Group members

- Mina nabil 22011290
- Abdelrahman Khaled 22010877
- Omar Khaled 22010962
- Yaseen asaad 22011349

Table of Contents

Introduction

Project Overview: Web Paint Application

This project is a web-based paint application built with **React.js** for the front-end and **Spring Boot** for the back-end. The application allows users to create, edit, and manipulate shapes on a canvas. Users can select from various shapes (e.g., rectangles, circles), apply colors, adjust stroke properties, and use shadow effects. The paint area dynamically resizes based on the window size and supports real-time shape creation, movement, and modification.

Key features:

- Draw and manipulate shapes (rectangles, circles, etc.)
- Adjust shape properties like color, stroke, opacity, and shadows
- Real-time shape rendering with state synchronization
- Responsive canvas that adjusts to different screen sizes

User Guide

Project GitHub Repository:

Mina-Nabil2004/Web-Paint

Instructions to Download and Run the Project

1. **Cloning the Project from GitHub:**
   - Open your Git Bash terminal.
   - Run the following command to clone the project:

     ```
     git clone https://github.com/Mina-Nabil2004/Web-Paint.git
     ```

2. **Running the Back-End Server:**
   - Open the **Back-End** project folder in your preferred IDE (e.g., Visual Studio Code, IntelliJ).
   - Run the project using the **Run** button or the terminal in your IDE.
3. **Running the Front-End Server:**
   - **Install Node.js**: Download and install Node.js from the official website.
   - **Install React Dependencies**:
     - Open your Command Prompt (or terminal) and navigate to the **Front-End** project folder.
     - Run the following command to install necessary dependencies:

       ```
       npm install
       ```

   - **Install React-Konva**:
     - Run the following command to install `react-konva` and `konva`:

       ```
       npm install react-konva konva --save
       ```

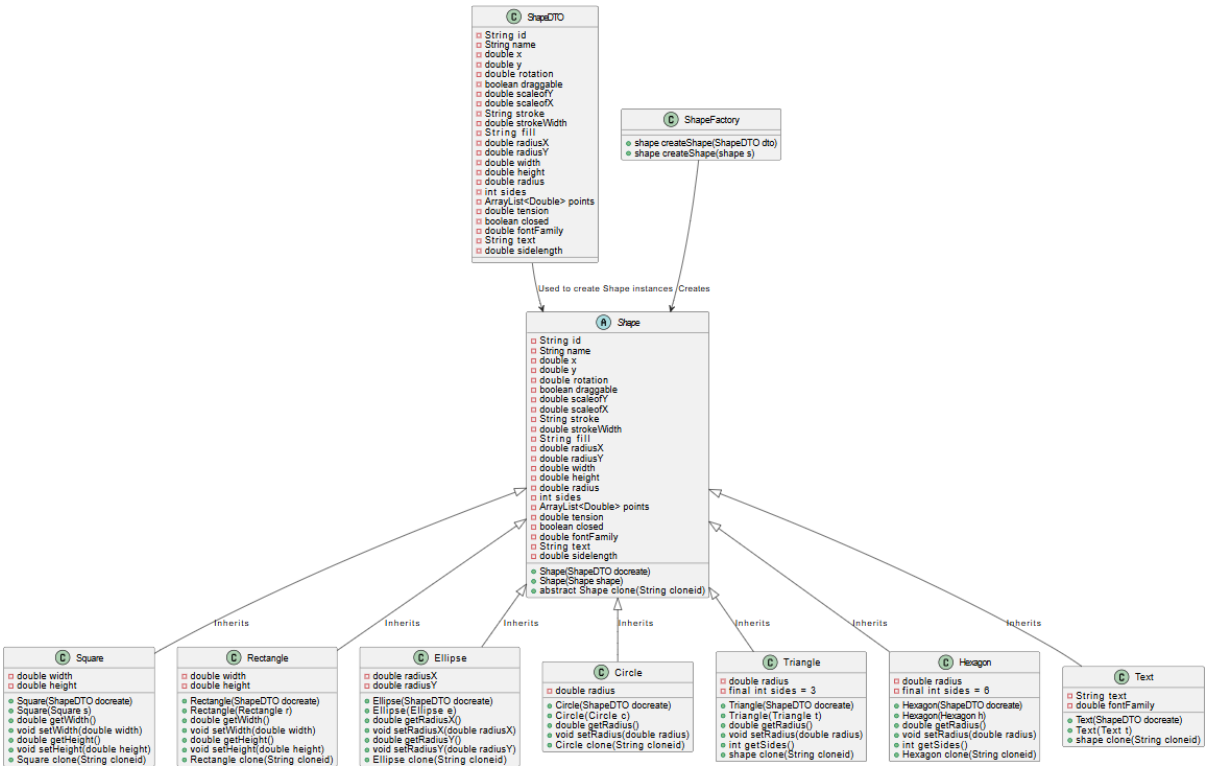
   - **Run the Front-End Server:**
     - In the terminal, run:

       ```
       npm start
       ```

     - This will start the React development server, and the project should be running locally.

Once these steps are completed, both the **Back-End** and **Front-End** servers will be running, and you can start working with the web-based Paint application.

UML Diagram


ShapeDTO
String id
String name
double x
double y
double rotation
boolean draggable
double scaleofY
double scaleofX
String stroke
double strokeWidth
String fill
double radiusX
double radiusY
double width
double height
double radius
int sides
ArrayList<Double> points
double tension
boolean closed
double fontFamily
String text
double sidelength
ShapeFactory
shape createShape(ShapeDTO dto)
shape createShape(shape s)
Used to create Shape instances
Creates
Shape
String id
String name
double x
double y
double rotation
boolean draggable
double scaleofY
double scaleofX
String stroke
double strokeWidth
String fill
double radiusX
double radiusY
double width
double height
double radius
int sides
ArrayList<Double> points
double tension
boolean closed
double fontFamily
String text
double sidelength
Shape(ShapeDTO docreate)
Shape(Shape shape)
abstract Shape clone(String cloneid)
Inherits
Square
double width
double height
Square(ShapeDTO docreate)
Square(Square s)
double getWidth()
void setWidth(double width)
double getHeight()
void setHeight(double height)
Square clone(String cloneid)
Rectangle
double width
double height
Rectangle(ShapeDTO docreate)
Rectangle(Rectangle r)
double getWidth()
void setWidth(double width)
double getHeight()
void setHeight(double height)
Rectangle clone(String cloneid)
Ellipse
double radiusX
double radiusY
Ellipse(ShapeDTO docreate)
Ellipse(Ellipse e)
double getRadiusX()
void setRadiusX(double radiusX)
double getRadiusY()
void setRadiusY(double radiusY)
Ellipse clone(String cloneid)
Circle
double radius
Circle(ShapeDTO docreate)
Circle(Circle c)
double getRadius()
void setRadius(double radius)
Circle clone(String cloneid)
Triangle
double radius
final int sides = 3
Triangle(ShapeDTO docreate)
Triangle(Triangle t)
double getRadius()
void setRadius(double radius)
int getSides()
shape clone(String cloneid)
Hexagon
double radius
final int sides = 6
Hexagon(ShapeDTO docreate)
Hexagon(Hexagon h)
double getRadius()
void setRadius(double radius)
int getSides()
Hexagon clone(String cloneid)
Text
String text
double fontFamily
Text(ShapeDTO docreate)
Text(Text t)
shape clone(String cloneid)

Design patterns

1. Factory Method Pattern:

Purpose:
To create shapes dynamically without specifying the exact class of the object that will be created.

Class Involved:
`ShapeFactory` is responsible for creating various shape objects based on `ShapeDTO`.

```
// ShapeFactory.java
public class ShapeFactory {
    public Shape createShape(ShapeDTO dto) {
        switch (dto.getName()) {
            case "Circle":
                return new Circle(dto);
            case "Rectangle":
                return new Rectangle(dto);
            case "Square":
                return new Square(dto);
            case "Ellipse":
                return new Ellipse(dto);
            case "Triangle":
                return new Triangle(dto);
            case "Hexagon":
                return new Hexagon(dto);
            case "Text":
                return new Text(dto);
            default:
                throw new IllegalArgumentException("Invalid shape type");
        }
    }
}
```

In this case, the `ShapeFactory` method decides which shape object to create based on the provided `ShapeDTO`. This allows for dynamic creation of shape objects without needing to manually specify the exact shape class.

2. Prototype Pattern:

Purpose:
To clone objects of various shapes, allowing for the creation of new objects that are copies of existing ones.

Class Involved:
Each shape class (like `Circle`, `Square`, etc.) implements a `clone()` method.

```
// Shape.java (abstract class)
public abstract class Shape {
    // Other fields...
```

```
    // Abstract method for cloning
    public abstract Shape clone(String cloneId);
}

// Circle.java (concrete class)
public class Circle extends Shape {
    private double radius;

    public Circle(ShapeDTO docreate) {
        // Initialization logic
    }

    public Circle(Circle c) {
        this.radius = c.radius;
    }

    @Override
    public Shape clone(String cloneId) {
        return new Circle(this);
    }
}
```

The `clone()` method in each shape class allows for duplicating a shape with the same properties, enabling an easy way to generate new shapes from existing ones.

3. Singleton Pattern:

Purpose:
To ensure that a class has only one instance and provide a global point of access to it.

Class Involved:
Depending on your application, you may have a service class, a configuration manager, or a logging utility that is needed globally and should only be instantiated once.

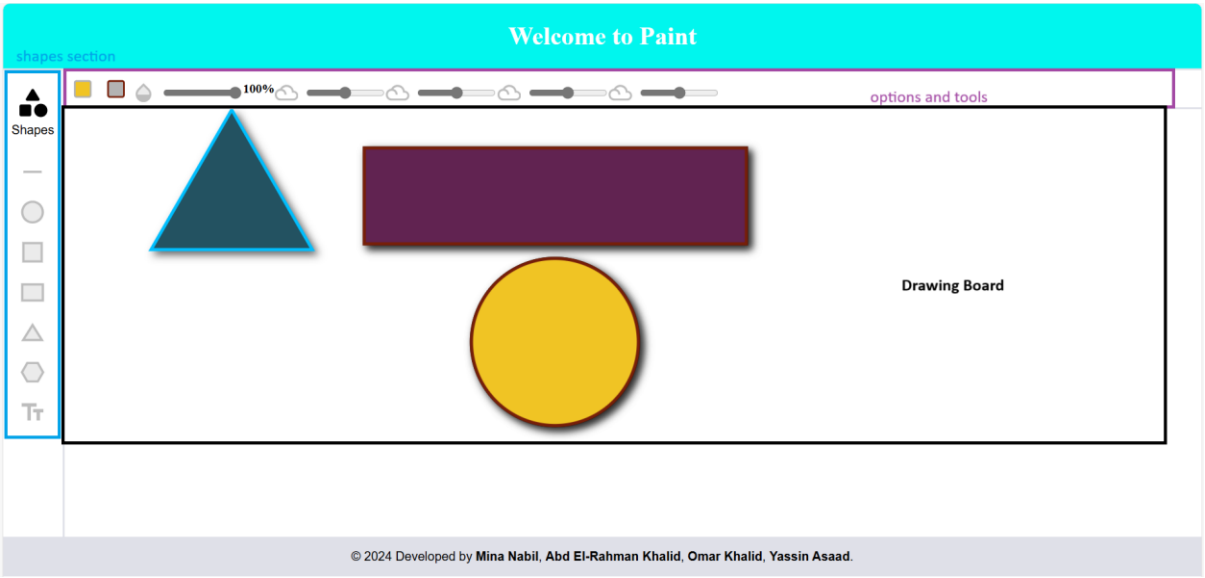
```
// Example of a Singleton class for managing shapes globally
public class ShapeManager {
    private static ShapeManager instance;

    // Private constructor to prevent instantiation
    private ShapeManager() {}

    // Static method to get the single instance
    public static ShapeManager getInstance() {
        if (instance == null) {
            instance = new ShapeManager();
        }
        return instance;
    }
}
```

In this example, `ShapeManager` ensures that only one instance is created during the lifecycle of the application, making it globally accessible.

UI Overview



Available Features and Drawing Tools

1 undo and redo

2 save and load

3 fill and strocke colours

4 delete

5 resize

6 opacity

7 shadow effects

Implementation details

Refrences

Material Icons - Material UI

react-konva - declarative canvas components for React | Konva - JavaScript 2d canvas library

JSON