Paint App

Team Members

The project was developed collaboratively by the following team members:

- **Abdel-Rahman Amgad Hassan** (ID: 22010871)
- **Jana Mohamed** (ID: 22010691)
- **Mahmoud Hesham Mohamed** (ID: 22011201)
- **Mohamed ElSayed** (ID: 22011102)

1. Project Description

The **Paint App** is a user-friendly, web-based drawing application that enables users to create and manipulate shapes with attributes such as colors, sizes, and positions.

2. Technologies Used

The following technologies were utilized in the development of the application:

- **React.js:** Used for frontend development, ensuring a smooth and interactive user experience.
- **Spring Boot:** A powerful framework for backend development, enabling the implementation of RESTful APIs for communication.

3. Functionalities

The application offers the following core functionalities:

- **Shape Creation and Manipulation:** Users can draw and modify various shapes, including rectangles, circles, and lines, on the canvas.
- **Attribute Editing:** Users can adjust shape attributes such as color, size, and position dynamically.

- **Save and Load:** Users can save their designs for future editing or reuse.
- **Undo and Redo:** Actions can be undone or redone to provide better control over the drawing process.
- **Backend Interaction:** Shapes are stored and retrieved through dynamic communication between the frontend and backend.

4. How to Run the Application

Follow these steps to set up and run the application:

1. **Frontend:**
   - Navigate to the frontend directory.
   - Install the necessary dependencies by running: `npm install`.
   - Start the development server with the command: `npm run dev`.
2. **Backend:**
   - Navigate to the backend directory.
   - Launch the backend application by running the `BackendApplication` file.
3. **Access the Application:**
   - Open your browser and navigate to: `http://localhost:5173` (default port).
   - The application's functionalities will now be fully accessible.

5. Additional Features

0.1 Advanced Save/Load Functionality

- **Local Saving and Loading:** The application allows users to open the file explorer to save and load drawings in XML or JSON format anywhere on their local machine.
- **Cloud Saving:** Users can save their drawings to the cloud, where the backend generates a unique ID for each drawing. Both the JSON/XML file and an associated PNG image are stored for each entry.
- **Cloud Gallery:** The application displays a list of all saved drawings' images from the cloud, enhancing user experience by allowing easy browsing and selection.

0.2 Support for Images

- Users can drag and drop images from their local machine directly onto the canvas, integrating these images seamlessly into their drawings.

6. Application Design

0.3 Implemented Design Patterns

- **Factory Pattern:** A `ShapeFactory` class is used to create various shapes with different attributes dynamically.
- **Prototype Pattern:** The Undo/Redo feature utilizes the Prototype Pattern to clone shapes and push identical versions to the undo stack, ensuring a reliable history mechanism.
- **Singleton Pattern:** A `Drawing` class, implemented as a Singleton, holds all the application's data, ensuring a single source of truth for the drawing state.
- **State Pattern:** The application employs a state-based logic system for different modes, such as selection mode and drawing mode, to handle user interactions efficiently.

0.4 UML Class Diagram

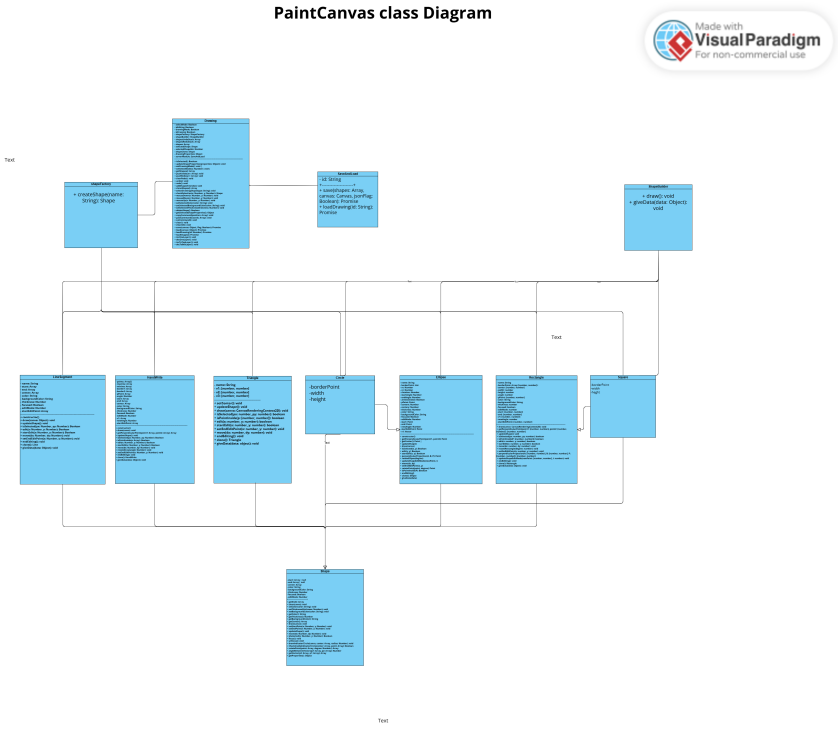

Figure 1: Class Diagram

7. Design Choices

- **Shape Creation and Logic:** The creation of shapes, along with logic for undo and redo operations, is handled on the frontend to ensure fast, real-time performance.
- **Cloud Saving and Loading:** The backend is responsible for saving files to the cloud. A unique ID is generated for each JSON file and its corresponding PNG image. These files are then returned to the frontend when needed for display or further interaction.

8. Backend API Endpoints

1. Save Drawing (JSON)

POST /drawings/json

- Saves a drawing as a JSON file with a unique ID.

2. Update Drawing (JSON)

PUT /drawings/id/json

- Updates an existing drawing (identified by its unique ID) with a new JSON representation.

3. Save Drawing (XML)

POST /drawings/xml

- Saves a drawing as an XML file with a unique ID.

4. Update Drawing (XML)

PUT /drawings/id/xml

- Updates an existing drawing (identified by its unique ID) with a new XML representation.

5. Save Image

POST /drawings/id/image

- Saves an image associated with a specific drawing (identified by its ID).

6. Get Drawing by ID

GET /drawings/id

- Retrieves a drawing (either JSON or XML format) by its unique ID.

7. Get All Images with IDs and Content

GET /drawings/images

- Retrieves a list of all saved images, including their unique IDs and Base64-encoded image content.

9. UI Snapshots

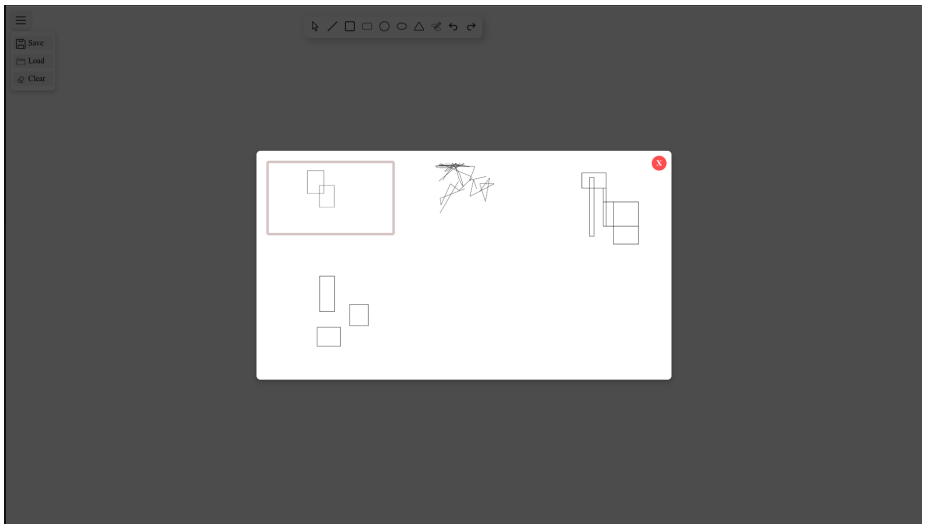

Figure 2: Loading From Gallery

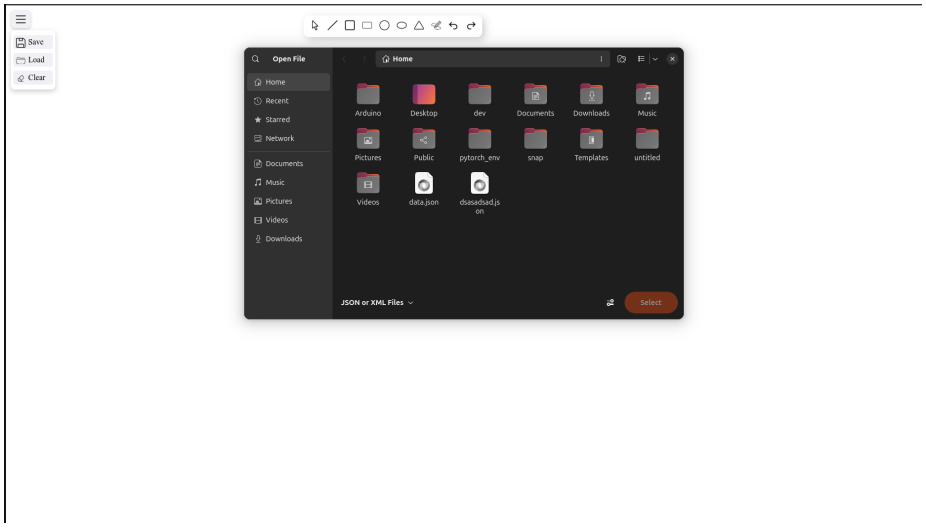

Figure 3: Loading From file explorer

6. User Manual

- **Saving Files:**
  - Click the save button located in the upper left corner of the interface.
  - Choose between saving your file locally (any location on your device) or to the cloud, where the backend handles the saving process.
  - Select the desired file format (JSON or XML) to save your drawing.
- **Loading Files:**
  - You can load files from the **Gallery**, which displays a list of saved drawings as images stored in the backend. Simply select the desired drawing to load it.
  - Alternatively, you can manually load files from your local device using the file explorer.