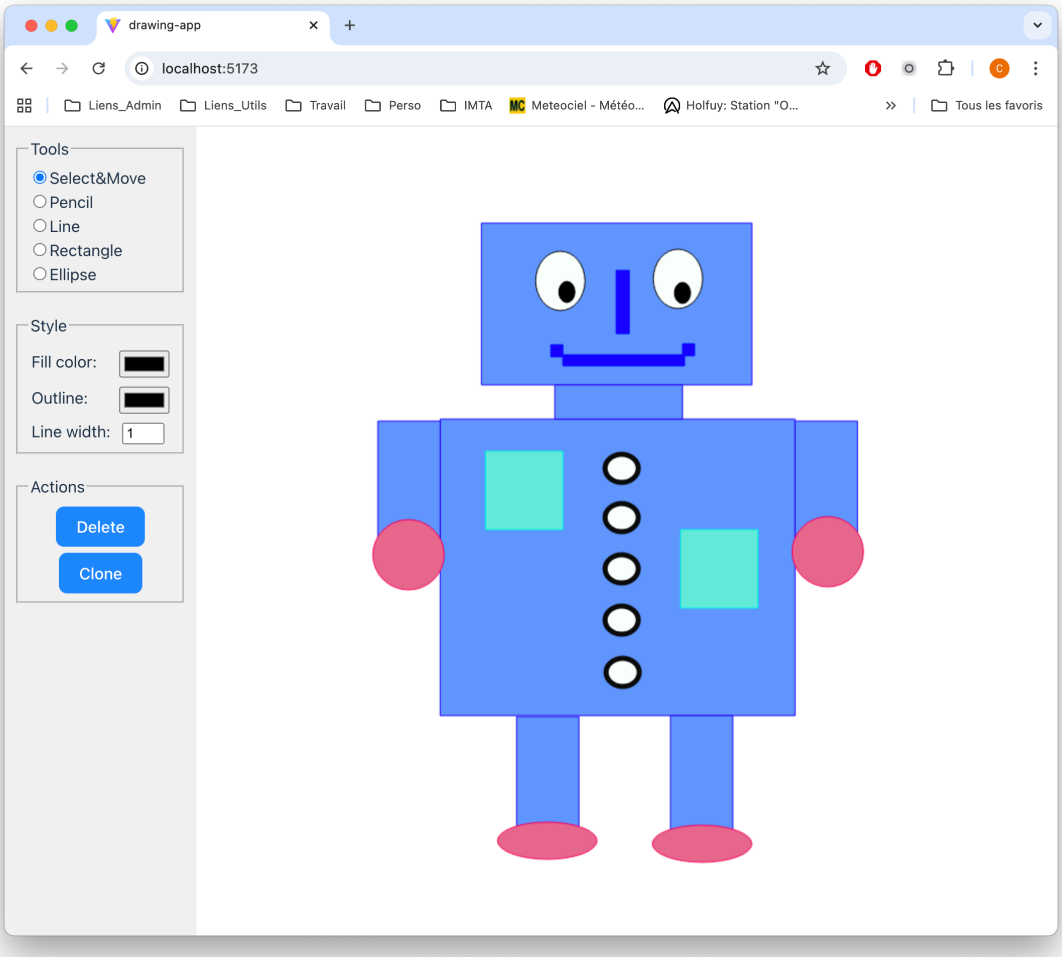
IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TAF RVSI - UE ISI
Année 2025 - 2026

Lab3: React application development

Create a drawing tool



1. Explore the provided Graphical User Interface of the application

We have provided you with a code template with a mockup GUI already implemented for this application:

- Download `lab3-drawing-app.zip` on Moodle and decompress it.
- In the terminal, run "`npm install`" to install the node modules. It should fix the errors in your project.
- In the terminal, run "`npm run dev`" and open the provided URL in a browser.

This interface contains the following functionalities:

- The possibility to choose the active drawing tool among a mode for selecting/moving 2D objects and modes for drawing rectangles, ellipses, lines, and paths.
- The possibility to choose two colors: one to fill the drawn objects and the other for their outline.
- The possibility to set the size of the outline in pixels.
- The possibility to delete the selected object.
- The possibility to clone the selected object.

Explore the code of this React project. The main components are :

- `src/App.tsx` contains the main application code. The main web page is composed of two elements : a toolbar (left) and a drawing canvas (right). It already contains hooks (`useRef` and `useEffect`) to initialize the drawing canvas and to set its size. It also contains a hook (`useState`) to save the persistent elements of the application.
- `src/Toolbar.tsx` contains the JSX code for the toolbar.
- `src/canvas/Canvas.tsx` contains the JSX code for the canvas, using `HTMLCanvasElement`.
- `src/canvas/CanvasItem.ts` contains an interface which represents any generic items drawn on the canvas and exposes core functionalities such as setting the color, drawing the object, move the object, etc.
- `src/canvas/RectangleItem.ts` implements `CanvasItem.ts`. It is an example that will allow you to draw rectangles on the canvas.
- `src/PersistentElements.ts` is a class storing the persistent elements of the application. For now, it contains references to the canvas and its context to allow us to draw on the canvas, as well as an array of CanvasItems, which stores all the items to draw on the canvas.

Each JSX component has its own CSS file, but you are not supposed to modify them for this lab.

2. Draw rectangles with the PersistentElements

The first step is to draw the simple rectangle in the Canvas. For that, you will check the TODO left in the code and you will need to modify the following three files:

- `PersistentElements.ts`: implement at least the "`addItem`" function to add the item to the items array and set it as the selected item.
- `Canvas.tsx`: add callback functions to the canvas in order to handle mouse events to be able to draw rectangles in the canvas. Each rectangle starts to be drawn when the user clicks in the canvas (`onMouseDown`), appears as soon as the user drags the mouse (`onMouseMove`) and remains fixed when the user releases the mouse (`onMouseUp`). For now, you can choose any color to fill the rectangle and its outline.
- `RectangleItem.ts`: the constructor and draw function are already implemented, so you just need to implement the update function, which update the width and the height of the rectangle when the user drags the mouse.

3. Manage the colors and the line width

Now, we want to connect the input elements of the colors and the line width to the rest of the application to set the fill and outline color, as well as the line width of the future shapes, but also of the currently selected shape. To achieve this, you need to modify the following files:

- `PersistentElements.ts`: add the three attributes to save them at the application level. Also add them in the constructor parameters and implement setter functions to modify them from external components.

- `App.tsx`: update the `PersistentElements` instantiation accordingly to set the initial colors and line width. In addition, give the `persistentElements` hook as a prop of the `Toolbar` component.
- `Toolbar.tsx`: add the `persistentElements` hook as a prop of the component and add input event handler on the dedicated input elements (`onChange`).
- `Canvas.tsx`: update the code to create rectangle with the defined colors and line width.

4. Manage the different modes

In this part, you need to implement a solution to enable the user to select the current mode among the modes for selecting/moving 2D objects, drawing rectangles, ellipses, lines, paths, etc. It is recommended, for a better code decomposition and for an easier evolution of the editor, not to make a switch or a succession of `if` tests to determine which mode must be used during the interaction. It is better to use a class that correctly manages the interaction according to the selection mode (you can use here the **State** design pattern).

- To help you, we provide you with the `Mode` interface which includes abstract methods for the basic functionalities of each mode.
- Create a hook to share the current `Mode` between the `Toolbar` and the `Canvas` components. Add prop in these components to access or modify this current `Mode`.
- Create new class for each mode by implementing the `Mode` interface. It is better to create each new class in a separate `.ts` file.
- To test the mode selection easily, you can display the name of the current mode in the browser console as a first step (`console.log("...")`).
- Move the specific code which create `RectangleItems` from the `Canvas.tsx` to this related new class (e.g. `RectangleMode`).
- Update the code of the `Canvas.tsx` to draw rectangle when the rectangle mode is selected (and do nothing when other modes are selected for now).

5. Implement the Select/Move mode

Enable the shape selection: when the "Select/Move" mode is active, when the user clicks on the canvas, the shape directly under the cursor should be selected.

- Implement the `contains` and `move` functions of the `RectangleItem` class. For `contains`, you can use the `CanvasRenderingContext2D.isPointInPath()` from the canvas context: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/isPointInPath
- Implement the `getItemAt` function from the `PersistentElements` class to handle the picking of a shape at a location `(x, y)` on the canvas. To do that, you have to check one by one each `CanvasItem` displayed on the canvas with the function `contains`.
- Implement the `SelectMoveMode` accordingly.
- Make sure that when the user selects a shape, the color and line-width input fields are updated to reflect the fill and outline colors of this shape.
- Later, you could also include a feedback to show to the user which item is selected. You can simply change the color of the outline or increase its line width.

6. Implement the behavior of Delete and Clone buttons

- Set the appropriate actions for the Delete and Clone buttons so that users can delete or clone (with a little translation) the selected shape.
- Later, you could also activate/deactivate the two buttons if an object is selected or not.

7. Draw the other shapes

Add the support of other shapes by creating the `EllipseItem`, `LineItem` and `PathItem` classes. They should support the same features: move, delete, color change, etc. To draw `LineItem` and `PathItem` ("Pencil" mode), you will have to directly use the `moveTo` and `LineTo` functions of the `Path2D`:
https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/moveTo
https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/lineTo

8. Manage the depth order of the objects (Bonus)

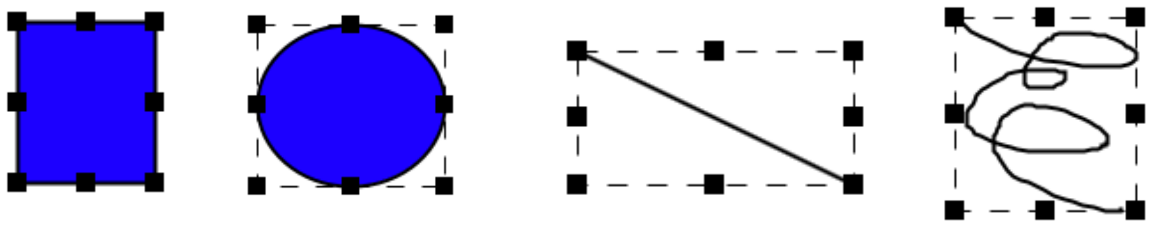
Enable users to change the depth order of the objects in the canvas by using the mouse wheel. It should allow users to bring objects to the foreground or to move them to the background. The `PersistentElements` achieves a basic rendering and displays the objects in the order the `Array` used to store the objects:

- The first object of the `Array` is rendered first and thus appears on the background,
- The following objects are rendered on top of each other.

Consequently, to change the depth order of the objects, you will just have to change their order in the `Array` of `PersistentElements`.

To detect the actions on the mouse wheel, you can handle the `onWheel` events on the `Canvas.tsx`.

9. More 2D drawing: add deformation to the objects (Bonus)



The goal of this part is to add modification handles on the bounding box of any objects and to allow users to deform the object by dragging these handles.

- First, draw a dashed line to represent the bounding box of the selected object as on the picture. You should compute the bounding box of each type of shapes.
- Then, add the eight handles around the selected object. You must find a generic solution to easily position the eight handles relative to the selected object.
- Implement the deformation to have a consistent behavior when users drag the handles for each type of shapes. You also need to consider the case in which users « reverse » the object by dragging the handles more than the widht or the height of the objects. You can try to use the `scale` function for that:
https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/scale

10. Contact

Cédric Fleury - cedric.fleury@imt-atlantique.fr