

Google Summer of Code 2022

Improve line and branch coverage for the backend and frontend

Anshuman Maurya

Section 1: About You

What project are you applying for?

Achieve 100% Per-File Branch and Line Coverage for the Frontend and the Backend.

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

I learned about Oppia from one of my college seniors who volunteered at Oppia's codebase. When I gave it some time, I understood the problem Oppia is trying to solve. It made me realize how lucky and privileged I am to attain a quality education and teaching. The best I could do now is to give back to society and help others who lack the resources.

More than 60 million children worldwide lack the resources to attain primary education. I belong to India, where more than 30 million children aged 6-14 cannot attend school, and almost half of the girls of this age are illiterate. About half the habitation lacks primary schooling facilities [\[src\]](#). If we cannot bring these children to school, we can take schools to them in the form of online platforms like Oppia. This could significantly impact the world, and I would consider myself fortunate if I am a small part of it. At Oppia, I am happy to be part of a community that shares an equal enthusiasm and motivation towards the goal we are trying to achieve. Every bug I fix and every feature I add seems to be a small step towards providing quality education to all, which keeps me motivated.

Oppia has a large codebase with Angular JS/Angular 2+ at its frontend and Python at its backend. It is crucial to test the code to avoid any bug or error popping up in production. That's why I chose to work on the project that aims to achieve 100% Per-File Branch and Line Coverage for the Frontend and the Backend. A fully covered codebase would significantly reduce the chances of any regressions. I already have experience in testing, and I have covered 25+ files in the backend and frontend of Oppia's codebase. The project seems to be an excellent opportunity to make myself proficient in testing code.

Prior experience

I started my development journey in the past year and have learned quite a few technologies since then. I am familiar with Typescript and Python and various frontend and backend frameworks like Vue, Angular, and Django.

I have been contributing to Oppia for the past 3 to 4 months and merged about 20+ PRs gaining familiarity with the codebase. I am a part of the Automated QA team and the Learner and Creator Experience team. I am currently working on the User checkpoints project under the mentorship of Sean Lip. The project aims to achieve exploration checkpointing on the learner's side so that no learner may lose his progress if he has to quit an exploration in between. I was responsible for the entire backend changes to be made regarding the storage of checkpoint progress of a logged-in user.

My submitted PRs include:

1. [#15213](#) - User checkpoints: Backend changes for logged in users
2. [#14876](#) - Strip invalid tags and attributes from SVGs instead of showing a warning
3. [#14641](#) - Increased backend test coverage of some files to 100%
4. [#14729](#) - Increased frontend test coverage of some files to 100%
5. [#14506](#) - Changed deprecated python methods

The complete list of my merged PRs to Oppia can be found [here](#).

Along with this, I also worked on many group and personal projects which can be viewed on my [GitHub](#) profile.

Project size

The project size is medium(~175 hours).

Project timeframe

June 20 - September 19

Contact info and timezone(s)

Email: anshumanmaurya111@gmail.com

Phone no: (+91) 8318904105

Preferred mode of communication: Google chat, Gmail, Gmeet, Discord

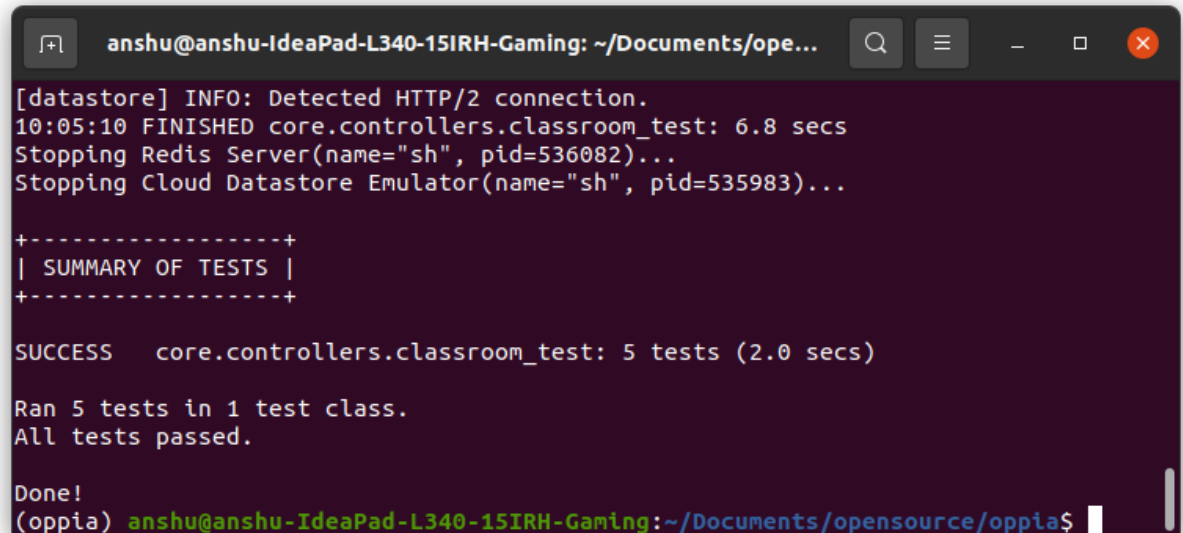
Timezone: Asia/Kolkata (IST) (+5:30 GMT)

Time commitment

I would work for 20hrs per week to complete the project within due time.

Essential Prerequisites

- I am able to run a single backend test target on my machine.



```
anshu@anshu-IdeaPad-L340-15IRH-Gaming: ~/Documents/ope...
[datastore] INFO: Detected HTTP/2 connection.
10:05:10 FINISHED core.controllers.classroom_test: 6.8 secs
Stopping Redis Server(name="sh", pid=536082)...
Stopping Cloud Datastore Emulator(name="sh", pid=535983)...

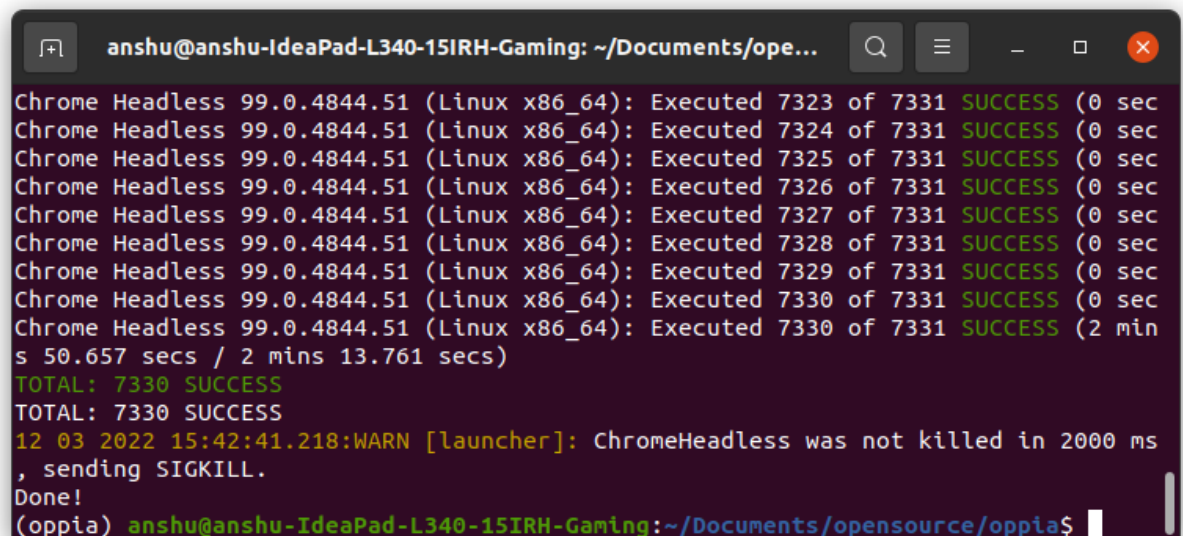
+-----+
| SUMMARY OF TESTS |
+-----+

SUCCESS   core.controllers.classroom_test: 5 tests (2.0 secs)

Ran 5 tests in 1 test class.
All tests passed.

Done!
(oppia) anshu@anshu-IdeaPad-L340-15IRH-Gaming: ~/Documents/opensource/oppia$
```

- I am able to run all the frontend tests at once on my machine.



```
anshu@anshu-IdeaPad-L340-15IRH-Gaming: ~/Documents/ope...
Chrome Headless 99.0.4844.51 (Linux x86_64): Executed 7323 of 7331 SUCCESS (0 sec
Chrome Headless 99.0.4844.51 (Linux x86_64): Executed 7324 of 7331 SUCCESS (0 sec
Chrome Headless 99.0.4844.51 (Linux x86_64): Executed 7325 of 7331 SUCCESS (0 sec
Chrome Headless 99.0.4844.51 (Linux x86_64): Executed 7326 of 7331 SUCCESS (0 sec
Chrome Headless 99.0.4844.51 (Linux x86_64): Executed 7327 of 7331 SUCCESS (0 sec
Chrome Headless 99.0.4844.51 (Linux x86_64): Executed 7328 of 7331 SUCCESS (0 sec
Chrome Headless 99.0.4844.51 (Linux x86_64): Executed 7329 of 7331 SUCCESS (0 sec
Chrome Headless 99.0.4844.51 (Linux x86_64): Executed 7330 of 7331 SUCCESS (0 sec
Chrome Headless 99.0.4844.51 (Linux x86_64): Executed 7330 of 7331 SUCCESS (2 min
s 50.657 secs / 2 mins 13.761 secs)
TOTAL: 7330 SUCCESS
TOTAL: 7330 SUCCESS
12 03 2022 15:42:41.218:WARN [launcher]: ChromeHeadless was not killed in 2000 ms
, sending SIGKILL.
Done!
(oppia) anshu@anshu-IdeaPad-L340-15IRH-Gaming: ~/Documents/opensource/oppia$
```

- I am able to run one suite of e2e tests on my machine.

```
anshu@anshu-IdeaPad-L340-15IRH-Gaming: ~/Documents/ope...
Blog dashboard functionality
    ♦♦♦ should check user profile is visible on both blog dashboard and editor, c
create, edit and delete a blog post from blog dashboard
[16:57:22] W/element - more than one element found for locator By(css selector, .
toast-success) - the first result will be used
    ♦♦♦ should create, publish, and delete the published blog post from dashboar
d.
[16:57:38] W/element - more than one element found for locator By(css selector, .
toast-success) - the first result will be used
    ♦♦♦ should create, publish, check for thumbnail uploading error, unpublish a
nd delete the blog post
[16:57:48] W/element - more than one element found for locator By(css selector, .
toast-success) - the first result will be used
[16:58:09] W/element - more than one element found for locator By(css selector, .
toast-success) - the first result will be used
[2022-03-12 16:58:12 +0530] [597798] [INFO] Starting gunicorn 20.1.0
[2022-03-12 16:58:12 +0530] [597798] [INFO] Listening at: http://0.0.0.0:24994 (5
97798)
[2022-03-12 16:58:12 +0530] [597798] [INFO] Using worker: sync
[2022-03-12 16:58:12 +0530] [597802] [INFO] Booting worker with pid: 597802
    ♦♦♦ should create multiple blog posts both published and drafts and check fo
r navigation through list view

4 specs, 0 failures
Finished in 119.501 seconds

Executed 4 of 4 specs SUCCESS in 2 mins.
```

Other summer obligations

I do not have any such obligations this summer and I'll only be applying to Oppia for GSoC.

Communication channels

I intend to meet with my mentor 2 times a week to discuss progress and doubts clarification.
[On Gmeet, Discord, or any other platform].

The meeting time and the number can be flexible as per the mentor's convenience.

Section 2: Proposal Details

Problem Statement

Link to PRD (or N/A if there isn't one)	Achieve 100% backend line and branch coverage. Achieve 100% per-file backend line and branch coverage except for the files in core/domain/. Achieve 100% frontend line coverage except for the files that will be removed by Angular migration.		
Target Audience	Oppia developers		
Core User Need	The codebase should be completely covered, with 100% <i>per-file</i> branch and line coverage for both frontend and backend to prevent regressions. Here 'per-file' means that every non-test file is fully covered by its associated test file.		
What goals do we want the solution to achieve?	The timeframe of the project restricts us to cover only a part of our initial goal. Given below is the compilation of what we are going to achieve in this project:		
		Backend files	Frontend files
	Full line coverage (this also includes <u>all</u> files without associated test files)	Yes – also need to maintain 100% line coverage going forward, and make sure every file is fully tested.	Completed fully in this GSoC project – every line of frontend code (except the files in core/templates/tests) will be covered by tests, and this will be maintained going forward.
	Full branch coverage	Yes (and we will maintain 100% branch coverage going forward).	No (There are problems in accurately calculating branch coverage due to vague Karma coverage reports issue #15544)
	Per-file line coverage	No, but we have prevented new files from coming in which don't have per-file line coverage. (This is the	No. The only way to calculate per-file frontend coverage

		only case with a denylist, and the infrastructure for this already exists.)	includes avoiding use of combined-spec file. However, that does impose memory-allocation and increase in runtime of the coverage checks.
	Per-file branch coverage	No, but we will prevent new files from coming in which don't have per-file branch coverage. (Only the files in the above mentioned denylist will also be lacking per-file branch coverage – for the rest of the files I'll be covering them on a per-file basis.)	No. The only way to calculate per-file frontend branch coverage includes avoiding use of combined-spec file. However, that does impose memory-allocation and increase in runtime of the coverage checks.

Section 2.1: WHAT

Key User Stories and Tasks

#	Title	User Story Description (role, goal, motivation) <i>"As a ..., I need ..., so that"</i>	Priority ¹	List of tasks needed to achieve the goal (this is the "User Journey")	Links to mocks / prototypes, and/or PRD sections that spec out additional requirements.
1	Increase backend Line coverage to be 100%, including files that currently lack an associated test file.	As a developer in Oppia's team, I want 100% branch and line coverage of the backend.	Must have	<div>Cover all remaining backend files increasing line coverage to 100%</div> <div>Add checks to ensure that every backend file has an associated test file.</div>	After we achieve 100% backend line coverage, if a developer tries to push a file whose all lines are not fully covered, [this] message stops him from pushing the code.

¹ Use the **MoSCow** system ("Must have", "Should have", "Could have"). You can read more [here](#).

2	Increase backend Branch coverage to be 100%		Must have	Use the coverage.py tool to list out the files with incomplete branch coverage in `backend_test_incomplete_branch_coverage.txt`.	After we achieve 100% backend branch coverage, if a developer tries to push a file whose all branches are not fully covered, [this] message stops him from pushing the code.
				Cover all the branches of the files listed in `backend_test_incomplete_branch_coverage.txt` and remove the file once the goal of 100% per-file branch coverage is achieved.	
3	Increase per-file backend line coverage of files except in core/domain/ to 100%	As a developer in Oppia's team, I want 100% backend per-file line coverage, i.e, all non-test files should be covered by its associated test file. [However the scope of this projects leaves out the files in core/domain/]	Must have	Achieve 100% per-file backend line coverage except for the files in core/domain/.	After the project, if a developer tries to push a file outside core/domain/ to bring per-file coverage below 100%, [this] message stops him from pushing the code.
4	Achieve 100% line coverage of all frontend files except those files which will be removed by the Angular Migration.	As a developer in Oppia's team, I want all existing frontend code to be properly tested with all lines covered to avoid any bugs popping out at a later stage.	Must have	Achieve 100% line coverage of all frontend files except those in core/templates/tests, as well as the two individual files unit-test-utils.ts and schema-based-list-viewer.directive.ts (see #10427), since those will be removed by the Angular Migration.	After we achieve 100% frontend line coverage, if a developer tries to push a frontend file whose all lines are not fully covered, [this] message stops him from pushing the code.

Technical Requirements




Additions/Changes to Web Server Endpoint Contracts

None

Calls to Web Server Endpoints

None

UI Screens/Components

#	ID	Description of new UI component	i18n required ?	Mock/spec links	A11y requirements
1.	Terminal logs for backend line coverage checks	After we achieve 100% backend line coverage, if a developer tries to push a file whose all lines are not fully covered, a message stops him from pushing the code.	No	 Backend_line....	No
2.	Terminal logs for backend branch coverage checks	After we achieve 100% backend branch coverage, if a developer tries to push a file whose all branches are not fully covered, a message stops him from pushing the code.	No	 Backend_bra...	No
4.	Terminal logs for frontend line coverage checks	After we achieve 100% frontend line coverage, if a developer tries to push a file whose all lines are not fully covered, a message stops him from pushing the code.	No	 frontend_line....	No

Data Handling and Privacy

The project neither collects new user data nor changes the way how user data is collected. This is because the test files simply test the working of existing code and do not participate in the working of the code itself.

Other Requirements

Some frontend files which are still in Angular JS need to be migrated to Angular 2+ first before their tests are written.

The Angular Migration team aims to cover these files before the commencement of GSoC. However, if Angular Migration is running behind schedule, I will migrate these files first before writing tests for them.

Section 2.2: HOW

Existing Status Quo

BACKEND

We previously thought that the backend overall line coverage is 100%, but a deeper analysis showed that some files without associated test files were sneaking behind our coverage checks.

The list of backend files which do not have an associated test files is as follows:

	Files which are not covered at all
	Files which are covered by some other test files
	Files which do not need to be covered (Data files for testing)

Backend Files lacking associated test files

```
scripts.run_frontend_tests.py
scripts.create_expression_parser.py
scripts.run_custom_eslint_tests.py
scripts.run_backend_tests.py
scripts.run_lighthouse_tests.py
scripts.run_portserver.py
scripts.run_presubmit_checks.py
scripts.run_tests.py
scripts.start.py
setup.py
```

```
core.domain.caching_domain.py
core.domain.classroom_domain.py
core.domain.cron_services.py
```

core.domain.takeout_domain.py
core.platform.datastore.cloud_datastore_services.py
core.handler_schema_constants.py
core.domain.change_domain.py
extensions.visualizations.models.py
core.platform.transactions.cloud_transaction_services.py
core.feconf.py
core.domain.app_feedback_report_constants.py
extensions.actions.AnswerSubmit.AnswerSubmit.py
extensions.actions.ExplorationQuit.ExplorationQuit.py
extensions.actions.ExplorationStart.ExplorationStart.py
extensions.interactions.AlgebraicExpressionInput.AlgebraicExpressionInput.py
extensions.interactions.CodeRepl.CodeRepl.py
extensions.interactions.Continue.Continue.py
extensions.interactions.DragAndDropSortInput.DragAndDropSortInput.py
extensions.interactions.EndExploration.EndExploration.py
extensions.interactions.FractionInput.FractionInput.py
extensions.interactions.GraphInput.GraphInput.py
extensions.interactions.ImageClickInput.ImageClickInput.py
extensions.interactions.InteractiveMap.InteractiveMap.py
extensions.interactions.ItemSelectionInput.ItemSelectionInput.py
extensions.interactions.MathEquationInput.MathEquationInput.py
extensions.interactions.MultipleChoiceInput.MultipleChoiceInput.py
extensions.interactions.MusicNotesInput.MusicNotesInput.py
extensions.interactions.NumberWithUnits.NumberWithUnits.py
extensions.interactions.NumericExpressionInput.NumericExpressionInput.py
extensions.interactions.NumericInput.NumericInput.py
extensions.interactions.PencilCodeEditor.PencilCodeEditor.py
extensions.interactions.RatioExpressionInput.RatioExpressionInput.py
extensions.interactions.SetInput.SetInput.py
extensions.interactions.TextInput.TextInput.py
extensions.issues.CyclicStateTransitions.CyclicStateTransitions.py
extensions.issues.EarlyQuit.EarlyQuit.py
extensions.issues.MultipleIncorrectSubmissions.MultipleIncorrectSubmissions.py

mypy_imports.py
proto_files.text_classifier_pb2.py
proto_files.training_job_response_payload_pb2.py
scripts.linters.warranted_angular_security_bypasses.py

scripts.linters.test_files.invalid_annotations.py
scripts.linters.test_files.invalid_author.py
scripts.linters.test_files.invalid_copyright.py
scripts.linters.test_files.invalid_docstring.py
scripts.linters.test_files.invalid_import_order.py
scripts.linters.test_files.invalid_merge_conflict.py
scripts.linters.test_files.invalid_ndb.py
scripts.linters.test_files.invalid_no_newline.py
scripts.linters.test_files.invalid_pycodestyle_error.py
scripts.linters.test_files.invalid_pylint_id.py
scripts.linters.test_files.invalid_python_three.py
scripts.linters.test_files.invalid_request.py
scripts.linters.test_files.invalid_tabs.py
scripts.linters.test_files.invalid_todo.py
scripts.linters.test_files.invalid_urlopen.py
scripts.linters.test_files.valid.py
scripts.linters.test_files.valid_py_ignore_pragma.py
core.tests.build_sources.extensions.CodeRepl.py
core.tests.build_sources.extensions.DragAndDropSortInput.py
core.tests.data.failing_tests.py
core.tests.data.image_constants.py
core.tests.data.unicode_and_str_handler.py

FILES LACKING 100% PER-FILE LINE COVERAGE:

As mentioned in [scripts/backend_tests_incomplete_coverage.txt](#), following backend files lack per-file line coverage:.

Backend files lacking 100% per-file line coverage	Uncovered lines
core.storage.user.gae_models_test	0
core.storage.audit.gae_models_test	0
core.storage.classifier.gae_models_test	0
extensions.interactions.base_test	1
core.controllers.tasks_test	1
core.domain.value_generators_domain_test	1
scripts.linters.js_ts_linter_test	1
scripts.install_backend_python_libs_test	2
core.storage.job.gae_models_test	2
core.domain.image_validation_services_test	2
core.controllers.feedback_test	3
scripts.linters.pylint_extensions_test	3
core.domain.auth_services_test	4
core.domain.blog_domain_test	5
main_test	5
core.domain.collection_services_test	6
core.domain.subtopic_page_domain_test	9
core.storage.topic.gae_models_test	11
core.schema_utils_test	11
core.storage.skill.gae_models_test	12
core.storage.suggestion.gae_models_test	12
core.storage.email.gae_models_test	13
core.domain.feedback_domain_test	15
core.storage.feedback.gae_models_test	16
core.domain.user_domain_test	16
core.domain.app_feedback_report_domain_test	17
core.storage.app_feedback_report.gae_models_test	18
core.utils_test	21
core.domain.stats_services_test	24
core.storage.collection.gae_models_test	24
core.domain.suggestion_registry_test	28
core.domain.opportunity_services_test	28

core.storage.exploration.gae_models_test	28
core.controllers.domain_objects_validator_test	33
core.controllers.voice_artist_test	33
core.domain.learner_progress_services_test	33
scripts.docstrings_checker_test	34
core.storage.base_model.gae_models_test	42
core.domain.rights_manager_test	43
core.domain.exp_domain_test	60
core.domain.state_domain_test	68
core.domain.topic_domain_test	81
core.domain.story_domain_test	86
core.domain.user_services_test	86

Note: The lines above have been calculated by the coverage tool.

FILES LACKING 100% PER-FILE BRANCH COVERAGE:

To list out the files lacking 100% branch coverage, some changes were to be made in the `run_backend_test.py` file.

- The command for the coverage tool to execute, was modified to include the `'branch'` flag:

```
exc_list = [
    sys.executable, COVERAGE_MODULE_PATH, 'run',
    '--branch', TEST_RUNNER_PATH, test_target_flag
]
```

- Total coverage for a file was calculated from the process output as follows:

```
coverage_result = re.search(
    r'TOTAL\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(?P<total>\d+)\s+',
    process.stdout)
coverage = float(coverage_result.group('total'))
```

Backend files lacking 100% per-file branch coverage	Uncovered Branches
extensions.answer_summarizers.models.py	1
extensions.objects.models.objects.py	2

extensions.answer_summarizers.models.py	1
extensions.objects.models.objects.py	2
core.storage.blog.gae_models.py	1
core.storage.question.gae_models.py	1
core.storage.statistics.gae_models.py	1
core.controllers.creator_dashboard.py	1
core.controllers.payload_validator.py	1
core.controllers.base.py	1
core.controllers.library.py	2
core.controllers.skill_editor.py	1
core.controllers.questions_list.py	1
core.controllers.practice_sessions.py	1
core.controllers.topic_editor.py	4
core.controllers.learner_playlist.py	2
core.controllers.classroom.py	1
core.controllers.story_editor.py	1
core.controllers.blog_homepage.py	1
core.controllers.profile.py	5
core.controllers.topics_and_skills_dashboard.py	2
core.controllers.blog_admin.py	1
core.controllers.admin.py	4
core.controllers.subtopic_viewer.py	2
core.controllers.contributor_dashboard_admin.py	2
core.platform.bulk_email.mailchimp_bulk_email_services.py	3
core.platform.taskqueue.cloud_taskqueue_services.py	3
core.platform.taskqueue.cloud_tasks_emulator.py	2
core.platform.email.dev_mode_email_services.py	1
core.platform.auth.firebase_auth_services.py	2
core.controllers.contributor_dashboard.py	1
core.jobs.batch_jobs.math_interactions_audit_jobs.py	1
core.jobs.types.base_validation_errors.py	1
core.jobs.transforms.validation.base_validation.py	5
core.controllers.suggestion.py	2
core.controllers.reader.py	9

core.controllers.editor.py	11
core.domain.rights_domain.py	1
core.domain.platform_parameter_domain.py	14
core.domain.collection_domain.py	3
core.domain.user_query_domain.py	2
core.domain.feedback_services.py	4
core.domain.app_feedback_report_services.py	4
core.domain.question_services.py	3
core.domain.recommendations_services.py	3
core.domain.action_registry.py	2
core.domain.user_query_services.py	2
core.domain.email_manager.py	11
core.domain.playthrough_issue_registry.py	2
core.domain.visualization_registry.py	1
core.domain.param_domain.py	2
core.domain.config_domain.py	1
core.domain.question_domain.py	28
core.domain.summary_services.py	4
core.domain.learner_playlist_services.py	4
core.domain.calculation_registry.py	1
core.controllers.acl_decorators.py	8
core.domain.skill_domain.py	4
core.domain.translation_domain.py	1
core.domain.skill_services.py	13
core.domain.learner_goals_services.py	1
core.domain.topic_services.py	6
core.domain.event_services.py	1
core.domain.beam_job_services.py	1
core.domain.html_cleaner.py	2
core.domain.blog_services.py	1
core.domain.fs_services.py	4
core.domain.draft_upgrade_services.py	12
core.domain.interaction_registry.py	1
core.domain.search_services.py	1

core.domain.stats_domain.py	3
core.domain.html_validation_service.py	6
core.domain.platform_parameter_registry.py	1
core.domain.wipeout_domain.py	2
core.domain.rte_component_registry.py	2
core.domain.rules_registry.py	1
core.domain.classifier_services.py	4
core.domain.voiceover_services.py	2
core.domain.story_fetchers.py	2
scripts.linters.html_linter.py	5
scripts.linters.other_files_linter.py	3
scripts.linters.general_purpose_linter.py	3
scripts.release_scripts.repo_specific_changes_fetcher.py	3
scripts.create_topological_sort_of_all_services.py	2
scripts.clean.py	1
scripts.check_frontend_test_coverage.py	1
scripts.pre_commit_hook.py	1
scripts.install_third_party.py	6
scripts.rtl_css.py	1
scripts.servers.py	1
scripts.setup.py	1
scripts.build.py	3
scripts.concurrent_task_utils.py	1
scripts.install_third_party_libs.py	2
scripts.check_if_pr_is_low_risk.py	1
scripts.run_e2e_tests.py	2
scripts.common.py	6
scripts.pre_push_hook.py	6
core.domain.wipeout_service.py	5
core.domain.story_services.py	6

Note: The branches above have been calculated by the coverage tool.

Along with the above stated files, the files in the backend per-file line coverage denylist also lack per-file branch coverage.

FRONTEND

FILES LACKING 100% LINE COVERAGE:

The list NOT_FULLY_COVERED_FILENAMES in [scripts/check_frontend_test_coverage.py](#), mentions the files that lack 100% line coverage. They are listed below:

Frontend files lacking 100% line coverage	Uncovered lines
angular-html-bind.directive.ts	1
App.ts	12
Base.ts	15
ck-editor-4-rte.component.ts	146
ck-editor-4-widgets.initializer.ts	65
exploration-states.service.ts	33
expression-interpolation.service.ts	2
google-analytics.initializer.ts	7
learner-answer-info.service.ts	5
mathjax-bind.directive.ts	8
object-editor.directive.ts	29
oppia-interactive-music-notes-input.component.ts	243
oppia-interactive-pencil-code-editor.component.ts	61
oppia-root.directive.ts	13
parameterize-rule-description.filter.ts	94
python-program.tokenizer.ts	59
question-update.service.ts	6
rule-type-selector.directive.ts	26
select2-dropdown.directive.ts	27
translation-file-hash-loader-backend-api.service.ts	1
truncate-input-based-on-interaction-answer-type.filter.ts	3
voiceover-recording.service.ts	26

Note: The lines above have been calculated by Karma.

FILES LACKING 100% BRANCH COVERAGE:

We fetch information about frontend coverage from 'oppia/./karma_coverage_reports/lcov.info' file. To list out the files lacking 100% branch coverage, some changes were to be made in the 'check_frontend_test_coverage.py' file.

- Modified the 'LcovStanzaRelevantLines' class to include two new properties (namely total_branches and covered_branches):

```
match = re.search(r'BRF:(\d+)\n', stanza)
if match is None:
    raise Exception(
        'It wasn\'t possible to get the total branches of {} file.'
        'It\'s not possible to diff the test coverage correctly.'
        .format(file_name))
self.total_branches = int(match.group(1))

match = re.search(r'BRH:(\d+)\n', stanza)
if match is None:
    raise Exception(
        'It wasn\'t possible to get the covered branches of {} file.'
        'It\'s not possible to diff the test coverage correctly.'
        .format(file_name))
self.covered_branches = int(match.group(1))
```

- Modified the 'check_coverage_changes()' method to calculate 'total_branches' and 'covered_branches' along with 'total_lines' and 'covered_lines'. If the total branches and covered branches differed in a file, the filename was added in FILES_WITH_INCOMPLETE_BRANCH_COVERAGE list.

```
for stanza in stanzas:
    file_name = stanza.file_name
    total_lines = stanza.total_lines
    covered_lines = stanza.covered_lines
    total_branches = stanza.total_branches
    covered_branches = stanza.covered_branches
    if total_branches != covered_branches:
        FILES_WITH_INCOMPLETE_BRANCH_COVERAGE.append(file_name)
```

However, the list of files generated by the above steps is faulty, due to vague Karma coverage reports. [\[Link to the debugging doc\]](#)
A [tracker issue](#) has been created regarding the same.

Solution Overview

The goal of the project is to achieve the following:

INFRASTRUCTURE CHANGES

1. Ensure that all the backend files have an associated test file. This mainly includes modifying the 'pre_push_hook' and running checks on CI to ensure that files named as '<name>.py' also have an '<name>_test.py' file.
Use a denylist to exclude files in **core/tests/data**, **scripts/linters/test_files**, and **core/tests/build_sources**, since these folders contain files that are just used as inputs for our tests.
2. Update the 'run_backend_tests' to calculate per-file branch coverage alongside calculating per-file line coverage. This includes changing
 - a. The command to be executed-

```
exc_list = [  
    sys.executable, COVERAGE_MODULE_PATH, 'run',  
    '--branch', TEST_RUNNER_PATH, test_target_flag  
]
```

- b. The way coverage is calculated-

```
coverage_result = re.search(  
    r'TOTAL\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(?P<total>\d+)\s+',  
    process.stdout)  
coverage = float(coverage_result.group('total'))
```

This will ensure both per-file line and branch coverage is calculated.

For example: On running

```
python -m scripts.run_backend_tests  
--test_target=core.domain.user_services_test  
--generate_coverage_report  
following coverage report is shown:
```

```

+-----+
| SUMMARY OF TESTS |
+-----+

INCOMPLETE BRANCH COVERAGE (85.0%): core/domain/user_services.py
Name           Stmts    Miss Branch BrPart  Cover    Missing
-----
core/domain/user_services.py    660     86    222     22    85%  118, 133-137, 169, 515
->511, 593, 675-678, 704, 754, 1043->1042, 1065, 1102-1104, 1201-1203, 1214-1217, 1229-12
32, 1275-1281, 1336-1352, 1362-1365, 1402->exit, 1415->exit, 1517, 1597-1611, 1660, 1663,
1688-1696, 1710, 1711->exit, 1728, 1761-1764, 1808-1813, 1891, 1914, 1926, 1938, 2133-21
35, 2147->exit, 2179, 2190, 2194, 2213-2214, 2228, 2277-2279
-----
TOTAL                           660     86    222     22    85%

```

Here the value under 'Miss' heading signifies the number of lines that are not fully covered. The value under 'BrPart' signifies the branches that are not fully covered and those branches are mentioned under the 'Missing' heading. The overall coverage (85% in the above example) is 100% only when both the line and branch coverage is 100%.

TESTING

1. Achieve 100% backend line coverage. Ensure that every backend file has an associated test file so that coverage checks are guaranteed to be triggered for all files.
 - a. Add associated test files and cover these files on a per-file basis. (These files were sneaking past our coverage checks).

scripts.run_frontend_tests.py	105
scripts.create_expression_parser.py	21
scripts.run_custom_eslint_tests.py	45
scripts.run_backend_tests.py	470
scripts.run_lighthouse_tests.py	145
scripts.run_portserver.py	477
scripts.run_presubmit_checks.py	51
scripts.run_tests.py	26
scripts.start.py	122
setup.py	16
TOTAL	1478 lines

Note: The lines above have been estimated after ignoring the comments and imports at the top of each file.

- b. Add trivial test files for these files and add them to the per-file coverage denylist. (These files are already tested just not on a per-file basis.)

core.domain.caching_domain.py
core.domain.classroom_domain.py
core.domain.cron_services.py
core.domain.takeout_domain.py
core.platform.datastore.cloud_datastore_services.py
core.handler_schema_constants.py
core.domain.change_domain.py
extensions.visualizations.models.py
core.platform.transactions.cloud_transaction_services.py
core.feconf.py
core.domain.app_feedback_report_constants.py
extensions.actions.AnswerSubmit.AnswerSubmit.py
extensions.actions.ExplorationQuit.ExplorationQuit.py
extensions.actions.ExplorationStart.ExplorationStart.py
extensions.interactions.AlgebraicExpressionInput.AlgebraicExpressionInput.py
extensions.interactions.CodeRepl.CodeRepl.py
extensions.interactions.Continue.Continue.py
extensions.interactions.DragAndDropSortInput.DragAndDropSortInput.py
extensions.interactions.EndExploration.EndExploration.py
extensions.interactions.FractionInput.FractionInput.py
extensions.interactions.GraphInput.GraphInput.py
extensions.interactions.ImageClickInput.ImageClickInput.py
extensions.interactions.InteractiveMap.InteractiveMap.py
extensions.interactions.ItemSelectionInput.ItemSelectionInput.py
extensions.interactions.MathEquationInput.MathEquationInput.py
extensions.interactions.MultipleChoiceInput.MultipleChoiceInput.py
extensions.interactions.MusicNotesInput.MusicNotesInput.py
extensions.interactions.NumberWithUnits.NumberWithUnits.py
extensions.interactions.NumericExpressionInput.NumericExpressionInput.py
extensions.interactions.NumericInput.NumericInput.py
extensions.interactions.PencilCodeEditor.PencilCodeEditor.py
extensions.interactions.RatioExpressionInput.RatioExpressionInput.py

extensions.interactions.SetInput.SetInput.py
extensions.interactions.TextInput.TextInput.py
extensions.issues.CyclicStateTransitions.CyclicStateTransitions.py
extensions.issues.EarlyQuit.EarlyQuit.py
extensions.issues.MultipleIncorrectSubmissions.MultipleIncorrectSubmissions.py
mypy_imports.py
proto_files.text_classifier_pb2.py
proto_files.training_job_response_payload_pb2.py
scripts.linters.warranted_angular_security_bypasses.py

2. Achieve 100% per-file backend line coverage except for the files in core/domain/. This includes covering these files:

core.storage.user.gae_models_test	0	
core.storage.audit.gae_models_test	0	
core.storage.classifier.gae_models_test	0	
extensions.interactions.AlgebraicExpressionInput.AlgebraicExpressionInput.py	0	Trivial
extensions.interactions.ItemSelectionInput.ItemSelectionInput.py	0	Trivial
extensions.interactions.MathEquationInput.MathEquationInput.py	0	Trivial
extensions.interactions.NumericExpressionInput.NumericExpressionInput.py	0	Trivial
extensions.interactions.EndExploration.EndExploration.py	0	Trivial
extensions.interactions.Continue.Continue.py	0	Trivial
extensions.interactions.GraphInput.GraphInput.py	0	Trivial
extensions.interactions.ImageClickInput.ImageClickInput.py	0	Trivial
extensions.interactions.MultipleChoiceInput.MultipleChoiceInput.py	0	Trivial
extensions.interactions.MusicNotesInput.MusicNotesInput.py	0	Trivial
extensions.interactions.PencilCodeEditor.PencilCodeEditor.py	0	Trivial
extensions.interactions.SetInput.SetInput.py	0	Trivial
extensions.interactions.CodeRepl.CodeRepl.py	0	Trivial
extensions.interactions.DragAndDropSortInput.DragAndDropSortInput.py	0	Trivial
extensions.interactions.FractionInput.FractionInput.py	0	Trivial
extensions.interactions.InteractiveMap.InteractiveMap.py	0	Trivial
extensions.interactions.NumberWithUnits.NumberWithUnits.py	0	Trivial
extensions.interactions.NumericInput.NumericInput.py	0	Trivial
extensions.interactions.RatioExpressionInput.RatioExpressionInput.py	0	Trivial

extensions.interactions.TextInput.TextInput.py	0	Trivial
core.feconf.py	0	Trivial
core.domain.app_feedback_report_constants.py	0	Trivial
extensions.actions.AnswerSubmit.AnswerSubmit.py	0	Trivial
extensions.actions.ExplorationQuit.ExplorationQuit.py	0	Trivial
extensions.actions.ExplorationStart.ExplorationStart.py	0	Trivial
extensions.issues.CyclicStateTransitions.CyclicStateTransitions.py	0	Trivial
extensions.issues.EarlyQuit.EarlyQuit.py	0	Trivial
extensions.issues.MultipleIncorrectSubmissions.MultipleIncorrectSubmissions.py	0	Trivial
mypy_imports.py	0	Trivial
extensions.interactions.base_test	1	
core.controllers.tasks_test	1	
scripts.linters.js_ts_linter_test	1	
scripts.install_backend_python_libs_test	2	
core.storage.job.gae_models_test	2	
core.controllers.feedback_test	3	
scripts.linters.pylint_extensions_test	3	
scripts.linters.warranted_angular_security_bypasses.py	4	
main_test	5	
core.handler_schema_constants.py	5	
core.storage.topic.gae_models_test	11	
core.schema_utils_test	11	
core.platform.transactions.cloud_transaction_services.py	11	
core.storage.skill.gae_models_test	12	
core.storage.suggestion.gae_models_test	12	
proto_files.text_classifier_pb2.py	12	
core.storage.email.gae_models_test	13	
core.storage.feedback.gae_models_test	16	
core.storage.app_feedback_report.gae_models_test	18	
proto_files.training_job_response_payload_pb2.py	20	
core.utils_test	21	
core.storage.collection.gae_models_test	24	
core.storage.exploration.gae_models_test	28	
extensions.visualizations.models.py	31	

core.controllers.domain_objects_validator_test	33	
core.controllers.voice_artist_test	33	
scripts.docstrings_checker_test	34	
core.storage.base_model.gae_models_test	42	
core.platform.datastore.cloud_datastore_services.py	60	
TOTAL	469	lines

Note: For the trivial files, the line count is kept zero considering the fact that they have no real logic to be tested. The line count for the files which are currently in the incomplete coverage denylist have been calculated by the coverage tool.

3. Achieve 100% backend branch coverage. This includes covering all branches of these files:

extensions.answer_summarizers.models.py	1
extensions.objects.models.objects.py	2
extensions.answer_summarizers.models.py	1
extensions.objects.models.objects.py	2
core.storage.blog.gae_models.py	1
core.storage.question.gae_models.py	1
core.storage.statistics.gae_models.py	1
core.controllers.creator_dashboard.py	1
core.controllers.payload_validator.py	1
core.controllers.base.py	1
core.controllers.library.py	2
core.controllers.skill_editor.py	1
core.controllers.questions_list.py	1
core.controllers.practice_sessions.py	1
core.controllers.topic_editor.py	4
core.controllers.learner_playlist.py	2
core.controllers.classroom.py	1
core.controllers.story_editor.py	1
core.controllers.blog_homepage.py	1
core.controllers.profile.py	5
core.controllers.topics_and_skills_dashboard.py	2
core.controllers.blog_admin.py	1

core.controllers.admin.py	4
core.controllers.subtopic_viewer.py	2
core.controllers.contributor_dashboard_admin.py	2
core.platform.bulk_email.mailchimp_bulk_email_services.py	3
core.platform.taskqueue.cloud_taskqueue_services.py	3
core.platform.taskqueue.cloud_tasks_emulator.py	2
core.platform.email.dev_mode_email_services.py	1
core.platform.auth.firebase_auth_services.py	2
core.controllers.contributor_dashboard.py	1
core.jobs.batch_jobs.math_interactions_audit_jobs.py	1
core.jobs.types.base_validation_errors.py	1
core.jobs.transforms.validation.base_validation.py	5
core.controllers.suggestion.py	2
core.controllers.reader.py	9
core.controllers.editor.py	11
core.domain.rights_domain.py	1
core.domain.platform_parameter_domain.py	14
core.domain.collection_domain.py	3
core.domain.user_query_domain.py	2
core.domain.feedback_services.py	4
core.domain.app_feedback_report_services.py	4
core.domain.question_services.py	3
core.domain.recommendations_services.py	3
core.domain.action_registry.py	2
core.domain.user_query_services.py	2
core.domain.email_manager.py	11
core.domain.playthrough_issue_registry.py	2
core.domain.visualization_registry.py	1
core.domain.param_domain.py	2
core.domain.config_domain.py	1
core.domain.question_domain.py	28
core.domain.summary_services.py	4
core.domain.learner_playlist_services.py	4
core.domain.calculation_registry.py	1

core.controllers.acl_decorators.py	8
core.domain.skill_domain.py	4
core.domain.translation_domain.py	1
core.domain.skill_services.py	13
core.domain.learner_goals_services.py	1
core.domain.topic_services.py	6
core.domain.event_services.py	1
core.domain.beam_job_services.py	1
core.domain.html_cleaner.py	2
core.domain.blog_services.py	1
core.domain.fs_services.py	4
core.domain.draft_upgrade_services.py	12
core.domain.interaction_registry.py	1
core.domain.search_services.py	1
core.domain.stats_domain.py	3
core.domain.html_validation_service.py	6
core.domain.platform_parameter_registry.py	1
core.domain.wipeout_domain.py	2
core.domain.rte_component_registry.py	2
core.domain.rules_registry.py	1
core.domain.classifier_services.py	4
core.domain.voiceover_services.py	2
core.domain.story_fetchers.py	2
scripts.linters.html_linter.py	5
scripts.linters.other_files_linter.py	3
scripts.linters.general_purpose_linter.py	3
scripts.release_scripts.repo_specific_changes_fetcher.py	3
scripts.create_topological_sort_of_all_services.py	2
scripts.clean.py	1
scripts.check_frontend_test_coverage.py	1
scripts.pre_commit_hook.py	1
scripts.install_third_party.py	6
scripts.rtl_css.py	1
scripts.servers.py	1

scripts.setup.py	1
scripts.build.py	3
scripts.concurrent_task_utils.py	1
scripts.install_third_party_libs.py	2
scripts.check_if_pr_is_low_risk.py	1
scripts.run_e2e_tests.py	2
scripts.common.py	6
scripts.pre_push_hook.py	6
core.domain.wipeout_service.py	5
core.domain.story_services.py	6
TOTAL	314 branches

Note: The branches above have been calculated by the coverage tool.

- Achieve 100% line coverage of all frontend files except those in core/templates/tests, as well as the two individual files unit-test-utils.ts and schema-based-list-viewer.directive.ts (see [#10427](#)), since those will be removed by the Angular Migration. This includes covering all lines in these files on a per-file basis:

angular-html-bind.directive.ts	1
App.ts	12
Base.ts	15
ck-editor-4-rte.component.ts	146
ck-editor-4-widgets.initializer.ts	65
exploration-states.service.ts	33
expression-interpolation.service.ts	2
google-analytics.initializer.ts	7
learner-answer-info.service.ts	5
mathjax-bind.directive.ts	8
object-editor.directive.ts	29
oppia-interactive-music-notes-input.component.ts	243
oppia-interactive-pencil-code-editor.component.ts	61
oppia-root.directive.ts	13
parameterize-rule-description.filter.ts	94
python-program.tokenizer.ts	59

question-update.service.ts	6
rule-type-selector.directive.ts	26
select2-dropdown.directive.ts	27
translation-file-hash-loader-backend-api.service.ts	1
truncate-input-based-on-interaction-answer-type.filter.ts	3
voiceover-recording.service.ts	26
TOTAL	882 lines

Note: The lines above have been calculated by Karma.

Third-Party Libraries

No new third-party libraries are required.

“Service” Dependencies

This feature adds no new service dependencies.

Impact on Other Oppia Teams

The project is of Dev workflow and the checks it will add will force Oppia developers to first cover all of their files (including both line and branch coverage) before pushing their code to Oppia.

Risks and mitigations

No potential risks.

Implementation Approach

WRITING BACKEND TESTS

Oppia uses Python’s [unittest framework](#) to test backend code. Backend test files live alongside the backend code files they test. For example, alongside `core/controllers/base.py` we'll find `core/controllers/base_test.py`. Note that each file is entirely code or entirely tests. No file mixes code and tests.

Inside test files, tests are organized into classes whose names end in Tests. Test cases are methods of these classes, and the method names begin with test_. Note that only methods beginning with test_ and inside classes that inherit from 'unittest.TestCase' are executed as tests. Here is an example of test from 'core/domain/collection_domain_test.py':

```
class CollectionChangeTests(test_utils.GenericTestBase):

    def test_collection_change_object_with_missing_cmd(self) -> None:
        with self.assertRaisesRegex( # type: ignore[no-untyped-call]
            utils.ValidationError, 'Missing cmd key in change dict'):
            collection_domain.CollectionChange({'invalid': 'data'})

    def test_collection_change_object_with_invalid_cmd(self) -> None:
        with self.assertRaisesRegex( # type: ignore[no-untyped-call]
            utils.ValidationError, 'Command invalid is not allowed'):
            collection_domain.CollectionChange({'cmd': 'invalid'})
```

Notice that the test class inherits from test_utils.GenericTestBase which in turn inherits from unittest.TestCase.

To write backend tests, following steps should be followed:

1. First understand the working of the function a test is going to cover. Understand the parameters it receives and how it makes use of them to update something or return something.
2. Try to test the interface, not the implementation. Treat the function as a black box and test its behavior. Don't test that the function uses a particular implementation. This will help to design a better API from an external user's perspective.
3. Give meaningful names to the tests. Following naming syntax has been recommended by Oppia:

test_{{action}}_with_{{with_condition}}_{{has_expected_outcome}}

where {{action}}, {{with_condition}}, and {{has_expected_outcome}} are replaced with appropriate descriptions.

4. Tests should use the following general structure:
 - a. Setup - this is where you prepare any inputs/environment needed for the test.
 - b. Baseline verification - check the values without performing any action. This step is only needed if your action is state-changing (i.e. if the same assert statement would lead to one result at the baseline and a different result at the endline). Check the same values here as you check at the endline.
 - c. Action - perform the action or function call that leads to the expected change.
 - d. Endline verification - check that the values from the baseline verification have changed accordingly.

5. Try to keep the logic of the test simple. Write the test as a series of straightforward, descriptive and meaningful commands (also known in programming circles as "DAMP"). Use comments wherever necessary.
6. To check outputs, following points should be followed:
 - a. Test each output as exactly and completely as possible. For example, it's better to compare equality for an entire dict rather than just checking that a particular value has changed.
 - b. Use `assertTrue(value)` or `assertFalse(value)` instead of `assertEqual(value, True)` or `assertEqual(value, False)`.
 - c. Use `assertIsNone(value)` instead of `assertEqual(value, None)`.
7. If you want to test code within a private method/function, test it by instead calling a public function that makes use of that function, or move it to a utility (if it's general-purpose) where it becomes public.
8. You should test each method of a class individually, with one or more test cases for each method. Define a `setUp()` method in the test if functionality or variables are going to be reused between tests. Here is an example of a good `setUp()` method from 'core/domain/exp_fetchers_test.py':

```
class ExplorationRetrievalTests(test_utils.GenericTestBase):
    """Test the exploration retrieval methods."""

    EXP_1_ID = 'exploration_1_id'
    EXP_2_ID = 'exploration_2_id'
    EXP_3_ID = 'exploration_3_id'

    def setUp(self):
        super(ExplorationRetrievalTests, self).setUp()
        self.signup(self.OWNER_EMAIL, self.OWNER_USERNAME)
        self.owner_id = self.get_user_id_from_email(self.OWNER_EMAIL)
        self.exploration_1 = self.save_new_default_exploration(
            self.EXP_1_ID, self.owner_id, title='Aa')
        self.exploration_2 = self.save_new_default_exploration(
            self.EXP_2_ID, self.owner_id, title='Bb')
        self.exploration_3 = self.save_new_default_exploration(
            self.EXP_3_ID, self.owner_id, title='Cc')

    def test_get_exploration_summaries_matching_ids(self):
        summaries = exp_fetchers.get_exploration_summaries_matching_ids([
            self.EXP_1_ID, self.EXP_2_ID, self.EXP_3_ID, 'nonexistent'])
        self.assertEqual(summaries[0].title, self.exploration_1.title)
        self.assertEqual(summaries[1].title, self.exploration_2.title)
        self.assertEqual(summaries[2].title, self.exploration_3.title)
        self.assertIsNone(summaries[3])

    def test_get_exploration_summaries_subscribed_to(self):
        summaries = exp_fetchers.get_exploration_summaries_subscribed_to(
            self.owner_id)
        self.assertEqual(summaries[0].title, self.exploration_1.title)
```

```
self.assertEqual(summaries[1].title, self.exploration_2.title)
self.assertEqual(summaries[2].title, self.exploration_3.title)
```

Oppia's wiki about writing [backend-tests](#) can be referred to know more about writing good tests.

The files to be tested in milestone1 are following methods can be used while testing them:

- a. Most of the files in the scripts folder call various subprocess methods whose real working is harder to test. However, they can be tested by creating mock functions to ensure that necessary calls are made. An example:

```
self.cmd_token_list = []
def mock_check_call(cmd_tokens, **unsued_kwargs):
    self.cmd_token_list.append(cmd_tokens)
    return MockTask()

self.swap_Popen = self.swap(
    subprocess, 'Popen', mock_check_call)
```

'cmd_token_list' will contain the list of all commands called using subprocess in a method. We can check that to ensure correct commands were made.

- b. We also need to ensure that correct messages are displayed on the console while the tests are running. To check that we can create a mock like this:

```
self.print_arr = []
def mock_print(msg):
    self.print_arr.append(msg)
self.print_swap = self.swap(builtins, 'print', mock_print)
```

'print_arr' will contain a list of all messages that were printed on the console while the method was run. We can check that to ensure correct messages were displayed.

WRITING FRONTEND TESTS

Oppia uses [Jasmine](#) and Karma for frontend testing. Frontend test files live alongside the frontend code files they test.

For example, alongside `core/templates/pages/admin-page/admin-page.component.ts` we'll find `core/templates/pages/admin-page/admin-page.component.spec.ts`. Note that each file is entirely code or entirely tests. No file mixes code and tests.

Inside the test files, there are many test function, some of them are:

- describe

The describe function has a string parameter which should contain the name of the component being tested or (when nested within another describe function) should describe the conditions imposed on the specific context pertaining to the tests in that describe block. Here are some examples:

- An outer describe function

```
describe('Component Name', function() {  
  ...  
});
```

- Two describe functions nested inside another describe function:

```
describe('Component Name', function() {  
  describe('when it is available', function() {...});  
  
  describe('when it is not available', function() {...});  
});
```

- **beforeEach**

The beforeEach function is used to set up essential configurations and variables before each test runs. This function is mostly used for three things:

- Injecting the modules to be tested or to be used as helpers inside the test file.
- Mocking the unit test's external dependencies (only on AngularJS files).
- Providing Angular2+ services in downgrade files when the AngularJS service being tested uses any upgraded (Angular2+) service as a dependency.

Here is a real example from Oppia's codebase:

```
describe('Admin Page component ', () => {  
  let component: AdminPageComponent;  
  let fixture: ComponentFixture<AdminPageComponent>;  
  
  let adminRouterService: AdminRouterService;  
  let mockWindowRef: MockWindowRef;  
  
  beforeEach(() => {  
    mockWindowRef = new MockWindowRef();  
    TestBed.configureTestingModule({  
      imports: [HttpClientTestingModule],  
      declarations: [AdminPageComponent],  
      providers: [  
        AdminRouterService,  
        ChangeDetectorRef,  
        {  
          provide: WindowRef,  
          useValue: mockWindowRef  
        }  
      ],  
    });  
  });  
});
```



```

    provide: PlatformFeatureService,
    useClass: MockPlatformFeatureService
  },
],
schemas: [NO_ERRORS_SCHEMA]
}).compileComponents();

fixture = TestBed.createComponent(AdminPageComponent);
component = fixture.componentInstance;
});

```

- it

The 'it' function is where the test happens. Like the describe function, its first parameter is a string which should determine the action to be tested and the expected outcome of the tests. The string should have a clear description of what is going to be tested. Also, the string must start with "should".

Here is a real example from Oppia's codebase:

```

it('should save state content', function() {
  stateEditorService.setActiveStateName('First State');
  expect(explorationStatesService.getState('First State').content).toEqual(
    SubtitledHtml.createFromBackendDict({
      content_id: 'content',
      html: 'First State Content'
    }));

  var displayedValue = SubtitledHtml.createFromBackendDict({
    content_id: 'content',
    html: 'First State Content Changed'
  });
  ctrl.saveStateContent(displayedValue);

  expect(explorationStatesService.getState('First State').content).toEqual(
    displayedValue);
  expect(ctrl.interactionIsShown).toBe(true);
});

it('should save state interaction id', function() {
  stateEditorService.setActiveStateName('First State');
  stateEditorService.setInteraction(
    explorationStatesService.getState('First State').interaction);

  expect(stateEditorService.interaction.id).toBe('TextInput');

  var newInteractionId = 'Continue';
  ctrl.saveInteractionId(newInteractionId);

  expect(stateEditorService.interaction.id).toBe(newInteractionId);
});

```

```
});
```

- **afterEach**

The `afterEach` function runs after each test, and it is not used often. It's mostly used when we are handling async features such as HTTP and timeout calls (both in AngularJS and Angular 2+).

- **afterAll**

The `afterAll` function runs after all the tests have finished, but it is almost never used in the codebase. There is a specific case which might be very helpful: when a global variable needs to be reassigned during the tests, you need to reset it to the default value after all the assertions are finished.

- **expect**

The `expect` function is used to assert a condition in the test. More can be learned about its methods in the [Jasmine documentation](#).

Here is a real example from Oppia's codebase:

```
it('should initialize the component and set values', fakeAsync(() => {
  componentInstance.ngOnInit();
  expect(componentInstance.explorationId).toEqual(explId);
  expect(componentInstance.expTitle).toEqual(expTitle);
  expect(componentInstance.expDesc).toEqual(expDesc);

  expect(componentInstance.expTitleTranslationKey).toEqual(
    'I18N_EXPLORATION_explId_TITLE');
  expect(componentInstance.expDescTranslationKey).toEqual(
    'I18N_EXPLORATION_explId_DESCRIPTION');

  // Translation is only displayed if the language is not English
  // and its hacky translation is available.
  let hackyExpTitleTranslationIsDisplayed = (
    componentInstance.isHackyExpTitleTranslationDisplayed());
  expect(hackyExpTitleTranslationIsDisplayed).toBe(true);
  let hackyStoryTitleTranslationIsDisplayed = (
    componentInstance.isHackyStoryTitleTranslationDisplayed());
  expect(hackyStoryTitleTranslationIsDisplayed).toBe(true);
  let hackyExpDescTranslationIsDisplayed = (
    componentInstance.isHackyExpDescTranslationDisplayed());
  expect(hackyExpDescTranslationIsDisplayed).toBe(false);
}));
```

Other good practise about writing frontend tests and naming conventions can be learned from Oppia's wiki about writing [frontend-tests](#).

Storage Model Layer Changes

None

Domain Objects

No new domain objects to be created and none updated.

User Flows (Controllers and Services)

None

Web frontend changes

None

Documentation changes

The documentation changes to be made include:

- Updates in [backend-tests wiki](#) explaining users how to run branch coverage checks (currently wiki mentions only the commands for checking line coverage of a file).
- Documentation recording any infrastructure changes made while working on this project

Testing Plan

Backend-Testing-Framework Testing

All the testing for the backend-testing infrastructure changes proposed above would be manual where I will be trying to push files with incomplete coverage and check if the backend coverage checks fail.

Note: *Idle function* in tests below means a function that simply returns a value that is passed to it as a parameter.

#	Test name	Initial setup step	Step	Expectation
1.	Check if files lacking 100% per-file backend line coverage fail	<p>Add an idle function (let's say A) in any 1 python code file that has an associated test file and do not write tests for it.</p> <p>Add a function (let's say B) in another python code file code file that calls function A. Write tests for function B in the</p>	Run backend coverage checks.	Coverage checks fail due to less than 100% per-file backend line coverage.

	backend coverage checks.	associated test file. This ensures that overall line coverage is 100% since function A is also tested but the per-file line coverage is less than 100%.		
2.	Check if files lacking 100% per-file backend branch coverage fail backend coverage checks.	Add a modified idle function that returns back the value passed to it only when the parameter has an specific value, i.e, add an <i>if</i> condition. Write tests for the function passing only that specific value to it. This will ensure that the per-file line coverage is 100% but the per-file branch coverage is less than 100% since the condition when <i>If</i> is not matched is never met.	Run backend coverage checks.	Coverage checks fail due to less than 100% per-file backend branch coverage.
3.	Check if files lacking 100% per-file backend line coverage fails pre_push hook.	Add an idle function (let's say A) in any 1 python code file that has an associated test file and do not write tests for it. Add a function (let's say B) in another python code file code file that calls function A. Write tests for function B in the associated test file. This ensures that overall line coverage is 100% since function A is also tested but the per-file coverage is less than 100%.	Commit the changes and try to push the changes to GitHub repo.	Push fails due to incomplete backend per-file line coverage.
4.	Check if files lacking 100% per-file backend branch coverage fail pre_push hook.	Add a modified idle function that returns back the value passed to it only when the parameter has an specific value, i.e, add an <i>if</i> condition. Write tests for the function passing only that specific value to it. This will ensure that the per-file line coverage is 100% but the per-file branch coverage is less than 100% since the condition when <i>If</i> is not matched is never met.	Commit the changes and try to push the changes to GitHub repo.	Push fails due to incomplete backend per-file branch coverage.
5.	Check if files lacking	Add an idle function (let's say A) in any 1 python code file that has an associated test file	Commit the changes and push them to	GitHub checks fail due to incomplete backend per-file line coverage.

	100% per-file backend line coverage fail GitHub checks.	and do not write tests for it. Add a function (let's say B) in another python code file code file that calls function A. Write tests for function B in the associated test file. This ensures that overall line coverage is 100% since function A is also tested but the per-file coverage is less than 100%.	GitHub using the `no verify` flag. Make a PR to a separate branch of the repo.	
6.	Check if files lacking 100% per-file backend branch coverage fail GitHub checks.	Add a modified idle function that returns back the value passed to it only when the parameter has an specific value, i.e, add an <i>if</i> condition. Write tests for the function passing only that specific value to it. This will ensure that the per-file line coverage is 100% but the per-file branch coverage is less than 100% since the condition when <i>if</i> is not matched is never met.	Commit the changes and push them to GitHub using the `no verify` flag. Make a PR to a separate branch of the repo.	GitHub checks fail due to incomplete backend per-file branch coverage.

Feature testing

Does this feature include non-trivial user-facing changes?

NO

Implementation Plan

Milestone 1:

Key Objectives:

- **Infrastructure:** Ensure that all backend files have an associated test file. Use a denylist to exclude files in core/tests/data, scripts/linters/test_files, and core/tests/build_sources, since these folders contain files that are just used as inputs for our tests.
- **Testing:** Ensure that every backend file has an associated test file (even a trivial one) so that coverage checks are guaranteed to be triggered for all files. It is OK if these added files are added to the incomplete per-file coverage denylist.

- **Infrastructure:** Calculate backend branch coverage on CI and use a temporary denylist (to be removed before end of M2) to ensure that backend branch coverage does not regress.
- **Testing:** Achieve 100% backend line coverage, including for those files that previously slipped past our coverage checks because they had no associated test file.

No.	Description of PR / action	Prereq PR numbers	Target date for PR creation	Target date for PR to be merged
1	Check that every backend file has an associated test file and add trivial test files + denylist for data files.	-	22nd June	28th June
2	Cover 2 script files previously lacking associated test file.	-	27th June	4th July
3	Cover 2 script files previously lacking associated test file.	-	3rd July	10th July
4	Cover 2 script files previously lacking associated test file.	-	9th July	15th July
5	Cover 2 script files previously lacking associated test file.	-	14th July	20th July
6	Cover 3 script files previously lacking associated test file.	-	19th July	25th July
7	Update the workflow of backend tests to calculate and check backend branch coverage + add a denylist.	-	24th July	30th July

Milestone 2:

Key Objectives:

- **Testing:** Achieve 100% line coverage of all frontend files except those in core/templates/tests, as well as the two individual files unit-test-utils.ts and schema-based-list-viewer.directive.ts (see [#10427](#)), since those will be removed by the Angular Migration. By the end of Milestone 2, the frontend test denylist should only contain the aforementioned folder + 2 files.
- **Testing:** Achieve 100% backend branch coverage.
- **Testing:** Achieve 100% per-file backend line coverage except for the files in core/domain/.
- **Infrastructure:** Remove the temporary denylist for backend branch coverage.

No.	Description of PR / action	Prereq PR numbers	Target date for PR creation	Target date for PR to be merged
1	Cover 30% of the backend files lacking per-file coverage in the denylist (except those in core/domain/).	-	2nd August	8th August
2	Cover 30% of the backend files lacking per-file coverage in the denylist (except those in core/domain/).	-	7th August	13th August
3	Cover 40% of the backend files lacking per-file coverage in the denylist (except those in core/domain/).	-	12th August	18th August
4	Cover 150 uncovered branches from the files from incomplete branch coverage denylist.	-	19th August	25th August
5	Cover 164 uncovered branches from the files from incomplete branch coverage denylist.	-	26th August	1st September
6	Cover 50% of the files in the frontend incomplete coverage denylist (except the aforementioned folder + 2 files)	-	2nd September	8th September
7	Cover 50% of the files in the frontend incomplete coverage denylist (except the aforementioned folder + 2 files)	-	10th September	16th September

Future Work

- **BACKEND**
 - We will have achieved 100% backend line and branch coverage, but making the backend files in core/domain/ will not be covered on a per-file basis.
- **FRONTEND**
 - Find a way to measure per-file frontend coverage without the performance hits we saw when removing the combined-tests spec file.
 - Achieve 100% frontend branch coverage.
 - Make frontend coverage per-file and enforce using CI checks.