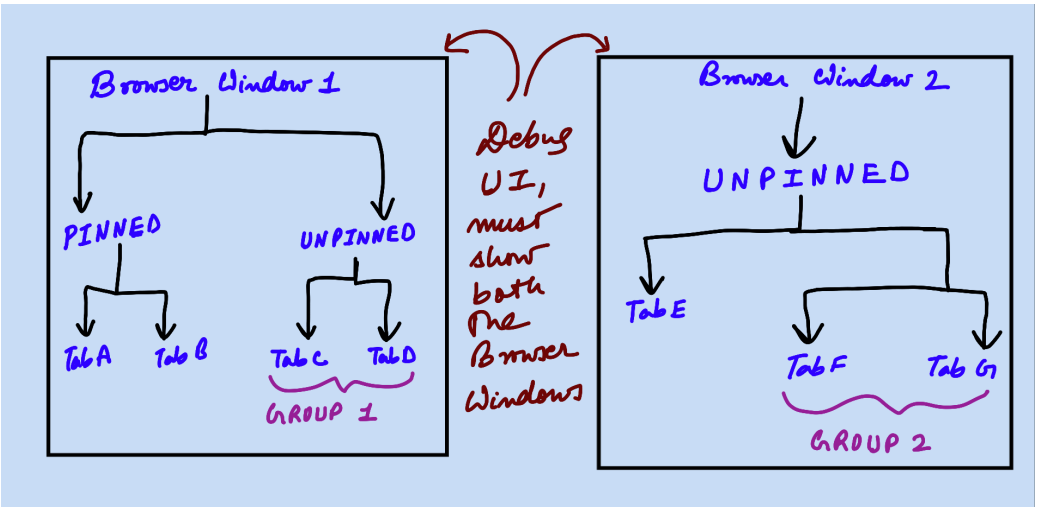
■ *Register WebUI*
*in chrome/browser/ui/webui/chrome_web_ui_controller_factory.cc.*
*(chrome::kChromeUITabStripDebugHost)*



- *Data Structures used in Mojo Interface Definition (tabstrip_debug.mojom) :*
  - *What is .mojo ?*
    *In simple words, before C++ can send data to javascript WebUI ,they need to agree on the same exact format of data. That agreement is defined in the .mojom file.*
  - *Chrome does not just have a flat list of tabs in a window. It uses a hierarchy:*
    - *The main 'TabStripCollection' represents the whole tab strip in a window.*
    - *This often contains a PinnedTabCollection and an UnpinnedTabCollection.*
    - *The UnpinnedTabCollection contains individual TabModel objects (normal tabs) and TabGroupCollection objects (representing tab groups).*
    - *This shows tree structure internally.*
  - *// Represents the kind of collection a tab directly belongs to.*
  - *struct TabCollectionData {*
  - *// Type of the collection, e.g., "PINNED", "UNPINNED", "GROUP", "SPLIT".*
  - *string collection_type;*
  - *// If collection_type is "GROUP", this holds the group's unique ID.*
  - *string? group_id;*
  - *};*
  - *struct TabData {*
  - *int32_t tab_id; // Unique ID*
  - *int index; // Index in the TabStripModel*
  - *string title; // Tab title*
  - *string url; // Tab URL*
  - *bool active; // Is the active tab ?*
  - *bool pinned; // set tab pinned?*
  - *bool selected ; // Is the tab part of the current multi-selection ?*
  - *bool is_tab_discarded; //Is this tab discarded to save memory ?*
  - *int32? opener_tab_id; // ID of the tab that opened this tab, if any*
  - *TabCollectionData? collection_data; //Information about the collection this tab belong*
  - *};*
  - *struct TabGroupData {*
  - *string group_id;*
  - *string title;*
  - *string TabGroupColor ;*
  - *bool collapsed;*
  - *bool is_saved;*
  - *};*

- *Create a PageHandler:*
    - *In tabstrip_debug_ui.h, I will create a PageHandler interface (Mojo interface). This function collects data from all open browser windows in the chrome session. For each window, it gathers details about all its tabs – things like title, URL, and whether it's active or pinned. I will add a method: GetTabstripData(). This method will eventually return all the data needed to display the tab strip state. It should return this data asynchronously as a Promise to the WebUI.*
    - *I will use chromium's BrowserList to iterate over all open Browser instances.*
    - *BrowserList* browser_list = BrowserList::GetInstance();*



```
18
19  void TabStripDebugPageHandler::GetTabstripData(
20      GetTabstripDataCallback callback) {
21
22    tab_strip_debug::mojom::AllBrowsersDataPtr all_browsers_data =
23        tab_strip_debug::mojom::AllBrowsersData::New();
24    Profile* current_profile = browser_->profile();
25    BrowserList* browser_list = BrowserList::GetInstance();
26
27    for (Browser* browser : *browser_list) {
28      if (browser->profile() != current_profile) continue;
29      TabStripModel* tab_strip_model = browser->tab_strip_model();
30      if (!tab_strip_model) continue;
31
32      tab_strip_debug::mojom::BrowserTabStripDataPtr browser_specific_data =
33          tab_strip_debug::mojom::BrowserTabStripData::New();
34      browser_specific_data->browser_window_id = browser->session_id().id();
35
36      // Populate tab data
37      for (int i = 0; i < tab_strip_model->count(); ++i) {
38        content::WebContents* contents = tab_strip_model->GetWebContentsAt(i);
39        tabs::TabInterface* tab = tab_strip_model->GetTabAtIndex(i);
40        if (!contents || !tab) continue;
41
42        tab_strip_debug::mojom::TabDataPtr tab_data =
43            tab_strip_debug::mojom::TabData::New();
44
45        tab_data->tab_id = extensions::ExtensionTabUtil::GetTabId(contents);
46        tab_data->index = i;
47        tab_data->title = base::UTF16ToUTF8(tab->GetTitle());
48        tab_data->url = contents->GetLastCommittedURL().spec();
49        tab_data->active = (tab_strip_model->active_index() == i);
50        tab_data->pinned = tab_strip_model->IsTabPinned(i);
51        tab_data->selected = tab_strip_model->IsTabSelected(i);
52
53        ThumbnailTabHelper* thumbnail_helper = ThumbnailTabHelper::FromWebContents
          (contents);
54        tab_data->is_tab_discarded = thumbnail_helper ?
          thumbnail_helper->is_tab_discarded() : false;
55
56        tabs::TabInterface* opener = tab_strip_model->GetOpenerOfTabAt(i);
57        if (opener && opener->GetContents()) {
58          tab_data->opener_tab_id =
59              extensions::ExtensionTabUtil::GetTabId(opener->GetContents());
60        } else {
61          tab_data->opener_tab_id = std::nullopt;
62        }
63
64        // Basic Session Data
65        const auto& controller = contents->GetController(); // controller
66        tab_data->navigation_count = controller.GetEntryCount();
67        tab_data->current_nav_index = controller.GetCurrentEntryIndex();
68        tab_data->collection_data = GetTabCollectionData(tab->owning_collection
          ());
69
70        browser_specific_data->tabs.push_back(std::move(tab_data));
71      }
```

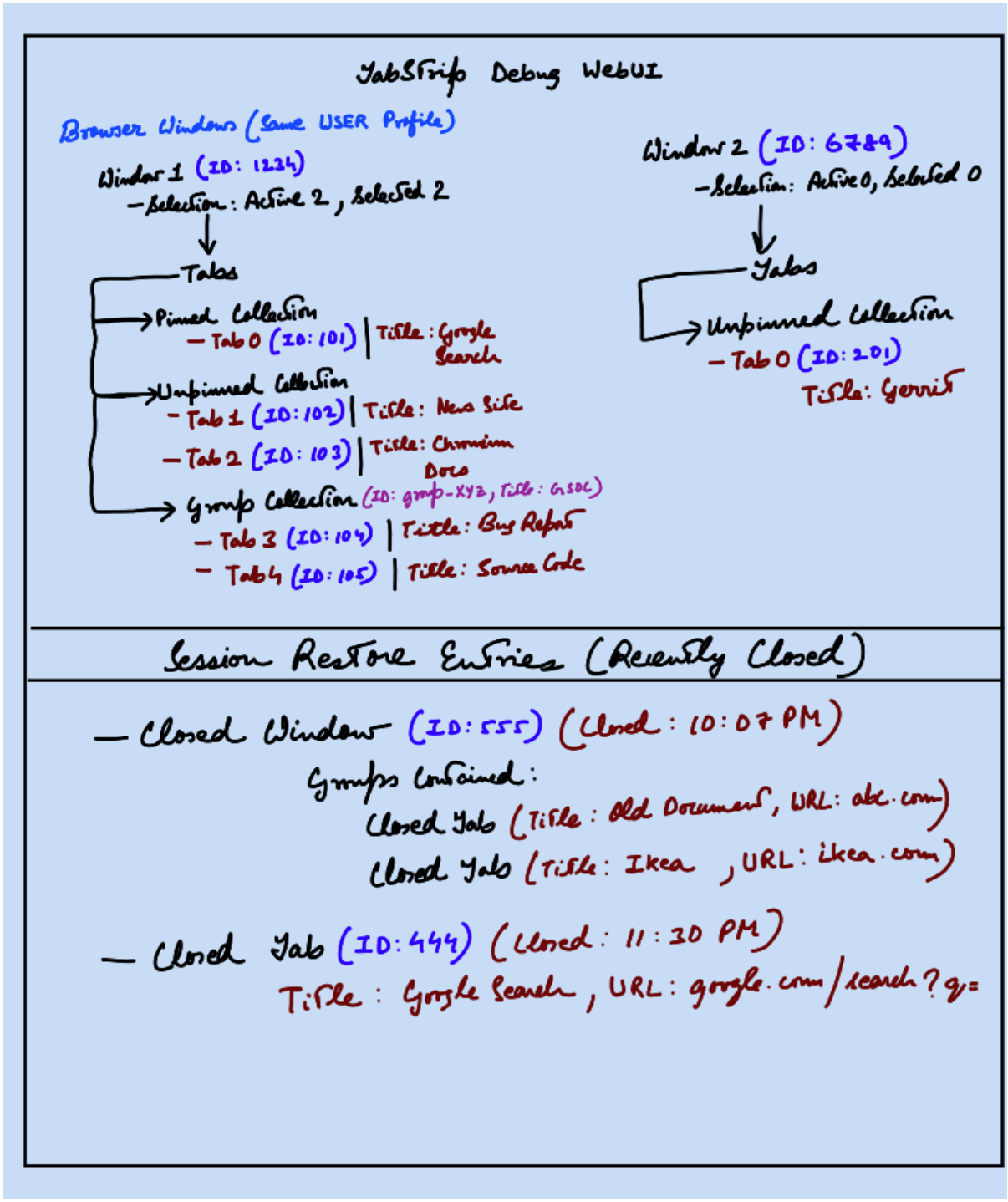
```
72
73    // Populate group data
74    if (tab_strip_model->SupportsTabGroups()) {
75      TabGroupModel* group_model = tab_strip_model->group_model();
76      if (group_model) {
77        for (const tab_groups::TabGroupId& group_id : group_model->ListTabGroups
          ()) {
78          TabGroup* group = group_model->GetTabGroup(group_id);
79          if (!group) continue;
80          tab_strip_debug::mojom::TabGroupDataPtr group_data =
81              tab_strip_debug::mojom::TabGroupData::New();
82          group_data->group_id = group_id.ToString();
83          const tab_groups::TabGroupVisualData* visual_data = group->visual_data
            ();
84          if (visual_data) {
85            group_data->title = base::UTF16ToUTF8(visual_data->title());
86            group_data->color = TabGroup::GetColorString(visual_data->color());
87            group_data->collapsed = visual_data->is_collapsed();
88            group_data->is_saved = visual_data->is_saved();
89          } else {
90            group_data->title = "[No Visual Data]";
91            group_data->color = "grey";
92            group_data->collapsed = false;
93            group_data->is_saved = false;
94            LOG(WARNING) << "TabGroup found without visual data: " << group_id.
              ToString();
95          }
96          browser_specific_data->groups.push_back(std::move(group_data));
97        }
98      }
99    }
100
```

```
100
101   // Populate selection model data
102     tab_strip_debug::mojom::SelectionModelDataPtr selection_model_data =
103         tab_strip_debug::mojom::SelectionModelData::New();
104
105     const ui::ListSelectionModel& selection_model =
      tab_strip_model->selection_model();
106
107     const ui::ListSelectionModel::SelectedIndices& selected_indices_set =
108         selection_model.selected_indices();
109     selection_model_data->selected_indices.assign(selected_indices_set.begin(),
110                                                   selected_indices_set.end());
111
112     if (selection_model.active().has_value()) {
113       selection_model_data->active_index = selection_model.active().value();
114     } else {
115       selection_model_data->active_index = std::nullopt;
116     }
117     if (selection_model.anchor().has_value()) {
118       selection_model_data->anchor_index = selection_model.anchor().value();
119     } else {
120       selection_model_data->anchor_index = std::nullopt;
121     }
122     browser_specific_data->selection_model_data = std::move
      (selection_model_data);
123
124   all_browsers_data->browser_data.push_back(std::move(browser_specific_data));
125 }
126
127 std::move(callback).Run(std::move(all_browsers_data));
128 }
129
```

- *Session Information:*
  - *The primary source for session restore information will be the sessions::TabRestoreService. We can obtain an instance using that.*
  - *The core data we need is the list of closed entries, accessible via the TabRestoreService::entries() method, which returns a const Entries& (likely a std::list of unique pointers to tab_restore::Entry).*
  - *Data Structure (Mojo):*
    - ***SessionEntryType Enum:*** *To distinguish between entry types.*
    - *enum SessionEntryType {*
    - *TAB,*
    - *GROUP,*
    - *WINDOW*
    - *};*
    - ***SessionNavigationEntryData:*** *A simplified representation of a navigation entry (likely mirroring parts of sessions::SerializedNavigationEntry ).*
    - *struct SessionNavigationEntryData {*
    - *string url;*
    - *string title;*
    - *};*
    - ***SessionTabData:*** *Represents a closed tab entry.*
    - *struct SessionTabData {*
    - *int32 session_id;*
    - *int32 current_navigation_index;*
    - *string? group_id_if_grouped;*
    - *bool pinned;*
    - *string? extension_app_id;*
    - *};*
    - ***SessionGroupData****: Represents a closed group entry.*
    - *struct SessionGroupData {*
    - *string group_id;*

- *string title;*
- *string color;*
- *bool is_saved;*
- *array<int32> contained_tab_session_ids;*
- *];*
- ***SessionWindowData**: Represents a closed window entry.*
- *struct SessionWindowData {*
- *int32 session_id;*
- *string? app_name;*
- *int32 selected_tab_index_in_window;*
- *int32 tab_count;*
- *array<int32> contained_tab_session_ids;*
- *array<string> contained_group_ids;*
- *];*
- ***SessionEntryData**:*
- *struct SessionEntryData {*
- *int32 entry_id;*
- *SessionEntryType type;*
- *int64 timestamp;*
- *SessionTabData? tab_data;*
- *SessionGroupData? group_data;*
- *SessionWindowData? window_data;*
- *];*

        - ***AllSessionEntries**: The top-level structure that will be returned.*
        - *struct AllSessionEntries {*
        - *  array<SessionEntryData> entries;*
        - *};*
    - *After this , I will get the sessions::TabRestoreService* for the relevant profile.*
    - *I will check if it's loaded (IsLoaded()). If not loaded, maybe return empty or indicate loading state.*
    - *Then, iterate through tab_restore_service->entries().*
    - *For each std::unique_ptr<tab_restore::Entry>& entry:*
        - *Determine its type (tab_restore::Type::TAB, WINDOW, GROUP).*
        - *Get the common entry->id (this is the restore entry ID, different from tab/window SessionIDs).*
        - *Cast the entry.get() to the appropriate derived type (tab_restore::Tab, tab_restore::Window, tab_restore::Group).*
        - *Then, populate the corresponding Mojo struct (SessionTabData, SessionGroupData, SessionWindowData) by extracting necessary fields (e.g., tab->id for SessionTabData::session_id, window->window_id for SessionWindowData::session_id, group->group_id for SessionGroupData::group_id, visual data, contained tab/group references, simplified navigation data).*
        - *Create a SessionEntryData Mojo struct, set its type and entry_id, and assign the populated specific data struct (e.g., entry_data->tab_data = std::move(populated_tab_data)).*
        - *Add the SessionEntryData to the AllSessionEntries result array.*
    - *Finally, run the callback with the populated AllSessionEntries.*
    - *Finally display session data into the WebUI.*




- *Challenges to tackle :*
  - *Keeping the UI in sync could get tricky. I might need observers in the tabstrip model or lean on Mojo callbacks. I'll test with scenarios like dragging tabs to ensure it holds up.*

- *Rollout Strategy: Start with a basic version of the WebUI displaying the tabstrip tree, then incrementally add group metadata, selection states, and session data in subsequent releases. A phased rollout will allow for early feedback and bug fixes.*

- *Security Considerations: Since the WebUI is read-only, it will not modify tabstrip or session data. Secure communication between the frontend and backend will be ensured using existing WebUI security practices, and sensitive user data (e.g., tab titles) will be anonymized or restricted as needed.*

  *The primary technologies used will be HTML, CSS, JavaScript for the WebUI frontend, and C++ for backend integration with Chromium's tabstrip and session systems. The WebUI will be lightweight and optimized for rendering efficiency, ensuring it does not impact browser performance.*
  *I want to discuss ideas with my mentors during the bonding period for better implementation.*

## Code Affected

---

- *chrome/browser/ui/tabs/*
  - *For accessing tabstrip model data and related classes.*
  - *Interface with TabGroupModel for group data.*
- *chrome/browser/sessions/*
  - *For fetching SessionService and SessionStore information.*
  - *Extract information about saved tab sessions*
- *chrome/browser/ui/webui/*
  - *Add new WebUI handler and resources*
  - *Register the new chrome:// URL*

## Alternatives Considered (Optional)

---

- *My mother used to say, "An unrevised paper is an unfinished paper."*
- *That sentiment resonates deeply in software development -> any feature added without thorough testing is similarly incomplete. For the chrome://tabstrip-debug project, simply displaying data is not enough - it needs to be correct and presented clearly.*
- *To achieve this, I plan to adopt a two approach, as given by the standard practices within Chromium for WebUI development. This involves both C++ browser tests to validate the backend data fetching and logic, and TypeScript/JavaScript tests to verify the correct rendering and behavior of the frontend UI. This dual approach will give us*

*confidence that the tool reliably reflects the state of the TabStripModel and related components.*

- *Backend Test*
  - *Create a file named tabstrip_debug_ui_browsertest.cc inside chrome/browser/ui/webui/tabstrip_debug/ directory.*
  - *It should inherit WebUIMochaBrowserTest (like as in inspect_ui_browsertest.cc) or InProcessBrowserTest (like as in tab_search_ui_browsertest.cc)*
  - *Execution Steps (Inspired by inspect_ui_browsertest.cc and tab_search_ui_browsertest.cc):*
    - *To navigate to the url , I will use ui_test_utils::NavigateToURL to navigate to chrome://tabstrip-debug URL. This is a basic load check.*
    - *Create various browser states for different test cases. Examples:*
      - *A single browser window with a few simple tabs.*
      - *A window with pinned tabs.*
      - *A window with tab groups (some collapsed, some saved).*
      - *Multiple browser windows belonging to the same profile.*
      - *A browser window with some recently closed tabs/groups (to test TabRestoreService interaction later).*
    - *Use content::EvalJs (as used in tab_search_ui_browsertest.cc) to execute JavaScript in the WebUI's renderer.*
    - *I might use functions like chrome::AddTabAt, browser->tab_strip_model()->AddToNewGroup, etc., similar to how TabSearchUIBrowserTest sets up its tabs.*
- *Frontend Test*
  - *Create a file named tabstrip_debug_test.ts inside chrome/test/data/webui/tabstrip_debug/ directory.*
  - *Use the 'chrome://webui-test' framework from 'chrome://webui-test/chai_assert.js' and 'chrome://webui-test/test_util.js'*
  - *Implement a mock TestTabStripDebugApiProxy (similar to TestTabSearchApiProxy) to test the C++ backend, providing controlled Mojo data for GetTabstripData and GetSessionRestoreData.*
  - *Test initial rendering by providing mock data representing various scenarios (multiple windows, different tab/group states, session entries) and using DOM queries (querySelector, querySelectorAll) along with assertions (chai_assert.js).*

## Related Work

---

- *chrome://inspect/: Provides User interface for dev tools.*

- *chrome://tab-search.top-chrome/: Provides a user interface for searching and managing tabs, but lacks detailed model state information*
- *chrome://memory-internals/: Shows memory usage of tabs and processes, showing how internal state can be exposed through WebUI.*
- *chrome://crashes/: Presents crash reports and diagnostic information.*
- *chrome://tracing/: Provides timeline visualization of browser events.*

## Pre proposal Work

---

- *Contributions to Chromium:*
    - https://chromium-review.googlesource.com/c/chromium/src/+/6361177 *(GSOC Starter bug), (Under review)*
      *Skills involved: C++, Understanding of tabstrip controller (tab_strip_ui.cc), and how tabstrip handles webui top touch chrome using tab_strip_page_handler.cc/h and thumbnail_tracker.cc/h*

    - https://github.com/wootzapp/wootz-browser/pull/108
      (*First Open Source Contribution*), *(Under review)*
      *Skills involved: C++, Web development (TypeScript, HTML/CSS)*
      *Project Briefing: As, I was very much fascinated by how chromium works, I started figuring out chromium codebase. But as it was very large and complicated for a first-time contributor, I thought it was not my cup of tea. Luckily, I stumbled upon this wootzapp browser. I found it interesting as they were building a web3 browser on top of chromium. So, I reached out the CEO, providing him some of my proof of work and requesting him, if I can be of any use. He then, asked me to study brave browser's codebase and chromium codebase and understand the overall difference in their system design. It was a long one-month procedure. After that, he asked me to create a help-page WebUI and also told me to first understand the difference between web-page and WebUI. I then, started going through all the docs of chromium , especially "Creating WebUI Interfaces" and "WebUI BUILD.gn" . I then asked him, if I am on the right track? and he said yes. Finally, I followed the guide and created a simple help-page UI.*

- *I've built Chromium on my local system (Intel 64 Linux ) by reading documentation, interacting with mentors, and asking doubts in the Chromium-dev group. I have also read about mojo in C++ and learnt related documents to it. Mostly in my pre proposal period, I took guidance from my mentors, who helped me to navigate across the codebase and make contributions according to the style guide.*

# Schedule of Deliverables (timeline)

---

*This is a rough estimate of the things I am planning to do. I want to make it more detailed in my Community Bonding Period, by discussing it with my project mentors.*

## May 8 – June 1 (Community Bonding Period)

- *Community Bonding Period In the first one or two weeks, I want to do more research on how to get the sessions data into the WebUI and the code of conduct, strengthen the relationship with my mentors by getting to know what they expect, and have a better understanding of what we are going to accomplish so that we don't run into any problems later. I also want to talk about my alternative ideas and come up with a final design for this project. and after that, I want to make the most of the time I have left by starting to code in order to produce the intended outcome as accurately and efficiently as I can.*

- *I will implement the basic WebUI framework, as I already have experience in building WebUI so, I won't take much time.*

- *Implement initial Mojo interfaces.*
- *Build the 'TabstripDebugPageHandler' and hook it up to the tabstrip model.*

## June 2 - July 14 (Phase I)

- *Week 1-2:*
  - *Implement basic tabstrip model visualization.*
  - *Set up the data flowing to the WebUI page via Mojo.*
  - *Documenting as I implement things, as much as possible.*
- *Week 3-4:*
  - *Display tabs and their ordering.*
  - *Test with simple tab setups to ensure accuracy.*
  - *Create initial tree view structure and add tab group visualization.*
- *Week 5-6:*
  - *Display group information (color, title).*
  - *Show tabs within groups.*
  - *Implement collapse/expand functionality.*

## July 14 - August 25 (Phase II)

- *Week 7-8:*

    - *Implement selection model visualization and display current selected tabs.*
    - *Highlight active tabs in the UI.*
    - *Add visual indicators for pinned tabs.*
    - *More Documentation.*
- *Week 9-10:*
    - *Add session information.*
    - *Retrieve and display session restore data.*
    - *Show relationship between current tabs and session data.*
- *Week 11-12:*
    - *Refine, test, and document.*
    - *Fix bugs and edge cases.*
    - *Optimize performance.*
    - *Write documentation, prepare final documentation and presentation.*

### August 25 - November 9 (For Extended Timelines)

- *Add tests as mentioned in the 'Alternatives considered' section.*
- *Implement auto-refresh mechanism for live updates.*
- *Implement advanced features based on mentor feedback.*
- *Add optional visualizations or improved UI elements*

## Communications

---

*I am flexible with any means of communication, and I have no problem adjusting to my mentor's time zone. In the summer, I have no obligations as such because of the summer vacation, and I am going to commit full time to this project and am ready to devote 40-50 hours per week (approximately 7-8 hours on weekdays and 5-6 hours on weekends). And in case If any delay occurs, I would first assess the situation and determine the reason for the delay. Then, I would communicate with the mentor to discuss the situation and see if any adjustments needed to be made to the timeline or deliverables.*

- *Timezone: GMT +5:30 (Indian Standard Time)*
- *Working Hours: Flexible*
- *Email: Koushik*
- *Alternative email: mca40057.22@bitmesra.ac.in*

- *Phone: +91-7047274427*
- *Slack: koushikbug123@gmail.com*


If something else is needed, I will join it with pleasure.

## About Me

---

*I'm Koushik Kumar Bug, a recent graduate from BIT Mesra with experience in web development and cloud technologies. Currently, I am doing course certifications through online classes. I love creating video content (educational , entertainment) in my pass time. My programming expertise includes C++, Solidity, JavaScript, TypeScript, and HTML/CSS, which are relevant to WebUI development in Chrome. Previously, I served as an intern as ML Ops in Amazon and prior that, also as an intern in PrepInsta Technologies Pvt.Ltd.*
*My journey with browsers began during my university days when I was fascinated by how web browsers interpret and render web content. Like, we all know that using CSS, we can provide colours in our websites. But I was eager to know, how does the browser know that this specific color is red? Why cannot I say, GREEN is RED or RED is GREEN? Seems dumb and funny! Right? But made very much sense to me at that time. Then I read the Google Chrome Comic Book .This interest led me to explore browser internals and eventually contribute to the Chromium project. I've made several contributions to the codebase, particularly in the WebUI components and tab-related functionality.*
*During my internship at PrepInsta, I worked on web development projects that gave me a solid foundation in frontend technologies. Later, at Amazon Development Centre India (ADCI), I gained experience in ML ops, which helped me understand large-scale systems and their development processes.*
*What draws me to this particular project is the challenge of visualizing complex state management systems. Having experienced first-hand the difficulties of debugging webui-tabstrip related issues, I'm passionate about creating a tool that will make this process more efficient for developers for maintaining and improving complex software systems, and I'm excited about the opportunity to contribute to this aspect of Chromium's development ecosystem.*

## Prior Experience with open source

---

- *GitHub Id:* https://github.com/KoushikBaagh

- *Contributions to Chromium:*
    - https://chromium-review.googlesource.com/c/chromium/src/+/6361177
      (*GSOC Starter bug*) *(Under Review)*
    - https://chromium-review.googlesource.com/c/chromium/src/+/6345010
      (*GSOC starter bug*) *(Merged)*

- Other Contributions:
    - https://github.com/wootzapp/wootz-browser/pull/108 *(Under Review)*

*I'm a beginner to open source; this is my first time participating in such a big open source program, and I hope this is a great kick start for me to learn and contribute to projects that will impact so many users for the better.*

## My Projects

---

- *Created a React-based frontend website for my residential society*
    - *GitHub Link: https://github.com/KoushikBaagh/Shilpakanan-Frontend*
    - *Deployed Link: https://shilpakanan.netlify.app/*

- *'Konversify' A real-time communicative platform for BIT Alumni's*
    - *Developed a communicative platform for BIT Alumni, integrating features such as asynchronous request handling, real-time texting, file sharing, video calling, and screen sharing. Scaled the support to over 70 concurrent users while maintaining latency below 50 milliseconds thereby achieving a 40% increase in alumni engagement and a 25% increase in networking efficiency. Built the platform using a comprehensive technology stack, including REACT, JavaScript, Node.js, Express.js, multer, Socket.IO, MongoDB, RESTful API, JWT, and Bcrypt.js.*
    - *GitHub Link: https://github.com/KoushikBaagh/konversify-chatApp*
    - *Deployed Link: https://konversify-x-bit-mesra.onrender.com/*

## Why Chromium?

---

*I have chosen the Chromium organization for Google Summer of Code because it is a well-established organization with a large community of developers working on a wide range of projects related to web technologies. As an open-source enthusiast, I am passionate about contributing to projects that have a significant impact on the web and the way we use it. Chromium is a project that fits this description perfectly, with its focus on creating an open-source browser that is fast, secure, and reliable.*

*Additionally, I am impressed by the Chromium project's commitment to community involvement and the support it provides to new contributors. Through my research, I have found that the organization has a vibrant community of developers, which makes it an ideal place for me to learn and grow as a developer. Overall, I believe that participating in the Google Summer of Code program with the Chromium organization will provide me with a unique opportunity to work with a talented group of developers and contribute to an important open-source project that can impact a large number of users for the better.*

## Feedback from Chromium

---

*If you read this document, please provide your short general feedback in the section below. Please also feel free to make comments above.*

| Username | Date | Comment |
|---|---|---|
| | | |
| | | |

## I am 100% dedicated to Chromium and have no plans whatsoever to submit a proposal to any other organization.