Deliverable #4

Team 10 - UTSGSC

Kanban Board Link:

https://utsgsc.atlassian.net/secure/RapidBoard.jspa?rapidView=3&projectKey=UFF&selectedIssue=UFF-4&atlOrigin=eyJpIjoiNjY1MGU0ZjM0Y2FkNDJhMjg5YmNmOTgxOTVkZjRlNDUiLCJwIjoiaiJ9

Forked Repo Link: https://github.com/CSCD01-team10/focus-android

Feature Request Link: https://github.com/mozilla-mobile/focus-android/issues/3808

PR to Firefox Focus Repo Link: https://github.com/mozilla-mobile/focus-android/pull/4492

Acceptance Test Suite

Complete these steps before any running acceptance test case:

1. Clone the repo https://github.com/CSCD01-team10/focus-android and follow the build instructions
2. Open Firefox Focus application
3. Skip all onboarding menus (if this is a fresh install)

Test 1: Check '@' sign displays all search engines available when search suggestions is turned on

Steps:

1. Tap the 3 vertical dots at top right corner
2. Tap 'Settings'
3. Tap 'Search'
4. Turn on 'Get search suggestions'
5. Go back to main screen by tapping back twice
6. In search box enter '@'

Expected Result: A list of all available search engines should be shown, which are amazon, google, duckduckgo, twitter, and wikipedia. Each item should have an icon correlating with the search engine.

Test 2: Check choosing search engine suggestion shows the user that they are going to search with the specified engine.

Steps:

1. Tap the 3 vertical dots at top right corner
2. Tap 'Settings'
3. Tap 'Search'
4. Turn on 'Get search suggestions'
5. Go back to main screen by tapping back twice

6. In search box enter ‘@amazon books’

Expected Result: User should be shown a bar that says “Books - Search with Amazon.ca”

Test 3: Check ‘@’ sign does not display all search engines available when search suggestions is turned off
Steps:
1. Tap the 3 vertical dots at top right corner
2. Tap ‘Settings’
3. Tap ‘Search’
4. Turn off ‘Get search suggestions’
5. Go back to main screen by tapping back twice
6. In search box enter ‘@’

Expected Result: A list of search engines should not be shown as suggestions.

Test 4: Check using search shortcut searches the correct search engine correctly
Steps:
1. In search box enter ‘@duckduckgo test search’
2. Tap enter button

Expected Result: Browser should direct to duckduckgo searching for ‘test search’.

Test 5: Check searching without shortcut still searches default search engine
Steps:
1. In search box enter ‘covid-19’
2. Tap enter button

Expected Result: Browser should direct to google searching for ‘covid-19’.

Test 6: Check entering search engine shortcut without search query should search shortcut in default search engine
Steps:
1. In search box enter ‘@twitter’
2. Tap enter button

Expected Result: Browser should direct to google searching for ‘@twitter’.

Unit Test Suite

Our user test suite can be found in our team repos code for this feature. Backend logic unit tests are in app/src/test/java/org/mozilla/focus/utils/UrlUtilsTest.java (https://github.com/CSCD01-team10/focus-android/pull/3/files#diff-2fb6fbda62deb25873c7ba8a919cbeb7).

UI Testing

As we attempted to implement UI testing for the features, we realized that it actually wasn't necessary to test the UI. Since the search engines are displayed in a recyclerview, the individual items in the recyclerview are hard to extract for testing. Since the dropdown by default is already a feature in the app, we didn't actually alter the UI, but instead we just replaced the suggestions shown. The only thing we can effectively test is to see if the dropdown is displayed after an "@" symbol is typed, as well as if the icon matches the search engine. Showing if the dropdown is displayed is trivial, as the app already has that functionality. Since the icon as well as the search engine name is generated in the code, it is not necessary to check the contents of each item.

User Guide

Feature: Adding a search shortcut in the URL bar
Description:
Allow users to use built-in shortcuts to search, similar to Firefox desktop, where prefixing the search with "@" will show a list of search engines as suggestions (Figure 1). Specifically, the search engines that are suggested are Amazon, Google, DuckDuckGo, Twitter, and Wikipedia.

To use Search provider shortcuts:

1. First open "Focus"
2. Type the "@" character into the url search bar
3. The search suggestions will display a list of search providers (engines) with their associated icons
4. You can type in more characters to narrow down the engine you want
   - For example: "@a" if you want to narrow it down to search engines that start with "a"
5. Press any search suggestion to automatically select it
   - For example: press the "@amazon" suggestion to automatically fill the url entry bar - now your search query will be for amazon
6. Type in your search query after the search provider shortcut (ie. "@amazon") to search directly on the search search provider's website
   - For example: your query for "monitor" will look like this: "@amazon monitor" on the url input bar
7. Press enter and you should be taken to the search page corresponding to your query, on the search provider's website
   - For example: if your url search bar contained "@amazon monitor", you will now be on amazon's search page, displaying results for monitors

To cancel a selected shortcut, simply backpress until the shortcut is no longer in the url input bar

Here is an example of the above search with screenshots included:

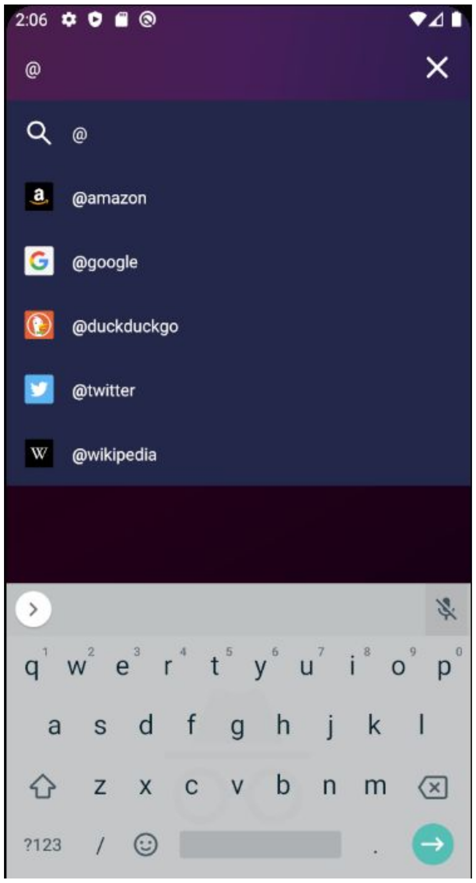


Figure 1: Search Engine Suggestions

The user can then select any of these search engines. Selecting a suggested search engine would add the text "@<Search Engine name> " to the URL bar. It is not necessary for a user to select a search engine from the suggested list, as this is shown for convenience and better user experience reasons. From there the user may enter their desired query and press enter. In summary, to use a search shortcut, the user's input on the URL bar must be as follows, "@<Search Engine Name> <Query>".

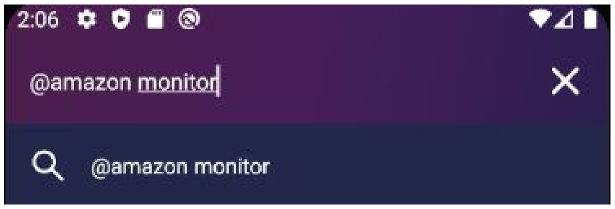
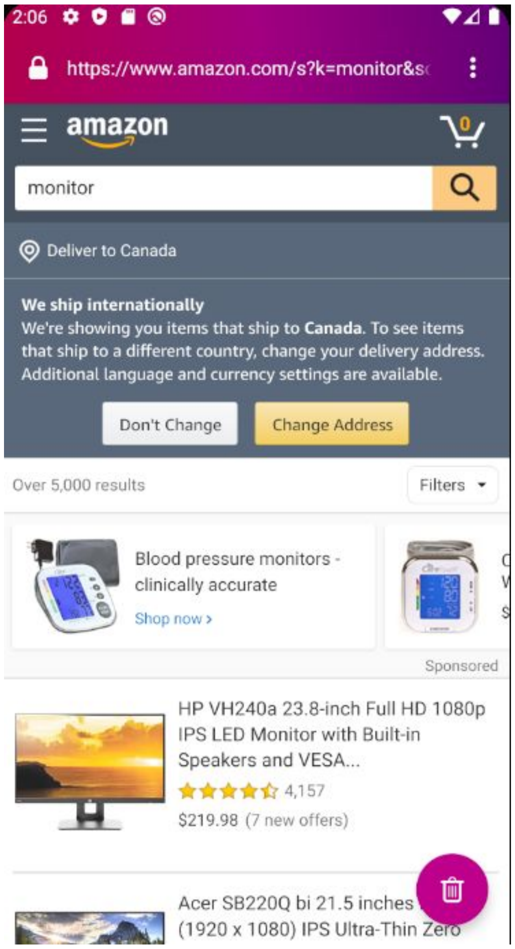


Figure 2: Search shortcut query format

Here, the user uses Amazon to search for a monitor. Once enter is pressed, the user will then be directed to the results of the query to the selected search engine (shown in Figure 3).



We ship internationally
We're showing you items that ship to **Canada**. To see items that ship to a different country, change your delivery address. Additional language and currency settings are available.

Blood pressure monitors - clinically accurate

HP VH240a 23.8-inch Full HD 1080p IPS LED Monitor with Built-in Speakers and VESA...
4,157
$219.98 (7 new offers)

Acer SB220Q bi 21.5 inches (1920 x 1080) IPS Ultra-Thin Zero

A page is then loaded from Amazon with the search results of “monitor”. The same behaviour could be expected from any supported search engine.

Figure 3: Example of using the amazon search shortcut to query “monitor”

Document design of code

When developing our feature, we wanted to make sure it was properly integrated with the existing system. Specifically, this meant we had to examine how search worked as well as leveraging existing classes such as the SearchProvider and SearchEngineManager to build an effective solution.

Context:
There are two parts to the feature we added

1. **Autocomplete suggestions for the search provider shortcut:**
   For example, we use the “@” before the name of the search shortcut to use it, so typing “@..” will show a list of matching search engines you can quickly enable
2. **Searching using search provider:**
   This encapsulates the actual search functionality which involves constructing the search url for the query based on the search engine and using it to create a new session.

Autocomplete suggestions for the search provider shortcut:

Context:
When initializing a search, the visual components representing the search and search suggestions functionality are opened. The UrlInputFragment is straightforward. It represents the search input component and subscribes to changes to the search input and notifies the SearchSuggestionsViewModel of changes. The SearchSuggestionsViewModel is in charge of handling the logic to retrieve suggestions for a given query when it changes and notifying its observers (SearchSuggestionsFragment) of the new suggestions exposed through an observable (LiveData) list. The SearchSuggestionsFragment uses a recyclerview to render each of the search suggestions.

When a search query is entered in the search bar, this is a high level overview of the process:

1. User changes search query
2. UrlInputFragment -> notifies the SearchSuggestionsViewModel of the changes
3. SearchSuggestionsViewModel -> fetch suggestions list using the SearchSuggestionsFetcher
4. SearchSuggestionsViewModel -> notifies the suggestion list observers
5. SearchSuggestionsFragment -> notified of change to search suggestions
6. SearchSuggestionsFragment -> renders search suggestions

Solution:
We injected our code into the SearchSuggestionsFetcher so that the suggestions it fetched included both auto-complete suggestions (which already exist in the app) and search provider suggestions. We did this by adding an additional check in the getSuggestions method to see if the query was the beginning of any search engine shortcuts (but more on the check will come later, for now we'll focus on the new flow). If it was, it would return a list of the matching search provider shortcuts. Because the list of search suggestions exposed by the SearchSuggestionsFetcher class is essentially a list of strings, we can easily introduce the new suggestions without modifying the existing API - the string is the "interface" we're programming to.

In this way, the SearchSuggestionsViewModel also does not need to be aware of the different search suggestions and can remain untouched. It is the SearchSuggestionsFragment that needs to be updated.

The SearchSuggestionsFragment renders the search suggestions using a recyclerview following the viewholder pattern. For reference, the viewholder pattern increases the efficiency of rendering items by caching the view in viewholder instances. Since viewholder instances are responsible for rendering individual search suggestions, we introduce a check in the SuggestionViewHolder to see if the suggestion is a search shortcut and render the appropriate search engine icon if it is.

Searching using search provider:

Context:
Once the search provider is selected and the user presses enter for the query, the UrlInputFragment calls its onCommit method to handle the search logic in its method. It uses the normalizeUrlAndSearchTerms to build an url from the query with the default search engine.

It’s important to note that the normalizeUrlAndSearchTerms method uses the SearchUtils to build the url from the query.

When a search is initiated, this is a high level overview of the process:

7. User changes search query and presses enter
8. UrlInputFragment -> calls its onCommit method
9. UrlInputFragment.onCommit -> call SearchUtils.normalizeUrlAndSearchTerms to build url
10. SearchUtils.normalizeUrlAndSearchTerms -> build url using default search engine
11. UrlInputFragment.onCommit -> creates a session using the url

Solution:
We once again injected our code into this method such that it also checks if the query uses a search provider shortcut and generates the appropriate url associated with the shortcut provider from it. By doing this, we continue to abstract the details of how queries generate urls from the UrlInputFragment.

It’s important to note that we rely heavily on the static processing functions we added to the SearchUtils class!

SearchUtils
The SearchUtils class just encapsulates a variety of search utility functions used by search classes. We added the bulk of the logic for handling search provider shortcuts here because

a. There was a high degree of cohesion between the functionality of the existing search utilities and the search provider shortcut logic
b. The search provider shortcut logic didn’t require state and felt much better as a set of static utility functions

We also created a static map between search provider shortcuts (Strings) and the name used to retrieve it from the SearchEngineManager (also a String). The beauty of the feature is that much of the complexity associated with building an url from a query as well as obtaining its icon has already been built out in the form of the SearchEngine class. Whenever we had a search provider string, all we had to do was get the search engine name associated with it using the mapping we created and then supply it to the SearchEngineManager to obtain the SearchEngine instance corresponding to it.

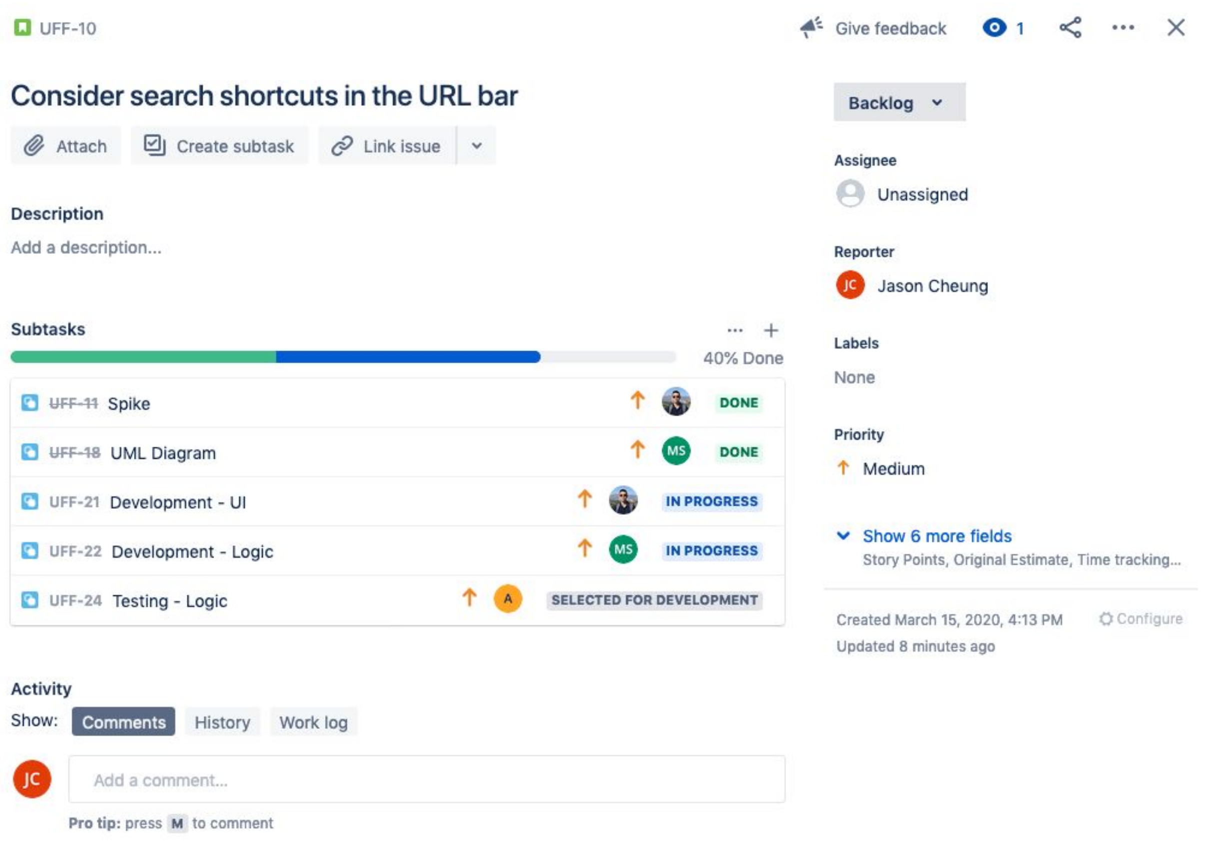
We also added a few utility functions:

- splitShortcutFromQuery - to split a string into the shortcut and query
- getURLForSearchEngineShortcut - to parse the input as an url using the correct search engine
- isSearchShortcut - to quickly check if we should render a search icon for a suggestion
- getSearchEngine - to get the search engine for a search engine suggestion so that we can render its icon

Evidence of Software Development Process

The following is a screenshot of our Kanban Board. There is a parent issue which is adding search shortcuts to the URL bar, **UFF-10.** Within the parent issue there are three subtasks which are similar to when we were previously working on our bug fixes. Namely, “Development - UI”, “Development - Logic” and “Testing - Logic”. These subtasks must be completed in order for an issue to be considered done. Our WIP Limit is 2 cards which is why you can see there are no columns with more than 2 cards in it except the “done” column. Deployment was done as soon as these subtasks were all in the “Done” column.

Note: The Kanban board filters out older issues that are in the “done” column to make the board less cluttered, but the parent issue still contains those subtasks.


UFF-10

Consider search shortcuts in the URL bar

Description
Add a description...

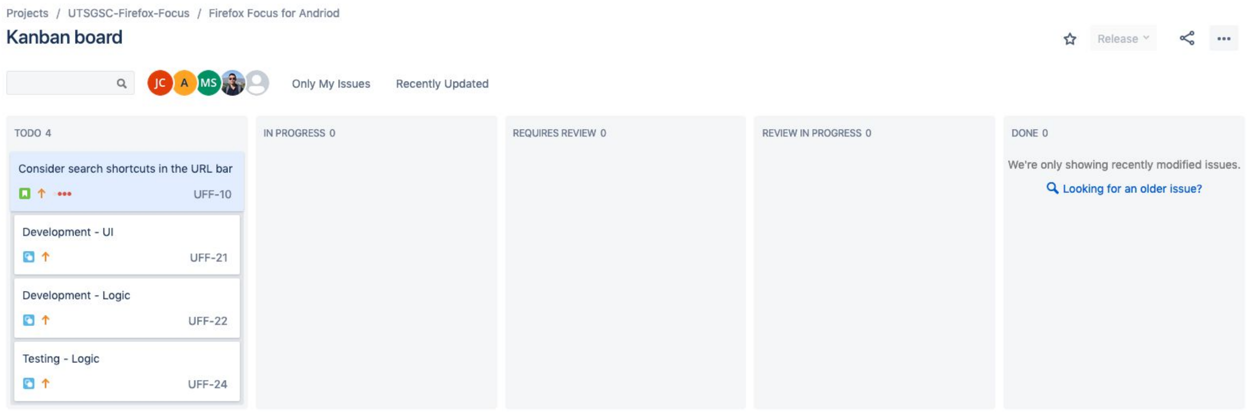
Subtasks — 40% Done

~~UFF-11~~	Spike		DONE
~~UFF-18~~	UML Diagram	MS	DONE
UFF-21	Development - UI		IN PROGRESS
UFF-22	Development - Logic	MS	IN PROGRESS
UFF-24	Testing - Logic	A	SELECTED FOR DEVELOPMENT

Assignee: Unassigned
Reporter: Jason Cheung
Labels: None
Priority: Medium
Show 6 more fields
Story Points, Original Estimate, Time tracking...
Created March 15, 2020, 4:13 PM
Updated 8 minutes ago

Activity
Show: Comments History Work log
Add a comment...
Pro tip: press M to comment

Figure 1: Issue that contains the appropriate subtasks (from deliverable 3 and 4).


Projects / UTSGSC-Firefox-Focus / Firefox Focus for Andriod

Kanban board

Release

JC A MS

Only My Issues   Recently Updated

TODO 4
- Consider search shortcuts in the URL bar — UFF-10
- Development - UI — UFF-21
- Development - Logic — UFF-22
- Testing - Logic — UFF-24

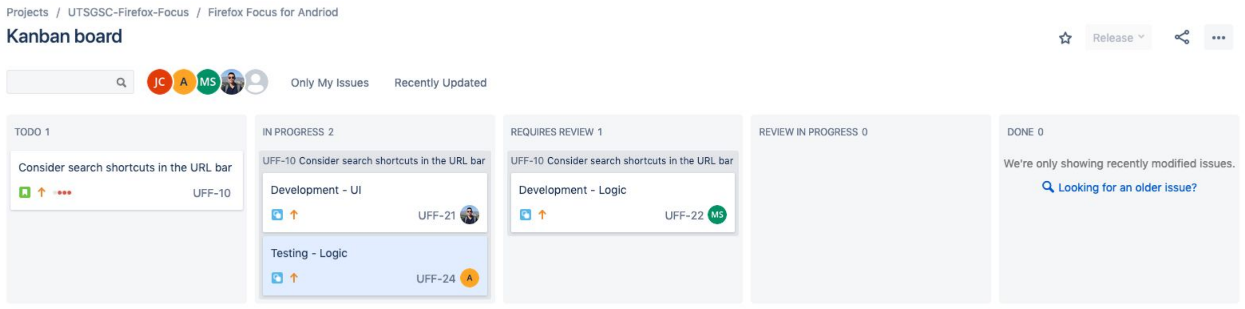
IN PROGRESS 0

REQUIRES REVIEW 0

REVIEW IN PROGRESS 0

DONE 0
We're only showing recently modified issues.
Looking for an older issue?

Figure 2: Our Kanban Board prior to starting the feature development.


Projects / UTSGSC-Firefox-Focus / Firefox Focus for Andriod

Kanban board

Release

JC A MS

Only My Issues   Recently Updated

TODO 1
- Consider search shortcuts in the URL bar — UFF-10

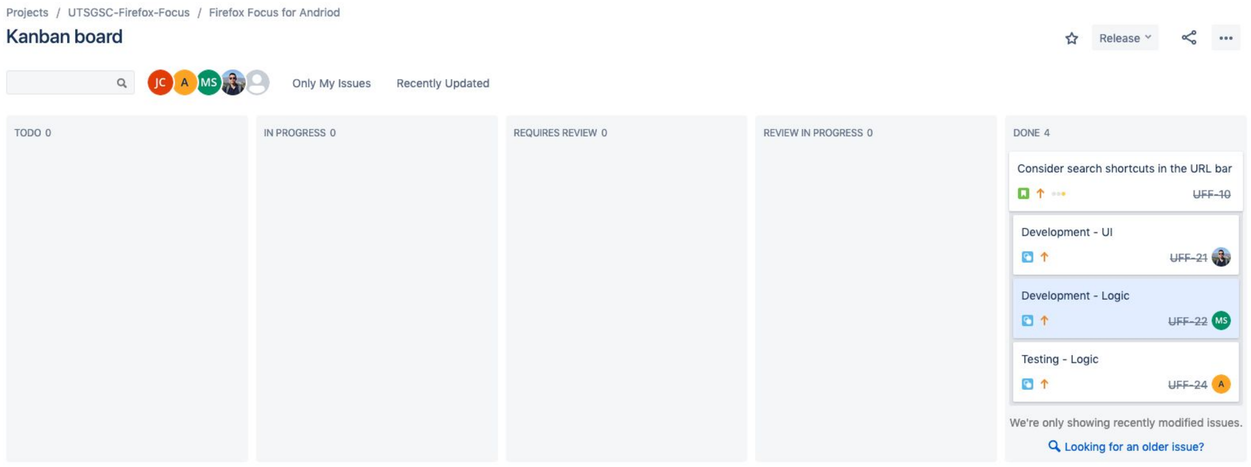
IN PROGRESS 2
UFF-10 Consider search shortcuts in the URL bar
- Development - UI — UFF-21
- Testing - Logic — UFF-24 A

REQUIRES REVIEW 1
UFF-10 Consider search shortcuts in the URL bar
- Development - Logic — UFF-22 MS

REVIEW IN PROGRESS 0

DONE 0
We're only showing recently modified issues.
Looking for an older issue?

Figure 3: Our Kanban Board while we were doing feature development.


Projects / UTSGSC-Firefox-Focus / Firefox Focus for Andriod

Kanban board

Release

JC A MS

Only My Issues   Recently Updated

TODO 0

IN PROGRESS 0

REQUIRES REVIEW 0

REVIEW IN PROGRESS 0

DONE 4
- Consider search shortcuts in the URL bar — ~~UFF-10~~
- Development - UI — ~~UFF-21~~
- Development - Logic — ~~UFF-22~~ MS
- Testing - Logic — ~~UFF-24~~ A

We're only showing recently modified issues.
Looking for an older issue?

Figure 4: Our Kanban Board once we finished feature development.

Meetings and Feedback Loops:

Each of these meetings had every member present. In between these, there were also smaller **daily syncs** in the Facebook Messenger group chat so each team had visibility into everyone's progress.

- Tuesday March 24, 2020: Distribute work between team members.
- Thursday March 26, 2020: Group Status Meeting
- Saturday March 31, 2020: Technical meeting team to discuss challenges of implementation details.
- Thursday April 2nd, 2020: Group Status Meeting including a demo to show completed feature within team

Planned deliverables for each phase

- Specification/Requirements Engineering
    - Document outlining description of the issue, expected UI and UX of feature.
        - For us, this document is the **GitHub issue page** for the feature we built because it contained this information, some of which was provided by Mozilla engineers.
- Design and Development
    - Development should produce a working feature that adheres to the acceptance criteria of the **GitHub issue** and a WIP: Pull Request in the forked repo for other members of the team to review.
    - WIP because we need to have tests written and on this branch before proceeding.
- Testing
    - Test(s) that are able to verify any newly introduced code to ensure it works as intended.
    - If development for the feature is approved, testing can be added onto that branch directly. If development for the issue is not approved, a separate branch can be created, reviewed with a PR and merged into the branch using for fixing the issue when approved.