M4 : Real-Time Collaborative Digital Canvas and Creative Workspace

Problem Statement:

This project involves building a real-time collaborative digital drawing and sketching workspace that allows multiple remote users to co-create and interact with visual content simultaneously on a shared canvas. The goal is not to produce a basic offline paint application but to design a distributed, low-latency synchronization environment that supports multi-user editing with strong guarantees of consistency, responsiveness, and usability. Users should be able to draw, annotate, erase, insert shapes and text, manipulate layers, and collaboratively iterate on artwork, diagrams, or design concepts while observing smooth live updates from other collaborators regardless of network variation or geographic distance.
To support complex concurrent editing, students are encouraged to explore advanced real-time synchronization technologies such as Operational Transformation (OT), Conflict-Free Replicated Data Types (CRDTs), WebSocket-based messaging, or distributed state reconciliation strategies. Technical extensions may include timeline replay of drawing history, gesture recognition, AI-assisted enhancements such as stroke smoothing or auto-segmentation, version control and rollback support, or extensible plugins for exporting artwork to external media. The platform may include moderation controls, permission-based access for shared sessions, distributed storage for persistent canvas states, and protection against vandalism or accidental overwriting. This project requires balancing distributed systems engineering, HCI considerations, and performance optimization, demonstrating thoughtful innovation beyond simple GUI-based drawing tools.

User Stories:

Epic 1: Session Management & Access Control

1.1. Create Session

- **User Story:** As a User, I want to generate a unique, shareable URL for a new canvas, **so that** I can start a private collaboration session.
- **Tasks:**
    1. Implement API endpoint (POST /api/session) to generate unique UUIDs.
    2. Configure client-side routing to handle dynamic /board/:id URLs.
    3. Create a "Start New Whiteboard" landing page button.

1.2. Join via URL

- **User Story:** As a User, I want to load an existing canvas by pasting a URL, so that I can immediately join my team's workspace.
- **Tasks:**
    1. Create API endpoint (GET /api/session/:id) to validate session existence.
    2. Implement loading state/spinner while fetching session data.

3. Handle "404 Session Not Found" errors appropriately.

1.3. User Identification

- **User Story:** As a User, I want to set a display name when joining, so that my collaborators know who I am.
- **Tasks:**
  1. Create a modal/input prompt for "Enter Name" on entry.
  2. Store the nickname in localStorage for persistence.
  3. Update the WebSocket connection handshake to include user metadata.

1.4. Presence Badge

- **User Story:** As a User, I want to see a counter showing how many people are online, so that I know if the team is present.
- **Tasks:**
  1. Maintain a set of active socket connections per room ID.
  2. Broadcast "user_joined" and "user_left" events to all clients.
  3. Create a UI Badge component in the header displaying the count.

1.5. Read-Only Mode (Permissions)

- **User Story:** As a User managing the session, I want to lock the board to "Read-Only" for guests, so that accidental edits are prevented during a review.
- **Tasks:**
  1. Add a is_locked boolean flag to the Session database model.
  2. Disable all toolbar inputs if is_locked state is true.
  3. Reject any write operations (POST/PUT) from clients if the session is locked.

Epic 2: Core Drawing Tools

2.1. Freehand Drawing

- **User Story:** As a User, I want to draw freehand lines with adjustable brush sizes, so that I can sketch ideas naturally.
- **Tasks:**
  1. Implement pointer event handling for modifying points in an HTML5 Canvas or SVG, adapted for React/Next.js using the framework’s event system.
  2. Develop a smoothing algorithm (e.g., Catmull-Rom spline) to reduce jitter.
  3. Create a UI slider component for selecting brush stroke width.

2.2. Standard Shapes

- **User Story:** As a User, I want to insert geometric shapes (Rectangles, Circles), so that I can create structured diagrams.
- **Tasks:**
  1. Define a JSON data structure for shapes.
  2. Implement "drag-to-create" logic for calculating shape dimensions.

3. Render SVG elements dynamically based on the shape data array.

2.3. Text Annotation

- **User Story:** As a User, I want to add text labels to the canvas, so that I can explain my diagrams.
- **Tasks:**
  1. Implement a contentEditable overlay or Input field that appears on click.
  2. Sanitize text input to prevent XSS injection (Security requirement).
  3. Render final text as SVG <text> elements for scalability.

2.4. Eraser Tool

- **User Story:** As a User, I want to remove specific objects or line segments, so that I can correct mistakes.
- **Tasks:**
  1. Implement "Hit Testing" logic to detect which object is under the cursor.
  2. Add logic to remove the identified object ID from the local state array.
  3. Create an API/Socket event for DELETE_OBJECT.

2.5. Color Selection

- **User Story:** As a User, I want to choose different colors for my strokes and fills, so that my designs are visually distinct.
- **Tasks:**
  1. Integrate a color picker library or create a preset palette UI.
  2. Update the current "context" state with the selected hex code.
  3. Apply the selected color property to new SVG/Canvas elements.

Epic 3: Real-Time Synchronization Engine

3.1. Live Cursors

- **User Story:** As a User, I want to see the cursors of other users moving in real-time, so that I avoid editing the same area they are working on.
- **Tasks:**
  1. Throttle mousemove events (e.g., every 50ms) to reduce network load.
  2. Broadcast cursor coordinates (X, Y, UserID) to all other clients in the room.
  3. Render remote cursors with user-specific name labels.

3.2. Instant Object Updates

- **User Story:** As a User, I want objects added by others to appear on my screen instantly without refreshing, so that collaboration feels seamless.
- **Tasks:**
  1. Establish a WebSocket connection (e.g., Socket.io or native WS) upon joining.

2. Route incoming "draw_event" messages to all connected clients except the sender.
3. Update the Redux/State store immediately upon receiving a socket message.

3.3. Concurrent Editing (Conflict Resolution)

- **User Story:** As a User, I want the system to handle simultaneous edits gracefully, so that the board state remains consistent for everyone.
- **Tasks:**
    1. Select a synchronization strategy (CRDT like Yjs or Operational Transformation).
    2. Implement the conflict resolution logic on the server (or relay CRDT updates).
    3. Implement unique ID generation for every object to prevent ID collisions.

3.4. Network Optimization

- **User Story:** As a User, I want the drawing experience to remain smooth even if the network is slightly slow, so that my creative flow isn't interrupted.
- **Tasks:**
    1. Implement "Optimistic UI" updates (draw immediately, sync later).
    2. Implement message batching (grouping multiple updates into one packet).
    3. Add visual indicators (e.g., greyed out) for unsynced changes.

3.5. State Reconciliation

- **User Story:** As a User, I want to ensure my canvas looks exactly like everyone else's, so that there are no "ghost" objects.
- **Tasks:**
    1. Maintain a canonical "Master State" in memory (e.g., Redis).
    2. Periodically request a checksum of the state to verify consistency.
    3. Logic to re-fetch full state if checksum mismatch occurs.

Epic 4: Object Manipulation & Layout

4.1. Selection Tool

- **User Story:** As a User, I want to click to select one or multiple objects, so that I can modify them as a group.
- **Tasks:**
    1. Implement a "bounding box" calculation for selected items.
    2. Add visual handles (corners) to indicate selection state.
    3. Implement "Shift+Click" logic for multi-selection.

4.2. Move and Translate

- **User Story:** As a User, I want to drag objects to new positions, so that I can rearrange the layout.
- **Tasks:**

    1. Calculate delta (change in X/Y) during drag events.
    2. Update object coordinates locally in real-time.
    3. Broadcast final position update on mouseup (to save bandwidth).

4.3. Resize and Rotate

- **User Story:** As a User, I want to resize or rotate objects using handles, **so that** I can fit them into specific areas.
- **Tasks:**
    1. Implement transformation matrix logic for rotation.
    2. Lock aspect ratio when resizing with "Shift" key held.
    3. Update object properties (width, height, rotation_angle) in the DB.

4.4. Z-Index (Layers)

- **User Story:** As a User, I want to bring objects to the front or send them to the back, **so that** I can layer elements correctly.
- **Tasks:**
    1. Implement "Bring Forward" / "Send Backward" buttons.
    2. Reorder the object array based on the requested action.
    3. Broadcast the new array index order to peers.

4.5. Undo/Redo

- **User Story:** As a User, I want to undo my last action, **so that** I can fix mistakes immediately.
- **Tasks:**
    1. Implement a Command Pattern or Stack to track user actions.
    2. Logic to pop the last action and invert it.
    3. Ensure "Undo" only affects the specific user's actions v , not others'.

Epic 5: Canvas Navigation & Viewport

5.1. Infinite Panning

- **User Story:** As a User, I want to scroll in any direction to find new space, **so that** I am not restricted by screen size.
- **Tasks:**
    1. Implement a virtual coordinate system separate from screen pixels.
    2. Listen for scroll wheel or "Space+Drag" events to update Viewport Offset.
    3. Render only objects within the visible viewport (culling) for performance.

5.2. Zooming

- **User Story:** As a User, I want to zoom in for detail and zoom out for an overview, **so that** I can work at different levels of granularity.
- **Tasks:**
    1. Listen for Ctrl+Scroll or pinch gestures.

2. Apply CSS transform: scale() or Canvas scale logic.
3. Display the current zoom percentage (e.g., 150%) in the UI.

5.3. Mini-Map

- **User Story:** As a User, I want to see a mini-map in the corner, **so that** I can quickly navigate to different parts of a large diagram.
- **Tasks:**
  1. Create a secondary, smaller canvas overlay.
  2. Render a simplified representation of all objects on the mini-map.
  3. Create a draggable "View Box" on the mini-map that controls the main viewport.

5.4. Return to Center

- **User Story:** As a User, I want a button to reset my view to the center, **so that** I don't get lost in the infinite space.
- **Tasks:**
  1. Create a "Recenter" floating action button.
  2. Function to animate Viewport Offset (X,Y) back to (0,0).
  3. Reset Zoom level to 100%.

5.5. Grid Snapping

- **User Story:** As a User, I want a background grid that objects snap to, **so that** my diagrams are neatly aligned.
- **Tasks:**
  1. Render a grid pattern (CSS background or Canvas lines).
  2. Implement math logic to round object coordinates to nearest grid interval (e.g., nearest 10px).
  3. Add a toggle in settings to enable/disable "Snap to Grid."

Epic 6: aDta Persistence & Export

6.1. Auto-Save

- **User Story:** As a User, I want my drawings to be saved automatically, **so that** I don't lose work if my browser crashes.
- **Tasks:**
  1. Implement a debounce function to trigger save 2 seconds after last edit.
  2. Write current board state to persistent database (MongoDB/Postgres).
  3. Show "Saving..." vs "Saved" status indicator.

6.2. Export to Image

- **User Story:** As a User, I want to download the canvas as a PNG or JPG, **so that** I can share it in reports.
- **Tasks:**

    1. Use a library (like html2canvas or native API) to rasterize the canvas.
    2. Trigger a browser download of the Blob object.
    3. Allow user to select "Whole Board" or "Current View" for export.

6.3. Export to SVG

- **User Story:** As a User, I want to export as SVG, **so that** the drawing remains scalable for high-quality printing.
- **Tasks:**
    1. Serialize the internal shape state into a valid XML SVG string.
    2. Handle cleaning up internal metadata (IDs, user names) before export.
    3. Trigger file download with .svg extension.

6.4. Export Canvas to PDF

- **User Story:** As a User, I want to export the canvas as a PDF, so that I can include high-quality, uneditable versions of my diagrams in external reports or presentations.
- **Tasks:**
    1. Research and integrate a client-side PDF generation library (e.g., jsPDF or pdf-lib).
    2. Implement logic to calculate the "Bounding Box" of all drawn elements to determine the PDF page size.
    3. Write a function to serialize the canvas state (SVG/Canvas) into the PDF document context.
    4. Add a "Download PDF" option to the Export dropdown menu in the UI.

6.5. Clear Canvas

- **User Story:** As a User, I want to clear the entire board with one click, **so that** I can start fresh without creating a new session.
- **Tasks:**
    1. Create "Clear All" button with a confirmation modal (prevention).
    2. API endpoint to delete all objects associated with the session ID.
    3. Broadcast "Clear" event so all users see the board wipe instantly.

Epic 7: Offline Connectivity & Resilience

7.1. Offline Detection

- **User Story:** As a User, I want to be notified if my internet connection drops, **so that** I know my edits aren't being synced.
- **Tasks:**
    1. Add event listeners for window.addEventListener('offline').
    2. Display a "Connection Lost - Working Offline" toast notification.
    3. Change the UI theme or header color to indicate offline mode.

7.2. Offline Editing (Queue)

- **User Story:** As a User, I want to continue drawing while offline, **so that** my work is not interrupted by connection blips.
- **Tasks:**
    1. When offline, push all create/update actions to a local queue (Array).
    2. Store this queue in IndexedDB or LocalStorage to survive refresh.
    3. Prevent actions that strictly require server (e.g., uploading images).

7.3. Auto-Reconnect & Sync

- **User Story:** As a User, I want my offline edits to sync automatically when I come back online, **so that** I don't lose work.
- **Tasks:**
    1. Listen for window.addEventListener('online').
    2. Iterate through the local queue and send batched updates to the server.
    3. Process the batch and resolve any timestamp conflicts.

7.4. Server Disconnect Handling

- **User Story:** As a User, I want the application to auto-reconnect if the server restarts, **so that** I stay in the session.
- **Tasks:**
    1. Implement exponential backoff strategy for WebSocket reconnection attempts.
    2. Reload the latest board state from server upon successful reconnection.
    3. Compare local state vs server state to find missing items.

7.5. Frame/Group Management

- **User Story:** As a User, I want to group items into a "Frame" so that I can organize content logically.
- **Tasks:**
    1. Implement a "Frame" object type that acts as a container.
    2. Logic to treat children of the frame as relative coordinates to the frame.
    3. Allow selecting the Frame to move all children at once.

Epic 8: Advanced Creative Tools (Fast Follow)

8.1. Timeline Replay

- **User Story:** As a User, I want to replay the history of the drawing session, **so that** I can understand the evolution of ideas.
- **Tasks:**
    1. Store all events with timestamps in a separate "History" table/collection.
    2. Create a timeline slider UI control.
    3. Implement a "playback engine" that reconstructs the canvas state up to the selected timestamp.

8.2. Stroke Smoothing using AI

- **User Story:** As a User, I want my rough hand-drawn shapes to be automatically smoothed, **so that** my diagrams look professional.
- **Tasks:**
    1. Implement a heuristic algorithm to detect if a stroke resembles a line or circle.
    2. Replace the raw path data with the calculated perfect geometric shape.
    3. Add a "Magic Pencil" tool toggle to enable this mode.

- **Story 8.3: Smart Connectors**
    - **As a User**, I want lines to automatically attach to the edge of shapes, so that I can move shapes without breaking the flowchart.

- **Story 8.4: Gesture Controls**
    - **As a Touch User**, I want to undo an action by double-tapping with two fingers, so that I can work faster on my tablet.

- **Story 8.5: Auto-Arrangement**
    - **As a User**, I want a "Tidy Up" button to automatically align and space out my selected objects, so that the diagram looks organized.

Epic 8: Security & Administration

- **Story 7.1: Read-Only Mode**
    - **As a Host**, I want to switch the room to "Read-Only," so that guests can view the diagram without accidentally changing it.
- **Story 7.2: Anti-Spam Protection**
    - **As a User**, I want the system to automatically ignore bots or users who draw impossibly fast, so that the canvas doesn't get vandalized.
- **Story 7.3: Input Sanitization**
    - **As a User**, I want the system to block any malicious scripts hidden in text boxes, so that my browser remains secure while I collaborate.
- **Story 7.5: Secure Encryption**
    - **As an Enterprise User**, I want my session connection to be encrypted, so that my sensitive diagrams are safe from eavesdroppers

Epic 5: Fort Knox Security

User Stories & Tasks:

- **Story 5.1: Anti-XSS**

- **User Story:** As a User, I want text inputs sanitized so hackers can't inject scripts.
    - **Acceptance Criteria:** <script> tags render as text.
    - **Tasks:**
        - Run npm install dompurify.
        - Create a wrapper function sanitizeText(input).
        - Apply this wrapper to all incoming WebSocket text messages before rendering.
- **Story 5.2: Secure Headers**
    - **User Story:** As a DevOps, I want CSP headers so I prevent CSRF attacks.
    - **Acceptance Criteria:** Headers include Content-Security-Policy.
    - **Tasks:**
        - Run npm install helmet.
        - Configure app.use(helmet()) in the Express server.
        - Whitelist the WebSocket domain in the CSP configuration.
- **Story 5.3: Rate Limiting**
    - **User Story:** As a System, I want to block spam bots so the room doesn't crash.
    - **Acceptance Criteria:** Block IPs sending >200 msgs/sec.
    - **Tasks:**
        - Implement rate-limiter-flexible middleware.
        - Add logic to terminate the WebSocket connection on violation.