Prakhar Srivastava: User Authentication & Permissions Management

Step 1: Research

- Investigate **OAuth** and **JWT** to determine which authentication mechanism fits your project.
    - **OAuth** for third-party login options like Google or Facebook.
    - **JWT (JSON Web Tokens)** for token-based authentication to avoid storing session data on the server.

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Create registration and login forms using React components. Include form validation (e.g., email validation).
    - Design screens for permissions management, allowing different roles (e.g., admin, viewer) to be assigned.

Step 3: Backend Implementation

- **Flask & MongoDB**:
    - Set up endpoints in Flask to handle registration and login (e.g., /register, /login).
    - Implement JWT in Flask to create tokens upon successful login, which will be stored on the frontend.
    - Store user data in MongoDB, including roles for permissions management.

Step 4: Frontend Integration

- **React**: On successful registration or login, store the received JWT token in localStorage or sessionStorage.
- Implement conditional rendering in React based on user roles for permissions (e.g., admins see different screens).

Priyanshu Yadav: Real-Time Drawing & Editing Tools

Step 1: Research

- Explore **Fabric.js** or **Konva.js** for drawing tools, and use **WebSockets** for real-time syncing.

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Create a drawing workspace with a canvas where users can draw lines, shapes, etc.
    - Include basic toolbars for selecting drawing tools, colors, and erasers.

Step 3: Backend Implementation

- **Flask-SocketIO & MongoDB**:
    - Set up WebSocket connections in Flask to sync real-time changes between users.
    - Store drawing session data (e.g., drawings, shapes) in MongoDB.

Step 4: Frontend Integration

- **React**:
    - Implement WebSocket communication in React to receive and broadcast changes in real-time.
    - Ensure the canvas updates instantly when another user draws something.

Abhay Giri: Live Cursor & Presence Awareness

Step 1: Research

- Look into **Yjs** or **ShareDB** for handling live presence tracking and cursor updates using WebSockets.

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Design UI to display the position of other users' cursors on the canvas or workspace.
    - Include indicators to show which users are currently active in the session.

Step 3: Backend Implementation

- **Flask-SocketIO**:
    - Broadcast each user’s cursor movement in real-time using WebSocket channels.
    - Store active user information and session details in MongoDB.

Step 4: Frontend Integration

- **React**:
    - Track and broadcast the cursor’s position through WebSocket, rendering live positions of other users on the canvas in real-time.

Ananya Singh: Chat & Collaboration Tools

Step 1: Research

- Explore real-time chat technologies, like **Socket.IO** for WebSocket-based messaging.

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Design a chat interface within the workspace, allowing users to send and receive messages.
    - Include message timestamps and user information.

Step 3: Backend Implementation

- **Flask-SocketIO & MongoDB**:
    - Implement a WebSocket channel to handle chat messages.
    - Store chat messages in MongoDB for persistence and future reference.

Step 4: Frontend Integration

- **React**:
    - Set up real-time chat functionality with WebSocket communication. Incoming messages should be displayed instantly in the chat interface.

Jyotika Kishor: Version Control & Undo/Redo Functionality

Step 1: Research

- Investigate **Operational Transformation (OT)** and **Conflict-Free Replicated Data Types (CRDT)** for handling real-time version control and undo/redo features.

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Design a toolbar with undo/redo buttons for the drawing workspace.
    - Indicate the history of changes that can be undone or redone.

Step 3: Backend Implementation

- **Flask & MongoDB**:
    - Implement endpoints in Flask to track the history of changes in MongoDB.
    - Store versions of documents or drawings, so users can undo/redo operations.

Step 4: Frontend Integration

- **React**:
    - Use WebSocket or API calls to apply undo/redo actions across all users in real-time.

Anmeha Pandey: File & Image Upload System

Step 1: Research

- Study best practices for file uploads using **Flask** and explore cloud storage options (e.g., AWS S3).

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Create a file upload form with drag-and-drop support for images or files.
    - Preview uploaded images before submission.

Step 3: Backend Implementation

- **Flask & MongoDB**:
    - Use Flask to handle multipart file uploads and store file metadata in MongoDB.
    - Optionally, use a cloud service like AWS S3 to store the actual files.

Step 4: Frontend Integration

- **React**:
    - Once the file is uploaded, display it in the workspace, ensuring it's accessible to all active users.

Rohit Singh: Advanced Collaborative Tools & Widgets

Step 1: Research

- Explore advanced real-time widgets (e.g., sticky notes, checklists) by studying tools like **Trello** or **Miro**.

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Design UI components for these tools (e.g., sticky notes, text boxes).
    - Include functionality to add, edit, and delete these widgets.

Step 3: Backend Implementation

- **Flask-SocketIO & MongoDB**:
    - Use WebSocket to sync these widgets across all users.
    - Store widget data in MongoDB so that users can load it in future sessions.

Step 4: Frontend Integration

- **React**:
    - Update the workspace in real-time as users add or modify these widgets.

Khem Chand: External Application Integration

Step 1: Research

- Investigate API integration techniques, OAuth flows, and third-party app connections (e.g., Google Drive, Slack).

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Provide options for users to connect external apps, like linking their Google Drive or Slack accounts.

Step 3: Backend Implementation

- **Flask**:
    - Use OAuth in Flask to authorize third-party apps.
    - Set up routes for API communication with external services.

Step 4: Frontend Integration

- **React**:
    - After successful OAuth integration, display external resources (e.g., files from Google Drive) within the workspace.

Rahul Gupta: Customizable Templates & Design Themes

Step 1: Research

- Explore tools like **Theme-UI** or **Tailwind CSS** for enabling theme customization.

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Create a settings panel for users to choose from different templates or themes.
    - Allow users to customize colors, fonts, and layouts.

Step 3: Backend Implementation

- **Flask & MongoDB**:
    - Store user preferences (themes, templates) in MongoDB.
    - Load user preferences upon login using Flask.

Step 4: Frontend Integration

- **React**:
    - Use React state management to dynamically switch themes and templates in real-time.

Upendra Sisodiya: Security & Privacy Controls

Step 1: Research

- Investigate WebSocket security, HTTPS, database encryption, and Flask best practices for secure authentication and data management.

Step 2: Design Foundational Screens

- **Frontend (React)**:
    - Create UI for managing security settings, such as toggling two-factor authentication (2FA) and setting permissions.

Step 3: Backend Implementation

- **Flask**:
    - Implement security mechanisms, including token validation (JWT), HTTPS, and encrypted communication.
    - Store sensitive user data (e.g., passwords) in an encrypted format in MongoDB.

Step 4: Frontend Integration

- **React**:
    - Ensure security controls are applied correctly in the user interface, based on the permissions and settings.