جامعة الإسكندرية
ALEXANDRIA
UNIVERSITY

Teammates Names

Omar Tammam - 20011016

Joseph Shokry Soliman - 20010439

Osama Elsayed Belal - 20010269

Marwan Essam eldien Rashad - 20011859

1.1 GitHub Repositories

- **Back-End Repository**

  On main branch :

  https://github.com/josephShokry/paint-back-end.git

- **Front-End Repository**

  On main branch :

  https://github.com/Osama-Belal/Paint-front-end.git

1.2 Instruction to download codes

- **Downloading codes from GitHub repositories**
  1. Open your Git Bash terminal.
  2. Cloning Back-End files to your folder
     "git clone https://github.com/josephShokry/paint-back-end.git" .
  3. Cloning Front-End files to your folder
     "git clone https://github.com/Osama-Belal/Paint-front-end.git" .

1.3 Instruction to Run Back-End Server

- **Running Back-End codes**
  1. Open Back-End files to your favorite IDE to be run.
  2. Use the normal Run button in your IDE.

1.4 Instruction to Run Front-End Server

- **Running Front-End codes**
  1. Install Node.js from the official website
  2. Open your Command Prompt.
  3. In your Command Prompt "npm install -g @angular/cli" .
  4. Open Front-End files to your favorite IDE to be run.
  5. Running Angular server from prompt "ng serve --open" .

2.1 UML link

- **Full UML link:** https://lucid.app/lucidchart/db77523c-c02e-46f0-ae5f-7b8824747cf9/view?page=0_0&invitationId=inv_b7fa1ee2-0c9b-4678-93e5-e1e9fb213807#

2.2 UML hierarchy

- **Controller**

  Receives all front-end requests and delegates it to paint service.

- **Paint Service**

  Main Class in Back-end hierarchy

  1. Call Database methods.
  2. Delegate saving & loading requests to Saver Class.
  3. Delegate command to Command Factory Class.

- **Command Factory**

  Creates all Command Objects then Delegate the requested command to them.

- **Commands**

  Extend Command abstract Class & implements intended action according to their logic.

- **Shape Factory**

  Creates all Shape Objects then pass DTO to them to set their attributes.

- **Shapes**

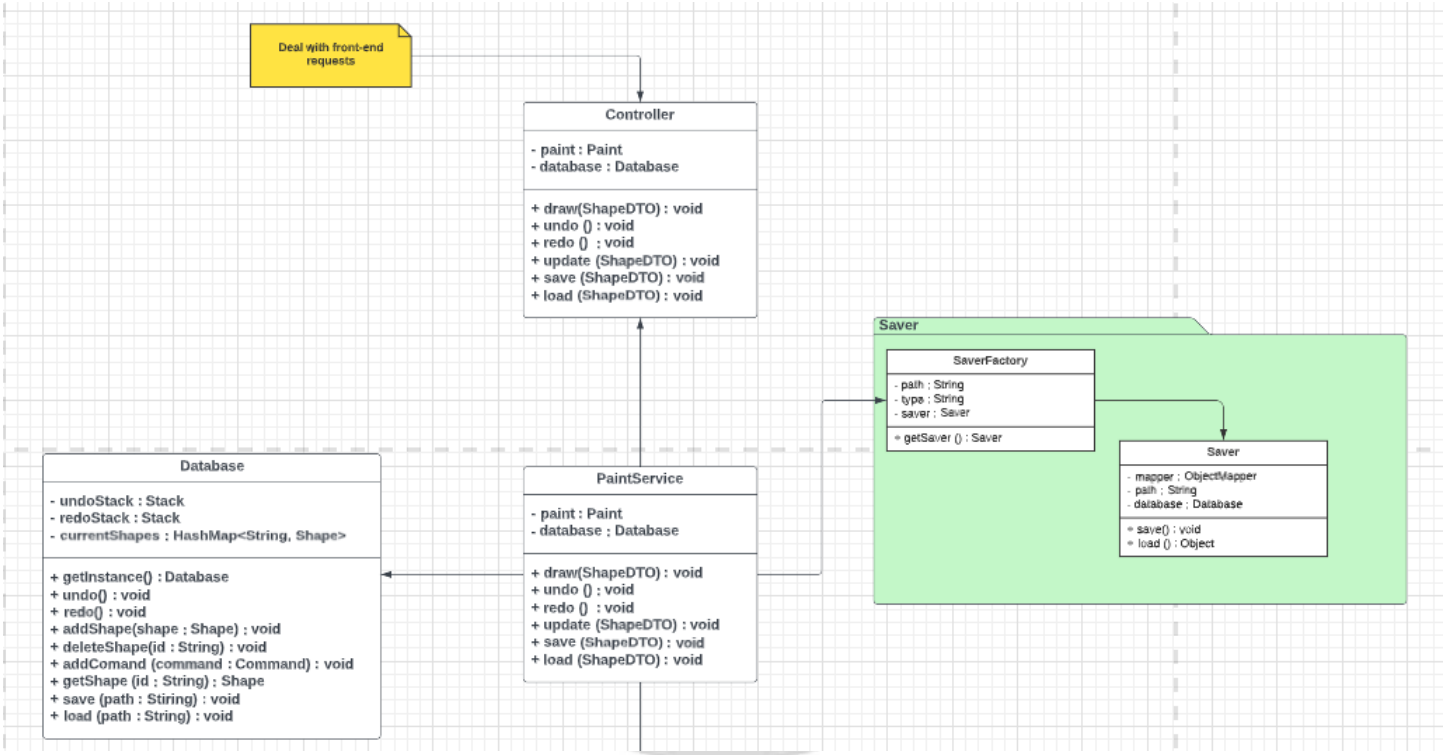
  Extend Shape abstract Class & Set their special and different attributes.

- **ShapeDTO (Data Transfer Object)**

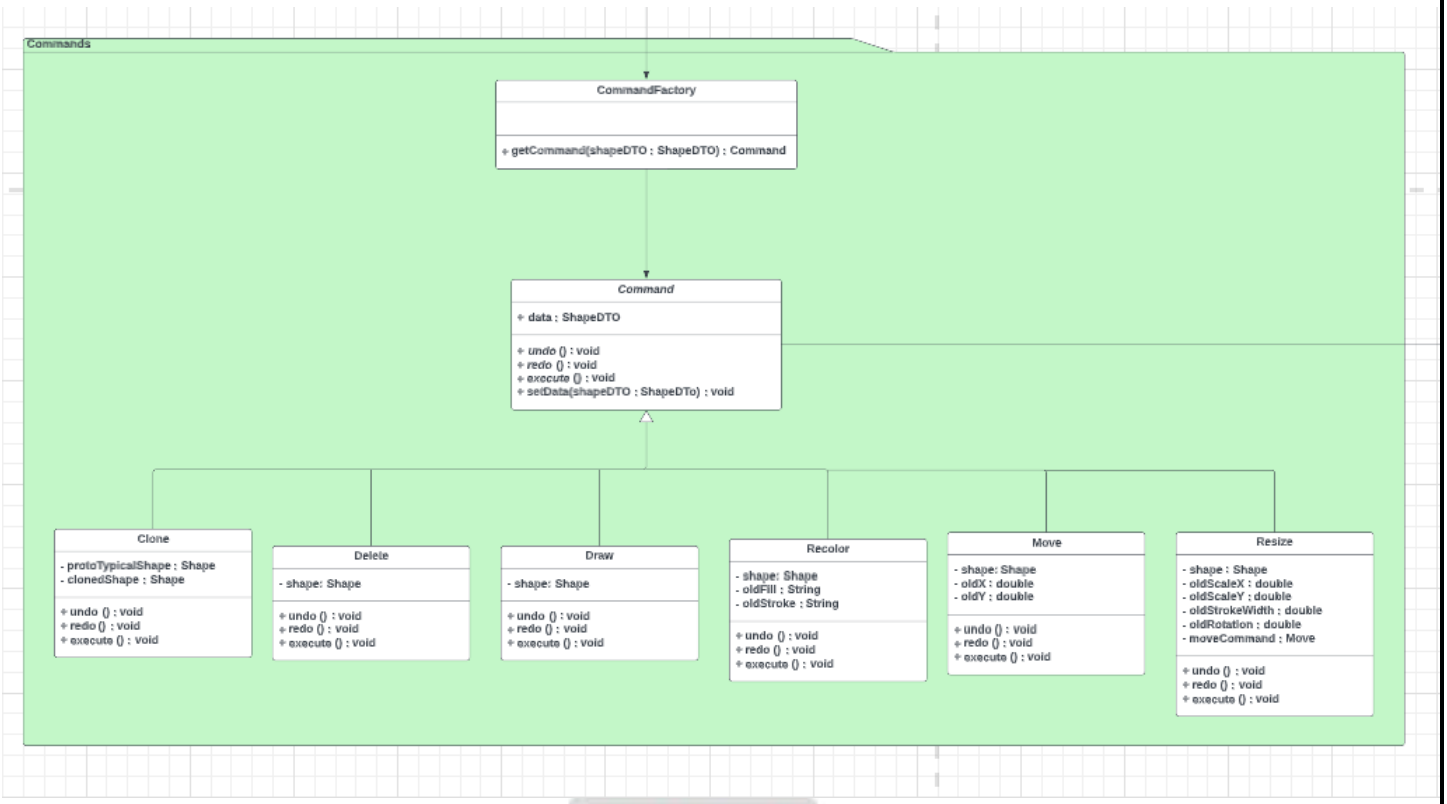
  Has all the attributes needed in our application.

2.3 UML snippets

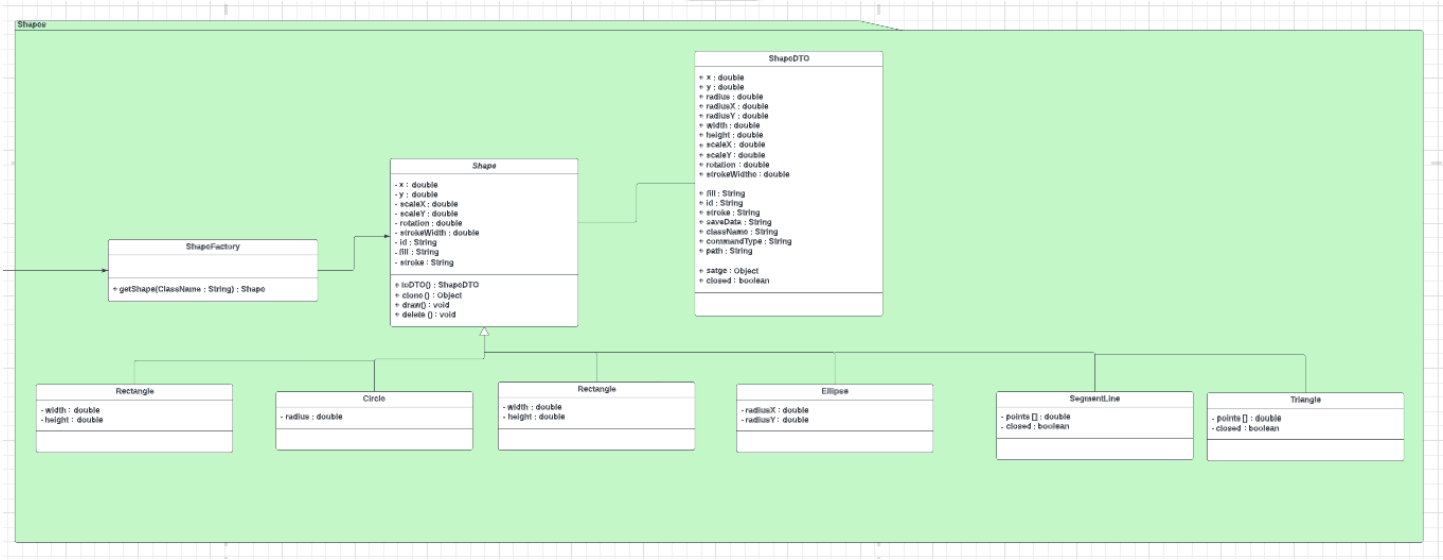
Controller & Database & Paint service & Saver



Command Package



Shapes Package



3.1 Command Design Pattern

- **Motivation**

  Any paint application is built using commands to do basic actions like draw or resize. Knowing this, we had to design software that would handle these actions in an abstract way. We had to Encapsulate each action in its own class in a way that would make implementing undo and redo functions in an easy way.

- **Commands**

  Each command class is responsible for its own execute, undo, and redo functionality. Doing this enabled us to handle functions like the undo and redo in a more clear and concise way. Each class has its own undo and redo function that defines how that action can be undone or redone.

- **Stacks**

  Next, we had to find a way to store all these actions. So, we made a database class that has 2 stacks for Undo and Redo, whenever we undo an action, we push that actions object to our redo stack, and vice versa.

  We also built a controller class that calls our Command factory and calls the commands we need and its undo & its redo as well.

3.2 Factory Design Pattern

- **Command Factory**

  We implemented factory design pattern to create all allowable commands according to their types.

- **Shape Factory**

  - We implemented factory design pattern to create all allowable shapes according to their types.
  - We implemented it in our front-end to handle the creation of konva shapes.

3.3 Prototype Design Pattern

We implemented prototype design pattern to clone all allowable shapes instead of creation of it again with the same attributes except the ID of the shape.

3.4 Singleton Design Pattern

We applied it for Database Class to return only the same object.

3.5 Adapter Design Pattern

- We used Konva to help us to draw shapes in our application and Konva uses certain data format to draw shapes, but In the backend, we used a slightly different format.
  This problem called for the Adapter design pattern.
- In our frontend, we implemented an adapter service, that we used to send data back and forth between our server's data and Konva's data, this service simply formats the data in a way that both sides would understand it.

4.1 Decisions

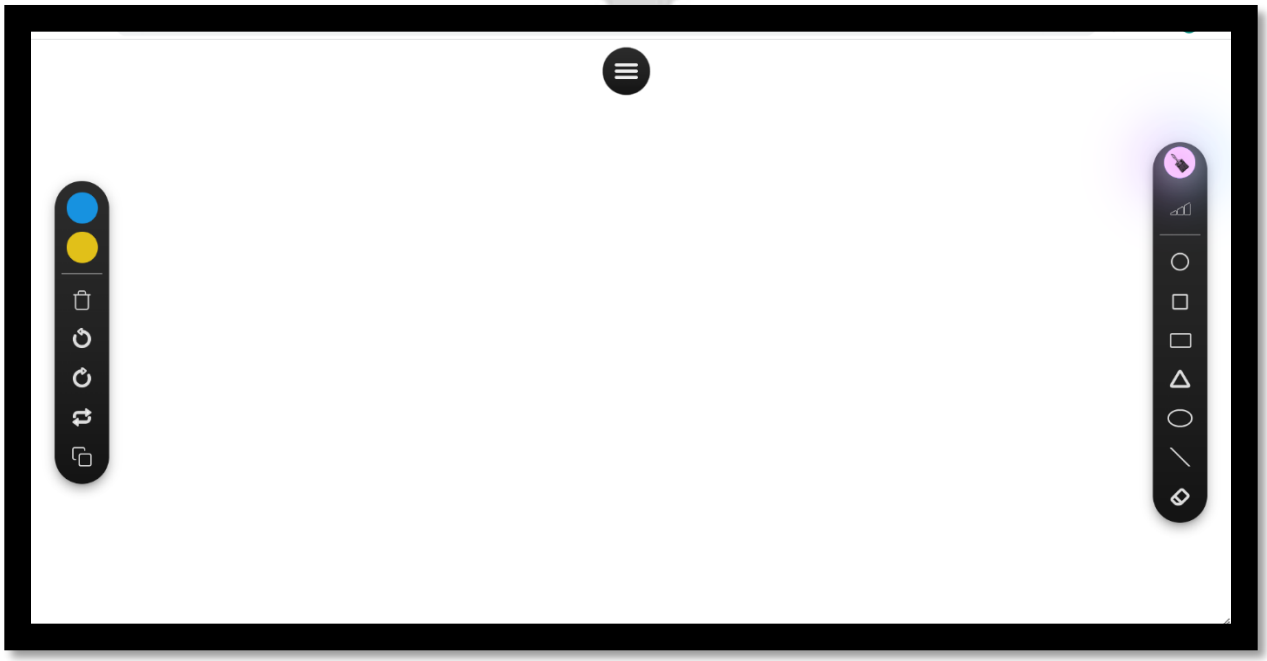
- **Used Konva library for drawing.**
- **Implemented Command DP to handle commands execution.**
- **Implemented the main design patterns in back-end.**
- **Can't do redo after execution of any command.**
- **Can't do Undo & Redo for freehand drawings.**
- **Save files are generated and loaded in the backend.**
- **Only one save file can be generated and loaded at a time.**

5.1 UI snippets

I Recommend for you to have a look on 5-min demo video on

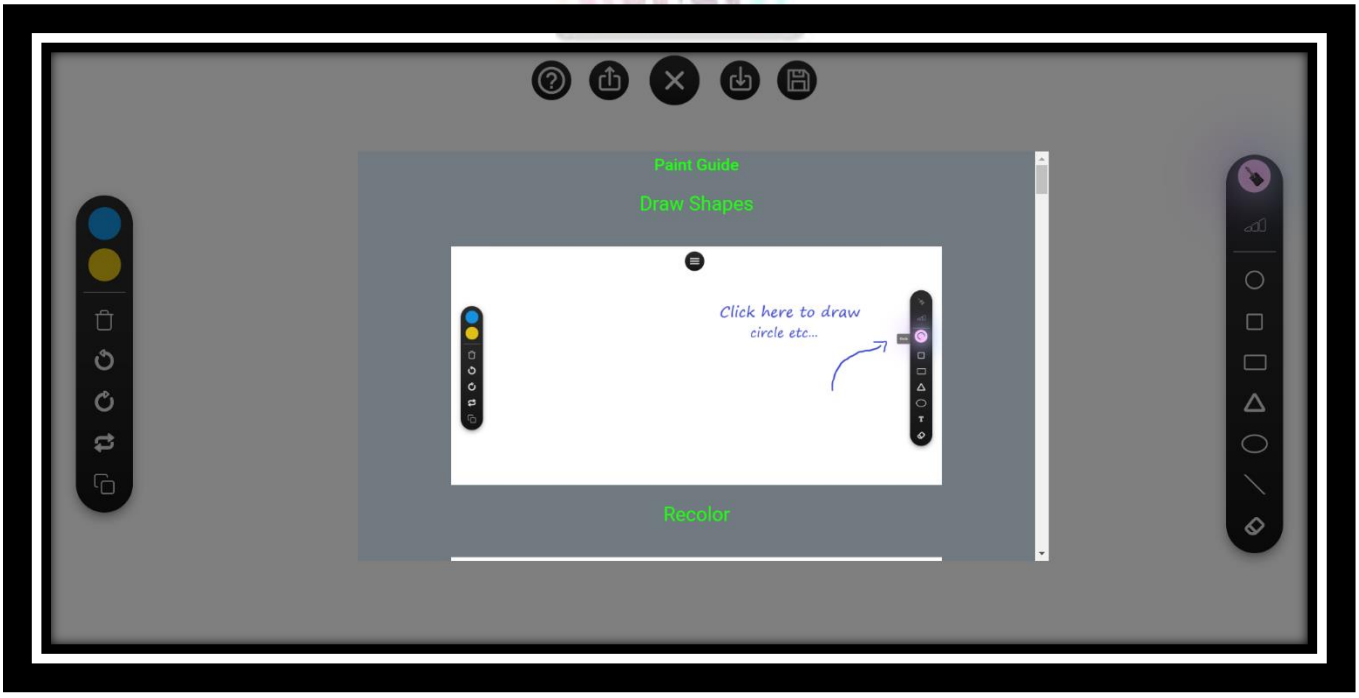
6.1 Demo Video

to get the whole right experience but this is a quick view of UI.



5.2 User Guide snippets

We uploaded a full user guide showing full steps to do any thing by photos to be clearer.



6.1 Demo Video

You are welcomed to have a look on our demo website, We hope you have a good experience.

- **Link of the video on Google Drive :**

  Click here to redirect to the video

That's it If you want to know more about authors of this documentation go through

7 Authors

Thank you ☺

7 Authors

- **Who we are ?**

  We are a team of Alexandria University of Computers and Systems Engineering department.

- **What is this ?**

  This is a programming assignment of design pattern course that we are asked to implement a full paint website.

7.1 Accounts

- **Marwan Essam Rashad**

  1. GitHub link.

  2. LinkedIn.

- **Osama Belal**

  1. GitHub link.

  2. LinkedIn.

- Joseph Shokry

  1. GitHub link.

  2. LinkedIn.

- Omar Tammam

  1. GitHub link.

  2. LinkedIn.