Send A:
Project Deliverable 2

Team Members:
Anthony Le,
Birathan Somasundaram,
Pratana Atikhomkamalasai,
Suxin Hong,
Yuewen Ma

Table of Contents

List of issues considered

1. Choose where to save files downloaded from the web

   https://github.com/mozilla-mobile/focus-android/issues/2787

   The problem of current product:

   Currently, Firefox-focus will create a default directory when saving any file from the browser and users will have to find this file manually. There is no option to select the destination directory for downloads.

   Feature that user wants:

   Let users choose where they would like to save the files similar to the "save as" option when downloading files in windows, it should prompt users to select a directory from a directory chooser UI with "Download" as the default option.

   Estimate amount of work:

   This feature requires the team to know:

   1. The flow of the download action; where the initial download dialog is activated, where the file information is modified and saved, where this information is passed and where the download request is created.
   2. We need to modify the initial download dialog to prompt our directory chooser in **onCreateDialog** in **DownloadDialogFragment.kt.**
   3. We also need to save the destination directory from the directory chooser to **pendingDownload.**
   4. we need to link the listener to call **onFinishDownloadDialog** to **queueDownload.** Both **onFinishDownloadDialog** and **queueDownload** are in **BrowserFragment.kt**.
   5. In the frontend, the team will need to implement or import a third party library directory chooser, whIch we estimated to take at least **2 days** to learn and use

6. In the backend, the team will have to make sure the destination directory from directory chooser is saved and passed along. Backend can be implemented directly and finished within a day, since we only have to modify the code to save the destination directory.

This feature requires both backend and frontend work and in total, it should be completed within 3 days.

2. Replace the delete button by a panel menu

https://github.com/mozilla-mobile/focus-android/issues/3873

The problem of current product:

In Firefox Focus, users clean the browsing history by clicking the “Delete” button at the bottom right of the screen. After the browsing history is deleted, Users need to use Android’s back buttons to go back to the previous page. This is not fast or intuitive.

Feature that user wants:

Swiping from left/right side of the screen will open an expanded panel where some predefined actions are available where some actions are also collapsed and can be further expanded. This wanted feature can be compared to a similar feature called ChromePie, which can be seen in the following video: https://www.youtube.com/watch?v=Btn4qGeJp-c

Estimate amount of work:

This feature requires the team to know:

1. How to detect the trigger actions made by the users (when users are touching the side of the screen),
2. How to display a pie controller on the screen for choosing sub actions (expanding an option)
3. Requires some backend work as in making the full screen and customizing the pie items. We would need to replace the **floating**

erase button that the product currently has and add new UI components and triggers for user swipe actions.

4. The frontend work might require **3 days** since the team needs to learn how to listen to trigger events and use animations to show the menu and submenu,
5. The backend work might require another **2 days**, since the erase history function does not need to change (we can reuse it), some other functions like go back or full screen requires the team to learn and use.

Thus, in total this feature might take the team **at least 5 days** to complete.

3. Make the delete button movable

https://github.com/mozilla-mobile/focus-android/issues/4477

The problem of current product:

Current “Delete” button is stuck to the bottom right of the screen.
It prevents part of the screen from being seen. The position is also close to where the home button is, which is where users normally hold their phone. Users might click the button accidentally when browsing. This is not user friendly and can make the user unintentionally exit the current browsing session.

Feature that user wants:

Make the button moveable like the AssistiveTouch on iPhone. It can either be stuck to the middle left or right side of the screen and the functionality of deleting browsing history remains.

Estimate amount of work:

Adding this feature involves modifying the current **FloatingEraseButton** component in the program. The team needs to override the **onTouchEvent** so that the button can listen to user movement actions. It is only frontend work, where the team has to add event listeners that involve pressing,

dragging and releasing the button. We estimate that this feature should be done **within 2 days**.

4. URL autocomplete entry editing is ignored

   https://github.com/mozilla-mobile/focus-android/issues/4445

   The problem of current product:
   In the current browsing session, when the user tries to fill in the URL entry an autocomplete URL is presented, and when the user edits this autocomplete URL and presses enter, the browser goes to the autocomplete URLs website, instead of the manually typed one.

   Feature that user wants:
   When editing the autocomplete URL, the browser should go to the intended manually typed website and not the autocomplete website underneath.

   Estimate amount of work:
   This bug needs the team to look at **UrlInputFragement** and **SearchSuggestionsFragment** which deal with the user's input URL and auto completion, respectively. The team also needs to trace inside the **BrowserFragment** to see how the program loads the url when the url is auto completed. The total work time would be around 2 days since we would have to reproduce this bug on the specified version of the product and machine and then look into the issue.

5. Convert widgets classes to Kotlin

   https://github.com/mozilla-mobile/focus-android/issues/4095
   The problem of current product:
   Currently, there are both Java classes and Kotlin under the widgets directory where some UI elements are stored.

   Feature that user wants:
   Convert all the Java files under widgets directory into Kotlin files.

Estimate amount of work:

All the team members must understand current Java classes and their relationships with other components so that when converting those files, the structure of the program will not be damaged. Also the team must learn Kotlin and be sufficiently familiar with it. **AboutPreference, DefaultBrowserPreference, DrawableWarpper, FloatingActionButtonBehavior, FloatingEraseButton, FloatingSessionsButton, LocaleListPreference, MozillaPreference, ResizableKeyboardCoordinatorLayout, ResizableKeyboardLinearLayout, ResizableKeyboardViewDelegate** and **ShiftDrawable** are the 12 java classes that need to be converted. The work should take **at least 3 days,** 1 day to learn Kotlin basics and another 2 days to convert the code.

Selected issues

Choose where to save files downloaded from the web

This feature is not complicated and does not require deep knowledge of the application when compared to other features. We only need to know the sequence diagram for downloading. Apart from importing and implementing the third party directory chooser, it only requires a new destination directory to be saved in the download object and the download request-creating function to use that directory. Implementing the directory chooser can be challenging but saving the destination directory is very straightforward . We only need to make a small modification to the Download object and as for testing , we only have to test if the file is downloaded into the right folder with the right name. This will be tested manually since we are primarily dealing with UI components.

The moveable button

This feature is interesting to work on since none of us have tried to make this before and it only requires us to analyse and evaluate the button component. This feature is not overly complicated so we can have two people working on it and it will, without a doubt, be completed before the deliverable is due. Also, since this feature focuses on the frontend, it would be difficult to write UI tests for it and thus it would have to be manually tested, which saves some work.

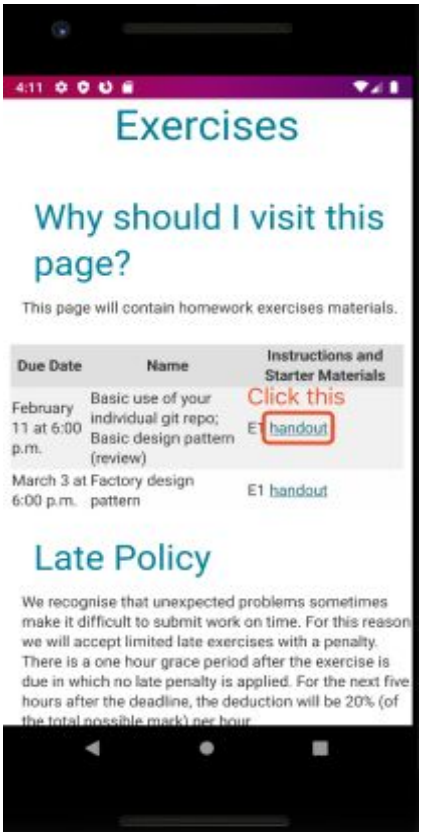
How to test these two features

Choose where to save files downloaded from the web

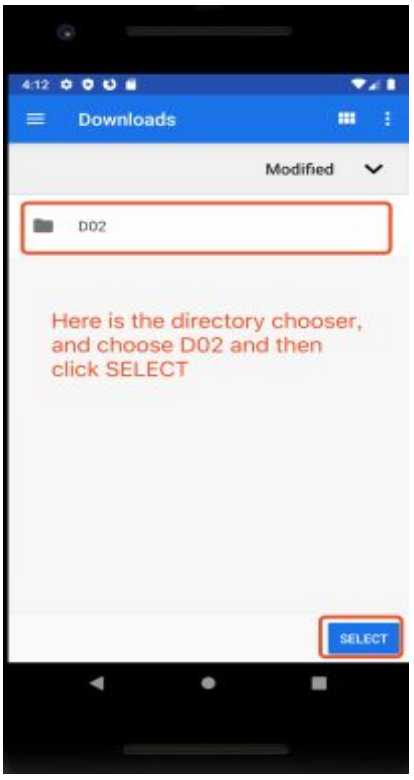
We did not implement unitTests since there are too many dependencies and classes to be mocked. We also cannot select directories from the backend. It is only possible for us to test the **parser** function in PathClass. In addition, The feature we implemented is mostly the UI. For manual testing, Users can open the app and try downloading the file.
A directory chooser should be prompted, users should be able to pick any destination directory and continue using the application afterward and finally, users should be able to see the file saved in the intended directory.

1. Search and go to D01 website.

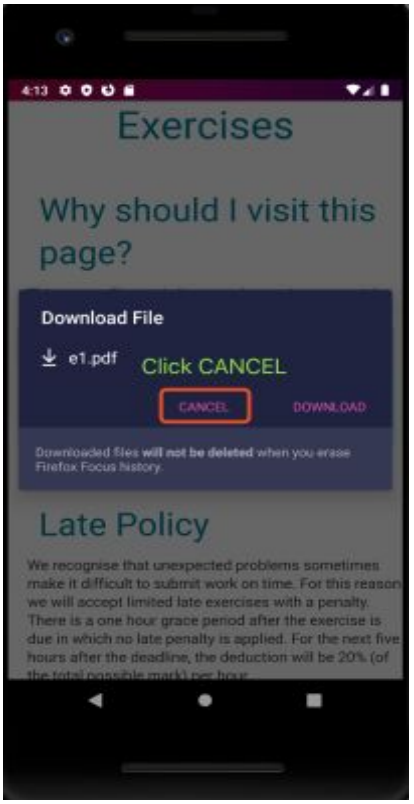
2. Go to Exercises page.



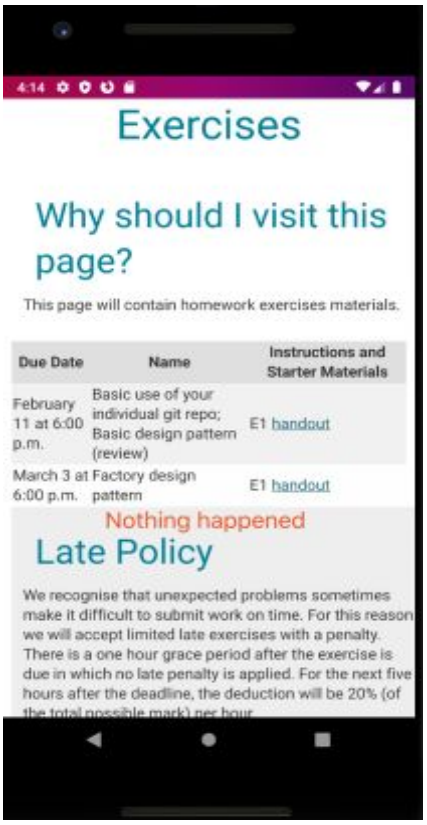
3. Click E1 handout and try to download.



4. Choose a directory and SELECT.

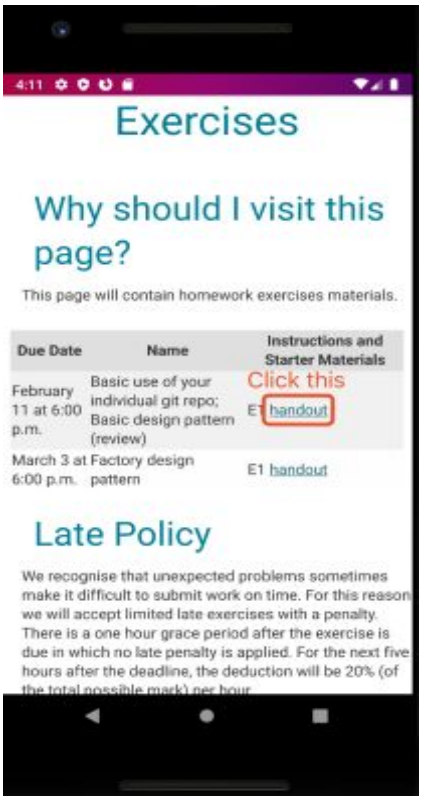


5. Try CANCEL Download.

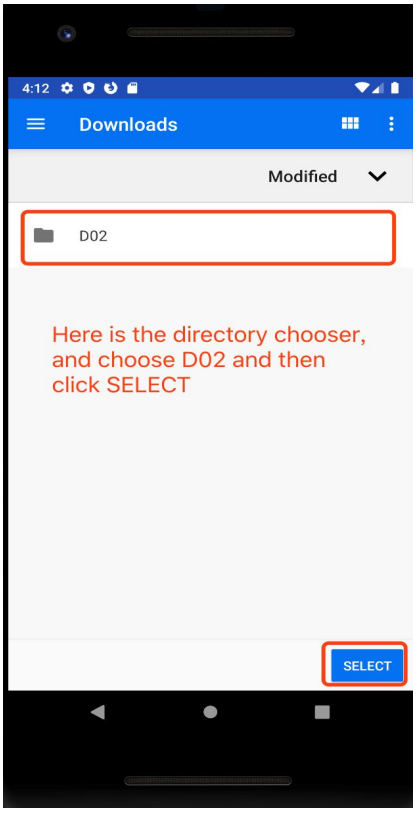


6. None downloaded.

7. None downloaded.

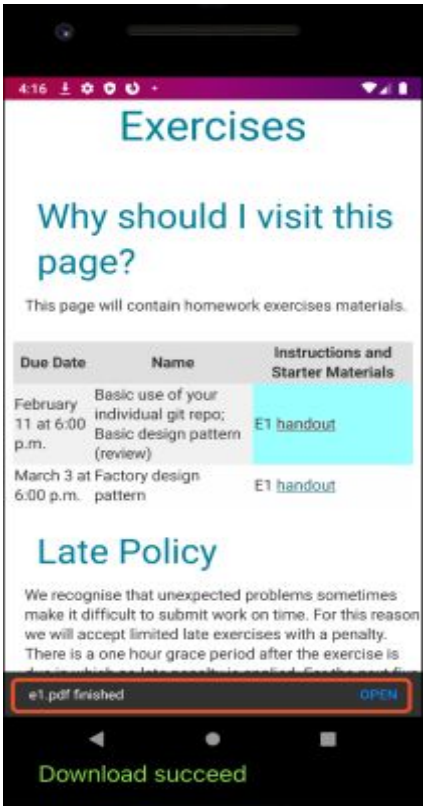


8. Click handout again and try downloading this time.



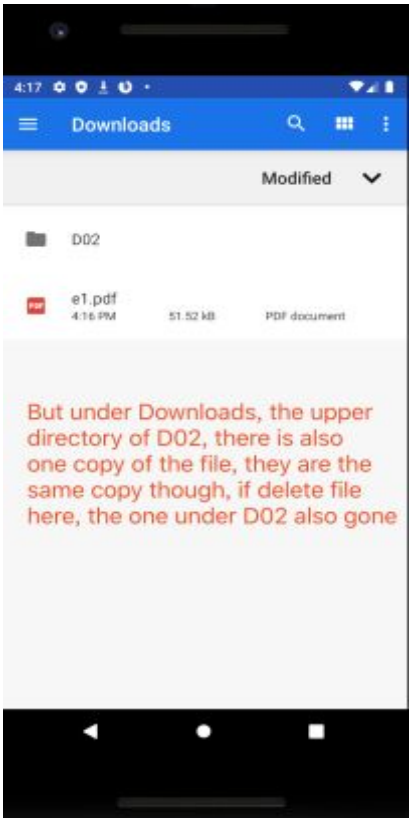
9. Choose D02 and SELECT.

10. Click DOWNLOAD this time.

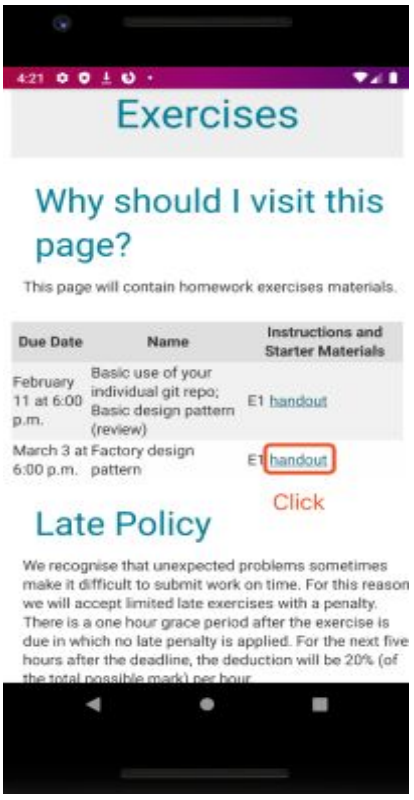


11. A message shows when download is completed.

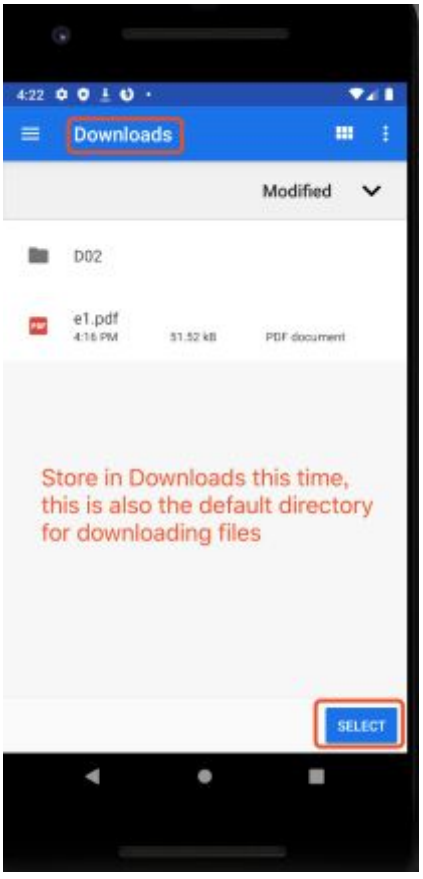
12. Go to Files and check.



13. The issue we met (the original code has the same problem).

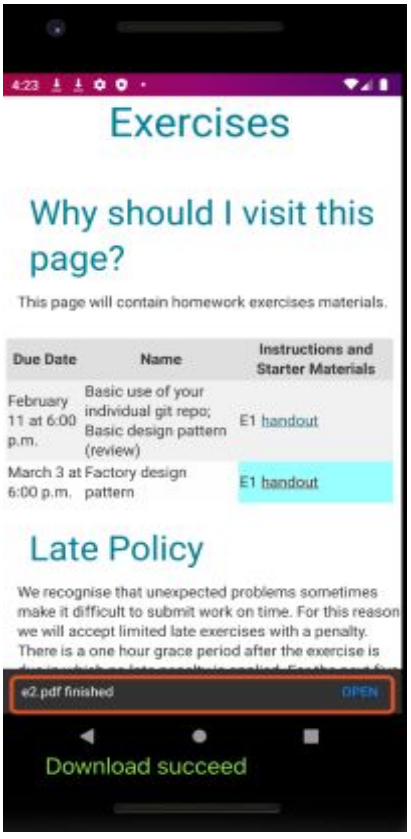


14. Try another download.

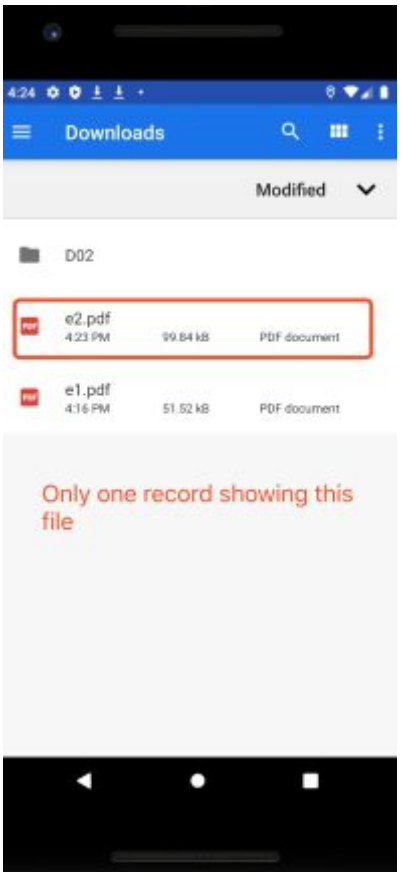


15. Save under Downloads directory this time.

16. Click DOWNLOAD.



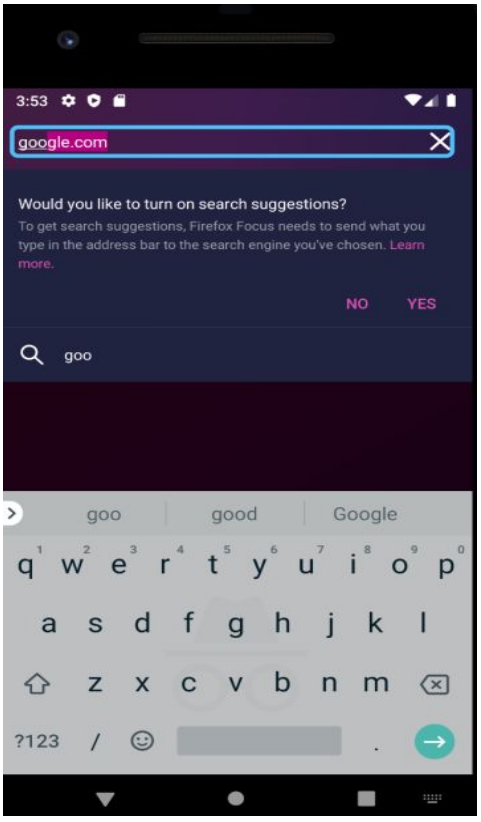
17. The download is completed.



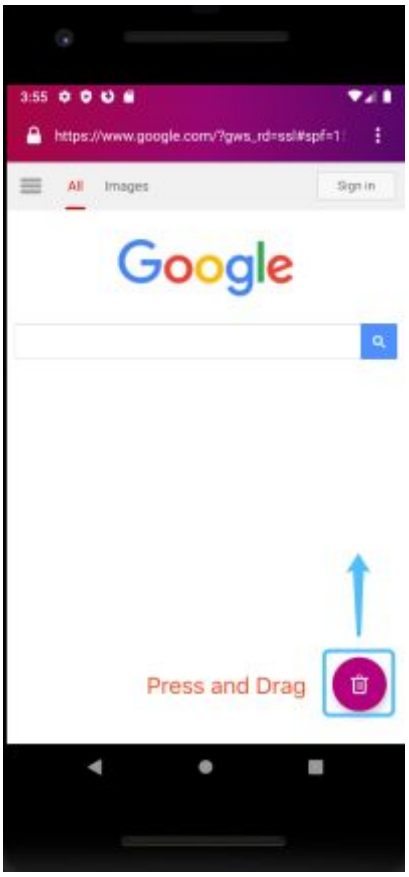
18. Go to Files and check this time, only one record.

The movable button

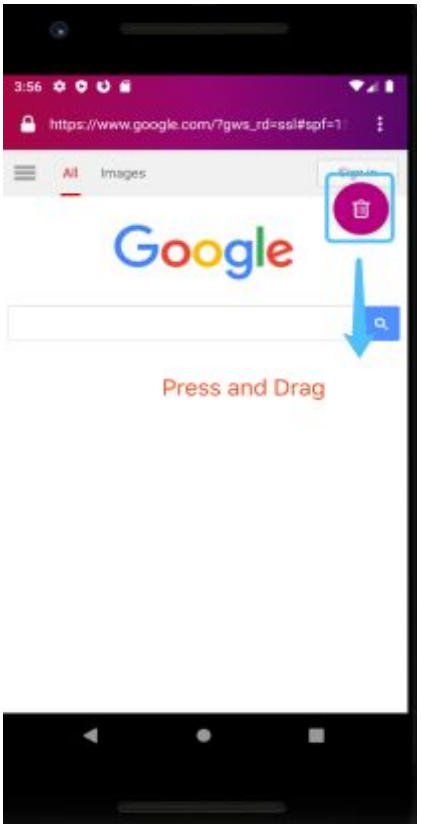
Launch the app and go to any website, then the user will see the “Delete” button displayed on the bottom right corner of the screen by default. Then the user can try to drag the button on the screen and as soon as the dragging stops, the button should stick to the left or right side of the screen, depending on which is closer. If the user clicks the button, the erase browsing history should continue to work as it did before.



1. Go to google.com



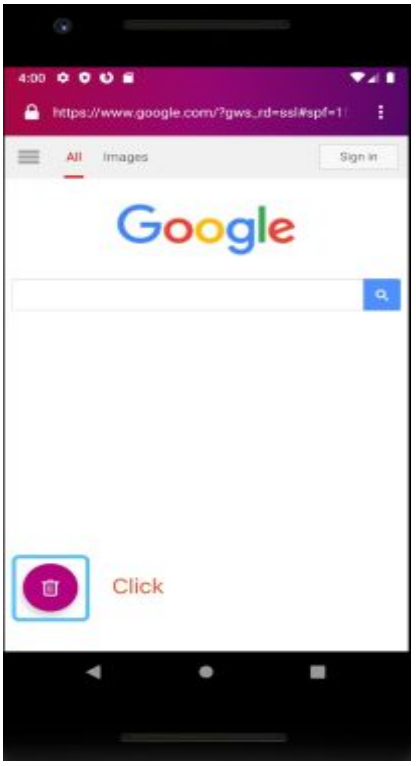
2. Try dragging the delete button up.



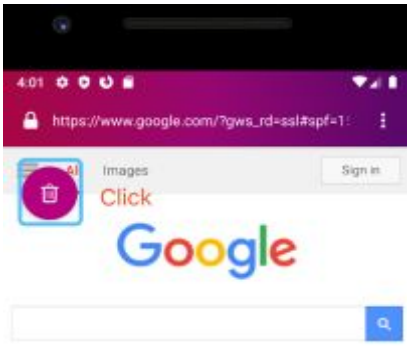
3. Try to drag down.

4. Try to drag to the left.

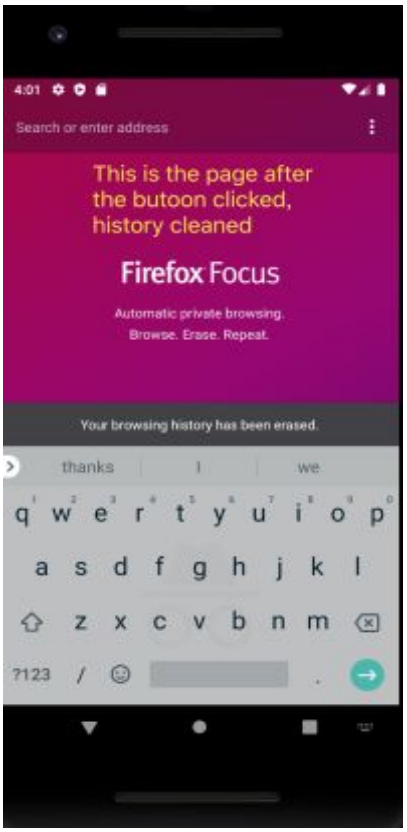
5. Try to drag up or down along the left side.



6. Clicking the button will clean the history.



7. Clicking the button will clean the history.



8. History cleaned.

Technical commentary

Choose where to save files downloaded from the web

The feature is mostly implemented and working as intended, but the bugs we encountered are more complicated than we thought. There are some unforeseen behaviours that we did not know of or figure out how to account for.

Users can and should choose the directory inside the Downloads folders. There are many dependencies we have to consider when parsing the directory path. Storage tree is different for every different android device. Path from SD cards is also machine dependent.

Another problem is when changing the download destination directory, two copies of files are created. One is saved into the default directory “Downloads” and the other is saved into the chosen destination directory. Only 1 copy is created if the default destination is used. We were not able to debug it and find the cause. It is likely due to the way download is implemented in android. Even though two copies are created, they share the same reference. If one is deleted , the other is deleted as well.

For the fix, we only modified the **BrowserFragment.kt.** We prompt a directory chooser for the user to select a directory. We save that path to a static variable in Path class. We parse and retrieve this path variable. We then save it to the new Download object when adding into the queue waiting to be downloaded.

In the function onDownloadStart:
when the user presses download on a document file like pdf, a new intent will be created and the directory chooser will be displayed. The directory chooser

is attached to the Activity by creating a new Intent to open a document tree. After the activity is started, the user will be able to pick a directory.

In the function onActivityResult:
When the user finishes choosing the directory, this function will store the absolute path that the user chose for our later use, and this function only has the result when the activity ends.

In the class Path:
There is a static variable that stores the default destination directory path and a parser function to get the relative path from the path received in onActivityResult. Also, there is a reset method to reset the download path after we overwrite the download object in onFinishedDownloadDialog.

In the class onFinishedDownloadDialog:
This is where we overwrite the download object by updating the user's choice of path, we have a helper function to build the new download object and we simply replace the old object with the new one and put the new one into the download queue.

The movable button

The **FloatingEraseButton.java** is the only file modified (It is under app/src/main/java/org/mozilla/focus/widget/). We added to the onTouchEvent function and overwrote the button ‘s behaviours by listening to the user's action to the button: press the button, drag the button and release the button and we also implemented a function to check if the button is moved or not.

In the function onTouchEvent:
case MotionEvent.ACTION_DOWN is the part for listening to the press button action, we save the start position and get the range of the area that the button can move;
case_MotionEvent.ACTION_MOVE is for listening for button dragging actions, we simply calculate the distance the button moved so that we can store the new position of the button and wait for the user to release the button;
case_MotionEvent.ACTION_UP is for listening for the button-releasing action, when users release the button, we use animations to attract the button to stick to the side of the screen, it depends on the area where the user release it, if it is released on the right half area of the screen, attract to the right, otherwise, attract to the left.

The function isNotDarg is used for checking whether the user is dragging the button or not where the isDrag is a global boolean value and we modify its value in case_MotionEvent.ACTION_MOVE.