Deliverable 3

BYTE

Description of feature requests

Feature request 1

#1662 [Enhancement] Top level toggle for easy switching (filtering) between containers
Link: https://github.com/mozilla/multi-account-containers/issues/1662

Description

The feature requested by the user is to be able to hide all the tabs that do not belong to the container that the user clicks on, that is, only show the tabs that are assigned to the selected containers. This action should be easy to carry out by clicking once on the container.

Brief user interface and behavior description:

- There should be a top-level toggle button with all containers listed in the popup component.
    - By clicking the given container name, all other tabs and bookmarks will be hidden.
    - When switching between containers, it should always show the last opened tab of a given container, or a blank home page if there is no currently opened tab of the given container
- There should be another toggle option “Show All” above all the containers
    - Toggle that option to allow all the containers to be visible.

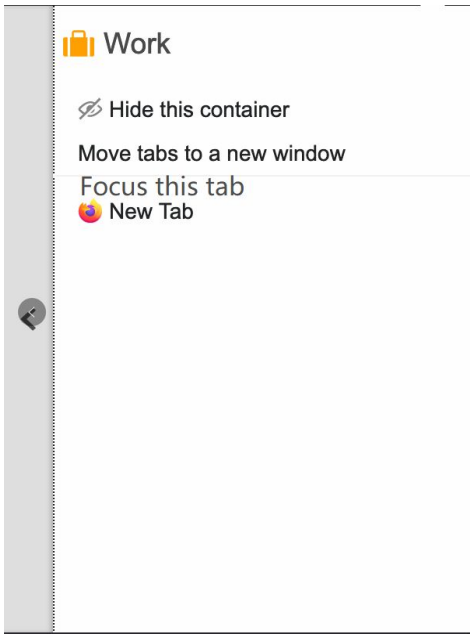
Moreover, the user wants to have bookmark categorizations or tags based on containers which can help the user recognize which container the page belongs to on the bookmark or in the bookmark folder.

Brief user interface and behavior description:

- The bookmarks or bookmark folders would be easily identified to which container they belong, by its color or the icon of the container.

Workload

In order to show the top-level toggle button, we need to modify the frontend UI. In the frontend UI (`popup.html`) **line 159-167**, there are two options already on the page. (‘Hide this container’ and ‘Moves tabs to a new window’).



Therefore, we can add a new button (‘Focus this tab’) within the same section. Once the button is clicked, we need to write logic in popup.js to detect which container was clicked. Once we receive the container that has been clicked by the user, we can create a new case in the init function in messageHandler.js **line 7-78** to write the logic of hiding tabs. We can call the hidetabs function in the background.js **line 289-299** to hide all the tabs that are not in the clicked container. If there is no tab in the clicked container, then we close the window.

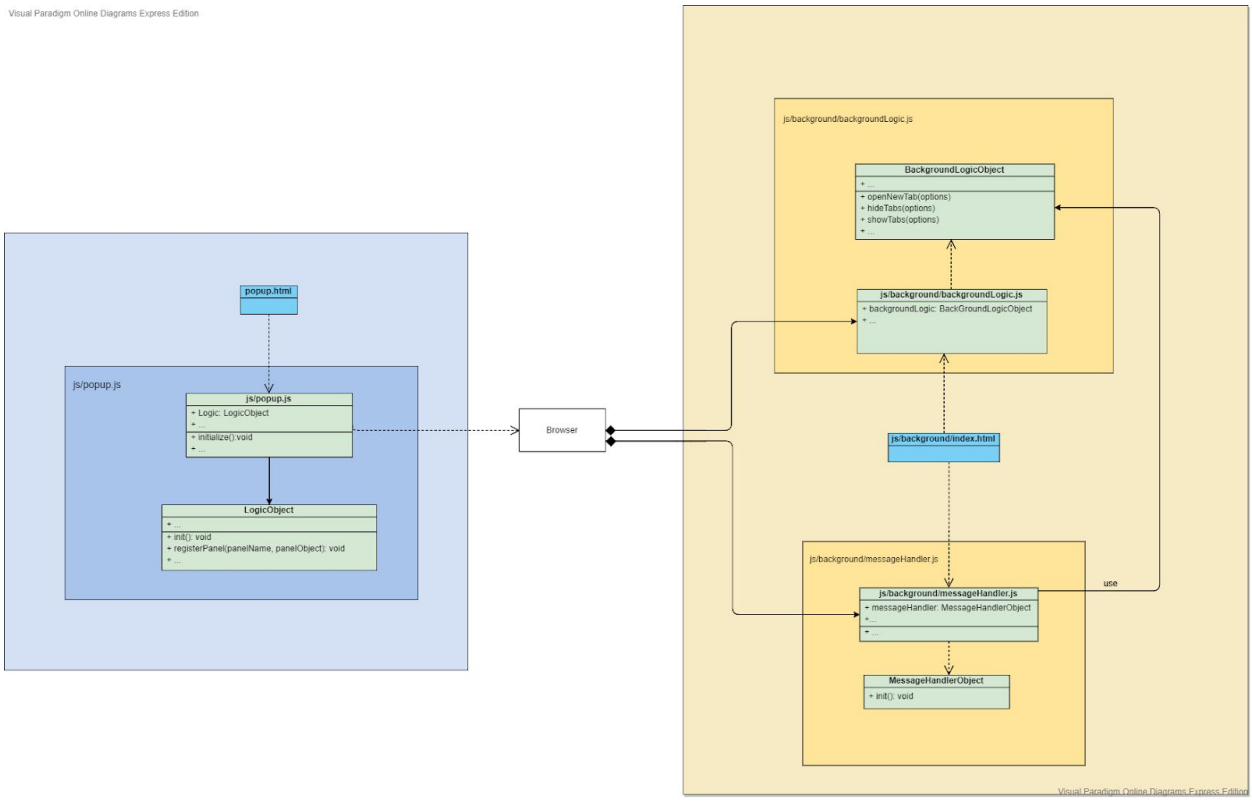
```
async hideTabs(options) {
 const requiredArguments = ["cookieStoreId", "windowId"];
 this.checkArgs(requiredArguments, options, "hideTabs");
 const { cookieStoreId, windowId } = options;

 const userContextId =
backgroundLogic.getUserContextIdFromCookieStoreId(cookieStoreId);

 const containerState = await identityState.storeHidden(cookieStoreId, windowId);
 await this._closeTabs(userContextId, windowId);
 return containerState;
},
```

Whenever a new container is clicked, we need to write a new case in messageHandler.js **line 7-78** to hide the tabs that are not in the new container.
To sum up, we need to do following changes to our current code in order to implement the new logic:

1. In the popup.html, add a button ‘Focus this tab’
2. Register an event in the popup.js and call the messagehandler.
3. Create new cases inside the `background/messageHandler.js` file to hide the tabs.

4. Build up the business logic inside the `background/backgroundLogic.js` file to hide tabs.

UML Diagram



Link to UML:
https://github.com/CSCD01/team_05-project/blob/master/docs/issue-%231662-UML.pdf

The blue box contains the frontend part of the project. It has the user interface and logic that controls the frontend part and includes the buttons clicking event and the form submissions, so this is where we will add the 'Focus this tab' button logic and register click events for it. The popup.html has to be modified as well by adding the button 'Focus this tab'.

The `Browser` acts as a middleman that communicates between the frontend and backend. The yellow box contains the backend part which controls all the logic behind the scene, so this is where we would add the logic for showing and hiding tabs for each container.

`MessageHandler` will receive a message from the frontend side through the `Browser` which commands it to call functions in the `BackgroundLogic` to handle which tabs to show or hide.

Feature request 2

#1607 Feature Request: Reorganize windows (not just tabs) by category
Link: https://github.com/mozilla/multi-account-containers/issues/1607

Description

The feature requested by the user is to be able to organise tabs into new windows based on their containers. For example, suppose the user had the following set up:

Window 1's tabs
- Personal tab 1
- Personal tab 2
- Bank tab 1
- Shopping tab

Window 2's tabs
- Bank tab 2
- Personal tab 3
- Default tab

The user wants to be able to click a button to sort these tabs into new windows, such as:

Window 1's tabs
- Personal tab 1
- Personal tab 2
- Personal tab 3

Window 2's tabs
- Bank tab 1
- Bank tab 2

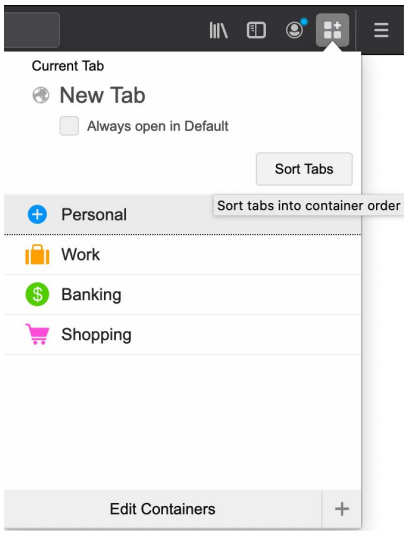
Window 3's tabs
- Shopping tab

Window 4's tabs
- Default tab

This way, the tabs of different containers are all separated into their own windows.

Workload

In order to figure out the best approach for implementing this feature, there is a similar feature already in the codebase which we can learn from. The feature is to sort tabs. In the popup menu, there is a button named "Sort Tabs".



Once the button is clicked, tabs will be sorted in the containers' order. This feature uses Firefox API `browser.runtime.sendMessage(...)` to ask the handler to deal with the request in the file `popup.js` **line 626-635**.

```
    Logic.addEnterHandler(document.querySelector("#sort-containers-link"),
async () => {
      try {
        await browser.runtime.sendMessage({
          method: "sortTabs"
        });
        window.close();
      } catch (e) {
        window.close();
      }
    });
```

The handler for `sendMessage(...)` is implemented in `background/messageHandler.js` **line 43-45**, which calls the `sortTabs()` method.

```
      case "sortTabs":
        backgroundLogic.sortTabs();
        break;
```

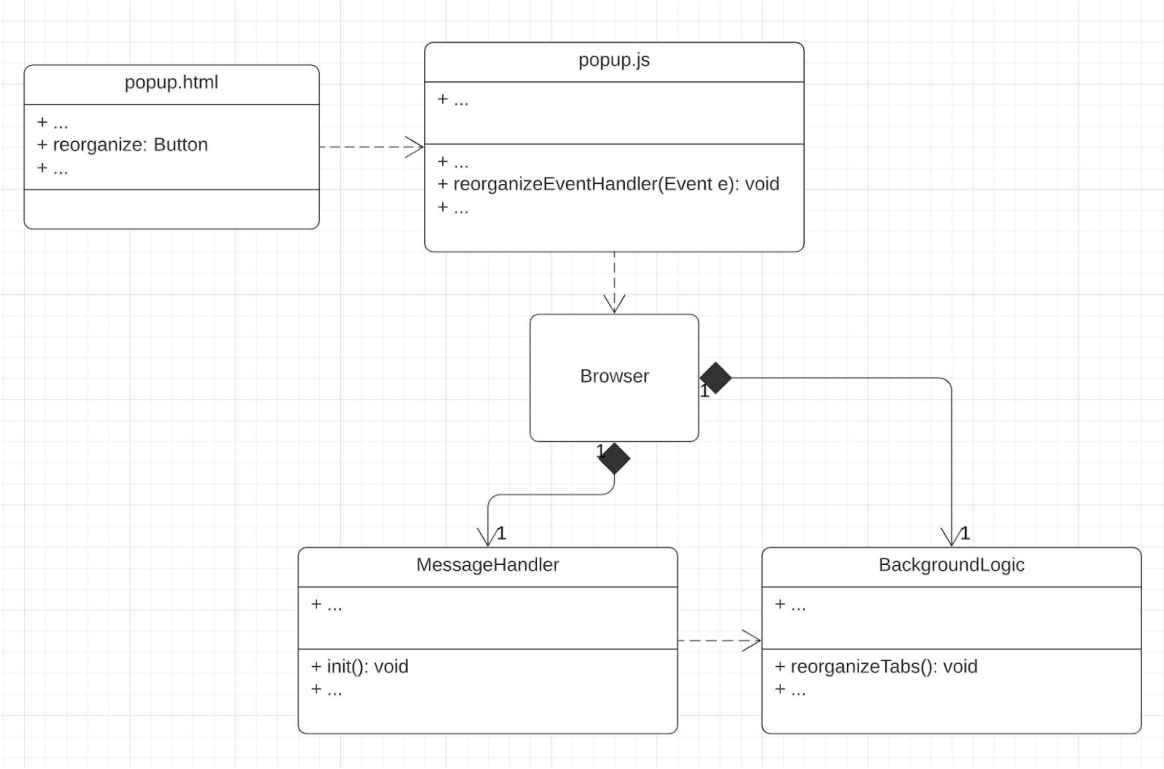
Then, the method is implemented in `background/backgroundLogic.js` **line 240-247**, and the `_sortTabsInternal(...)` is the final function executing the sorting algorithm in the same file.

```
  async sortTabs() {
    const windows = await browser.windows.getAll();
    for (let windowObj of windows) { // eslint-disable-line prefer-const
      // First the pinned tabs, then the normal ones.
      await this._sortTabsInternal(windowObj, true);
      await this._sortTabsInternal(windowObj, false);
    }
  },
```

By now, we understand the files that need to change. In order to implement this feature, we need to do the following. (For more details, please read the **Implementation details**)

1. In the `popup.html` file add a new button (maybe) asking the user to trigger.
2. Register the event in the `popup.js` file and call the handler.
3. Create a new case inside the `background/messageHandler.js` file for re-organizing the windows.
4. Build up the business logic inside the `background/backgroundLogic.js` file to re-organizing the window.

UML Diagram



(Link: https://github.com/CSCD01/team_05-project/blob/master/docs/issue-%231607.pdf)

As we can see from the UML diagram, we have four files (excluding `Browser`) related to the feature. Adding the button from the HTML file is the only interactive part with the users. Then, the button clicking event is registered in the JavaScript file, which means once the button is clicked, then the event handler will be triggered.

The `Browser` supports most of the Firefox API for Addons. The other two classes are in the background for Firefox. Background is a place to store JavaScript files and is loaded as soon as the Addon is loaded, and stays loaded until the extension is disabled or uninstalled.

Our JavaScript creates two classes `MessageHandler` and `BackgroundLogic` for the `Browser`. Therefore, the handler in the `popup.js` just passes the message to the `Browser`. Then, it calls `MessageHandler` to handler the message. The real logic is in `MessageHandler`, and the handler will call it to do the real work.

Issue chosen to be implemented

The issue we have chosen to implement this time has a focus on both frontend and background. For the frontend, we need to create interactive buttons for users and record their behavior using event listeners. For the backend, we need to deal with the background logic for Firefox (For example, how to sort the position of tabs, how to create new windows, how to move tabs between different tabs). After investigating both features, we found out that the workflow for those two issues are similar (both frontend and backend are similar). However, we found out that for the first feature request, we just need to change the logic of the background cases instead of creating new background logic. To be more specific, we found out there are applicable hide and show tabs functions in the background logic and thus we just need to write an iteration for the background logic. Therefore we would like to choose the second feature request so that we can demonstrate our understanding of the project more convictively.

Feature request 2

We chose this issue because being able to separate tabs in different windows (according to containers) at once would be very convenient. It is sometimes annoying to see a huge amount of tabs (10 or more) just in one window. As a user, I can visit the websites that are in one specific container more easily.

One problem we anticipate running into would be understanding how to write the control logic for Firefox in backgound.js. We also need to consider the behaviour when the user has already opened multiple tabs.

Implementation details

In the workload part, we illustrated the fundamental workload. Here is the detailed implementation plan.

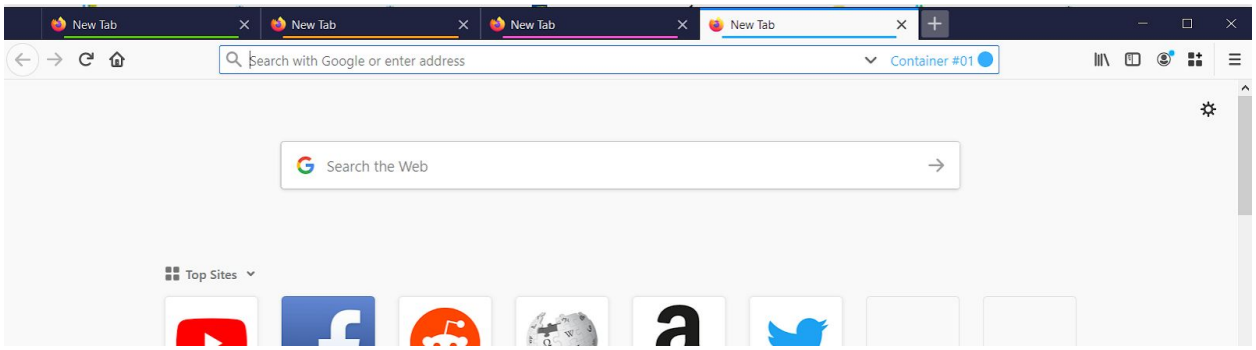
1. In the `popup.html` file,
    a. Add a button in the first page of the addon popup for the user to click on, next to the "Sort Tabs" button
    b. Call it "Sort Tabs by Window" and style it in the same way as the "Sort Tabs" button
2. In the `popup.js` file,
    a. Register a new event handler for clicking the newly added button in step 1 inside the function call `Logic.registerPanel(P_CONTAINERS_LIST, {...})`
    b. Use `browser.runtime.sendMessage(...)` to pass the message "reorganize"
3. In the `background/messageHandler.js` file,
    a. Add one case inside the `init` function for the message received from `popup.js` "reorganize"
    b. Call the `reorganize` function inside `background/backgroundLogic.js` file

4. In `background/backgroundLogic.js` file, create a new function named `reorganize` inside the function,
    a. Go through each tab inside each window
    b. Assign/reopen each tab in the correct window according to the container of the tab and the container assigned to the window
    c. Create a new window if the current number of windows is not enough
    d. Delete redundant windows at the end

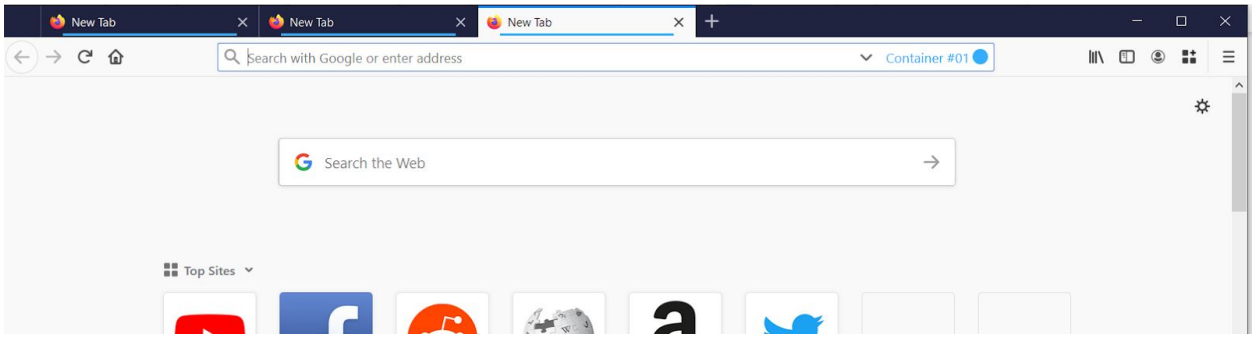
Acceptance tests

Customer Acceptance Test 1 (Multiple windows and different containers)

Step 1: Open the Firefox web browser in three different windows.
Step 2: Pick 4 containers to test the new feature.
Step 3: For each window, open at least one new tab for each selected container.



Step 4: Click the "Sort Container by Windows" button.
Step 5: Verify number of Firefox windows opened. There should be exactly 4 Firefox windows open.
Step 6: Verify each window only has tabs for a unique corresponding selected container. Meaning each window has only the tabs that belong to the corresponding selected container.



Step 7: Verify that all the tabs before the sort operation still exist in those 4 windows.

Note: If it passed those 7 steps and gives the desired result, redo the 7 steps again with the actual websites instead of the new tabs.

Customer Acceptance Test 2 (Multiple windows and one container)

Step 1: Open the Firefox web browser in three different windows.
Step 2: Open a new tab of one container in each window (e.g. Personal).
Step 3: Click the "Sort Container by Windows" button.
Step 4: Verify number of Firefox windows opened. There should be exactly 1 Firefox window open.
Step 5: Verify this window has three tabs, each of one container (e.g. Personal).
Step 6: Verify that all the tabs before the sort operation still exist in that 1 window.

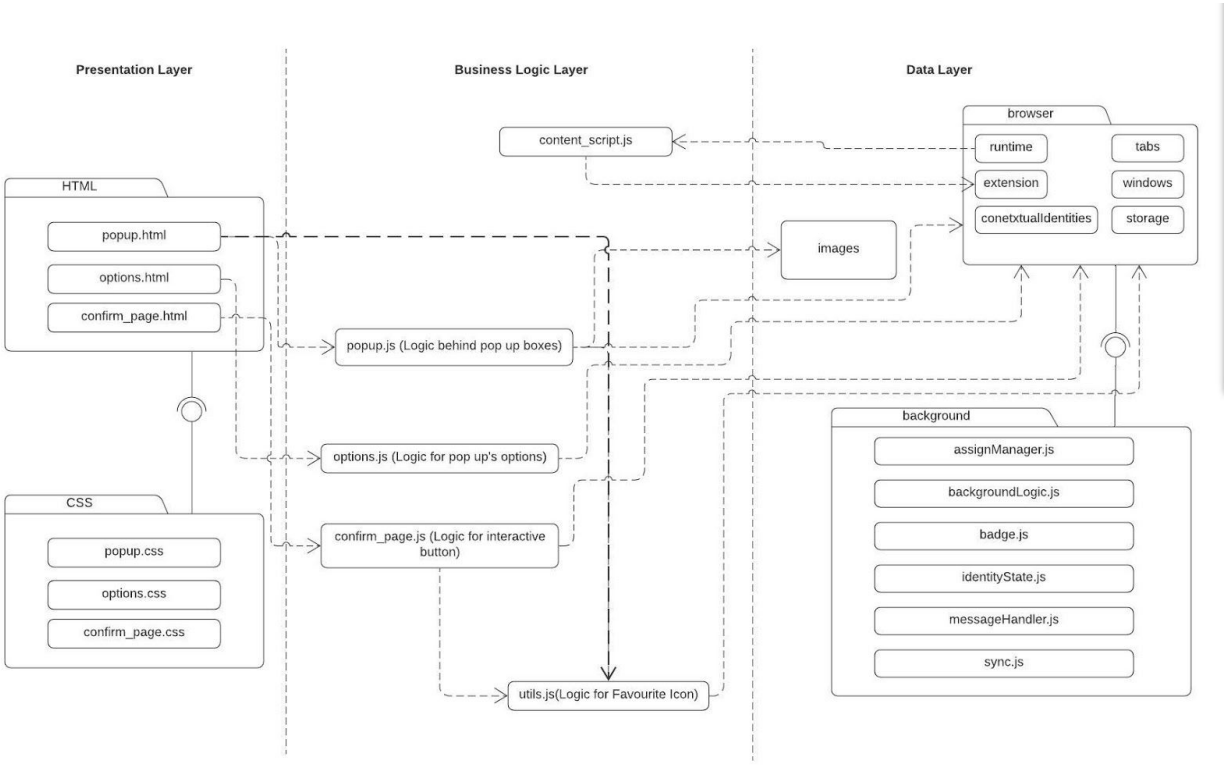
Customer Acceptance Test 3 (One window with different containers)

Step 1: Open one Firefox web browser window.
Step 2: Open a new tab of 4 different containers in this window.
Step 3: Click the "Sort Container by Windows" button.
Step 4: Verify number of Firefox windows opened. There should be exactly 4 Firefox windows open.
Step 5: Verify each window only has one tab, each of a unique container.
Step 6: Verify that all the tabs before the sort operation still exist in those 4 windows.

Customer Acceptance Test 4 (One window and one container)

Step 1: Open one Firefox web browser window.
Step 2: Open 4 new tabs of one container in this window.
Step 3: Click the "Sort Container by Windows" button.
Step 4: Nothing should happen because the windows are already sorted by container.

Architecture of the project



(Link: https://github.com/CSCD01/team_05-project/blob/master/docs/d3-architecture.pdf)

From analyzing the architecture of the project, we think it is following the **Layered Architecture**, where the Data Layer contains the background module, the Presentation Layer contains the HTML and CSS files, and Business Logic Layer contains the JavaScript files. We separated the project into these layers because each layer provides a specific set of services.

From the diagram, you can clearly see there is something different from the standard Layered Architecture. There is no Persistence Layer and Database Layer in the graph. This is because our project is very special, it is a Firefox web extension project. Thus there isn't really a database in here. However we do retrieve data from the browser. Hence in some sense, the browser can be our database and the background module will be the Persistence Layer. But the browser isn't really a database, so after carefully reviewing the definition of the Layered Architecture, we decided to combine those two together as a Data Layer.

More specifically, we think this is **Closed Layer Architecture**, because from reviewing the code base in detail, we didn't find any direct interaction between the Presentation Layer and Data Layer. This means it follows the idea of layers of isolation. Hence the change in one layer A will only affect the component A and A's associated layer. This can reduce the possible interdependencies between components/layers, so it improves the testability of the project.

This also improves the overall development speed, because this architecture pattern is relatively easy to implement and it falls with the natural way of the business application development, where the companies separate people with different skills into groups.

Another thing we noticed is that the project follows the **Single Responsibility Principle**. This principle means every module in a program should have only one responsibility, and the responsibility should be encapsulated only by the responsible class. In this project, they separate files according to their responsibility in the Frontend part. For example, popup.js only handles user interactions happening in popup.html, options.js only handles user interactions happening in options.html. Also, each function only does one thing and we can easily understand what it does by reading the function names.