Collaborative Whiteboard Development Assignment

The deadline for this task is 7/21/2025 11:59:59 am

Project Overview

Develop a real-time collaborative whiteboard application using the MERN stack (MongoDB, Express.js, React.js, Node.js) with WebSocket support for live collaboration.

Core Requirements

1. Technology Stack

- **Frontend**: React.js
- **Backend**: Node.js with Express.js
- **Database**: MongoDB
- **Real-time Communication**: Socket.io
- **Styling**: CSS/Styled Components (your choice)

2. Functional Requirements

Room Management

- Users can join a whiteboard room by entering a room code
- No authentication or user registration required
- Room codes should be simple (alphanumeric, 6-8 characters)
- Create new rooms dynamically when a non-existing room code is entered

Drawing Functionality

- **Single Tool**: Pencil/pen tool only
- **Drawing Features**:
    - Smooth drawing lines

    - Adjustable stroke width (simple slider)
    - Basic color selection (black, red, blue, green)
    - Clear canvas option
- **Canvas**: HTML5 Canvas element for drawing

Live Collaboration Features

- **Real-time Cursor Tracking**: Show all connected users' cursor positions in real-time
- **Live Drawing Sync**: All drawing actions should be synchronized across all connected users instantly
- **User Presence**: Display number of active users in the room

3. Technical Specifications

Frontend (React.js)

```
Components Structure:
├── App.js
├── components/
│   ├── RoomJoin.js          // Room code input
│   ├── Whiteboard.js        // Main whiteboard component
│   ├── DrawingCanvas.js     // Canvas drawing logic
│   ├── Toolbar.js           // Simple drawing controls
│   └── UserCursors.js       // Display other users' cursors
```

Backend (Node.js + Express)

```
API Endpoints:
- POST /api/rooms/join      // Join/create room
- GET /api/rooms/:roomId    // Get room info

Socket Events:
- 'join-room'               // User joins room
- 'leave-room'              // User leaves room
- 'cursor-move'             // Cursor position update
- 'draw-start'              // Start drawing stroke
- 'draw-move'               // Drawing path data
- 'draw-end'                // End drawing stroke
- 'clear-canvas'            // Clear entire canvas
```

Database Schema (MongoDB)

javascript

```javascript
// Room Schema
{
  roomId: String (unique),
  createdAt: Date,
  lastActivity: Date,
  drawingData: Array // Store drawing commands for persistence
}

// Drawing Command Schema
{
  type: String, // 'stroke', 'clear'
  data: Object, // Contains path data, color, width, etc.
  timestamp: Date

}
```

4. Implementation Details

Drawing Synchronization

- Capture mouse/touch events on canvas
- Send drawing data as small incremental updates (not entire canvas)
- Implement efficient data structures for path storage
- Ensure smooth rendering across all connected clients

Cursor Tracking

- Track mouse position over entire whiteboard area
- Send cursor coordinates at reasonable intervals (throttled to ~60fps)
- Display cursors with different colors for each user
- Hide cursor when user is inactive

Performance Considerations

- Throttle cursor position updates to prevent overwhelming the server
- Implement drawing data compression for large canvases
- Clean up old room data (rooms inactive for 24+ hours)

5. User Experience Requirements

Interface Design

- Clean, minimal interface focused on the whiteboard
- Room code input should be prominent and easy to use
- Toolbar should be simple and unobtrusive
- Responsive design for both desktop and tablet use

Real-time Feedback

- Immediate visual feedback for all drawing actions
- Smooth cursor movement animations
- Connection status indicator
- User count display

6. Deliverables

Code Structure

```
project-root/
├── client/                 // React frontend
│   ├── src/
│   ├── public/
│   └── package.json
├── server/                 // Node.js backend
│   ├── models/
│   ├── routes/
│   ├── socket/
│   └── server.js
├── README.md
└── package.json
```

Documentation Required

1. **Setup Instructions**: How to install and run the application
2. **API Documentation**: Socket events and REST endpoints
3. **Architecture Overview**: High-level system design
4. **Deployment Guide**: Instructions for production deployment