Send A:

Project Deliverable 2

Process Progress

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

Table of Contents

Introduction

The team is divided into two groups, one group with 2 people (Suxin and Yuewen), the other group with 3 people (Anthony, Birathan and Pratana). One group is after one of the features we chose to work on.

Each subgroup will have their own meetings during each phase of our waterfall model, and each group has to provide a meeting note about each development phase along with a well explained outcome of each phase so that the whole team will understand the project through reading this documentation.

The outcome of Requirement Specification will give the details of the user requirements, with this, all the team members can gain enough idea about what exactly the users are looking for in the feature.

Then, the outcome of the next phase, Component Analysis, will give members an idea of where changes need to be made and how affected components interact with each other, so that people in the group will have a clear idea about what should be implemented and what can be reused.

During the System Design and Reuse phase, each group should have their own tasks that are assigned among group members, taking into consideration reusable parts to cut down on total work. Using Trello to track all the tasks progress helps the team to be aware of how the solution is designed and the implementation is going.

After the Development and Unit Testing phases, each group should give a detailed test flow and test cases with the implemented features, so that the whole team understands how the new features work and all of its functionalities.

Lastly, Integration and System Validation will let the team know whether the features work all together as a whole system.

With the waterfall model and the reuse oriented development process, the team can save some work with reusable components and communication with the process outcomes. These outcomes will help each member to have enough information about where the

implementations should be done and how the whole team is progressing during development.

Movable Button:

1. Requirement Specification

Meeting Running:

Members: Suxin and Yuewen

Date: Mar 2nd, 2020

Duration: 20 mins

Purpose: Understand the user requirements and the outcome of this meeting should be about a detailed specification of the user requirement.

Outcome: Detailed specifications of the user requirements.

Meeting Outcome:

Let the “Delete” button at the bottom right corner of the screen be movable, users can drag the button to anywhere on the screen and stick to the nearest edge of the screen according to the last position where the user release the button and the delete function should remain the same. An existing example of this feature is the Assistive Touch on iPhone.

2. Component Analysis

Meeting Running:

Members: Suxin and Yuewen

Date: Mar 2nd, 2020

Duration: 1.5 hours

Purpose: Analyse the existing components in the program and the outcome of this meeting is the details about the component we need to modify.

Outcome: Details about the components and functions we need to modify.

Meeting Outcome:

```
private fun initialiseNormalBrowserUi(view: View) {
    val eraseButton = view.findViewById<FloatingEraseButton>(R.id.erase)
    eraseButton.setOnClickListener(this)

    urlView!!.setOnClickListener(this)

    val tabsButton = view.findViewById<FloatingSessionsButton>(R.id.tabs)
    tabsButton.setOnClickListener(this)

    val sessionManager = requireComponents.sessionManager
    sessionManager.register(object : SessionManager.Observer {
        override fun onSessionAdded(session: Session) {
            tabsButton.updateSessionsCount(sessionManager.sessions.size)
            eraseButton.updateSessionsCount(sessionManager.sessions.size)
        }

        override fun onSessionRemoved(session: Session) {
            tabsButton.updateSessionsCount(sessionManager.sessions.size)
            eraseButton.updateSessionsCount(sessionManager.sessions.size)
        }

        override fun onAllSessionsRemoved() {
            tabsButton.updateSessionsCount(sessionManager.sessions.size)
            eraseButton.updateSessionsCount(sessionManager.sessions.size)
        }
    })

    tabsButton.updateSessionsCount(sessionManager.sessions.size)
    eraseButton.updateSessionsCount(sessionManager.sessions.size)
}
```

Figure 1: Where the erase button initialized

This feature requires us to modify the existing **FloatingEraseButton**. Since we need to keep the original function work so that we have to be careful we won't break the existing relationships with other components. We looked into the **BrowserFragment** where the widget has been called (Figure 1). Also, by adding and overriding the **onTouchEvent** function in **FloatingActionButton** to listen to users' actions, we can make the button ready for dragging. In the end, we still need animation to attract the button to the edge and this is why we also need to import the ObjectAnimator component in Android.

3. System Design With Reuse

Meeting Running:

Members: Suxin and Yuewen

Date: Mar 3rd, 2020

Duration: 30 mins

Due Date For Tasks: Mar 8th, 2020

Purpose: This meeting is for designing tasks and assigning tasks to members to finish. The outcome of this meeting is tasks designed and we should use Trello to keep track of each task.

Outcome: A solution based on component analysis

Meeting Outcome:

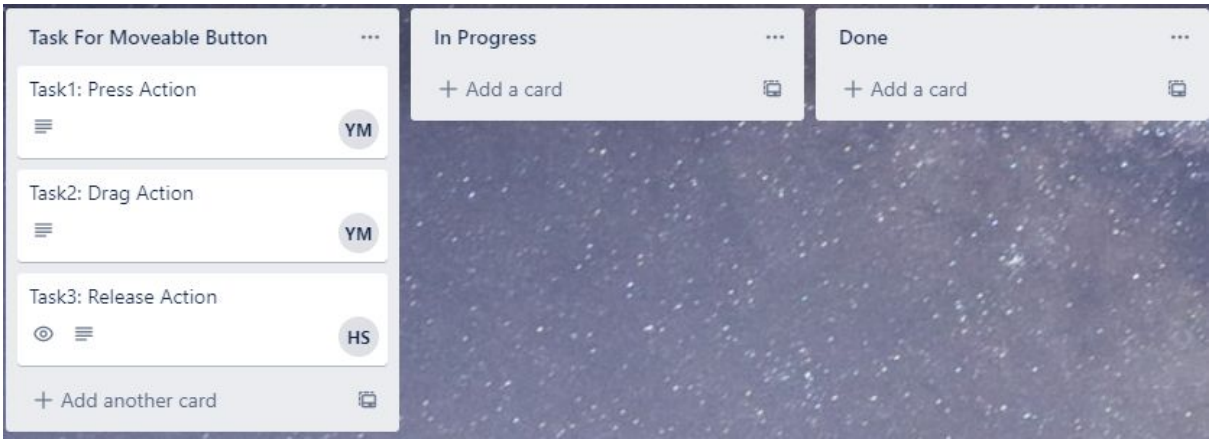
After analyzing the component, we decided to divide tasks based on the actions that users can make: press the button, drag the button and release the button.

The following is the tasks that designed:

Task 1: When the user presses the button, save the current location and listen to further action.

Task 2: When the user drags the button, calculate the distance of the moment and store the position when the drag finished.

Task 3: After dragging the button, make the button stick to the nearest side of the screen, either the left/right.



Task 1 and 2 are assigned to Yuewen, and Task 3 is assigned to Suxin.

4. Development and unit testing

Meeting Running:

Members: Suxin and Yuewen

Date: Mar 7th, 2020

Duration: 1 hour

Purpose: This meeting is for testing the feature we have added to the program, the outcome of this meeting should be a tested component.

Outcome: A tested and work component.

Test Steps: Since the feature is a frontend work, we hardly can write a test for this, so we tested it manually on Android simulators and the following the is the steps we conduct the test:

Step 1: Launch the app in IntelliJ or Android Studio and keep a simulator running.

Step 2: In the URL toggle bar, type google.ca and enter.

Step 3: Move the button up and down on the right side of the screen, the button should still stick to the right side of the screen but the height of the button should change.

Step 4: Click the button and the should leave google.ca to the homepage of Firefox-Focus (delete the browsing history as wanted).

Step 5: In the URL toggle bar, type youtube.com and go.

Step 6: Move the button to the left side of the screen.

Step 7: Move the button up and down on the left side of the screen, the button should still stick to the left side of the screen but the height of the button should change.

Step 8: Click the button and the should leave youtube.ca to the homepage of Firefox-Focus (delete the browsing history as wanted).

Test No.	Test Cases	Operation	Expected	Actual	Pass/Failed
1	Test if the button will stick to the right side of the screen	User presses the button the drag it only in the right half area of the screen	The button should stick to the right side of the screen automatically	The button sticks to the right side of the screen automatically	Pass
2	Test if the button can be dragged to the left side of the screen from the right side of the screen	User presses the button and drag it to the left half area of the screen and release	The button should stick to the left side of the screen automatically	The button sticks to the left side of the screen automatically	Pass
3	Test if the button will stick to the left side of the screen	User presses the button the drag it only in the left half area of the screen	The button should stick to the left side of the screen automatically	The button sticks to the left side of the screen automatically	Pass
4	Test if the button can be dragged to the right side of the screen from the left side of the screen	User presses the button and drag it to the right half area of the screen and release	The button should stick to the right side of the screen automatically	The button sticks to the right side of the screen automatically	Pass
5	Test if the delete browsing history function remains	User clicks the button when the button is stick to either side of the screen at any position	Current browsing history should be cleaned and showing the default homepage of Firefox-Focus	Current browsing history cleaned and the default homepage of Firefox-Focus is displayed	Pass

Meeting Outcome:

The “Delete” button is tested and it works as the requirement specified.

5. Integration and System Validation

Meeting Running:

Members: Anthony, Birathan, Pratana, Suxin and Yuewen

Date: Mar 11th, 2020

Meeting: 1 hour

Purpose: Integration and validation of our solution

Outcome: Code should be ready to be merged.

Meeting Outcome:

After this feature is added and merged with the Download Manager feature, the button works as specified and the functionality of cleaning browsing histories remains. Code is validated and ready to be merged.

Download Manager

1. Requirement Specification

Meeting Running:

Members: Anthony, Birathan and Pratana

Date: Mar 4th, 2020

Duration: 30 mins

Purpose: Understand the user requirements

Outcome: Detailed specifications of the user requirements.

Meeting Outcome:

When you download a file from a website, Focus should prompt a directory chooser with Downloads as a default directory to assign as a designated directory. This dialog should show up after the user clicks “ok” on the initial download dialog.

2. Component Analysis

Meeting Running:

Members: Anthony, Birathan and Pratana

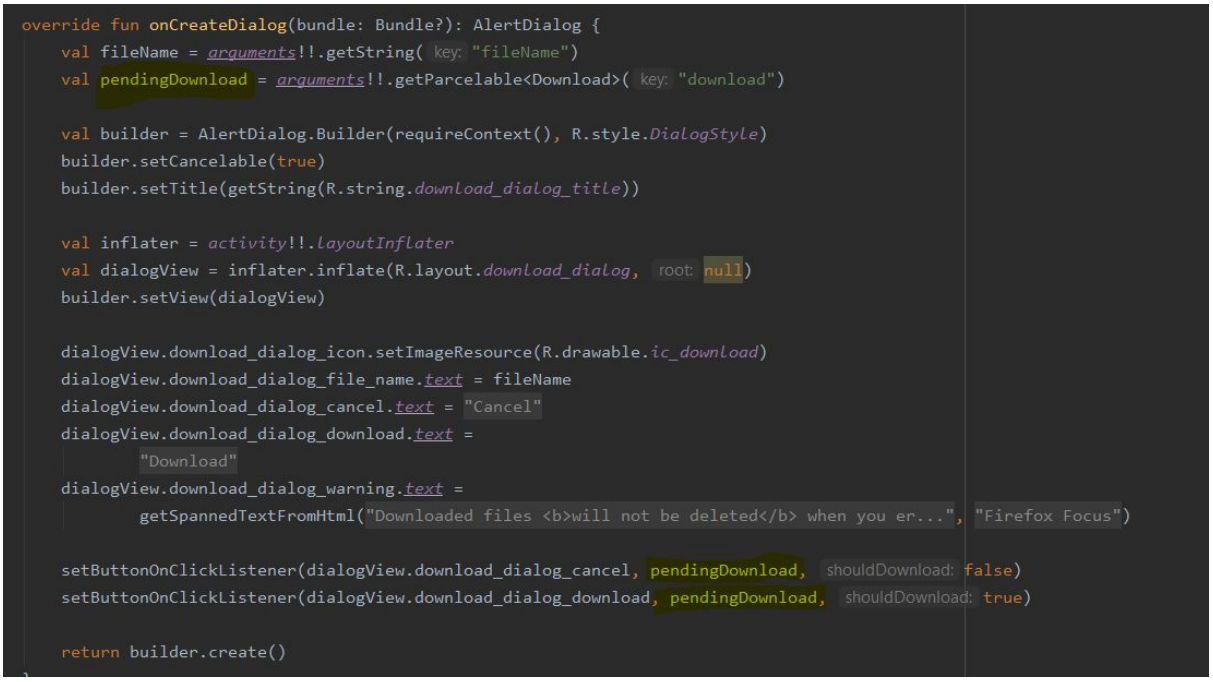
Date: Mar 4th, 2020

Duration: 1.5 hours

Purpose: Analyse the existing components in the program

Outcome: Details about the components and functions we need to modify.

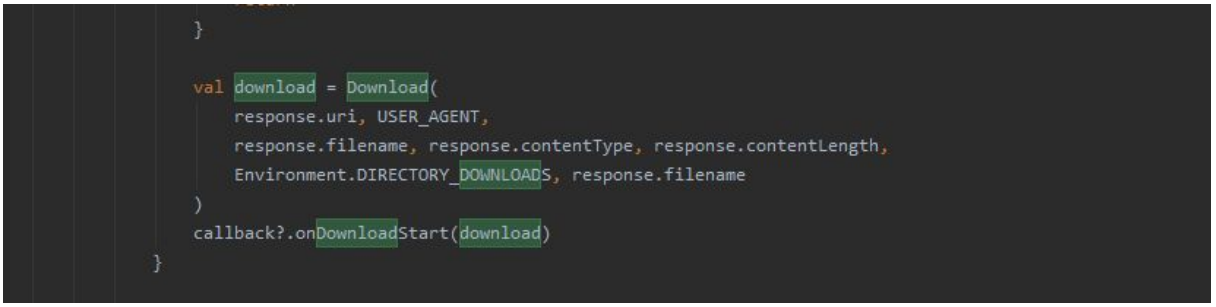
Meeting Outcome:



```
override fun onCreateDialog(bundle: Bundle?): AlertDialog {
    val fileName = arguments!!.getString( key: "fileName")
    val pendingDownload = arguments!!.getParcelable<Download>( key: "download")

    val builder = AlertDialog.Builder(requireContext(), R.style.DialogStyle)
    builder.setCancelable(true)
    builder.setTitle(getString(R.string.download_dialog_title))

    val inflater = activity!!.layoutInflater
    val dialogView = inflater.inflate(R.layout.download_dialog,  root: null)
    builder.setView(dialogView)

    dialogView.download_dialog_icon.setImageResource(R.drawable.ic_download)
    dialogView.download_dialog_file_name.text = fileName
    dialogView.download_dialog_cancel.text = "Cancel"
    dialogView.download_dialog_download.text =
            "Download"
    dialogView.download_dialog_warning.text =
            getSpannedTextFromHtml("Downloaded files <b>will not be deleted</b> when you er...", "Firefox Focus")

    setButtonOnClickListener(dialogView.download_dialog_cancel, pendingDownload,  shouldDownload: false)
    setButtonOnClickListener(dialogView.download_dialog_download, pendingDownload,  shouldDownload: true)

    return builder.create()
}
```

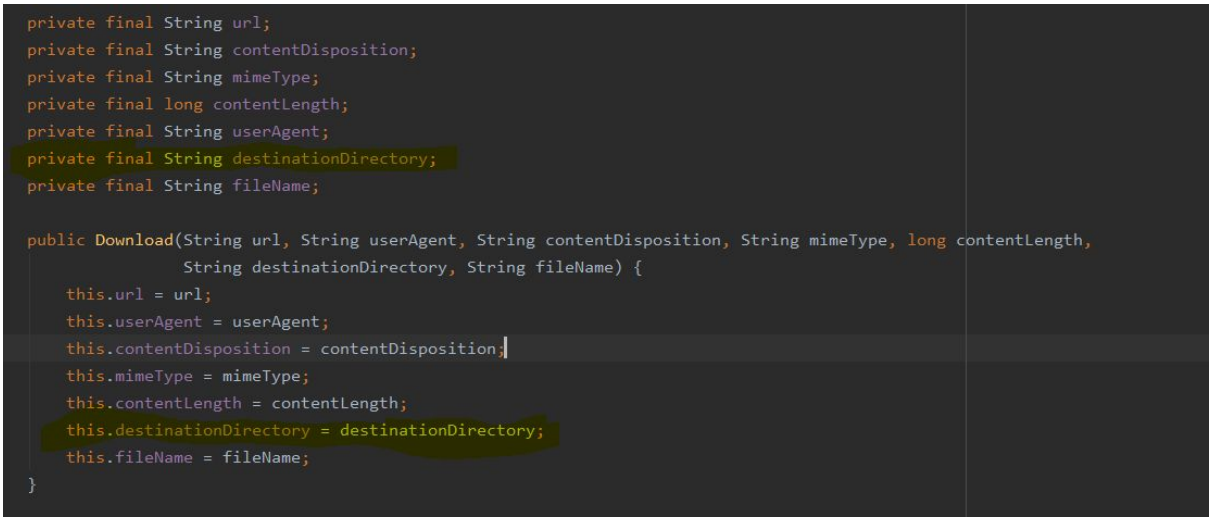
Figure 1. onCreateDialog

In **DownloadDialogFragment.kt,** the initial download dialog is called from **onCreateDialog** An event listener, **setButtonOnClickListener** is attached to both “Cancel” and “Download” buttons. **setButtonOnClickListener** when clicked will pass a **Download** object called **pendingDownload** which contains all necessary information about download; URL, filename, **destination directory**, etc. This Download object is created once inside **OnExternalResponse** in **GeckoWebViewPrvodier.kt**. **OnExternalResponse** reads a web response and saves it to a Download object.



```
        }

        val download = Download(
            response.uri, USER_AGENT,
            response.filename, response.contentType, response.contentLength,
            Environment.DIRECTORY_DOWNLOADS, response.filename
        )
        callback?.onDownloadStart(download)
    }
```

Figure 2 . onCreateDialog



```
private final String url;
private final String contentDisposition;
private final String mimeType;
private final long contentLength;
private final String userAgent;
private final String destinationDirectory;
private final String fileName;

public Download(String url, String userAgent, String contentDisposition, String mimeType, long contentLength,
                String destinationDirectory, String fileName) {
    this.url = url;
    this.userAgent = userAgent;
    this.contentDisposition = contentDisposition;
    this.mimeType = mimeType;
    this.contentLength = contentLength;
    this.destinationDirectory = destinationDirectory;
    this.fileName = fileName;
}
```

Figure 3. Download

We will need to modify **onCreateDialog** to prompt a user a directory and pass it along to the **OnExternalResponse** to be saved into a Download object.

After clicking on the “Download” button on the initial download dialogue, **sendDownloadDialogButtonClicked** is called. Inside, it calls **onFinishDownloadDialog** in **BrowserFragment.kt** with **pendingDownload** and **shouldDownload** boolean which we will not need for this feature.

Inside **BrowserFragment.kt , onFinishDownloadDialog** checks if **shouldDownload** before adding **pendingDownload** into the queue. **queueDownload** handles setting request variables and executing the download request.

3. System Design With Reuse

Meeting Running:

Members: Anthony, Birathan and Pratana

Date: Mar 6th, 2020

Duration: 30 mins

Purpose: This meeting is for designing a solution based on the component analysis. We will try to assign a task to everyone equally. It is to be noted, the deadline is on Mar 10th. Every member is occupied.

Outcome: A solution based on component analysis

Meeting Outcome:

Solution: Select directory from directory chooser, save the directory path and pass that to make the request.

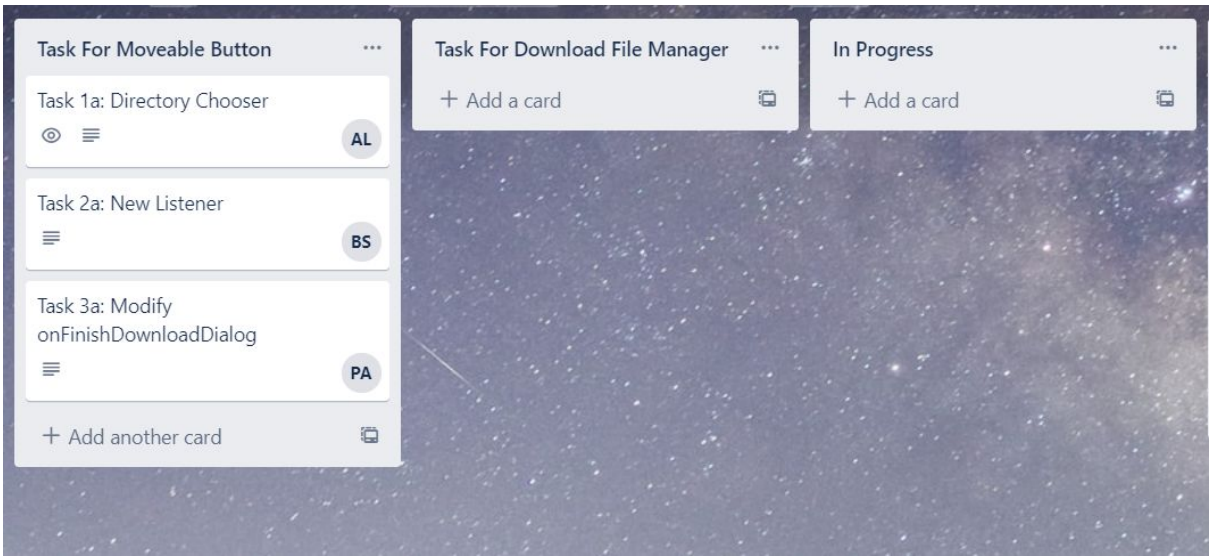
Since Android does not have directory/file chooser. We decided to import a third party directory chooser since it would have taken a lot of time writing one ourselves.

We will modify **DownloadDialogFragment.kt** and **BrowserFragment.kt.**

Task 1: **Directory chooser,** implement the directory chooser to save directory path and pass to **OnExternalResponse.**

Task 2: **New listener,** to be attached to the button for directory chooser

Task 3: **Modify onFinishDownloadDialog**



Task 1a is assigned to Anthony, Task 2a is assigned to Birathan and Task 3 is assigned to Pratana.

4. Development and unit testing

Meeting Running:

Members: Anthony, Birathan and Pratana

Date: Mar 6rd, 2020

Meeting: 2 hours

Due Date For Tasks: Mar 10th, 2020

Purpose: This meeting is for developing and planning unit tests and manual tests.

Outcome: A tested and work component.

Meeting Outcome:

The bug is actually more complicated than we have thought. First we tried using Android storage access framework to access the directory. It turned out it only selected a file but not the directory. Afterward, we try implementing directory chooser by overriding the function **onDownloadStart** in **BrowserFragment.kt.**

We create a new intent to open a document tree, and this will open a directory chooser.

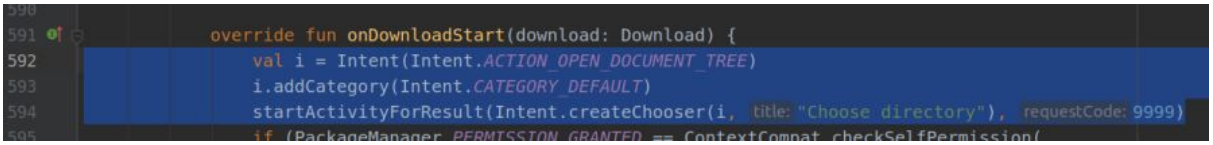


Figure 4. onDownloadStart

Figure 5. onActivityResult

Afterward we override **onActivityResult** in **BrowserFragment.kt.** , to get the destination directory path , we save the path to a static variable in Path class

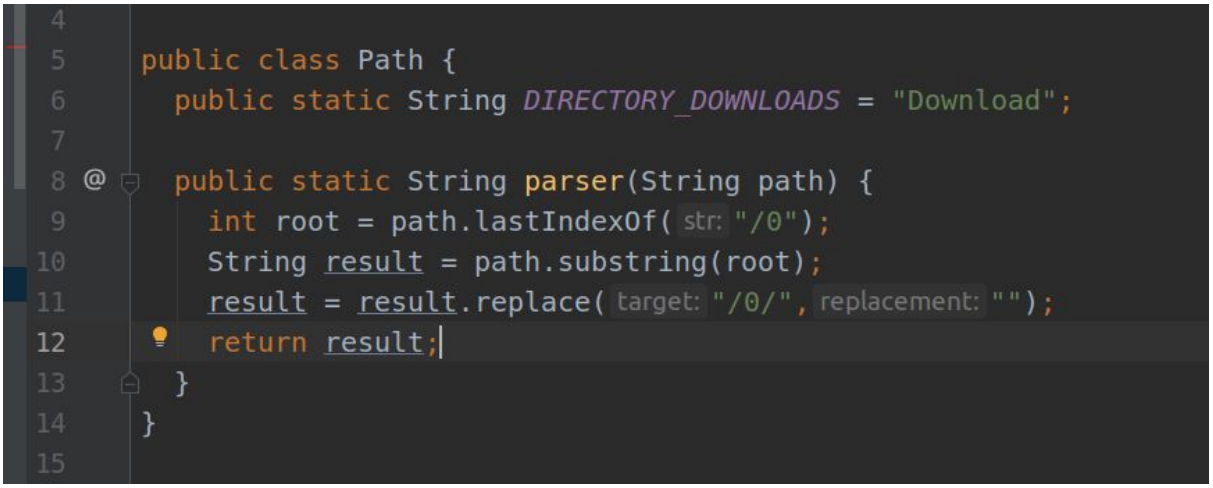
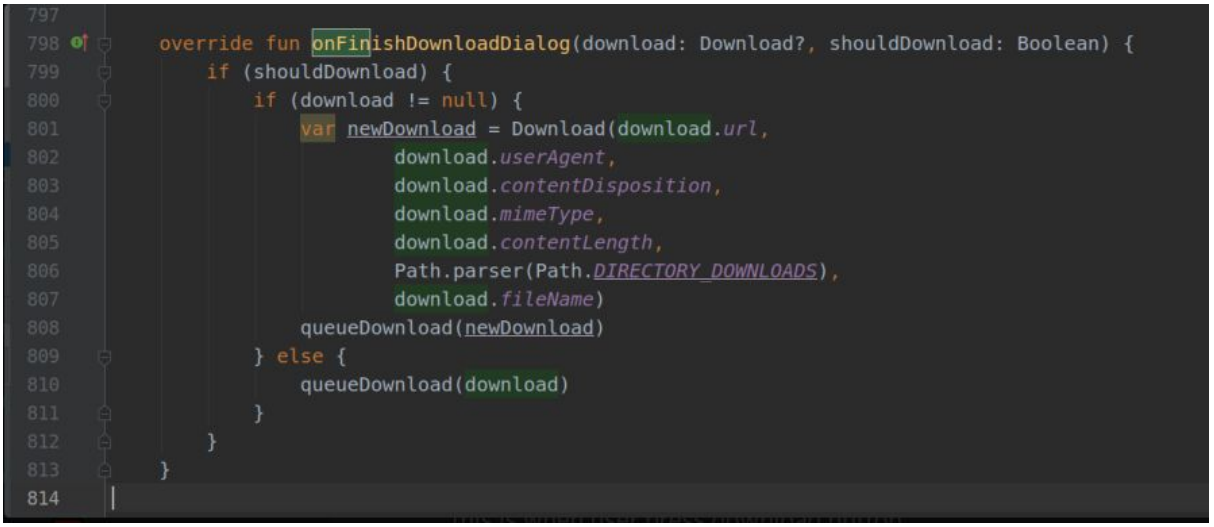


Figure 6. Path Class



```
797
798 override fun onFinishDownloadDialog(download: Download?, shouldDownload: Boolean) {
799     if (shouldDownload) {
800         if (download != null) {
801             var newDownload = Download(download.url,
802                     download.userAgent,
803                     download.contentDisposition,
804                     download.mimeType,
805                     download.contentLength,
806                     Path.parser(Path.DIRECTORY_DOWNLOADS),
807                     download.fileName)
808             queueDownload(newDownload)
809         } else {
810             queueDownload(download)
811         }
812     }
813 }
814
```

Figure 7. onFinishedDownloadDialog

Afterward, we call the **Path.Parser** which parsers our static destination directory path into the correct format inside the override **onFinishDownloadDialog in BrowserFragment.kt.** Then, we create a new **Download** object with our parsed destination directory saved. The parser only works for simple cases there are many dependencies that need to be considered. E.g. the parser only can parse the location from Downloads/ if the user chooses location from sd card or from the root, the parser cannot parse. The path is also machine dependent when choosing from sd card. There are also problems with file downloading. When a user download a file, it is saved into both location; new location and download. They are the same copy and cannot be deleted. If one of them is deleted , the other is deleted also. This is controlled by Android code, we are unable to tell what has happened.

As for testing , we can only test the parser. The majority of the implementation depends on the frontend. We cannot select the directory from the backend. Therefore, we could do the unittest for the parser for the path. Everything else will be done through manual testing

Following the is the steps we conduct the manual tests:

Step 1: Launch the app in IntelliJ or Android Studio and keep a simulator running.

Step 2: In the URL toggle bar, type in any website with file to download

Step 3: Download the file

Step 4: Directory Chooser should be prompted

Step 5: Change the directory and press select

Step 6: Press cancel

Step 7: Navigate to the selected directory

Step 8: Verify the file is not saved

Step 8: Repeat 1- 5

Step 9: Press download

Step 10: Verify the file is saved

Step 11: Download Same file

Step 12: Follow step 4 - 5 , with same directory selected

Step 13: Press download

Step 14: Verify that there are 2 files with the same name in the selected directory

Step 15: Download the new file

Step 16: Directory Chooser should be prompted

Step 17: Press select

Step 18: Verify the new file is saved in Downloads

Step 15: Download another file

Step 16: Directory Chooser should be prompted

Step 17: Change directory , then change again

Step 18: Verify the file is saved in the last selected directory

Test No.	Test Cases	Operation	Expected	Actual	Pass/Fail
1	Test if the file is download	User click download and select the directory	File should be saved into that directory	File is saved into that directory	Pass
2	Test if clicking cancel in the directory, cancel the file download	User click download but cancel before selecting the directory	File should not be download	File is not download	Pass
3	Test if file with the same name are saved	User download the same file and save into the same directory	The new file with the same name should be saved	The new file is saved	Pass
4	Test default destination folder	User click download	The default folder should be “Downloads”	The default folder is “Downloads”	Pass
5	Test changing folder destination	User click download , change the directory, change the directory again	File should be saved into the last directory saved	File is saved into the last directory	Pass

5. Integration and System Validation

Meeting Running:

Members: Anthony, Birathan, Pratana, Suxin and Yuewen

Date: Mar 11th, 2020

Meeting: 1 hours

Purpose: Integration and validation of our solution

Outcome: Code is ready to be merged.

Meeting Outcome:

After this feature is added and merged with the Download Manager feature, at minimum, users are able to select a directory and download the file, and the file is then saved into the correct directory. We validated the solution. The code is ready to be merged.