Programming II

Assignment 3: Paint

Team members:

Abdel Rahman Mohamed Ahmed Ahmed Nasr - 22010887

Lojine Sameh Abdel Moneam Abdel Gawad Morsy - 22011054

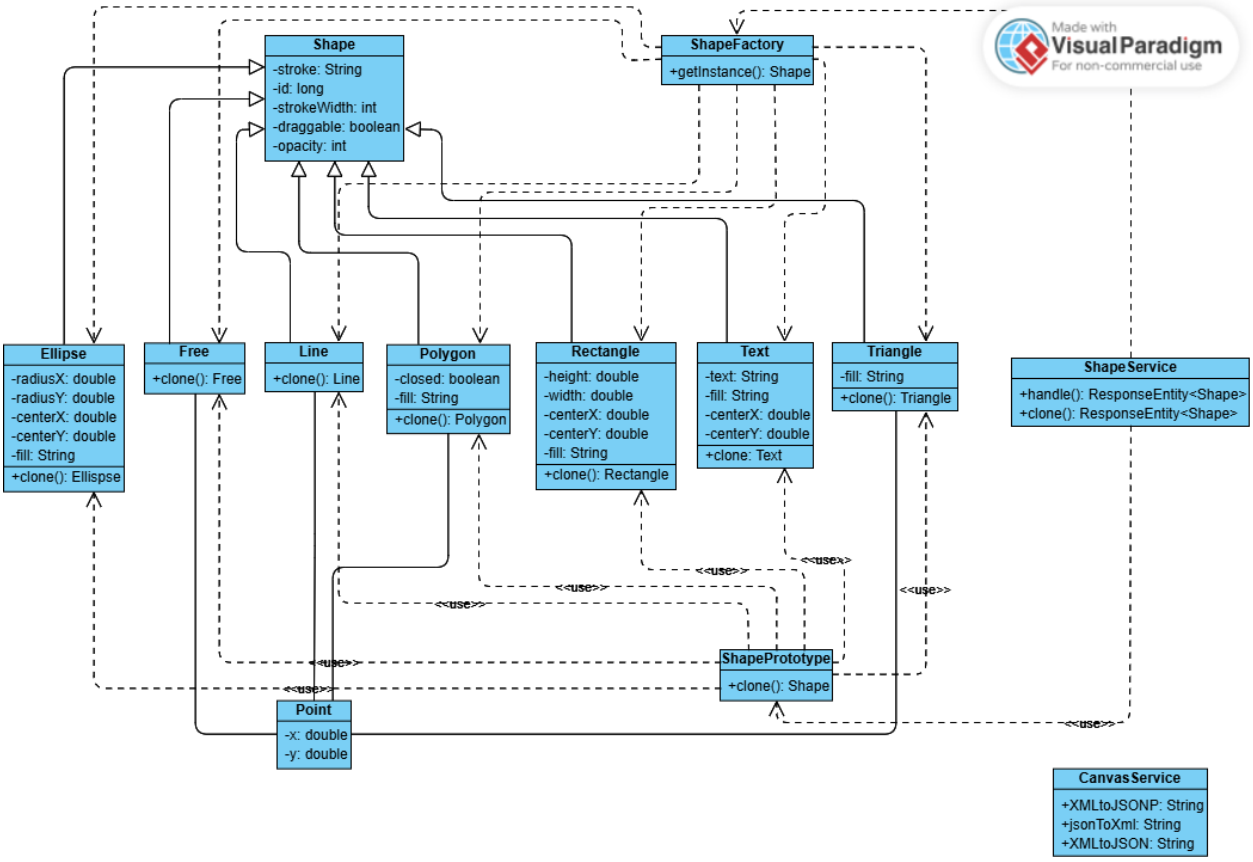
Ahmed Ragy Hussien - 22011690

Nour Eldien Taha Ahmed - 22011310

Steps to use:

1. First you need to initialize the spring boot application / server.
2. Head into the following file. "Paint\Backend\src\main\java\Paint\Deadline\DeadlineApplication.java" and run it in order to run the backend server.
3. Then onto the React application, Head over to the root directory of the React application and run “npm start” command to run the application
4. Now the application is completely functional.

UML Diagram:



This diagram envelopes all our codebase in the backend, that consists of all the:

- *Services (facades)*
- *Abstract Classes*
- *Concrete Classes*
- *Helper Classes*
- *Factories*
- *Prototypes*

Design decisions:

- Points are used to draw some shapes (triangle, polygon, line and free drawing). Points act vertices of that shape. This makes for easy, highly customizable shape drawing.
- Shape serves as the parent class for all drawable objects and defines shared properties. This ensures consistency and reusability by sharing common behavior and properties among all shape types.
- Implementation of clone(), which suggests the use of the Prototype Design Pattern. This allows for creating copies of shape objects efficiently without knowing their exact class. By defining cloning in ShapePrototype rather than tightly coupling it to Shape, allowing flexibility.
- Every shape has a unique identifier (id: long). This ensures traceability and allows shapes to be managed distinctly, even when cloned or manipulated.
- Load and save for JSON files are done in the frontend, while load and saving XML files are done in the backend. This reduces the risk of overlapping responsibilities or redundant implementations.

Implemented Design Patterns:

- ***Factory.***
  There is a shapeFactory class implemented that receives the required shape that should be returned to the frontend in order to be drawn, it sets a default for all constructors of all concrete shapes and takes the resulting shape subclass (Rectangle , Ellipse , etc..) and sends it to the front end using the ShapeController API.
- ***Prototype.***
  There is a ShapePrototype class implemented that receives a concrete object of one of the subclasses of Shape, checks its class, and calls the relative clone method that returns an exact 1-to-1 copy of the shape that it is received (NOTE : the prototype does NOT return the same object as a new object but rather creates a new object using the clone methods implemented in each concrete shape class).
- ***Facades.***
  Facades are subtly implemented in the code in the form of the canvasService and ShapeService classes, the ShapeService class is the class responsible for using the ShapeFactory and ShapePrototypes modules, it is the only module directly accessible in the ShapeController, it is used to create and clone objects. The CanvasService also acts a facade being the only accessible module in the canvasCotnroller, it carries its functionality within it unlike the shapeService that

passes the operations to either modules it includes, it is resposible for the save and load operations , often used to convert from JSON to XML and from XML to JSON.

There are other Design Patterns that are implemented using The Spring Boot Framework. Examples are :

- ***Singleton and Dependency Injection.***
  The services are passed to the controllers as Java Beans (using the @Service annotation) , they are injected using the @Autowired annotation rather than opting to constructor injection , Java Beans also ensure that classes are instantiated once. The ShapeFactory and ShapePrototype are also injected into the ShapeService using the @autowired annotation (each of them is marked as a Java Bean using @Component annotation). The use of singleton and DI in the backend ensures that resources are saved and usage and creation is kept to a minimum.