Low-Level Design (LLD) for Real-Time Collaborative Whiteboard

Components Overview

1. **Frontend (ReactJS):**
    - **Authentication**: User login and registration using JWT.
    - **Whiteboard UI**: Canvas for drawing, with tools for erase, color, and size.
    - **WebSocket Client**: Real-time communication with the backend using `socket.io-client`.
    - **Session Management**: Join or create a new session.
    - **User Presence**: Display active users in the current session.
2. **Backend (Node.js, Express, Socket.io):**
    - **Authentication**: JWT-based authentication for secure API access.
    - **WebSocket Server**: Handle real-time updates and manage connected users.
    - **Session Management**: Create, join, and manage whiteboard sessions.
    - **State Management**: Store session and whiteboard state in MongoDB for persistence.
3. **Database (MongoDB):**
    - **User Collection**: Store user credentials and profile data.
    - **Session Collection**: Store session metadata and whiteboard state.

Detailed Design

1. Frontend Design

React Components:

- **Login/Register Component**: Handles user authentication.
- **Dashboard Component**: Allows users to create or join sessions.
- **Whiteboard Component**: Implements the drawing canvas and WebSocket logic.

Frontend Workflow:

1. **Login/Register**:
    - POST request to `/auth/login` or `/auth/register`.
    - Store JWT token in local storage.
    - Redirect to the dashboard upon successful login.
2. **Session Management**:
    - Create a new session: POST request to `/api/session`.
    - Join an existing session: Navigate to `/whiteboard/:sessionId`.
3. **Whiteboard**:

    - Initialize WebSocket connection with `socket.io-client`.
    - Emit `drawing` events on user interactions.
    - Listen for real-time updates from the server and render changes.

---

2. Backend Design

Routes:

- **Authentication**:
    - `POST /auth/register`: Register a new user.
    - `POST /auth/login`: Authenticate user and return a JWT.
- **Session Management**:
    - `POST /api/session`: Create a new session.
    - `GET /api/session/:id`: Fetch session details, including whiteboard state.

WebSocket Events:

- **Connection**: Establish connection with the client.
- **Session Events**:
    - `joinSession`: User joins a session.
    - `userJoined`: Notify other users about the new participant.
    - `userLeft`: Notify other users about a participant leaving.
- **Whiteboard Events**:
    - `drawing`: Broadcast drawing updates to all clients.
    - `canvasState`: Sync the current canvas state with new users.

---

3. Database Design

User Collection:

```
{
  "_id": "ObjectId",
  "email": "string",
  "password": "hashed_password"
}
```

Session Collection:

```
{
  "_id": "ObjectId",
  "sessionId": "string",
  "canvasState": [
    {
      "type": "line|erase",
      "color": "string",
      "size": "number",
      "points": [{ "x": "number", "y": "number" }]
    }
  ],
  "users": ["userId1", "userId2", ...]
}
```

---

4. WebSocket Flow

Event Flow:

1. **User Joins a Session**:
   - Client emits `joinSession` with `sessionId` and `userId`.
   - Server adds the user to the session room and broadcasts `userJoined`.
2. **Real-Time Drawing**:
   - Client emits `drawing` with data about the drawn shape (e.g., color, points).
   - Server broadcasts the `drawing` event to all other clients in the session.
3. **Canvas State Synchronization**:
   - Server saves canvas updates in MongoDB.
   - When a new user joins, the server sends the current `canvasState` to the client.

---

5. APIs

Endpoint	Method	Description
`/auth/register`	POST	Register a new user.
`/auth/login`	POST	Authenticate a user and return a JWT.
`/api/session`	POST	Create a new session.
`/api/session/:id`	GET	Fetch session details (e.g., whiteboard state).

---

6. WebSocket Server Design

WebSocket Event Handlers:

```
io.on('connection', (socket) => {
  // Join Session
  socket.on('joinSession', ({ sessionId, userId }) => {
    socket.join(sessionId);
    socket.to(sessionId).emit('userJoined', { userId });
  });
  // Handle Drawing
  socket.on('drawing', ({ sessionId, drawingData }) => {
    socket.to(sessionId).emit('drawing', drawingData);
    // Optional: Save canvas state in the database
    saveCanvasState(sessionId, drawingData);
  });

  // User Disconnection
  socket.on('disconnect', () => {
    // Notify other users in the session
    io.emit('userLeft', { userId: socket.id });
  });
});
```

---

7. Authentication

JWT Workflow:

1. **User Login**:
    - Generate a JWT using `jsonwebtoken` after successful authentication.
    - Send the JWT to the client for subsequent requests.
2. **Protected Routes**:
    - Middleware verifies the JWT before granting access.

---

8. Whiteboard Drawing Logic

Frontend (React):

- Use `react-canvas-draw` or implement a custom canvas with `<canvas>` API.
- Emit WebSocket events for `mousemove` and `mousedown` to broadcast drawing actions.

Backend (Socket.io):

- Broadcast drawing data (e.g., points, color, size) to all users in the session.
- Persist canvas state in MongoDB for session synchronization.

---

9. Bonus Features

- **User Presence**:
    - Track connected users in a session and display a list in the frontend.
- **Canvas Persistence**:
    - Save and load the canvas state from MongoDB to allow session resumption.

This low-level design ensures scalability, real-time performance, and user session management for a collaborative whiteboard application.