Deliverable 4

BYTE

User Guide

1. Install Firefox from Mozilla's official website

(For the purposes of our project, run the following commands as well:

2. Open a terminal window in the root folder (the folder that contains the src and bin folders)
3. For Mac or Linux users, run '`npm install`' and '`./node_modules/web-ext/bin/web-ext run -s src/`'.
   For Windows users, run '`npm install -global web-ext`' to install web-ext globally and then '`web-ext run -s src/`' to run the code
4. After running those commands, a Firefox browser will open automatically)

5. Find the extension icon on the top right corner of the browser
6. Open several tabs and windows that belong to different containers
7. Find the button called "Sort Tabs by Windows" after going through the extension's instructions
8. All tabs will then be sorted by containers in separate windows (more detailed screenshots are located in the acceptance tests)

Feature 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

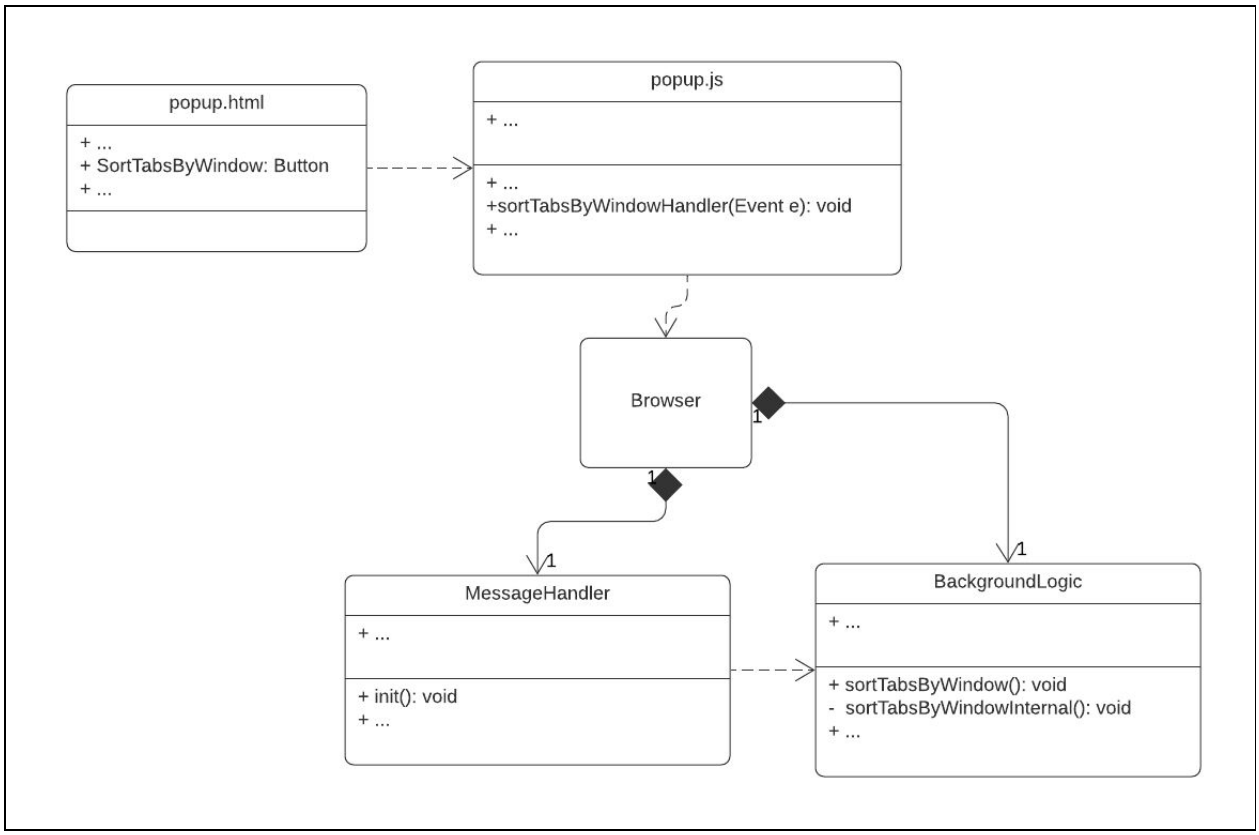
Window 4's tabs

- Default tab

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

Design

UML



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

As we can see from the UML diagram, we have four files (excluding `Browser`) related to the feature. The button in the HTML file `popup.html` is the only part with user interaction. The button click event is registered in the JavaScript file `popup.js`, which means once the "Sort Tabs by Window" button is clicked, the event handler will be triggered.

The `Browser` supports most of the Firefox API for addons. The other two classes `MessageHandler` and `BackgroundLogic` are in the background for Firefox. The 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.

`MessageHandler` and `BackgroundLogic` are created when the Firefox addon is run and are then stored in the `Browser`. Therefore, the handler in `popup.js` just passes the message to the

`Browser`. Then, it calls `MessageHandler` to handle the message. The real logic is in `BackgroundLogic`, and the handler will call it to do the real work.

The design we have in Deliverable 4 does not differ by the design we created in Deliverable 3.

Modified Source Code Files

Presentation Layer

`popup.html`
Line 131-132: Add a button labelled "Sort Tabs by Window" so that the user can see a new button in the top left of the main panel

`popup.css`
After line 496: Add CSS for the new button so that it is aligned with the existing "Sort Tabs" button

Business Logic Layer

`popup.js`
After line 580: Add an event listener for the new button so that the handler will be triggered whenever the new button is clicked

`messageHandler.js`
Add one case inside the `init` function for the message received from `popup.js` "sortTabsByWindow". Secondly, call the `sortTabsByWindow` function inside `background/backgroundLogic.js`

`backgroundLogic.js`
Add one function for moving the tabs according to the current window (around line 240):
After investigating the code, we decide to break the logic into different cases
Normal case (for tabs that are not pinned):

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

Pinned case (for tabs that are pinned):

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 with the pinned tabs as the first few tabs

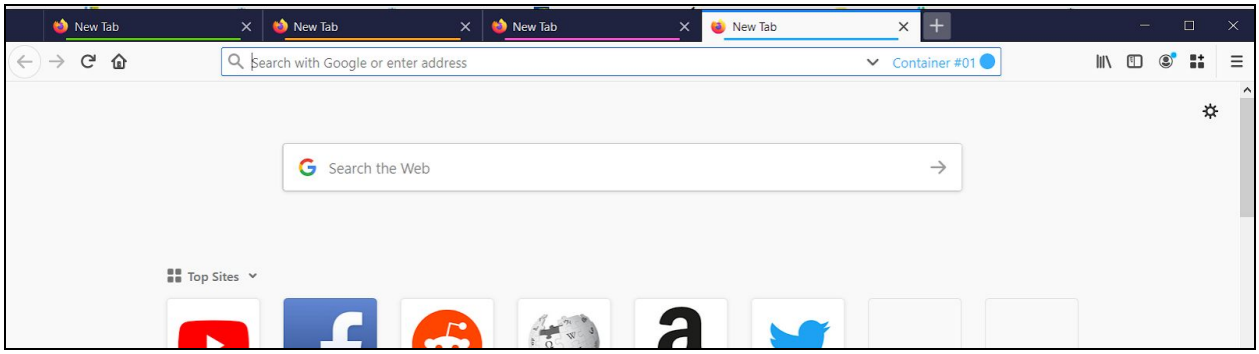
d. Record the tab ID for the tabs that need to be moved. Loop through the windows, delete all the tabs that do not belong to the window

Testing

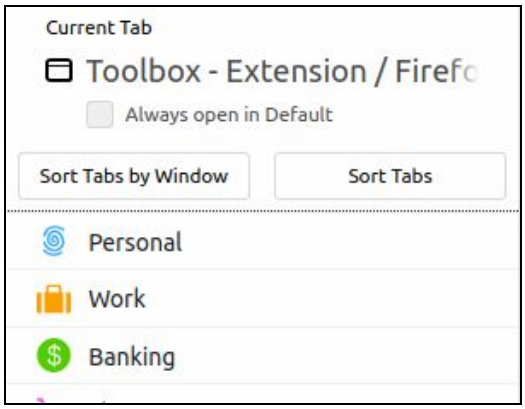
Acceptance tests

Test 1 (Multiple windows and different containers)

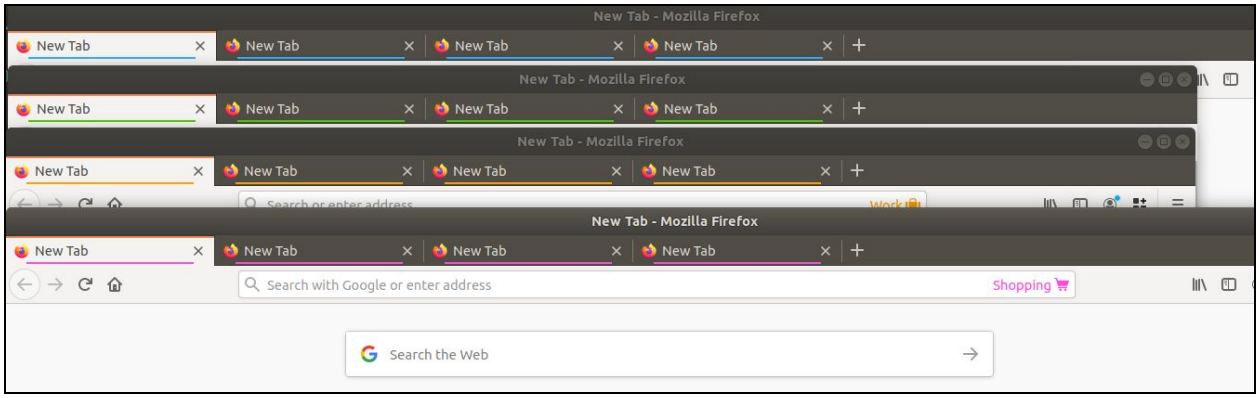
1. Open the Firefox web browser in three different windows.
2. Pick 4 containers to test the new feature.
3. For each window, open at least one new tab for each selected container.



4. Click the "Sort Tabs by Windows" button.



5. Verify the number of Firefox windows opened. There should be exactly 4 Firefox windows open.
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.



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.

Test 2 (Multiple windows and one container)

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

Test 3 (One window with different containers)

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

Test 4 (One window and one container)

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

Test 5 (One window with multiple containers, including the default container)

1. Open one Firefox web browser window.
2. Open a new tab of 3 different containers in this window, one of them being the default container (i.e. tab that is not assigned to any containers).
3. Click the “Sort Tabs by Windows” button in the addon popup menu.
4. Verify the number of Firefox windows opened. There should be exactly 3 Firefox windows open.
5. Verify each window only has one tab, each of a unique container.
6. Verify that all tabs before the sort operation still exist in those 3 windows.

Unit tests

Initial plans

We planned on performing unit tests on the functions we created by writing code that tests functionality similar to the ones we wrote for our acceptance tests. However, we came across problems in our attempts.

Problems

The functions `sortTabsByWindow()` and `_sortTabsByWindowInternal()` which we wrote for the background logic use the browser's `windows` object to detect whether the tabs are opened in the desired browser windows and whether the tabs have moved to that window. When we test these functions using the project's existing testing framework, we found that we cannot retrieve the browser's `windows` object that we want, as it always returns `undefined`. However, when we test the functions in the actual browser, we get the `windows` object we need.

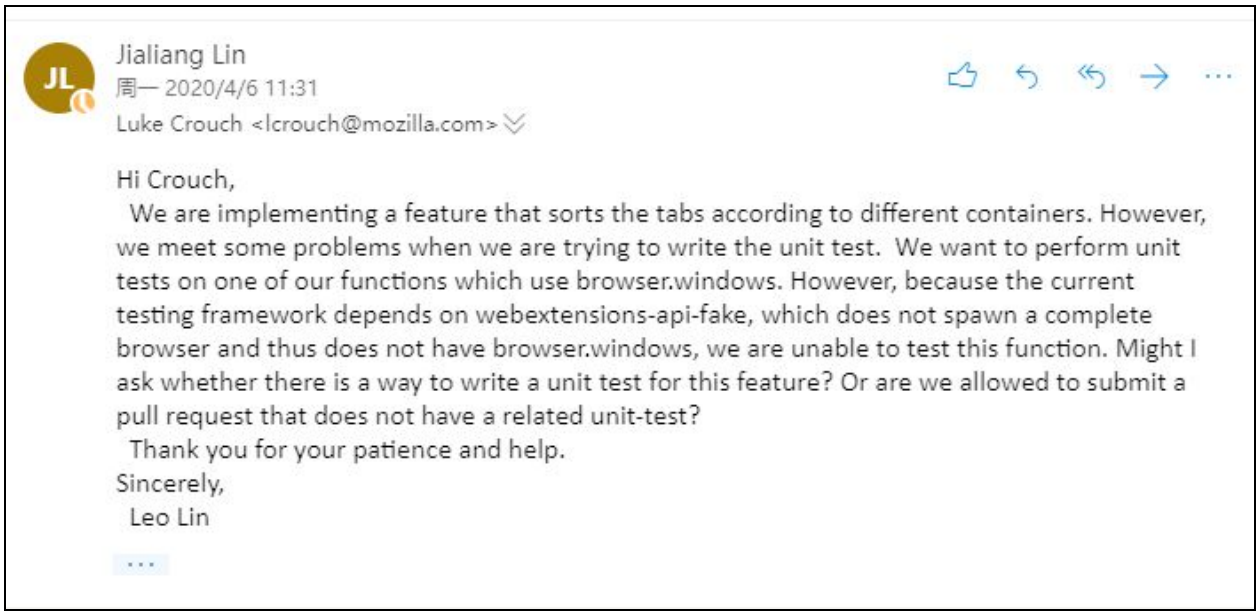
Investigation and attempts

After investigating further, we found that the project's testing framework uses and depends on `webextensions-api-fake`, which provides a working but fake implementation of the API in-memory, without spawning a complete browser. More importantly, this fake implementation of the API doesn't provide the `windows` object that we need.

We tried to install other packages to get the global browser, such as `mocha-jsdom` and `jsdom-global`, but it seems neither of them is compatible with this project.

We should stick to the testing framework they currently use to keep unit tests consistent and we shouldn't add new packages to the project without the maintainers' permission. Moreover, we shouldn't change the implementation of `webextensions-api-fake` and the code in `webextensions-jsdom` package which uses `webextensions-api-fake` because they're external packages and are not a part of the project.

Response from the maintainers



Jialiang Lin
周一 2020/4/6 11:31
Luke Crouch <lcrouch@mozilla.com>

Hi Crouch,
We are implementing a feature that sorts the tabs according to different containers. However, we meet some problems when we are trying to write the unit test. We want to perform unit tests on one of our functions which use browser.windows. However, because the current testing framework depends on webextensions-api-fake, which does not spawn a complete browser and thus does not have browser.windows, we are unable to test this function. Might I ask whether there is a way to write a unit test for this feature? Or are we allowed to submit a pull request that does not have a related unit-test?
Thank you for your patience and help.
Sincerely,
Leo Lin

We have sent an email to the maintainers about this problem and asked for suggestions or advice, but unfortunately, we still haven't received any reply from them.

We encountered this problem with unit testing on Friday, March 3, and sought to let our TA know as soon as possible. His suggestion was to contact the maintainers of the project. However, the maintainers of the project reply to emails on workdays only, which is why we sent the email on Monday, March 6.

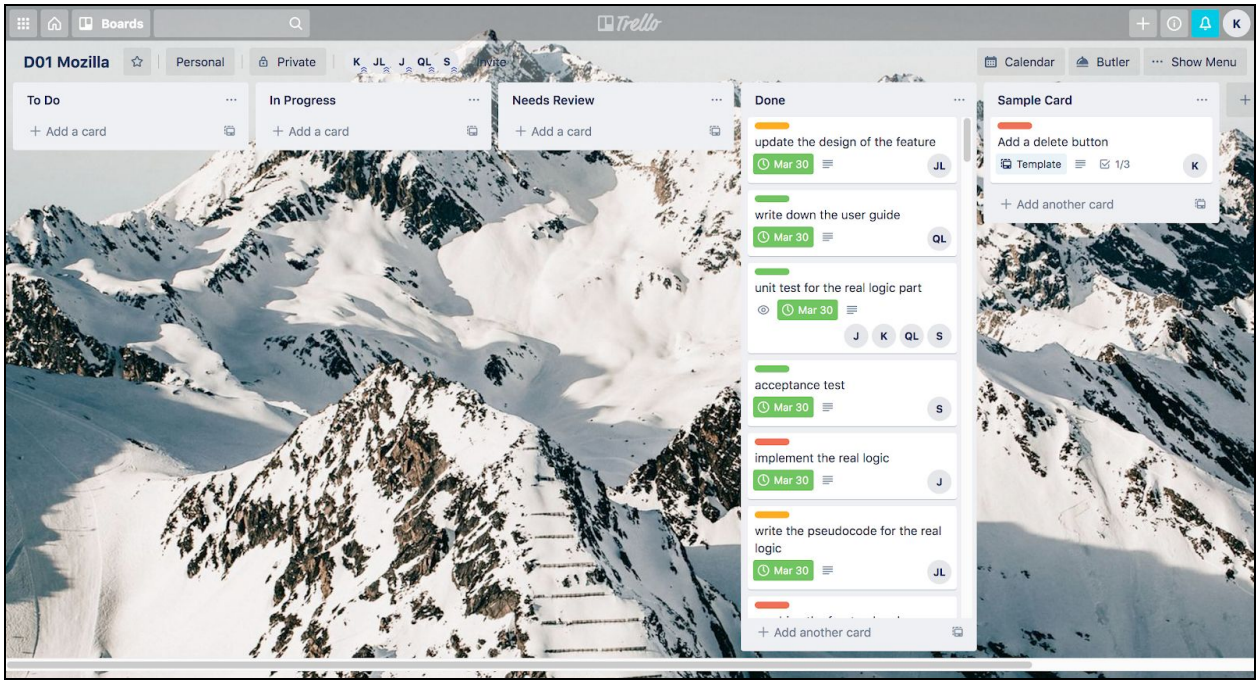
Solutions

Since this feature mostly focuses on the behaviour of the browser and user experience, we decided to combine the unit tests and acceptance tests and present them manually (i.e. in the TA meeting).

Software development process

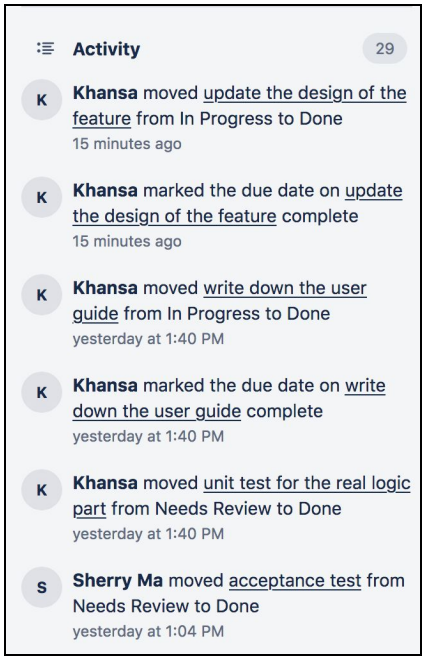
Kanban board

Our Kanban board is hosted on Trello, pictured below.



Link: https://trello.com/invite/b/HxNEWM5G/e95972a95b77b4e2c7e2d8f4a33b6617/d01-mozilla

The activity history is shown whenever the user clicks the menu.



Kanban card

For each Kanban card, we assigned a priority to it. For this deliverable, the card that is related to the main functionality or branch is assigned with high priority. The card that is related to documentation is assigned with medium priority. The card that refers to small changes (polishing UI) is assigned with low priority. For tasks that have a specific deadline (which is discussed in our standup meeting), a due date is added.

WIP limit

According to the activity history, none of our teammates has exceeded the WIP limit (10). In other words, none of our teammates has taken more than 10 cards under the **In Progress** section.

Daily stand ups

Attached is the link for our daily stand up meetings:
https://docs.google.com/document/d/1claVMfMSwhIJTSXqujMSIFz1SSRiUMxXy1pbYNNQ5F0/edit?usp=sharing

We modified Kanban's daily stand ups to stand ups that occur every two days.

Additionally, we held weekly face-to-face meetings on Wednesday before self-quarantine happened. To be more precise, on March 25, we reanalysed the design, estimated the workload and assigned the tasks on our Trello board. We summarised all the code and documentation during our second meeting on April 1st.

Git branch

We first created a branch for our issue called issue-1607. From this branch, we created a new branch for each subtask: user interface, background logic, and testing. A pull request is only allowed to be merged into the master branch when it is approved by all team members.