Single Line Item Type Tech Spec

[I. Item Type Overview 2](#_Toc1955363407)

[A. JSON Structure 2](#_Toc876522105)

[B. Other Database Fields Used 3](#_Toc1360709494)

[C. Scoring Types Used 4](#_Toc727734378)

[D. Item Validation Rules 4](#_Toc1547882593)

[E. Third Party Libraries Used 4](#_Toc1436400280)

[II. Item Content and Response View 5](#_Toc1488408428)

[III. Technical Updates 8](#_Toc1584409360)

[A. Outstanding Defects 9](#_Toc1520631351)

[B. Business Flow 9](#_Toc672365560)

[C. Implementation 11](#_Toc1728776442)

[1. Create 11](#_Toc1900301629)

[2. Response 14](#_Toc453088481)

[3. Preview 21](#_Toc51736806)

[4. Stories 22](#_Toc817659960)

[D. Software Updates Needed to Align with Project Standards 23](#_Toc477073114)

[E. How to consume 23](#_Toc465852353)

[F. Click History 24](#_Toc1695589239)

[IV. Test Scenarios 24](#_Toc1791955552)

[V. Final Effort Estimate 27](#_Toc375433307)

[VI. Questions 27](#_Toc204433930)

1. Item Type Overview
   1. JSON Structure

Single Line JSON structure below.

{

"item": {

"item\_json": {

"uuid": "522c10-4adc-03c2-4be-361fbb304127",

"slope": true,

"width": 500,

"height": 500,

"maxLines": 1,

"showGrid": true,

"lineArrow": false,

"bottomLeft": "-10, -10",

"graphTitle": "<p>Graph Title</p>\n",

"snapToGrid": true,

"upperRight": "10, 10",

"xAxisLabel": "<p>X Axis Label</p>\n",

"xAxisScale": 1,

"xAxisTitle": "<p>X Axis Title</p>\n",

"xIntercept": false,

"yAxisLabel": "<p>Y Axis Label</p>\n",

"yAxisScale": 1,

"yAxisTitle": "<p>Y Axis Title</p>\n",

"yIntercept": false,

"stemContent": "<p>Stem content for straight line</p>\n",

"itemTypeCode": "S1L",

"minItemWidth": 0,

"minItemHeight": 0,

"orderPairOne": true,

"orderPairTwo": false,

"extendedLines": true,

"incrementLabel": "ALL",

"lineRelationships": false,

"graphBackground": "/media/15009/400579/4001864/1/kite-04.jpg",,

"backgroundPosition": {

"x": -107,

"y": 45

},

"backgroundMediaDetails": {

"id": 400579,

"fileName": " kite-04.jpg”

"contentType": "media",

"attachmentId": 4001864,

"attachmentVersion": 1,

"assessmentProgramId": 15009

}

“axisLineSize”: 5,

“axisLabelSize”:7,

"correctResponse": [

{

"id": "response-id-1",

"value": {

"line": [

{

“x": 12,

"y": 6

},

{

"x": 6,

"y": 0

}

]

}

}

]

},

"rationaleOption": {

"optionList": []

}

}

}

* 1. Other Database Fields Used
  + We are not using any other specific database fields for this item type other than the standard field for an item.
  1. Scoring Types Used
* The Scoring Types for Single Line are

1. No Scoring needed
2. Correct Only
3. Partial Credit
   1. Item Validation Rules

These are the validations used in this item

* The Stem, Passage is required fields, must not be empty.
* SCORE\_METHOD\_REQUIRED- Scoring Method is required.
* CORRECT\_ANSWER\_REQUIRED\_LEN- For Any Scoring Method, Correct Answer is required. Must select at least one response as correct.
* MAX\_SCORE\_REQUIRED - Maximum Score is required and it cannot be zero. Validation is not required for No Scoring.
* SCORE\_NEGATIVE – Scoring value cannot be negative.
* SCORE\_EXCEED - Total score cannot be greater than maximum score.
* SCORE\_DECEED - Total score cannot be less than maximum score.
* SUM\_TO\_MAX\_SCORE – For partial credit, only sum of score should be equal to max score.

**Common Validations**

* In Attributes and Metadata accordion Content Area, Grade Level, Framework, Cognitive Taxonomy, Cognitive Category, Testing program, and Usage validations are included for this item.
* ERR\_MISSING\_CONTENT\_AREA – Should have content area when we save item other than draft state.
* ERR\_MISSING\_GRADE – Should have grade level when we save item other than draft state.
* FRAMEWORK – Should have framework when we save item other than draft state.
* CONTENT\_CODE\_SUGGESTED – At least one primary content code should be selected.
* COGNITIVE TAXONOMY - Should have taxonomy when we save items other than draft state.
* COGNITIVE CATEGORY - Should have cognitive category when we save item other than draft state.
* TESTING PROGRAM - Should have a testing program when we save items other than draft state.
* Mathematical Practice – This validation is for Mathematics content area. At least one primary content code should be selected.
  1. Third Party Libraries Used
* In New CP, Single Line item can be developed by using chart.js(3.8.0), chartjs-plugin-datalabels(2.0.0) and chartjs-plugin-dragdata(2.2.4).
* To handle draggable functionality for the image in graph, we can go for react-draggable with latest version as 4.4.5.

React Draggable - <https://www.npmjs.com/package/react-draggable/v/4.4.5>

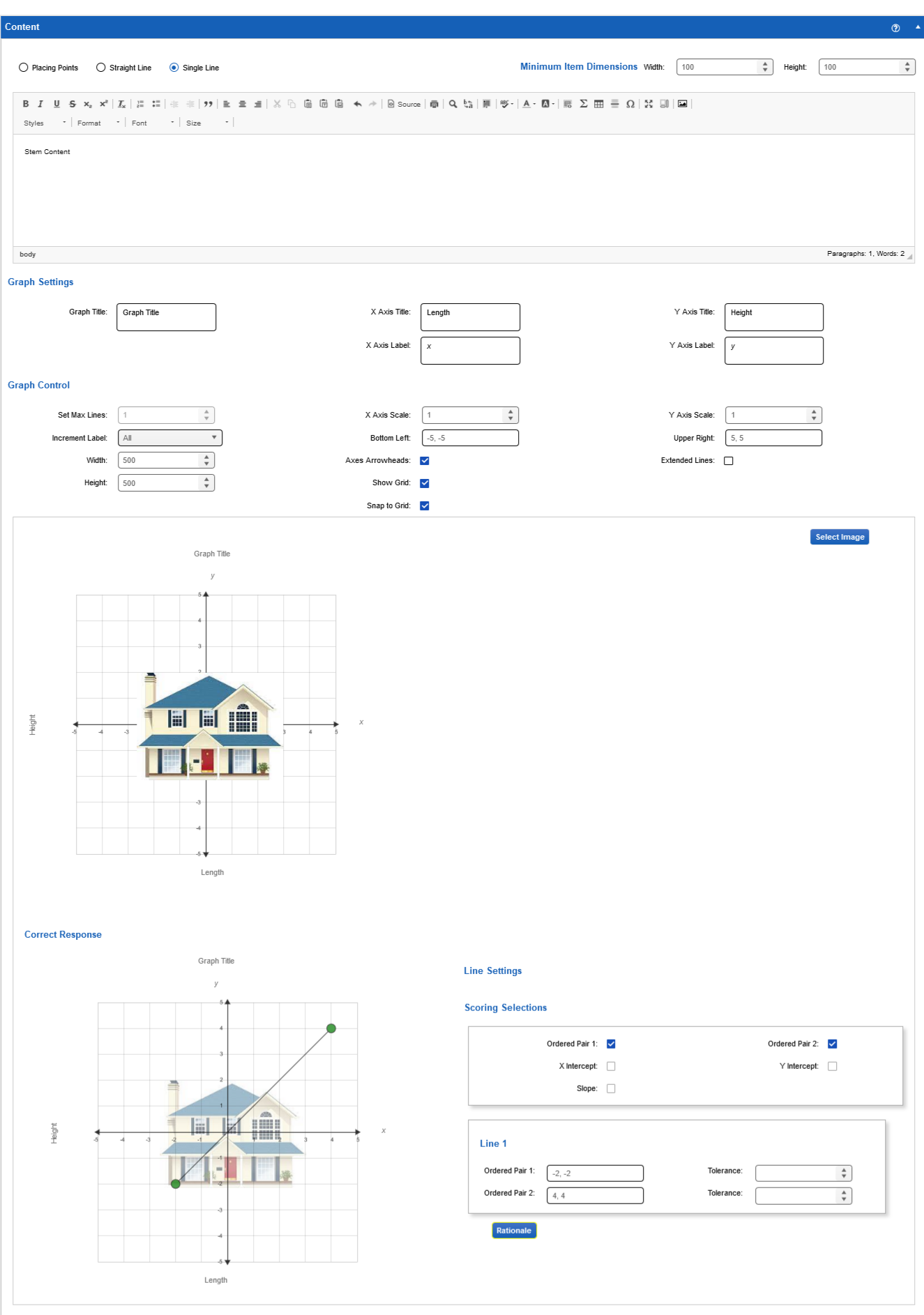
Chart.js - <https://www.npmjs.com/package/chart.js/v/3.8.0>

Chartjs plugin datalabels - <https://www.npmjs.com/package/chartjs-plugin-datalabels/v/2.0.0>

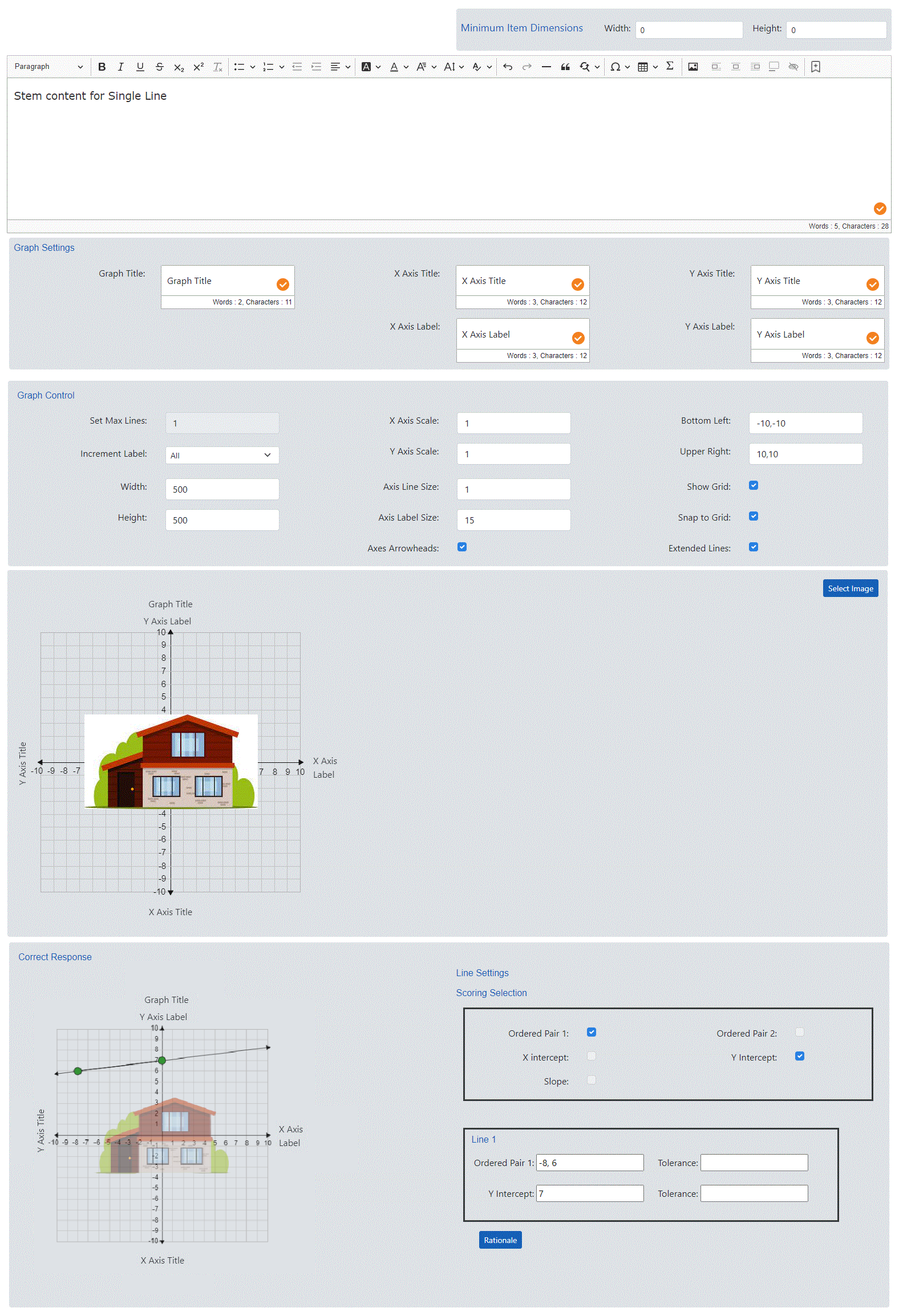
Chartjs plugin dragdata - <https://www.npmjs.com/package/chartjs-plugin-dragdata/v/2.2.4>

1. Item Content and Response View

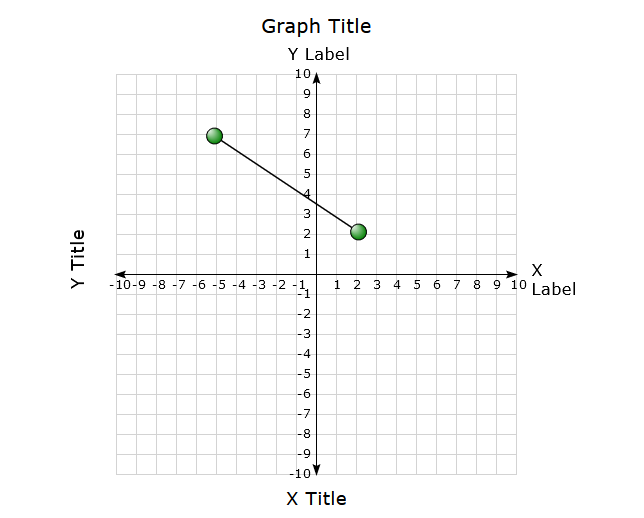
* Current CP – Content Page.



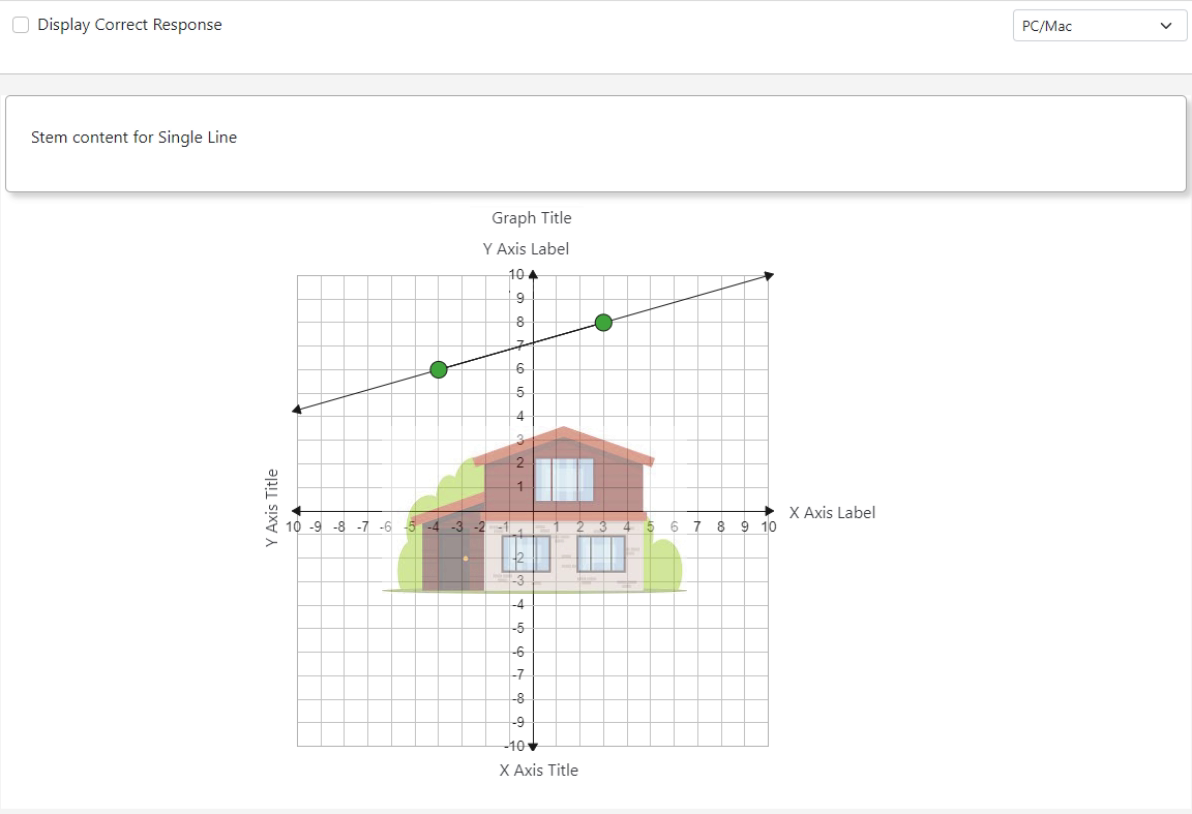
* New CP – Content Page



* Old CP – Preview



* New CP – Preview



1. Technical Updates
   1. Outstanding Defects

* X and Y Axis Titles do not follow movement of axis when bottom left, and upper right are adjusted. Title should be along the corresponding axis, and the labels should be at the end of the x and y axis lines. -- Already taken care while developing Straight line item.
* Extended line arrow point rotation is not straight when bottom left and upper right have different values (i.e: BottomLeft – (-5,-10) and UpperRight – (10, 6)).
* Console error while clicking over the graph (plugin error). -- Already taken care while developing Straight line item.
* US19753 Unable to adjust the thickness of the axes to differentiate them from the grid lines – we need separate field to increase the thickness of the axis line. -- Already taken care while developing Straight line item.
* US19754 Unable to format the increment labels to adjust sizing – plugin having some restriction. -- Already taken care while developing Straight line item.
  1. Business Flow
* The Min item dimensions functionality will be same as every item.
* Stem content will have the questions.
* Users can set the titles and labels for the graph.
* Based on titles and labels given in Graph setting area same will be populated in the correct response area/preview.
* The user can change the graph controls options from default values (i.e., X Axis scale = 1, Y Axis Scale = 1).
* The value for set max lines will be 1 and it will be in disabled mode.
* We can change the X Axis and Y Axis scale values.
* We can choose labels for the graph from the incremental label dropdown option.
* The bottom left and upper right can be changed (I.e., Bottom left = [-10, -10], Upper right = [10,10]), bottom left is to specify the left and bottom side of the axis line of the graph and the upper right is to specify top and right-side axis line of the graph.
* The height and width of the graph can be changed from default values. (I.e., Height = 500 and width = 500).
* The User can enable / disable Axes Arrowheads, show grid, snap to grid options, which needs to be shown in the graph.
* Users can avail themselves of the extended line option by enabling the checkbox.
* Users can adjust the thickness of the axis line using the Axis Line Size field. Default value will be I.e., Axis Line Size = 1.
* Users can adjust the thickness of the axis label using the Axis Label Size field. Default value will be I.e., Axis Label Size = 10.
* Users can add an image by clicking on the select image button and able to drag the image to any position in graph.
* The User can enable / disable Ordered Pair1, Ordered Pair2, X Intercept, Y Intercept and slope.
* If Ordered Pair1 and Ordered Pair2 are enabled then user can give value in Ordered Pair1, Ordered Pair2 and tolerance field. Also, X Intercept, Y Intercept and Slope will be in disabled mode.
* If Ordered Pair1 and X Intercept are enabled then user can give value in Ordered Pair1, X Intercept and tolerance field. Also, Y Intercept, Ordered Pair2 and Slope will be in disabled mode.
* If Ordered Pair1 and Y Intercept are enabled then user can give value in Ordered Pair1, Y Intercept and tolerance field. Also, X Intercept, Ordered Pair2 and Slope will be in disabled mode.
* If Ordered Pair1 and Slope are enabled then user can give value in Ordered Pair1, Slope and tolerance field. Also, X Intercept, Y Intercept, and Ordered Pair2 will be in disabled mode.
* If Ordered Pair2 and X Intercept are enabled then the user can give value in Ordered Pair2, X Intercept and tolerance field. Also, Y Intercept, Ordered Pair1 and Slope will be in disabled mode.
* If Ordered Pair2 and Y Intercept are enabled then user can give value in Ordered Pair2, Y Intercept and tolerance field. Also, X Intercept, Ordered Pair1 and Slope will be in disabled mode.
* If Ordered Pair2 and Slope is enabled then user can give value in Ordered Pair2, Slope and tolerance field. Also, X Intercept, Y Intercept, and Ordered Pair1 will be in disabled mode.
* Users can choose two points to draw a line.
* User can drag the points to change the co-ordinate of the point.
* The User will be able to plot two points in the preview to draw the line.
* If display correct response is enabled, then correct response will be set in the Preview and user will not be able do any action on it.
  1. Implementation
     1. Create
* A folder will be created inside the src/components/create/singleline with the respective item type name.
* Then creating a component for the item type with SingleLine.jsx extension in the respective folder.
* This SingleLine.jsx is used to render item content creation.
* CSS file for the respective item type will be placed inside styles/item folder.
* The shared components for the minimum item dimensions and stem content will be used here.

import ItemDimensions from '../shared/ItemDimensions';

import StemContent from '../shared/StemContent';

<ItemDimensions

minWidth={item?.item\_json?.minItemWidth || 0}

minHeight={item?.item\_json?.minItemHeight || 0}

onChange={(dimension) => {

if (dimension?.minWidth !== item?.item\_json?.minItemWidth) {

updateItemJson('minItemWidth',dimension.minWidth);

}

if (dimension?.minHeight !== item.item\_json.minItemHeight) { updateItemJson('minItemHeight',dimension.minHeight);

}

}}

/>

<StemContent data={item.item\_json?.stemContent}

onUpdate={updateItemJson}

fieldName={'stemContent'}

/>

* A shared component GraphSettings.jsx from components/create/shared/lineitem folder and used for rendering the Titles and Labels field.

import GraphSettings from '../shared/lineitem/GraphSettings';

//Graph Settings will be called by passing all the required parameters

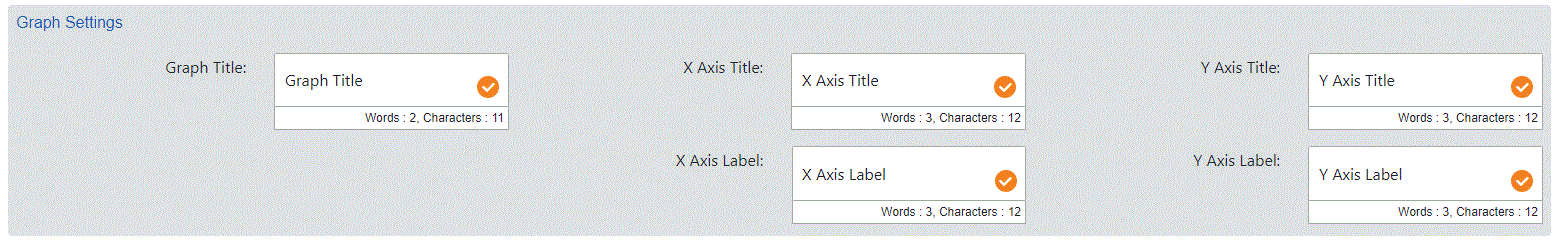
<GraphSettings

Content={item?.item\_json}

onUpdate = { onUpdate }

config={config}

/>



* A shared component GraphControl.jsx from components / create / shared/lineitem folder will be used in SingleLine.jsx for rendering graph input operations and values.
* The set Max Lines field will be in disable mode with value as 1.

import GraphControl from '../shared /lineitem/GraphControl';

//Graph Control will be called by passing all the required parameters

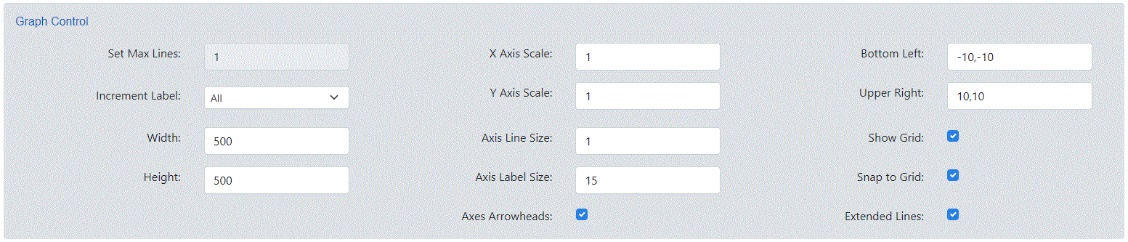
<GraphControl

content = { item?.item\_json }

onUpdate = { onUpdate }

config = { config }

/>



* A new sub-component SingLineImage.jsx will be created for adding the image in graph and all dragging functionality will be handled in that component.
* To add image, will be using shared component from ‘../shared/AddMedia’

Import AddMedia from ‘../shared/AddMedia’;

Import Draggable from ‘react-draggable’;

<AddMedia

setMedia={onAddImage}

media={filteredMedia}

cancelMedia={() =>

setSelectMedia({

show: false

})

}

<Draggable bounds="parent" defaultPosition={backgroundPosition} position={backgroundPosition} axis="both" onDrag={tracePosition} onStop={tracePosition} >

<div className="ci-draggable-bg-image-cp">

<img draggable={false} src={graphBackground} alt={ALTTEXT} />

{ (backgroundPosition.x !== x && backgroundPosition.y !== y && (x > 0 || x < 0) && (y > 0 || y < 0))

&& setBackgroundPosition({ "x": x, "y": y })

}

</div >

</Draggable >

* A new sub-component CorrectResponse.jsx will be created to render the ItemResponse component (Graph) and scoring selection section.
* A shared component ItemResponse.jsx from ../../display/response/graph/itemresponse from will be used in CorrectResponse.jsx for rendering graph.
* A shared component ScoringSelection.jsx from components / create / shared/lineitem folder will be used in CorrectResponse.jsx for rendering scoring selection section.

Import ScoringSelection from '../shared/ ScoringSelection;

//Scoring Selection will be called by passing all the required parameters

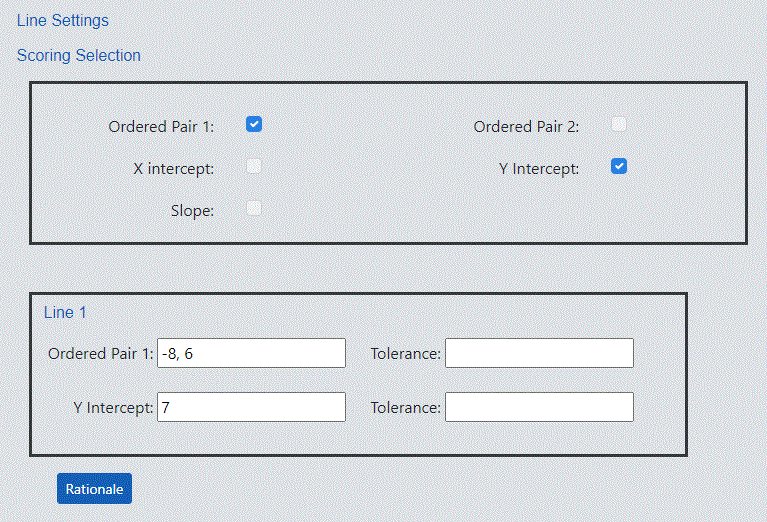
< ScoringSelection

content={item?.item\_json}

onUpdate={onUpdate}

config={config}

/>



* Required constants, labels, and default values will be added in the appropriate files.
* Test files will be created for all the component files.
* After implementation, the SingleLine component is imported and mapped with unique item code in CreateInterface.jsx file which will be in components/create folder.

import SingleLine from './singleline/ SingleLine;

const componentManifest = Object.freeze({

S1L: < SingleLine {...payload} />

});

* + 1. Response
       - A shared component ItemResponse.jsx from /display/response/graph/ItemResponse.jsx will be used for single line correct Response(Graph).
       - ItemResponse.jsx is the main component of responses that will receive the inputs and render the correct response section.
       - The Following method will Identify the graph boundary and calculate the dummy points to draw arrowheads. Based on the points given, the endpoint will be calculated.

const infiniteLine = (pointA, pointB, bottomLeft, upperRight) => {

let points = null;

if (pointA && pointB) {

let finiteA = { dummyPoint: true }, finiteB = { dummyPoint: true };

let pointAB = Math.sqrt(Math.abs(Math.pow(pointA.x - pointB.x, 2.0) - Math.pow(pointA.y - pointB.y, 2.0)));

if (isNaN(pointAB) === false) {

if (pointAB === 0) {

pointAB = 1;

}

//Calculating Extended line points based on end point of graph

let midPoint = null; //To find mid point of selected two points

let extendPoint = null; //Actual Mid point which used calculating positive / negative points

let checkData = null;//line equation calculation

//Assiging the value to variable if the points not equal

if (pointB.x !== pointA.x && pointA.y !== pointB.y) {

extendPoint = (pointB.y - pointA.y) / (pointB.x - pointA.x);

midPoint = extendPoint;

checkData = pointA.y - (extendPoint \* (pointA.x));

}

// calculating angle is 0 to 90 and 180 to 270

if (midPoint > 0 && pointA.x !== pointB.x && pointA.y !== pointB.y) {

let y = (extendPoint \* (upperRight[0])) + checkData;

if (y <= upperRight[1]) {

finiteA.x = upperRight[0];

finiteA.y = y;

} else {

let x = (upperRight[1] - checkData) / extendPoint;

if (x <= upperRight[0]) {

finiteA.x = x;

finiteA.y = upperRight[1];

}

}

let y1 = (extendPoint \* (bottomLeft[0])) + checkData;

if (y1 >= bottomLeft[1]) {

finiteB.x = bottomLeft[0];

finiteB.y = y1;

} else {

let x1 = (bottomLeft[1] - checkData) / extendPoint;

if (x1 >= bottomLeft[0]) {

finiteB.x = x1;

finiteB.y = bottomLeft[1];

}

}

} else if (midPoint < 0 && pointA.x !== pointB.x && pointA.y !== pointB.y) { // calculating angle is 90 to 180 and 270 to 360

let y = (extendPoint \* (bottomLeft[0])) + checkData;

if (y <= upperRight[1]) {

finiteA.x = bottomLeft[0];

finiteA.y = y;

} else {

let x = (upperRight[1] - checkData) / extendPoint;

if (x >= bottomLeft[0]) {

finiteA.x = x;

finiteA.y = upperRight[1];

}

}

let y1 = (extendPoint \* (upperRight[0])) + checkData;

if (y1 >= bottomLeft[1]) {

finiteB.x = upperRight[0];

finiteB.y = y1;

} else {

let x1 = (bottomLeft[1] - checkData) / extendPoint;

if (x1 <= upperRight[0]) {

finiteB.x = x1;

finiteB.y = bottomLeft[1];

}

}

} else if (pointA.x === pointB.x && pointA.y !== pointB.y) { //if x points are same and y are different this condition will be executed

finiteA.x = pointA.x;

finiteA.y = upperRight[1];

finiteB.x = pointB.x;

finiteB.y = bottomLeft[1];

} else if (pointA.y === pointB.y && pointA.x !== pointB.x) { //if y points are same and x are different this condition will be executed

finiteA.x = upperRight[0];

finiteA.y = pointA.y;

finiteB.x = bottomLeft[0];

finiteB.y = pointB.y;

} else { // if All the points are same

finiteA.x = upperRight[0];

finiteA.y = pointA.y;

finiteB.x = bottomLeft[0];

finiteB.y = pointB.y;

}

points = Object.assign([], [finiteA, pointA, pointB, finiteB]);

}

}

return points;

}

* + - To Handle the Extended Lines Axes Arrowheads rotation

pointRotation: (data) => {

let rotation = 0;

if (showLine && extendedLines && data && data.dataset && data.dataset.data && data.dataset.data.length > 0

&& ((data.dataset.data[data.dataIndex].dummyPoint) || (!data.dataset.data[data.dataIndex].dummyPoint))) {

let line = data.dataset.data;

let index = data.dataIndex;

line = data.dataset.data.filter((d) => d.dummyPoint);

index = data.dataIndex === 0 ? 0 : 1;

if (line && line.length === 2) {

let firstPoints = Object.assign({}, line[index]);

let secondPoint = Object.assign({}, line[index === 0 ? 1 : 0]);

if (firstPoints && Object.keys(firstPoints).length >= 2 && secondPoint && Object.keys(secondPoint).length >= 2) {

if (firstPoints.x !== undefined && secondPoint.x !== undefined && firstPoints.y !== undefined && secondPoint.y !== undefined) {

let switched = false;

if (index === 0 && firstPoints.y >= secondPoint.y) {

switched = true;

} else if (index === 1 && firstPoints.y <= secondPoint.y) {

switched = true;

}

if (switched) {

firstPoints = Object.assign({}, secondPoint);

secondPoint = Object.assign({}, line[index]);

}

let rotation1 = Math.atan2((firstPoints.x - secondPoint.x), (firstPoints.y - secondPoint.y)) \* (180 / Math.PI) + 360;

rotation1 = (rotation1 + 180) % 360;

rotation = parseInt((switched) ? rotation1 : (rotation1 + 180) % 360);

if (index === 0) {

if (lastRotation) {

lastRotation.rotation = rotation;

}

} else {

if (lastRotation.rotation === rotation) {

rotation = parseInt((parseInt(rotation1) !== rotation) ? rotation1 : ((rotation1 - 180) % 360));

}

}

}

}

}

}

return rotation;

}

* + - A component ItemResponse.jsx from / display / response /graph/ItemResponse.jsx will be used here to handle the graph operations and configurations.

import ItemResponse from './ItemResponse;

// Graph will be called by passing with all the required parameters.

< ItemResponse

item={item}

stemComponent={null}

config={config}

showCorrectResponse={true}

graphNative = {false}

/>

* + - Inside the Graph response the Drag functionality for the plotted points will be enabled using the 'chartjs-plugin-dragdata' plugin.

//Adding chartDragData plugin while creating the chart and enabling the Drag Functionality.

Importing the point draggable plugin for chart js

import ChartDragData from 'chartjs-plugin-dragdata';

chart = new Chart(ctx.current, { type: 'scatter', data: chartData, options: scatterOptions, plugins: [ChartDataLabels, ChartDragData] });

* + - This same ItemResponse component will be called from SingleLinePreview (Preview) component.
    - The readable response for the correct response will be done with following structure.

"readable\_response": "\"\"Ordered Pair 1\n\n\"\"\"X intercept\n\n\""

* + - The readable response will be generated using following method

const generateReadableResponse = (itemData) => {

let readableResponse = '';

let responseValue = [];

let responseValue1 = [];

for (let i = 0; i < itemData.correctResponse.numberOfResponses; i++) {

itemData.correctResponse.responses.forEach((res) => {

if (res && res.point && res.point[0]) {

responseValue.push(

res.point[0].x + "," + res.point[0].y

);

}

if (res && res.point && res.point[1]) {

responseValue1.push(res.point[1].x + "," + res.point[1].y)

}

});

let singlePoint = responseValue.join()

let singlePoint1 = responseValue1.join()

let responseRow = singlePoint;

let responseRow1 = singlePoint1;

if (i === 0) {

readableResponse = `"${responseRow}"Ordered Pair 1\n\n""${responseRow1}"X intercept\n\n"`;

}

}

return readableResponse;

}

**Math Logic for calculating the upper and lower points in line items**

* + - To Find the slope

m = ((y2-y1)/(x2-x1));

* + - Find Y-Intercept

c = (y1 - mx1);

Line equation - y=mx+c;

* + - Calculate Upper point - ex upper right is (10,20)
      * First calculate upper point with x value

yu = m\*10 + c;

* + - * If yu is less than or equal to 20, then upper point is (10,yu), else calculate upper point with y value

xu = (20-c)/m;

* + - * If xu is less than or equal to 10, then upper point is (xu, 20)
    - Calculate Lower point - ex lower left is (-10,-20)
      * First calculate lower point with x value

yl = m\*-10 + c;

* + - * If yl is greater than or equal to -20, then lower point is (-10,yl), else calculate lower point with y value

xl = (-20-c)/m;

* + - * If xl is greater than or equal to -10, then lower point is (xl, -20)
    - Finally, once you get the upper point and lower point - you can place an arrow at that location.
    1. Preview
    - New folder ‘singleline’ will be created in components/display/item for the Single Line item preview.
    - SingleLinePreview.jsx is the component responsible to rendering the item preview.
    - ItemResponses.jsx component will be called from the SingleLinePreview.
    - While calling the ItemResponses will be passing the stem content with help of StemFormatter.

import SingleLineResponses from '../../response/singleline/SingleLineResponses;

<div className='row'>

{itemJson?.stemContent ? itemJson.stemContent: null}

</div>

// ItemResponses will be called by passing with all the required parameters.

<ItemResponses

item = { item }

config ={ config }

onUpdate = { handleResponse }

showCorrectResponse = { showCorrectResponse }

/>

* Then the SingleLinePreview component is imported and mapped with unique item code in DisplayInterface.jsx file which will be in components/display folder.

import SingleLinePreview from './item/singleline/ SingleLinePreview’;

const componentManifest = Object.freeze({ s1l: < SingleLinePreview

{...payload}

/>

});

* + 1. Stories
* For all Single Line components Create, Response and Preview will have stories to render and check UI.
* A folder will be created inside stories/assets which contain json files with all possible combinations.
* SingleLineNew.stories.jsx will be created to call the create single line component with default UI.
* SingleLine.stories.jsx will be created under the stories/create/alternate to check the create single line component with alternate UI.

import SingleLine from '../../../components/create / singleline/ SingleLine;

const TemplateAlt = (args) =>

< SingleLine {...args} config={ alternateConfig }/>;

* SingleLineDisplay.stories.jsx will be created under the stories/display for Single Line preview check with default UI.
* Inside the SingleLineDisplay.stories will be using all the json that we have created under the asset which help us to check the preview in all the combinations.
* SingleLinePreview.stories.jsx will be created under the stories/display/alternate to check the create Single Line preview component with alternate UI.

import SingleLinePreview from '../../../components /display/item/singleline/ SingleLinePreview;

const AlternateTemplate = (args) => (

<SingleLinePreview {...args} config={alternateConfig}

/> );

* For the Single Line response will have a separate story SingleLineResponse.stories.jsx under stories/response.
* For accessibility, we should add proper id for the label and input and other required attributes to support the key accessibility for both content and preview.
  1. Software Updates Needed to Align with Project Standards
* To handle draggable functionality, we can go for react-draggable with latest version as 4.4.5.

React Draggable - <https://www.npmjs.com/package/react-draggable/v/4.4.5>

* 1. How to consume
     + - This kite-react-items component will be used in the cp-prototype application and CreateInterface and PreviewInterface will be called.

<CreateInterface

item = {item}

onUpdate = {onUpdate}

config = {config}

/>

<PreviewInterface

Item = {item}

onUpdate = {onUpdate}

config = {config}

/>

* + - * In the kite-react-items, Single Line Response component will be used only in Preview components.
      * In the Display component it will be called StemFormatter like below.

<StemFormatter

stemContent = { item?.item\_json?.stemContent }

/>

< SingleLineResponse

item={item}

config={config}

onUpdate={handleResponse}

/>

* 1. Click History
* Single Line click history structure below:

"responses": [

{

"responseoption": [ {

"line": [

{"x": 12, "y": 1},

{"x": -2.3, "y": -2.8}

]

} ],

"ts": "Thu Feb 25 2021 09:05:26 GMT-0600 (Central Standard Time)",

"score": 0,

"referencescore": null,

"isCorrectResponse": false

}

]

1. Test Scenarios

**Content**

* + The User will be able to select Single Line item and open the content accordion.
  + Users will be able to save the Item by giving the required fields. The user will be able to edit the Item after saving it in Draft/Final/Approved/Complete.
  + Users will be able to save the item in Draft status without completing all required fields. Once saved, those fields can be edited and saved again.
  + The User will be able to set Item dimensions for the item in the item dimensions field.
  + The User will be able enter content in stem.
  + Users can edit the Graph Settings section consisting of graph title, x-axis title, y-axis title and x-axis label, y-axis label that will reflect on correct response.
  + Default Set Max Lines value will be 1 and the user cannot modify this value.
  + Users can change the x-axis scale, y-axis scale fields for changing the graph scale.
  + Users can change the incremental label, bottom-left, upper-right fields, snap grid for changing the graph properties in the correct response.
  + While changing the incremental label as ALL user can see all the labels of axis line in the graph. If it is NONE –no labels will be displayed and for First & Last Starting and ending label of the axis line.
  + User can change the default values of Bottom Left and Upper Right, based on this axis line will be drawn in the graph.
  + If the user enables snap to grid option, the decimal value of plotted point will get round off based on the scale value.
  + Users can adjust the thickness of the axis line using the Axis Line Size field.
  + Users can adjust the thickness of the axis label using the Axis Label Size field.
  + Users can adjust the dimensions of graph by providing values in width and height fields in graph control section.
  + Users can check the Axes Arrowheads to show axis line arrows in graph.
  + Users can check the show grid to show the grid lines in the graph. If the show grid is unchecked, axes and increment labels will still appear.
  + Users can check the extended line to extend the line till the edges of the graph and arrow heads will be shown at the end of the line.
  + Users can add an image over the graph by clicking on the select image button.
  + Users can drag the image across the graph area.
  + If Ordered Pair1 and Ordered Pair2 are enabled then user can give value in Ordered Pair1, Ordered Pair2 and tolerance field. Also, X Intercept, Y Intercept and Slope will be in disabled mode.
  + If Ordered Pair1 and X Intercept are enabled then user can give value in Ordered Pair1, X Intercept and tolerance field. Also, Y Intercept, Ordered Pair2 and Slope will be in disabled mode.
  + If Ordered Pair1 and Y Intercept are enabled then user can give value in Ordered Pair1, Y Intercept and tolerance field. Also, X Intercept, Ordered Pair2 and Slope will be in disabled mode.
  + If Ordered Pair1 and Slope are enabled then user can give value in Ordered Pair1, Slope and tolerance field. Also, X Intercept, Y Intercept, and Ordered Pair2 will be in disabled mode.
  + If Ordered Pair2 and X Intercept are enabled then the user can give value in Ordered Pair2, X Intercept and tolerance field. Also, Y Intercept, Ordered Pair1 and Slope will be in disabled mode.
  + If Ordered Pair2 and Y Intercept are enabled then user can give value in Ordered Pair2, Y Intercept and tolerance field. Also, X Intercept, Ordered Pair1 and Slope will be in disabled mode.
  + If Ordered Pair2 and Slope is enabled then user can give value in Ordered Pair2, Slope and tolerance field. Also, X Intercept, Y Intercept, and Ordered Pair1 will be in disabled mode.
  + User will be able to adjust the plotted points in graph by changing x, y in correct response or drag the points in the graph.
  + Users can remove the plotted points by clicking over it.
  + User needs to choose any two-scoring selection option to draw a line in the graph.
  + User can add the Tolerance value for each point plotted in the graph.

**Preview**

* + The User will be able to open the preview, by clicking on the preview button in common header.
  + The Stem content that we created should be shown in CP (Content Portal) Preview.
  + The empty graph will be populated in the preview.
  + When the user doesn’t enable the display correct response checkbox, the User will be able to plot the points and draw the line.
  + If display correct response is enabled, and the image is added in content page for the graph, the same reflect in preview.
  + If display correct response is enabled, then correct response will be set in the Preview what we have set in content page and user will not be able do any action on it.

**Scoring**

* Users can set the scores in scoring accordion.
* The scoring method for Single line item are No scoring needed and correct only.
* Users will be able to choose the scoring method from the drop-down menu.
* When a user selects no scoring needed as scoring methods, no additional fields will be displayed.
* When the user selects correct only as scoring methods, the correct response table will be displayed.

**Attribute and Metadata**

* Users will be able to open the attribute and metadata accordion and will be able to select Item metadata, content codes, cognitive settings, tags, and item details.
* These are mandatory fields and should be filled in before saving the Item in a Final /Approved state.

1. Final Effort Estimate

|  |  |
| --- | --- |
| **Tasks** | **Effort in Hours** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **Grand Total** |  |

1. Questions

Questions about Line Items Scoring Selections – Expected Functionality

1. When Ordered Pairs are used to set up the correct response, does the order the points are selected matter for scoring the item correctly? For example, in Editor they’re called “Starting” and “Ending” points and you must select the starting point first in order to get the item correct.
2. When Slope and either x or y intercept are selected for the Scoring Selection, you enter the intercept and the slope value in the Content Accordion. The Correct Response graph shows the intercept and then tries to plot another point at the intercept on the other axis. These two points are what also shows up when you display the Correct Response in the Preview. In the Scoring Accordion, the Slope appears as the coordinates for that second point.

a. Is the slope showing up as a coordinate the expected functionality?

b. An example I ran into when recreating an item we have in CB that seemed odd was a line with y intercept= 8 and slope=3. In the Scoring Accordion, the slope was displaying as the coordinates for the x intercept for the line (-2.666………, it never showed the “, 0”). This seems odd because the graph is set to Snap to Grid so that point of -2.666 isn’t selectable for test taker.

1. If the slope is supposed to display as a coordinate point in the Scoring Accordion, will the item still score any line created with the same slope and intercept as correct?
2. When you have a y intercept and a slope of 0, it appears in the Scoring Accordion as “-Infinity, 0” –is that the expected functionality?
3. In CP, for Scoring Selection, when you pick “slope” you’re also able to pick either ordered pair 1 or ordered pair 2. If the ordered pairs work like they do in Editor today (as the point that needs to be selected first and the point that needs to be selected second), will they work to set up with Slope?
4. We do not have any items set up in Editor/CB with Line Relationships so we did not have anything to compare against when testing. You are able to set up a key with Line Relationships selected, but it doesn’t appear to function any differently if you have Line Relationships selected or not.