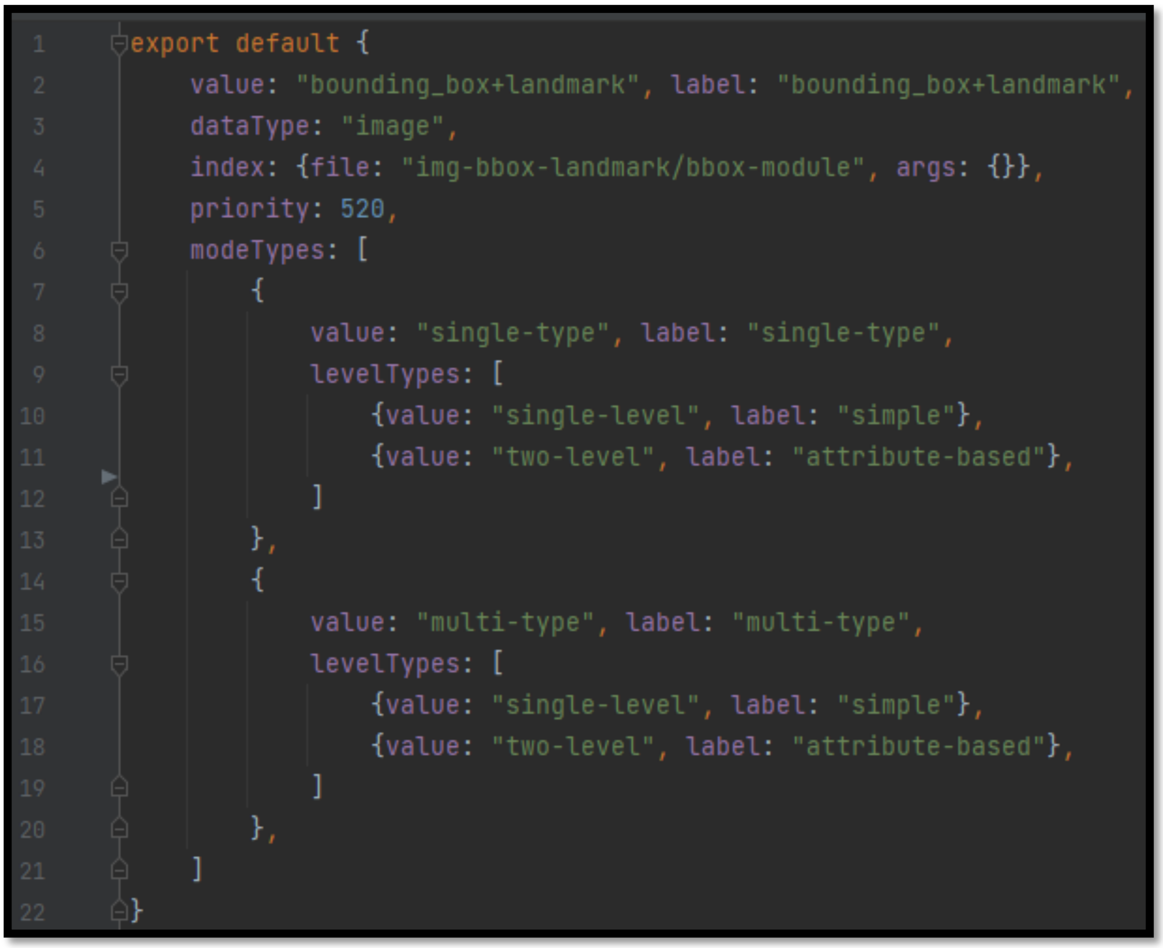
# tika-work New Module Tutorial

# image bounding box + landmarks

This tutorial is for writing a tika-work module that has both image bounding box and landmarks functionalities.

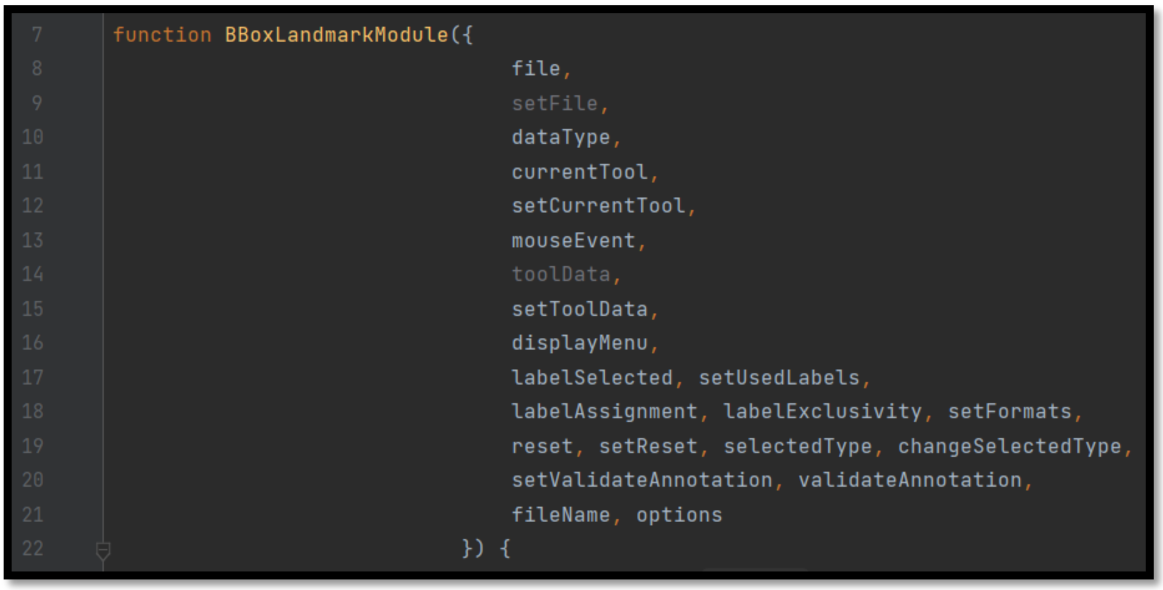
## A. Basic File System

- Create the folder img-bbox-landmark inside the modules folder of the project.
- Add config.js file to the module folder. In this case, it will be single and multi-type with two options by each, simple (single-level) and attribute-based (two-level)



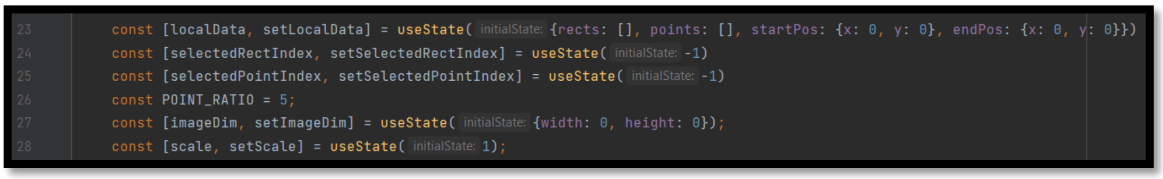
```
export default {
    value: "bounding_box+landmark", label: "bounding_box+landmark",
    dataType: "image",
    index: {file: "img-bbox-landmark/bbox-module", args: {}},
    priority: 520,
    modeTypes: [
        {
            value: "single-type", label: "single-type",
            levelTypes: [
                {value: "single-level", label: "simple"},
                {value: "two-level", label: "attribute-based"},
            ]
        },
        {
            value: "multi-type", label: "multi-type",
            levelTypes: [
                {value: "single-level", label: "simple"},
                {value: "two-level", label: "attribute-based"},
            ]
        },
    ]
}
```

- Add the main JS file with the React Component BBoxLandmarkModule and use the params that will be passed by the main code when the module is loaded.



```
function BBoxLandmarkModule({
                                file,
                                setFile,
                                dataType,
                                currentTool,
                                setCurrentTool,
                                mouseEvent,
                                toolData,
                                setToolData,
                                displayMenu,
                                labelSelected, setUsedLabels,
                                labelAssignment, labelExclusivity, setFormats,
                                reset, setReset, selectedType, changeSelectedType,
                                setValidateAnnotation, validateAnnotation,
                                fileName, options
                            }) {
```

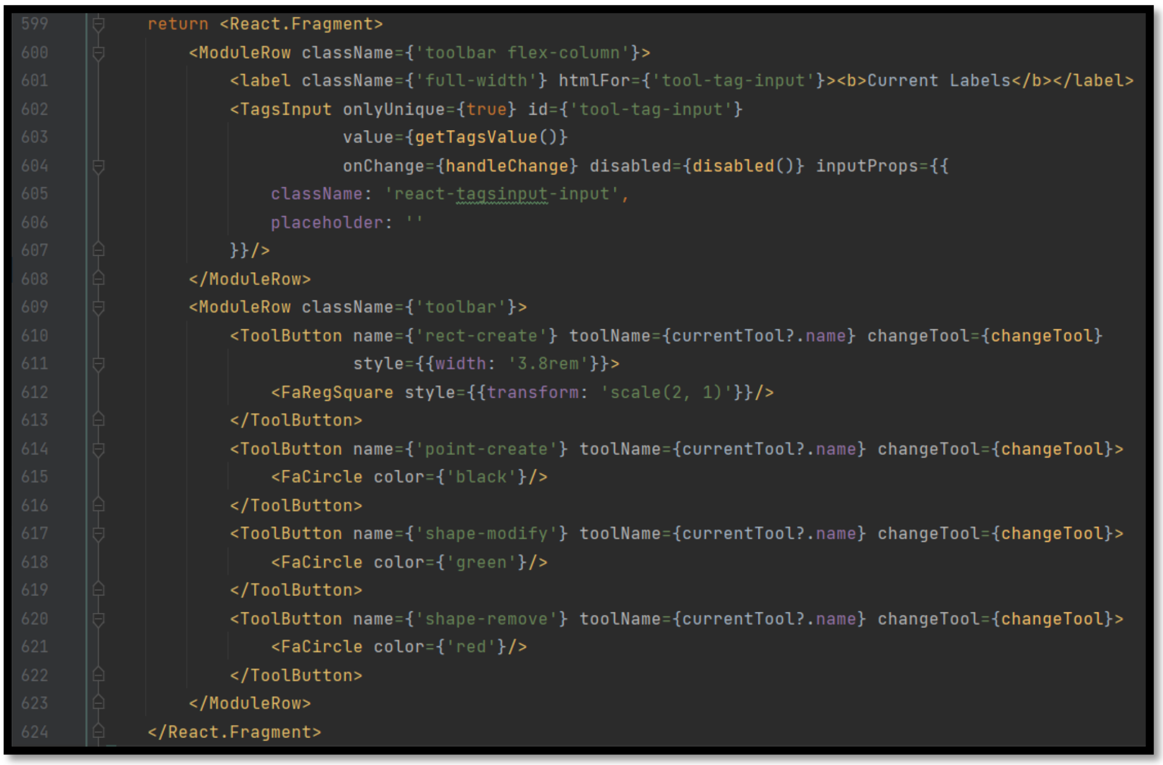
- Use custom variables to set the data state. LocalData for shape objects and temporal positions, selectRectIndex and selectPointIndex to set index of current selected shape, PointRatio is a constant for ratio to draw circles, imageDim to width and height of loaded image, scale to store relation between image and view container



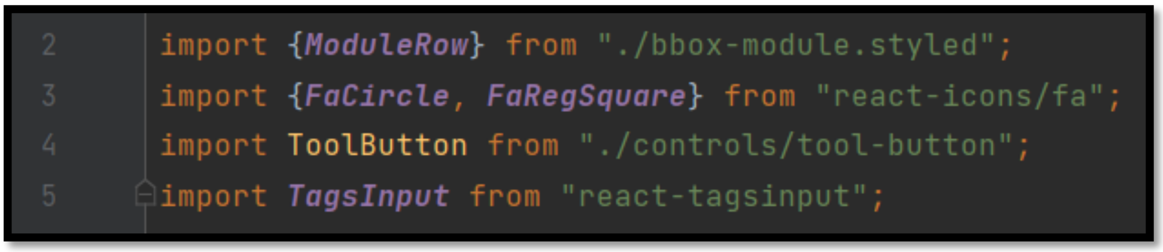
```
    const [localData, setLocalData] = useState( initialState: {rects: [], points: [], startPos: {x: 0, y: 0}, endPos: {x: 0, y: 0}})
    const [selectedRectIndex, setSelectedRectIndex] = useState( initialState: -1)
    const [selectedPointIndex, setSelectedPointIndex] = useState( initialState: -1)
    const POINT_RATIO = 5;
    const [imageDim, setImageDim] = useState( initialState: {width: 0, height: 0});
    const [scale, setScale] = useState( initialState: 1);
```

## B. React Html components

- Return React Html to show module custom controls for user interface. You can use external, internal or styled components. It has two toolbars, first with the tags input control and second with tool buttons for create, modify and remove shapes.

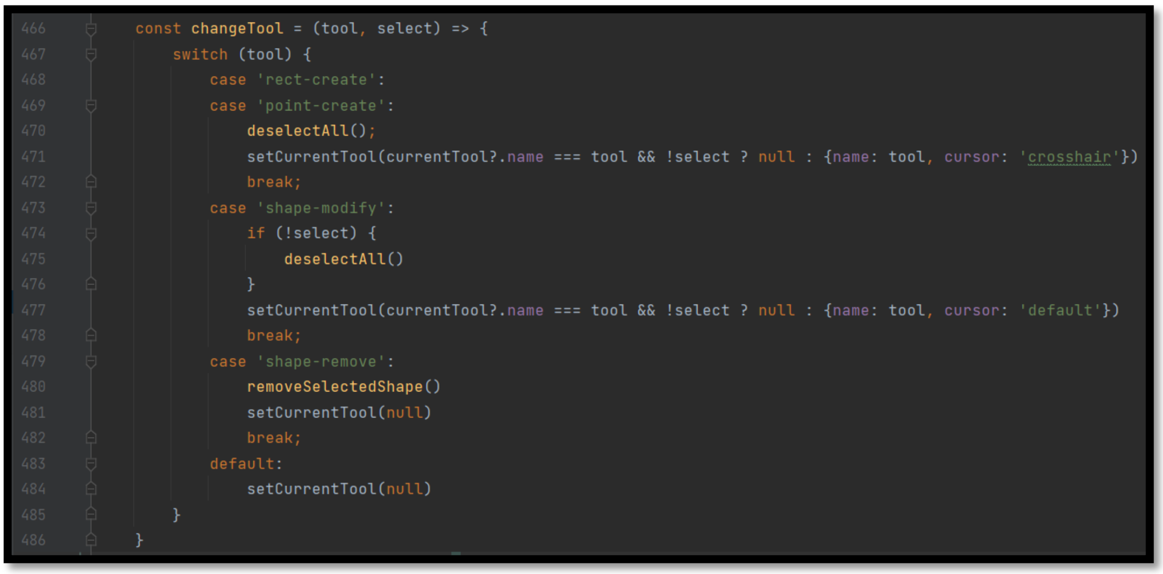


```jsx
return <React.Fragment>
    <ModuleRow className={'toolbar flex-column'}>
        <label className={'full-width'} htmlFor={'tool-tag-input'}><b>Current Labels</b></label>
        <TagsInput onlyUnique={true} id={'tool-tag-input'}
                   value={getTagsValue()}
                   onChange={handleChange} disabled={disabled()} inputProps={{
            className: 'react-tagsinput-input',
            placeholder: ''
        }}/>
    </ModuleRow>
    <ModuleRow className={'toolbar'}>
        <ToolButton name={'rect-create'} toolName={currentTool?.name} changeTool={changeTool}
                    style={{width: '3.8rem'}}>
            <FaRegSquare style={{transform: 'scale(2, 1)'}}/>
        </ToolButton>
        <ToolButton name={'point-create'} toolName={currentTool?.name} changeTool={changeTool}>
            <FaCircle color={'black'}/>
        </ToolButton>
        <ToolButton name={'shape-modify'} toolName={currentTool?.name} changeTool={changeTool}>
            <FaCircle color={'green'}/>
        </ToolButton>
        <ToolButton name={'shape-remove'} toolName={currentTool?.name} changeTool={changeTool}>
            <FaCircle color={'red'}/>
        </ToolButton>
    </ModuleRow>
</React.Fragment>
```



```jsx
import {ModuleRow} from "./bbox-module.styled";
import {FaCircle, FaRegSquare} from "react-icons/fa";
import ToolButton from "./controls/tool-button";
import TagsInput from "react-tagsinput";
```

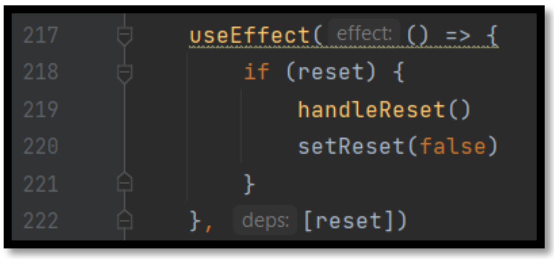
- These controls allow you to change the internal state and also send the viewer the current tool.



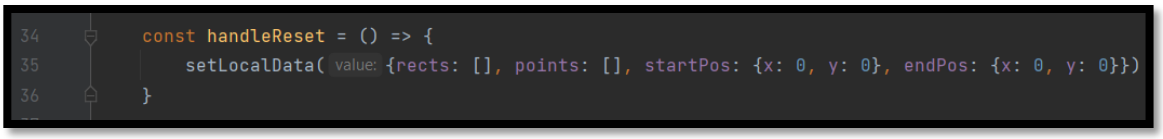
```jsx
const changeTool = (tool, select) => {
    switch (tool) {
        case 'rect-create':
        case 'point-create':
            deselectAll();
            setCurrentTool(currentTool?.name === tool && !select ? null : {name: tool, cursor: 'crosshair'})
            break;
        case 'shape-modify':
            if (!select) {
                deselectAll()
            }
            setCurrentTool(currentTool?.name === tool && !select ? null : {name: tool, cursor: 'default'})
            break;
        case 'shape-remove':
            removeSelectedShape()
            setCurrentTool(null)
            break;
        default:
            setCurrentTool(null)
    }
}
```

## C. Events from platform

- On the other hand, it is necessary to keep the module listening for external state changes to perform the indicated actions or respond to events. E.g. when change reset option set local data to initial state

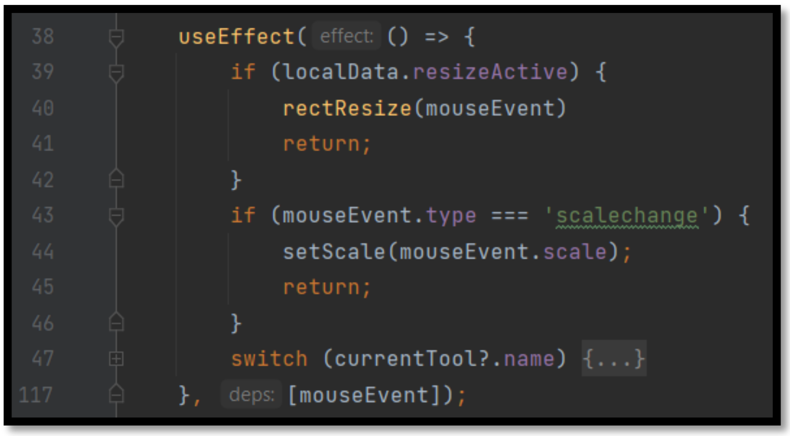


```
useEffect( effect: () => {
    if (reset) {
        handleReset()
        setReset(false)
    }
}, deps: [reset])
```



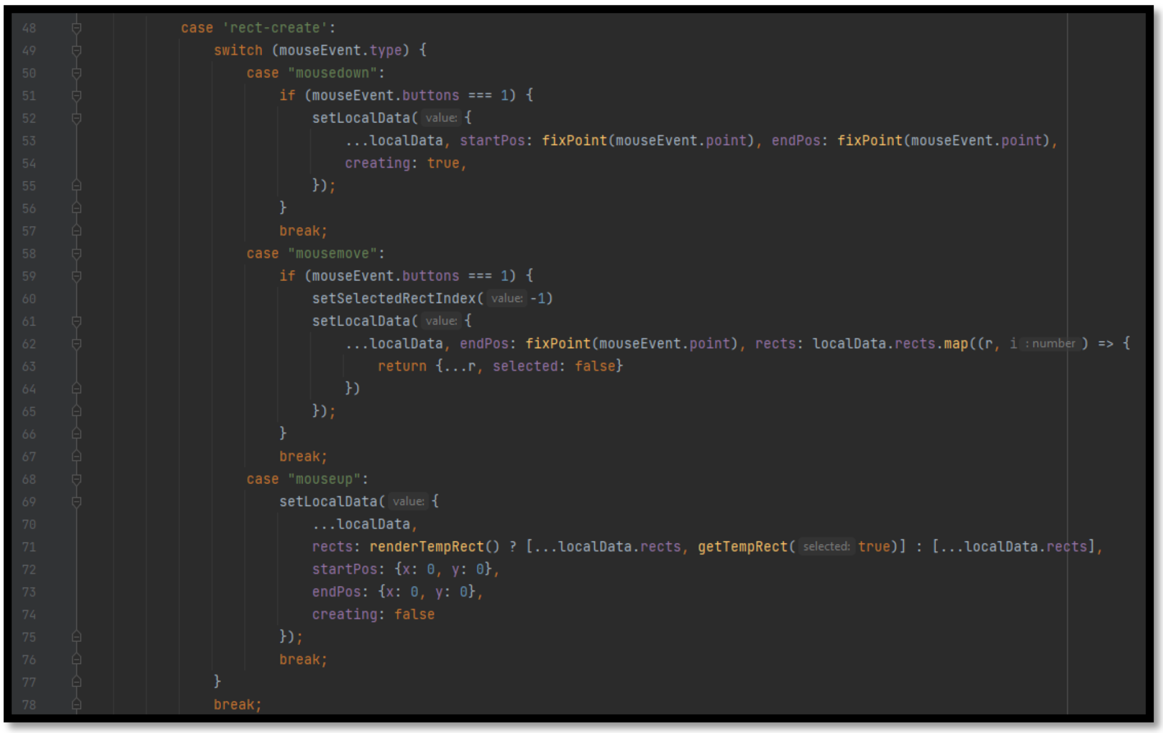
```
const handleReset = () => {
    setLocalData( value: {rects: [], points: [], startPos: {x: 0, y: 0}, endPos: {x: 0, y: 0}})
}
```

- React to each event depending on the tool that is currently active. Redirect the event or update state for general cases and respond to specifics:



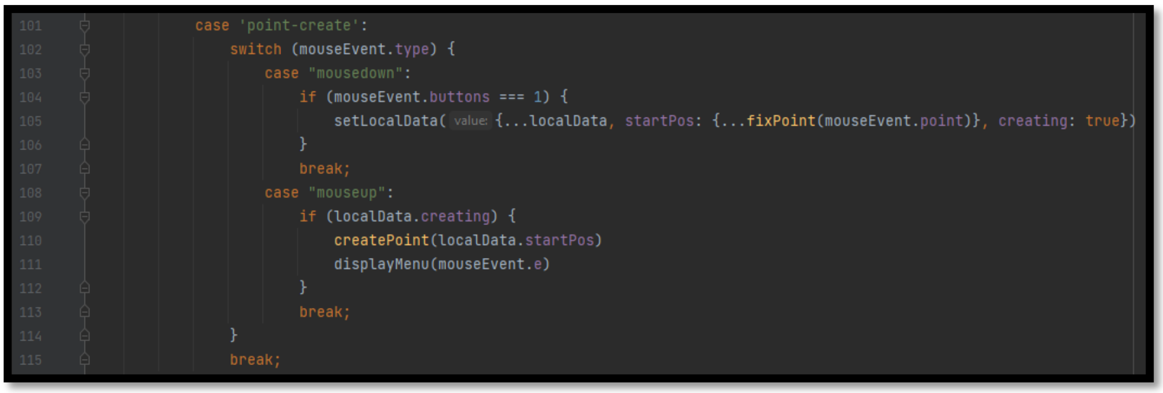
```
useEffect( effect: () => {
    if (localData.resizeActive) {
        rectResize(mouseEvent)
        return;
    }
    if (mouseEvent.type === 'scalechange') {
        setScale(mouseEvent.scale);
        return;
    }
    switch (currentTool?.name) {...}
}, deps: [mouseEvent]);
```

   - Mouse events when is creating a rectangle. With rect-create tool, it use mousedown, and mouseup to create a new rectangle, and mousemove to unselected and create a new one.



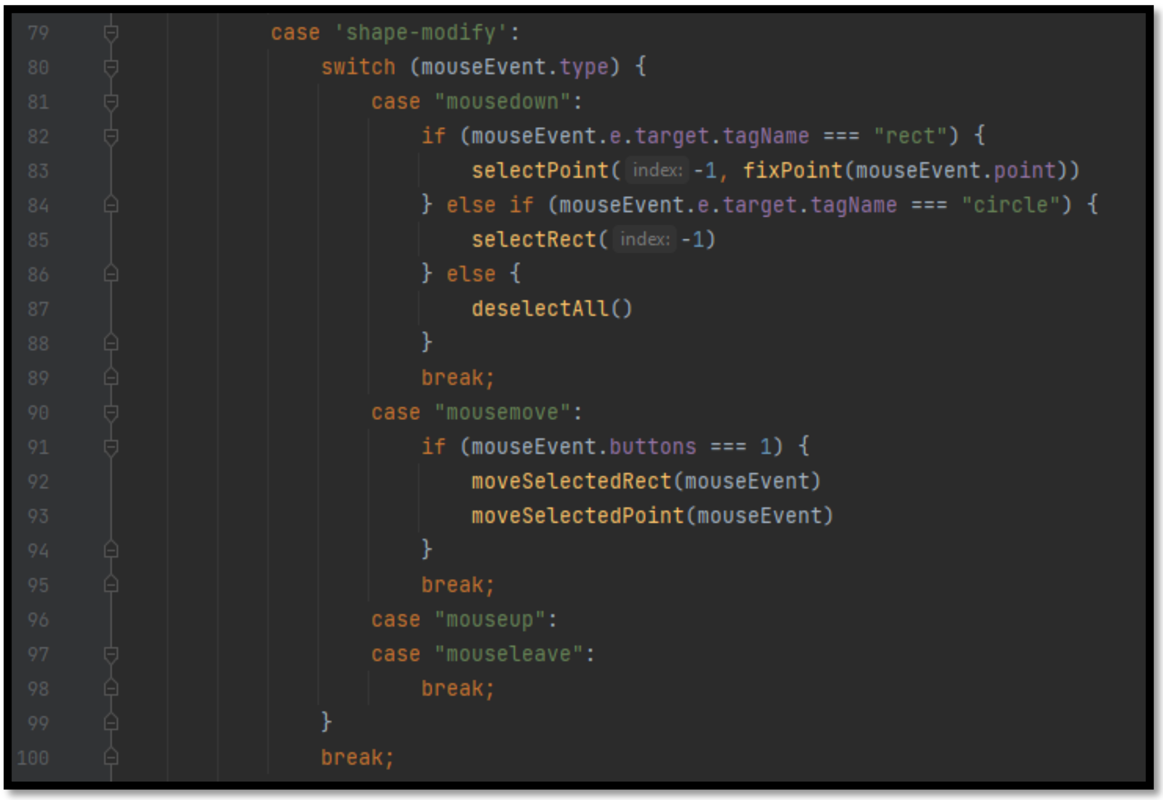
```
case 'rect-create':
    switch (mouseEvent.type) {
        case "mousedown":
            if (mouseEvent.buttons === 1) {
                setLocalData( value: {
                    ...localData, startPos: fixPoint(mouseEvent.point), endPos: fixPoint(mouseEvent.point),
                    creating: true,
                });
            }
            break;
        case "mousemove":
            if (mouseEvent.buttons === 1) {
                setSelectedRectIndex( value: -1)
                setLocalData( value: {
                    ...localData, endPos: fixPoint(mouseEvent.point), rects: localData.rects.map((r, i :number ) => {
                        return {...r, selected: false}
                    })
                });
            }
            break;
        case "mouseup":
            setLocalData( value: {
                ...localData,
                rects: renderTempRect() ? [...localData.rects, getTempRect( selected: true)] : [...localData.rects],
                startPos: {x: 0, y: 0},
                endPos: {x: 0, y: 0},
                creating: false
            });
            break;
    }
    break;
```

- Mouse events when is creating a point. With point-create tool, it use mousedown, and mouseup to create a new point, and display context menu with labels.



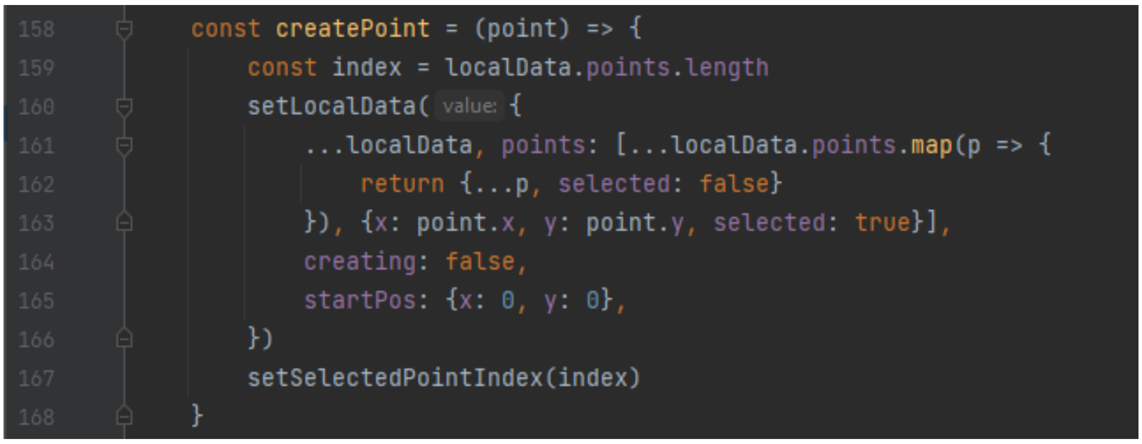
```
case 'point-create':
    switch (mouseEvent.type) {
        case "mousedown":
            if (mouseEvent.buttons === 1) {
                setLocalData( value: {...localData, startPos: {...fixPoint(mouseEvent.point)}, creating: true})
            }
            break;
        case "mouseup":
            if (localData.creating) {
                createPoint(localData.startPos)
                displayMenu(mouseEvent.e)
            }
            break;
    }
    break;
```

- Mouse events when is modifying a shape (rectangle or point). With shape-modify tool, it use mousedown, and mousemove to select the shape and move it.

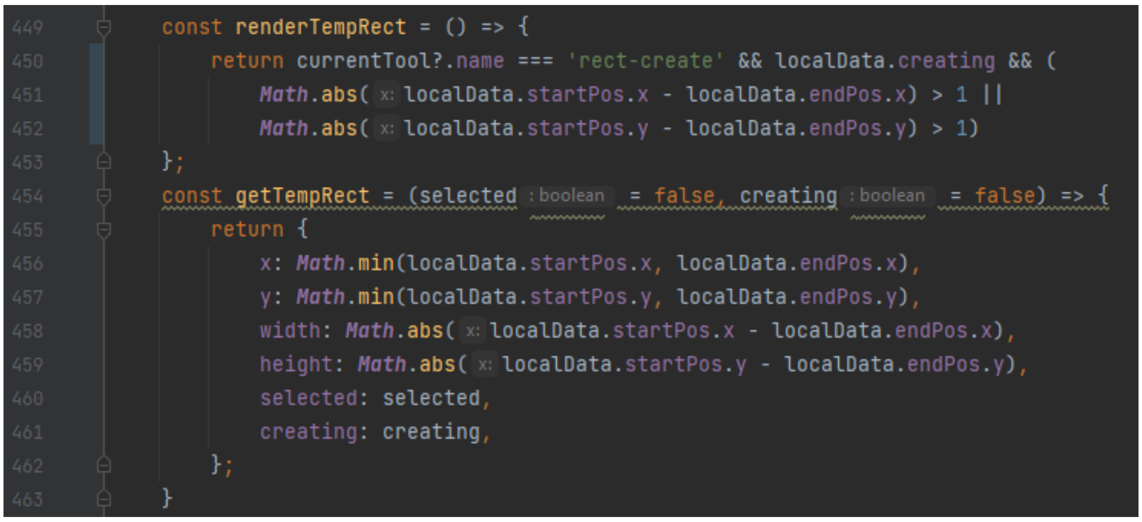


```
            case 'shape-modify':
                switch (mouseEvent.type) {
                    case "mousedown":
                        if (mouseEvent.e.target.tagName === "rect") {
                            selectPoint( index: -1, fixPoint(mouseEvent.point))
                        } else if (mouseEvent.e.target.tagName === "circle") {
                            selectRect( index: -1)
                        } else {
                            deselectAll()
                        }
                        break;
                    case "mousemove":
                        if (mouseEvent.buttons === 1) {
                            moveSelectedRect(mouseEvent)
                            moveSelectedPoint(mouseEvent)
                        }
                        break;
                    case "mouseup":
                    case "mouseleave":
                        break;
                }
                break;
```

- Using these mouse events, the internal state of the module is updated with the new shapes created. The local state is updated adding the new shape as selected on the final of the list with the rest unselected and it reset more basic data.

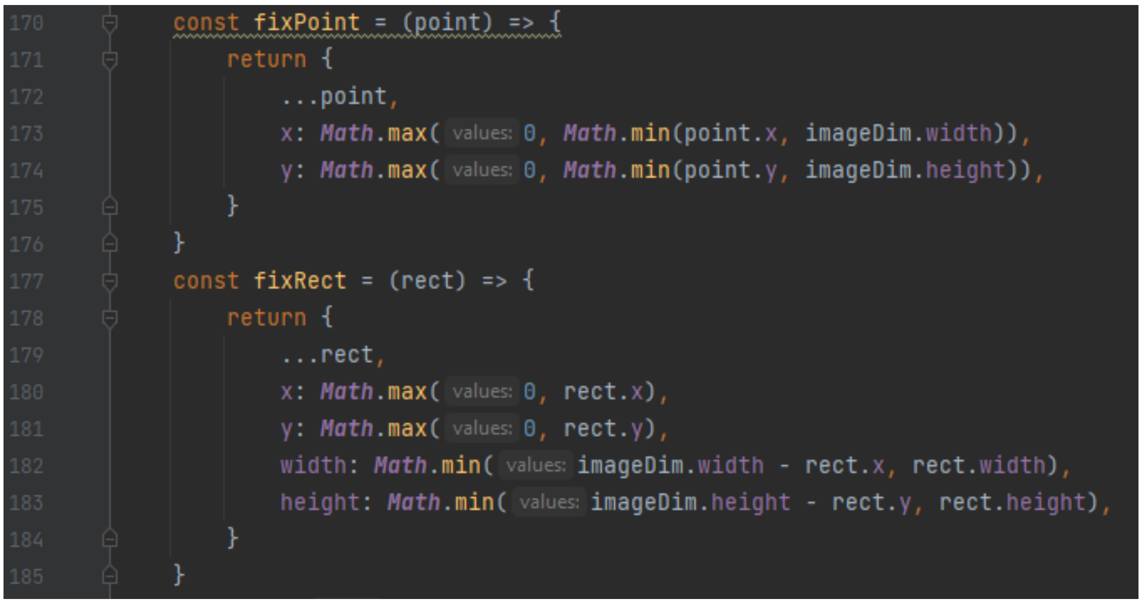


```
const createPoint = (point) => {
    const index = localData.points.length
    setLocalData( value: {
        ...localData, points: [...localData.points.map(p => {
            return {...p, selected: false}
        }), {x: point.x, y: point.y, selected: true}],
        creating: false,
        startPos: {x: 0, y: 0},
    })
    setSelectedPointIndex(index)
}
```



```
const renderTempRect = () => {
    return currentTool?.name === 'rect-create' && localData.creating && (
        Math.abs( x: localData.startPos.x - localData.endPos.x) > 1 ||
        Math.abs( x: localData.startPos.y - localData.endPos.y) > 1)
};
const getTempRect = (selected :boolean  = false, creating :boolean  = false) => {
    return {
        x: Math.min(localData.startPos.x, localData.endPos.x),
        y: Math.min(localData.startPos.y, localData.endPos.y),
        width: Math.abs( x: localData.startPos.x - localData.endPos.x),
        height: Math.abs( x: localData.startPos.y - localData.endPos.y),
        selected: selected,
        creating: creating,
    };
}
```

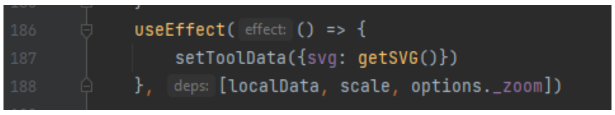
- Auxiliary functions to correct the coordinates at your convenience.



```
const fixPoint = (point) => {
    return {
        ...point,
        x: Math.max( values: 0, Math.min(point.x, imageDim.width)),
        y: Math.max( values: 0, Math.min(point.y, imageDim.height)),
    }
}
const fixRect = (rect) => {
    return {
        ...rect,
        x: Math.max( values: 0, rect.x),
        y: Math.max( values: 0, rect.y),
        width: Math.min( values: imageDim.width - rect.x, rect.width),
        height: Math.min( values: imageDim.height - rect.y, rect.height),
    }
}
```

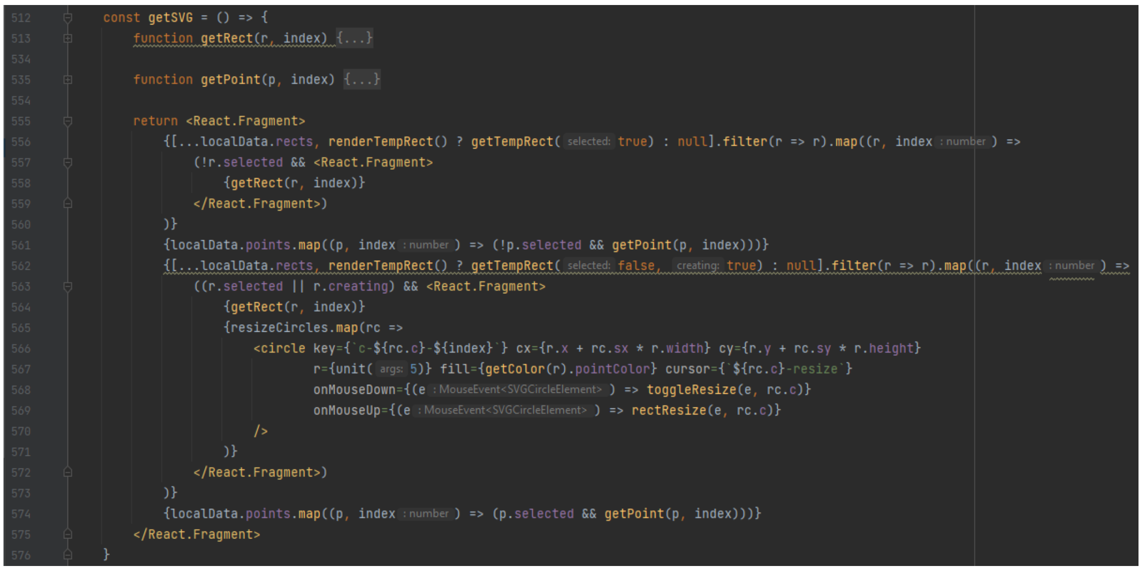
## D. Graphic representation

- In the case of image modules, it is possible to draw on the image by sending an SVG object.



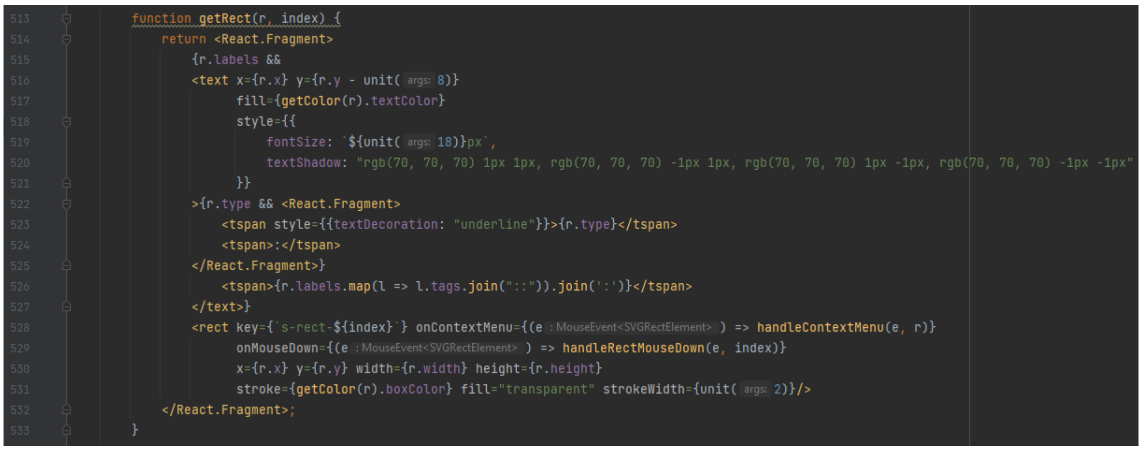
```
useEffect( effect: () => {
    setToolData({svg: getSVG()})
},  deps: [localData, scale, options._zoom])
```

- The shapes are differentiated in the case of being selected, adding extra elements that allow their manipulation and better visualization.

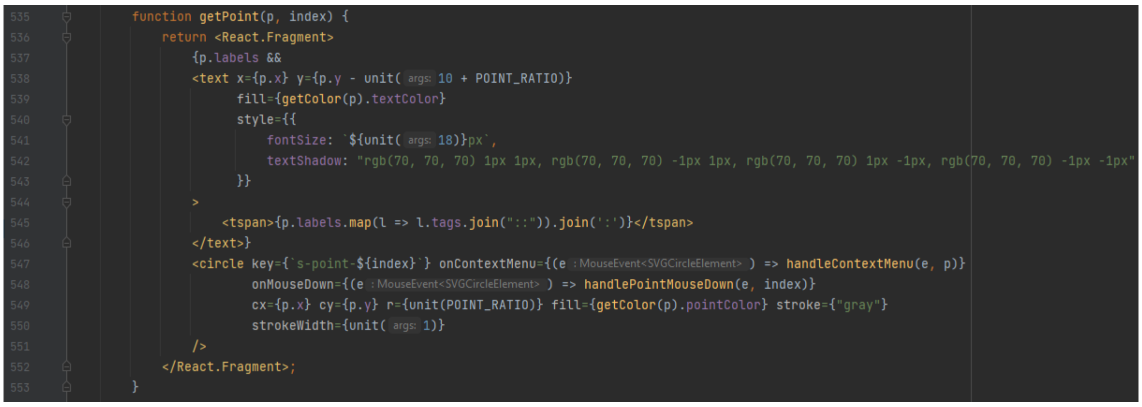


```
const getSVG = () => {
    function getRect(r, index) {...}

    function getPoint(p, index) {...}

    return <React.Fragment>
        {[...localData.rects, renderTempRect() ? getTempRect( selected: true) : null].filter(r => r).map((r, index :number ) =>
            (!r.selected && <React.Fragment>
                {getRect(r, index)}
            </React.Fragment>)
        )}
        {localData.points.map((p, index :number ) => (!p.selected && getPoint(p, index)))}
        {[...localData.rects, renderTempRect() ? getTempRect( selected: false,  creating: true) : null].filter(r => r).map((r, index :number ) =>
            ((r.selected || r.creating) && <React.Fragment>
                {getRect(r, index)}
                {resizeCircles.map(rc =>
                    <circle key={`c-${rc.c}-${index}`} cx={r.x + rc.sx * r.width} cy={r.y + rc.sy * r.height}
                            r={unit( args: 5)} fill={getColor(r).pointColor} cursor={`${rc.c}-resize`}
                            onMouseDown={(e :MouseEvent<SVGCircleElement> ) => toggleResize(e, rc.c)}
                            onMouseUp={(e :MouseEvent<SVGCircleElement> ) => rectResize(e, rc.c)}
                    />
                )}
            </React.Fragment>)
        )}
        {localData.points.map((p, index :number ) => (p.selected && getPoint(p, index)))}
    </React.Fragment>
}
```

- Various effects can be achieved using basic SVG objects and color formats



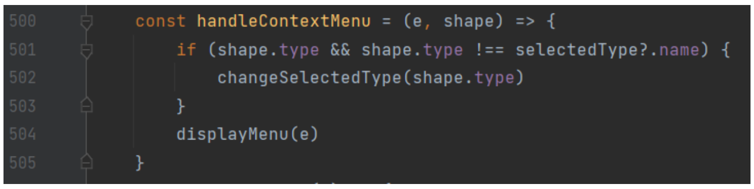
```
    function getRect(r, index) {
        return <React.Fragment>
            {r.labels &&
            <text x={r.x} y={r.y - unit( args: 8)}
                  fill={getColor(r).textColor}
                  style={{
                      fontSize: `${unit( args: 18)}px`,
                      textShadow: "rgb(70, 70, 70) 1px 1px, rgb(70, 70, 70) -1px 1px, rgb(70, 70, 70) 1px -1px, rgb(70, 70, 70) -1px -1px"
                  }}
            >{r.type && <React.Fragment>
                <tspan style={{textDecoration: "underline"}}>{r.type}</tspan>
                <tspan>:</tspan>
            </React.Fragment>}
                <tspan>{r.labels.map(l => l.tags.join("::")).join(':')}</tspan>
            </text>}
            <rect key={`s-rect-${index}`} onContextMenu={(e :MouseEvent<SVGRectElement> ) => handleContextMenu(e, r)}
                  onMouseDown={(e :MouseEvent<SVGRectElement> ) => handleRectMouseDown(e, index)}
                  x={r.x} y={r.y} width={r.width} height={r.height}
                  stroke={getColor(r).boxColor} fill="transparent" strokeWidth={unit( args: 2)}/>
        </React.Fragment>;
    }
```



```
    function getPoint(p, index) {
        return <React.Fragment>
            {p.labels &&
            <text x={p.x} y={p.y - unit( args: 10 + POINT_RATIO)}
                  fill={getColor(p).textColor}
                  style={{
                      fontSize: `${unit( args: 18)}px`,
                      textShadow: "rgb(70, 70, 70) 1px 1px, rgb(70, 70, 70) -1px 1px, rgb(70, 70, 70) 1px -1px, rgb(70, 70, 70) -1px -1px"
                  }}
            >
                <tspan>{p.labels.map(l => l.tags.join("::")).join(':')}</tspan>
            </text>}
            <circle key={`s-point-${index}`} onContextMenu={(e :MouseEvent<SVGCircleElement> ) => handleContextMenu(e, p)}
                    onMouseDown={(e :MouseEvent<SVGCircleElement> ) => handlePointMouseDown(e, index)}
                    cx={p.x} cy={p.y} r={unit(POINT_RATIO)} fill={getColor(p).pointColor} stroke={"gray"}
                    strokeWidth={unit( args: 1)}
            />
        </React.Fragment>;
    }
```

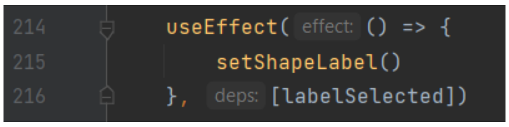
## I. Annotations

- You can show the context menu for choose a label over any shape

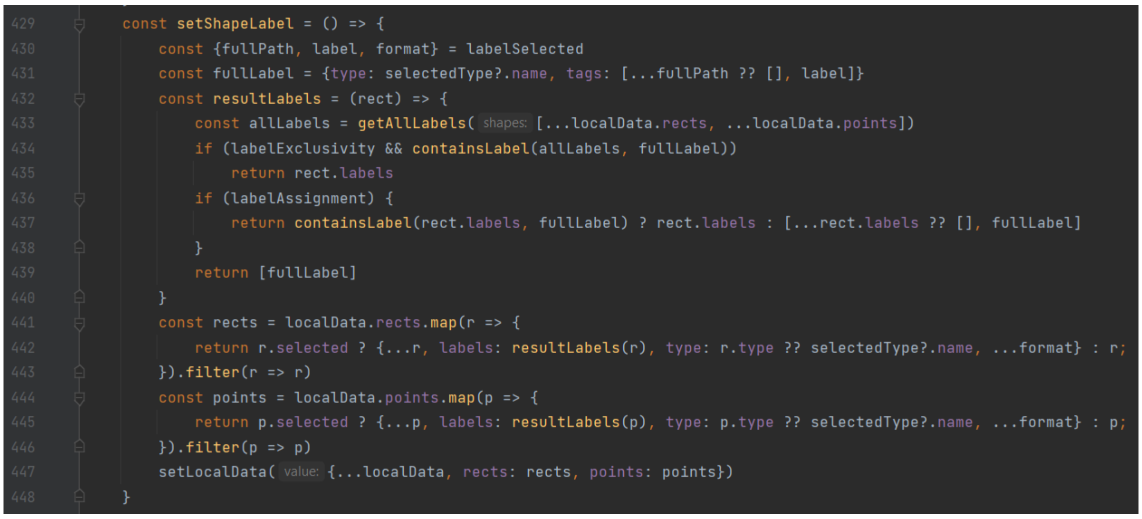


```
const handleContextMenu = (e, shape) => {
    if (shape.type && shape.type !== selectedType?.name) {
        changeSelectedType(shape.type)
    }
    displayMenu(e)
}
```

- When a label is selected on the context menu the variable labelSelected will change the value with the data of the label. You must consult labelExclusivity and labelAssignment to respect the logic of the system and assign the label to the selected shape.

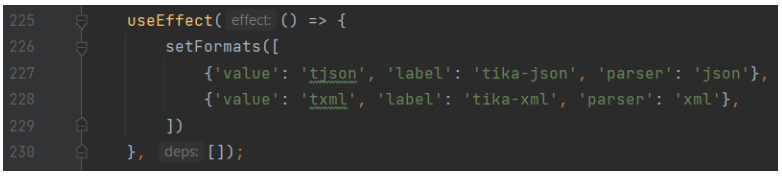


```
useEffect( effect: () => {
    setShapeLabel()
}, deps: [labelSelected])
```



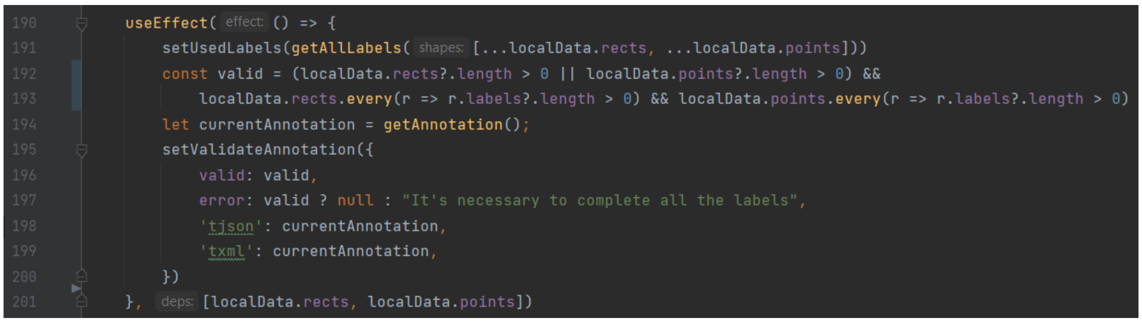
```
const setShapeLabel = () => {
    const {fullPath, label, format} = labelSelected
    const fullLabel = {type: selectedType?.name, tags: [...fullPath ?? [], label]}
    const resultLabels = (rect) => {
        const allLabels = getAllLabels( shapes: [...localData.rects, ...localData.points])
        if (labelExclusivity && containsLabel(allLabels, fullLabel))
            return rect.labels
        if (labelAssignment) {
            return containsLabel(rect.labels, fullLabel) ? rect.labels : [...rect.labels ?? [], fullLabel]
        }
        return [fullLabel]
    }
    const rects = localData.rects.map(r => {
        return r.selected ? {...r, labels: resultLabels(r), type: r.type ?? selectedType?.name, ...format} : r;
    }).filter(r => r)
    const points = localData.points.map(p => {
        return p.selected ? {...p, labels: resultLabels(p), type: p.type ?? selectedType?.name, ...format} : p;
    }).filter(p => p)
    setLocalData( value: {...localData, rects: rects, points: points})
}
```

- The module must specify the formats that it allows to export and declare what type of data corresponds to each one, XML or JSON.



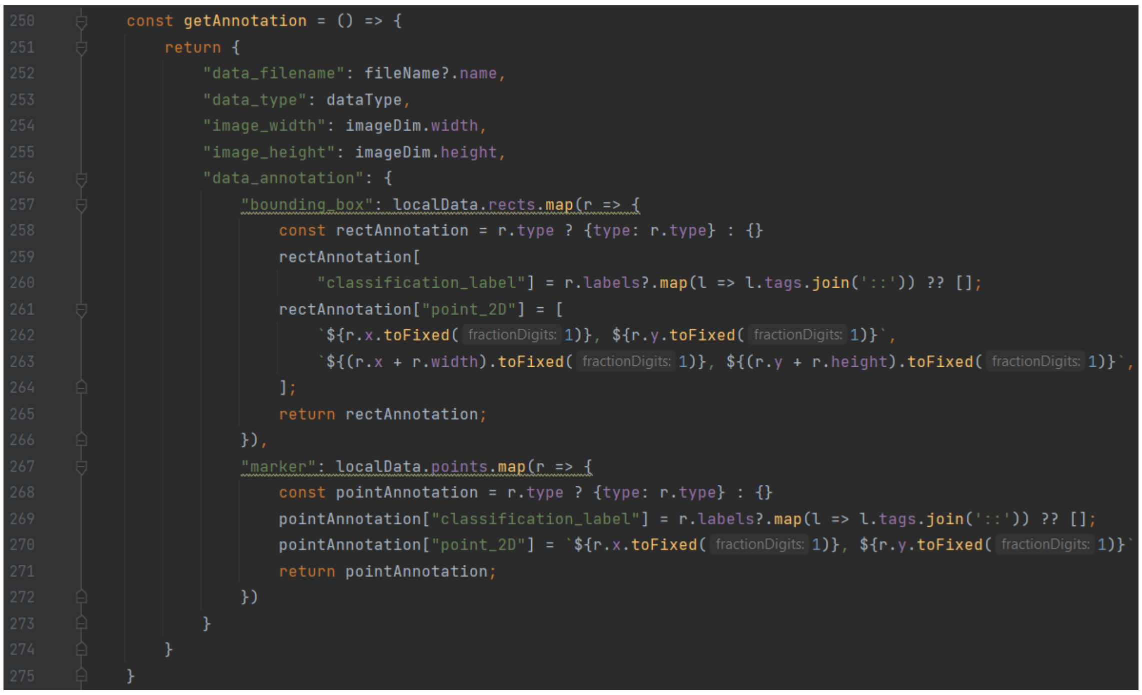
```
useEffect( effect: () => {
    setFormats([
        {'value': 'tjson', 'label': 'tika-json', 'parser': 'json'},
        {'value': 'txml', 'label': 'tika-xml', 'parser': 'xml'},
    ])
}, deps: []);
```

- The module information must be kept updated after each change, indicating if it is valid for export, the data for each format or the existing error.



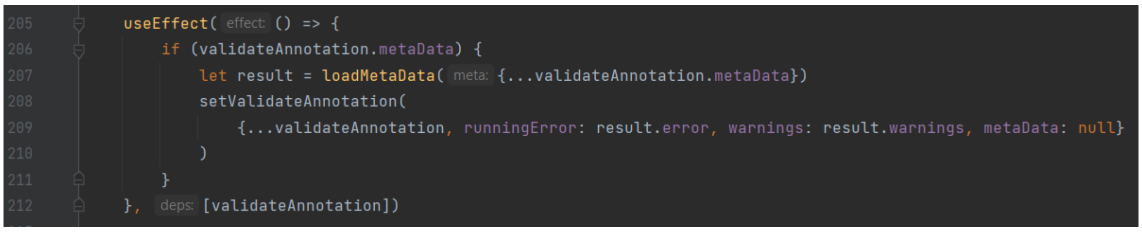
```
useEffect( effect: () => {
    setUsedLabels(getAllLabels( shapes: [...localData.rects, ...localData.points]))
    const valid = (localData.rects?.length > 0 || localData.points?.length > 0) &&
        localData.rects.every(r => r.labels?.length > 0) && localData.points.every(r => r.labels?.length > 0)
    let currentAnnotation = getAnnotation();
    setValidateAnnotation({
        valid: valid,
        error: valid ? null : "It's necessary to complete all the labels",
        'tjson': currentAnnotation,
        'txml': currentAnnotation,
    })
},  deps: [localData.rects, localData.points])
```

1- The data to be exported depends in a general sense on the specific format being exported and the type of data. The most specific data of the annotation are obtained from the lists of shapes stored in the local data and the assigned labels.



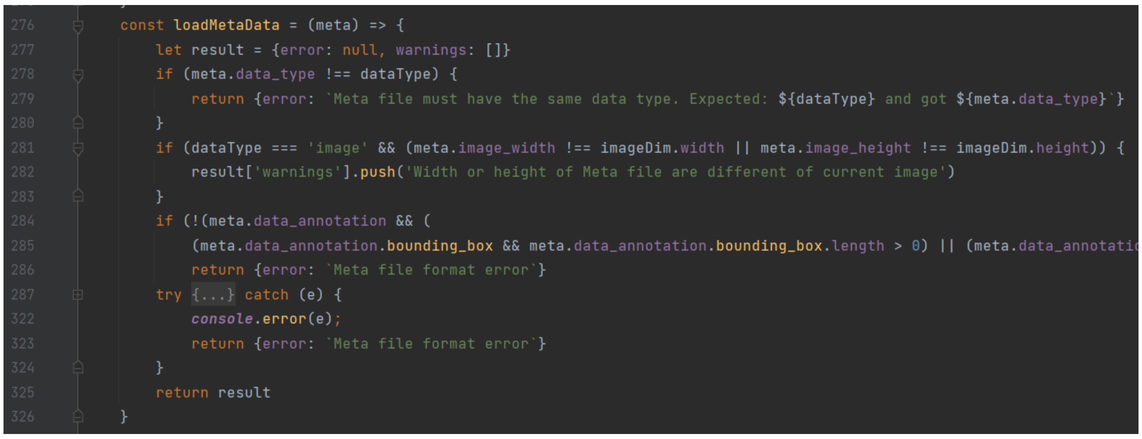
```
const getAnnotation = () => {
    return {
        "data_filename": fileName?.name,
        "data_type": dataType,
        "image_width": imageDim.width,
        "image_height": imageDim.height,
        "data_annotation": {
            "bounding_box": localData.rects.map(r => {
                const rectAnnotation = r.type ? {type: r.type} : {}
                rectAnnotation[
                    "classification_label"] = r.labels?.map(l => l.tags.join('::')) ?? [];
                rectAnnotation["point_2D"] = [
                    `${r.x.toFixed( fractionDigits: 1)}, ${r.y.toFixed( fractionDigits: 1)}`,
                    `${(r.x + r.width).toFixed( fractionDigits: 1)}, ${(r.y + r.height).toFixed( fractionDigits: 1)}`,
                ];
                return rectAnnotation;
            }),
            "marker": localData.points.map(r => {
                const pointAnnotation = r.type ? {type: r.type} : {}
                pointAnnotation["classification_label"] = r.labels?.map(l => l.tags.join('::')) ?? [];
                pointAnnotation["point_2D"] = `${r.x.toFixed( fractionDigits: 1)}, ${r.y.toFixed( fractionDigits: 1)}`
                return pointAnnotation;
            })
        }
    }
}
```

2- If a metadata file has been imported, the change is reported through the variable validateAnnotation.metaData which should be processed and reset to null.



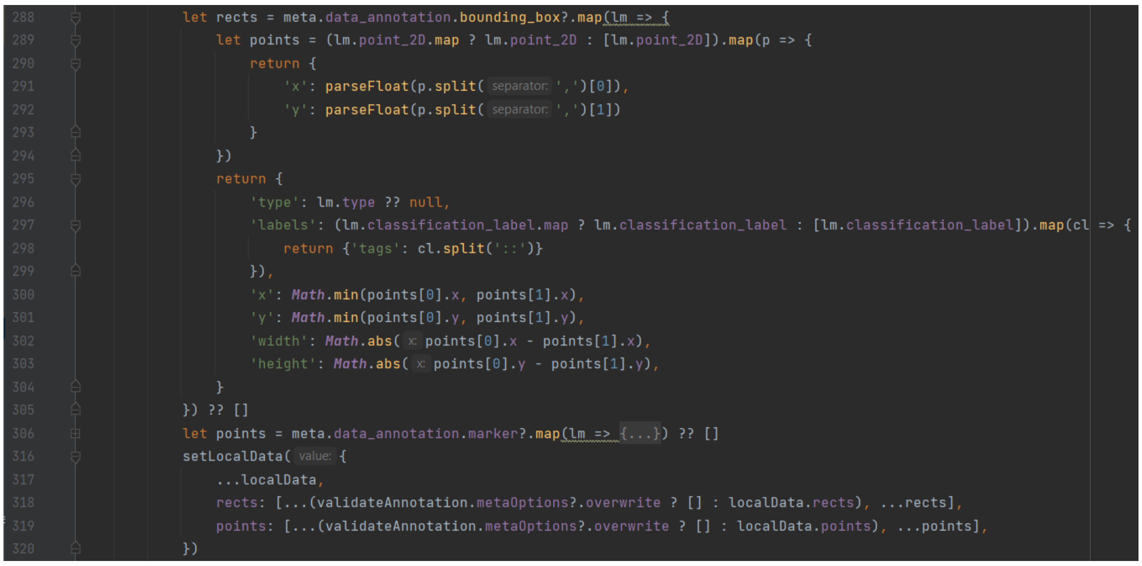
```
useEffect( effect: () => {
    if (validateAnnotation.metaData) {
        let result = loadMetaData( meta: {...validateAnnotation.metaData})
        setValidateAnnotation(
            {...validateAnnotation, runningError: result.error, warnings: result.warnings, metaData: null}
        )
    }
},  deps: [validateAnnotation])
```

3- During the import process it is possible to make the pertinent verifications and return the error or the necessary warnings.



```javascript
const loadMetaData = (meta) => {
    let result = {error: null, warnings: []}
    if (meta.data_type !== dataType) {
        return {error: `Meta file must have the same data type. Expected: ${dataType} and got ${meta.data_type}`}
    }
    if (dataType === 'image' && (meta.image_width !== imageDim.width || meta.image_height !== imageDim.height)) {
        result['warnings'].push('Width or height of Meta file are different of current image')
    }
    if (!(meta.data_annotation && (
        (meta.data_annotation.bounding_box && meta.data_annotation.bounding_box.length > 0) || (meta.data_annotatic
        return {error: `Meta file format error`}
    try {...} catch (e) {
        console.error(e);
        return {error: `Meta file format error`}
    }
    return result
}
```

4- After the verifications are done, the objects must be rebuilt according to the module. Verifying the variable validationAnnotation.metaOptions.overwrite allows knowing the mod to proceed with the existing data in the local data.



```javascript
        let rects = meta.data_annotation.bounding_box?.map(lm => {
            let points = (lm.point_2D.map ? lm.point_2D : [lm.point_2D]).map(p => {
                return {
                    'x': parseFloat(p.split( separator: ',')[0]),
                    'y': parseFloat(p.split( separator: ',')[1])
                }
            })
            return {
                'type': lm.type ?? null,
                'labels': (lm.classification_label.map ? lm.classification_label : [lm.classification_label]).map(cl => {
                    return {'tags': cl.split('::')}
                }),
                'x': Math.min(points[0].x, points[1].x),
                'y': Math.min(points[0].y, points[1].y),
                'width': Math.abs( x: points[0].x - points[1].x),
                'height': Math.abs( x: points[0].y - points[1].y),
            }
        }) ?? []
        let points = meta.data_annotation.marker?.map(lm => {...}) ?? []
        setLocalData( value: {
            ...localData,
            rects: [...(validateAnnotation.metaOptions?.overwrite ? [] : localData.rects), ...rects],
            points: [...(validateAnnotation.metaOptions?.overwrite ? [] : localData.points), ...points],
        })
```

**This is only a variant for the implementation, the platform provides freedoms for the design, the objects and events that want to be represented by the SVG object