GSoC 2021 Proposal

Move custom JS\TS lint checks to eslint

(Aditya Dubey)

About You

I am Aditya Dubey and I'm pursuing a Bachelor of technology in Computer Science and Engineering from Sister Nivedita University. I'm currently in my 2nd year. I've always been fascinated by open source and its interesting organizational structure. I started my open-source contribution journey with Oppia foundation. I've been contributing to the Oppia foundation for 6+ months and I'm a member of the Oppia project.

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

Oppia's mission is to provide free education and to help anyone learn anything they want in an effective and enjoyable way. I resonate with the vision of the organisation and I am glad to be a part of this community. On the other hand, I get a chance to work with talented and experienced mentors/contributors from around the world.

I choose this project because it helps developers to follow the organization's coding principles. As in this project one had to know learn the best practices followed in the industry for JavaScript, and protractor. I've also worked on issues related to adding new lint checks and being a part of Oppia's linter team, the experience I've gained helped me understand the structure of the linter project.

Prior experience

I started contributing code to Oppia in October of 2020. Currently, I'm part of the linter team in the oppia project and I've recently been added as a member of the Oppia organisation. In the last 6 months I have fixed a number of issues and most of them are related to the Linter project.

My contributions:

Merged PRs (related to linter project):

1. **[New check in eslint]** Fix #12389: Disallow usage of xdescribe and xit in frontend unit test

2. **[New check in eslint]** Fix part of #11496: Adds lint check to disallow browser.sleep() calls
3. **[Linter optimization]** Fix part of #12117: Remove loop from html linter file
4. **[Debugging]** Fix #11070: empty docstring linter error
5. **[Debugging]** Fix css linter script issue
6. **[Enables new css built-in check]** Fix part of #5097: Add length-zero-no-unit rule for CSS lint check
7. **[New check for CODEOWNER file]** Fix #9887: Adds inline comment checker for code owner

Merged PRs (Other):

1. **[Refactoring]** Fix #10349: Move getter methods from topic_services to topic_fetchers
2. **[Remove duplicate codes]** Fix #8174: Remove duplicate DIRS_TO_ADD_TO_SYS_PATH from the code base
3. **[Minor UI fixes]** Fix #11105: Adds correct styling in the audio player to maintain its height
4. **[Minor refactor]** Fix part of #10306: Adds async keyword to asynchronous functions.

Contact info and timezone(s)

Country: India
Email: 74500dubey@gmail.com
Github: AdityaDubey0
Timezone: Indian Standard Time (IST)
Preferred method of communication: Hangouts and email (74500dubey@gmail.com)

Time commitment

I am committed to spending 4 hours a day on this project on weekdays (Monday to Saturday). The time I devote to the project on Sunday will depend upon the work to be completed in the project in that week. In total, I am committed to working about 24 hours a week.

Essential Prerequisites

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

```
----------------------------------------
Tasks still running:
  scripts.linters.python_linter_test (started 09:35:16)
----------------------------------------
04:05:32 FINISHED scripts.linters.python_linter_test: 15.8 secs

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

SUCCESS   scripts.linters.python_linter_test: 16 tests (3.4 secs)

Ran 16 tests in 1 test class.
All tests passed.

Done!
```

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

```
Chrome Headless 89.0.4389.90 (Linux x86_64): Executed 4468 of 4469 SUCCESS (0 se
cs / 1 min 15.021 secs)
LOG: 'Spec: InteractionSpecsService checking whether an interaction can be train
Chrome Headless 89.0.4389.90 (Linux x86_64): Executed 4469 of 4469 SUCCESS (0 se
Chrome Headless 89.0.4389.90 (Linux x86_64): Executed 4469 of 4469 SUCCESS (1 mi
n 26.297 secs / 1 min 15.025 secs)
TOTAL: 4469 SUCCESS
TOTAL: 4469 SUCCESS
12 04 2021 09:57:25.811:WARN [launcher]: ChromeHeadless was not killed in 2000 m
s, sending SIGKILL.
Done!
```

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

```
Servers have come up.
Note: If ADD_SCREENSHOT_REPORTER is set to true in core/tests/protractor.conf.js
, you can view screenshots of the failed tests in ../protractor-screenshots/
[10:15:30] I/launcher - Running 1 instances of WebDriver
[10:15:30] I/hosted - Using the selenium server at http://localhost:4444/wd/hub
Started
Jasmine started
.
  Profile menu flow

    profile dropdown menu
      ? should visit the profile page from the profile dropdown menu




Ran 1 of 8 specs
1 spec, 0 failures
Finished in 51.289 seconds

Executed 1 of 8 specs INCOMPLETE (7 SKIPPED) in 51 secs.
```

Other summer obligations

I have no other obligations during this summer.

Communication channels

I am comfortable with any mode of communication that the mentor chooses, be it email or hangouts.

Application to other orgs

I'm only applying to **Oppia** and only to this project

Project Details

The Oppia development workflow uses lint checks to help detect style errors and helps developers follow the organization's coding principles.

Currently, the lint checkers for js/ts (javascript/typescript) files are scattered in multiple places:

1. in scripts/linters/js_ts_linter.py file written in python,
2. in scripts/linters/general_purpose_linter.py file with list of bad-regex-pattern for js/ts files,

the built-in eslint checks and the custom eslint checks inside scripts/linters/custom_eslint_checks/rules

It makes it harder for the dev-workflow team to maintain all these linters in different places and having multiple scripts to run all these different checkers. At the same time, developers use built-in/integrated ESLint linters in their IDE (e.g, VsCode, Sublime etc.) but the linters in IDE don't catch all other custom linters written in python.

This project aims to migrate all the JS\TS checks written in python using raw parsing algorithms and regex pattern checkers to custom eslint checks. The migration is going to affect developer experience in following ways:

1. Eslint will hold all the checks related to js/ts script in the codebase.
    a. This will reduce the number of processes to run js/ts related linter checks to one.
2. Enables developers to run all the js/ts lint checks through IDE.
3. Inforce developers to write new lint checks using one and only structure i.e, eslint custom checkers.

Product Design

Product design for four newly added lint checks:

1. Do not call .first(), .last(), or .get(i) calls on ElementArrayFinder object
    a. **Pattern to detect:**

```
// Unexpected .first() call below.
editiorCategoryDropdown.first();
...
...
```

    b. **User facing message:** Please use a specific classname selector instead of calling .first().

2. All root element selectors use HTML classes that start with "protractor-test-
    a. **Pattern to detect:**

```
// Unexpected classname for element selector.
element(by.css('.unexpected-class-name-pattern'));

...
...
```

    b. **User facing message:** Please use “protractor-test-” prefix classname selector instead of “<incorrect-classname>”

3. Keeping element selectors at the tops of files
    a. **Pattern to detect:**

```
var SomeModule = function() {
 var someElement = element(
   by.css('.protractor-test-some-container'));

 var clickSomeButton = function() {
   var someButton = element(
     by.css('.protractor-test-some-button'));
   actions.click(someButton);;
 }
}

exports.SomeModule = SomeModule;
```

    b. **User facing message:** Please declare the “someButton” element in the topmost scope of the SomeModule function.

4. Constant variable names are in all-caps in protractor files
    a. **Pattern to detect:**

```
const protractor = 'someValue';
```

    b. **User facing message:** Please make sure constant names are in all-caps.

Technical Design

Architectural Overview

This project is not going to introduce any new architecture in the codebase.

Implementation Approach

The project can be divided into three parts:

1. Migrate all of the BAD_PATTERNS_JS_AND_TS_REGEXP checks to custom eslint checks from general_purpose_linter.py.
2. Migrate all of the JS\TS lint checks from js_ts_linter.py to custom Eslint checks.
3. Implement E2E lint checks from #8423

Sub-project 1: Migrate all of the BAD_PATTERNS_JS_AND_TS_REGEXP checks to custom eslint checks from general_purpose_linter.py.

There are sum-total 19 checks inside BAD_PATTERNS_JS_AND_TS_REGEXP. All the details including pseudo code to implement all such checks in eslint are as following:

1. Disallow browser.explore() call
    a. **Detail:** Disallow usage of browser.explore() as this method requires the control flow in protractor.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** protractor-practices
    d. **Pseudo algorithm:**

```
Selector: 'CallExpression[callee.object.name=browser]'
   '[callee.property.name=explore]': function(node) {
       Report(
           'Please do not use browser.explore() '
           'in tests.')
```

```
    }
```

2. Disallow browser.pause() call.
    a. **Detail:** Disallow usage of browser.pause() as this method requires the control flow in the protractor.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** protractor-practices
    d. **Pseudo algorithm:**

```
Selector: 'CallExpression[callee.object.name=browser]'
    '[callee.property.name=pause]': function(node){
        Report('Please do not use browser.explore() '
            'in tests.')
    }
```

3. Disallow browser.waitForAngular().
    a. **Detail**: Disallow usage of browser.waitForAngular() as this method requires the control flow in protractor.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** protractor-practices
    d. **Pseudo algorithm:**

```
Selector: 'CallExpression[callee.object.name=browser]'
    '[callee.property.name=waitForAngular]':
function(node){
        Report(
            'Please do not use browser.waitForAngular()'
            'in tests')
    }
```

4. Disallow usage of word “bypass”
    a. **Detail:** Disallow usage of word “bypass” for variables/function/class etc. i.e, all identifiers.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** disallow-identifier-phrase
    d. **Options:**
        i. Disallowed-names: List of disallowed names in lower-letters.
    e. **Pseudo algorithm:**

```
Selector: Identifier: function(node) {
    if(node.name.toLower() in options.disallowed-params) {
```

```
        Report ('Please do not use word <node.name>.')
    }
}
```

5. Disallow “xdescribe” & “fdescribe”
    a. **Details:** Disallow usage of “xdescribe” & “fdescribe” as it blocks one or more parts of unit tests.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** no-test-blockers
    d. **Pseudo algorithm:**

```
Selector:
'CallExpression[callee.name=/^(xdescribe|fdescribe)$/]':
function(node) {
    Report(`Please use "describe" instead of
'${node.callee.name}'.`)
}
```

Note: Currently, we don’t disallow xdescribe instead we disallow ddescribe which is deprecated. (ref). I’ve changed the requirement accordingly!

6. Disallow ‘iit’ & ‘fit’
    a. **Details:** Disallow usage of “iit” & “fit” as it blocks one or more parts of unit tests.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** no-test-blockers
    d. **Pseudo algorithm:**

```
Selector: 'CallExpression[callee.name=/^(iit|fit)$/]':
function(node) {
    Report `Please use "it" instead of
'${node.callee.name}'.`
}
```

7. Disallow calling ‘inject’
    a. **Details:** Disallow usage of ‘inject’ function.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** use-angular-mock-inject
    d. **Pseudo algorithm:**

```
```

```
Selector: 'CallExpression[callee.name=inject]':
function(node) {
   Report('Please use “angular.mock.inject”  instead of
“inject”.')
}
```

8. Disallow calling ‘toThrow’
    a. **Details:** Disallow usage ‘toThrow’ function.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** incomplete-throw
    d. **Pseudo algorithm:**

```
Selector: 'CallExpression[callee.name=toThrow]':
function(node) {
   Report ('Please use “toThrowError”  instead of
“toThrow”.')
}
```

9. Disallow usage of ‘throw’ statement
    a. **Details:** Disallow usage of ‘throw’ statement
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** incomplete-throw
    d. **Pseudo algorithm:**

```
Selector: 'ThrowStatement[argument.type!=NewExpression]':
function(node) {
   Report('Please use throw new instead of throw')
}
```

10. Disallow usage of ‘throw’ statement’
    a. **Details:** Disallow usage of ‘throw’ statement.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** incomplete-throw
    d. **Pseudo algorithm:**

```
Selector:
'ThrowStatement[argument.type=NewExpression][argument.cal
lee.name!=Error]': function(node) {
   Report ('Please use "throw new Error" instead of
"throw"')
}
```

11. Disallow access $parent property
    a. **Details:** Disallow access of parent property of a scope.
    b. **Parser:** AST (Abstract syntax tree)
    c. **checker-name:** no-parent-access
    d. **Pseudo algorithm:**

```
Selector: 'MemberExpression[property.name=$parent]':
function(node) {
   Report ('Please do not access parent properties using
$parent. Use the scope object for this purpose.')
}
```

12. Disallow use of relative imports
    a. **Details:** Disallow use relative imports in require().
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** disallow-relative-imports
    d. **Pseudo algorithm:**

```
Selector:
'CallExpression[callee.name=require][arguments.length=1]'
: function(node) {
   if(node.arguments[0].type === 'Literal' &&
node.arguments[0].raw.startsWith('.')) {
     Report ('Please don\'t use relative imports in
require().')
   }
 }
```

13. Disallow uses of word ‘innerHTML’
    a. **Details:** Disallow usage of word ‘innerHTML’ for identifiers
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** disallow-innerHTML-property
    d. **Pseudo algorithm:**

```
Selector: 'MemberExpression[property.name=innerHTML]':
function(node){
   Report('Please do not use innerHTML property.')
}
```

14. Disallow eslint-(enable|disable) camelCase
    a. **Details:** Disallow eslint enable and es-lint disable for camelCase
    b. **Parser:** Token

c. **Checker-name:** disallow-comments
d. **Pseudo algorithm:**

```
Selector: 'Program': function(node) {
   var sourceCode = context.getSourceCode();
   var Comments =  sourceCode.getAllComments();
   var regex = /^ eslint-(enable|disable) camelcase/;
   For each comments
       if(regex.test(Comments[0].value)) {
       Report (
           'Please do not use eslint enable|disable
           'for camelcase. If you are using this '
           'statement to define properties in an '
           'interface for a backend dict. Wrap '
           'the property name in single quotes instead.')
       }
}
```

15. Disallow no-explicit-any

a. **Details:** Disallow 'no-explicit-any' types in comments
b. **Parser:** Token
c. **Checker-name:** disallow-comments
d. **Pseudo algorithm:**

```
Selector: 'Program': function(node) {
   var sourceCode = context.getSourceCode();
   var Comments =  sourceCode.getAllComments();
   var regex = /no-explicit-any/;
   For each comments
       if(regex.test(Comments[0].value)) {
           Report (
               'Please do not define "any" types.
               'You can refer <guide-link> if '
               'you\'re having trouble declaring types.')
       }
}
```

16. Disallow uses of word $broadcast

a. **Details:** Disallow usage of word '$broadcast' for identifiers
b. **Parser:** AST (Abstract syntax tree)

c. **Checker-name:** disallow-$broadcast-$on
d. **Pseudo algorithm:**

```
Selector: 'MemberExpression[property.name=$broadcast]':
function(node){
   Report(
       'Please do not use $broadcast/$on for propagating'
       'events. Use @Input/@Output instead')
}
```

17. Only allow imports relevant part of lodash
    a. **Details:** Please do not use "import { someFunction } from \'lodash\'". Use "import someFunction from \'lodash/someFunction\'" instead.
    b. **Parser:** AST (Abstract syntax tree)
    c. **checker-name:** import-lodash
    d. **Pseudo algorithm:**

```
Selector 'ImportDeclaration[source.value=lodash]':
function(node) {
   if(node.specifiers[0].type !== 'ImportDefaultSpecifier') {
       Report (
           'Please do not use "import { someFunction }'
           'from  \'lodash\'". Use "import someFunction from'
           '\'lodash/someFunction\'" instead   .')
   }
}
```

18. Disallow uses of HttpClient
    a. **Details:** An instance of HttpClient is found in this file. You are not allowed to create http requests from files that are not backend api services.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** disallow-HttpClient
    d. **Pseudo algorithm:**

```
Selector: 'ImportDeclaration': function(node) {
   if(node.specifiers[0].imported.name === 'HttpClient'){
       Report (
           'An instance of HttpClient is found in '
           'this file. You  are not allowed to '
           'create http requests from files that '
```

```
            'are not backend api services.')
    }
}
```

19. Use require to add template.
    a. **Details:** Value of templateUrl is always a callexpression and name of the callee is require
    b. **Parser:** AST(Abstract syntax tree)
    c. **Checker-name:** require-in-templateUrl
    d. **Pseudo algorithm:**

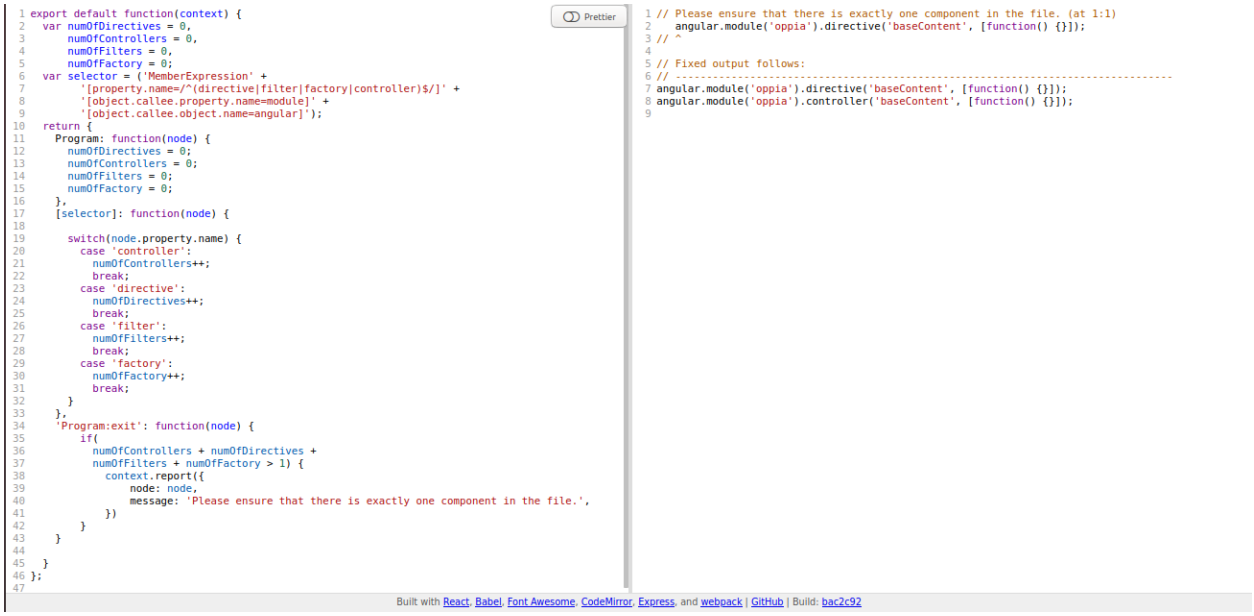
```
var selector = 'CallExpression' +
   '[callee.property.name=directive]' +
   '[callee.object.callee.property.name=module]' +
   '[callee.object.callee.object.name=angular]';
[selector]: function(node) {
    var returnDictProperties =  (
       node.arguments[1].elements[1]
           .body.body[0].argument.properties)
   returnDictProperties.forEach(function(property) {
       if(property.key.name == 'templateUrl') {
           if (
               property.value.type != 'CallExpression' ||
               property.value.callee.name !== 'require') {
                   Report (
                       'Please use require to add template')
           }
       }
   });
}
```

Sub-project 2: Migrate all of the JS\TS lint checks to custom Eslint checks from [js_ts_linter.py](#).

There are sum-total 9 custom checks written in js_ts_linter.py. All the details including pseudo code to implement such checks are as following:

1. check_js_and_ts_component_name_and_count
    a. **Detail:** Checks whether js\ts file should contain one component.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** check-component
    d. **Pseudo algorithm:**

```
var numOfDirective = 0,
   numOfController = 0,
   numOfFilter = 0,
   numOfFactory = 0;

var selector = ('MemberExpression' +

'[property.name=/^(directive|filter|factory|controller)$/
]' +
   '[object.callee.property.name=module]' +
   '[object.callee.object.name=angular]' );

[selector]: function(node) {
   Check (node.property.name)
       if it is directive numOfDirective++
       If it is controller numOfController++
       If it is filter numOfFilter++
       If it is factory numOfFactory++
   }

Selector: 'Program:exit': function(node) {
   var numOfItems = (
       numOfDirective + numOfController +
       numOfFilter + numOfFactory)
   If (numOfItems > 1) {
       Report (
           'Please ensure that there is exactly '
           'one component in the file')
   }
}
```



2. Check_directive_scope
    a. **Detail:** Checks that all directives have an explicit Scope: {} and scope should not set to true.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** check-component
    d. **Pseudo algorithm:**

```
var selector = (

'CallExpression[callee.object.callee.object.name=angular]
'
   '[callee.object.callee.property.name=module]'
   '[callee.property.name=directive]')

[selector]: function(node) {
   For second argument of the directive function  {
       If second argument is not an Array
           return;
       If the last element of the Array is not function
           return;

       returnProperties = function's return dict
       If 'scope' not found in returnProperties:
           Report (
               'Please ensure that the directive in '
               'file has a scope: {}.')
```

```
        If 'scope' is set to true:
            Report (
                'Please ensure that directive in '
                'file does not have scope set to '
                'true')
    }
}
```

```
export default function(context) {
  var selector = 'CallExpression[callee.object.callee.object.name=angular]' +
    '[callee.object.callee.property.name=module]' +
    '[callee.property.name=directive]'
  return {
    [selector]: function(node) {
      if(node.arguments.length !== 2 || node.arguments[1].type !== 'ArrayExpression') {
        return;
      }
      var controllerFunctionNode = node.arguments[1].elements.pop();

      if (controllerFunctionNode.type !== 'FunctionExpression') {
        return;
      }
      if(controllerFunctionNode.body.body[0].type !== 'ReturnStatement') {
        return;
      }
      var returnDictProperties = controllerFunctionNode.body.body[0].argument.properties;

      var scopeFound = false;
      returnDictProperties.forEach(function(property){
        if(property.key.name !== 'scope') {
            return;
        }

        scopeFound = true;

        if(property.value.raw === 'true') {
          context.report({
              node: property,
              message: 'Please ensure that directive in %s file does not have scope set to true'
          });
        }
      });

      if(!scopeFound) {
        context.report({
              node: node,
              message: 'Please ensure that %s directive in %s file has a scope: {}.'
          });
      }

    }
  };
};
```

```
// Please ensure that directive in %s file does not have scope set to true (at 5:7)
        scope: true,
// ------^

// Please ensure that %s directive in %s file has a scope: {}. (at 17:1)
  angular.module('oppia').directive('schemaBasedViewer', [
// ^

// Fixed output follows:
// ------------------------------------------------------------------------------
angular.module('oppia').directive('schemaBasedViewer', [
  'UrlInterpolationService', function(UrlInterpolationService) {
    return {
      restrict: 'E',
      scope: true,
      bindToController: {
        schema: '&',
        localValue: '='
      },
      templateUrl: UrlInterpolationService.getDirectiveTemplateUrl(
        '/components/forms/schema-viewers/schema-based-viewer.directive.html'),
      controllerAs: '$ctrl',
      controller: [function() {}]
    };
  }]);

angular.module('oppia').directive('schemaBasedViewer', [
  'UrlInterpolationService', function(UrlInterpolationService) {
    return {
      restrict: 'E',
      bindToController: {
        schema: '&',
        localValue: '='
      },
      templateUrl: UrlInterpolationService.getDirectiveTemplateUrl(
        '/components/forms/schema-viewers/schema-based-viewer.directive.html'),
      controllerAs: '$ctrl',
      controller: [function() {}]
    };
  }]);
```

3. Check_sorted_dependencies
    a. **Detail:** checks that the dependencies which are imported in the controllers/directives/factories in JS files are in the following pattern: dollar imports, regular imports, and constant imports, all in sorted order.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** check-component
    d. **Pseudo algorithm:**

```
Selector:
'CallExpression[callee.property.name=/(controller|directi
ve|factory)/]': function(node) {
    var args = node.arguments;

    var dependenciesLiteralNodes = args[1].elements;
    dependenciesLiteralNodes.pop()
```

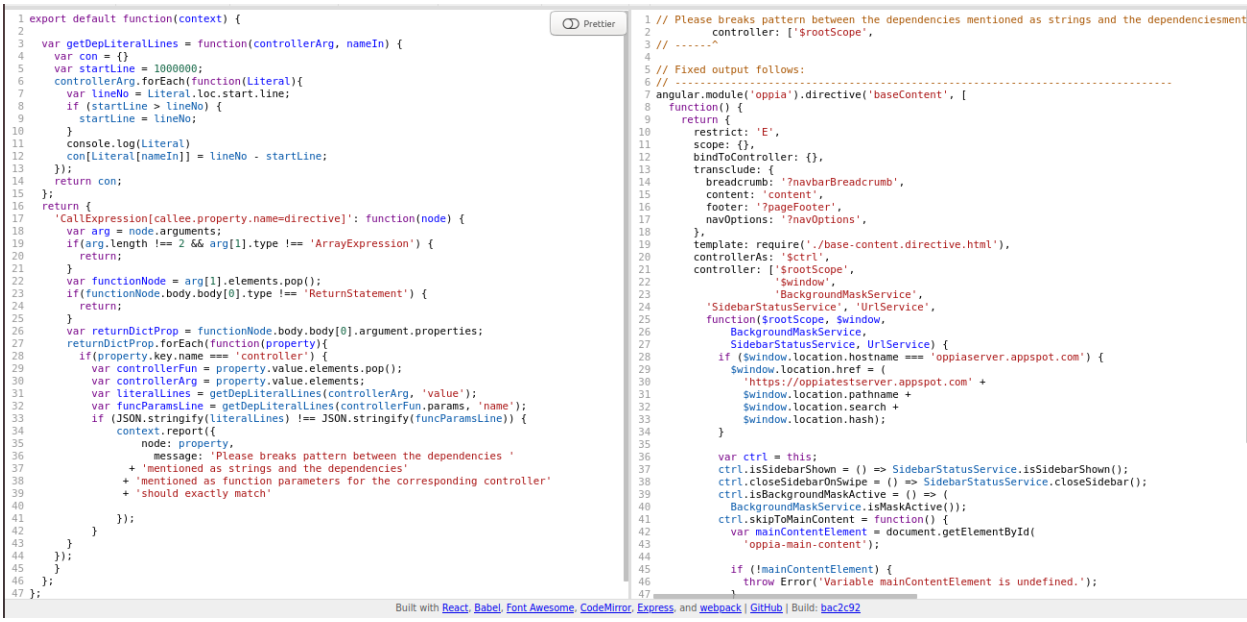
```
    var dependenciesLiterals = [];

    var dollarInjection = [],
        localInjection = [],
        constantInjection = [];

    for each node in dependenciesLiteralNodes {
        dependenciesLiterals.push(node.value)
        If node.value starts with '$':
            dollarInjection.push(node.value)
        If node.value includes [a-z]:
            localInjection.push(node.value)
        else:
            constantInjection.push(node.value)
    }
    var sortedLiterals = [
        ...sorted(dollarInjection),
        ...sorted(localInjection),
        ...sorted(constantInjection)];

    if (dependenciesLiterals != sortedLiteras) {
        Report (
            'Please ensure that the injected '
            'dependencies should be in the '
            'following manner: dollar imports, '
            'local imports and constant imports, all in'
            'sorted-order')
    }
}
```

4. Match_line_breaks_in_controller_dependencies
    a. **Detail**: checks whether the line breaks between the dependencies listed in the controller of a directive or service exactly match those between the arguments of the controller function.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** check-component
    d. **Pseudo algorithm:**

Selector ‘Callexpression[callee.property.name=directive]’: function(node) {
First catch the last element of array of second arguments of function
In return statement of function:
Catch ‘controller’ property
Make a dict (literalsLineIndex) with literal name of property
as a key and line No of that as a value
Catch ‘Function’ in controleer
Make a dict (argsLineIndex) with argument name
Of property as a key and line No of that as a value

Compare both dict ‘literalsLineIndex’ and ‘argsLineIndex’
if(false):
Report: ('Please breaks pattern between the
dependencies mentioned as strings and the
dependencies 'mentioned as parameters for the
corresponding controller' 'should exactly matc)')

}



```
export default function(context) {

 var getDepLiteralLines = function(controllerArg, nameIn) {
   var con = {}
   var startLine = 1000000;
   controllerArg.forEach(function(Literal){
     var lineNo = Literal.loc.start.line;
     if (startLine > lineNo) {
       startLine = lineNo;
     }
     console.log(Literal)
     con[Literal[nameIn]] = lineNo - startLine;
   });
   return con;
 };
 return {
   'CallExpression[callee.property.name=directive]': function(node) {
     var arg = node.arguments;
     if(arg.length !== 2 && arg[1].type !== 'ArrayExpression') {
       return;
     }
     var functionNode = arg[1].elements.pop();
     if(functionNode.body.body[0].type !== 'ReturnStatement') {
       return;
     }
     var returnDictProp = functionNode.body.body[0].argument.properties;
     returnDictProp.forEach(function(property){
       if(property.key.name === 'controller') {
         var controllerFun = property.value.elements.pop();
         var controllerArg = property.value.elements;
         var literalLines = getDepLiteralLines(controllerArg, 'value');
         var funcParamsLine = getDepLiteralLines(controllerFun.params, 'name');
         if (JSON.stringify(literalLines) !== JSON.stringify(funcParamsLine)) {
             context.report({
                 node: property,
                   message: 'Please breaks pattern between the dependencies '
               + 'mentioned as strings and the dependencies'
              + 'mentioned as function parameters for the corresponding controller'
              + 'should exactly match'

             });
         }
     }
   });
   }
 };
};
```

```
// Please breaks pattern between the dependencies mentioned as strings and the dependenciesment
        controller: ['$rootScope',
// ------^

// Fixed output follows:
// ------------------------------------------------------------------------------
angular.module('oppia').directive('baseContent', [
  function() {
    return {
      restrict: 'E',
      scope: {},
      bindToController: {},
      transclude: {
        breadcrumb: '?navbarBreadcrumb',
        content: 'content',
        footer: '?pageFooter',
        navOptions: '?navOptions',
      },
      template: require('./base-content.directive.html'),
      controllerAs: '$ctrl',
      controller: ['$rootScope',
                   '$window',
                'BackgroundMaskService',
       'SidebarStatusService', 'UrlService',
       function($rootScope, $window,
           BackgroundMaskService,
           SidebarStatusService, UrlService) {
         if ($window.location.hostname === 'oppiaserver.appspot.com') {
           $window.location.href = (
             'https://oppiatestserver.appspot.com' +
             $window.location.pathname +
             $window.location.search +
             $window.location.hash);
         }

         var ctrl = this;
         ctrl.isSidebarShown = () => SidebarStatusService.isSidebarShown();
         ctrl.closeSidebarOnSwipe = () => SidebarStatusService.closeSidebar();
         ctrl.isBackgroundMaskActive = () => (
           BackgroundMaskService.isMaskActive());
         ctrl.skipToMainContent = function() {
           var mainContentElement = document.getElementById(
             'oppia-main-content');

           if (!mainContentElement) {
             throw Error('Variable mainContentElement is undefined.');
           }
```

Built with React, Babel, Font Awesome, CodeMirror, Express, and webpack | GitHub | Build: bac2c92

5. Check_constants_declaration
    a. **Details:** Checks the declaration of constants in the TS files to ensure that the constants are not declared in files other than *.constants.ajs.ts and that the constants are declared only single time. This also checks that the constants are declared in both *.constants.ajs.ts (for AngularJS) and in *.constants.ts (for Angular 8).
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** check-component
    d. **Pseudo algorithm:**

```
var selector  = (
   'CallExpression[callee.property.name=constant]' +
   'callee.object.callee.object.name=angular]' +
   '[callee.object.callee.property.name=module]')

var contantsInFile = [];

[selector]: function(node) {
   If (filename not ends with constant.ajs.ts) {
       Report('Constant is used in non-constant file')
   }
   if (constantName in contantsInFile) {
       Report(
           'There are two constants in this file')
```

```
    }
    contantsInFile.push(constantName);
}
```

```
export default function(context) {
  var constnatsDecelerations = [];
  var args;
  var fileName = context.getFilename();
  return {
    'CallExpression[callee.property.name=constant][callee.object.callee.object.nam
      if(!fileName.includes('ajs.ts')) {
        context.report ({
            node: node,
            message: 'Constant is used in non-constant file'
        });
      }
      args = node.arguments[0].value ;
      if(!constnatsDecelerations.includes(args)) {
        constnatsDecelerations.push(args);
      }
      else {
        context.report ({
            node: node.arguments[0],
            message: 'There are two constants in this file.'
        });
      }
    }
  };
};
```

```
// Constant is used in non-constant file (at 1:1)
   angular.module('oppia').constant(
// ^

// Constant is used in non-constant file (at 4:1)
   angular.module('oppia').constant(
// ^

// There are two constants in this file. (at 5:3)
     'ADMIN_ROLE_HANDLER_URL', AdminPageConstants.ADMIN_ROLE_HANDLER_URL);
// --^

// Fixed output follows:
// ------------------------------------------------------------------------
angular.module('oppia').constant(
  'ADMIN_ROLE_HANDLER_URL', AdminPageConstants.ADMIN_ROLE_HANDLER_URL);

angular.module('oppia').constant(
  'ADMIN_ROLE_HANDLER_URL', AdminPageConstants.ADMIN_ROLE_HANDLER_URL);
```

6. Check_comments
    a. **Detail:** Checks whether comments follow correct style. Below are some formats of correct comment style:
        1. A comment can end with the following symbols: ('.', '?', ';', ',', '{', '^', ')', '}', '>').
        2. If a line contain any of the following words or phrases('@ts-ignore', '--params', 'eslint-disable', 'eslint-enable', 'http://', 'https://') in the comment.
    b. **Parser:** Token
    c. **Checker-name:** comments-style
    d. **Pseudo algorithm:**
        i. In source code
            Comments = sourceCode.getAllComments()
            For each one line and multiline Comments
                If comment starts with '@ts-expect-error', '@ts-ignore', '--params', 'eslint-disable', 'eslint-enable':
                    Continue
                If the last line of a comment does not ends with ('.', '?', ';', ',','{', '^', ')', '}', '>':
                    Report (‘Invalid punctuation used at the end of the comment.’)

7. Check_ts_ignore
    a. **Detail:** Check if @ts-ignore is used then add a comment above it explaining the requirement in the following format:
        ‘This throws <explain-error>. This needs to be suppressed because <reason-for-ignoring-the-error>'

b. **Parser:** Token
c. **Checker-name:** comments-style
d. **Pseudo algorithm:**
    i. In source code
        Comments = sourceCode.getAllComments()
        For each one line and multiline Comments
            If comments starts with @ts-ignore
                Check if there is no comment above it
                    Report (
                    'Please add a comment above the @ts-ignore explaining the @ts-ignore at line %s. The format of comment should be -> This throws "...". This needs to be suppressed because ...')

8. Check_ts_expect_error
    a. **Detail:** Checks the following:
        i. Disallow **@ts-expect-error** usage in non spec file.
        ii. if **@ts-expect-error** is used in spec file then there must be a comment above it explaining the requirement in the following format:
            'This throws <explain-error>. This needs to be suppressed because <reason-for-ignoring-the-error>'
    b. **Parser:** Token
    c. **Checker-name:** comments-style
    d. **Pseudo algorithm:**
        i. In source code
            Comments = sourceCode.getAllComments()
            For each one line and multiline Comments
                If file is non spec and comment starts with @ts-expect error
                    Report ('@ts-expect-error found at line %s.
                        'It can be used only in spec files.')
                Else if comments starts with @ts-ignore
                    Check if there is no comment above it
                        Report (
                        'Please add a comment above the @ts-expect-error explaining the @ts-expect-error at line %s. The format of comment should be -> This throws "...". This needs to be suppressed because ...')

9. Check_angular_services_index
    a. **Detail:** Finds all @Injectable classes and makes sure that they are added to Oppia root and Angular Services Index.

b. **Parser:** check-angular-service-indexing
c. **Pseudo algorithm:**

```
import {angularServices} from angular-services.index

var serviceAvailibility = {}

for (service in angularServices) {
   serviceAvailibility[service[0]] = true;
}

const selector = (
 'ClassDeclaration >
Decorator[expression.callee.name="Injectable"]');

Selector: [selector]: function(node) {
   if(node.id.name not in serviceAvailibility) {
       Report(
           'Please add the service %s in the '
           'angularServices of '
           'angular-servicies.index.ts file')
   }
}
```

Sub-project 3: Implement 4 E2E lint checks from [#8423]

1. Making sure constant variable names are in all-caps
   a. **Detail:** constant variable names are in all-caps eg:-(const ADI = 5;)
   b. **Parser:** AST (Abstract syntax tree)
   c. **Checker-name:** protractor-practices
   d. **Pseudo algorithm:**

```
Selector: 'VariableDeclaration[kind=const]': function(node) {
        If identifier name of const variable are not in all-caps:
            Report (Please make sure constant variable
                names are in all-caps)
    }
```

```
export default function(context) {
  return {
    'VariableDeclaration[kind=const]': function(node) {
      var m = node.declarations[0].id.name;
      if (m != m.toUpperCase()) {
        console.log(m.isUpperCase);
        context.report({
            node: node,
            message: 'Please make sure constant variable names are in all-caps'
        });
      }
    }
  };
};
```

```
// Please make sure constant variable names are in all-caps (at 3:1)
   const yuadsf = 45;
// ^

// Fixed output follows:
// ----------------------------------------------------------------------
const AD = 5;
var dnj = 6
const yuadsf = 45;

```

2. Do not call .first(), .last(), or .get(i) calls on ElementArrayFinder objects
    a. **Detail:** Disallow .first(), .last(), .get() on protractor files.
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** protractor-practices
    d. **Pseudo algorithm:**
        i. Selector: 'CallExpression[callee.property.name=/first|last|get/]': function(node) {
            Report ('please do not call '+node.callee.property.name)
            }

```
export default function(context) {
  return {
    'CallExpression[callee.property.name=/first|last|get/]': function(node) {
        context.report({
            node: node.callee.property,
            message:'please do not call '+node.callee.property.name

        });
    }

  };
};

```

```
// please do not call first (at 1:24)
   editorCategoryDropdown.first()
// ----------------------^

// please do not call last (at 2:41)
   element.all(by.css('.CodeMirror-code')).last()
// ----------------------------------------^

// please do not call get (at 3:13)
   libraryPage.get()
// ------------^

// Fixed output follows:
// ----------------------------------------------------------------------------
editorCategoryDropdown.first()
element.all(by.css('.CodeMirror-code')).last()
libraryPage.get()
```

3. Making sure all root element selectors use HTML classes that start with "protractor-test
    a. **Detail:** Making sure all root element selectors use HTML classes that start with "protractor-test-", e.g. element(by.css('.protractor-test-parent-element')).element(by.css('.class-of-element-you-cannot-change'))
    b. **Parser:** AST (Abstract syntax tree)
    c. **Checker-name:** protractor-practices
    d. **Pseudo algorithm:**

```
Selector:
'CallExpression[callee.object.name=by][callee.property.name=css]': function(node) {
 if(!node.arguments[0].value.startsWith(".protractor-test-"))
{
   Report(
```

```
      'Please use “protractor-test-” ' +
      'prefix class selector')
  }
}
```

```
export default function(context) {
  return {
    'CallExpression[callee.object.name=by][callee.property.name=css]': function(node) {
      if(!node.arguments[0].value.startsWith(".protractor-test-"))
        context.report({
            node: node.arguments[0],
            message:'Please use “protractor-test” prefix class selector'

        });
    }

  };
};
```

```
// Please use “protractor-test” prefix class selector (at 2:12)
      by.css('.protractor-collection-editor-objective-input'));
// ----------^

// Fixed output follows:
// ------------------------------------------------------------------------
var collectionEditorObjectiveInput = element(
    by.css('.protractor-collection-editor-objective-input'));
```

4. All element selectors in protractor files should be on top
    a. **Detail:** All element selectors in protractor files should be on top
    b. **Parser:** AST (Abstract syntax tree)
    c. **Pseudo algorithm:**

```
var exportFunctionName = null

Selector: 'ExportNamedDeclaration': function(node) {
   exportFunctionName = node.decleration.name;
}

Selector: 'FunctionExpression:exit': function(node) {
   var elementsDeclerationEnded = false;
   if (node.id.name == exportFunctionName) {
       while(node) {
           if (node.type = Assignment node and the left node
is call to element()) {
               if(elementsDeclerationEnded) {
                   Report('Please declare the elemnts on
top.')
               }
               node = node.next();
               continue;
           }
           if (node.type == FunctionExpression) {
               elementsDeclerationEnded = True;
               node = node.next();
```

```
                continue;
            }


        }
    }
}
```

Files to create/update

The name of the files which will be created/updated [c/u] under this projects:

From general_purpose_linter.py:

- [U] protractor-practices.js
- [C] disallow-identifier-phrase.js
- [C] no-test-blockers.js
- [C] use-angular-mock-inject.js
- [C] incomplete- throw.js
- [C] no-parent-access.js
- [C] disallow-relative-imports.js
- [C] disallow-inner-html-property.js
- [C] disallow-comments.js
- [C] disallow-broadcast-on.js
- [C] import-lodash.js
- [C] disallow-http-client.js
- [C] require-in-template-url.js

From js_ts_linter.py

- [C] check-component.js
- [C] comments-style.js
- [C] check-angular-service-indexing.js

From #8423

- [U] protractor-practices.js

Third-party Libraries*

This project will not add any new third-party lib in the repository.

Testing Approach

1. All the new lint checks added/migrated to eslint in this project will be covered with unit tests.
    a. Unit test for each of the checks will have the following structure:

i. Name of the file: <check-name>.spec.js
ii. Test code pattern:

```
var ruleTester = new RuleTester();
ruleTester.run('<checker-name>', rule, {
 valid: [
   {
     code: '<correct-pattern>'
   },
   {
     code: '<correct-pattern>'
   },
   ...
 ],

 invalid: [{
     code: '<incorrect-pattern>',
     errors: [{
       message: '<message>',
       type: '<ast-node>',
     }],
   },{
     code: '<incorrect-pattern>',
     errors: [{
       message: '<message>',
       type: '<ast-node>',
     }],
   }, ...]
});
```

Milestones

The details of work that need to be done under this project is given below:

Milestone 1

Key Objective:
The key objective in milestone 1 is given as following as:

1. Migrate all custom checks written in [js_ts_linter.py]() to eslint:

a. Migrate all of the angular/angularJS related JS/TS lint checks to custom eslint check from.
    i. [M 1.1] Migrate check_js_and_ts_component_name_and_count
    ii. [M 1.1] Migrate check_directive_scope
    iii. [M 1.1] Migrate check_sorted_dependencies
    Note:- The above checks^ will be added in check-component.js file.
    iv. [M 1.2] Migrate match_line_breaks_in_controller_dependencies
    v. [M 1.2] Migrate check_constants_declaration

b. Migrate all comment related JS/TS lint checks to custom eslint checks from js_ts_linter.py.
    i. [M 1.3] Migrate check_comments
    ii. [M 1.3] Migrate check_ts_ignore
    iii. [M 1.3] Migrate check_ts_expect_error
    NOTE:- These above^ checks will be added in comment-style.js.

c. [M 1.4] Migrate check_angular_services_index lint checks to custom eslint check from js_ts_linter.py.

2. Implement 3 new E2E lint checks
    a. [M 1.5] Implement liint check for Making sure constant variable names are in all-caps
    b. [M 1.6] Implement lint for check Do not call .first(), .last(), or .get(i) calls on ElementArrayFinder objects
    c. [M 1.7] Implement lint check for Making sure all root element selectors use HTML classes that start with "protractor-test-”

Number of pull requests

Total number of pull requests for milestone1 should be **7**

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
1.1	Migrate components related angular/angularJS related lint checks to custom eslint checks from js_ts_linter.py.	none	11/06/21	14/06/21
1.2	Migrate remaining angular/angularJS related lint checks to custom eslint checks from js_ts_linter.py.	1.1	15/06/21	18/06/21

1.3	Migrate all comment related JS/TS lint checks to custom eslint checks from js_ts_linter.py.	none	19/06/21	23/06/21
1.4	Migrate check_angular_services_index lint checks to custom eslint check from js_ts_linter.py.	none	23/06/21	25/06/21
1.5	Implement lint check for making sure constant variable names are in all-caps	none	27/06/21	30/06/21
1.6	Implement lint for check Do not call .first(), .last(), or .get(i) calls on ElementArrayFinder objects	none	02/06/21	04/07/21
1.7	Implement lint check for Making sure all root element selectors use HTML classes that start with "protractor-test-"	none	07/07/21	11/07/21

Milestone 2

Key Objective:

The key objective in milestone 2 is given as following as:

1. Implement remaining new e2e lint check:
    a. [M 2.1] Implement lint check for keeping element selectors at the tops of files.
2. Migrate all BAD_PATTERNS_JS_AND_TS_REGEXP to eslint:
    a. [M 2.2] Group 1:
        i. Migrate disallow browser.explore() check
        ii. Migrate disallow browser.pause() check
        iii. Migrate disallow browser.waitForAngular() check
    b. [M 2.3] Group 2
        i. Migrate disallow usage of word “bypass” in new file disallow-identifier-phrase.py
        ii. Migrate disallow “xdescribe” & “fdescribe” in new file no-test-blocker.js
        iii. Migrate disallow ‘iit’ & ‘fit’ in file no-test-blocker.js
        iv. Migrate disallow calling ‘inject’ in new file use-angular-mock-inject.js

    c. [M 2.4] Group 3
        i. Migrate disallow calling ‘toThrow’ in new file incomplete- throw.js
        ii. Migrate disallow usage of ‘throw’ statement in file incomplete-throw.js

iii. Migrate disallow usage of ‘throw’ statement in file incomplete-throw.js
iv. Migrate disallow uses of word $parent in new file no-parent-access.js
v. Migrate disallow use of relative imports in new file disallow-relative-imports.js

d. [M 2.5] Group 4
   i. Migrate disallow uses of word ‘innerHTML’ in new file disallow-innerHTML-property.js
   ii. Migrate disallow es-lint-(enable|disable) camelcase in new file disallow-comments.js
   iii. Migrate Disallow no-explicit-any in file disallow-comments.js
   iv. Migrate Disallow uses of $broadcast call

e. [M 2.6] Group 5
   i. Migrate Only allow imports relevant part of lodash in new file import-lodash.js
   ii. Migrate Disallow uses of HttpClient in new file disallow-HttpClient.js
   iii. Migrate Disallow calling templateUrl

No.	Description of PR	Prereq PR numbers	Target date for PR submission	Target date for PR to be merged
2.1	Implement lint check for keeping element selectors at the tops of files.	none	20/07/21	25/07/21
2.2	[Group 1] Migrate some of the BAD_PATTERNS_JS_AND_TS_REGEXP checks to custom eslint checks from general_purpose_linter.py	none	24/07/21	27/07/21
2.3	[Group 2] Migrate some of the BAD_PATTERNS_JS_AND_TS_REGEXP checks to custom eslint checks from general_purpose_linter.py	none	28/07/21	01/08/21
2.4	[Group 3] Migrate some of the BAD_PATTERNS_JS_AND_TS_REGEXP checks to custom eslint checks from general_purpose_linter.py	none	31/07/21	03/08/21

2.5	[Group 4] Migrate some of the BAD_PATTERNS_JS_AND_TS_REGEXP checks to custom eslint checks from general_purpose_linter.py	none	04/08/21	08/08/21
2.6	[Group 5] Migrate some of the BAD_PATTERNS_JS_AND_TS_REGEXP checks to custom eslint checks from general_purpose_linter.py	none	07/08/21	10/08/21

Optional Sections

Additional Project-Specific Considerations

None.

Privacy

None. This project doesn't touch any user-facing features.

Security

None. This project doesn't touch any user-facing features.

Accessibility (if user-facing)

None. This project doesn't touch any user-facing features.

Documentation Changes*

None. This project migrates existing lint checks to eslint so it doesn't need to add any new documentations.

Ethics*

None.

Future Work

None.