# Google Summer of Code 2019

# Asking students why they picked a particular answer

# (Oppia)

## Personal Details:

**Name** : Anubhav Sinha
**Email** : anubhavsinha98@gmail.com
**GitHub Handle**: anubhavsinha98
**College**: Jaypee Institute of Information Technology, Sec 62, Noida
**Program**: B.Tech in Computer Science(4 year course)

I am currently pursuing my 2nd year of B.Tech in Computer Science at Jaypee Institute of Information Technology. JIIT Noida is one of the renowned institutes for computer science in the Delhi NCR. I am having a cumulative GPA of 8.9 out of 10 i.e securing as follow
1st Semester : 8.1 SGPA, 2nd Semester: 9.5 SGPA, 3rd Semester : 9.3 SGPA.
I am a member of IEEE Student Branch JIIT Noida and Open Source Developers Club of my college. I have been working on Python since 2 years and JavaScript for 1 year. I am a full stack developer. I started my web development journey during the mid of 2018. My other skills are Angular JS, C++, PHP, Node.js, GUI Development using Python, CSS & Photoshop.

## Project Details:

### Project Name: Asking students why they picked a particular answer

### Why I am interested in working with Oppia:

Oppia's mission is to help anyone learn anything they want in an effective and enjoyable way. And I want to contribute towards the mission so that via Oppia students can learn the various subjects and its topics. Oppia also helps educators around the world who want to express their knowledge and make some interesting explorations so that they can help the students. And explorations are much interesting to learn a topic/subject, as explorations include videos, images, graphics, interactive questions which helps in creating interest among the students. I also prefer this kind of learning, as it helps in boosting critical and natural thinking.

And the team at Oppia is very helpful and friendly. The support from the Oppia team is truly amazing, this helps the first time contributor to merge a successful PR which proves as a motivational boost for the contributor. I would conclude that the whole team of Oppia is really helpful and cooperative.
Also I like the way that every new contributor is assigned to a mentor, which helps the contributor to directly ask for help to the mentor.

## What interests me about this project? Why is it worth doing?

Questions are the best parameter to judge a person that he knows about that topic.
Whenever a person makes an online course, he adds various tests at every step in the course to ensure that the learner has learned the previous topics. But sometimes it happens that the learner is not able to attempt that question correctly in 2-3 attempts which must not be expected by the creator of the course, as he has added the questions according to the course he has created. But the actual insights can only be provided by the learner who is solving the questions related to the course. So in this case if a learner is able to provide a response to the creator about his approach that what is leading to the submission(i.e how the learner landed on this answer) it would be really helpful to the creator to reach out those responses filled by the learner, so that he can edit the course accordingly which will help in increasing the learner experience and in future he will be able to make more good courses which will improve the feedback or review from the learner. And through this response, the learner will also able to analyze what is the reason for their submission.
The best part of this project is that the creator will get to know how the learners are applying their approach, same as what happens in actual class when students discuss their approach for attempting a question from their teachers such that it helps the creator to make the course/exploration more user-friendly.
So I think this project will help the creators of the Oppia to enhance their explorations, enriching the learner's experience, which will overall improve the Oppia's feedback.
The learner will also able to analyze more about their happenings of the wrong submission.

## Prior Experience:

I have worked on the following projects:

- **Get It Registered** - A python project which includes an interface built using Tkinter module, which is currently being used in my college by various hubs in order to do registration of students for any event conducted in college.

- **Library Management** - Web development project which includes the management of the Library, like the students can view the books are currently available in Library and can hold that book if they want to issue that book, but for a limited period only. Backend was built in PHP, Database used was MySql and frontend with HTML, CSS and JavaScript.

- **Learn Blockchain** - A website which help teenagers to learn about the Blockchain technology. Various UI interactions, games using javascript are build to help the teenagers learn blockchain technology easily.

- **Best Route** - Data Structures project on finding the best route from destination airport to the arrival one. Built in C++, Binary Heaps, Priority Queue and Graphs were the data structures used.

- **Bill Split** - Project built on C. Helpful for splitting the bill for a group of people. User account is also maintained, such that all the transactions can be viewed done by the user. UI was also built using C.

- **Task Killer** - Basic task killer which kills the tasks by clicking it, built in python.

I am an active member of Open Source Developers Club, JIIT Noida, and IEEE Student Branch JIIT Noida. I have also taken lectures on Python and C++, held by my college under the workshops conducted by IEEE Student Branch JIIT Noida.

I have also attended various hackathons in the Delhi NCR, like HACK CBS, DSC Hackathon, Hack With Tony, etc. and gained experience on how to make real-world projects & I am also active in various conferences around Delhi NCR like I have attended PyDelhi meetup 2018, LinuxChix Meetup 2018, various tech talks conducted in our college.
I have also completed a course on Python “Automate the Boring Stuff with Python: Practical Programming for Total Beginners” - By Al Sweigart.
I also do competitive programming sometimes & play capture the flag events.

I am an active contributor at Oppia, I am a part of Dev Workflow Team lead by Apurv Bajaj, Overall Learner Experience lead by Akshay Anand, Bug Fixing Team, Oppia Onboarding Team, and also an Oppia Team Member. I have been contributed to Oppia since November 2018 so I am very much familiar with the codebase both frontend and backend.

### Link to important PRs:

I have been contributing to Oppia since November 2018, and till now I have my 18 PRs merged, and 4 are open.

1. One off job, to populate message count in GeneralFeedbackThreadModel #5999
2. Added story viewer backend handler #6237
3. Added subtopic page data handler #6327
4. Updating Feedback threads in real time #6183
5. Added a presubmit check for JS files and refactoring the files required #6125

I have fixed many bugs related to frontend and backend, and for a full list of PR click here.

# Project Plan:

## Overview:

This project aims to introduce a new feature of asking the students why they landed on a particular answer, such that students can explain what lead to this situation. This will help students to encourage reflection on their part, as well as this will also help in providing the anonymized information to the creators about student misconceptions, so that creator can improve the Oppia's feedback for future students. Through this creators can enhance their explorations basically the questions quality such that creators can vary the difficulty of the questions at different stages of the explorations through the statistical information received & this will improve the Learner's Experience too.

## Why do we need this feature?

This feature will help in making the explorations more robust, such that creators can learn from the student misconceptions and get an in-depth knowledge of their published explorations, which will help in editing the explorations and creating more learner-friendly explorations in future which will lead to improved feedback of Oppia in future. From the student's side, this will benefit them to recognize their thought process that leads to the submission of the answer. This will create a class like environment for Oppia, as in classes the teacher gets to know the approach a student is applying while solving any question, in the same way this response that will be the approach described by the learner will help the creator to understand what actually the students are applying and thus the creator can improve the exploration.Thus resulting in a good feedback for Oppia.

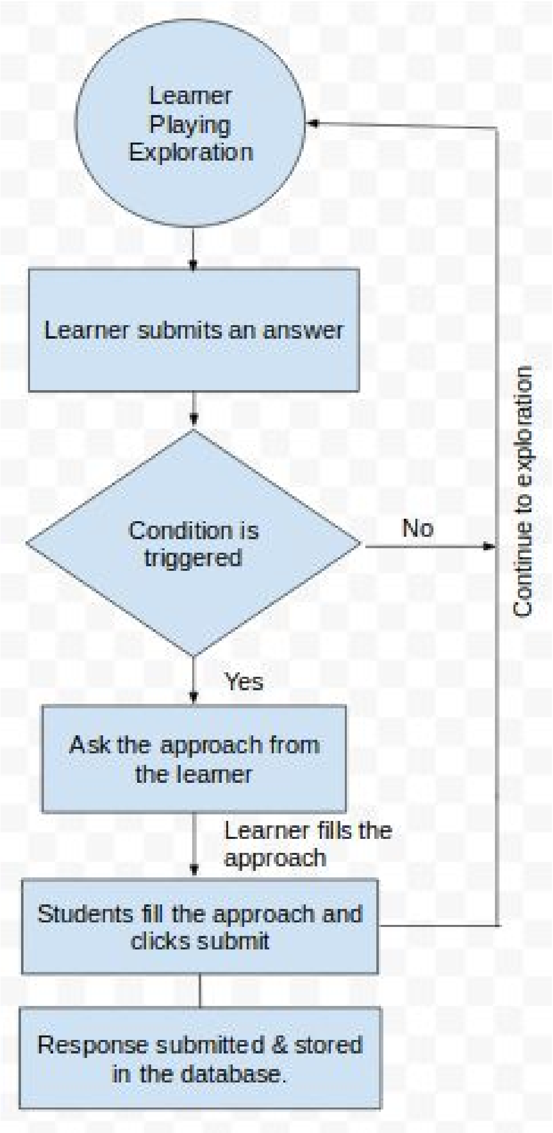
## Interactions which will get this feature:

Currently, the Text Input interaction will get this feature as it will be an easy workflow for the learners such that they enter the answer for the question asked and then they submit the approach. There will be a checkbox which will enable the option to ask learners for their approach when the creator adds a text input interaction, this will help to ask for the responses where the creator wants. The feature can be extended to other interactions also in the future.

## Technical Outline:

**Learner's Aspect:**

Learner Control Flow for submitting the approach of an answer submission.


Learner Playing Exploration
Learner submits an answer
Condition is triggered
No
Continue to exploration
Yes
Ask the approach from the learner
Learner fills the approach
Students fill the approach and clicks submit
Response submitted & stored in the database.

# Condition which will trigger this flow:

## Approach:

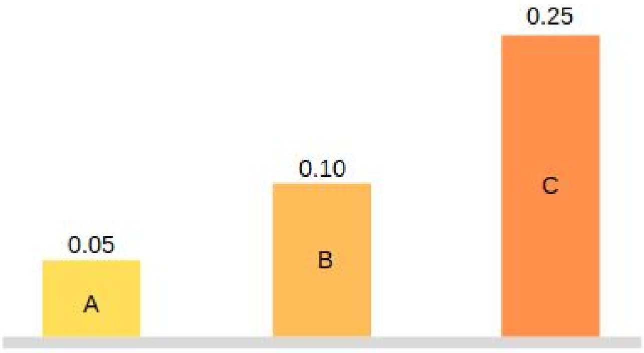
The condition will be triggered randomly, such that a function will be created in the frontend services which will generate a random number in the range [0,1]. **If the answered group probability index will be greater than or equal to the randomly generated value then the learner will be asked for the response**. For generating the random value, the random function of the math module will be used.
There are three types of answer groups:

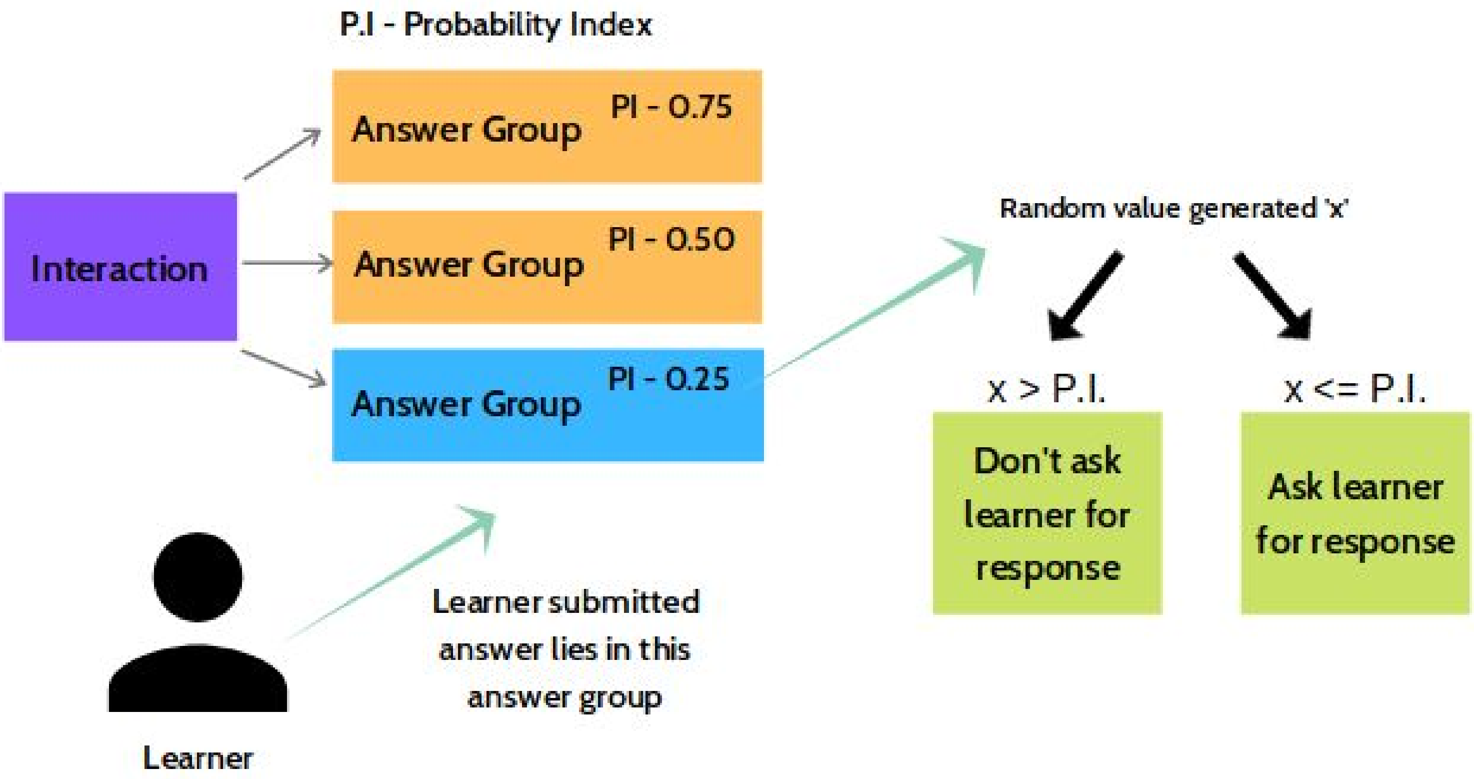
- Answer group with default outcome (A)
- Answer group with outcome labeled as correct (B)
- Answer group without default outcome (C)

The answer groups with correctness feedback enabled and marked as incorrect will be considered as a subset of of type C.
The learner answer will lie in the above answer groups, after answering the question a random value will be generated which will be checked by the probability index of the answer group.

## About probability index(weights):

The probability index variable will be maintained in the frontend for all the answer groups of the text input interaction. Those answer groups which have default outcome will have the maximum probability index value like 0.25, and answer groups which don't have default outcome will have the minimum probability index like 0.05, also the answer groups which have marked as correct by the creator can be assigned to the value 0.10. These indexes of the answer group will indicate the probability to ask the learner for its approach, such as the higher the probability more is the chance of asking the learner. The probability index will be introduced in the constant file such that it can be changed in the future.



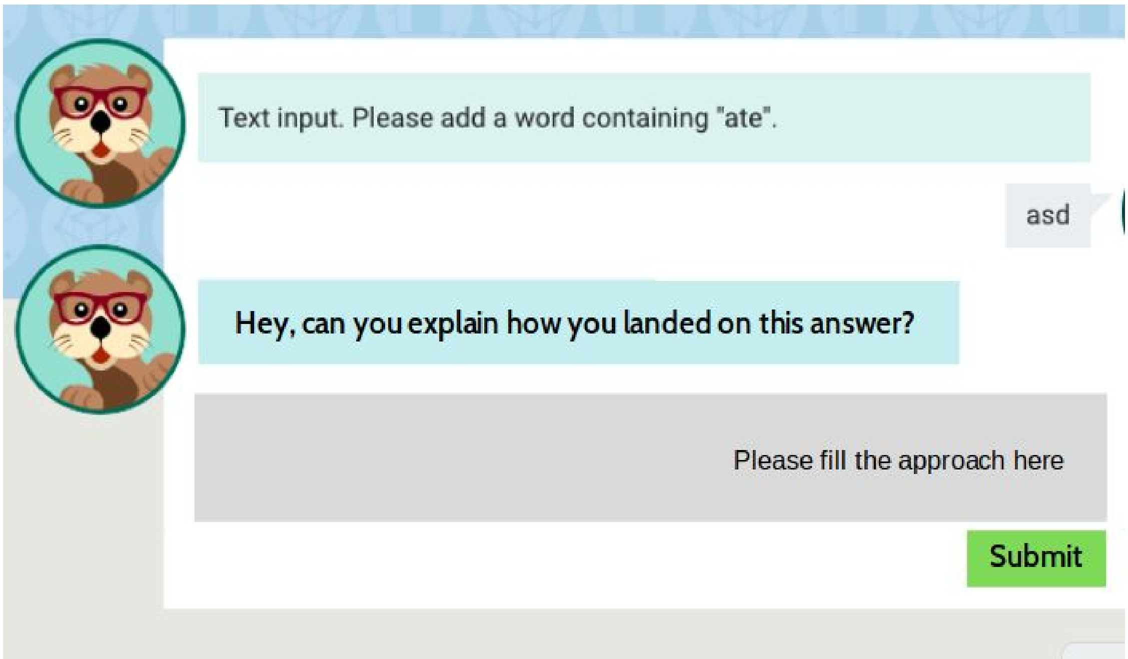


## Overview of learners view:

When the learner will submit any answer, the answer will be checked that it belongs to which answer group and then the probability index of the answer group will be calculated such that a random value will be generated after that through the computations (as mentioned above). And if random value will be less than or equal to the probability index of the answer group in which the learner answers lie will indicate that the condition is triggered and the learner needs to be asked for the response. A response form will get opened in the exploration player such that the learner will be asked to fill their approach such that they landed on this answer. Response Form will include the textbox, such that the learner can write about its approach that leads to the submission. And then submit the form which will save the response and then a message will appear which will thanks the learner and the learner can then continue back to the exploration i.e move to next state. There will be a count of number of responses submitted by the learner in an exploration for a session such that no response will be asked if the learner has already submitted 2-3 responses.

*(Please Note that these are just the very basic preliminary designs which will be improved upon with the help of community feedback)*

## Form where learner will submit his approach:

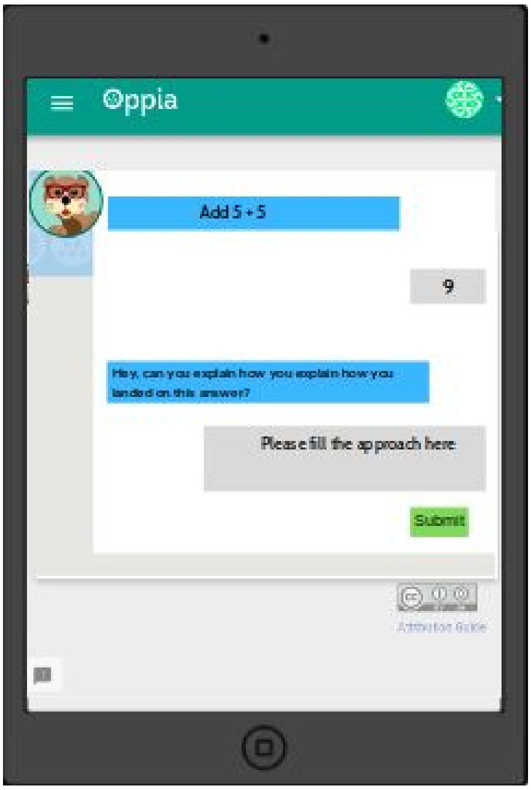


Text input. Please add a word containing "ate".

asd

Hey, can you explain how you landed on this answer?

Please fill the approach here



Add 5 + 5

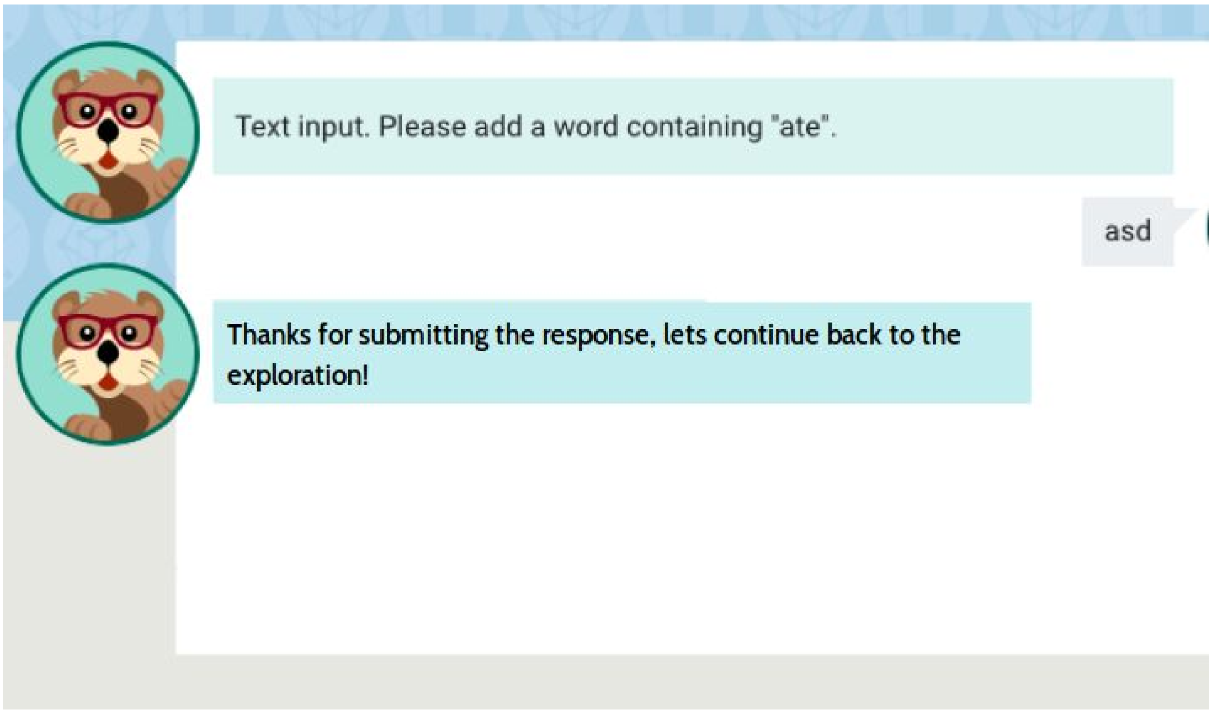
9

Hey, can you explain how you explain how you landed on this answer?

Please fill the approach here

View of the learner in the mobile

## UI for thanking learner for submitting the response:



Text input. Please add a word containing "ate".

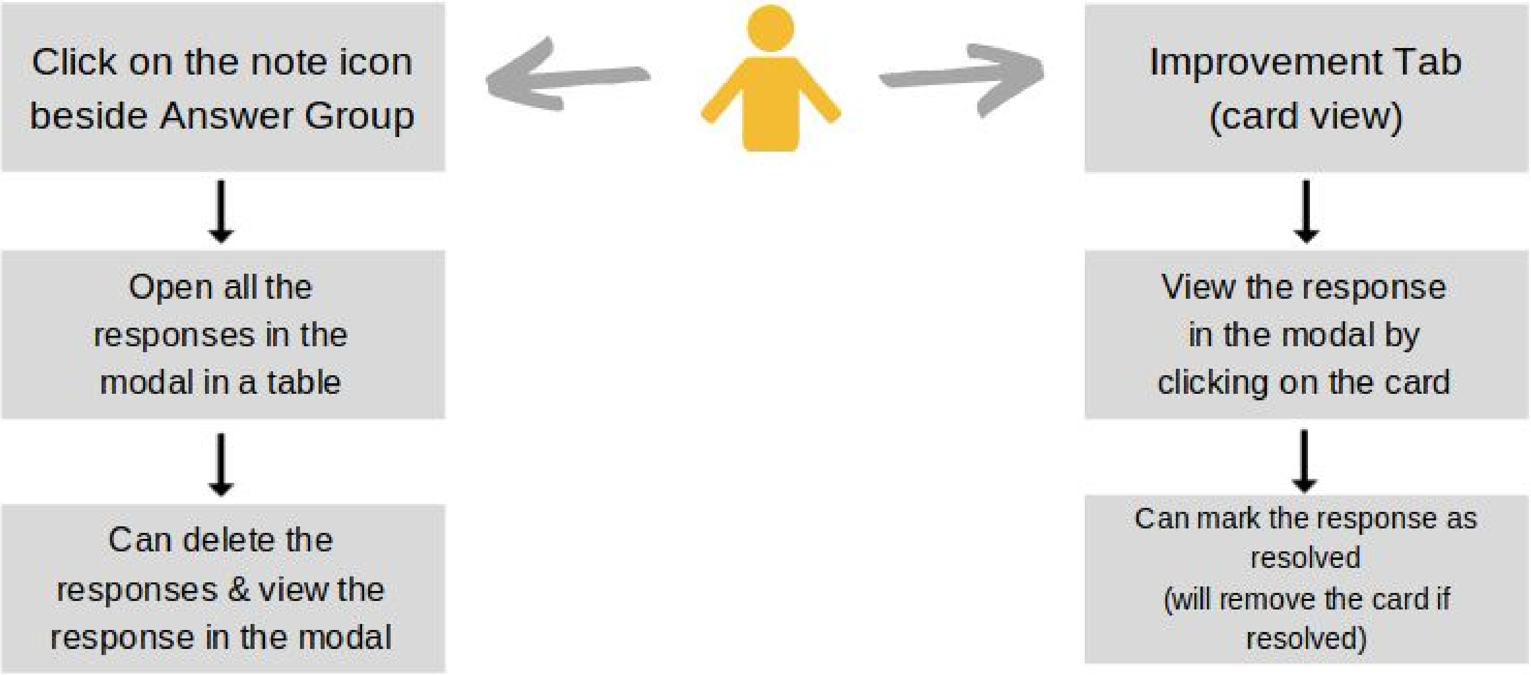
asd

Thanks for submitting the response, lets continue back to the exploration!

## Creator Aspect :

The creator can view the responses received for any exploration by going to its Improvement Tab in nav bar or click on the note icon beside the answer groups in the exploration editor tab. The creator will be able to delete as well as resolve the responses received. There will be a To-Do task for the creator to resolve the cards in the Improvements Tab and for reference purposes, the responses will be shown in the Editor Tab beside the answer groups so that the creator can modify the answer groups accordingly.

# Control Flow for Creator:



Click on the note icon beside Answer Group

Open all the responses in the modal in a table

Can delete the responses & view the response in the modal

Improvement Tab (card view)

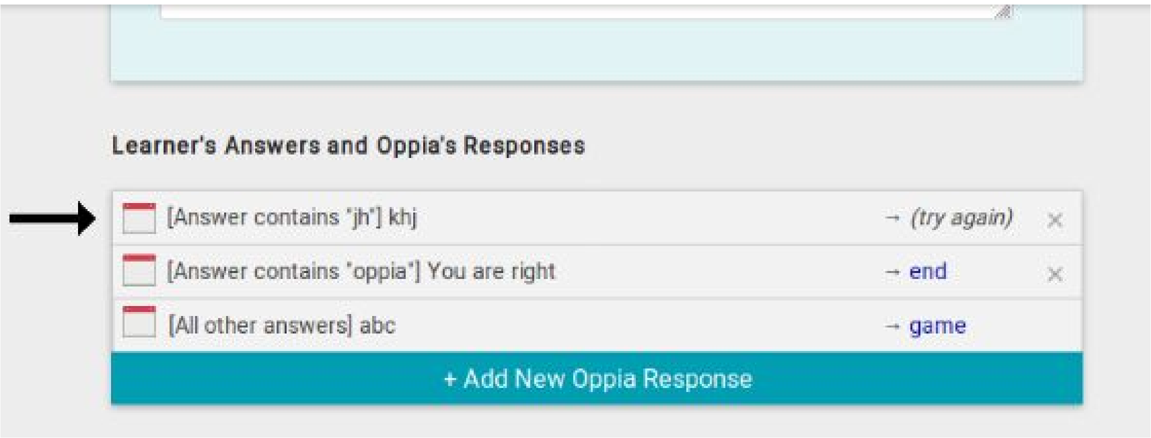
View the response in the modal by clicking on the card

Can mark the response as resolved (will remove the card if resolved)

The creator can view the responses from two places one in the Improvement Tab and one by clicking beside the Answer Groups such that a new modal will be open which will contain all the responses.

## Clicking on the icon besides the answer group in the exploration editor tab:

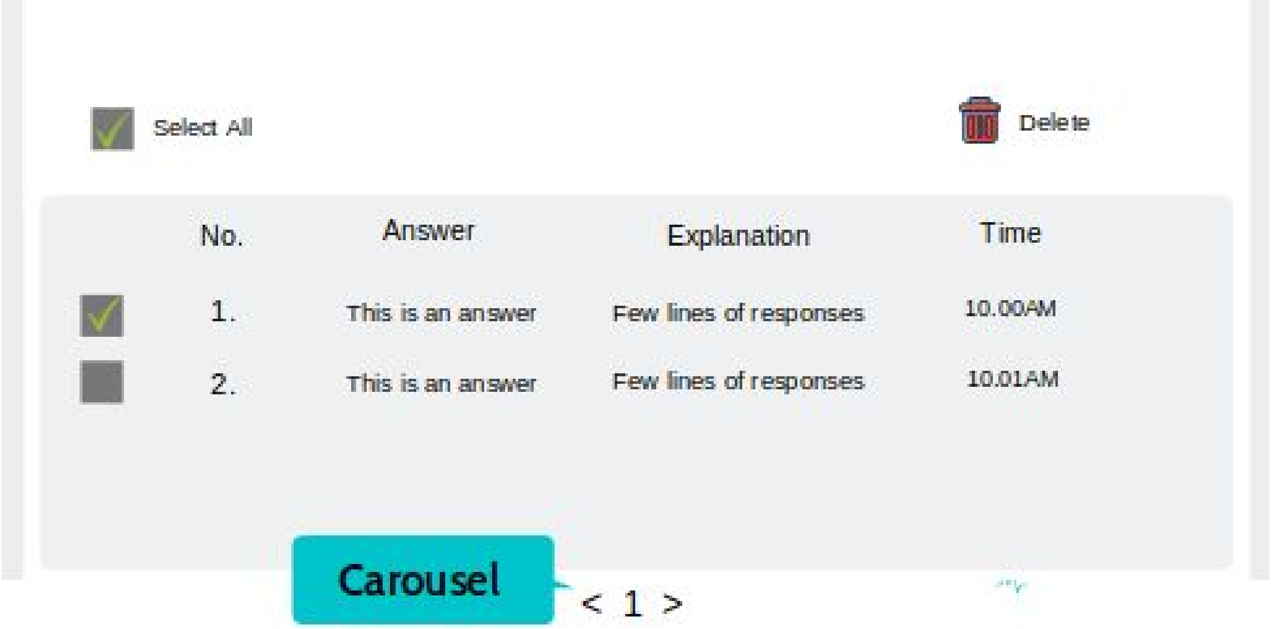
The creator will be able to view the responses by clicking on the note icon which will be shown beside the answer groups. Such that when the creator clicks on the icon a new modal window will get open which will include a table which will hold all the responses received for that answer groups and when the creator will click on a particular response a new directive will be loaded which will show the full response, also the answer which the user has entered.



Learner's Answers and Oppia's Responses

| | |
|---|---|
| [Answer contains 'jh'] khj | → *(try again)* |
| [Answer contains 'oppia'] You are right | → end |
| [All other answers] abc | → game |

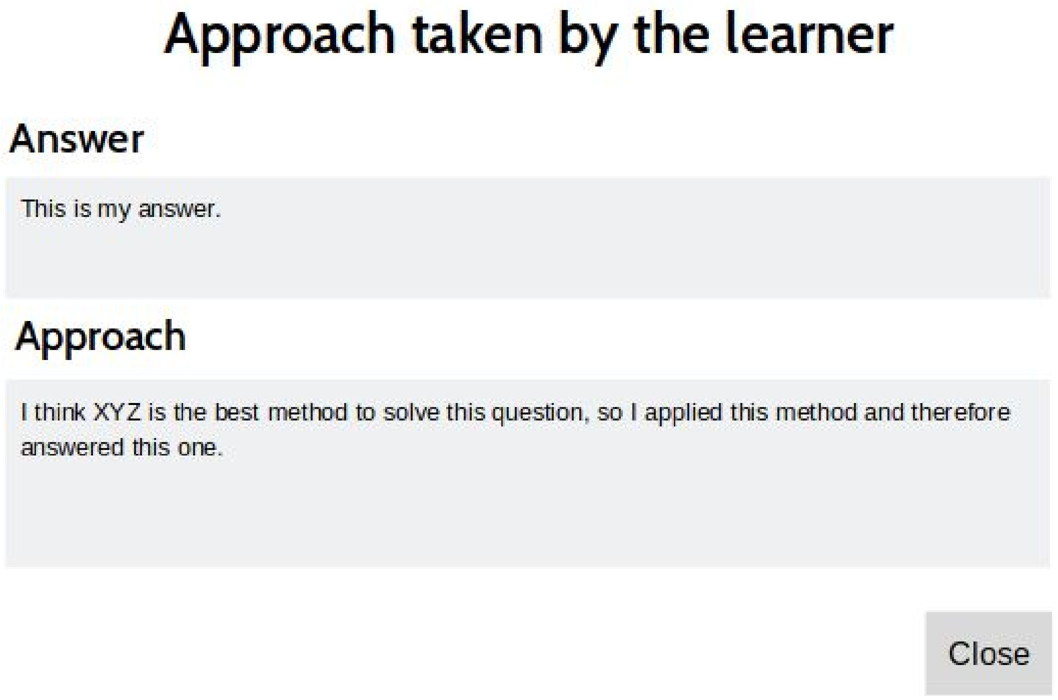
On clicking on the icon a modal with responses table will get open up.



| | No. | Answer | Explanation | Time |
|---|---|---|---|---|
| ☑ | 1. | This is an answer | Few lines of responses | 10.00AM |
| □ | 2. | This is an answer | Few lines of responses | 10.01AM |

On a single page, only 10 responses will be shown and the next one can be shown by moving to the next page(i.e the carousel feature). On clicking on each response a directive will be changed which will contain the complete response. And through a back button the directive will be  changed i.e. the table directive will be open.

After clicking on a particular response, model mentioned below will get open up.



Approach taken by the learner

Answer

This is my answer.

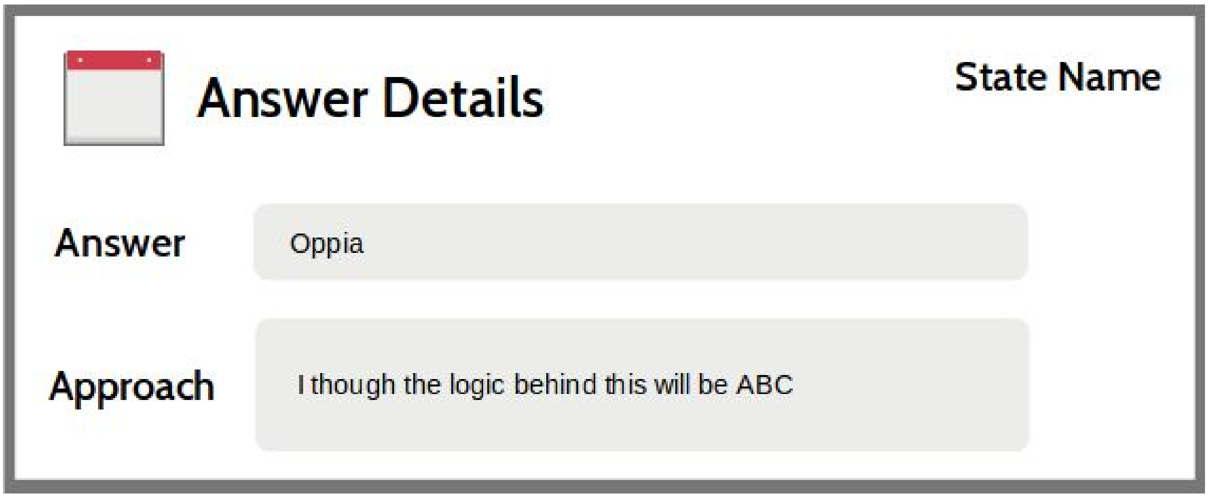
Approach

I think XYZ is the best method to solve this question, so I applied this method and therefore answered this one.

### Improvements Tab view:

A new card known as “Answer Detail” card will be implemented in the Improvements Tab, such that the creator will be needed to resolve the Cards received. By clicking on the card a modal window will get open which will include all the details related to the responses and in the modal window there will be a resolve button which will mark the response as resolved once clicked and this will remove the card from the Improvements Tab. This feature will be a kind of To-Do for the creators.

The UI for the card will be as follow, it will include the answer, approach and the state name.



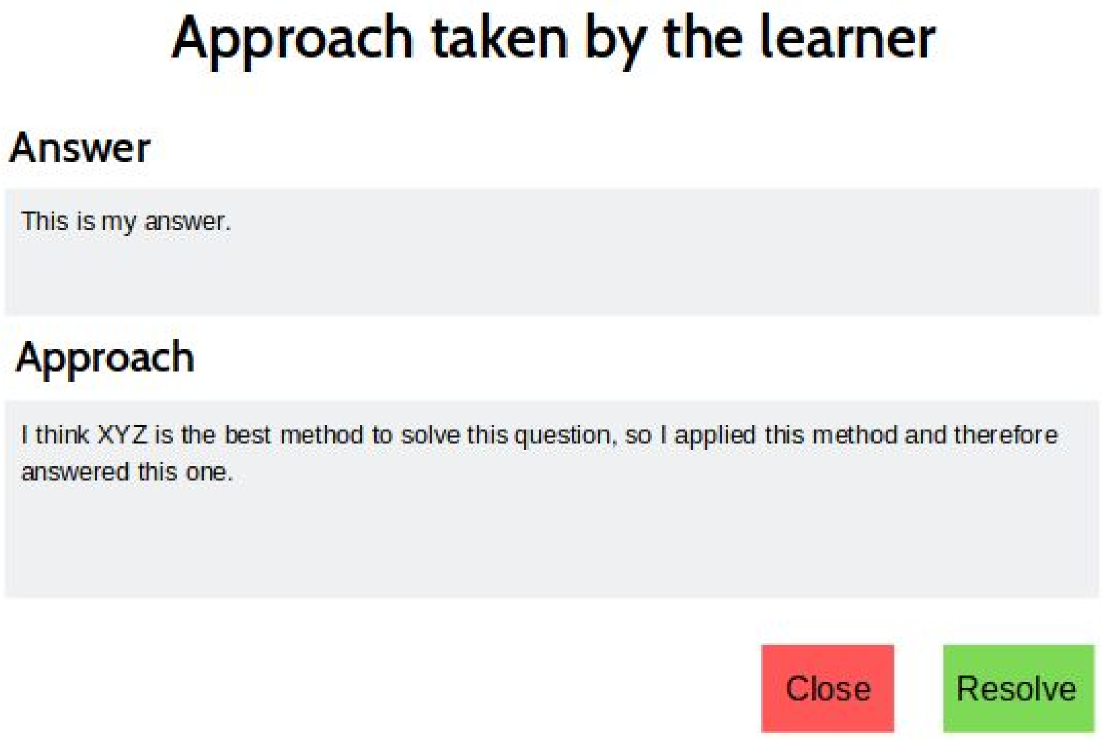
Answer Details

State Name

Answer: Oppia

Approach: I though the logic behind this will be ABC

Modal will open when the learner will click on the card



Approach taken by the learner

Answer

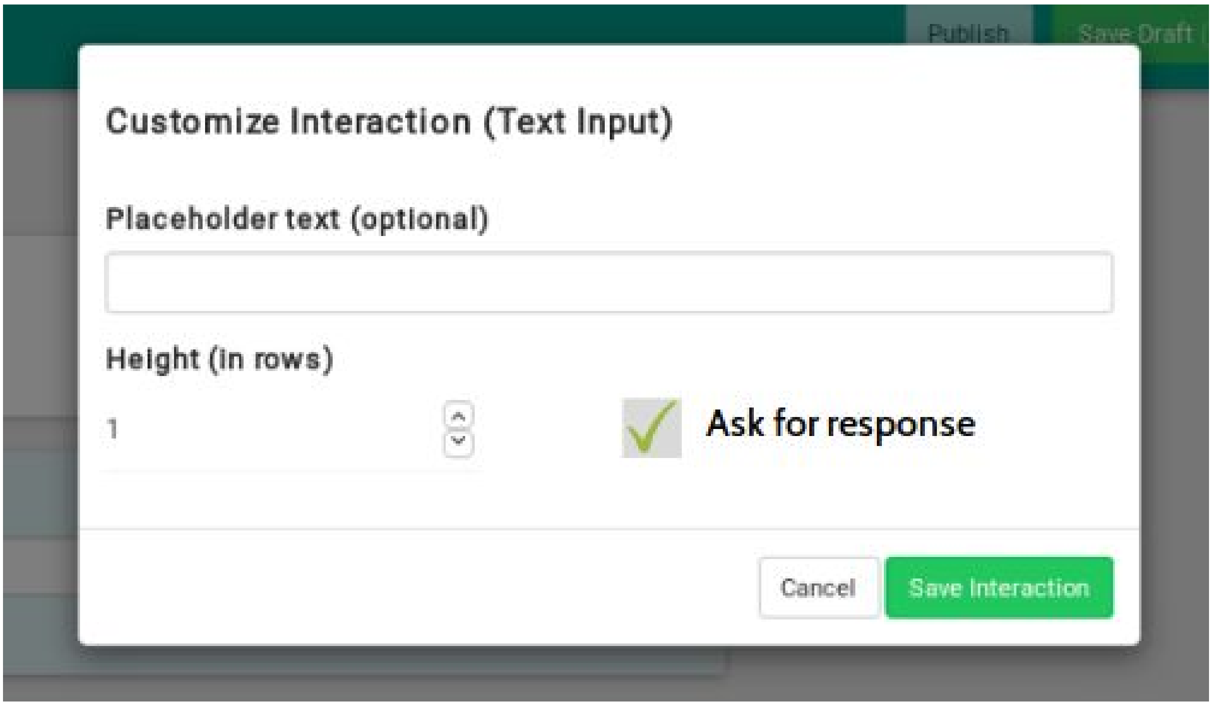
This is my answer.

Approach

I think XYZ is the best method to solve this question, so I applied this method and therefore answered this one.

## The option from where the creator can disable the ask learners for response option for a particular state:

A checkbox will be added which will ensure that learner wants to ask for a response in that interaction, and currently, this property can only be set for Text Input interaction. This will help the creator to get the response from that states only in which he has enabled this option.



Customize Interaction (Text Input)

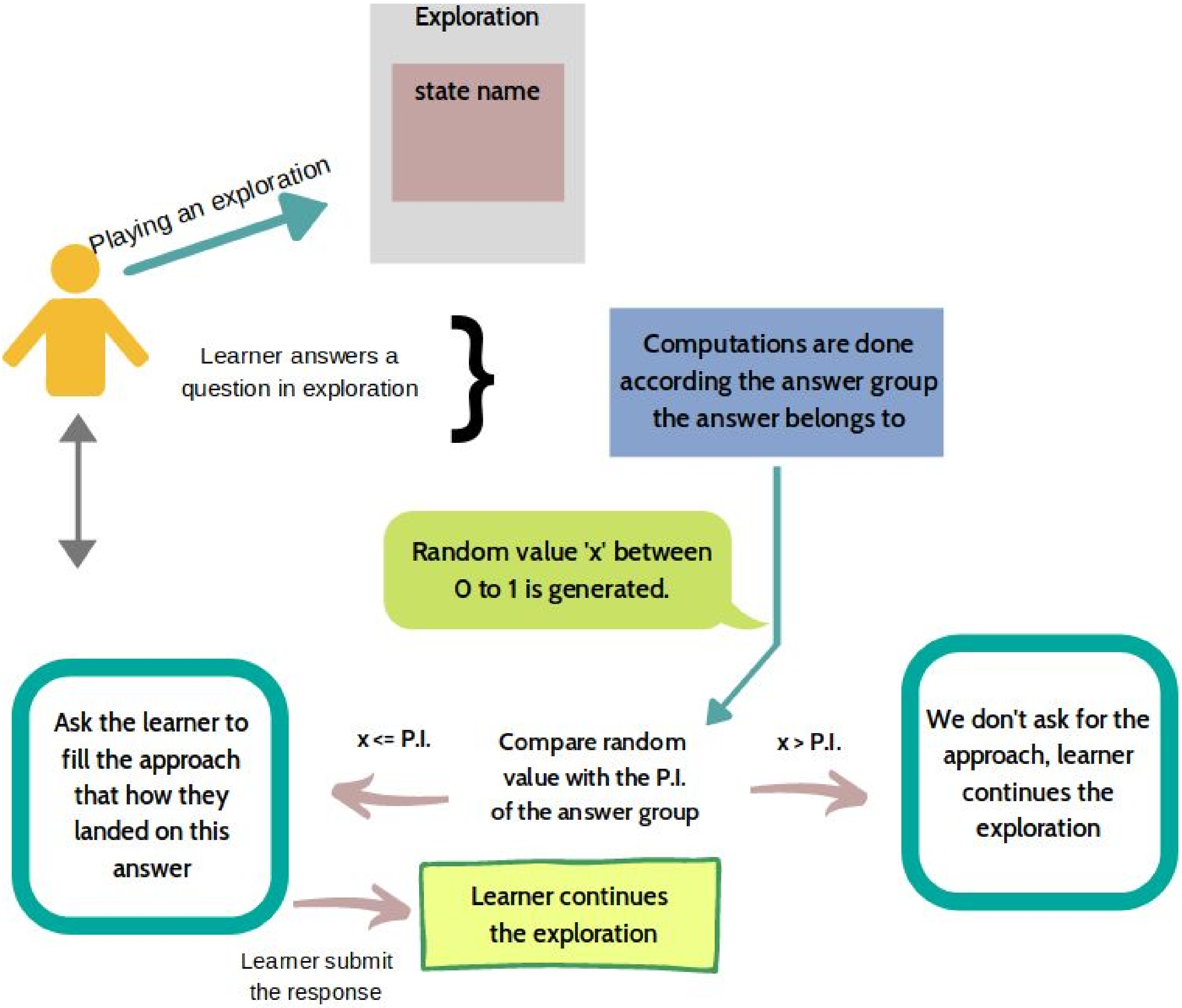
Placeholder text (optional)

Height (in rows)

1

☑ Ask for response

## Diagrammatic overview of the project:


Exploration
state name
Playing an exploration
Learner answers a question in exploration
Computations are done according the answer group the answer belongs to
Random value 'x' between 0 to 1 is generated.
Compare random value with the P.I. of the answer group
x <= P.I.
x > P.I.
Ask the learner to fill the approach that how they landed on this answer
We don't ask for the approach, learner continues the exploration
Learner continues the exploration
Learner submit the response

# Milestones:

## Milestone 1:

The creator can add an option, for each state, to ask learners for a response. The infrastructure for storing, retrieving and processing responses from the datastore is implemented.

1. The first task will be to add a ask_learners_for_response boolean variable to the State class in state_domain.py, which will ensure that in which state we have to ask for the response from the learners. Initially its assigned value will be false and if the creator will enable it then it will also get the true value, currently this action can be performed over Text Input Interaction only. The init method of the State class will be changed. Later on if required we can extend the same functionality over the other Interactions too.
   After that do the migrations of the state with the help of Writing the state migrations mentioned in the Oppia wiki, which will ensure that every state instance in the database has this boolean variable.
   - Edit the exploration editor page to add a checkbox for the Text Input Interaction such that the creator can enable it from the editor page.

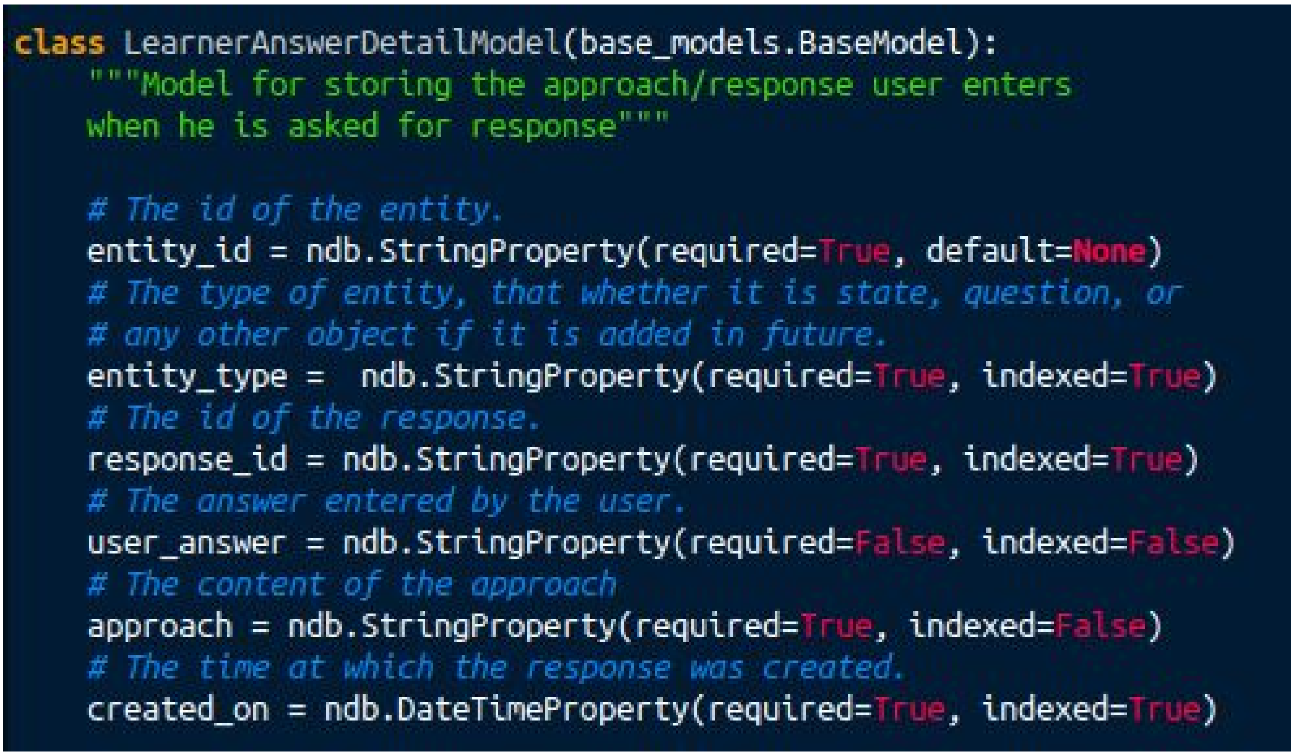
2. New model will be created in folder storage/statistics in file gae_modes.py for storing the response/approach received by the user and similarly the corresponding test will be added in gae_models.py. The name of the model will be **LearnerAnswerDetailModel**

   The instance of the model will be created every time a new response is received. Every response will be stored in different instances because:
   *One reason is that entities are limited to 1MB in size. If one post gets a huge number of comments, then there will be a danger of exceeding the limit and the code would crash.*
   *Another reason is that if we want to consider read/write rates for entities and scalability. If we use JSON, then we need update the LearnerApproachModel entity every time a comment is made. If a lot of people are writing comments at the same time, then we will need transactions and have contention issues. If we have a separate model for response, then we can easily scale to a million responses per second*!

   **Source:Stackoverflow**

   **LearnerAnswerDetailModel:**
   And the model will be as:

- Entity_id: This will contain the id of the entity. For the state of an exploration the id will be ‘exploration_id.state_name’ and for the question the id will be the question id only.
- Entity_type: This will be of string property and it will assigned to the constants that will be declared in the feconf.py file. Such that if the model is being used for the state then that constant for state will be assigned to the entity_type and for similarly for the question. Also this will help to generalise the model for the objects which will hold this model in future, just a constant will be added for that object in the feconf.py. Like currently these constants can be declared in the feconf.py file
    - THRESHOLD_ENTITY_STATE : ‘exploration_state’
    - THRESHOLD_ENTITY_QUESTION : ‘question’
- Response_id: The unique id of the response.
- User_answer: The answer entered by the user.
- Approach: The approach the learner has submit.
- Created_on: The time and date on which the approach was received.



```
class LearnerAnswerDetailModel(base_models.BaseModel):
    """Model for storing the approach/response user enters
    when he is asked for response"""

    # The id of the entity.
    entity_id = ndb.StringProperty(required=True, default=None)
    # The type of entity, that whether it is state, question, or
    # any other object if it is added in future.
    entity_type =  ndb.StringProperty(required=True, indexed=True)
    # The id of the response.
    response_id = ndb.StringProperty(required=True, indexed=True)
    # The answer entered by the user.
    user_answer = ndb.StringProperty(required=False, indexed=False)
    # The content of the approach
    approach = ndb.StringProperty(required=True, indexed=False)
    # The time at which the response was created.
    created_on = ndb.DateTimeProperty(required=True, indexed=True)
```

3. Corresponding services/functions for the this model will be created in the gae_models.py using the model queries.
    - create_response(entity_id, entity_type, user_answer, approach, created_on): Create the response with the given response content
    - get_responses_by_exploration_id(exp_id, entity_type) : It will fetch all the responses linked to the exploration.
    - get_response_by_state(exp_id, state_name, entity_type):It will fetch all the responses for a given state of an exploration.

- delete_response_by_exploration_id(exp_id, entity_type): It will delete all the responses of an exploration.
- delete_response_by_state(exp_id, state_name, entity_type): It will delete all the responses received for a state in an exploration.
- All the above model query functions will be called in response_services.py and its corresponding tests will be added in response_services_test.py
- These functions will get in sync with **_save_exploration** function in exp_services.py, such if any change in change in exploration is there changes in this model also occurs. Like if the state is deleted then all the response will also get deleted linked to that state, and if the exploration is deleted then all the responses of that exploration needs to be deleted.

## **Timeline for Milestone 1**:

| | |
|---|---|
| **Community Bonding Period**<br>7 May - 26 May | Finish ongoing PRs, communicate with my mentor to make any required changes to the implementation plan, and complete responsibilities for May release. |

My end semester exams will get over around 22 May, so I can start the work before 27th May.

| PR Details | Date for first draft | Date to be merged by |
|---|---|---|
| Add boolean value ask learners for response in the State class and write all its services and tests in state_services.py | 4-5 June 2019 | 8-9 June 2019 |
| Write the state migrations and a new one-off job (ExplorationMigrationValidationJob) | 12-13 June 2019 | 15-16 June 2019 |
| Implement the LearnerApproachModel in gae_models with all its corresponding services and backend tests | 18-19 June 2019 | 22-23 June 2019 |

### Jobs to be run in production:

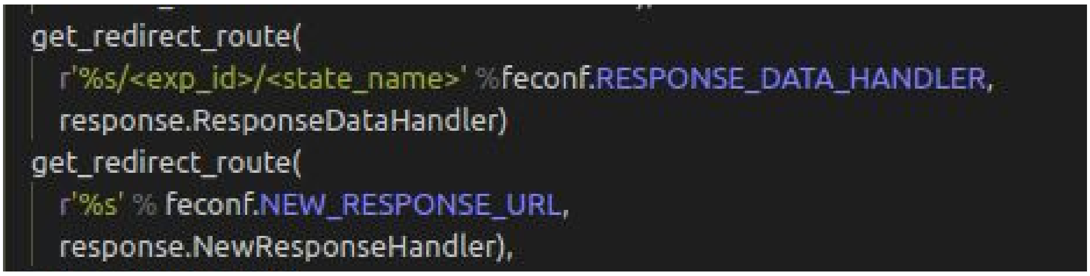
| Name of the job | Request submitted for running on test server | Request submitted for running on production server | Release Targeted |
|---|---|---|---|
| ExplorationMigrationValidationJob | 17 June 2019 | 22 June 2019 | July |

In case any error is reported on the test server, then there might be a delay of 3 days in the date of the submission of the request to run the job on the production server.

## Milestone 2:

Learners can submit responses for states (if the creator has asked for this)..

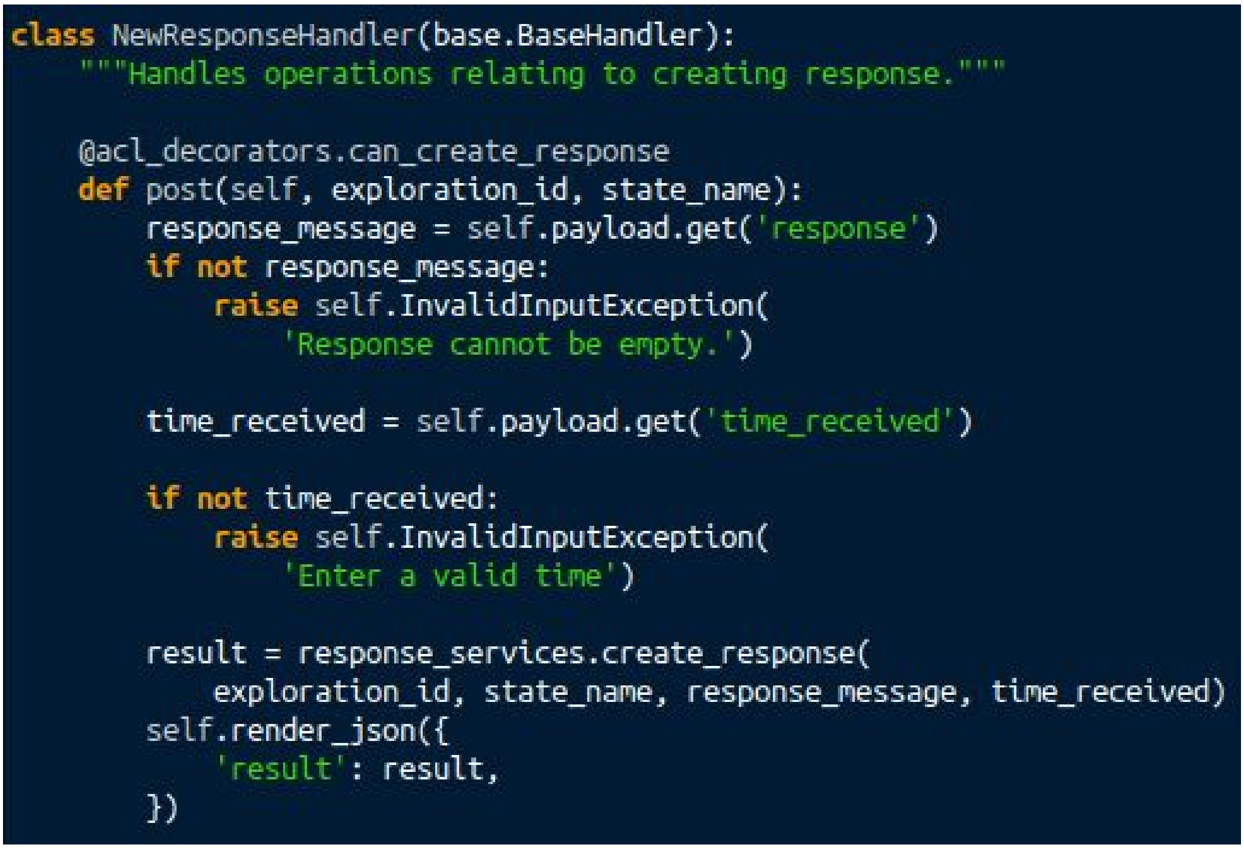
1. A file **response.py** will be created in core/controllers & corresponding response_test.py
    - New constants will be added in feconf.py
        - **RESPONSE_DATA_HANDLER**: ‘/response_data_handler’
        - **NEW_RESPONSE_URL**: ‘responsehandler/create_new_response’
    - New routes will be created in main.py for both the viewing the responses and storing it in database.



```
get_redirect_route(
  r'%s/<exp_id>/<state_name>' %feconf.RESPONSE_DATA_HANDLER,
  response.ResponseDataHandler)
get_redirect_route(
  r'%s' % feconf.NEW_RESPONSE_URL,
  response.NewResponseHandler),
```

    - In response.py two classes will be created.
        - Class **NewResponseHandler**: It will create the new response through post request, exp_id, state_name, response will be fetched using payload, which will use the services of approach_services.py to store response made by the learner.

- Class **ResponseDataHandler**: It will manage the responses to be displayed to creator for a particular state of an exploration. get(self, exp_id, state_name) will get the responses from the service get_responses_for_state. And will update the values and render it in json format.

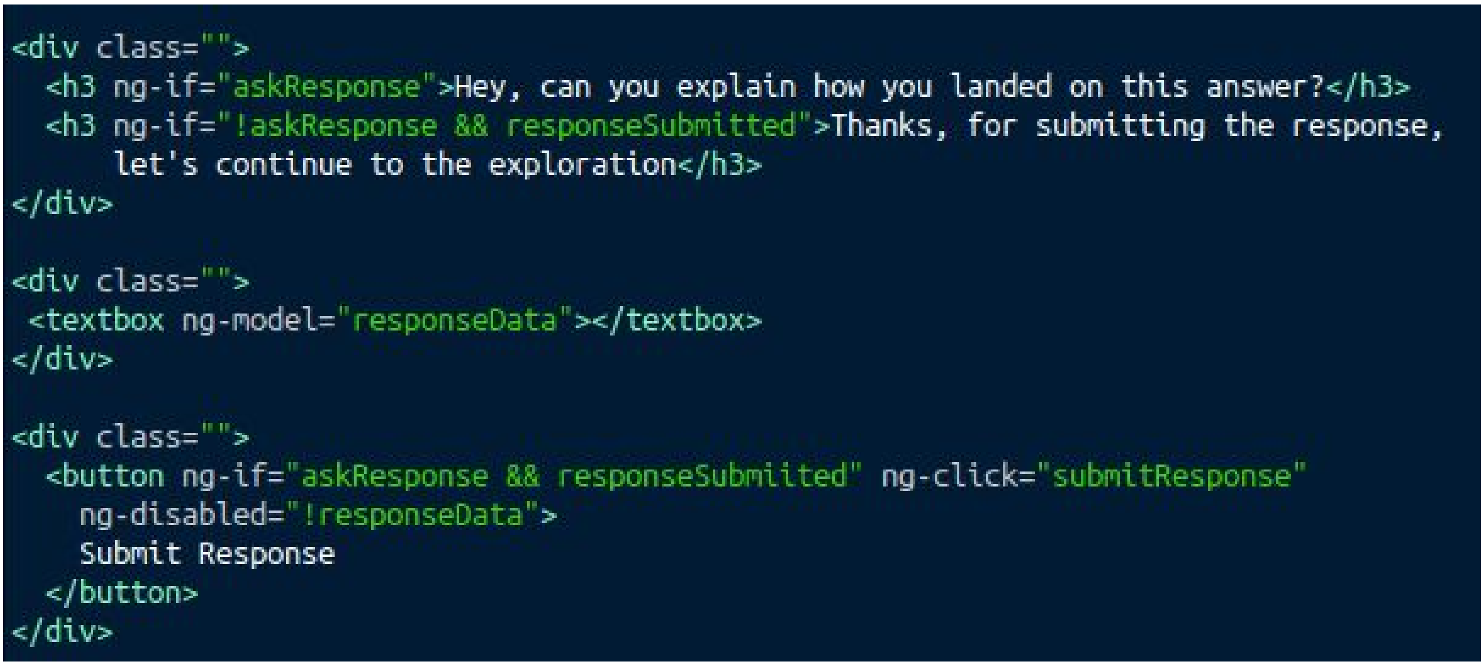


```
class NewResponseHandler(base.BaseHandler):
    """Handles operations relating to creating response."""

    @acl_decorators.can_create_response
    def post(self, exploration_id, state_name):
        response_message = self.payload.get('response')
        if not response_message:
            raise self.InvalidInputException(
                'Response cannot be empty.')

        time_received = self.payload.get('time_received')

        if not time_received:
            raise self.InvalidInputException(
                'Enter a valid time')

        result = response_services.create_response(
            exploration_id, state_name, response_message, time_received)
        self.render_json({
            'result': result,
        })
```

2. Now a HTML directive file will be created for the response form i.e. **response_form_directive.html** and its corresponding **ResponseFormDirective.js** will also be created. The response form directive will be added in the tutor_card_directive.html such that it will be viewable only if the value of $scope.askForResponse is true **(ng-if = “askForResponse && (responseSubmitCount<=2)”)**, by default the value of $scope.askForResponse will be false. When the learner will submit the answer, if the random value generated will be greater than the probability index of the answer group in which the learner answer lies then the learner will move to the next state, but if the generated value will be less than or equal to the probability index then the condition is triggered so before going to the next state a response form will get open below the answered question, such that learner will be asked to fill is approach and submit it.

The function $scope.submitAnswer in ConversationSkinDirective.js will be modified such that it loads the next state after the learned has submitted the response. Before going to the next state, and asking for response from the learner will help in getting the genuine response from the learner.
In this way the the response form will be shown to the learner.
Once the learner submits the response, a thanking message will be displayed to learner like “Thanks, for submitting the response, let's continue to the exploration”. Then the count of the submitted response will be incremented by 1 and learner will move to the next corresponding state.

**response_form_directive.html**(form in which the student will explain their approach)



```
<div class="">
  <h3 ng-if="askResponse">Hey, can you explain how you landed on this answer?</h3>
  <h3 ng-if="!askResponse && responseSubmitted">Thanks, for submitting the response,
      let's continue to the exploration</h3>
</div>

<div class="">
 <textbox ng-model="responseData"></textbox>
</div>

<div class="">
  <button ng-if="askResponse && responseSubmiited" ng-click="submitResponse"
    ng-disabled="!responseData">
    Submit Response
  </button>
</div>
```

3. Triggering the condition by generating the random value and comparing it with probability index of the answer groups will be done via frontend service **AskResponseService.js**. This service will be responsible for generating the random value. When the learner is playing an exploration and answering a question, and if the outcome of that answer group is the default outcome of the state then the probability index of that answer group will be 0.25 and various computations will be done accordingly. If the outcome is not the default one then the probability index will be 0.05 but if the outcome is labelled as correct then the probability index will be 0.10.
So when the learner will submit the response the variable **responseSubmitted** will be marked as true, such that no further response is

asked from that learner in that state for the current session. The detail of the exploration will be fetched from backend that whether the creator has enabled the option of ask learners for response for that particular state or not. If the creator has enabled it, then only the following tasks will be done to trigger the response form to the learner.

- There will be a $scope.responseSubmitted will be added in ConservationSkinDirective.js, which will ensure that the response is not asked again from the learner in that session again for that particular state.

- A count will be maintained for a session which will help to not ask the learner for response more than 2-3 times.

4. The response form will be loaded when the **askForResponse** is marked true and **responseSubmitted** is marked false then the response form directive will be loaded, if the learner fills the approach and submit it **askForResponse** will be marked false and **responseSubmitted** will be marked true for that session in a particular state.

5. When the learner will submit the response, the **responseSubmitted** will be marked as true for that session of that particular state, and data will be sent through post request to the new response handler, such that the new response is stored in the database. After that a thanking message will be displayed which will say the learner “Thanks for submitting the response, let’s move back to the exploration.”

6. Testing of the frontend services(adding the test files), built during Milestone 2 and overall testing of the learner side will be done.

## Timeline for Milestone 2:

| PR Details | Date for first draft | Date to be merged by |
|---|---|---|
| Implementing the backend controllers for handling the task related to responses | 1-2 July 2019 | 4-5 July 2019 |
| Implementing the UI for asking the response from the learner and creator will be able to disable this option from the advance features in the settings tab | 7-8 July 2019 | 10-11 July 2019 |

| | | |
|---|---|---|
| Making frontend service to trigger the condition of asking the response from the learner. | 13-15 July 2019 | 16-17 July 2019 |
| Linking all the aspects to the exploration player and testing whether the learner is able to fill the response form | 18-19 July 2019 | 21-22 July 2019 |

## Milestone 3:

The creator can view all responses received for a state, and can resolve a set of responses by creating a new answer group or updating an existing one.

1. Now the responses viewed will be shown to the creator in the Improvements Tab. The tab now consists of Feedback Cards and Playthrough Cards, so a new card will be implemented here which will be named as Answer Detail Cards.
   The fields the card will include will be:
   - Answer - Entered by the user.
   - Time - Card was received.
   - Approach -  The approach filled by the user.
   - State -  The name of the state for which the response was received.

   So to create this card **answer_detail_card_directive.html** will be made in the improvements_tab folder & correspondingly AnswerDetailCardDirective.js will be created which will use the AnswerDetailService.js to open the modal window and mark the response as resolved.

2. Additional to that there will be a “Resolve” button on the card which will help to resolve the Answer Detail card i.e it will make the is_resolved true of that response such that the response will not be shown in the Improvement Tab again. The response id will be sent to the backend which will mark the is_resolved as true.

3. Other than that responses will be shown in the exploration editor also, just beside the the answer group there will be a note icon, such that when we click on that note icon a modal window will get open which will contain the responses which lie in that answer group. This will act as a reference for the

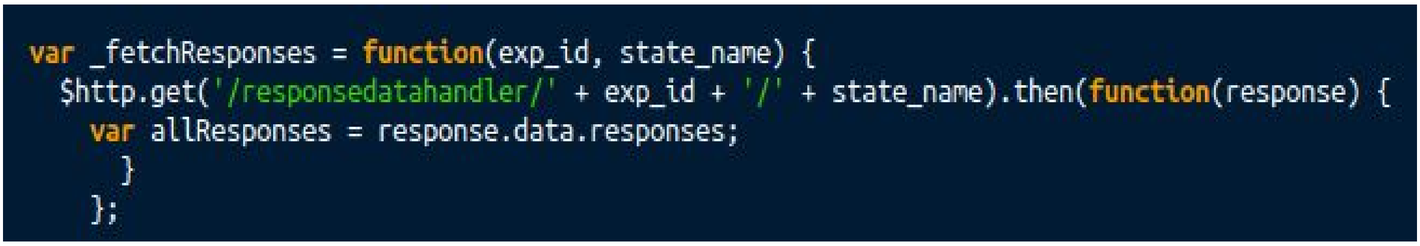
creator to improve the answer groups in future. A new modal directive will be made which will contain the table and show up the responses received.

4. The creator will be able to delete the responses from the modal window, such that he can select the responses which he want to delete or he can click on the option “Select all” and then delete if he wants all the responses to be cleared for that answer groups. He will be able to view the full response with the answer which was entered by the user by clicking on the response.

5. The user answer will be first checked through the answer group rules to know in which answer group the response lies so that the responses can be sorted according to the answer groups.

6. A new directive **response_table_modal_directive.html** will be made which will be responsible for showing the responses received in an exploration for a state in an answer group. In the table there will be limit, such that only 10 responses will be shown and there will be a **carousel feature** such that creator can click on the arrow button to move to the next page and view the other 10 responses and similarly next, this will help to increase creator UX.
   The overview of the HTML file:

7. **ResponseTableDirective.js** will be created, which will be responsible to display the data in the html directive. The responses will be fetched via **ResponseDataService.js**, which will be responsible for get request to fetch all the responses related to a state of an exploration.



```
var _fetchResponses = function(exp_id, state_name) {
  $http.get('/responsedatahandler/' + exp_id + '/' + state_name).then(function(response) {
    var allResponses = response.data.responses;
      }
    };
```

   **DateTimeFormatService** will be used for to display the time the response was received.
   $scope.getLocaleAbbreviatedDatetimeString=
   (DateTimeFormatService.getLocaleAbbreviatedDatetimeString

8. Controllers needs to be implemented for deletion of response & marking the response as resolved. Controllers will be made such as

**ResponseDeleteHandler** : It will handle post request with parameters exp_id, state_name and the response ids list which needs to be deleted will be passed as parameter which can be fetched through payload.

**ModifyResponseHandler**: It will help in modifying the response i.e. marking a response as resolved(It will be a general controller used for modifying the response, as per now it's only functionality will be to mark the response as resolved, but if the features of the responses are increased in future then we can perform the modification with this handler only). A constant MARK_RESPONSE_AS_RESOLVED will be send as parameter along with response id, state name, exp id and the note_message, hence in the controller if-condition will be checked for the constant.

Feconf.py will contain a constant : **DELETE_RESPONSE_HANDLER** : '/responsedeletehandler'
**MODIFY_RESPONSE_HANDLER**: '/modifyresponsehandler'
And the new routes for both the handlers will be added in the main.py

9. After clicking on the response (**ng-click = "onClickRow(response.id)"**) a modal will be opened such as the creator can view the response on which he has clicked. **ResponseServices.js** will be created which will handle the deletion part and other modification parts (like marking the response as resolved) related to the responses.
   For the deletion of responses, the explorationId, stateName, and the list of responseIds will be sent to the controller such that responses gets deleted after that.
   **deleteResponse(expId, stateName, responseId)** : which deletes a particular response.
   **deleteMultipleResponses(expId, stateName, responseIds)**: will delete multiple responses.
   **markResponseAsResolved(expId, stateName, responseId)**: will mark the response as resolved.

10. Testing of the frontend services(adding the test files), built during Milestone 3.

11. Creating e2e tests & overall testing the full cycle of submitting the response, till viewing, resolving and deleting it.

## Timeline for Milestone 3:

| PR Details | Date for first draft | Date to be merged by |
|---|---|---|
| Add a card for answer details in the Improvements Tab | 27-28 July 2019 | 30-31 July 2019 |
| Implement the UI in the editor tab which includes the table to show the responses | 9-10 August 2019 | 15-16 August 2019 |
| Adding e2e tests & testing the overall cycle of the functionality i.e. from submitting the response to viewing the response, resolving and deleting it. If any error seems to come then fix it. | 18-19 August 2019 | 20-21 August 2019 |

## Documentation:

A Google Doc will be maintained throughout all the Milestones, such that it will include the documentation which needs to be added at each milestone. So at last the Google Doc will contain all the documentation that needs to be published. It will be submitted by 22 August 2019.

## Alternatives Considered:

**Note**: These alternatives were also thought while making the proposal.

1. For viewing the response form, we can hide the current state i.e. conversation skin directive, whenever there is a need of submitting the response form. Such that ng-hide directive will be used to hide the conversation skin if the askForResponse is true, whereas the response form will use the ng-if directive, such that if askForResponse is true and responseSubmitted is false, than the response form will be shown.

2. The other thing we can provide the response form is by using modal window. Such that a modal window will get opened up, when the learner needs to submit the response/approach.

3. The view section can be shifted to the Statistics Page.

## Future Extensions of project (will be taken after completion of the GSoC period):

1. Extend the asking for approach model for the Questions in Oppia. As we are now working on Questions, so asking for an approach can be extended to those Questions.

2. Implement this feature to other Interaction too.

3. There will be a button in the Settings Tab of the exploration editor page, this feature will be available in the Advance Features option. The creator will be able to disable the aks learners for response option for the whole exploration, by that button such that if in future he doesn't want to get responses to the approaches made by the students. It will help the creator if he has enabled this feature for many states, then rather going state to state and disabling it he can directly disable it for all the states i.e. for the whole exploration from that option.

   Add a button in the settings tab of the exploration editor such that it will disable all the ask_learners_for_response enable instances in all the states of the exploration. Such that when the button will be clicked a warning modal window will get open which will ask the creator that if he wants to disable the option for all the states. If the creator clicks yes then the ask_learners_for_response will be set to False for all the States instance in that exploration. A update function will be declared in the State class in the file state_domain.py
   A new constant will be added in the exp_domain.py
   **STATE_PROPERTY_ASK_FOR_LEARNERS_RESPONSE = 'ask_learners_for_response'** which will be responsible for changing the state properties when the exploration is changed via apply_change_list function in exp_services.py.

## Time Zone where I will be able to commit to the project :

Indian Standard Time(IST) which is ahead of UTC by 5 hours and 30 minutes.

## Time which I will be able to commit to the project :

I will be able to devote 8-10 hrs a day, and about 5-6 hrs during the weekend. i.e. approximately 55-60 hrs in a week. In August my classes will be started hence time spend during weekdays will be lesser i.e 5-6 hrs while during weekend 8-10 hrs.

## What jobs, summer classes, and other obligations might you need to work around :

I have no commitments during the summer. I have also no classes or lectures till the end of July, my classes will start in August so my number of hours might get less during the weekdays, but I will cover it during the weekend.

However two days might be needed for traveling one around 24 May and other around 20 July, which I will cover by extending my work hours.

## Communication:

My contact information:

Email: anubhavsinha98@gmail.com

GitHub Handle: anubhavsinha98

I am also active on Gitter & Hangouts, by the way I am okay with any communication channel.

## How often, and through which channel(s), do you plan on communicating with your mentor?

I will be comfortable with any mode, be it Gitter or Hangouts and I am willing to choose any mode used by the mentors.

I will also plan to maintain a daily record of my progress.

I will be in continuous touch with the mentors, we can have biweekly(twice a week or as proposed by the mentor) meetings to discuss the workflow. And I am okay with any platform used for meetings.