Why are you interested in working with Oppia, and on your chosen project?

The word Oppia means to learn. An organization whose mission is to help anyone learn anything they want in an effective and enjoyable way. I would love to be a part of this beautiful mission which helps in spreading learning. A majority of the world does not have access to proper education facilities, and here, Oppia plays its role of helping those unprivileged children who lack these facilities. I would like to add my contribution to this global mission as much as I could. Oppia provides me a platform to be a part of this mission and I am therefore interested in working with Oppia and will do my best to make this mission successful.

Why the project: **Static Analysis Checks + Improvements** ?

A messy code can be written easily just to make the things work, while it may take a comparatively longer time to write a clean code. But, when we think of the long run, a clean code is far better than a messy code. I always think of things from the perspective of the long run, and hence this project interests me the most. I would love to make the process of code reviewing faster, so that we can save the time spent in fixing the lint and other similar errors during code reviews, and instead utilize it on reviewing the actual feature implementation or bug fix. The current codebase has static analysis checks to ensure this, but still it lacks numerous other very important checks, which are a must to be added. This project will help me in gaining more expertise on maintaining clean code and developing custom static analysis checks.

Prior experience

I am Sparsh Agrawal, an undergraduate in Indian Institute of Technology Roorkee pursuing Electronics and Communication Engineering. My love for development began in the 10th standard when I learned my first programming language. From there, I was introduced to the world of development. In my freshman year, I explored android development, and have been doing that from the past year. Also, I am an active member of the Information Management Group (IMG), IIT Roorkee which is a group of passionate developers and designers who constantly strive to lead in the innovation, development, and maintenance of advanced information technologies including computer systems, software and database systems in the institute. At IMG, I learned to work along in a product cycle with a team of approximately 50 members. In my pursuit of acquiring development skills, I started contributing to the open source community from the past one year, so that I can give back to the organizations whose products have been valuable assets to me. I have been involved in the both: Web and Mobile App development. I have been contributing to Oppia from the past 2-3 months, and therefore gained familiarity with the codebase via working on the issues and submitting the corresponding PRs.

My Github profile: Sparsh1212

My open source contributions in the past one year:

1. Oppia:
    a. PRs:
        i. #2856: Update ongoing topic logic
        ii. #2656: Fix talkback bug on activity transition to topic screen
        iii. #3030: Shift RecyclerViewMatcher to Central testing utility (Almost completed)
        iv. #2750: Add label for FAQList screen
        v. #2699: Remove event_logger_java_proto_lite export
        vi. #2645: Admin add pin
    b. Issues Filed:
        i. #2860: Variable name is displayed instead of actual incorrect answer

A list of all my contributions to Oppia can be found: here

2. Mifos Initiative:
    a. PRs:
        i. #1059: Add instructions to get the latest apk from artifacts (Github Actions)
        ii. #195: Setup ci using GitHub actions
        iii. #1579: Remove unnecessary ndk specifications from build.gradle files
        iv. #1603: Update fabric crashlytics to firebase crashlytics
        v. #1596: Redesign splash screen as per new design
        vi. #201: Convert null checks in Network.kt to kotlin style
    b. Issues Filed:
        i. #199: Add network change receiver

3. CircuitVerse:
    a. Issues Filed:
        i. #17: Disable add button on empty input field of Add Group Members
        ii. #18: Search button and hamburger menu not working in the webview
        iii. #19: Spelling error in image picker bottom sheet

4. Catrobat:
    a. PRs:
        i. #4010: Localize variable NO_VARIABLE_SELECTED

5. Mozilla:
    a. PRs:
        i. #9869: Trim search query
        ii. #9874: Hide search-btn outline
    b. Issues Filed:
        i. #9866: Search query is not trimmed
        ii. #9865: Search button alignment improper when clicked

Contact info and timezone(s)

Email - sparshagrawal1212@gmail.com
Phone - +91 7987255297
Timezone - Indian Standard Time (IST) (+5:30 GMT)
Preferred Method of communication - Email or Phone

Time commitment

- I would be working on the GSoC project throughout the 10-week period, from 7th June to 16th August.
- I would commit at least 5 hours per day during the coding period, and may increase it if necessary.

Other summer obligations

I have no other commitments during the summer. So, I will be able to devote the full time of my summer holidays to this project.

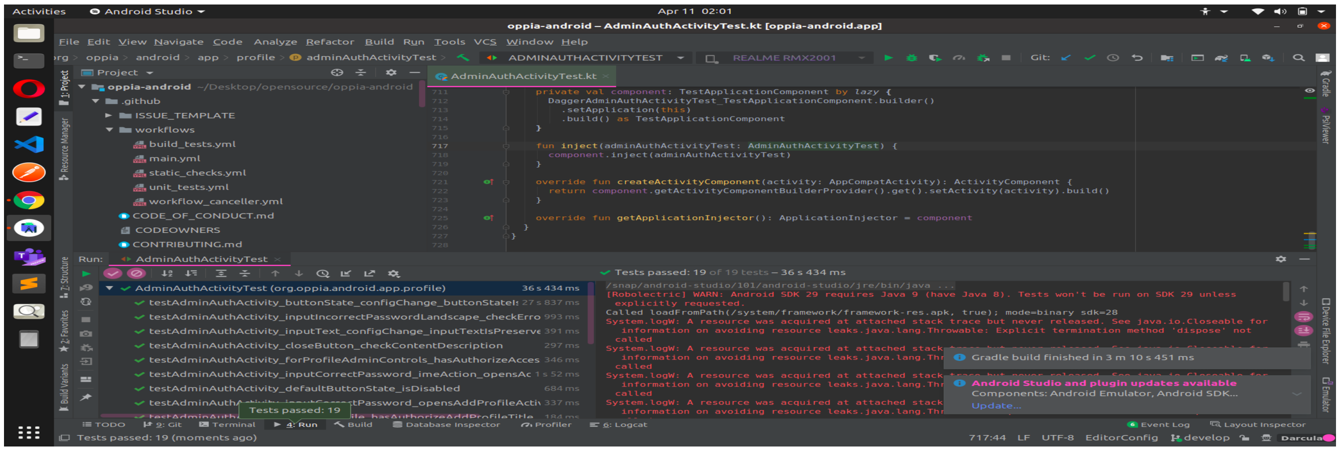
Communication channels

Reachable anytime through **email**, **gitter**, **slack**, **contact number**, or a planned video-session.

Essential Prerequisites

Answer the following questions (for Oppia Android GSoC students):

- *I am able to run a single Robolectric test target on my machine via Android Studio. (Show a screenshot of a successful test.)*



oppia-android – AdminAuthActivityTest.kt [oppia-android.app]

```
  private val component: TestApplicationComponent by lazy {
    DaggerAdminAuthActivityTest_TestApplicationComponent.builder()
      .setApplication(this)
      .build() as TestApplicationComponent
  }

  fun inject(adminAuthActivityTest: AdminAuthActivityTest) {
    component.inject(adminAuthActivityTest)
  }

  override fun createActivityComponent(activity: AppCompatActivity): ActivityComponent {
    return component.getActivityComponentBuilderProvider().get().setActivity(activity).build()
  }

  override fun getApplicationInjector(): ApplicationInjector = component
 }
}
```

```
AdminAuthActivityTest (org.oppia.android.app.profile)                      36 s 434 ms
  testAdminAuthActivity_buttonState_configChange_buttonStateI:              27 s 837 ms
  testAdminAuthActivity_inputIncorrectPasswordLandscape_checkErro           993 ms
  testAdminAuthActivity_inputText_configChange_inputTextIsPreserve          391 ms
  testAdminAuthActivity_closeButton_checkContentDescription                 297 ms
  testAdminAuthActivity_forProfileAdminControls_hasAuthorizeAcces           346 ms
  testAdminAuthActivity_inputCorrectPassword_imeAction_opensAc            1 s 52 ms
  testAdminAuthActivity_defaultButtonState_isDisabled                       684 ms
  testAdminAuthActivity_inputCorrectPassword_opensAddProfileActi            337 ms
```

```
Tests passed: 19 of 19 tests – 36 s 434 ms
/snap/android-studio/101/android-studio/jre/bin/java ...
[Robolectric] WARN: Android SDK 29 requires Java 9 (have Java 8). Tests won't be run on SDK 29 unless
  explicitly requested.
Called loadFromPath(/system/framework/framework-res.apk, true); mode=binary sdk=28
System.logW: A resource was acquired at attached stack trace but never released. See java.io.Closeable for
  information on avoiding resource leaks.java.lang.Throwable: Explicit termination method 'dispose' not
  called
System.logW: A resource was acquired at attached stack
  information on avoiding resource leaks.java.lang.Thr
  called
System.logW: A resource was acquired at attached stack
  information on avoiding resource leaks.java.lang.Thr
  called
System.logW: A resource was acquired at attached stack
  information on avoiding resource leaks.java.lang.Thr
```

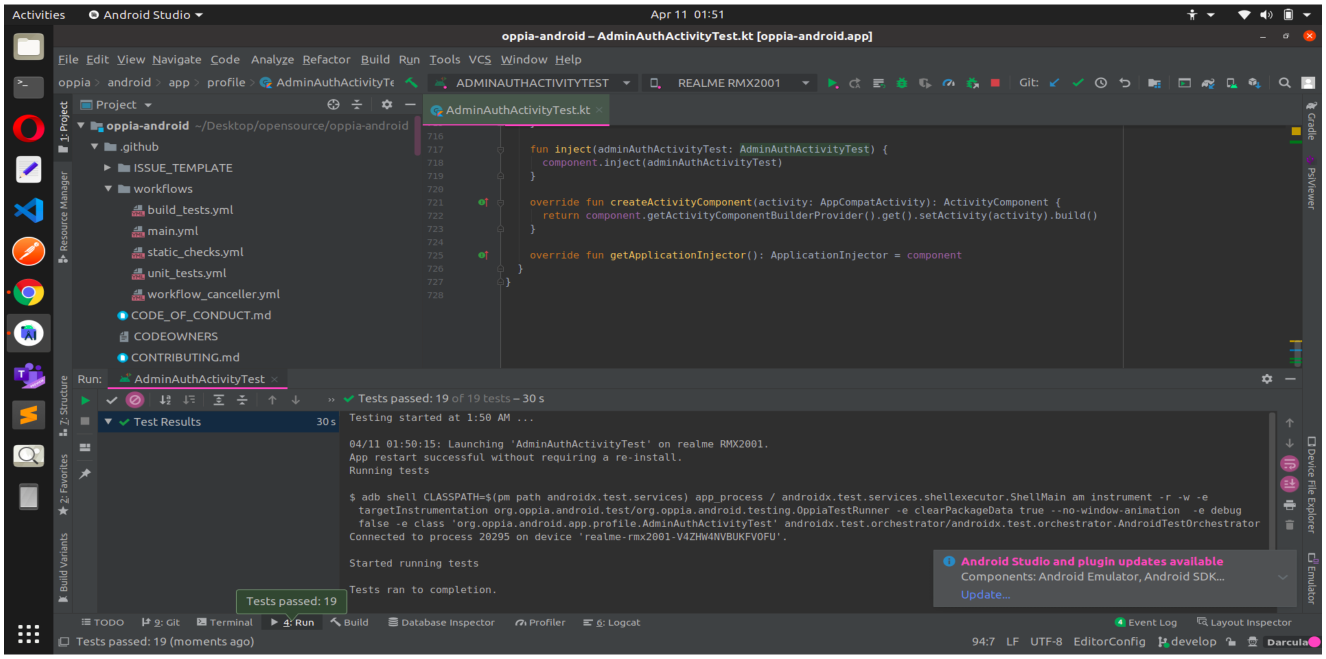
Gradle build finished in 3 m 10 s 451 ms

Android Studio and plugin updates available
Components: Android Emulator, Android SDK...
Update...

Tests passed: 19 (moments ago)

Apr 11 02:01
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ADMINAUTHACTIVITYTEST
REALME RMX2001
AdminAuthActivityTest.kt
Run: AdminAuthActivityTest
TODO 9: Git Terminal 4: Run Build Database Inspector Profiler 6: Logcat
Event Log Layout Inspector
717:44 LF UTF-8 EditorConfig develop Darcula

- *I am able to run a single Espresso emulator test target on my machine via Android Studio. (Show a screenshot of a successful test.)*


Activities
Android Studio
Apr 11 01:51
oppia-android – AdminAuthActivityTest.kt [oppia-android.app]
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
oppia > android > app > profile > AdminAuthActivityTe
ADMINAUTHACTIVITYTEST
REALME RMX2001
Git:
Project
oppia-android ~/Desktop/opensource/oppia-android
.github
ISSUE_TEMPLATE
workflows
build_tests.yml
main.yml
static_checks.yml
unit_tests.yml
workflow_canceller.yml
CODE_OF_CONDUCT.md
CODEOWNERS
CONTRIBUTING.md
AdminAuthActivityTest.kt
Run: AdminAuthActivityTest
Test Results 30 s
TODO 9: Git Terminal 4: Run Build Database Inspector Profiler 6: Logcat
Event Log Layout Inspector
94:7 LF UTF-8 EditorConfig develop Darcula

```kotlin
  fun inject(adminAuthActivityTest: AdminAuthActivityTest) {
    component.inject(adminAuthActivityTest)
  }

  override fun createActivityComponent(activity: AppCompatActivity): ActivityComponent {
    return component.getActivityComponentBuilderProvider().get().setActivity(activity).build()
  }

  override fun getApplicationInjector(): ApplicationInjector = component
 }
}
```

Tests passed: 19 of 19 tests – 30 s

```
Testing started at 1:50 AM ...

04/11 01:50:15: Launching 'AdminAuthActivityTest' on realme RMX2001.
App restart successful without requiring a re-install.
Running tests

$ adb shell CLASSPATH=$(pm path androidx.test.services) app_process / androidx.test.services.shellexecutor.ShellMain am instrument -r -w -e
 targetInstrumentation org.oppia.android.test/org.oppia.android.testing.OppiaTestRunner -e clearPackageData true --no-window-animation  -e debug
 false -e class 'org.oppia.android.app.profile.AdminAuthActivityTest' androidx.test.orchestrator/androidx.test.orchestrator.AndroidTestOrchestrator
Connected to process 20295 on device 'realme-rmx2001-V4ZHW4NVBUKFVOFU'.

Started running tests

Tests ran to completion.
```

Tests passed: 19

Android Studio and plugin updates available
Components: Android Emulator, Android SDK...
Update...

Tests passed: 19 (moments ago)

Project Details

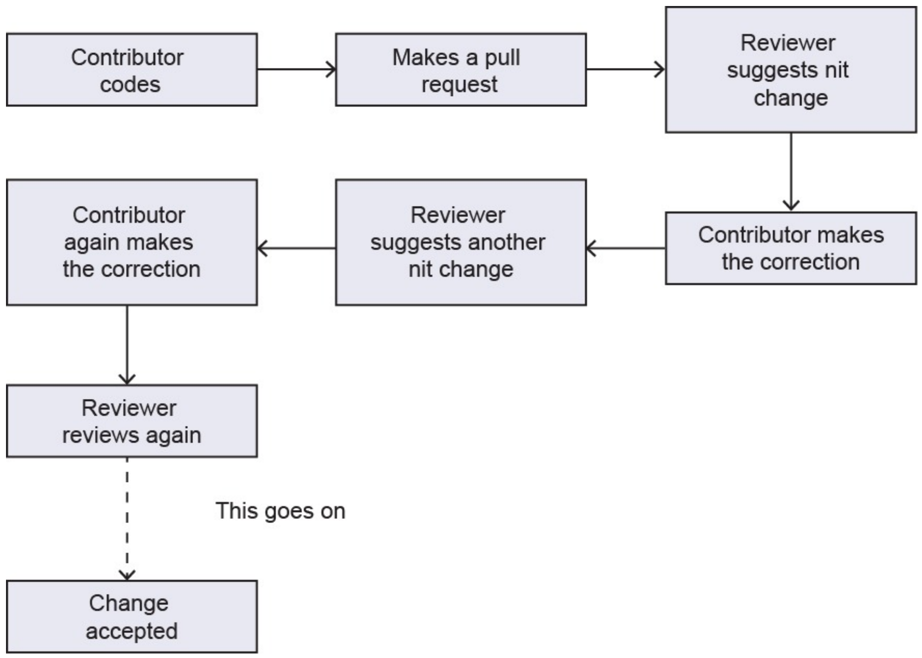
Product Design

What is static analysis ?

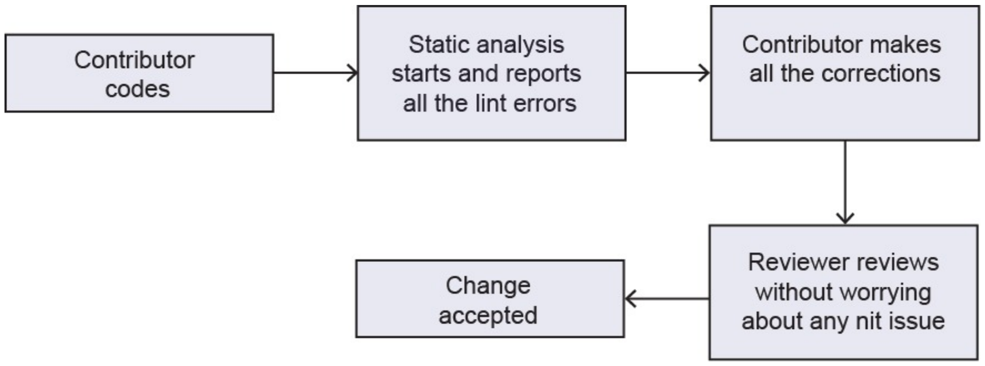
Static Code Analysis is a method of debugging by examining source code before a program is run. It's done by analyzing a set of code against a set (or multiple sets) of coding rules.

Why static code analysis is essential ?

It takes time for developers to do manual code reviews. Automated tools are much faster. Static code checking addresses problems early on. And it pinpoints exactly where the error is in the code. So, the developer will be able to fix those errors faster.


Contributor codes
Makes a pull request
Reviewer suggests nit change
Contributor makes the correction
Reviewer suggests another nit change
Contributor again makes the correction
Reviewer reviews again
This goes on
Change accepted

Development flow without static analysis


Contributor codes
Static analysis starts and reports all the lint errors
Contributor makes all the corrections
Reviewer reviews without worrying about any nit issue
Change accepted

Development flow with static analysis

We can see how a static analysis tool (in this case a linter) simplified the flow and saved the time of both the developer and the reviewer which was wasted earlier in correcting the lint errors. Hence, it is very essential to have Static Analysis checks in the development workflow of a project so as to save the precious time of the developers and the reviewers.

Also, the more we can improve these static checks on the codebase, the more we will be able to keep the code intact with the standard guidelines and conventions and will ensure that we maintain the best practices as the codebase gets bigger.

The users of this project are the **developers** of the Oppia team.

Current status of the static code analysis of the **oppia-android** codebase:

There are 4 lint tools which are currently used for the static code analysis of the codebase.

1. Checkstyle
   This ensures all the java code present in the codebase adheres to a coding standard.

2. Ktlint
   A static analysis tool which analyzes the kotlin part of the codebase

3. Buf
   This tool provides consistency and maintainability across a Protobuf schema.

4. Buildifier
   This takes care of the static analysis of Bazel BUILD and .bzl files.

With all these tools present in the project already, there are still numerous essential static analysis checks which the project currently lacks. The default behaviour of these linting tools is not enough to meet all the best current practices. This project aims to add all those essential static analysis checks which the project currently lacks so that the development team reaches optimal efficiency, the codebase remains clean, fully documented and without any violation of standard practice.

This project aims to add these additional checks with the combination of these tools/approach so as to utilize the best available option for achieving the corresponding use case/ implementing the particular check

Note: A detailed description of how these checks are going to be implemented in the project is provided in the **Technical Design** section of the proposal. This is just an overview of the approach.

1. Integrating the project with Github Super Linter: This project will make use of Github Super Linter as the linter to ensure all the XML files follow the team's style guide.

2. Use of **Scripts (Preferably Kotlin or Bash)**: To implement the custom checks for this particular use case, this project will involve writing custom Kotlin/bash scripts to implement the static checks mentioned in the Project idea.

Note: In the Technical Design Section of the proposal, to demonstrate the implementation of checks, python scripts have been used. However as per the actual Project Plan, the final scripts will be written in either Kotlin or bash. Python is just used in the proposal for demonstration purposes only.

3. Use of Github Actions: Since, the static analysis checks have to be executed during the CI workflow, this project will involve extensive use of Github Actions to run these checks during the CI.

Github Actions: Github Actions helps us to automate the tasks within the software development cycle. They are *event-driven.* Event-driven means that we can run a series of commands after a specified event has occurred. For ex: When someone closes an issue, we can automatically run a command that executes a custom script to check if the corresponding TODO of the issue has been addressed or not.

Components of Github Actions:

- **Workflow:** The workflow is an automated procedure that we add to the repository. It consists of one or more jobs and can be scheduled or triggered by an event. We can create a workflow to build, test, package, release, or deploy a project on GitHub.

- **Events:** It is a specific activity that triggers a workflow. For ex: activity can originate from GitHub when someone creates an issue, closes it, pushes a commit to the repository etc. Here is the link for ref. to all the Events that trigger workflows.

- **Jobs:** It is a set of steps that execute on the same runner. By default, a workflow with multiple jobs will run those jobs in parallel. We can also configure a workflow to run jobs sequentially by using `needs:`

```
jobs:
  one:
    - do stuff -
  two:
    needs: one
     - do more stuff -
```

- **Steps:** A step is an individual task that can run commands in a job. It can be either an action or a shell command.

- **Actions:** These are standalone commands that are combined into steps to create a job. Actions are the smallest portable building block of a workflow. We can create our own actions, or use actions created by the GitHub community.

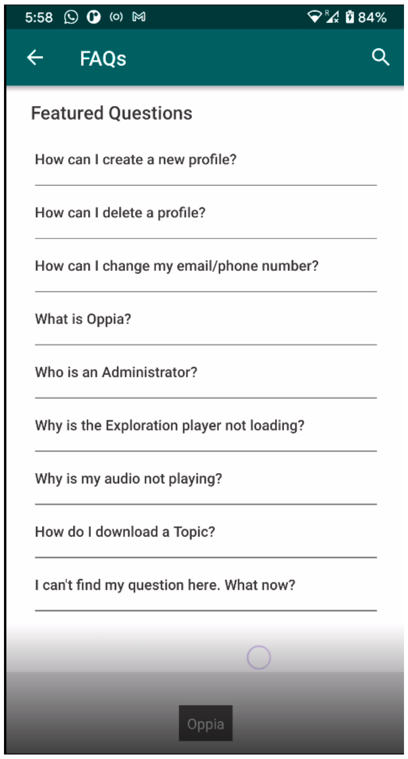
- **Runners:** It is a server that has the GitHub Actions runner application installed. A runner listens for available jobs, runs one job at a time, and reports the progress, logs, and results back to GitHub.

GitHub Actions uses YAML syntax to define the events, jobs, and steps. The YAML files are stored in a directory called `.github/workflows`.

Analysis of the essential checks that the project current lacks, the problems that the developers (users also in some worst cases) have to face because of the absence of these checks and how this project will improve the current scenario:

1. **Verify activities have accessibility labels**

For the A11y users, activity labels are very important for the transition purposes. Suppose, on the button click, the app transits to a new activity. A regular user will face no problem in understanding that a new screen/activity has opened. But A11y users need a form of feedback so that they could somehow understand/be notified that the transition to a new Activity has been successfully made. When this label is missing in the Activities, the screen readers don't speak a meaningful representation, and therefore the user (A11y in this case) will face problems in understanding which Activity has opened.
Currently in the codebase, there are no checks to ensure that the activities include the accessibility labels. As a result of absence of this check, the team has faced several issues in the past which are as follows:



5:58 84%

FAQs

Featured Questions

How can I create a new profile?

How can I delete a profile?

How can I change my email/phone number?

What is Oppia?

Who is an Administrator?

Why is the Exploration player not loading?

Why is my audio not playing?

How do I download a Topic?

I can't find my question here. What now?

Oppia

Because of the missing label on the activity, the Talback reads this screen as *Oppia* at the start, whereas it should be *Frequently Asked Questions (FAQs).*

Issue references: [#2633](#) [#2823](#) [#2822](#) [#2816](#) and many more similar issues.

This project will make sure that instances like these do not happen and every activity (which is meant for the users of the app) has these accessibility labels, and whenever any developer misses these labels, the script written for checking the accessibility label in the Activity will report the issue and will notify that which particular Activity lacks the accessibility label. Also, the CI workflow will fail because of the failure of this check. This will ensure that the app is accessible for users of all kinds and the developers do not miss these labels in any scenario.

Note: This check of testing whether the activity has accessibility labels will be tested only on those activities which are meant to be for the users of the app. The activities present in the codebase which are for testing purposes and not for the users of the app will not be tested for this check. For ex: ConceptCardFragmentTestActivity, AudioFragmentTestActivity, BindableAdapterTestActivity and other similar activities will be skipped for this check.

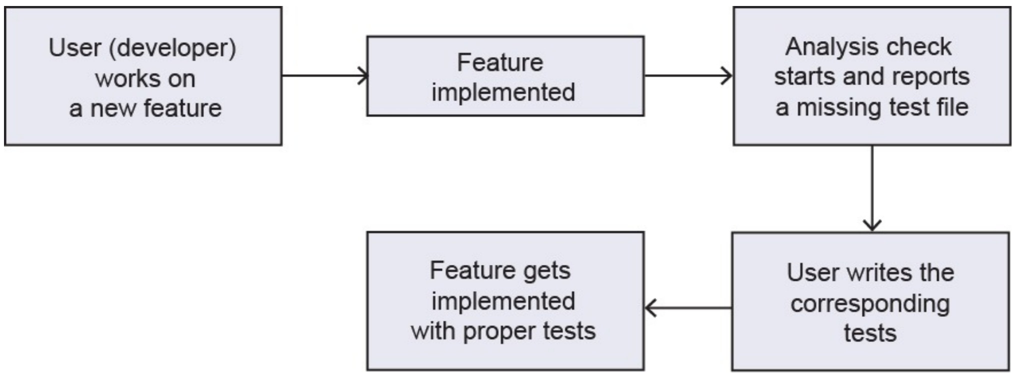
2. **Ensuring All XML files follow the XML style guide**

Oppia-Android Style Guide: here
The current codebase does not have any lint checks to ensure that the XML files follow the style guide or not. This project will integrate the current code base with Github Super Linter support, so that on every push, pull_request and workflow_dispatch the lint checks for the XML files will be triggered and if anywhere the linter founds the issue, it will report it and the workflow will fail. This way we will ensure that all the XML files are well formatted (just like ktlint is used currently for the kotlin files of the codebase).
Note: I have left a note for the maintainers regarding the lint tool for the XML files in the Technical Design section. Please refer to it.

3. **Verify all production files have a corresponding test file**

Testing is a very important phase in the development cycle of any project. We must test all the features, UI and every other thing present in the app thoroughly so that the app is error-prone and the users have a good experience using the app. Many times it happens that, while developing a particular feature for the app, the developer forgets to add a corresponding test and then pushes the code without writing a test for the feature implemented. This leads to a lot of time wastage, because the reviewer has to remind the developer to write the test during the review. This leads to longer development cycles, and in some cases where the reviewer also forgets to check that a test is present or not, it is possible that a feature gets added in the codebase without a proper test written for it. Hence, it is very essential that, during the workflow, we have to check that every production file has a test file associated with it, so that if the developer somehow forgets to add a test, the workflow fails and notifies to add a test file for the new production file added.

Note: This check will run only on the appropriate production files, for ex: **XML**, **BAZEL**, **gradle**, etc will be skipped for this check. The strategy of excluding them is explained in detail in the Technical Design section.


User (developer) works on a new feature
Feature implemented
Analysis check starts and reports a missing test file
User writes the corresponding tests
Feature gets implemented with proper tests

4. **Ensure KDocs are present for every non-private class, method, and field (even trivial ones)**

Documenting code is a very essential step, and it should not be missed at any cost. It helps the new contributors and even the already existing ones, to understand what the particular piece of code performs. It makes the process of understanding the codebase much easier. Hence, it is essential to check that any new class, method and field contains an associated KDoc with it.

5. **Ensuring future checks on file names and file contents**

In the current codebase, there are no checks to ensure that the proper file naming conventions are followed. Also, there are several instances where the filenames and identifiers are not in accordance with the standard convention. Hence, it is very important to make sure via checks that naming follows the convention.

For example:

File naming convention: https://developer.android.com/kotlin/style-guide#camel_case

The following files are violating this convention:

- https://github.com/oppia/oppia-android/blob/develop/app/src/sharedTest/java/org/oppia/android/app/faq/FAQListFragmentTest.kt
- https://github.com/oppia/oppia-android/blob/develop/app/src/sharedTest/java/org/oppia/android/app/faq/FAQSingleActivityTest.kt
- https://github.com/oppia/oppia-android/blob/develop/app/src/main/java/org/oppia/android/app/help/faq/RouteToFAQSingleListener.kt

  And many more such files ...

Reference discussion: here

Also, the project currently lacks checks on the file contents, to ensure that we aren't using any attributes like `marginLeft` in the project.

Here is a list of all those attributes which must be checked:

- marginLeft
- marginRight
- `paddingLeft`
- `paddingRight`
- `drawableLeft`
- `drawableRight`
- `layout_alignLeft`
- `layout_alignRight`

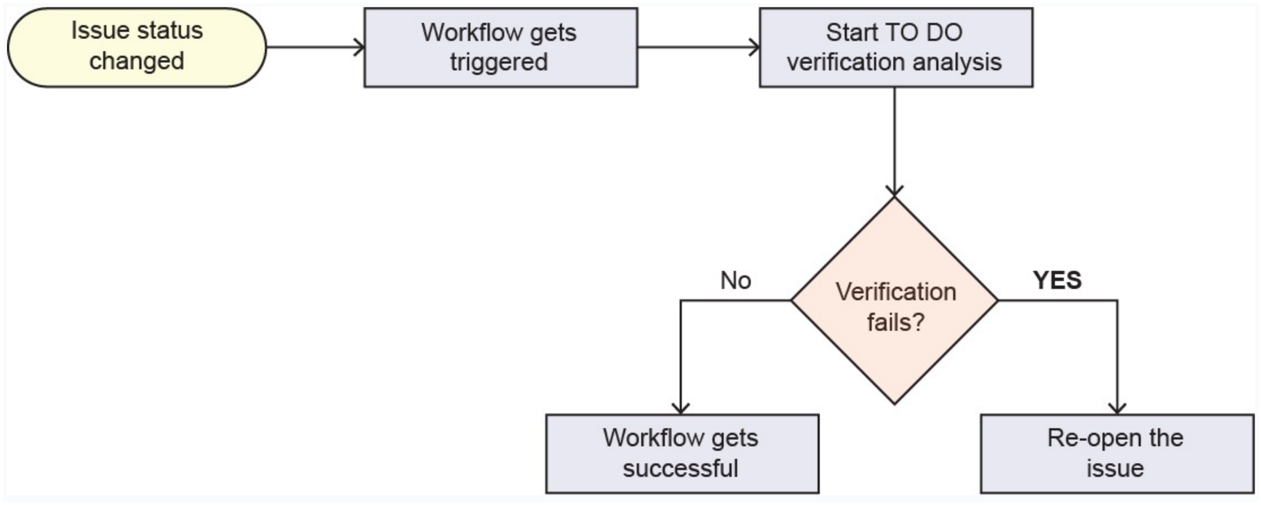
- `layout_alignParentLeft`
- `layout_alignParentRight`
- `layout_toLeftOf`
- `layout_toRightOf`
- `gravity=”left”`
- `gravity=”right”`
- `textAlignment=”start”`

The reason for checking these particular attributes is to support different languages and cultures. This project will ensure sure the app remains RTL supported as much as possible.

This project will make sure that all the files follow the file naming convention, and will add proper checks to check the usage of any of the above-listed attributes in the XML files. Violation of any check will result in the CI workflow with appropriate logging of the lint failures.

6. Introduce TODO Verification

In the current codebase, for every TODO, that is listed, a corresponding open issue must be present. And if the issue status is changed, (say the issue is closed), then this project will trigger a workflow check that the corresponding TODO is addressed. If not, then the issue must be reopened.



This workflow will ensure that all the TODOs in the project are maintained properly, and no issue gets closed without its corresponding TODO getting addressed. If any issue is closed without all of its TODOs being addressed then ***the issue will be reopened again with a new comment added*** (providing details of the files/LOCs of all of its unaddressed TODOs).

Approaching the project with Custom Android Lint Rules:
The custom checks can also be implemented with the use of Android Lint. But because of the following reasons the approach of using Android Lint as the tool for this project was rejected:

- This project requires some custom static analysis checks, and the default rules of Android Lint do not include those checks which fulfills our use case. Hence, if we want to implement those custom checks, then we have to write our custom Android Lint rules. The problem with custom Android Lint rules is that: It's a Gradle project. Ref could be found here (official page of android tools). And since, oppia-android will be fully migrated to the BAZEL build system in some time, so using some gradle stuff for the static analysis does not seem appropriate. Because that way, we won't be gradle free in complete sense.
- Also, officially the custom Android Lint rule is not supported for the BAZEL build system. I asked the question of building the custom lint rules without using gradle build system in the official google group of Android Lint, and it was answered by the current maintainer of the Android Lint project: **Tor Norbye**. To that question, the official answer was: "**Sorry, no -- that will depend on the build system you're using and we only support Gradle**"
  Reference to this conversation: here.
- In addition to this, the same question was asked in the form of an issue on the BAZEL build system Github repository to add the support of building Lint rules in BAZEL. To that issue, the reply from the maintainers side was: "**currently there are no plans to support Android Lint for BAZEL"**. Reference for the conversation: here. Hence, officially there's no support to use the Android Lint tools for the BAZEL build system. We have to use Gradle for building it. There might be some hack to use it with the BAZEL build system (although there's no info about it on the internet) but using some hackish way to implement the checks for this project which isn't supported officially doesn't seem appropriate in the long run.
- Another problem with the custom Android Lint rules is that it has no documentation of its APIs currently. Its documentation is released only recently (in the month of March this year), and that too is not complete. There's not much description of its APIs in it. This will lead to many problems in developing the custom lint rules, because of no proper documentation and resources, we might end up into some trouble while trying to develop the custom lint checks. Maintaining it in the long run may also lead to some additional costs because of very few resources available, only those who have some prior experience and very-decent knowledge on custom Lint rules can maintain the custom lint checks written in Android Lint for this project.

Hence, because of no official support for the BAZEL build system, no or very less documentation, very less resources, the approach of using Android Lint Lint was rejected for this project.

Technical Design

Note:

- *The implementation algorithms mentioned in the later sections of the proposal are not final. They might need a second thought/revision before finally implementing them in the actual codebase so as to optimize them to the best extent possible.*
- *To make them highly optimized, further revisions might be done to them during the coding period under the guidance of the mentors.*
- *Here to demonstrate the implementation of checks, python scripts have been used. However as per the actual Project Plan, the final scripts will be written in either Kotlin or bash. Python is just used for demonstration purposes only.*

Architectural Overview:

This project involves extensive use of Github Actions to implement the custom static checks and run them during the CI/CD workflow.

Understanding how the `static_checks.yml` workflow gets triggered:

```
name: Static Checks

on:
 workflow_dispatch:
 pull_request:
 push:
   branches:
     - develop
```

- The name of this workflow is "Static Checks"
- This workflow gets triggered on the events: workflow_dispatch, pull_request and push (to the develop branch)
  Note: Here workflow_dispatch is added so that we can also trigger this workflow manually.

This project has been broken down into three subtasks and for achieving every subtask, the respective architectural overview is as follows:

Subtask-1: **Ensuring XML style enforcement**

For this, the project will make use of the Github super-linter
The lint checks for the XML files will run in the **Github Actions** workflow.

For running this check in the CI/CD workflow the `static_checks.yml` file will be modified to perform the linting of the XML files.

```
workflows
  ├ build_tests.yml
  ├ main.yml
  ├ static_checks.yml (Modification)
  ├ unit_tests.yml
  ├ workflow_canceller.yml
```

File-tree overview

Subtask-2: **This subtask will comprise of implementing these checks**

1. Ensuring future checks on file names
2. Ensuring future checks on file contents
3. Verifying all production files have a corresponding test file
4. Verifying activities have accessibility labels
5. Verifying that TODOs correspond to current, open issues on GitHub
6. Ensure KDocs are present for every non-private class, method, and field (even trivial ones).
7. Ensure activities/fragments/views can't be used outside of the app module, or in testing

To implement these checks, ***one file will be added:***

- A custom script (Preferably Kotlin/Bash) in the `scripts` directory

One file will be modified:

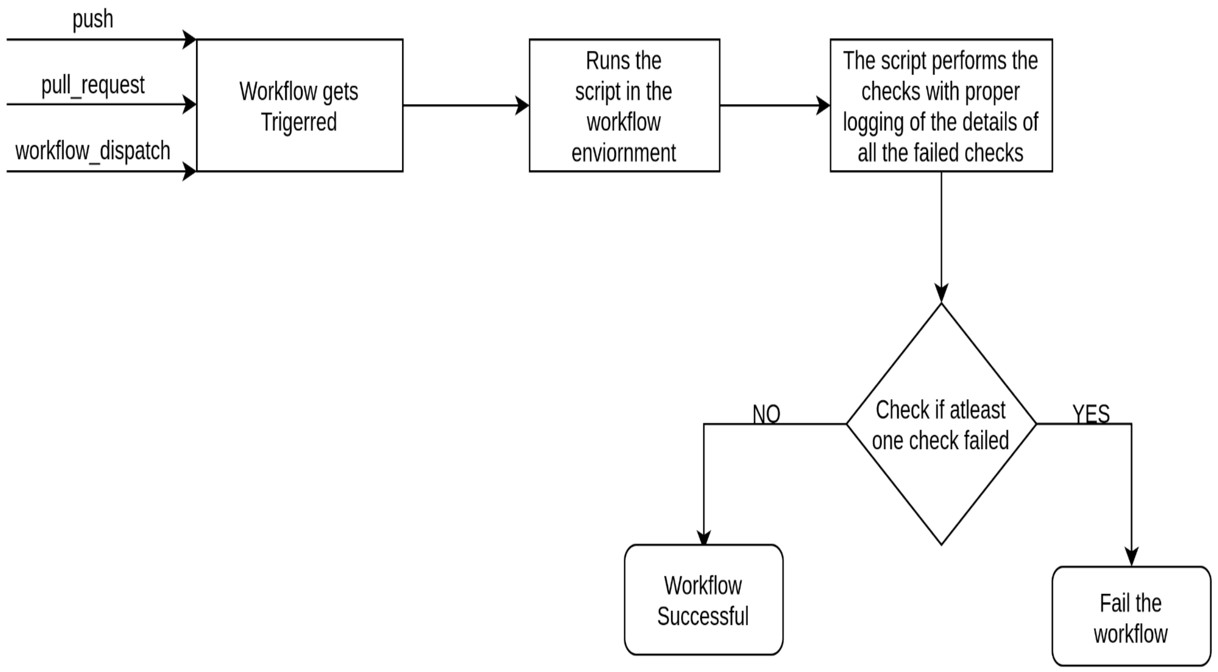
- The static_checks.yml in the .github/workflows directory has to be modified to add the code to run the new script checks

```
-scripts
  ├ custom_script (Addition)
-.github
  ├ workflows
    ├ static_checks.yml (Modification)
```

File-tree overview

All these checks will run on Github Actions workflow via the `static_checks.yml` file.
Along with the current static checks, this workflow file will run the new script `custom_script` in the workflow environment.

The `custom_script` is a script (Preferably Kotlin) which consists of functions to implement the mentioned custom checks.
Overview of how the custom checks will run in the CI environment (Github Actions):


push
pull_request
workflow_dispatch
Workflow gets Trigerred
Runs the script in the workflow enviornment
The script performs the checks with proper logging of the details of all the failed checks
Check if atleast one check failed
NO
YES
Workflow Successful
Fail the workflow

An overview of the main functions in `custom_script`:

```
# function to ensure future checks
# on Filenames
def generic_regex_matcher_filename():

# function to ensure future checks on File
# contents.
def generic_regex_matcher_filecontent():
# function to check the usage of any Non-RTL
# support attribute (say marginLeft)
def attribute_usage_checker():

# function to verify that test files are
# present for every production file
def test_files_present_checker():

# function to ensure activities have accessibility labels
def activity_accesbility_label_checker():

# function to ensure that every TODO
# corresponds to current open issue on Github
def todo_open_issue_verifier():
```

```
# function to ensure that KDocs are present for every
# non-private class, method and field
def kdoc_checker():

# function to ensure that activities/fragments/views can't be used
#  outside of the app module, or in testing
def afv_layer_checker():
```

Subtask-3: **Ensure that no issue is closed without its corresponding TODO being addressed**

To implement this check, two files will be added as follows:

1. `issue_todo_checker_script` in the `scripts` directory
2. `issue_closed.yml` in the `.github/workflows` directory

```
-scripts
  ├ issue_todo_checker.py (Addition)
-.github
  ├ workflows
    ├ issue_closed.yml (Addition)
```

The `issue_closed.yml` is a workflow that will be triggered whenever any issue is closed, it will run the `issue_todo_checker_script` and will pass the `${{ github.event.issue.number }} as a command line argument to` the script.
This script verifies that the corresponding TODO for this issue has been addressed. If it founds that the TODO for the issue hasn't been addressed then it will reopen the issue (along with a comment added) and in the end it will fail the `issue_closed.yml` workflow

An overview of the main functions of `issue_todo_checker_script:`

```
# function to walk over the codebase and check that if any TODO is
# present corresponding to issue provided in the command line argument (
sys.argv[1])
def todo_crawler(issue_number):


# function which makes a PATCH request to reopen the issue
def reopen_issue(issue_number):
```

Implementation Approach

To implement the script and the workflow files to run the scripts during the CI/CD workflow, the code samples are as follows:

1. **Ensuring XML style enforcement:**

Note for the reviewers:

When we search for the linting options for the XML files, there appears only to be two possible best solutions which work during the Github Actions workflow.

1. ***Using Github Super Linter (which uses xmllint to lint the Xml files)***
2. ***Using Android Lint***

The main advantage of using Android Lint over Github Super Linter for linting the XML files is that it offers tons of lint checks when we compare it to Github Super Linter.

Both AndroidLint and Github Super Linter were run on the codebase to get a comparative study of all the lint errors reported by both of them.

Lint errors/warnings reported by AndroidLint: Workflow link: here

- MissingContentDescription check
- UnusedResources check
- HardcodedText check
- OldTargetApi check
- VectorRaser check (warns limiting vector sizes to 200x200 to keep icon drawing fast)
- LockedOrientationActivity check
- Typos check
- Use CompoundDrawables check
- and many more ...

Lint errors/warnings reported by Github Super Linter: Workflow link: here

- No error/warning was reported.

Conclusion: **Android Lint checks are much more powerful when compared to xmllint (Github Super Linter). On one side Android Lint reported almost 10 errors/warnings and on the other side xmllint didn't report even a single error/warning.**

But for our project, there are some problems associated with using Android Lint in the CI workflow.
Android Lint can be run by two methods:

1. By invoking the lint task using the gradle wrapper (Reference: here)
2. Running Lint as a standalone tool. (Reference: here)

Since, we will completely migrate to Bazel in some time so the first option is of no use. Hence, we have to use the second option i.e. to by running Lint as a standalone tool.

Now, we have to run the lint checks in the Github Actions workflow, so first we have to set up the Android SDK tools in the workflow environment.
For achieving this the only Github Action I found that seems to work is: setup-android

Now, when I tried to run the Android Lint as a standalone tool over the codebase using this Github Action,
It throwed this error:

```
Error: build.gradle: Error: "app" is a Gradle project. To correctly
analyze Gradle projects, you should run "gradlew lint" instead.
[LintError]
```

This error tells us that, to run lint on a gradle project, we must have to use the gradle wrapper. if we would run lint using a standalone tool it will give this error.
To find a fix for this error, I searched for a solution on stackoverflow. Here is the link to it.
The solution suggests a hacky approach i.e. what it says is to first delete the gradle files from the repository, perform the lint checks and then again add the gradle files back to the repo once we are done with the linting part.
I applied the similar approach and here's the link to the code.
Well, the solution works too but here are some associated problems with it:

1. Because of deleting the gradle files, the linter started reporting the following error:

```
Class referenced in the layout file,
androidx.constraintlayout.widget.ConstraintLayout, was not found in the
project or the libraries [MissingClass]
```

Well, we can actually suppress this error/Issue by changing its severity from ERROR to IGNORE in the lint.xml file. More reference on this: here
So, I did the similar thing by adding a lint.xml file with this code:

```
<?xml version="1.0" encoding="UTF-8"?>
<lint>
<!-- Disable the given check in this project →
<issue id="MissingClass" severity="ignore" />
</lint>
```

- But unfortunately that didn't work. The linter again reported the same error. Link to the workflow: here. This seems to be a problem because of the Github Action that we are using. Because as per the Android Lint documentation this approach to suppress the Lint error must work, and there seems no reason to why it shouldn't work here, so it is most likely a problem with this Github Action.
- Another problem I observed with this Github Action was that, when the Lint is reporting ERRORs the ***workflow should fail, but here the workflow isn't failing in the end*** despite the Linter throwing errors. This seems another problem with this Github Action.

Conclusion: Among the two options for linting the XML files, Android Lint is undoubtedly a better option than Github Super Linter, but for our case since we can't use the gradle wrapper because of shifting to Bazel in some time we have to use lint as a standalone tool during the Github Actions workflow, but here we saw that, there are some problems associated with the Github Action which sets up the Sdk Tools. And this is the only Github Action present which somehow works. There are no other Github Actions to achieve this or even if there are, they don't seem to work.

Also, there is no option to enforce the team's style guide (oppia-android Style guide) in any of the two options. It seems like it is only a IntelliJ specific thing. And no possible linting solution gives us the customizability to enforce this style guide.

So, this is actually a problematic situation because on one side we have Github Super Linter which performs very few lint checks and on the other side there is Android Lint, which in our case is associated with the problems as mentioned above. Also, I have researched thoroughly and it seems like there are no other possible lint options for the Xml files other than these two that work during the Github Actions workflow. Since both the possible solutions are having these associated problems with them, I think that this is the case where the oppia-android team has to make some critical decisions on what should be done.

In the proposal I am continuing with the Github Super Linter approach to lint the Xml files (but we do have to compromise on the no. of checks that it performs) because the Android Lint solution is a hacky one and that too has many problems associated with it.

- The project makes use of Github Super Linter to perform the linting of the XML files.
- This linter works during the CI workflow.
- It is a collection of several linters, but since we need only the XML linter for our use case because we already have linters present for the other types of the files, hence I have customised the workflow file so that, the Github Super Linter only lints the XML files.

Workflow code to run the Github Super Linter to lint the XML files during the CI workflow:

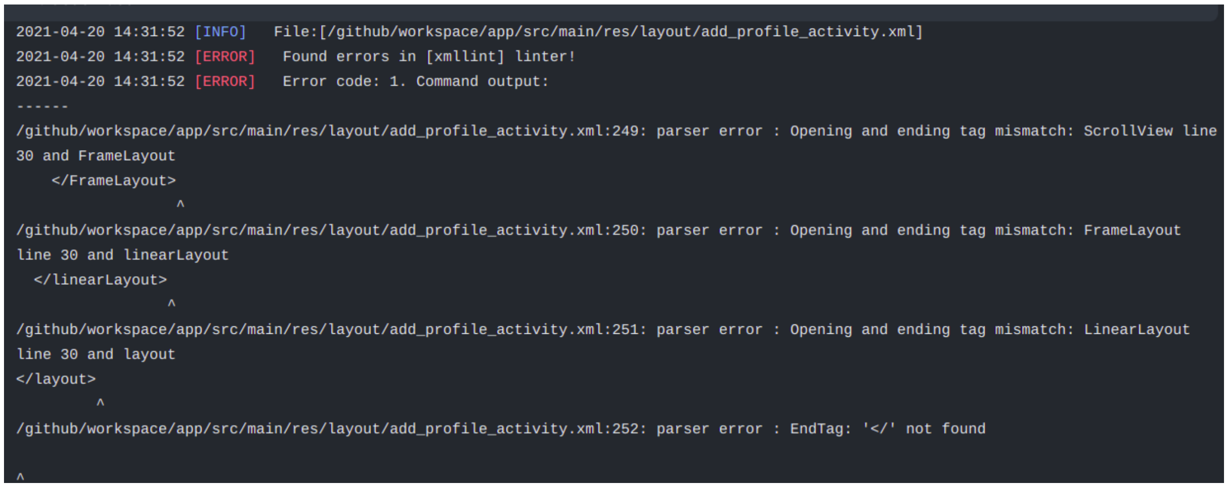
```
    - name: Lint Code Base
      uses: github/super-linter@v3
      env:
        VALIDATE_ALL_CODEBASE: true
        VALIDATE_XML: true
        DEFAULT_BRANCH: master
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        FILTER_REGEX_INCLUDE: .*app/.*
```

I intentionally made some Xml syntax errors in the layout files to check what all errors the Github Super Linter is able to catch.

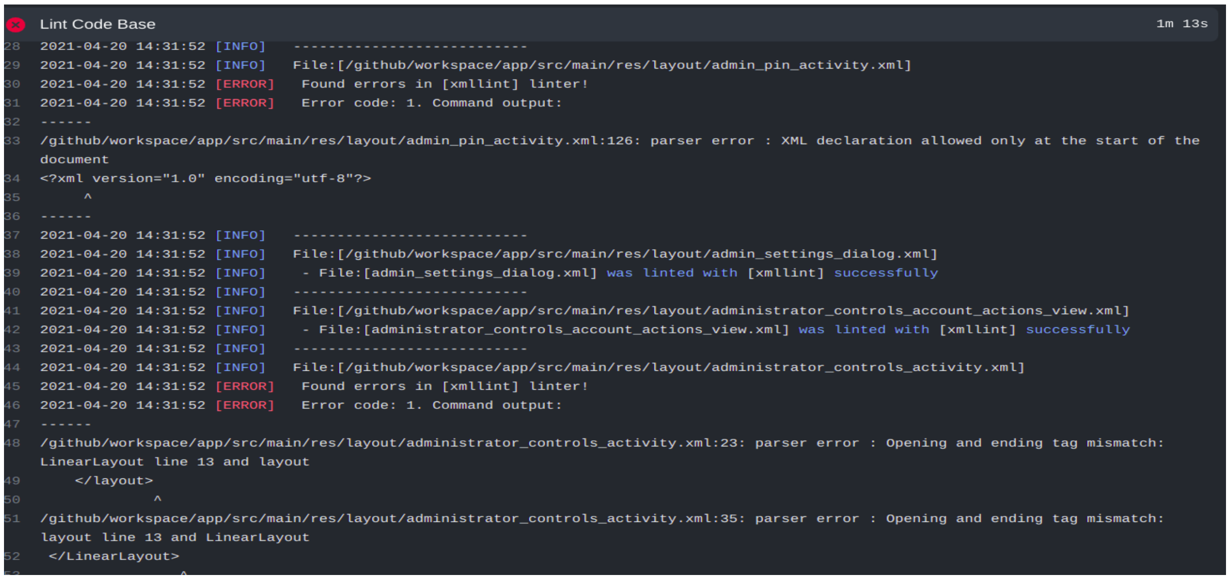
Here is a list of all the errors that the linter was able to catch:

- Intentionally two root elements were put in one of the Xml files and the linter caught it successfully.
- The XML Prolog (`<?xml version="1.0" encoding="UTF-8"?>`) was intentionally put at the end of the Xml file and the linter was able to identify this issue.
- For one of the elements, the closing tag was missing and the linter successfully caught it.
- The same attribute was used twice for an element and the linter reported this error.
- In one of the elements the closing tag was improper ("/" was missing) and the linter was able to catch this.
- In one of the elements the case of the closing tag was made different from the opening tag and the linter was able to catch this syntax error.
- The linter also reported the illegal character ("<") used in the Xml elements.
- It also checks for the correct syntax of the comments in Xml files.

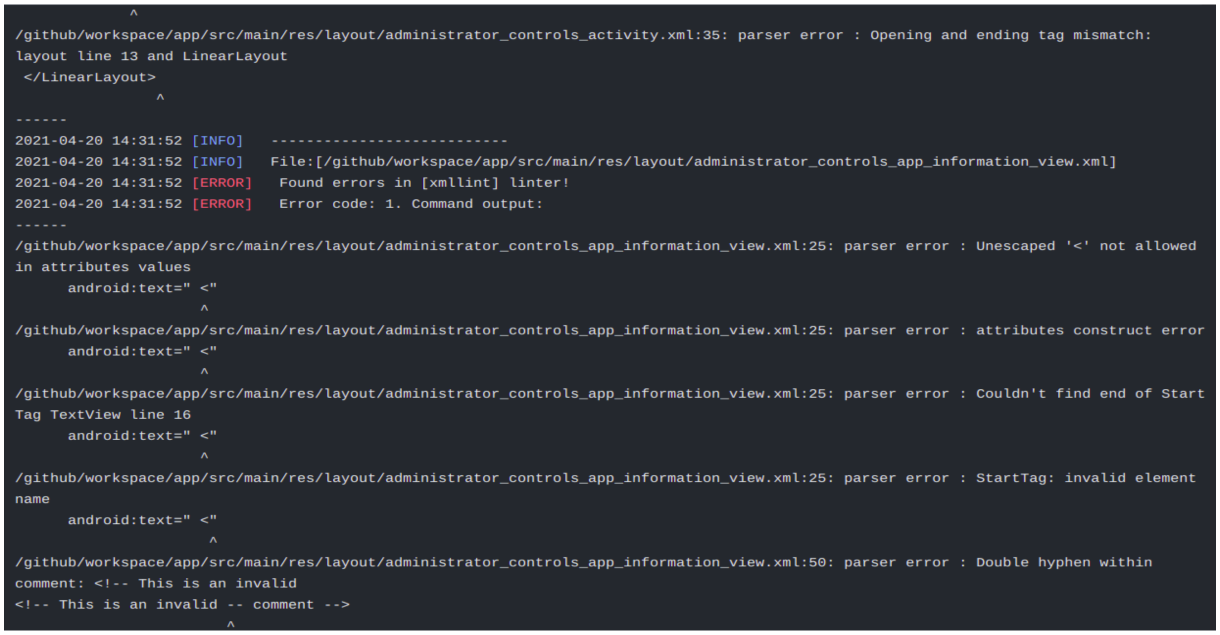
Snapshots of the workflow for reference:



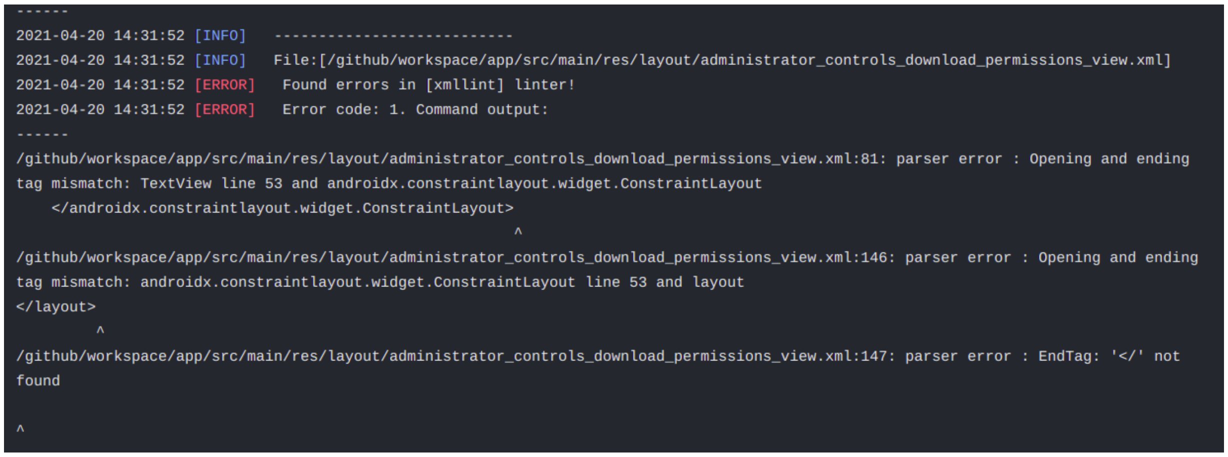
```
2021-04-20 14:31:52 [INFO]   File:[/github/workspace/app/src/main/res/layout/add_profile_activity.xml]
2021-04-20 14:31:52 [ERROR]   Found errors in [xmllint] linter!
2021-04-20 14:31:52 [ERROR]   Error code: 1. Command output:
------
/github/workspace/app/src/main/res/layout/add_profile_activity.xml:249: parser error : Opening and ending tag mismatch: ScrollView line
30 and FrameLayout
    </FrameLayout>
                  ^
/github/workspace/app/src/main/res/layout/add_profile_activity.xml:250: parser error : Opening and ending tag mismatch: FrameLayout
line 30 and linearLayout
  </linearLayout>
                 ^
/github/workspace/app/src/main/res/layout/add_profile_activity.xml:251: parser error : Opening and ending tag mismatch: LinearLayout
line 30 and layout
</layout>
         ^
/github/workspace/app/src/main/res/layout/add_profile_activity.xml:252: parser error : EndTag: '</' not found

^
```



```
Lint Code Base                                                                                                   1m 13s
2021-04-20 14:31:52 [INFO]   --------------------------
2021-04-20 14:31:52 [INFO]   File:[/github/workspace/app/src/main/res/layout/admin_pin_activity.xml]
2021-04-20 14:31:52 [ERROR]   Found errors in [xmllint] linter!
2021-04-20 14:31:52 [ERROR]   Error code: 1. Command output:
------
/github/workspace/app/src/main/res/layout/admin_pin_activity.xml:126: parser error : XML declaration allowed only at the start of the
document
<?xml version="1.0" encoding="utf-8"?>
     ^
------
2021-04-20 14:31:52 [INFO]   --------------------------
2021-04-20 14:31:52 [INFO]   File:[/github/workspace/app/src/main/res/layout/admin_settings_dialog.xml]
2021-04-20 14:31:52 [INFO]    - File:[admin_settings_dialog.xml] was linted with [xmllint] successfully
2021-04-20 14:31:52 [INFO]   --------------------------
2021-04-20 14:31:52 [INFO]   File:[/github/workspace/app/src/main/res/layout/administrator_controls_account_actions_view.xml]
2021-04-20 14:31:52 [INFO]    - File:[administrator_controls_account_actions_view.xml] was linted with [xmllint] successfully
2021-04-20 14:31:52 [INFO]   --------------------------
2021-04-20 14:31:52 [INFO]   File:[/github/workspace/app/src/main/res/layout/administrator_controls_activity.xml]
2021-04-20 14:31:52 [ERROR]   Found errors in [xmllint] linter!
2021-04-20 14:31:52 [ERROR]   Error code: 1. Command output:
------
/github/workspace/app/src/main/res/layout/administrator_controls_activity.xml:23: parser error : Opening and ending tag mismatch:
LinearLayout line 13 and layout
    </layout>
             ^
/github/workspace/app/src/main/res/layout/administrator_controls_activity.xml:35: parser error : Opening and ending tag mismatch:
layout line 13 and LinearLayout
 </LinearLayout>
                ^
```



```
              ^
/github/workspace/app/src/main/res/layout/administrator_controls_activity.xml:35: parser error : Opening and ending tag mismatch:
layout line 13 and LinearLayout
 </LinearLayout>
                ^
------
2021-04-20 14:31:52 [INFO]   --------------------------
2021-04-20 14:31:52 [INFO]   File:[/github/workspace/app/src/main/res/layout/administrator_controls_app_information_view.xml]
2021-04-20 14:31:52 [ERROR]   Found errors in [xmllint] linter!
2021-04-20 14:31:52 [ERROR]   Error code: 1. Command output:
------
/github/workspace/app/src/main/res/layout/administrator_controls_app_information_view.xml:25: parser error : Unescaped '<' not allowed
in attributes values
      android:text=" <"
                     ^
/github/workspace/app/src/main/res/layout/administrator_controls_app_information_view.xml:25: parser error : attributes construct error
      android:text=" <"
                     ^
/github/workspace/app/src/main/res/layout/administrator_controls_app_information_view.xml:25: parser error : Couldn't find end of Start
Tag TextView line 16
      android:text=" <"
                     ^
/github/workspace/app/src/main/res/layout/administrator_controls_app_information_view.xml:25: parser error : StartTag: invalid element
name
      android:text=" <"
                      ^
/github/workspace/app/src/main/res/layout/administrator_controls_app_information_view.xml:50: parser error : Double hyphen within
comment: <!-- This is an invalid
<!-- This is an invalid -- comment -->
                        ^
```



```
------
2021-04-20 14:31:52 [INFO]   --------------------------
2021-04-20 14:31:52 [INFO]   File:[/github/workspace/app/src/main/res/layout/administrator_controls_download_permissions_view.xml]
2021-04-20 14:31:52 [ERROR]   Found errors in [xmllint] linter!
2021-04-20 14:31:52 [ERROR]   Error code: 1. Command output:
------
/github/workspace/app/src/main/res/layout/administrator_controls_download_permissions_view.xml:81: parser error : Opening and ending
tag mismatch: TextView line 53 and androidx.constraintlayout.widget.ConstraintLayout
    </androidx.constraintlayout.widget.ConstraintLayout>
                                                       ^
/github/workspace/app/src/main/res/layout/administrator_controls_download_permissions_view.xml:146: parser error : Opening and ending
tag mismatch: androidx.constraintlayout.widget.ConstraintLayout line 53 and layout
</layout>
         ^
/github/workspace/app/src/main/res/layout/administrator_controls_download_permissions_view.xml:147: parser error : EndTag: '</' not
found

^
```

Link of the workflow: here

2. Ensuring future checks on file contents

Implementation Algorithm:

- First we have to create a list of all the prohibited attributes (say `noRtlSupportAttributes`) that we want to check for in the XML files.
- The next step is to iterate through the files and check if the file is an XML file by checking the file extension.
- The next step involves parsing the XML file to a XML tree so that we can iterate through the elements in the tree.

- Using the element iterator we have to traverse through each and every element in the root tree, its sub trees, their sub trees and so on.
- The final step would be to check that for every element, extract the list of all its attributes and check that if any attribute belongs to the list `noRtlSupportAttributes`, if true then log the details of the element for which the violation was found, after doing this for every element for which the violation occurred, fail the script check in the end with the exit code 1 (so as to fail the workflow)
- We can also filter this by folder too. It is necessary because this check is helpful for the layout/*.xml files whereas for colors.xml it is not much helpful. To filter by folder, we just have to change the target directory path (where it scans/looks for XML files) in the function.

Sample Implementation in Python:

```python
def attribute_usage_checker():

   target_dir = './app/src/main/res'
   global problem_flag
   noRtlSupportAttributes = ['marginLeft', 'marginRight', 'paddingLeft'
                              , 'paddingRight']
   for (root_dir_path, sub_dirs, files) in os.walk(target_dir):
       for file in files:
           if file.endswith('.xml'):
               try:
                   accessibilityTree = \
                       ET.parse(os.path.join(root_dir_path, file))
               except:
                   continue
               for elem in accessibilityTree.iter():
                   for attrib in elem.attrib:
                       for prohibitedAttrib in noRtlSupportAttributes:
                           if attrib.endswith(prohibitedAttrib):
                               problem_flag = True
                               print (prohibitedAttrib \
                                   + ' usage found in: ' + init_path \
                                   + os.path.join(root_dir_path,
                                       file)[1:])
```

Note: *This function currently checks for the usage of four prohibited attributes, if we want to check for any other attribute then, we just have to add that attribute to the `noRtlSupportAttributes` list. This way adding more checks in the future would be easy because we just have to add an extra item to the list.*

3. **Ensuring Generic Regex pattern matching against file names:**

   Implementation Algorithm:

   - To implement this particular check we have to supply the two Generic Regex patterns to the function.
   - Namely *Matching file paths generic regex* (say ***`matchingFilePathRegEx`***) and *Prohibited name generic regex* (say ***`prohibitedNameRegEx`***).
   - Now how the function will work is that it will iterate through the files of the codebase and will match the file path patterns as per the provided matchingFilePathRegEx. What this will result in the end is that we will finally have a matching set of files on which we desire to perform the file name check.
   - Now once we have got the matching set of files, we would then iterate over the file names to match them against the prohibitedNameRegEx.
   - If any of the filename matches with the prohibited RegEx then we will fail the workflow in the end along with proper logging of the details where all it failed.
   - This generic Regex pattern matching will apply for every type of files present, we just have to pass the ***`matchingFilePathRegEx`*** to make it search for the desired type of files that we want to check.

   Example usage:

   - For example: We can make use of this Generic Regex pattern matcher to check that the XML files are snake_case.
   - By providing ***`xmlFileRegEx`*** and ***`notSnakeCaseRegEx`*** to the generic function mentioned above. It will iterate through the codebase, will find all the XML files based on the xmlFileRegEx file path pattern and then on the matched set of files it will apply the notSnakeCaseRegEx check to check that if the file naming is not done as per the snake_case convention.

Sample Implementation of generic regex name matcher in Python:

```
def generic_regex_matcher_filename(filePathPatternRegEx,
                                   prohibitedFileNameRegEx):
    matching_files = []
    for (root_dir_path, sub_dirs, files) in os.walk(target_dir):
        for file in files:
                currentFilePath = os.path.join(root_dir_path, file)
                if re.match(filePathPatternRegEx, currentFilePath):
                    matching_files.append(currentFilePath)
```

```
for filePath in matching_files:
    fileName = file_name(filePath)
    if re.match(prohibitedFileNameRegEx, fileName):
        print ('Prohibited file naming for file: ' + filePath)
```

4. **Ensuring Generic Regex pattern matching against file contents:**

   Implementation Algorithm:

   - To implement this particular check we have to supply the two Generic Regex patterns to the function.
   - Namely *Matching file paths generic regex* (say ***`matchingFilePathRegEx`***) and *Prohibited content generic regex* (say ***`prohibitedContentRegEx`***).
   - Now how the function will work is that it will iterate through the files of the codebase and will match the file path patterns as per the provided matchingFilePathRegEx. What this will result in the end is that we will finally have a matching set of files on which we desire to perform the file content check.
   - Now once we have got the matching set of files, we would then iterate over the file line by line.
   - For every line string of the file, if the string matches with the ***`prohibitedContentRegEx`*** pattern, then we will fail the workflow (with logging the details of all failures)

   Example usage:

   - For example: We can make use of this Generic file content RegEx matcher to implement the **attribute_usage_checker()** function.
   - Note that, earlier we implemented this function by first parsing the XML file and then traversing over the XML tree generated and then many other steps involved further.
   - But using the generic RegEx matcher, we can achieve the same functionality by just passing the two RegEx patterns to the generic matcher. First is the XML layout file pattern Regex (say **layoutXmlRegEx**) and the second is the prohibited attribute RegEx (say **prohibitedAttrRegEx**). That's it! The rest of the job will be done by the generic Regex content matcher that we created. It will iterate over the codebase to first create a set of all the layout XML files and will then scan the XML files to match the prohibited attribute RegEx pattern.
   - This shows how easy it will be to add the future file content checks.

Sample Implementation of generic regex file content matcher in Python:

```python
def generic_regex_matcher_filecontent(filePathPatternRegEx,
        prohibitedContentRegEx):
    matching_files = []
    for (root_dir_path, sub_dirs, files) in os.walk(target_dir):
        for file in files:
            currentFilePath = os.path.join(root_dir_path, file)
            if re.match(filePathPatternRegEx, currentFilePath):
                matching_files.append(currentFilePath)

    for filePath in matching_files:
        f = open(filePath, 'r')
        for linestr in f:
            if re.match(prohibitedContentRegEx, linestr):
                print ('Prohibited content usage found in: ' + filePath)
```

5. Verifying activities have accessibility labels

Implementation Algorithm:

- The first step will be to provide the path of the `AndroidManifest.xml` to the XML parser so that the XML file could be parsed into an XML tree and then we could traverse it.
- In the XML tree generated we then have to iterate over every `activity` element and check the following conditions.
- Not every activity has to be checked for the accessibility labels, the Test Activities are not meant for the general users of the app and hence, we don't need to do the accessibility label check for those activities. So we have to make sure that these activities are skipped for the check.
- The way we will skip them is by checking the `name` attribute of the activity element. If the value of the `name` attribute includes the testing package then it implies that the corresponding activity is a Test activity and hence we will skip it.
- For the activity elements which do not have the testing package in their name attribute, those are the ones for which the check needs to be done and we will ensure that they have labels present or not by just checking whether they have the `label` attribute present in the attributes list or not.
- If not then that implies the corresponding activity does not have the accessibility label present and we will then log the detail for this activity element so as to provide the details of which activity failed this check.
- In the end, just like the previous cases we will fail the workflow if the check fails for any of the activity.

Sample Implementation in Python:

```
def activity_accesbility_label_checker():

   global problem_flag
   accessibilityTree = ET.parse('./app/src/main/AndroidManifest.xml')
   accessibilityRoot = accessibilityTree.getroot()

   for neighbor in accessibilityRoot.iter('activity'):
       if not 'testing' in neighbor.attrib.get(name_attrib):
           if not label_attrib in neighbor.attrib:
               problem_flag = True
               print ('No accessibility label found for ' \
                   + neighbor.attrib.get(name_attrib).split('.')[-1])
```

6. Verifying all production files have a corresponding test file

Implementation Algorithm 1:

1. Every prod. File is present in its corresponding sub package.

   Now here the question arises: How we will be find these prod files:

   - The answer to this is that the production files are spread across different modules. But we don't need to have a Test file for every type of file. For example: in the app module we don't need to have a Test file for "TopicActivityPresenter" or for "ViewPageAdapter".
   - So to find the correct set of prod. Files (the prod. Files for which we actually need to check for corresponding Test files present) some one-time manual work would be required. By one-time manual inspection, we first have to create a list of all those **types of files** which we think should be checked for the corresponding Test files present. For ex: Activity files, Fragment files, Controllers, and so on.
   - After figuring this out, we will have to create a generic regex pattern(s) (either a single RegEx which covers all the cases or a list of Regex patterns for each type) to match the pattern of these types of files.
   - Once we are ready with the generic Regex pattern which matches those files which needs to be checked for the presence of a Test file in the codebase, we can then simply iterate throughout the codebase and will apply this Regex on the file paths to get the list of all the desired prod files which needs to be checked for their corresponding Test files. Then we can move to the next step of the algorithm.

- We first have to collect all the prod file names in a list (say `prod_files_list`) by traversing through the codebase and checking if the file path pattern matches the prod file Regex pattern and then appending the file name in this list if it matches.
- Then we have to iterate over all the files in the `test` directories and append the name of all the Test files in a list (say `test_files_list`).
- After completing step-1 and step-2, we will have two lists i.e `prod_files_list` and `test_files_list` in which the former contains the names of all the prod files and the latter contains the names of all the test files in the codebase.
- The final step involves checking that for every `file_name` present in the `prod_files_list`, `file_name + 'Test'` must exist in the `test_files_list.`
- If for any file this is not true, then that implies that the particular prod file does not have a corresponding Test file present in the codebase. And hence, after logging the details for the particular file, we will fail the workflow if this check is not true for any of the files.
- Hence this approach will work even for the subpackages having multiple files because we are collecting all the prod files.

Running these checks on CI:

Implementation Algorithm:

- To run the scripts in the CI environment, we first have to mention the events for which the workflow should get triggered
- The next step would be to define the job and the first step of the job is to checkout the repository
- The next step involves changing the directory to the scripts directory and then run the desired script.
- For example: Here we are running the Python script, so before running the script we also have to setup Python in the workflow and then run the Python script, similarly we can adopt this procedure for any other script say bash/kotlin or any other scripts.

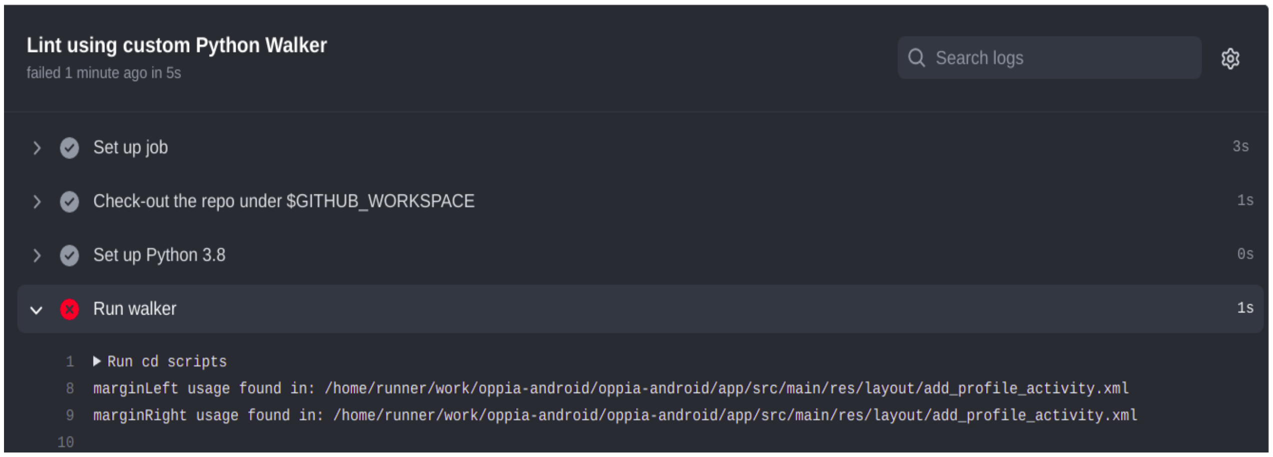
Sample workflow implementation to run Python script:

```
name: Custom Static Checks

on:
 workflow_dispatch:
 pull_request:
 push:

jobs:
 run-custom-linter-job:
```

```yaml
name: Lint using custom Python Walker
runs-on: ubuntu-latest
steps:
  - name: Check-out the repo under $GITHUB_WORKSPACE
    uses: actions/checkout@v2

  - name: Set up Python 3.8
    uses: actions/setup-python@v2
    with:
      python-version: '3.8'

  - name: Run walker
    run: |
      cd scripts
      python custom_linter.py
```

Working Proof:

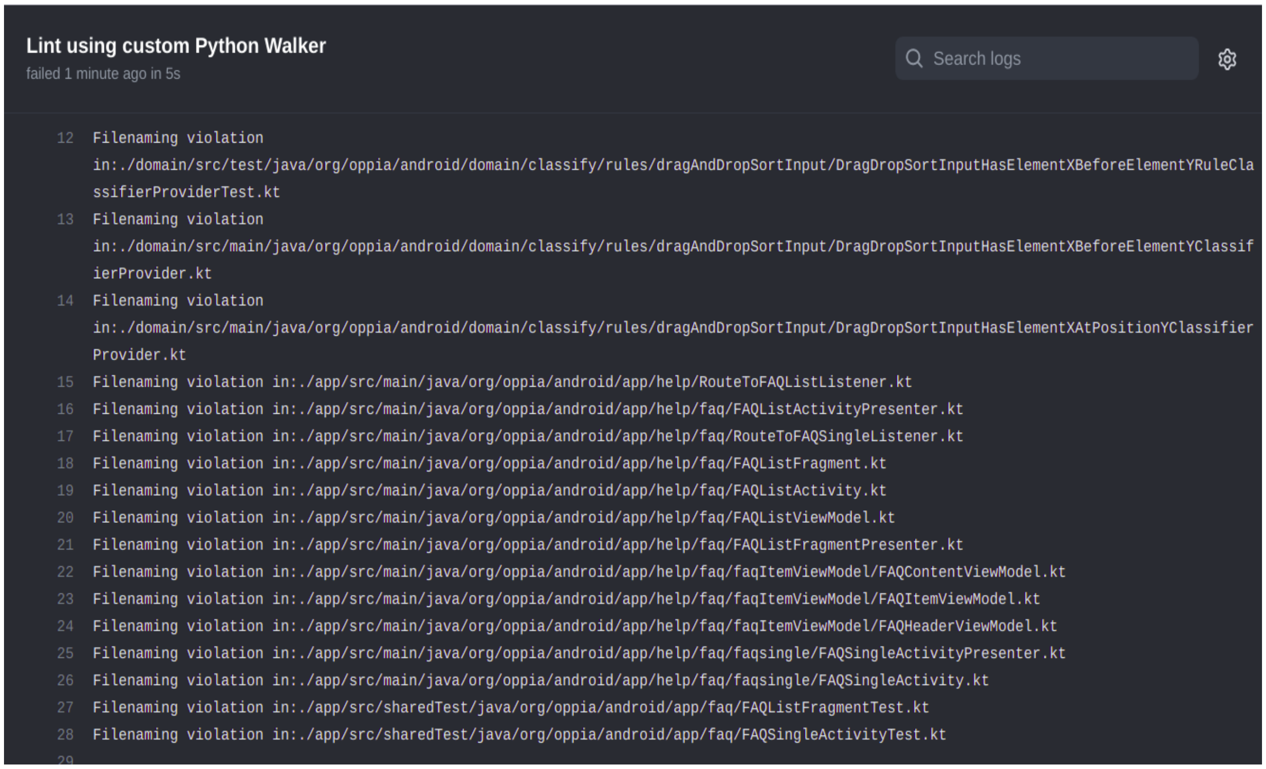
- Intentionally, `marginLeft` and `marginRight` were used as attributes in one of the XML files to check the custom lint checks, and it successfully caught those during the workflow.


Lint using custom Python Walker
failed 1 minute ago in 5s

Set up job 3s
Check-out the repo under $GITHUB_WORKSPACE 1s
Set up Python 3.8 0s
Run walker 1s

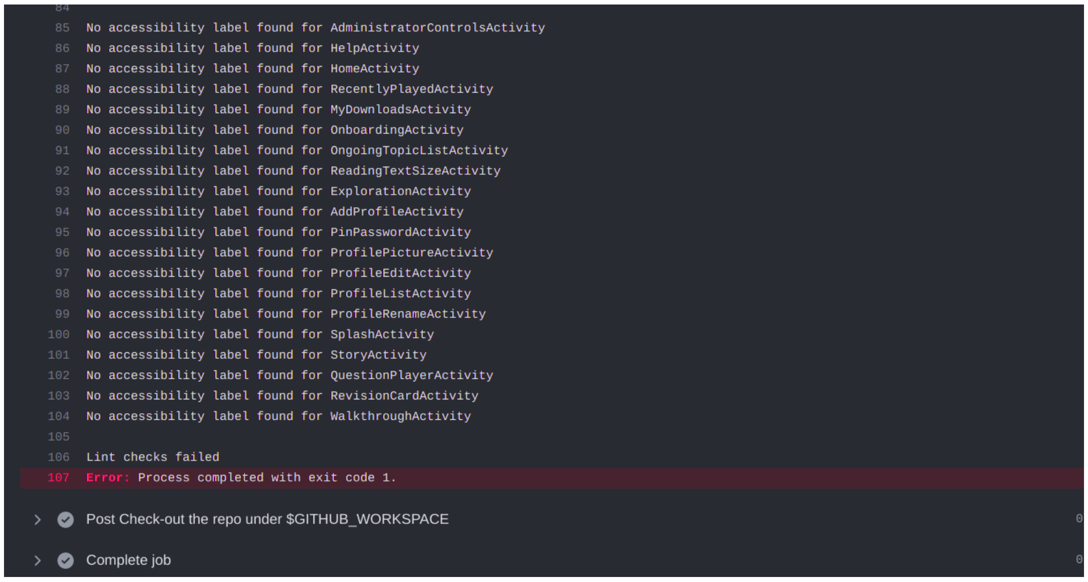
```
▶ Run cd scripts
marginLeft usage found in: /home/runner/work/oppia-android/oppia-android/app/src/main/res/layout/add_profile_activity.xml
marginRight usage found in: /home/runner/work/oppia-android/oppia-android/app/src/main/res/layout/add_profile_activity.xml
```

- The filenames for many of the kotlin files were not as per the camelcase convention. The custom lint checks made, were successfully able to detect the file naming violation.


Lint using custom Python Walker
failed 1 minute ago in 5s

```
Filenaming violation
in:./domain/src/test/java/org/oppia/android/domain/classify/rules/dragAndDropSortInput/DragDropSortInputHasElementXBeforeElementYRuleClassifierProviderTest.kt
Filenaming violation
in:./domain/src/main/java/org/oppia/android/domain/classify/rules/dragAndDropSortInput/DragDropSortInputHasElementXBeforeElementYClassifierProvider.kt
Filenaming violation
in:./domain/src/main/java/org/oppia/android/domain/classify/rules/dragAndDropSortInput/DragDropSortInputHasElementXAtPositionYClassifierProvider.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/RouteToFAQListListener.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/FAQListActivityPresenter.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/RouteToFAQSingleListener.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/FAQListFragment.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/FAQListActivity.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/FAQListViewModel.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/FAQListFragmentPresenter.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/faqItemViewModel/FAQContentViewModel.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/faqItemViewModel/FAQItemViewModel.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/faqItemViewModel/FAQHeaderViewModel.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/faqsingle/FAQSingleActivityPresenter.kt
Filenaming violation in:./app/src/main/java/org/oppia/android/app/help/faq/faqsingle/FAQSingleActivity.kt
Filenaming violation in:./app/src/sharedTest/java/org/oppia/android/app/faq/FAQListFragmentTest.kt
Filenaming violation in:./app/src/sharedTest/java/org/oppia/android/app/faq/FAQSingleActivityTest.kt
```

- The accessibility labels were missing for many of the activities (meant for the general users of the app). The custom lint checks identified those activities and reported them.



```
No accessibility label found for AdministratorControlsActivity
No accessibility label found for HelpActivity
No accessibility label found for HomeActivity
No accessibility label found for RecentlyPlayedActivity
No accessibility label found for MyDownloadsActivity
No accessibility label found for OnboardingActivity
No accessibility label found for OngoingTopicListActivity
No accessibility label found for ReadingTextSizeActivity
No accessibility label found for ExplorationActivity
No accessibility label found for AddProfileActivity
No accessibility label found for PinPasswordActivity
No accessibility label found for ProfilePictureActivity
No accessibility label found for ProfileEditActivity
No accessibility label found for ProfileListActivity
No accessibility label found for ProfileRenameActivity
No accessibility label found for SplashActivity
No accessibility label found for StoryActivity
No accessibility label found for QuestionPlayerActivity
No accessibility label found for RevisionCardActivity
No accessibility label found for WalkthroughActivity

Lint checks failed
Error: Process completed with exit code 1.
```

Link to the workflow: here
Link to the commit: here

7. Ensure KDocs are present for every non-private class, method, and field (even trivial ones)

Implementation Algorithm:

1. Iterate through the codebase and check if the file is a Kotlin file.

2. The next step involves parsing the Kotlin file and then while traversing through the AST check if `elementType == CLASS` is true.

3. The next step will be to check if the class is a non-private class. That could be done by accessing the child property of the node element. In this we have to check if the child has the `MODIFIER_LIST`, and in the `MODIFIER_LIST` check if it has the children's `elementType` as private present. If not then, we have to check for the presence of KDoc for this class, since it's a non-private class.

4. To check if KDoc is present we just have to check whether it has the `elementType == KDoc` present in its children.

5. The name of the library that will be used to parse the kotlin files to AST is **kotlinx/ast** The Github Link of the library can be found: here.

Note: *This approach was to check the presence of KDoc for a non-private class, we could extend this to functions and fields also.*

8. Verify that TODOs correspond to current, open issues on GitHub

Implementation Algorithm:

Approach for scraping the current open issues:

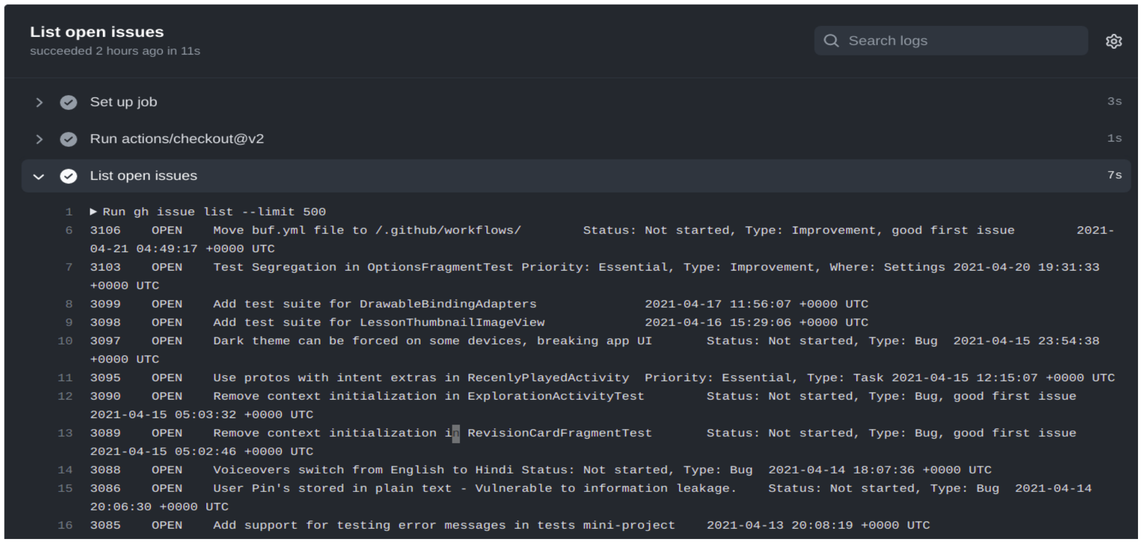
- In this approach we will make use of Github CLI to collect the issues in the Github Actions workflow.
- GitHub CLI comes pre-installed in GitHub Actions virtual environments. Hence, we can directly use its commands in the Github Actions workflow.
- To collect the all the open issues of our repository we will make use of the `gh` command: `gh issue list [flags]`
- Here we will pass the flag: `--limit 500` so as to fetch a maximum of 500 open issues.
- If we won't pass this flag then by default it results in displaying the recent 30 open issues. But for our case we want all the open issues hence, we will pass 500 to get all of them.
- Note that we can also limit it to 1000 or 2000 to be on a safer side, here 500 is just used as an example.
- This results in displaying all the open issues of the repository. We can then feed this issue data to the script by saving the output of the above command to a file.

POC of collecting all the open issues by the optimized approach:

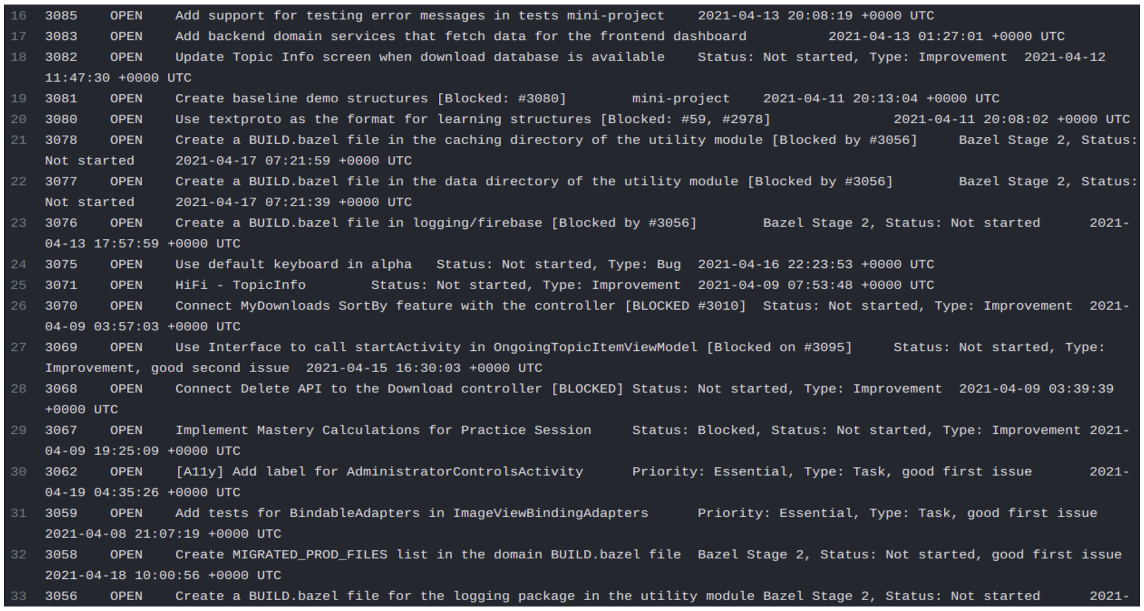
Sample workflow code to collect the open issues:

```
steps:
  - uses: actions/checkout@v2

  - name: List open issues
    run: |
      gh issue list --limit 500
    env:
      GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

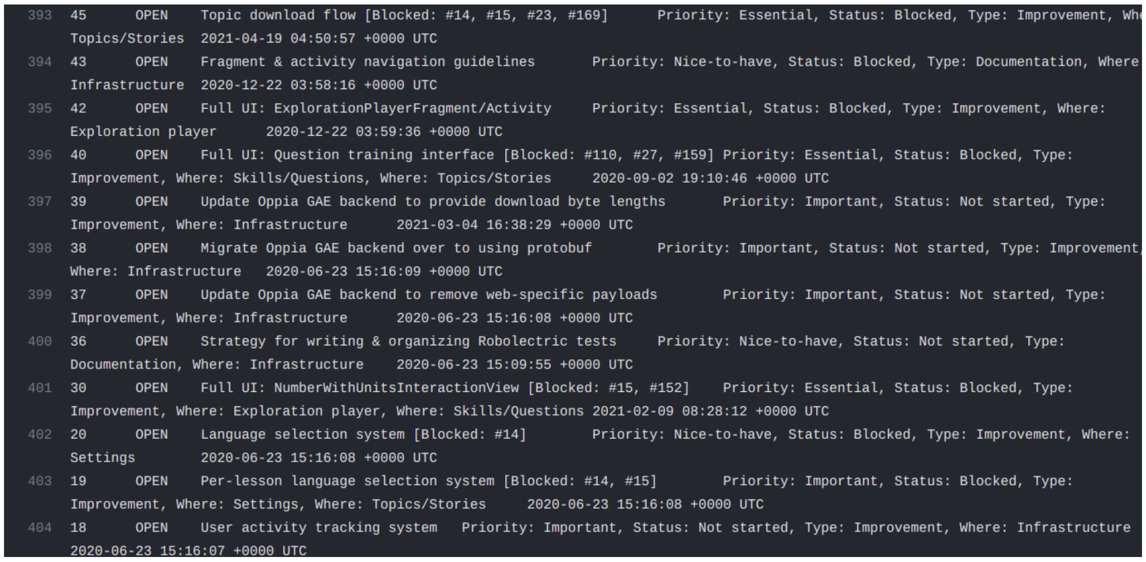
Workflow Results:



```
▶ Run gh issue list --limit 500
3106    OPEN    Move buf.yml file to /.github/workflows/        Status: Not started, Type: Improvement, good first issue        2021-04-21 04:49:17 +0000 UTC
3103    OPEN    Test Segregation in OptionsFragmentTest Priority: Essential, Type: Improvement, Where: Settings 2021-04-20 19:31:33 +0000 UTC
3099    OPEN    Add test suite for DrawableBindingAdapters              2021-04-17 11:56:07 +0000 UTC
3098    OPEN    Add test suite for LessonThumbnailImageView             2021-04-16 15:29:06 +0000 UTC
3097    OPEN    Dark theme can be forced on some devices, breaking app UI       Status: Not started, Type: Bug  2021-04-15 23:54:38 +0000 UTC
3095    OPEN    Use protos with intent extras in RecenlyPlayedActivity  Priority: Essential, Type: Task 2021-04-15 12:15:07 +0000 UTC
3090    OPEN    Remove context initialization in ExplorationActivityTest        Status: Not started, Type: Bug, good first issue 2021-04-15 05:03:32 +0000 UTC
3089    OPEN    Remove context initialization in RevisionCardFragmentTest       Status: Not started, Type: Bug, good first issue 2021-04-15 05:02:46 +0000 UTC
3088    OPEN    Voiceovers switch from English to Hindi Status: Not started, Type: Bug  2021-04-14 18:07:36 +0000 UTC
3086    OPEN    User Pin's stored in plain text - Vulnerable to information leakage.    Status: Not started, Type: Bug  2021-04-14 20:06:30 +0000 UTC
3085    OPEN    Add support for testing error messages in tests mini-project    2021-04-13 20:08:19 +0000 UTC
```



```
3085    OPEN    Add support for testing error messages in tests mini-project    2021-04-13 20:08:19 +0000 UTC
3083    OPEN    Add backend domain services that fetch data for the frontend dashboard          2021-04-13 01:27:01 +0000 UTC
3082    OPEN    Update Topic Info screen when download database is available    Status: Not started, Type: Improvement  2021-04-12 11:47:30 +0000 UTC
3081    OPEN    Create baseline demo structures [Blocked: #3080]        mini-project    2021-04-11 20:13:04 +0000 UTC
3080    OPEN    Use textproto as the format for learning structures [Blocked: #59, #2978]               2021-04-11 20:08:02 +0000 UTC
3078    OPEN    Create a BUILD.bazel file in the caching directory of the utility module [Blocked by #3056]     Bazel Stage 2, Status: Not started     2021-04-17 07:21:59 +0000 UTC
3077    OPEN    Create a BUILD.bazel file in the data directory of the utility module [Blocked by #3056]        Bazel Stage 2, Status: Not started     2021-04-17 07:21:39 +0000 UTC
3076    OPEN    Create a BUILD.bazel file in logging/firebase [Blocked by #3056]        Bazel Stage 2, Status: Not started      2021-04-13 17:57:59 +0000 UTC
3075    OPEN    Use default keyboard in alpha   Status: Not started, Type: Bug  2021-04-16 22:23:53 +0000 UTC
3071    OPEN    HiFi - TopicInfo        Status: Not started, Type: Improvement  2021-04-09 07:53:48 +0000 UTC
3070    OPEN    Connect MyDownloads SortBy feature with the controller [BLOCKED #3010]  Status: Not started, Type: Improvement  2021-04-09 03:57:03 +0000 UTC
3069    OPEN    Use Interface to call startActivity in OngoingTopicItemViewModel [Blocked on #3095]     Status: Not started, Type: Improvement, good second issue  2021-04-15 16:30:03 +0000 UTC
3068    OPEN    Connect Delete API to the Download controller [BLOCKED] Status: Not started, Type: Improvement  2021-04-09 03:39:39 +0000 UTC
3067    OPEN    Implement Mastery Calculations for Practice Session     Status: Blocked, Status: Not started, Type: Improvement 2021-04-09 19:25:09 +0000 UTC
3062    OPEN    [A11y] Add label for AdministratorControlsActivity      Priority: Essential, Type: Task, good first issue       2021-04-19 04:35:26 +0000 UTC
3059    OPEN    Add tests for BindableAdapters in ImageViewBindingAdapters      Priority: Essential, Type: Task, good first issue 2021-04-08 21:07:19 +0000 UTC
3058    OPEN    Create MIGRATED_PROD_FILES list in the domain BUILD.bazel file  Bazel Stage 2, Status: Not started, good first issue 2021-04-18 10:00:56 +0000 UTC
3056    OPEN    Create a BUILD.bazel file for the logging package in the utility module Bazel Stage 2, Status: Not started      2021-
```



```
45      OPEN    Topic download flow [Blocked: #14, #15, #23, #169]      Priority: Essential, Status: Blocked, Type: Improvement, Whe
Topics/Stories  2021-04-19 04:50:57 +0000 UTC
43      OPEN    Fragment & activity navigation guidelines       Priority: Nice-to-have, Status: Blocked, Type: Documentation, Where
Infrastructure  2020-12-22 03:58:16 +0000 UTC
42      OPEN    Full UI: ExplorationPlayerFragment/Activity     Priority: Essential, Status: Blocked, Type: Improvement, Where: Exploration player      2020-12-22 03:59:36 +0000 UTC
40      OPEN    Full UI: Question training interface [Blocked: #110, #27, #159] Priority: Essential, Status: Blocked, Type: Improvement, Where: Skills/Questions, Where: Topics/Stories     2020-09-02 19:10:46 +0000 UTC
39      OPEN    Update Oppia GAE backend to provide download byte lengths       Priority: Important, Status: Not started, Type: Improvement, Where: Infrastructure      2021-03-04 16:38:29 +0000 UTC
38      OPEN    Migrate Oppia GAE backend over to using protobuf        Priority: Important, Status: Not started, Type: Improvement,
Where: Infrastructure   2020-06-23 15:16:09 +0000 UTC
37      OPEN    Update Oppia GAE backend to remove web-specific payloads        Priority: Important, Status: Not started, Type: Improvement, Where: Infrastructure      2020-06-23 15:16:08 +0000 UTC
36      OPEN    Strategy for writing & organizing Robolectric tests     Priority: Nice-to-have, Status: Not started, Type: Documentation, Where: Infrastructure    2020-06-23 15:09:55 +0000 UTC
30      OPEN    Full UI: NumberWithUnitsInteractionView [Blocked: #15, #152]    Priority: Essential, Status: Blocked, Type: Improvement, Where: Exploration player, Where: Skills/Questions 2021-02-09 08:28:12 +0000 UTC
20      OPEN    Language selection system [Blocked: #14]        Priority: Nice-to-have, Status: Blocked, Type: Improvement, Where: Settings        2020-06-23 15:16:08 +0000 UTC
19      OPEN    Per-lesson language selection system [Blocked: #14, #15]        Priority: Important, Status: Blocked, Type: Improvement, Where: Settings, Where: Topics/Stories     2020-06-23 15:16:08 +0000 UTC
18      OPEN    User activity tracking system   Priority: Important, Status: Not started, Type: Improvement, Where: Infrastructure 2020-06-23 15:16:07 +0000 UTC
```

At the time of running this workflow, there were a total of **399** open issues in the repository. And we can clearly see that it reported the most recent issue #3106 at line no. **6** of the workflow run and the least recent open issue #18 at line no. **404**.
Since, at every new line an issue is displayed so we can count the total no. of issues displayed as **404 - 6 + 1 = 399** which matches exactly with the no. of open issues in the repository. Hence, we can conclude that this approach works correctly.

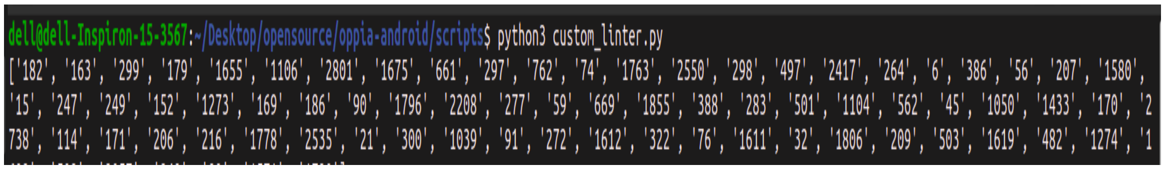
Link to the workflow: here.

- The next step is to extract all the TODOs present in the codebase by using the RegExp pattern of TODOs. We can achieve this by iterating on every line of the file, and check if the line string matches the TODO regex.
- For the line string which matches the TODO regex, the next task would be to extract the issue number from the string. We can achieve this easily by the string functions or maybe from RegExp patterns. And then append the TODO issue numbers to the `todo_issues_list`
- The final step will be to verify that every issue no. present in the `todo_issues_list` should also be present in the `open_issues_`list. And in the end fail the workflow if atleast one TODO didn't pass the check.

Sample Python code for extracting the TODO issues from the codebase:

```
def todo_extractor():

   target_dir = './'
   to_do_list = []
   issues = []
   keyword = 'TODO'
   regPattern = '^(?!.*([\'"]).*\\b' + keyword + '\\b.*\\1)(?!.*\\b' \
       + keyword + '\\.\\w)(?!.*\\b' + keyword + '\\s*=).*\\b' \
       + keyword + '\\b'
   for (root_dir_path, sub_dirs, files) in os.walk(target_dir):
       for file in files:
           if file.endswith('.kt'):
               filePath = os.path.join(root_dir_path, file)
               f = open(filePath, 'r')
               for x in f:
                   if re.match(regPattern, x):
                       x = x.replace('\n', '')
                       to_do_list.append(x)

   to_do_list = list(set(to_do_list))
   for todo in to_do_list:
       issues.append(issue_extractor(todo))
   issues = list(set(issues))
   print (issues)
```

Working proof of this function:



```
dell@dell-Inspiron-15-3567:~/Desktop/opensource/oppia-android/scripts$ python3 custom_linter.py
['182', '163', '299', '179', '1655', '1106', '2801', '1675', '661', '297', '762', '74', '1763', '2550', '298', '497', '2417', '264', '6', '386', '56', '207', '1580',
'15', '247', '249', '152', '1273', '169', '186', '90', '1796', '2208', '277', '59', '669', '1855', '388', '283', '501', '1104', '562', '45', '1050', '1433', '170', '2
738', '114', '171', '206', '216', '1778', '2535', '21', '300', '1039', '91', '272', '1612', '322', '76', '1611', '32', '1806', '209', '503', '1619', '482', '1274', '1
```

Here, we can see that this function extracted the list of all issues from the TODOs present in the kotlin files of the entire codebase. Note that this extractor function could extract the TODOs for every type of file, but for this example it is limited to kotlin files only.

8. Verify that the TODO of the closed issue has been addressed

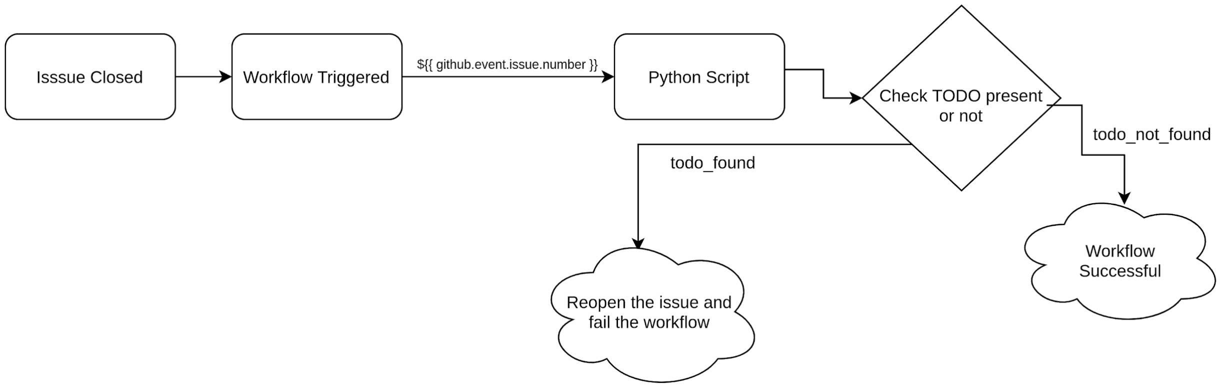
Implementation Algorithm:

1. The first step involves triggering a workflow on the event issue closed.

2. The workflow then passes the issue no. `${{ github.event.issue.number }}` of the closed issue to the script (via command line argument), where the task of the script will be to check that if the TODOs of the passed issue no. is still present in the codebase or not.
3. To check that the TODOs of the passed issue no. is still present in the codebase we will use the same algorithm like we did in the previous case where we iterate through the files, then in every file iterate through every line and match the TODO RegExp and then extract the issue number from the TODO. Then after creating a list of all TODO issue numbers, we will verify that `if ${{ github.event.issue.number }} in todo_issues_list` .

4. If `true`, then reopen the issue by following this approach:

- We will make use of the Github CLI command in the Github Actions workflow to reopen the issue.
- GitHub CLI comes pre-installed in GitHub Actions virtual environments. Hence, we can directly use its commands in the Github Actions workflow.
- To reopen the issue we will make use of the `gh` command:
  `gh issue reopen {<number> | <url>}`
- We can access the issue number through : `${{ github.event.issue.number }}`
- Combining the above two points, we can simply reopen the issue when the check fails in the CI workflow with the single command:
  `gh issue reopen ${{ github.event.issue.number }}`


Isssue Closed
Workflow Triggered
${{ github.event.issue.number }}
Python Script
Check TODO present or not
todo_not_found
todo_found
Workflow Successful
Reopen the issue and fail the workflow

Flow diagram representing the working of the workflow

Adding Comment before reopening the issue:

- Before sending the PATCH request to reopen the issue, we will first send the post request to this endpoint: "`https://api.github.com/repos/oppia/oppia-android/issues/{issue_number}/comments`" to add a comment like this: "This issue has been reopened because of unaddressed TODOs present" and then also adding the info of the files/LOCs where the unaddressed TODOs for this issue are found.

- We can get the info of the files/LOCs by simply maintaining a list while searching for the TODOs in the codebase. And then adding the file-name/file-path of the file containing the matched TODO and the line no. of the TODOs location to this list as soon as we encounter a matched TODO. And while sending the post request to add a comment, we can simply include this list in the body of the comment.

Sample workflow implementation to pass the closed issue to a script:

```yaml
name: Verify TODO Addressed

on:
 issues:
   types: [closed]

jobs:
 verify-todo-addressed:
   name: Verify that TODO is addressed
   runs-on: ubuntu-latest
   steps:
     - name: Check-out the repo under $GITHUB_WORKSPACE
       uses: actions/checkout@v2

     - name: Set up Python 3.8
       uses: actions/setup-python@v2
       with:
         python-version: '3.8'

     - name: Run Issue TODO checker
       run: |
         cd scripts
         python issue_todo_checker.py ${{ github.event.issue.number }}
```

Sample Implementation of Python script:

```python
def todo_crawler(issue_to_check):

   target_dir = './'
   to_do_list = []
   issues = []
   keyword = 'TODO'
   regPattern = '^(?!.*([\'"]).*\\b' + keyword + '\\b.*\\1)(?!.*\\b' \
       + keyword + '\\.\\w)(?!.*\\b' + keyword + '\\s*=).*\\b' \
       + keyword + '\\b'
   for (root_dir_path, sub_dirs, files) in os.walk(target_dir):
       for file in files:
           if file.endswith('.kt'):
               filePath = os.path.join(root_dir_path, file)
               f = open(filePath, 'r')
               for x in f:
                   if re.match(regPattern, x):
                       x = x.replace('\n', '')
                       to_do_list.append(x)

   to_do_list = list(set(to_do_list))
   for todo in to_do_list:
       issues.append(issue_extractor(todo))
   issues = list(set(issues))
   for issue in issues:
     if issue == issue_to_check:
       print('TODO not addressed for issue no. '+issue_to_check)
       return True

   return False


if(todo_crawler(sys.argv[1])):
 comment_and_reopen_issue(sys.argv[1])
 sys.exit(1)
else:
 sys.exit(0)
```

9. Ensure activities/fragments/views can't be used outside of the app module, or in testing

Implementation Algorithm:

- We can implement this check with the help of the generic Regex File content matcher. What basically we will be doing here is that, **we will prohibit the Android imports for View, Fragment and Activity outside the **/app/ file path**.

- The way it will work is that we will pass the two regex patterns to the generic file content regex matcher. The first regex pattern will be of the file path pattern which excludes the app module (say ***`notAppFilePathPatternRegex`***). This regex pattern will lead to a set of files outside the app module. Because we want to check for imports outside the app module.
- The second RegEx pattern will be the Android Activity/View/Fragment import RegEx pattern (say ***`importAVFRegEX`***). The way all this will work is that, first the generic matcher will get us a list of all the desired files outside the app module which we want to check for, and then it scans these files against the import RegEX pattern that we have created. If it matches any of the import patterns then that implies Activity/View/Fragment has been used outside of the app module, hence we will fail the workflow.

Rejected approach to implement this check:

- For any module (except the testing and app module) in order to use the activities/fragments/views from the app module it has to include the app library from the app module as a `deps`.
- So if any module (say x) wants to use any activity/fragment/view from the app module, then it must have to include `“//app:app”` as a `deps` in its `BUILD.bazel` file.
- Hence, the algorithm to achieve this check is to iterate over the BUILD.bazel files of the modules (except testing and app) and then check if `“//app:app”` is used as a `deps` or not.
- If yes then that implies that the activity/fragment or view has been used outside of the app layer/module. Thus, after logging the details of which module(s) failed this check, we have to fail the workflow in the end if atleast one of the modules fails this check.

Reason of rejection:

- The oppia-android project will soon migrate from having a large module-level target so //:app will be deleted, hence this approach won’t work for that case.

Third-party Libraries*

This project does not require the addition of any new third party libraries. However, it will add a linting tool for the XML files (in the static checks).

Testing Approach

We will add some automated tests for the scripts to ensure that they are working correctly. We have to verify that the script output is matching with what we expected against a test file. We will need to do this for every script check present. Some tests . PRs might also be needed for this. We also have to do some manual testing to demonstrate that the CI checks work as expected and the script checks are working correctly and are not reporting any faulty error.

Milestones

Milestone 1

Key Objective: The following checks now run during the Github Actions workflow:

- XML files linting support improved by the integration of an XML Linter.
- Support for generic Regex pattern matching against file names and file contents introduced.
- Integrate script check to verify test files present for the production files.
- Introduced check for verifying that the activities are defined with accessibility labels.

No.	**Description of PR**	**Prereq PR numbers**	**Target date for PR submission**	**Target date for PR to be merged**
	Become familiar with the scripting syntax of the scripting language finalized for the project (Kotlin/Bash) [During the Community Bonding Period]	None	May 17, 2021	June 7, 2021
1.1	PR that fixes all the lint errors reported by the tool (by first running the lint checks locally)	None	June 10, 2021	June 13, 2021
1.2	Integrate XML file linter tool to the project.	1.1	June 13, 2021	June 16, 2021
1.3	Introduce support for generic Regex pattern matching against file names and file contents	None	June 23, 2021	June 27, 2021
1.4	Integrate script check to verify that test files are present for the production files	1.3	July 01, 2021	July 05, 2021
1.5	Integrate script check to verify the activities are defined with accessibility labels.	None	July 05, 2021	July 08, 2021

Milestone 2

Key Objective: The following checks now run during the Github Actions workflow:

- Added check to ensure that all the TODOs correspond to current open issues on Github and all those issues have been reopened whose TODOs were not addressed and removed/updated those TODOs.
- For Issue status changes, verification added to check all the corresponding TODOs are addressed. Failure results in the issue being reopened with a comment mentioning all unaddressed TODOs.
- Introduced KDoc enforcement
- Script check added to ensure activities/fragments/views can't be used outside of the app module.
- All the documentation changes are done providing the information of the new checks added in the development workflow

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
2.1	Introduce script check to verify all TODOs correspond to open issues	None	July 17, 2021	July 21, 2021
2.2	Reopen all the issues whose TODOs are not addressed or remove/update those TODOs	2.1	July 22, 2021	July 25, 2021
2.3	Introduce check to verify TODOs addressed for closed issue	2.1 and 2.2	July 26, 2021	July 29,, 2021
2.4	Introduce script check to ensure activities/fragments/views can't be used outside of the app module	1.3	August 01, 2021	August 05, 2021
2.5	Introduce KDoc enforcement	1.3	August 07, 2021	August 10, 2021
2.6	Make all the documentation changes	All PRs	August 10, 2021	August 13, 2021

Optional Sections

Additional Project-Specific Considerations

Privacy

No this project does not collect any new user data or change how user data is collected.

Security

No, this feature does not provide any new opportunities for users to gain unauthorized access to user data or otherwise impact other users' experience on the site in a negative way.

Accessibility (if user-facing)

No, this project doesn't have any user facing components.

Documentation Changes*

Yes, we will be requiring some documentation changes for this project. We have to update the static checks readme to include the XML Linter and the custom checks developed, and also info about the new TODO verification system integration.

Ethics*

This project makes the process of code reviewing faster, so that the developers can save time spent in fixing the lint and other similar errors during code reviews, and instead utilize it on reviewing the actual feature implementation or bug fix.

Future Work

There are many File Content checks that could be added in the future. We can plan to research more on them and then work on integrating them in the static analysis checks. Also, it was a great experience contributing to the oppia-android project and I would love to continue contributing to the project as much as I could and also provide my support to the team in any way possible.