

Frontend testing of controllers and directives

About Me

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

I've started to work on Oppia in October 2019, during the Outreachy contribution period. One of the reasons why I chose Oppia at that time was because I'm really interested in the Angular framework (both Angular 2+ and AngularJS). Also, I'm really into FOSS with educational impact.

During Outreachy, I've worked with Increase frontend test coverage project, which I focused on services files. I was interested in working with unit tests because it was something that I wanted to learn by doing, I was looking for a challenging project and it was a really great experience.

Now, I'd like to work more on other types of files, like directives and controllers, in order to learn the full unit testing experience in Angular and acquire expertise in it.

Prior experience

I have a good prior experience, and working with Oppia is one the best that I had. I developed great changes in frontend testing process:

- [#8130](#) Created a Python script in order to make sure all the fully covered tests will not decrease.
- Fixed karma report problems.
 - [#8085](#) Removed spec files from it and added files that was being ignored.
 - [#8388](#) Added sourcemap to report correctly the lines coverage.
- [#8528](#) Added first tests for controllers.
- [#8136](#) [#8469](#) [#8729](#) Written tests for easy, medium and hard services.
- [#8753](#) Found a way to fully test services which uses \$uibModal.

Contact info and timezone(s)

Email: mari.zangue@gmail.com (I'm on Hangouts too!)

Gitter: marianazangrossi

I'll be working from Brazil, on BRT timezone (UTC -3).

Time commitment

I'll be working on this project 40 hours a week. I'm going to work approximately 6 hours a day from Monday to Sunday.

Essential Prerequisites

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

```
-----
Tasks still running:
  core.controllers.collection_viewer_test (started 21:05:15)
-----
00:05:21 FINISHED core.controllers.collection_viewer_test: 6.0 secs

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

SUCCESS   core.controllers.collection_viewer_test: 9 tests (4.8 secs)

Ran 9 tests in 1 test class.
All tests passed.

Done!
```

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

```
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1968 of 1972 SUCCESS (0 secs / 22.917 secs)
ERROR: 'Unexpected call to ExplorationDataService for pathname '
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1969 of 1972 SUCCESS (0 secs / 22.926 secs)
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1972 of 1972 SUCCESS (26.732 secs / 22.943 secs)
TOTAL: 1972 SUCCESS
TOTAL: 1972 SUCCESS
```

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

```
Started
Jasmine started
[20:59:07] W/element - more than one element found for locator By(css selector, .protractor-test-create-topic-button) - the first result will be used
[20:59:07] W/element - more than one element found for locator By(css selector, .protractor-test-create-topic-button) - the first result will be used
[20:59:07] W/element - more than one element found for locator By(css selector, .protractor-test-create-topic-button) - the first result will be used
.
  Topic editor functionality
    ✓ should edit topic name, abbreviated topic name, thumbnail and description correctly

1 spec, 0 failures
Finished in 30.562 seconds
Executed 1 of 1 spec SUCCESS in 31 secs.
```

Other summer obligations

I'm not going to have any summer obligations. My classes were suspended due to COVID-19, however if it comes back during the Summer of Code, it won't affect my work because I study at night.

Communication channels

- Meeting with the mentor twice a week on Hangouts to discuss how the tasks are going.
- Hangouts chat for quick upgrades, messages and questions.
- Gmail for reporting daily progress with mentor/coordinators.

Project Details

Product Design

Oppia is an educational online platform which uses different tools to improve the learning experience by mainly creating explorations, a way to teach any subject you'd like with interactive activities.

In order to have tools supporting all kinds of activities, Oppia has a large codebase and to be sure each of these tools are working properly and it won't affect the student experience, it needs to be tested in many ways.

One of the ways to test a feature is testing its units. Unit is the smallest piece of code that can be tested, like a function. When writing unit tests, the unit itself is being tested and the final result when many units is communication with each other.

When talking about this kind of test, we are not only focusing on the final user (students, teachers, etc) but on the developers as well. Unit testing is very important for software development for some reasons:

- **Check if the code is correct:** it's crucial to have an automated way to test if the code is dealing right with the happy and unhappy path, since it'd be very boring and delayed to test all the values by hand. Unit tests give the possibility to test more in less time.
- **Avoid regressions**, which means fix a bug that was already "solved". Regression is an obstacle for an uncomplicated development flow, because more time is "wasted" redoing a task, instead of focusing on new features. Using no unit test probably will cause a technical debit.
- **Documentation:** when a unit test is written, it creates a documentation for your code, since you're describing how it should work in all paths.
- **Readability:** to write the test, you should know how the code works. It turns the design

better, since the small pieces of the code are going to be tested. It'd be difficult to write a test suite for a function that has more than 300 lines. The best approach is to separate it into small functions, and that's a good practice in software development.

The reasons mentioned above can be resumed as: unit tests improve the productivity of the development team. Then, the developers can focus on creating and improving features instead of fixing bugs, the product turns out to be more consistent both to the engineering team and the final users.

This project aims to achieve 100% coverage of frontend controllers and test 23.5% of directives (71 directives), but also, keep the coverage stable, which means that after testing a file, it won't have the coverage decreased unexpectedly, avoiding any regressions. A fully coverage file means all lines of the file should have been tested correctly, expecting happy and unhappy paths. The explanation why directives are not following the same path as controllers are clearly explained during the proposal. However, by the end of the internship, the development team should have a nice overview of how frontend tests work and the Angular migration team should have a nice path on upgrading components and therefore the final users should have a smoother experience than before.

Technical Design

Architectural Overview

Oppia Folder Structure

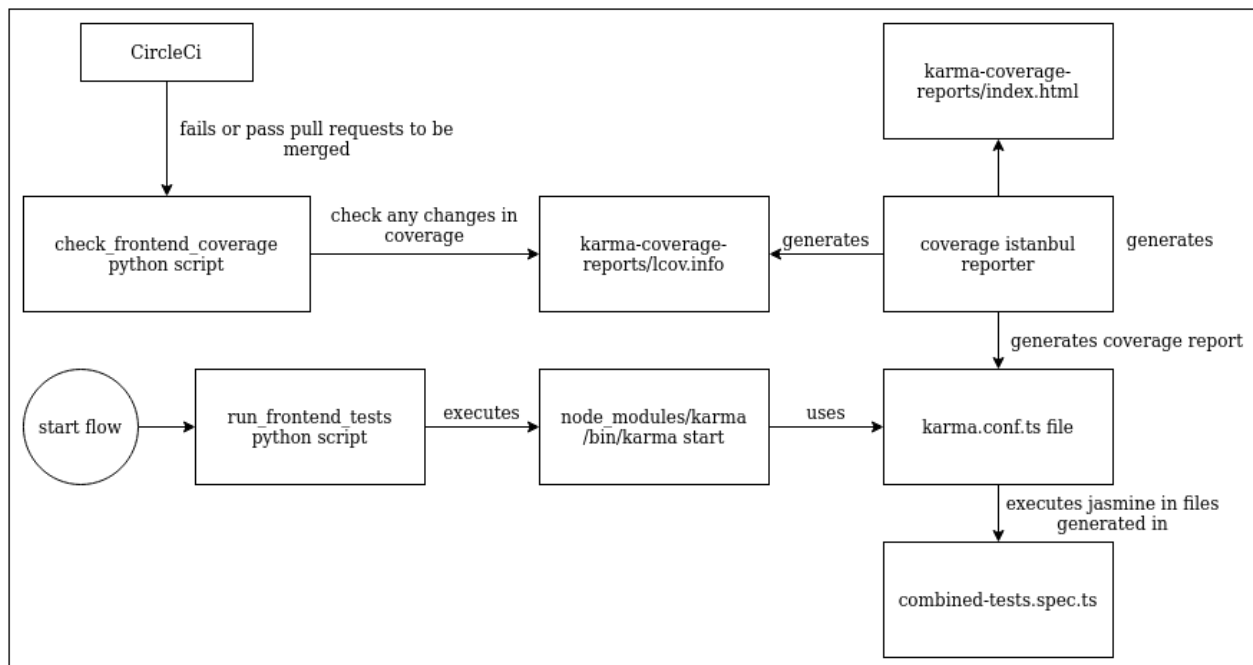
All the frontend code is in the `core/templates` folder, although some controllers and directives are in the `extensions` folder. It's important to note that `extensions` folder is tracked by Karma and then it should be tested as well. Here's the folders where controllers and directives are distributed:

- `core/templates`: frontend code
 - `base-components`: Core components, which is used once in the application.
 - `components`: Reusable components.
 - `directives`: Directives that are not associated with reusable components.
 - `domain`: The logic layer from frontend.
 - `pages`: Each page of the web application.

- `services`: Shared services through the application.
- `extensions`: extensions where can be implemented by the developers
 - `interactions`: Components that allow the learner to submit an answer.
 - `objects`: Objects that Oppia recognizes.
 - `rich_text_components`: Provides additional functionality for content that is shown to the learner.
 - `visualizations`.
 - `value_generators`: Functions which take some inputs and produce a single output.

Frontend test flow

Here's a diagram of how the frontend testing flow works, since the running of Python script to the CircleCI checking if a Pull Request is able to be merged without making any regressions on the coverage.



Here's each step explained in details:

- Everything starts on running command `python -m scripts.run_frontend_tests`. This command is also executed when pushing some commits to GitHub.

- This script gets karma executable from `node_modules` folder and starts it.
- Karma uses `karma.conf.ts` file as configuration.
- `combined-tests.spec.ts` join all spec files to be included in Karma.
- Coverage Istanbul Reporter plugin generates `index.html` and `lcov.info` files as reporters.
- The `index.html` is used as a local report when writing tests.
- The `lcov.info` file is used in `check_frontend_coverage` script to check if there are any changes in test coverage.
- `check_frontend_coverage` script is executed on CircleCI jobs to pass or block a Pull Request merge. In order to keep the coverage stable, all the tested files will be listed in this file, as you can see above:

```
# Contains the name of all files that reached up to 100% of coverage.
# This list must be kept up-to-date; the changes should be done manually.
# Please keep the list in alphabetical order.
# NOTE TO REVIEWERS: Please be circumspect about any PRs which delete elements
# from this list.
FULLY_COVERED_FILENAMES = [
    'about-page.controller.ts',
    'admin-config-tab-backend-api.service.ts',
    'admin-data.service.ts',
    'admin-page.constants.ajs.ts',
    'admin-page.constants.ts',
    'admin-router.service.ts',
    'admin-task-manager.service.ts',
    'alerts.service.ts',
    'angular-name.service.ts',
    'answer-groups-cache.service.ts',
```

When testing controllers and directives, this flow will be followed in the same way as steps described below.

Directives

AngularJS directives is a way to create custom behavior for DOM elements, transform it and its children and insert new content. It's one of the biggest features from the Angular ecosystem and it's being used a lot in the whole frontend codebase, counting **304 directives** in total.

There are many types of directives, each one is called in a certain way in HTML. Here's the two types that is used in the project:

- Attribute name

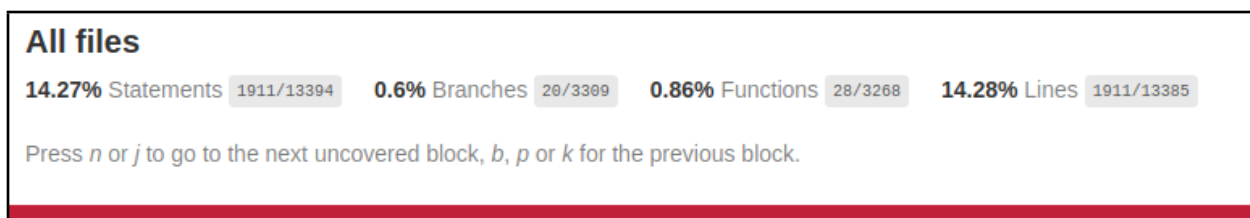
- Restrict: 'A'
- It's called as `<div my-directive></div>`
- Element name
 - Restrict: 'E'
 - It's called as `<my-directive></my-directive>`

Directives are identified as files with **.directive.ts** extension. These files are distributed in the codebase:

- 209 in the `core/templates` folder.
- 95 in the `extensions` folder.

Current Coverage

By the date of this proposal, directives are 14.27% covered.



There is only 8 directives which has unit tests:

state-interaction-editor.directive.ts	Ignored by combined-tests.spec.ts
state-content-editor.directive.ts	Ignored by combined-tests.spec.ts
rating-display.directive.ts	68.42%
apply-validation.directive.ts	100%
require-is-float.directive.ts	100%
state-name-editor.directive.ts	Ignored by combined-tests.spec.ts
exploration-editor-tab.directive.ts	54.64%
history-tab.directive.ts	36.75%

However, only 2 of them are fully covered. So, 302 directives need to be tested (3 need to be improved to reach 100% and 299 need to be tested from scratch). However, that's quite a lot to complete within 3 months. In Milestone 2 and 3, **71 directives are going to be tested**, in order

to fully cover some folders.

71 directives represents 23.5% directives from 100%. This number of directives was chosen on picking directives from `core/templates/pages` in order to achieve 100% coverage in some of its subfolders. One of these subfolders is `exploration-editor-page`, which is the biggest subfolder with a lot of different files to test, turning to be a good mark to work on the project. The other subfolders don't have a lot of files, however, it was chosen because of file quantity to work on Milestone 3 in order to fully cover it.

Controllers

Controllers are responsible for the business logic behind a view (HTML and CSS). Every interaction or change is made by the controller and the view “listen” to it and keep all the graphic interfaces updated.

There are 37 controllers in the codebase:

- 36 in `core/templates`, mainly in pages folder, because it's where all the most important business interactions happen.
- 1 in `extensions/interactions`.

However, 37 isn't the right number at all. There are more controllers across the codebase.

Controllers can control whole pages to small components. Oppia uses the [UI Bootstrap](#) library to display and configure common components easily, and is here where some controllers are “hidden” from the testing process and it needs to show up so the coverage percentage will be calculated correctly.

One of the components from UI Bootstrap is the modal, which is widely used in the frontend (modals are very important for user experience). To open a modal, `$uibModal.open` is called. Here's an example in the codebase:


```

var modalInstance = $uibModal.open({
  templateUrl: UrlInterpolationService.getDirectiveTemplateUrl(
    '/pages/topic-editor-page/modal-templates/' +
    'new-story-title-editor.template.html'),
  backdrop: true,
  controller: [
    '$scope', '$uibModalInstance',
    function($scope, $uibModalInstance) {
      $scope.storyTitle = '';
      $scope.isStoryTitleEmpty = function($storyTitle) {
        return (storyTitle === '');
      };
      $scope.save = function($storyTitle) {
        $uibModalInstance.close(storyTitle);
      };
      $scope.cancel = function() {
        $uibModalInstance.dismiss('cancel');
      };
    }
  ]
});

```

There is a controller property where a controller is defined to handle the logic behind the modal view. However, when the controller is written inline, it's not possible to test it when using unit tests. And it comes to two problems:

- Files that use `$uibModal.open` are tricky to test and may confuse new contributors (that are not aware of the problem yet). It'll affect the coverage of the file itself as well, since part of the file should be in another file that will be entirely responsible for it.
- The number of controllers are not precise therefore the coverage.

Here's is the list of files where uses inline controllers in `$uibModal.open`:

File name	Number of inline controllers
create-activity-button.directive.ts	1
exploration-embed-button.service.ts	1
exploration-creation.service.ts	1
story-creation.service.ts	1
topic-creation.service.ts	1
thumbnail-uploader.directive.ts	1
question-player.directive.ts	2
questions-list.directive.ts	6

skills-mastery-list.directive.ts	1
answer-group-editor.directive.ts	1
state-hints-editor.directive.ts	3
state-interaction-editor.directive.ts	2
state-responses.directive.ts	2
state-solution-editor.directive.ts	2
version-diff-visualization.directive.ts	1
collection-editor-navbar.directive.ts	2
contributions-and-review.directive.ts	2
question-opportunities.directive.ts	2
translation-opportunities.directive.ts	1
delete-account-page.controller.ts	1
exploration-editor-page.controller.ts	1
editor-navigation.directive.ts	1
exploration-graph.directive.ts	1
unresolved-answers-overview.directive.ts	1
feedback-tab.directive.ts	1
history-tab.directive.ts	1
preview-tab.directive.ts	1
autosave-info-modals.service.ts	3
exploration-save.service.ts	6
exploration-states.service.ts	1
settings-tab.directive.ts	4
statistics-tab.directive.ts	1
suggestion-modal-for-exploration-editor.service.ts	1
translation-tab.directive.ts	1
audio-translation-bar.directive.ts	3
state-translation-editor.directive.ts	1
learner-local-nav.directive.ts	2
learner-view-info.directive.ts	1
fatigue-detection.service.ts	1
hint-and-solution-modal.service.ts	3
refresher-exploration-confirmation-modal.service.ts	1
suggestion-modal-for-exploration-player.service.ts	2
learner-dashboard-page.controller.ts	1
suggestion-modal-for-learner-dashboard.service.ts	1

preferences-page.controller.ts	1
signup-page.controller.ts	2
skill-concept-card-editor.directive.ts	2
skill-misconceptions-editor.directive.ts	2
skill-prerequisite-skills-editor.directive.ts	1
skill-editor-navbar.directive.ts	2
story-editor-page.controller.ts	1
story-editor.directive.ts	2
story-node-editor.directive.ts	3
story-editor-navbar-breadcrumb.directive.ts	1
story-editor-navbar.directive.ts	1
topic-editor-stories-list.directive.ts	2
topic-editor-tab.directive.ts	1
topic-editor-navbar.directive.ts	2
subtopics-list-tab.directive.ts	2
topics-and-skills-dashboard-page.controller.ts	1
topics-and-skills-dashboard-navbar.directive.ts	1
skills-list.directive.ts	3
topics-list.directive.ts	1
Total	105

There are **143 controllers** in the codebase, including the inline controllers.

Current coverage

By the date of this proposal, controllers (not including inline ones) are 53.49% covered.

All files

53.49% Statements1187/2219

31.36% Branches148/472

42.03% Functions195/464

53.54% Lines1187/2217

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Oppia has 17 controllers that has unit tests:

File name	Coverage percentage
mark-all-audio-and-translations-as-needing-update.controller.ts	100%
rte-helper-modal.controller.ts	100%
learner-playlist-modal.controller.ts	100%

splash-page.controller.ts	100%
notifications-dashboard-page.controller.ts	100%
moderator-page.controller.ts	100%
email-dashboard-page.controller.ts	100%
email-dashboard-result.controller.ts	100%
about-page.controller.ts	100%
training-data-editor-panel-modal.controller.ts	100%
training-modal.controller.ts	100%
suggestion-modal-for-creator-view.controller.ts	100%
creator-dashboard-page.controller.ts	47.88%
preferences-page.controller.ts	51.20%
signup-page.controller.ts	63.11%
teach-page.controller.ts	100%
stewards-landing-page.controller.ts	100%
topic-landing-page.controller..ts	100%

There are 15 fully covered. So, 22 controllers (not including inline controllers) need to be tested (3 need to be improved to reach 100% and 19 need to be tested from scratch).

Implementation Approach

Level criteria

In order to define the implementation approach, the service files is going to be separated into the following criteria:

	Easy	Medium	Hard
Complexity	Up to 100 lines	Up to 250 lines	At least 250 lines

Time to test files of each type of level

	Easy	Medium	Hard
Time	4 files per day	3 files per 2 days	1 file per day

Directives


For element name directives, there is a special directive for it (instead of using `restrict: 'E'`). It is called a component (a very common feature on Angular 2+). As the codebase is being migrated from AngularJS to Angular 2+, it's important to keep the code organized to help the upgrade progress. Since the majority of directives in the codebase is matching with element name, it's better to change it to a component, and test it as a component as well.


Changing directive to component

```
angular.module('oppia').directive('correctnessFooter', [
  function() {
    return {
      restrict: 'E',
      scope: {},
      template: require('./correctness-footer.directive.html')
    };
  }]);
```

```
angular.module('oppia').component('correctnessFooter', {
  template: require('./correctness-footer.directive.html')
});
```

Its name must change too:

 correctness-footer.directive.ts

 correctness-footer.component.ts

This change is not only important for the upgrade project, but also to automatically increase the coverage of the file, mainly if it's just a static component and doesn't have a controller on it (like the example above).

The directive coverage before the change:



```

19 1x angular.module('oppia').directive('correctnessFooter', [
20     function() {
21     return {
22         restrict: 'E',
23         scope: {},
24         template: require('./correctness-footer.directive.html')
25     };
26     }]);

```

The directive coverage after the change (don't need a spec file just to covered it):



```

19 1x angular.module('oppia').component('correctnessFooter', {
20     template: require('./correctness-footer.directive.html')
21 });

```

There is only 3 directives that matches with attribute name:

- apply-validation.directive.ts
- require-is-float.directive.ts
- state-graph-visualization.directive.ts

The others **301 directives must be changed to components** during the testing process.

Directive testing list

There are 302 directives listed below:

- 186 easy directives.
- 77 medium directives.
- 39 hard directives.

Only 71 of them will be tested. It does not seem to be much, but the focus will be in some specific folders, as described on Milestone 2 and Milestone 3.

Filename	Lines	Level
correctness-footer.directive.ts	26	Easy
oppia-response-end-exploration.directive.ts	32	Easy
schema-based-html-viewer.directive.ts	32	Easy
continue-button.directive.ts	32	Easy
topics-and-skills-dashboard-navbar-breadcrumb.directive.ts	33	Easy

oppia-short-response-end-exploration.directive.ts	33	Easy
schema-based-expression-editor.directive.ts	33	Easy
schema-based-bool-editor.directive.ts	33	Easy
collection-history-tab.directive.ts	33	Easy
loading-dots.directive.ts	33	Easy
collection-statistics-tab.directive.ts	33	Easy
lazy-loading.directive.ts	34	Easy
admin-prod-mode-activities-tab.directive.ts	35	Easy
social-buttons.directive.ts	36	Easy
schema-based-primitive-viewer.directive.ts	37	Easy
collection-node-list.directive.ts	37	Easy
oppia-response-logic-proof.directive.ts	37	Easy
schema-based-html-editor.directive.ts	37	Easy
warning-loader.directive.ts	39	Easy
oppia-short-response-logic-proof.directive.ts	39	Easy
side-navigation-bar.directive.ts	39	Easy
focus-on.directive.ts	39	Easy
profile-page-navbar.directive.ts	39	Easy
circular-image.directive.ts	40	Easy
schema-based-custom-editor.directive.ts	40	Easy
graph-editor.directive.ts	40	Easy
collection-settings-tab.directive.ts	40	Easy
profile-link-text.directive.ts	41	Easy
oppia-short-response-item-selection-input.directive.ts	41	Easy
oppia-response-set-input.directive.ts	41	Easy
oppia-short-response-number-with-units.directive.ts	41	Easy
oppia-response-item-selection-input.directive.ts	41	Easy
oppia-response-text-input.directive.ts	41	Easy
oppia-short-response-fraction-input.directive.ts	41	Easy
schema-based-unicode-viewer.directive.ts	41	Easy
oppia-response-fraction-input.directive.ts	41	Easy
schema-based-custom-viewer.directive.ts	41	Easy
oppia-response-number-with-units.directive.ts	41	Easy
angular-html-bind.directive.ts	42	Easy
oppia-response-pencil-code-editor.directive.ts	42	Easy
mathjax-bind.directive.ts	42	Easy

schema-based-list-viewer.directive.ts	43	Easy
oppia-short-response-code-repl.directive.ts	43	Easy
playthrough-improvement-task.directive.ts	43	Easy
answer-details-improvement-task.directive.ts	43	Easy
state-param-changes-editor.directive.ts	43	Easy
attribution-guide.directive.ts	43	Easy
library-footer.directive.ts	43	Easy
oppia-response-multiple-choice-input.directive.ts	43	Easy
oppia-short-response-image-click-input.directive.ts	44	Easy
oppia-short-response-pencil-code-editor.directive.ts	44	Easy
oppia-response-math-expression-input.directive.ts	44	Easy
oppia-short-response-graph-input.directive.ts	44	Easy
oppia-response-continue.directive.ts	44	Easy
html-editor.directive.ts	44	Easy
oppia-short-response-text-input.directive.ts	44	Easy
topic-editor-navbar-breadcrumb.directive.ts	44	Easy
oppia-short-response-continue.directive.ts	44	Easy
oppia-short-response-numeric-input.directive.ts	45	Easy
oppia-response-numeric-input.directive.ts	45	Easy
background-banner.directive.ts	45	Easy
oppia-short-response-set-input.directive.ts	45	Easy
skill-editor-navbar-breadcrumb.directive.ts	45	Easy
oppia-noninteractive-tabs.directive.ts	45	Easy
sanitized-url-editor.directive.ts	45	Easy
oppia-noninteractive-math.directive.ts	45	Easy
oppia-short-response-interactive-map.directive.ts	46	Easy
exploration-objective-editor.directive.ts	46	Easy
oppia-short-response-math-expression-input.directive.ts	46	Easy
collection-footer.directive.ts	46	Easy
exploration-title-editor.directive.ts	46	Easy
oppia-response-drag-and-drop-sort-input.directive.ts	46	Easy
drag-and-drop-html-string-editor.directive.ts	47	Easy
oppia-short-response-drag-and-drop-sort-input.directive.ts	47	Easy
oppia-response-image-click-input.directive.ts	47	Easy
oppia-noninteractive-collapsible.directive.ts	47	Easy
oppia-visualization-frequency-table.directive.ts	48	Easy

collection-permissions-card.directive.ts	48	Easy
boolean-editor.directive.ts	48	Easy
real-editor.directive.ts	48	Easy
suggestion-improvement-task.directive.ts	48	Easy
schema-based-viewer.directive.ts	48	Easy
feedback-improvement-task.directive.ts	48	Easy
outcome-feedback-editor.directive.ts	49	Easy
collection-navbar.directive.ts	49	Easy
story-viewer-navbar-breadcrumb.directive.ts	49	Easy
schema-based-dict-viewer.directive.ts	49	Easy
int-editor.directive.ts	49	Easy
topic-summary-tile.directive.ts	49	Easy
schema-based-choices-editor.directive.ts	49	Easy
opportunities-list.directive.ts	49	Easy
list-of-unicode-string-editor.directive.ts	49	Easy
oppia-short-response-multiple-choice-input.directive.ts	50	Easy
summary-list-header.directive.ts	50	Easy
collection-local-nav.directive.ts	50	Easy
topic-viewer-navbar-breadcrumb.directive.ts	50	Easy
oppia-response-code-repl.directive.ts	50	Easy
story-summary-tile.directive.ts	51	Easy
drag-and-drop-positive-int-editor.directive.ts	51	Easy
html-select.directive.ts	52	Easy
nonnegative-int-editor.directive.ts	52	Easy
set-of-unicode-string-editor.directive.ts	52	Easy
topic-selector.directive.ts	53	Easy
oppia-short-response-music-notes-input.directive.ts	53	Easy
oppia-response-music-notes-input.directive.ts	53	Easy
schema-based-int-editor.directive.ts	53	Easy
subtopic-summary-tile.directive.ts	53	Easy
subtopic-viewer-navbar-breadcrumb.directive.ts	54	Easy
practice-tab.directive.ts	55	Easy
answer-submit-action.directive.ts	55	Easy
alert-message.directive.ts	55	Easy
copier.directive.ts	55	Easy
random-selector.directive.ts	55	Easy

value-generator-editor.directive.ts	56	Easy
thread-table.directive.ts	56	Easy
skill-rubrics-editor.directive.ts	57	Easy
oppia-response-graph-input.directive.ts	57	Easy
skill-editor-main-tab.directive.ts	58	Easy
object-editor.directive.ts	59	Easy
number-with-units-editor.directive.ts	59	Easy
code-string-editor.directive.ts	59	Easy
skill-selector-editor.directive.ts	60	Easy
schema-based-dict-editor.directive.ts	60	Easy
oppia-visualization-enumerated-frequency-table.directive.ts	61	Easy
codemirror-mergeview.directive.ts	61	Easy
skill-selector.directive.ts	61	Easy
login-required-message.directive.ts	61	Easy
profile-link-image.directive.ts	62	Easy
test-interaction-panel.directive.ts	63	Easy
schema-based-editor.directive.ts	63	Easy
collection-editor-tab.directive.ts	63	Easy
opportunities-list-item.directive.ts	66	Easy
set-of-html-string-editor.directive.ts	66	Easy
fraction-editor.directive.ts	67	Easy
graph-property-editor.directive.ts	67	Easy
learner-answer-info-card.directive.ts	67	Easy
oppia-visualization-bar-chart.directive.ts	68	Easy
parameter-name-editor.directive.ts	69	Easy
list-of-tabs-editor.directive.ts	71	Easy
collection-editor-navbar-breadcrumb.directive.ts	71	Easy
skill-description-editor.directive.ts	71	Easy
bar-chart.directive.ts	73	Easy
oppia-noninteractive-link.directive.ts	74	Easy
oppia-interactive-continue.directive.ts	75	Easy
pie-chart.directive.ts	75	Easy
score-ring.directive.ts	75	Easy
oppia-interactive-numeric-input.directive.ts	76	Easy
oppia-noninteractive-skillreview.directive.ts	76	Easy
activity-tiles-infinity-grid.directive.ts	76	Easy

concept-card.directive.ts	76	Easy
promo-bar.directive.ts	76	Easy
logic-error-category-editor.directive.ts	77	Easy
cyclic-transitions-issue.directive.ts	78	Easy
base-content.directive.ts	78	Easy
search-results.directive.ts	78	Easy
multiple-incorrect-issue.directive.ts	78	Easy
early-quit-issue.directive.ts	78	Easy
editor-navbar-breadcrumb.directive.ts	79	Easy
skill-mastery.directive.ts	79	Easy
exploration-footer.directive.ts	80	Easy
collection-editor-page.directive.ts	82	Easy
review-material-editor.directive.ts	83	Easy
story-editor-navbar-breadcrumb.directive.ts	84	Easy
math-latex-string-editor.directive.ts	84	Easy
oppia-response-interactive-map.directive.ts	84	Easy
skill-questions-tab.directive.ts	84	Easy
solution-editor.directive.ts	85	Easy
worked-example-editor.directive.ts	85	Easy
oppia-interactive-text-input.directive.ts	87	Easy
music-phrase-editor.directive.ts	88	Easy
normalized-string-editor.directive.ts	88	Easy
playthrough-issues.directive.ts	89	Easy
collection-node-editor.directive.ts	90	Easy
oppia-interactive-multiple-choice-input.directive.ts	90	Easy
unicode-string-editor.directive.ts	91	Easy
state-translation-status-graph.directive.ts	91	Easy
voiceover-opportunities.directive.ts	92	Easy
sharing-links.directive.ts	93	Easy
admin-page.directive.ts	94	Easy
solution-explanation-editor.directive.ts	95	Easy
improvements-tab.directive.ts	98	Easy
oppia-interactive-drag-and-drop-sort-input.directive.ts	98	Easy
coord-two-dim-editor.directive.ts	98	Easy
oppia-noninteractive-video.directive.ts	99	Easy
collection-summary-tile.directive.ts	99	Easy

response-header.directive.ts	99	Easy
hint-editor.directive.ts	102	Medium
multiple-incorrect-submissions-issue.directive.ts	103	Medium
rule-type-selector.directive.ts	103	Medium
audio-file-uploader.directive.ts	103	Medium
logic-question-editor.directive.ts	103	Medium
topics-list.directive.ts	106	Medium
rubrics-editor.directive.ts	106	Medium
role-graph.directive.ts	107	Medium
oppia-interactive-set-input.directive.ts	108	Medium
admin-config-tab.directive.ts	109	Medium
skills-mastery-list.directive.ts	110	Medium
rating-display.directive.ts	111	Medium
topic-questions-tab.directive.ts	111	Medium
admin-navbar.directive.ts	112	Medium
schema-based-float-editor.directive.ts	113	Medium
oppia-interactive-end-exploration.directive.ts	115	Medium
input-response-pair.directive.ts	115	Medium
topic-editor-stories-list.directive.ts	120	Medium
skill-prerequisite-skills-editor.directive.ts	120	Medium
select2-dropdown.directive.ts	120	Medium
image-uploader.directive.ts	120	Medium
review-test-page.directive.ts	121	Medium
oppia-noninteractive-image.directive.ts	125	Medium
oppia-interactive-number-with-units.directive.ts	126	Medium
training-panel.directive.ts	127	Medium
state-name-editor.directive.ts	129	Medium
oppia-interactive-item-selection-input.directive.ts	129	Medium
topics-and-skills-dashboard-navbar.directive.ts	130	Medium
state-content-editor.directive.ts	130	Medium
exploration-graph.directive.ts	135	Medium
list-of-sets-of-html-strings-editor.directive.ts	135	Medium
create-activity-button.directive.ts	136	Medium
supplemental-card.directive.ts	137	Medium
subtopics-list.directive.ts	141	Medium
schema-based-unicode-editor.directive.ts	142	Medium

topic-viewer-stories-list.directive.ts	142	Medium
admin-jobs-tab.directive.ts	144	Medium
state-editor.directive.ts	145	Medium
exploration-save-and-publish-buttons.directive.ts	145	Medium
collection-details-editor.directive.ts	146	Medium
oppia-interactive-graph-input.directive.ts	153	Medium
hint-and-solution-buttons.directive.ts	156	Medium
skill-editor-navbar.directive.ts	156	Medium
learner-local-nav.directive.ts	161	Medium
translator-overview.directive.ts	162	Medium
feedback-popup.directive.ts	162	Medium
topic-editor-tab.directive.ts	164	Medium
oppia-interactive-fraction-input.directive.ts	167	Medium
oppia-interactive-interactive-map.directive.ts	168	Medium
learner-dashboard-icons.directive.ts	168	Medium
skill-misconceptions-editor.directive.ts	169	Medium
story-editor-navbar.directive.ts	173	Medium
misconception-editor.directive.ts	174	Medium
progress-nav.directive.ts	174	Medium
outcome-destination-editor.directive.ts	180	Medium
learner-view-info.directive.ts	184	Medium
state-translation-editor.directive.ts	186	Medium
thumbnail-uploader.directive.ts	187	Medium
collection-node-creator.directive.ts	191	Medium
admin-misc-tab.directive.ts	193	Medium
oppia-interactive-pencil-code-editor.directive.ts	195	Medium
preview-tab.directive.ts	203	Medium
editor-navigation.directive.ts	203	Medium
skill-concept-card-editor.directive.ts	204	Medium
oppia-interactive-image-click-input.directive.ts	211	Medium
translation-opportunities.directive.ts	212	Medium
admin-dev-mode-activities-tab.directive.ts	212	Medium
question-opportunities.directive.ts	217	Medium
skills-list.directive.ts	221	Medium
exploration-summary-tile.directive.ts	222	Medium
topic-editor-navbar.directive.ts	222	Medium

oppia-interactive-math-expression-input.directive.ts	233	Medium
outcome-editor.directive.ts	233	Medium
schema-based-list-editor.directive.ts	234	Medium
question-editor.directive.ts	235	Medium
story-editor.directive.ts	244	Medium
ck-editor-4-rte.directive.ts	248	Medium
tutor-card.directive.ts	255	Hard
feedback-tab.directive.ts	259	Hard
oppia-interactive-code-repl.directive.ts	266	Hard
subtopics-list-tab.directive.ts	267	Hard
param-changes-editor.directive.ts	269	Hard
unresolved-answers-overview.directive.ts	272	Hard
rule-editor.directive.ts	276	Hard
exploration-editor-tab.directive.ts	276	Hard
search-bar.directive.ts	286	Hard
statistics-tab.directive.ts	299	Hard
audio-bar.directive.ts	301	Hard
collection-editor-navbar.directive.ts	305	Hard
state-hints-editor.directive.ts	306	Hard
top-navigation-bar.directive.ts	310	Hard
state-solution-editor.directive.ts	310	Hard
oppia-interactive-logic-proof.directive.ts	315	Hard
story-viewer-page.directive.ts	321	Hard
history-tab.directive.ts	328	Hard
version-diff-visualization.directive.ts	342	Hard
translation-tab.directive.ts	347	Hard
admin-roles-tab.directive.ts	366	Hard
collection-player-page.directive.ts	369	Hard
story-node-editor.directive.ts	371	Hard
library-page.directive.ts	372	Hard
state-translation.directive.ts	384	Hard
answer-group-editor.directive.ts	407	Hard
contributions-and-review.directive.ts	418	Hard
state-graph-visualization.directive.ts	421	Hard
state-interaction-editor.directive.ts	475	Hard
settings-tab.directive.ts	513	Hard

image-with-regions-editor.directive.ts	545	Hard
graph-viz.directive.ts	597	Hard
question-player.directive.ts	598	Hard
audio-translation-bar.directive.ts	644	Hard
state-responses.directive.ts	679	Hard
questions-list.directive.ts	792	Hard
oppia-interactive-music-notes-input.directive.ts	849	Hard
filepath-editor.directive.ts	849	Hard
conversation-skin.directive.ts	1308	Hard

Controllers

Inline controllers issue

The solution to fix the inline controllers is to move the inline controllers to new files, turning possible to test it property and to calculate the controllers coverage correctly. An issue was created for fixing it: [#8924](#).

```

story-creation-modal.controller.ts •
oppia > core > templates > components > entity-creation-services > story-creation-mo
1  angular.module('oppia').controller('NewController', [
2    '$scope', '$uibModalInstance',
3    function($scope, $uibModalInstance) {
4      $scope.storyTitle = '';
5      $scope.isStoryTitleEmpty = function($storyTitle) {
6        return (storyTitle === '');
7      };
8      $scope.save = function($storyTitle) {
9        $uibModalInstance.close(storyTitle);
10     };
11     $scope.cancel = function() {
12       $uibModalInstance.dismiss('cancel');
13     };
14   }
15   ]);

```



```
var modalInstance = $uibModal.open({
  templateUrl: UrlInterpolationService.getDirectiveTemplateUrl(
    '/pages/topic-editor-page/modal-templates/' +
    'new-story-title-editor.template.html'),
  backdrop: true,
  controller: 'NewController'
});
```

When inline controller uses variables from the external scope (from the file itself or the function where is being called), `resolve` property must be used to pass these variables as local in the new controller:

```
showModal: function(explorationId) {
  $uibModal.open({
    backdrop: true,
    templateUrl: UrlInterpolationService.getDirectiveTemplateUrl(
      '/components/button-directives/' +
      'exploration-embed-button.directive.html'),
    resolve: {
      explorationId: function() {
        return explorationId;
      }
    },
    controller: [
      '$scope', '$uibModalInstance', '$window', 'explorationId',
      function($scope, $uibModalInstance, $window, explorationId) {
        $scope.explorationId = explorationId;
      }
    ]
  });
}
```

The `resolve` property can be used to execute code that is important to controller work but it's responsibility is only to receive the return value, like checking if the user is logged:

```
resolve: {
  messages: ThreadDataService.getMessagesAsync(thread),
  isUserLoggedIn:
    UserService.getUserInfoAsync().then(u => u.isLoggedIn()),
  thread: () => thread
},
controller: 'ImprovementFeedbackThreadModalController',
```



```
angular.module('oppia').controller('ImprovementFeedbackThreadModalController', [
  '$scope', '$uibModalInstance', 'AlertsService', 'DateTimeFormatService',
  'EditabilityService', 'ThreadDataService', 'ThreadStatusDisplayService',
  'isUserLoggedIn', 'thread',
  function(
    $scope, $uibModalInstance, AlertsService, DateTimeFormatService,
    EditabilityService, ThreadDataService, ThreadStatusDisplayService,
    isUserLoggedIn, thread) {
```

Creating common controllers to be reusable

When creating a list of inline controllers, a bunch of them have just two actions: close and dismiss. For example:

```
controller: [
  '$scope', '$uibModalInstance',
  function($scope, $uibModalInstance) {
    $scope.continueToSolution = function() {
      $uibModalInstance.close();
    };

    $scope.cancel = function() {
      $uibModalInstance.dismiss('cancel');
    };
  }
]
```

hint-and-solution-modal.service.ts

```
controller: [
  '$scope', '$uibModalInstance',
  function($scope, $uibModalInstance) {
    $scope.cancel = function() {
      $uibModalInstance.dismiss('cancel');
    };

    $scope.close = function() {
      $uibModalInstance.close();
    };
  }
]
```

question-opportunity.directive.ts

```

controller: ['$scope', '$uibModalInstance', function(
  $scope, $uibModalInstance) {
    $scope.markNeedsUpdate = function() {
      $uibModalInstance.close();
    };

    $scope.cancel = function() {
      $uibModalInstance.dismiss('cancel');
    };
  }]

```

state-translation-editor.directive.ts

In this case, a generic controller with a generic name like `ConfirmCancelModalController` will be created and all modals will use the same controller. It will bring some benefits:

- Reduce code repetition.
- Write less tests instead of copy and paste the same code many times.

Some controllers follows the same logic but there is something different that makes it not completely equal:

- Calling services after dismiss (same for close):

```

controller: [
  '$scope', '$uibModalInstance', function(
    $scope, $uibModalInstance) {
    $scope.reallyDelete = function() {
      $uibModalInstance.close();
    };

    $scope.cancel = function() {
      $uibModalInstance.dismiss('cancel');
      AlertsService.clearWarnings();
    };
  }
]

```

Changes to:

```

controller: [
  '$scope', '$uibModalInstance', function(
    $scope, $uibModalInstance) {
    $scope.reallyDelete = function() {
      $uibModalInstance.close();
    };

    $scope.cancel = function() {
      $uibModalInstance.dismiss('cancel');
    };
  }
]
}).result.catch(function() {
  AlertsService.clearWarnings();
});

```

In this case, it won't need a new file for the controller, because it is going to be replaced by the generic controller.

- When there is more code than just dismiss and close actions:

```

controller: [
  '$scope', '$uibModalInstance',
  function($scope, $uibModalInstance) {
    $scope.storyTitle = '';
    $scope.isStoryTitleEmpty = function($storyTitle) {
      return (storyTitle === '');
    };
    $scope.save = function($storyTitle) {
      $uibModalInstance.close(storyTitle);
    };
    $scope.cancel = function() {
      $uibModalInstance.dismiss('cancel');
    };
  }
]

```

Changes to:

```

controller: [
  '$scope', '$uibModalInstance', '$controller',
  function($scope, $uibModalInstance, $controller) {
    $controller('ConfirmCancelModalController', {
      $scope: $scope,
      $uibModalInstance: $uibModalInstance
    });

    $scope.storyTitle = '';
    $scope.isStoryTitleEmpty = function(storyTitle) {
      return (storyTitle === '');
    };
  }
]

```

In this case, a new file will be created to test the other functions or evaluate the scope variables value.

Here's the list of controllers with the same logic. There's two types of fixing it:

- Replace: It's replaced by the generic controller.
- Inherit: It inherits the functions from the generic controller.

Filename	Inline controllers	Type
learner-local-nav.directive.ts	2	Replace and Inherit
oppia-noninteractive-skillreview.directive.ts	1	Inherit
oppia-interactive-pencil-code-editor.directive.ts	1	Replace
oppia-interactive-number-with-units.directive.ts	1	Replace
oppia-interactive-logic-proof.directive.ts	1	Replace
topics-list.directive.ts	1	Replace
exploration-embed-button.service.ts	1	Inherit
story-creation.service.ts	1	Inherit
topic-editor-tab.directive.ts	1	Replace
skills-mastery-list.directive.ts	1	Inherit
story-editor-navbar.directive.ts	1	Replace
story-editor-navbar-breadcrumb.directive.ts	1	Replace
state-responses.directive.ts	1	Replace
story-editor.directive.ts	1	Replace
story-editor-page.controller.ts	1	Replace
state-solution-editor.directive.ts	1	Replace

skill-prerequisite-skills-editor.directive.ts	1	Inherit
skill-concept-card-editor.directive.ts	1	Replace
signup-page.controller.ts	1	Inherit
preferences-page.controller.ts	1	Inherit
suggestion-modal-for-exploration-player.service.ts	1	Replace
hint-and-solution-modal.service.ts	1	Replace
fatigue-detection.service.ts	1	Replace
image-with-regions-editor.directive.ts	1	Replace
state-translation-editor.directive.ts	1	Replace
collection-editor-navbar.directive.ts	1	Inherit
question-opportunities.directive.ts	1	Replace
exploration-states.service.ts	1	Inherit
translation-opportunities.directive.ts	1	Inherit
autosave-info-modals.service.ts	1	Inherit
preview-tab.directive.ts	1	Inherit
history-tab.directive.ts	1	Inherit
exploration-editor-page.controller.ts	1	Inherit
delete-account-page.controller.ts	1	Replace
create-activity-button.directive.ts	1	Inherit
state-hints-editor.directive.ts	2	Replace and Inherit
audio-translation-bar.directive.ts	2	Replace and Inherit
skill-editor-navbar.directive.ts	2	Replace
story-node-editor.directive.ts	2	Inherit
state-interaction-editor.directive.ts	2	Replace
skills-list.directive.ts	2	Replace and Inherit
topic-editor-stories-list.directive.ts	2	Replace
topic-editor-navbar.directive.ts	2	Replace and Inherit
settings-tab.directive.ts	3	2 Replace and 1 Inherit
exploration-save.service.ts	3	2 Replace and 1 Inherit
questions-list.directive.ts	5	3 Replace and 2 Inherit
Total of replaces		37
Total of Inherit		26

There are a total of 37 replaces, it means that 37 controllers will be replaced by a generic one. When talking about inheritance, there are 26 controllers. Calculating all these number, we can get the total of controllers that need to be tested:

105 (total inline controllers) - 37 (replaces controllers) + 1(the generic controller) = **69 inline controllers to be tested.**

It's important to notice that these 69 inline controllers will be considered as medium level criteria. Each modal has its own complex logic, so the controllers have a lot of lines.

Name convention for inline controllers

As many files have more than one controller attached, naming the controller with the same file name (but finishing with `.controller.ts` extension) is not the best approach for most cases, and it should be used only in specific situations.

The best approach is: all the inline controllers will follow the file name of `templateUrl` property value. Example:

```

var modalInstance = $uibModal.open({
  templateUrl: UrlInterpolationService.getDirectiveTemplateUrl(
    '/components/question-directives/modal-templates/' +
    'change-question-difficulty-modal.template.html'),
  backdrop: true,
  controller: [
    '$scope', '$uibModalInstance',
    function($scope, $uibModalInstance) {
      var init = function() {
        $scope.linkedSkillsWithDifficulty =
          linkedSkillsWithDifficulty;
        $scope.skillIdToRubricsObject = skillIdToRubricsObject;
      };

      $scope.cancelModal = function() {
        $uibModalInstance.dismiss('cancel');
      };

      $scope.done = function() {
        $uibModalInstance.close($scope.linkedSkillsWithDifficulty);
      };

      init();
    }
  ]
});

```

When moving the controller to a new file, its name should be `change-question-difficulty-modal.controller.ts`.

However, if there are two controllers (with different logic) pointing to the same template, the name should have the method name where `$uibModal.open` is being called or the file name itself, finishing it with `*-modal.controller.ts`.

Controllers described as directive

In spite of 37 controllers, **29 of them are described as directive**. Here's an example, using `about-page.controller.ts`:

```
angular.module('oppia').directive('aboutPage', [
  'UrlInterpolationService', function(UrlInterpolationService) {
    return {
      restrict: 'E',
```

All the controllers described as directive uses `restrict: 'E'` as property. It means that the controller is actually a component, so it'll be changed using the same logic used in the [directive section](#).

Controller testing list

There are 91 controllers to be tested. The 22 are listed below and the other 69 controllers are inline ones (listed in the previous table).

Filename	Lines	Level
maintenance-page.controller.ts	38	Easy
thanks-page.controller.ts	43	Easy
error-page.controller.ts	50	Easy
skill-editor-page.controller.ts	60	Easy
delete-account-page.controller.ts	64	Easy
donate-page.controller.ts	69	Easy
topic-editor-page.controller.ts	74	Easy
classroom-page.controller.ts	75	Easy
exploration-player-page.controller.ts	76	Easy
subtopic-viewer-page.controller.ts	81	Easy
topic-viewer-page.controller.ts	92	Easy
story-editor-page.controller.ts	100	Easy
practice-session-page.controller.ts	116	Medium
community-dashboard-page.controller.ts	165	Medium
topics-and-skills-dashboard-page.controller.ts	203	Medium
profile-page.controller.ts	222	Medium
signup-page.controller.ts	240	Medium
preferences-page.controller.ts	299	Hard
logic-demo-test.controller.ts	308	Hard
creator-dashboard-page.controller.ts	397	Hard
learner-dashboard-page.controller.ts	628	Hard
exploration-editor-page.controller.ts	656	Hard

Testing Approach

Milestones

Setup milestone

Key Objective: Prepare all the files to be tested.

Starts: May 4th.

Ends: May 31th.

There are many files to fix and test during 3 months. However, to test the files properly and to have the correct coverage, some tasks must be done before the internship starts on June 1st.

Tasks:

- Fix inline controllers issue.
 - Create a generic controller and use it on common inline controllers.
 - Replace 37 inline controllers.
 - Inherit 26 controllers.
 - Create a new file for 42 left inline controllers.
- Change directives to components.
 - Change 29 controllers described as directives to components.
 - Change 301 directives with `restrict: 'E'` to components.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
1.1	Create a generic controller and replace the inline controllers.	-	05/05/2020	05/08/2020
1.2	Inherit the generic controller in files.	1.1	05/09/2020	05/13/2020
1.3	Create new files for left inline controllers.	-	05/10/2020	05/14/2020
1.4	Change controllers described as directives to components.	-	05/13/2020	05/18/2020
1.5	Change directives with <code>restrict: 'E'</code> to	-	05/25/2020	05/31/2020

	components.			
--	-------------	--	--	--

Milestone 1

Key Objective: Achieve 100% coverage of non-inline controllers and fully covered 33 out of 69 inline controllers.

Starts: June 1st.

Ends: June 29th.

Tasks:

- Work on 55 controllers following the level criteria.
 - Easy: 12 controllers.
 - Medium: 5 controllers.
 - Hard: 5 controllers.
 - Work on 36 controllers that were inline before (medium level criteria).

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
1.1	Work on easy controllers (4 files)	-	06/01/2020	06/04/2020
1.2	Work on easy controllers (4 files)	-	06/02/2020	06/05/2020
1.3	Work on easy controllers (4 files)	-	06/03/2020	06/06/2020
1.5	Work on medium controllers (3 files)	-	06/04/2020	06/07/2020
1.6	Work on medium controllers (2 files)	-	06/05/2020	06/09/2020
1.7	Work on hard controllers (3 files)	-	06/08/2020	06/12/2020
1.8	Work on hard controllers (2 files)	-	06/10/2020	06/14/2020
1.9	Work on inline controllers. (3 files).	-	06/11/2020	06/14/2020
1.10	Work on inline controllers. (3 files).	-	06/12/2020	06/15/2020
1.11	Work on inline controllers. (3 files).	-	06/14/2020	06/17/2020
1.12	Work on inline controllers. (3 files).	-	06/15/2020	06/18/2020

1.13	Work on inline controllers. (3 files).	-	06/17/2020	06/20/2020
1.14	Work on inline controllers. (3 files).	-	06/18/2020	06/21/2020
1.15	Work on inline controllers. (3 files).	-	06/20/2020	06/23/2020
1.16	Work on inline controllers. (3 files).	-	06/21/2020	06/24/2020
1.17	Work on inline controllers. (3 files).	-	06/23/2020	06/26/2020
1.18	Work on inline controllers. (3 files).	-	06/24/2020	06/27/2020
1.19	Work on inline controllers. (3 files).	-	06/26/2020	06/29/2020
1.20	Work on inline controllers. (3 files).	-	06/27/2020	06/30/2020

Milestone 2

Key Objective: Achieve 100% coverage of inline controllers and 87.5% files of `/core/templates/pages/exploration-editor-page` folder.

This folder was chosen because it's a large folder with many files. Which means that a significant part of coverage will be increased testing it.

Starts: June 30th.

Ends: July 28th.

Tasks:

- Work on 33 controllers that were inline before (medium level criteria).
- Work on 35 directives that are distributed as:
 - 20 easy directives.
 - 9 medium directives.
 - 6 hard directives.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
2.1	Work on inline controllers. (3 files).	-	06/28/2020	07/01/2020
2.2	Work on inline controllers. (3 files).	-	06/29/2020	07/02/2020
2.3	Work on inline controllers. (3 files).	-	07/01/2020	07/04/2020

2.4	Work on inline controllers. (3 files).	-	07/02/2020	07/05/2020
2.5	Work on inline controllers. (3 files).	-	07/04/2020	07/07/2020
2.6	Work on inline controllers. (3 files).	-	07/05/2020	07/08/2020
2.7	Work on inline controllers. (3 files).	-	07/07/2020	07/10/2020
2.8	Work on inline controllers. (3 files).	-	07/08/2020	07/11/2020
2.9	Work on inline controllers. (3 files).	-	07/10/2020	07/13/2020
2.10	Work on inline controllers. (3 files).	-	07/11/2020	07/14/2020
2.11	Work on inline controllers. (3 files).	-	07/13/2020	07/16/2020
2.12	Work on easy directives (4 files)	-	07/14/2020	07/17/2020
2.13	Work on easy directives (4 files)	-	07/15/2020	07/18/2020
2.14	Work on easy directives (4 files)	-	07/16/2020	07/19/2020
2.15	Work on easy directives (4 files)	-	07/17/2020	07/20/2020
2.16	Work on easy directives (4 files)	-	07/18/2020	07/21/2020
2.17	Work on medium directives (3 files)	-	07/19/2020	07/22/2020
2.18	Work on medium directives (3 files)	-	07/20/2020	07/23/2020
2.19	Work on medium directives (3 files)	-	07/22/2020	07/25/2020
2.20	Work on hard directives (3 files)	-	07/25/2020	07/28/2020
2.21	Work on hard directives (3 files)	-	07/28/2020	07/31/2020

Milestone 3

Key Objective: Achieve 100% on some `/core/templates/pages` folders:

- `/exploration-editor-page`.
- `/subtopic-viewer-page`.
- `/profile-page`.
- `/classroom-page`.
- `/library-page`.
- `/story-viewer-page`.
- `/community-dashboard-page`.

- /review-test-page.
- /learner-dashboard-page.

This milestone aims to achieve 100% coverage on the /exploration-editor-page folder which wasn't possible to complete in the previous milestone. The other folders were selected because it has a few files that won't take much time to test. The last week of the internship will be regarded as a buffer week, in order to fix possible remaining work and things that may go wrong.

Starts: July 29th.

Ends: August 24th.

Tasks:

- Work on 36 directives is distributed as:
 - 20 easy directives.
 - 6 medium directives.
 - 10 hard directives.

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
3.1	Work on hard directives (3 files)	-	07/31/2020	08/04/2020
3.2	Work on hard directives (2 files)	-	08/02/2020	08/05/2020
3.3	Work on easy directives (4 files)	-	08/03/2020	08/06/2020
3.4	Work on easy directives (4 files)	-	08/05/2020	08/08/2020
3.5	Work on easy directives (4 files)	-	08/06/2020	08/09/2020
3.6	Work on easy directives (4 files)	-	08/07/2020	08/10/2020
3.7	Work on easy directives (4 files)	-	08/08/2020	08/11/2020
3.8	Work on medium directives (2 files)	-	08/09/2020	08/13/2020
3.9	Work on medium directives (2 files)	-	08/10/2020	08/13/2020
3.10	Work on medium directives (2 files)	-	08/11/2020	08/14/2020
3.11	Work on hard directives (3 files)	-	08/14/2020	08/18/2020

3.12	Work on hard directives (2 file)	-	08/16/2020	08/19/2020
3.13	Work on remaining work and things that may go wrong.	-	08/17/2020	08/24/2020

Future Work

For future work, directives should reach 100% of coverage. It can be achieved by other contributors helping to test the missing directives. An issue like [#4057](#) will be created listing all the missing directives, separating them by folder name and level criteria.

Additional Project-Specific Considerations

Documentation Changes

Since it's not possible to test all directives during the internship, the frontend tests wiki page will be changed to include guides of how to properly test all levels of directives (easy, medium and hard). This guide will be written during the internship while I learn new patterns and the best ways to test specific scenarios.

A guide about controllers will be written as well, so if controllers have new changes, the contributor will know how to test these new changes following the same pattern.

The guide will contain:

- How to set up the test (with `beforeEach` block).
- Best practices for testing both controllers and directives.
- How to handle common errors (if any) when testing controllers and directives.