Paint Application - Technical Report

Team Members

- Jana Mohamed Rashed (23010359)
- Ahmed Sherif Abd El-Moniem Ali Abo El-Soud (23010199)
- Mohamed Radwan (23010745)

---

1. Introduction

This report documents the design and implementation of our Object-Oriented Paint Application developed using:

- **Frontend:** Angular (TypeScript) with Fabric.js for canvas rendering
- **Backend:** Java Spring Boot with RESTful APIs
- **Data Exchange:** JSON and XML serialization
- **Architecture:** Client-server model with clear separation of concerns

The application enables users to draw geometric shapes (circles, rectangles, triangles, ellipses, lines, freehand paths), manipulate them through transformations (move, scale, rotate), customize visual properties (colors, stroke width, opacity), and persist drawings through import/export functionality.

---

2. Assumptions

To maintain architectural clarity and implementation feasibility, we established the following assumptions:

- **Single User Session:** The application operates as a single-user desktop tool without concurrent multi-user collaboration features.
- **Coordinate System:** All shapes use absolute canvas coordinates with the origin (0,0) at the top-left corner.
- **Shape Identity:** Each shape maintains a unique UUID for tracking across frontend-backend communication.
- **State Management:** The backend maintains the authoritative state; the frontend synchronizes after each operation.
- **Browser Compatibility:** Modern browsers with ES6+ support and File System Access API for save/load operations.

---

3. Application Flow (Frontend → Backend)

- ➢ Shape Creation Flow
  When a user draws a shape:
  - **Frontend:** User selects a tool from the side toolbar and draws on the canvas using mouse events
  - **Fabric.js:** Creates a fabric object with visual properties and assigns a UUID
  - **DTO Conversion: FabricToDtoService** transforms the fabric object into a ShapeDTO containing type, coordinates, and properties
  - HTTP Request: Angular's **HttpClient** sends a POST request to /drawing/add with the DTO
  - **Backend Processing: DrawingController** receives the DTO, ShapeFactory creates the appropriate shape instance, and DrawingService stores it in a LinkedHashMap
  - **Response:** Backend confirms successful addition, frontend renders the shape

- ➢ Updating a Shape
  - User modifies shape properties (color, size) or transforms it (move, rotate, scale)
  - Fabric.js emits an **object:modified** event
  - Frontend converts updated fabric object to DTO
  - PUT request to **/drawing/update** with the complete shape state
  - Backend replaces the existing shape in the map
  - Canvas re-renders automatically

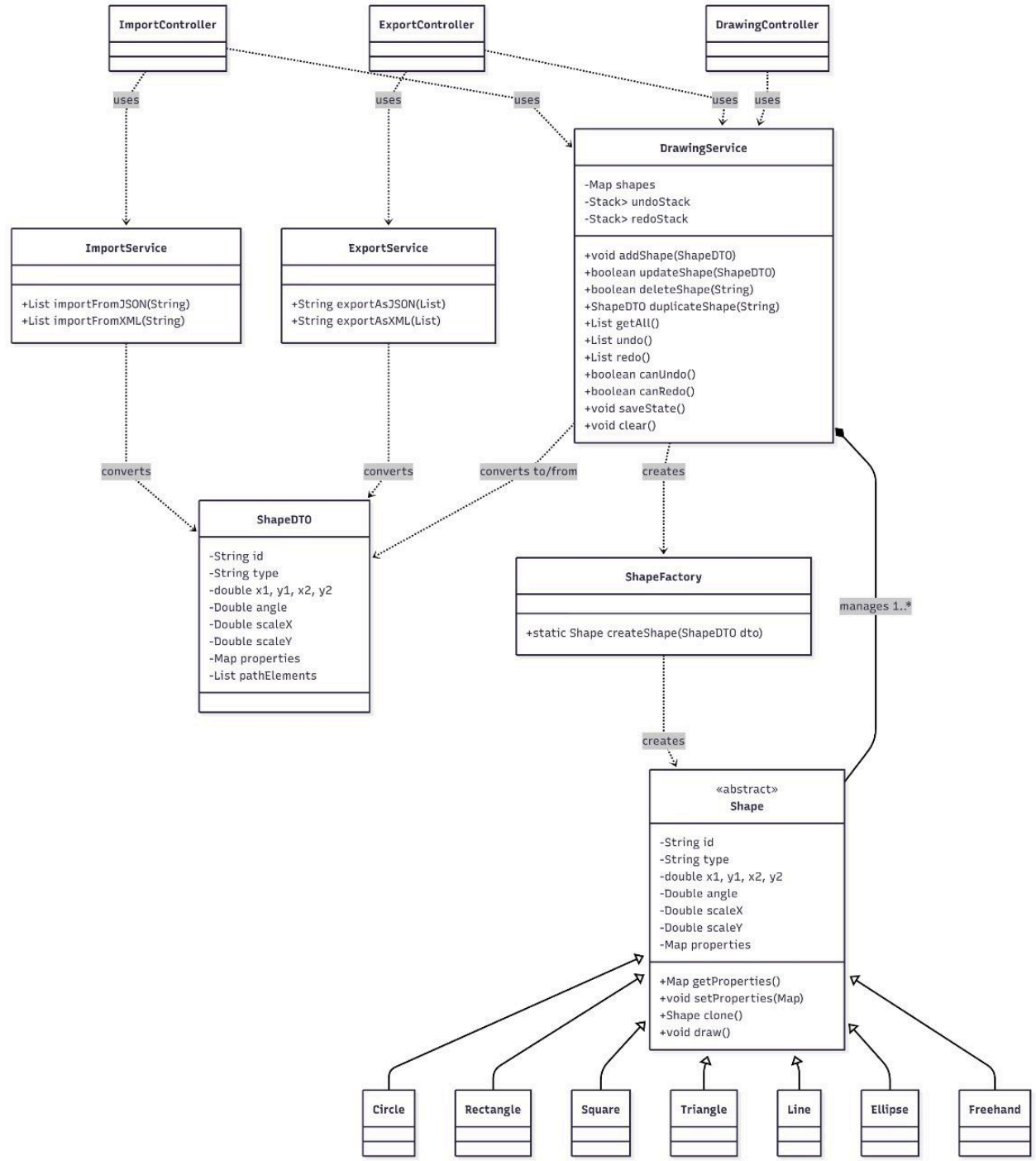
- ➢ Undo/Redo Operations
  - Save State: Before each operation, **DrawingService** creates a deep copy of the current shapes map and pushes it to the undo stack
  - **Undo:** Pops from undo stack, pushes current state to redo stack, restores previous state
  - **Redo:** Reverse of undo operation
  - **Frontend Sync:** Backend returns the complete shape list, frontend clears canvas and redraws all shapes from DTOs

- ➢ Save / Load Flow
  - User clicks Save → File System Access API prompts for location
  - Backend's **ExportService** iterates through shapes, converting each to the target format
  - JSON uses Jackson's **ObjectMapper** for straightforward serialization
  - XML requires custom wrapper classes **(DrawingWrapper, XMLShapeDTO)** for proper structure
  - File downloads to user's chosen location

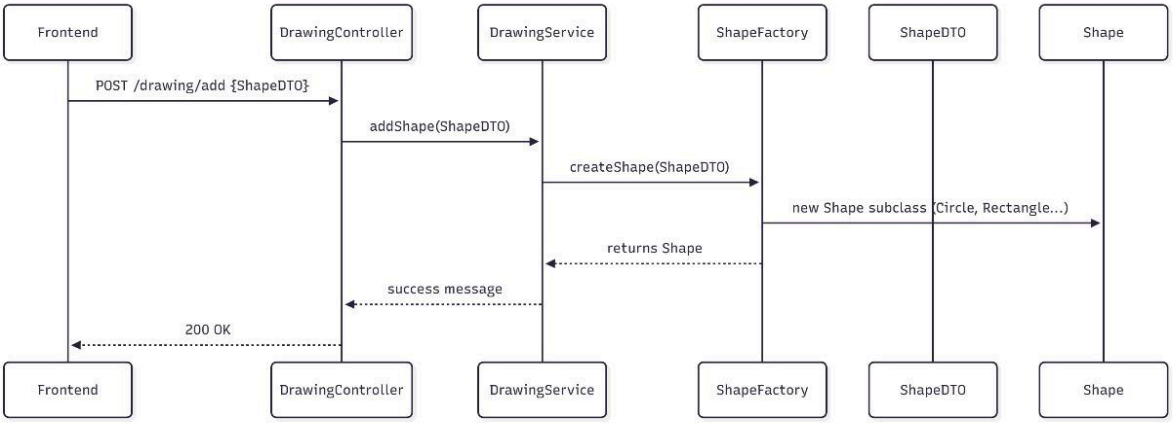
---

4. Object-Oriented Design

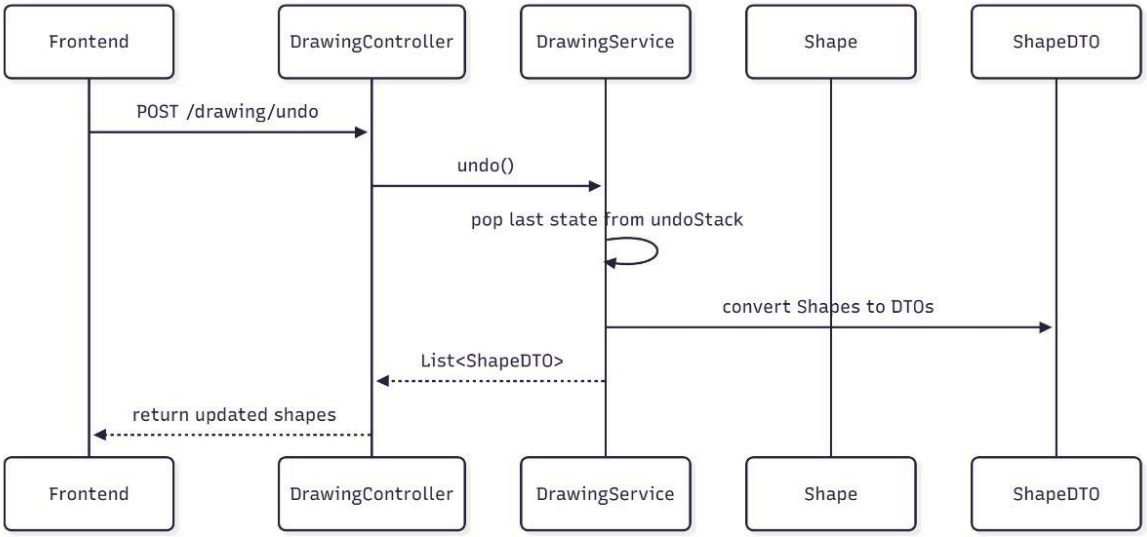
Class Diagram



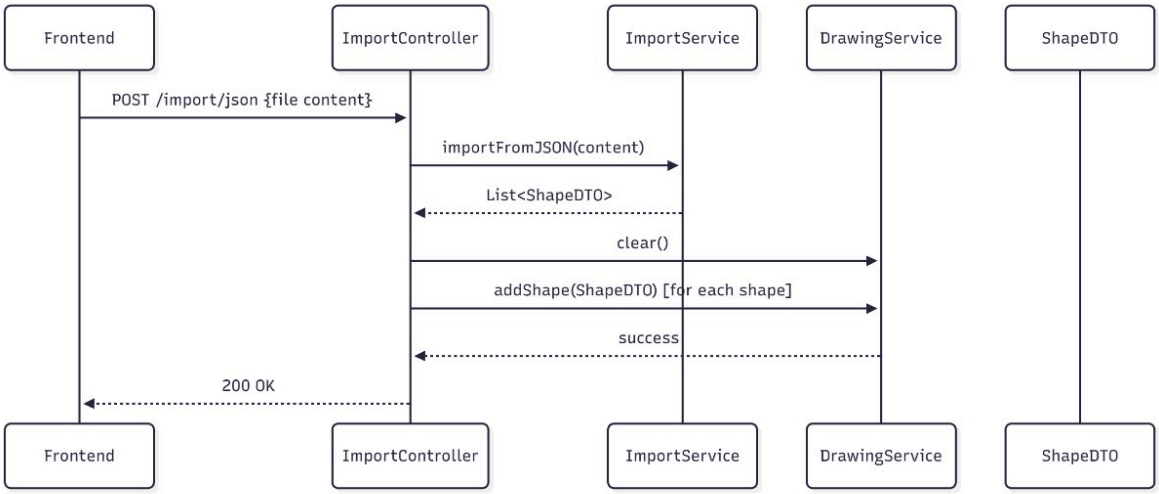
Add Shape Sequence Diagram



Undo Sequence Diagram



Import JSON Sequence Diagram



---

5. Design Patterns Used

- ➢ Factory Pattern
  - ➔ **Purpose:** Encapsulate shape instantiation logic and decouple shape creation from shape usage.
  - ➔ **Implementation:** The **ShapeFactory** class provides a single static method that takes a **ShapeDTO** and returns the appropriate **Shape** subclass
  - ➔ This allows the **DrawingService** to create shapes without knowing their concrete types, making it easy to add new shapes without modifying existing code.

- ➢ Prototype Pattern
  - ➔ **Purpose:** Enable efficient shape duplication without re-parsing DTOs.
  - ➔ **Implementation:** The **Shape** base class implements **Cloneable** and provides a **clone()** method:
  - ➔ The **duplicateShape** endpoint uses this to create copies with offset positions, preserving all visual properties while assigning a new UUID.

- ➢ MVC Pattern
  - ➔ **Implementation:**
    Model: Shape hierarchy and **DrawingService** (backend),
    Fabric.js objects (frontend)
    View: Angular components (Canvas, Toolbars, Properties Panel)
    Controller: DrawingController handles HTTP requests, Angular services coordinate between view and model

---

6. Design Decisions

- ➢ Why Fabric.js?

Decision: Replace custom canvas rendering with Fabric.js library.

Rationale:

- **Built-in Transformations:** Handles rotation, scaling, skewing without manual matrix math
- **Object Model:** Provides first-class objects with events rather than pixel manipulation
- **Selection Handles:** User-friendly resize/rotate handles out of the box
- **Performance:** Optimized rendering and event handling for complex scenes

- ➢ State Synchronization Strategy

Decision: Backend as single source of truth with frontend synchronization after each operation.

Rationale:

- **Consistency:** Eliminates frontend-backend state divergence
- **Undo/Redo:** Centralizes history management on the server
- **Reliability:** Network failures don't corrupt state; can always reload from backend

➢ DTO vs Direct Serialization

Decision: Use explicit **ShapeDTO** for communication rather than serializing Shape objects directly.

Rationale:

- **Decoupling:** Frontend and backend shape representations can evolve independently
- **Validation:** DTOs provide a clear contract and validation layer
- **Multiple Formats:** Easier to support both JSON and XML with custom serialization

➢ LinkedHashMap for Storage

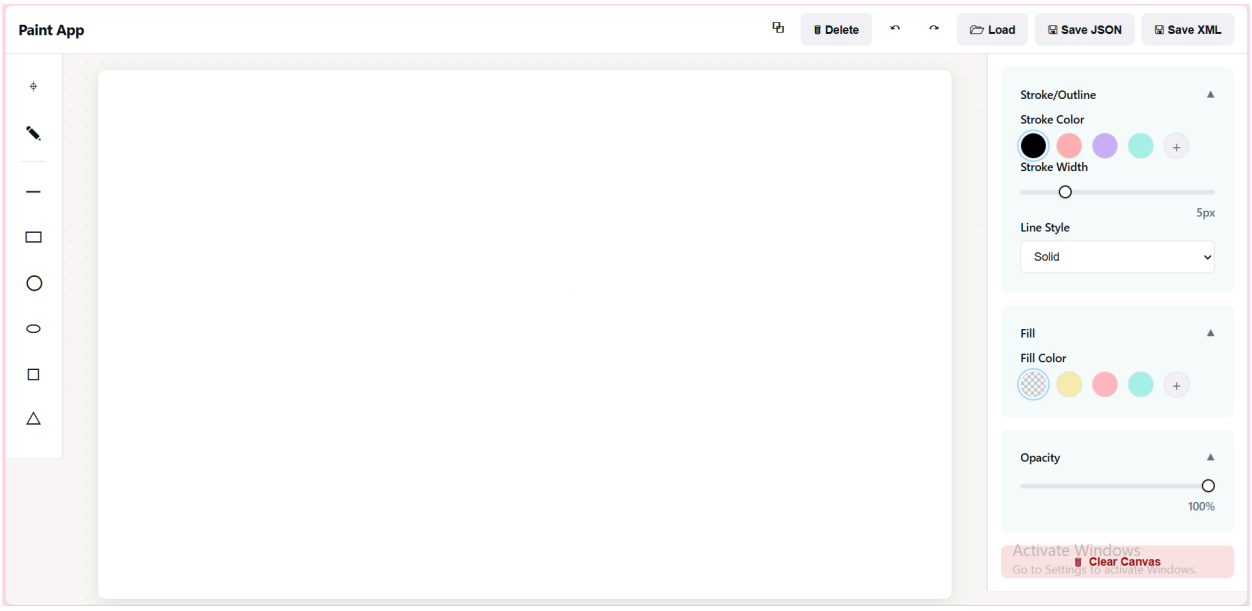
Decision: Store shapes in **LinkedHashMap<String, Shape>** instead of **ArrayList<Shape>.**

Rationale:

- **Fast Lookups:** O(1) access by ID for updates and deletes
- **Insertion Order:** Maintains drawing order for proper rendering layers
- **No Duplicates:** Prevents accidental duplicate IDs

---

7. User Interface (UI) Overview



The interface follows a standard creative tool layout:

- Header Toolbar:
  - Application title
  - Duplicate, Delete, Undo/Redo buttons
  - Save/Load buttons (JSON/XML)

- Side Toolbar:
  - Selection tool (pointer icon)
  - Freehand/pencil tool
  - Shape tools (line, rectangle, circle, ellipse, square, triangle)
  - Active tool highlighted with light blue background

- Canvas:
  - White drawing surface (1050×650px) with subtle dot grid background
  - Fabric.js handles object selection with blue control handles
  - Crosshair cursor during drawing operations

- Properties Panel:
  - Collapsible sections for Stroke, Fill, and Opacity
  - Color pickers with preset swatches and custom color input
  - Stroke width slider (1-20px)
  - Opacity slider (0-100%)
  - Line style dropdown (solid/dashed)
  - Clear Canvas button at bottom

---

8. User Guide

Getting Started:

- ➢ Run Backend: Execute DemoApplication.java (Spring Boot starts on port 8080)
- ➢ Run Frontend: Navigate to the Frontend/ directory and run ng serve (Angular dev server starts on port 4200)
- ➢ Open Browser:
  Go to: http://localhost:4200

Drawing Shapes:

- ➢ Click a shape tool from the side toolbar (e.g., **Rectangle**)
- ➢ Click and drag on the canvas to draw the shape
- ➢ Release mouse to finalize the shape
- ➢ The shape becomes **selectable automatically**

Modifying Shapes:

Visual Properties:

- ➢ Select a shape by clicking it
- ➢ Adjust properties in the right panel:
  - ○ Fill color
  - ○ Stroke color
  - ○ Stroke width
  - ○ Opacity

Transformations:

- ➢ Select a shape
- ➢ Drag corner handles to **resize**
- ➢ Drag the rotation handle (top center) to **rotate**
- ➢ Click and drag the shape body to **move**

Deleting Shapes:

- ➢ Select one or multiple shapes (hold **Shift** for multi-select
- ➢ Click the **Delete** button in the header, or press the **Delete** key
- ➢ Shapes are removed from both the canvas **and** backend

Undo/Redo:

- ➢ Click ↶ **Undo** to revert the last action
- ➢ Click ↷ **Redo** to reapply the undone action
- ➢ The system stores up to **50 operations** in history

Saving:

- ➢ Click **Save JSON** or **Save XML** in the header
- ➢ Browser prompts for save location
- ➢ Choose filename & destination
- ➢ File downloads containing the **complete drawing state**

Loading:

- ➢ Click the **Load** button
- ➢ Select a .json or **.xml** file previously saved
- ➢ Canvas clears and redraws all shapes from the file
- ➢ Backend state synchronizes automatically

---

9. Conclusion

This paint application successfully demonstrates object-oriented design principles through a practical, feature-rich implementation. Key achievements include:

- ➢ Design Pattern Application: Factory pattern for shape creation, Prototype for duplication, and MVC for architectural organization showcase classic OOP patterns in a real-world context.
- ➢ Full-Stack Integration: Seamless communication between Angular frontend and Spring Boot backend through RESTful APIs demonstrates modern web development practices.
- ➢ User Experience: Fabric.js integration provides professional-grade canvas interactions (transformations, selections) without sacrificing our OOP backend architecture.
- ➢ Data Persistence: Support for both JSON and XML formats demonstrates flexibility in serialization strategies while maintaining a consistent internal object model.
- ➢ Extensibility: The factory-based shape creation and abstract base class design make adding new shapes straightforward—simply extend Shape, implement required methods, and register in the factory.
- ➢ Future enhancements could include layer management, grouping objects, gradient fills, image import, and collaborative editing features, all building upon the solid OOP foundation established in this implementation.