Title: Report on “DoodleHub: A Real-Time Collaborative Drawing and Doodle Management Platform”

Author: Surbhi Sharma
Date: 27 November 2025
Institution: GLA University

2. Executive Summary / Abstract

This report presents *DoodleHub*, a real-time collaborative drawing platform that allows multiple users to draw together on a shared canvas. The project integrates modern full-stack technologies such as React.js, Express.js, MongoDB, and Socket.io to enable synchronized drawing, stroke-based rendering, authentication, paginated doodle galleries, and persistent room states. The objective of the project was to design an interactive digital environment that supports real-time creativity along with backend persistence and user-specific doodle management.

This report discusses the architecture, data flow, system components, technical implementation, and insights gained during the development of DoodleHub.

3. Introduction

3.1 Background

Collaborative drawing tools have become increasingly popular, enabling digital creativity, brainstorming, and teamwork. Such applications require real-time communication, synchronized states, and interactive UIs. DoodleHub was developed as a hands-on project to understand real-time web architectures and enhance skills in frontend, backend, and socket-based event handling.

3.2 Purpose

DoodleHub aims to provide users with a platform where they can:

- Draw collaboratively in real time
- Create or join private rooms
- Save their artworks to a personal gallery
- Access paginated doodles
- Manage their accounts securely

3.3 Objectives

- To implement real-time drawing using WebSockets
- To design a stroke-based canvas rendering architecture
- To integrate user authentication via JWT
- To save doodles and implement gallery pagination
- To maintain persistent room state for collaborative sessions

4. Body/Discussion

4.1 System Overview

DoodleHub is a full-stack web application with:

Frontend: React.js

Backend: Express.js + MongoDB

Real-Time Engine: Socket.io (WebSockets)

Authentication: JWT + bcrypt

Storage: MongoDB (Users, Doodles, RoomState)

The platform allows multiple users to draw simultaneously on a shared canvas, with every stroke synced across clients.

4.2 Core Concept: Strokes

The heart of DoodleHub's canvas is **stroke-based architecture**.

A *stroke* represents one continuous line drawn by the user:


```
{
  "id": "unique-stroke-id",
  "color": "black",
  "width": 4,
  "points": [{ "x": 112, "y": 210 }, { "x": 120, "y": 214 }]
}
```


Why strokes?

- Simple undo/redo
- Efficient storage
- Fast synchronization
- Easy replay when users join
- Ideal for multi-user collaboration

4.3 Frontend Implementation

4.3.1 Canvas Rendering

- Built using HTML5 <canvas> API
- Pointer events (pointerdown, pointermove, pointerup) capture drawing motions
- Each movement is recorded as points → grouped as strokes

4.3.2 Real-Time Sync

When the user draws:

1. Stroke is created locally
2. Sent to backend via Socket.io
3. Rendered on all other clients instantly

4.3.3 User Tools

- Undo / Redo
- Brush color and width
- Clear canvas
- Live cursors of other participants

4.3.4 Exporting & Saving

Canvas is converted to a **base64 image** and stored in the database as a doodle.

4.4 Backend Implementation

4.4.1 Authentication (JWT)

- Signup/Login
- Password hashed using bcrypt
- Token stored on frontend
- Protected routes require “Authorization: Bearer <token>”

4.4.2 Doodles API

- Save doodle
- Fetch user doodles with pagination
- Delete doodle
- Update title

Pagination logic:

page = req.query.page

limit = req.query.limit

skip = (page - 1) * limit

4.4.3 RoomState Persistence

Each room stores:

- strokes
- version number
- timestamps
- participants list

This ensures that when a user joins a room, they see the full drawing history.

4.5 Real-Time Collaboration Using Socket.io

4.5.1 Joining Rooms

Users join specific rooms via a unique room ID, enabling separate collaborative sessions.

4.5.2 Event Communication

Socket.io handles events like:

- stroke:create – broadcast new stroke
- stroke:undo – sync undo action
- canvas:clear – clear canvas for all users
- cursor:move – show other users' cursors
- participants:update – live online users list

4.5.3 Server Responsibilities

- Store incoming strokes in the database
- Broadcast strokes to connected clients
- Maintain participant maps
- Send full room state to newly joined users

4.5.4 Persistence

If the server restarts, users do not lose drawing data because **RoomState** is stored in MongoDB.

4.6 Gallery + Pagination

The gallery displays user-saved doodles with pagination:

Backend:

- Returns paginated results
- Includes total count
- Calculates total pages

Frontend:

- Renders doodles in a responsive grid
- Allows page switching
- Uses caching for performance

4.7 Data Flow Summary

Drawing

User draws → stroke created → sent via socket → saved → broadcast → others render

Saving Doodles

Export canvas → send image to backend → stored in DB → appears in gallery

Fetching Gallery

Frontend requests page n → backend returns results → grid render

5. Conclusion

DoodleHub successfully provides a real-time collaborative drawing environment that integrates advanced concepts such as WebSocket communication, stroke-based rendering, backend persistence, authentication, and paginated content delivery.
The project demonstrates a strong understanding of both frontend and backend architecture, efficient state synchronization, and the ability to design interactive user experiences.
It also highlights important engineering concepts such as event-driven communication, persistent room states, token-based security, and scalable API development.

6. Recommendations

To enhance and scale DoodleHub:

- Implement voice chat inside rooms
- Add layers, color palettes, shape tools
- Introduce replay mode to watch drawings animate
- Add collaboration cursors with user names
- Deploy using Docker + CI/CD pipeline
- Add image-to-doodle classification using a small ML model
- Add collaborative note-taking or whiteboarding modes

7. References

- Socket.io Documentation
- React.js official documentation
- MongoDB Developer Docs
- Express.js API Reference
- HTML5 Canvas API
- JWT.IO
- Tutorials on WebSocket communication and state synchronization