Architect a Collaborative Kanban or Project Management Board (like Trello)

🗋 Table of Contents

Table of Contents

Clarify the Problem and Requirements

☐ Back to Top

Problem Understanding

☐ Back to Top

Design a collaborative Kanban board system that enables teams to manage projects through visual workflow management, similar to Trello, Asana, or Jira. The system must support real-time collaboration, drag-and-drop interactions, flexible board configurations, and seamless synchronization across multiple users and devices.

Functional Requirements

☐ Back to Top

- **Board Management**: Create, edit, delete boards with customizable workflows

- **Card System**: Tasks/cards with descriptions, attachments, comments, labels, due dates
- **Column/List Management**: Configurable workflow stages, WIP limits, custom fields
- **Drag & Drop**: Intuitive card movement between columns, reordering
- **Real-time Collaboration**: Multi-user editing, live cursor tracking, conflict resolution
- **Team Features**: User assignments, permissions, activity feeds, notifications
- **Rich Content**: Markdown support, file attachments, checklists, time tracking
- **Board Templates**: Pre-configured workflows for different project types

Non-Functional Requirements

☐ Back to Top

- **Performance**: <200ms card movement, <100ms real-time updates, 60fps animations
- **Scalability**: 10K+ boards, 100K+ cards, 1K+ concurrent users per board
- **Availability**: 99.9% uptime with offline capability and conflict resolution
- **Real-time**: <50ms latency for collaborative updates
- **Cross-platform**: Web, mobile apps, desktop with feature parity
- **Accessibility**: WCAG 2.1 AA compliance, keyboard navigation, screen reader support

Key Assumptions

☐ Back to Top

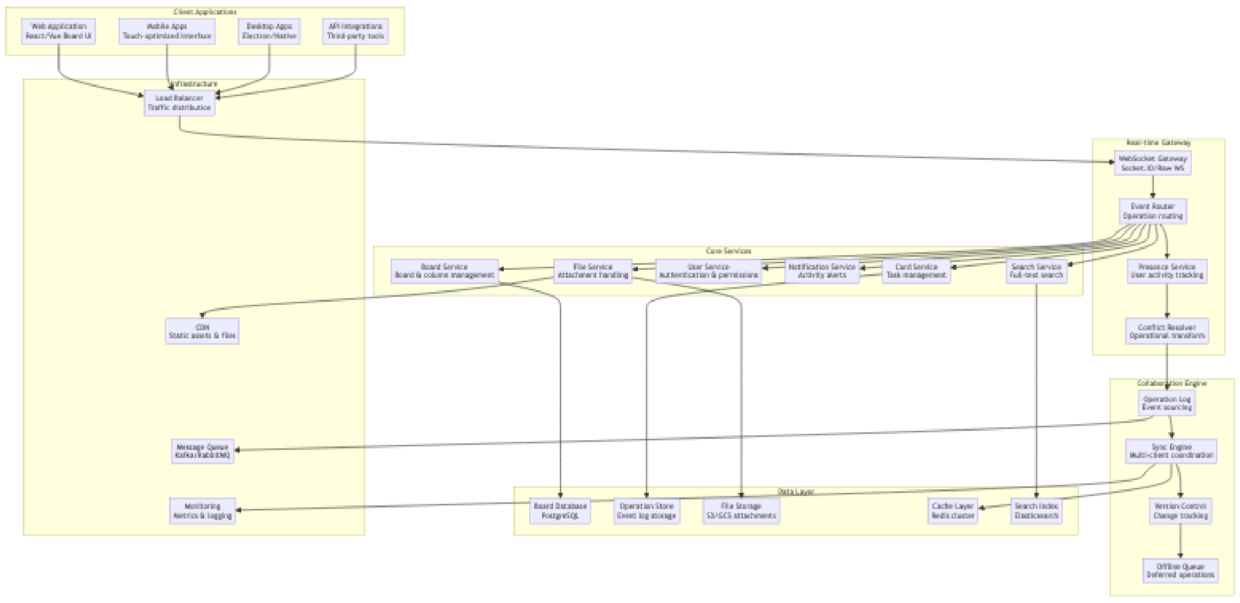
- Average board: 5-20 columns, 50-500 cards
- Team size: 5-50 members per board
- Concurrent editors: 5-20 users simultaneously
- Update frequency: 100-500 operations/hour during active use
- Attachment sizes: Max 10MB per file, 100MB per card
- Browser support: Modern browsers with HTML5 drag-and-drop API

High-Level Architecture

☐ Back to Top

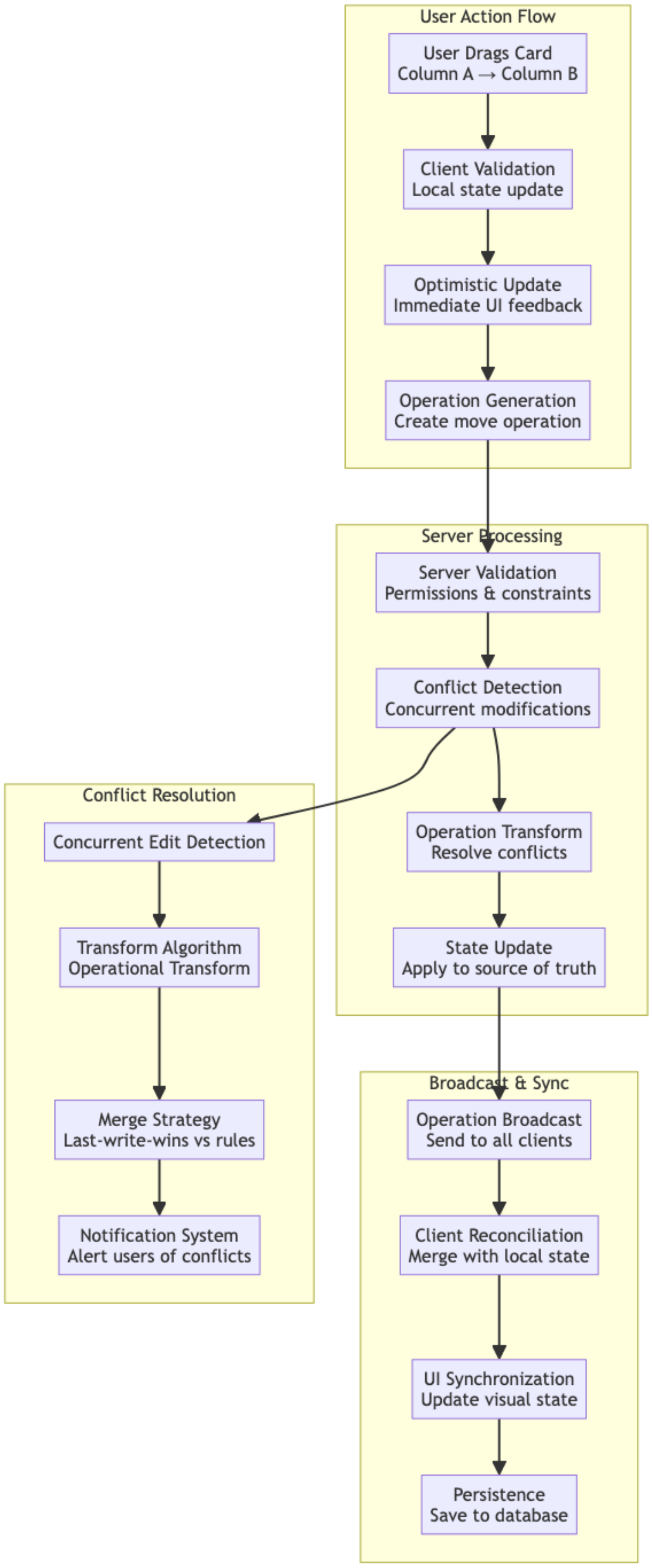
Collaborative Board System Architecture

□ Back to Top


Client Applications
Web Application React/Vue Board UI
Mobile Apps Touch-optimized interface
Desktop Apps Electron/Native
API Integrations Third-party tools
Infrastructure
Load Balancer Traffic distribution
CDN Static assets & files
Message Queue Kafka/RabbitMQ
Monitoring Metrics & logging
Real-time Gateway
WebSocket Gateway Socket.IO/Raw WS
Event Router Operation routing
Presence Service User activity tracking
Conflict Resolver Operational transform
Core Services
Board Service Board & column management
File Service Attachment handling
User Service Authentication & permissions
Notification Service Activity alerts
Card Service Task management
Search Service Full-text search
Collaboration Engine
Operation Log Event sourcing
Sync Engine Multi-client coordination
Version Control Change tracking
Offline Queue Deferred operations
Data Layer
Board Database PostgreSQL
Operation Store Event log storage
File Storage S3/GCS attachments
Cache Layer Redis cluster
Search Index Elasticsearch

Real-time Collaboration Flow

□ Back to Top

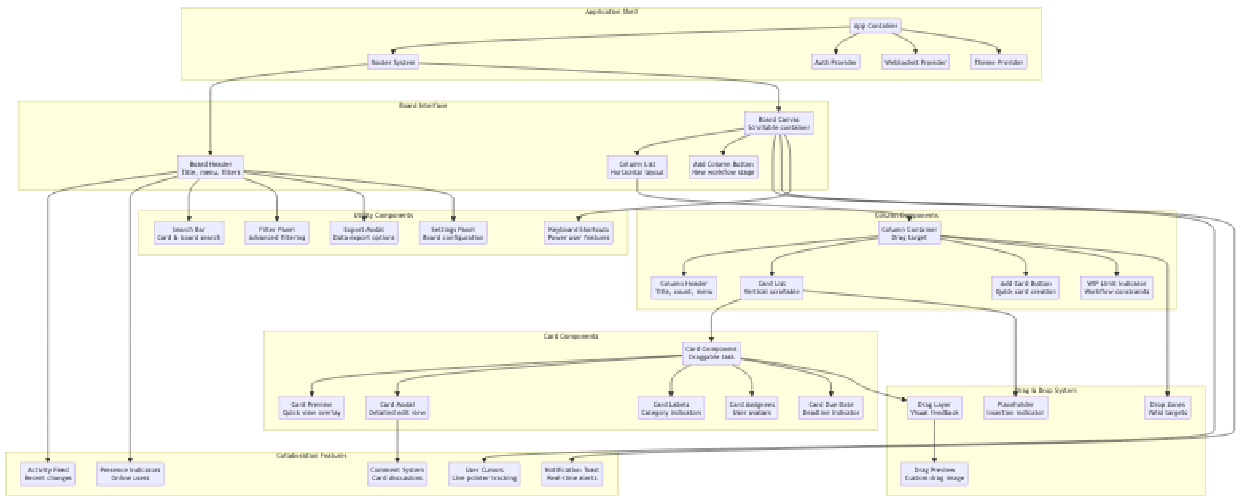

User Action Flow
User Drags Card
Column A → Column B
Client Validation
Local state update
Optimistic Update
Immediate UI feedback
Operation Generation
Create move operation
Server Processing
Server Validation
Permissions & constraints
Conflict Detection
Concurrent modifications
Operation Transform
Resolve conflicts
State Update
Apply to source of truth
Conflict Resolution
Concurrent Edit Detection
Transform Algorithm
Operational Transform
Merge Strategy
Last-write-wins vs rules
Notification System
Alert users of conflicts
Broadcast & Sync
Operation Broadcast
Send to all clients
Client Reconciliation
Merge with local state
UI Synchronization
Update visual state
Persistence
Save to database

UI/UX and Component Structure

☐ Back to Top

Frontend Component Architecture

☐ Back to Top



React Component Implementation ☐ Back to Top

KanbanBoard.jsx

```jsx
import React, { useState, useEffect, useCallback } from 'react';
import { DndProvider } from 'react-dnd';
import { HTML5Backend } from 'react-dnd-html5-backend';
import { KanbanProvider } from './KanbanContext';
import BoardHeader from './BoardHeader';
import ColumnList from './ColumnList';
import { useWebSocket } from './hooks/useWebSocket';

const KanbanBoard = ({ boardId, userId }) => {
  const [board, setBoard] = useState(null);
  const [columns, setColumns] = useState([]);
  const [cards, setCards] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [selectedCards, setSelectedCards] = useState([]);

  const { socket, isConnected } = useWebSocket(`/boards/${boardId}`);

  useEffect(() => {
    loadBoardData();
  }, [boardId]);

  useEffect(() => {
    if (socket) {
      socket.on('card:moved', handleCardMoved);
      socket.on('card:updated', handleCardUpdated);
      socket.on('column:updated', handleColumnUpdated);

      return () => {
        socket.off('card:moved');
        socket.off('card:updated');
        socket.off('column:updated');
      };
    }
  }, [socket]);

  const loadBoardData = async () => {
    setIsLoading(true);
    try {
      const response = await fetch(`/api/boards/${boardId}`);
      const data = await response.json();

      setBoard(data.board);
      setColumns(data.columns);
      setCards(data.cards);
    } catch (error) {
      console.error('Failed to load board:', error);
    } finally {
      setIsLoading(false);
    }
  };

  const moveCard = useCallback(async (cardId, sourceColumnId, targetColumnId, targetInde
    // Optimistic update
    setCards(prevCards => {
      const updatedCards = [...prevCards];
      const cardIndex = updatedCards.findIndex(card => card.id === cardId);

      if (cardIndex === -1) return prevCards;
```

```
      const card = updatedCards[cardIndex];
      updatedCards[cardIndex] = {
        ...card,
        columnId: targetColumnId,
        position: targetIndex
      };

      return updatedCards;
    });

    try {
      const response = await fetch(`/api/cards/${cardId}/move`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          sourceColumnId,
          targetColumnId,
          position: targetIndex
        })
      });

      if (!response.ok) {
        // Revert on failure
        loadBoardData();
      } else {
        // Broadcast to other users
        socket?.emit('card:move', {
          cardId,
          sourceColumnId,
          targetColumnId,
          position: targetIndex
        });
      }
    } catch (error) {
      console.error('Failed to move card:', error);
      loadBoardData();
    }
  }, [socket]);

  const addCard = useCallback(async (columnId, cardData) => {
    try {
      const response = await fetch('/api/cards', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          ...cardData,
          columnId,
          boardId
        })
      });

      const newCard = await response.json();
      setCards(prev => [...prev, newCard]);

      socket?.emit('card:created', newCard);
    } catch (error) {
      console.error('Failed to add card:', error);
    }
  }, [boardId, socket]);

  const updateCard = useCallback(async (cardId, updates) => {
    setCards(prev => prev.map(card =>
      card.id === cardId ? { ...card, ...updates } : card
    ));

    try {
      await fetch(`/api/cards/${cardId}`, {
        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(updates)
      });

      socket?.emit('card:updated', { cardId, updates });
    } catch (error) {
      console.error('Failed to update card:', error);
      loadBoardData();
    }
  }, [socket]);

  const handleCardMoved = useCallback((data) => {
    setCards(prev => prev.map(card =>
      card.id === data.cardId
        ? { ...card, columnId: data.targetColumnId, position: data.position }
        : card
    ));
  }, []);

  const handleCardUpdated = useCallback((data) => {
    setCards(prev => prev.map(card =>
      card.id === data.cardId ? { ...card, ...data.updates } : card
    ));
  }, []);

  const handleColumnUpdated = useCallback((data) => {
    setColumns(prev => prev.map(col =>
      col.id === data.columnId ? { ...col, ...data.updates } : col
    ));
  }, []);

  if (isLoading) {
    return (
      <div className="kanban-loading">
        <div className="loading-spinner" />
        <p>Loading board...</p>
      </div>
    );
  }

  return (
    <KanbanProvider value={{
      board,
      columns,
      cards,
      selectedCards,
      setSelectedCards,
      moveCard,
      addCard,
      updateCard,
      isConnected
    }}>
      <DndProvider backend={HTML5Backend}>
        <div className="kanban-board">
          <BoardHeader board={board} />
          <ColumnList />
        </div>
      </DndProvider>
    </KanbanProvider>
  );
};

export default KanbanBoard;
```

CardComponent.jsx

```
import React, { useState, useContext } from 'react';
import { useDrag } from 'react-dnd';
import { KanbanContext } from './KanbanContext';
import CardModal from './CardModal';
import CardLabels from './CardLabels';
import CardAssignees from './CardAssignees';

const CardComponent = ({ card }) => {
  const { updateCard, selectedCards, setSelectedCards } = useContext(KanbanContext);
  const [showModal, setShowModal] = useState(false);
  const [isSelected, setIsSelected] = useState(selectedCards.includes(card.id));

  const [{ isDragging }, drag] = useDrag({
    type: 'card',
    item: { id: card.id, columnId: card.columnId },
    collect: (monitor) => ({
      isDragging: monitor.isDragging()
    })
  });

  const handleCardClick = (e) => {
    if (e.ctrlKey || e.metaKey) {
      // Multi-select
      const newSelection = isSelected
        ? selectedCards.filter(id => id !== card.id)
        : [...selectedCards, card.id];
      setSelectedCards(newSelection);
      setIsSelected(!isSelected);
    } else {
      setShowModal(true);
    }
  };

  const formatDueDate = (date) => {
    if (!date) return null;
    const dueDate = new Date(date);
    const today = new Date();
    const diffTime = dueDate - today;
    const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));

    if (diffDays < 0) return { text: 'Overdue', className: 'overdue' };
    if (diffDays === 0) return { text: 'Today', className: 'due-today' };
    if (diffDays === 1) return { text: 'Tomorrow', className: 'due-tomorrow' };
    return { text: `${diffDays} days`, className: 'due-future' };
  };
```

```
const dueDateInfo = formatDueDate(card.dueDate);

return (
  <>
    <div
      ref={drag}
      className={`card-component ${isDragging ? 'dragging' : ''} ${isSelected ? 'selec
      onClick={handleCardClick}
      style={{
        opacity: isDragging ? 0.5 : 1
      }}
    >
      {card.coverImage && (
        <div className="card-cover">
          <img src={card.coverImage} alt="" loading="lazy" />
        </div>
      )}

      <div className="card-content">
        <CardLabels labels={card.labels} />

        <h3 className="card-title">{card.title}</h3>

        {card.description && (
          <p className="card-description">{card.description.slice(0, 100)}...</p>
        )}

        <div className="card-footer">
          <div className="card-meta">
            {dueDateInfo && (
              <span className={`due-date ${dueDateInfo.className}`}>
                  {dueDateInfo.text}
              </span>
            )}

            {card.attachments && card.attachments.length > 0 && (
              <span className="attachment-count">
                  {card.attachments.length}
              </span>
            )}

            {card.comments && card.comments.length > 0 && (
              <span className="comment-count">
                  {card.comments.length}
              </span>
              )}
            </div>

            <CardAssignees assignees={card.assignees} />
          </div>
        </div>
      </div>

      {showModal && (
        <CardModal
          card={card}
          onClose={() => setShowModal(false)}
          onUpdate={updateCard}
        />
      )}
    </>
  );
};

export default CardComponent;
```

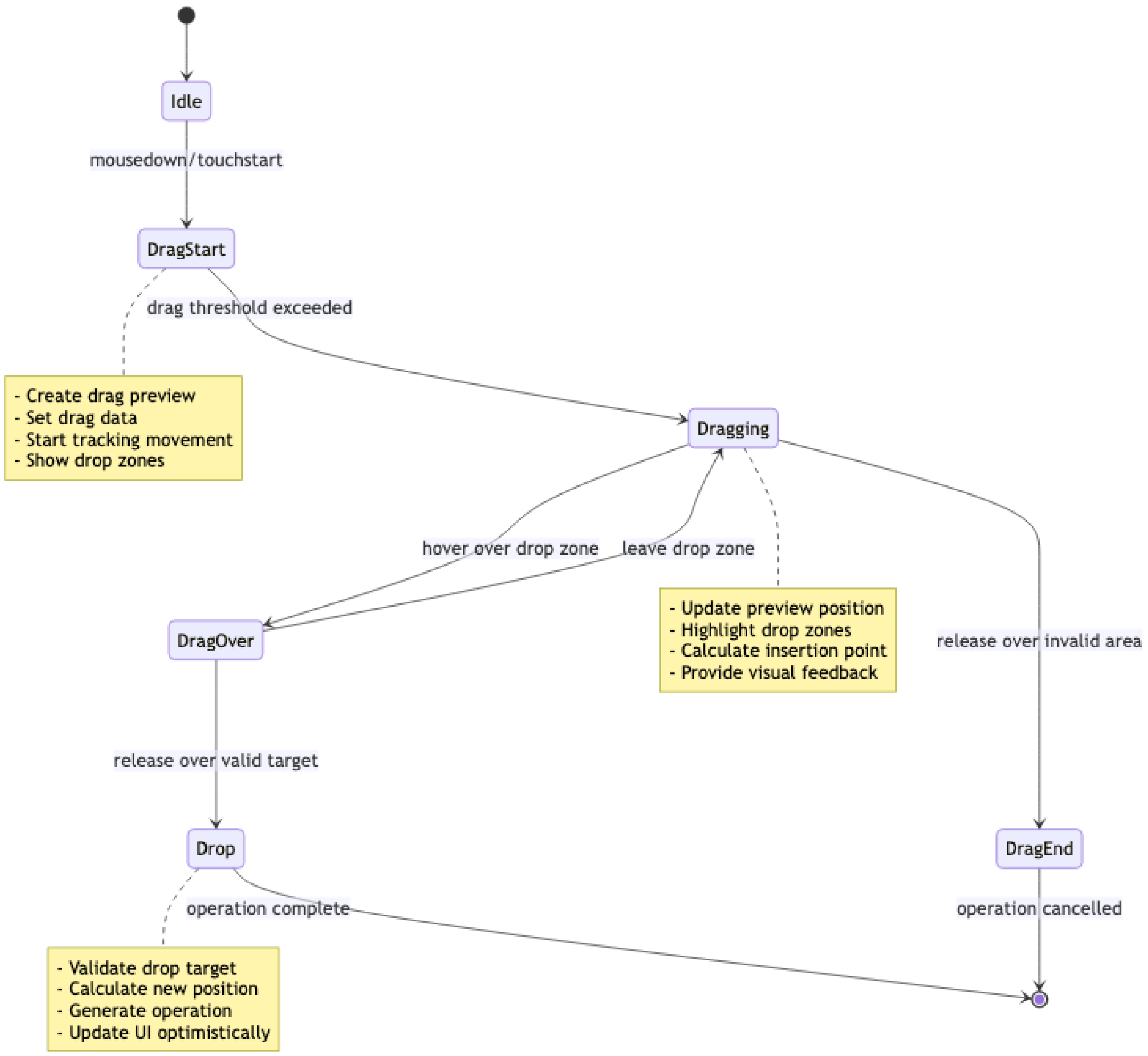
ColumnContainer.jsx

```
import React, { useContext, useState } from 'react';
import { useDrop } from 'react-dnd';
import { KanbanContext } from './KanbanContext';
import CardComponent from './CardComponent';
import AddCardButton from './AddCardButton';
import WipLimitIndicator from './WipLimitIndicator';

const ColumnContainer = ({ column }) => {
  const { cards, moveCard } = useContext(KanbanContext);
  const [showAddCard, setShowAddCard] = useState(false);

  const columnCards = cards
    .filter(card => card.columnId === column.id)
    .sort((a, b) => a.position - b.position);

  const [{ isOver, canDrop }, drop] = useDrop({
    accept: 'card',
    drop: (item, monitor) => {
      if (item.columnId !== column.id) {
        const targetIndex = columnCards.length;
        moveCard(item.id, item.columnId, column.id, targetIndex);
      }
    },
    collect: (monitor) => ({
      isOver: monitor.isOver(),
      canDrop: monitor.canDrop()
    })
  });

  const canAcceptCard = () => {
    if (!column.wipLimit) return true;
    return columnCards.length < column.wipLimit;
  };

  return (
    <div
      ref={drop}
      className={`column-container ${isOver ? 'drag-over' : ''} ${!canAcceptCard() ? 'wi
    >
      <div className="column-header">
        <h2 className="column-title">
          {column.title}
          <span className="card-count">({columnCards.length})</span>
        </h2>

        {column.wipLimit && (
          <WipLimitIndicator
            current={columnCards.length}
            limit={column.wipLimit}
          />
        )}
      </div>

      <div className="card-list">
        {columnCards.map((card, index) => (
          <CardComponent key={card.id} card={card} />
        ))}

        {isOver && canDrop && (
          <div className="drop-indicator">
            Drop card here
          </div>
        )}
      </div>

      <AddCardButton
        columnId={column.id}
        show={showAddCard}
```

```
        onToggle={setShowAddCard}
      />
    </div>
  );
};

export default ColumnContainer;
```

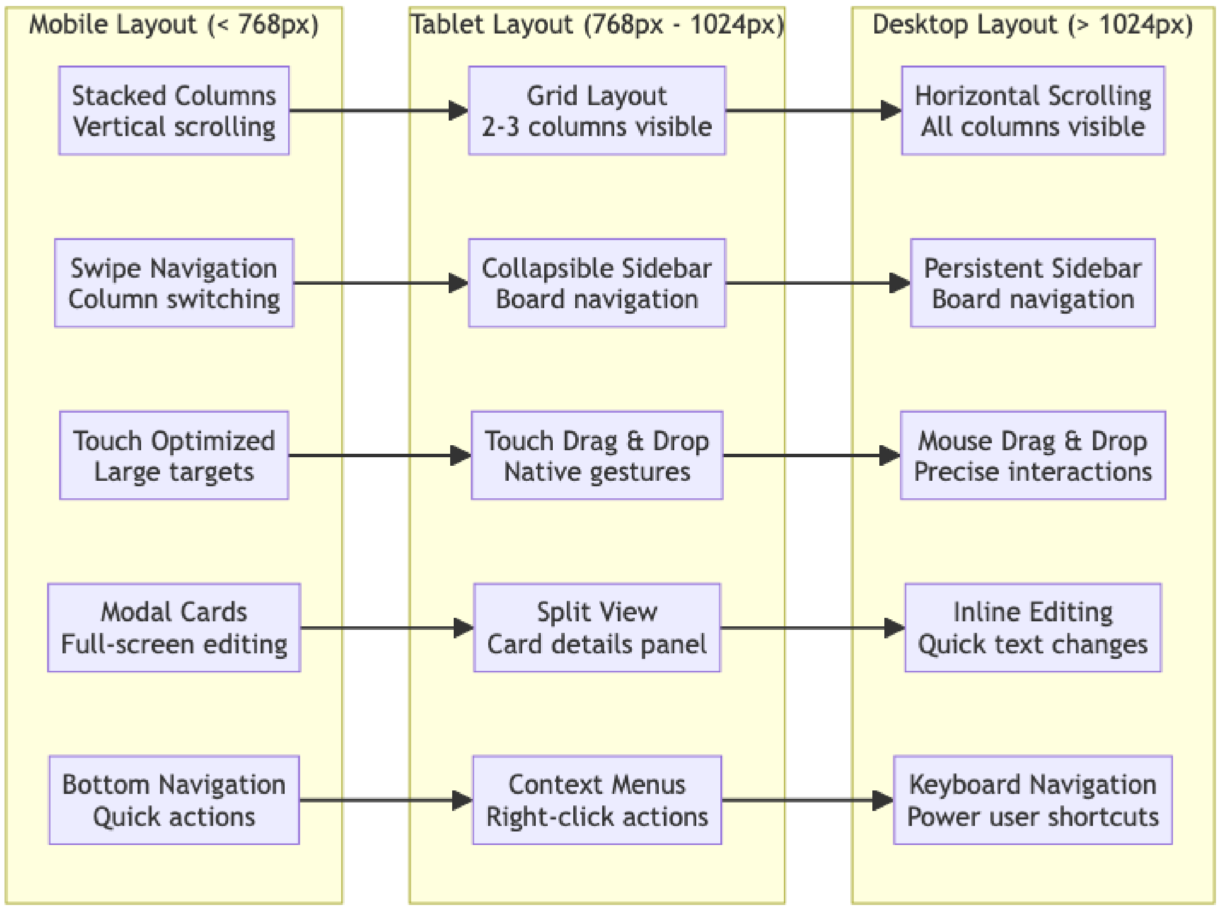
Drag & Drop Implementation

☐ Back to Top



Responsive Board Layout

☐ Back to Top


Mobile Layout (< 768px)
Tablet Layout (768px - 1024px)
Desktop Layout (> 1024px)
Stacked Columns Vertical scrolling
Grid Layout 2-3 columns visible
Horizontal Scrolling All columns visible
Swipe Navigation Column switching
Collapsible Sidebar Board navigation
Persistent Sidebar Board navigation
Touch Optimized Large targets
Touch Drag & Drop Native gestures
Mouse Drag & Drop Precise interactions
Modal Cards Full-screen editing
Split View Card details panel
Inline Editing Quick text changes
Bottom Navigation Quick actions
Context Menus Right-click actions
Keyboard Navigation Power user shortcuts

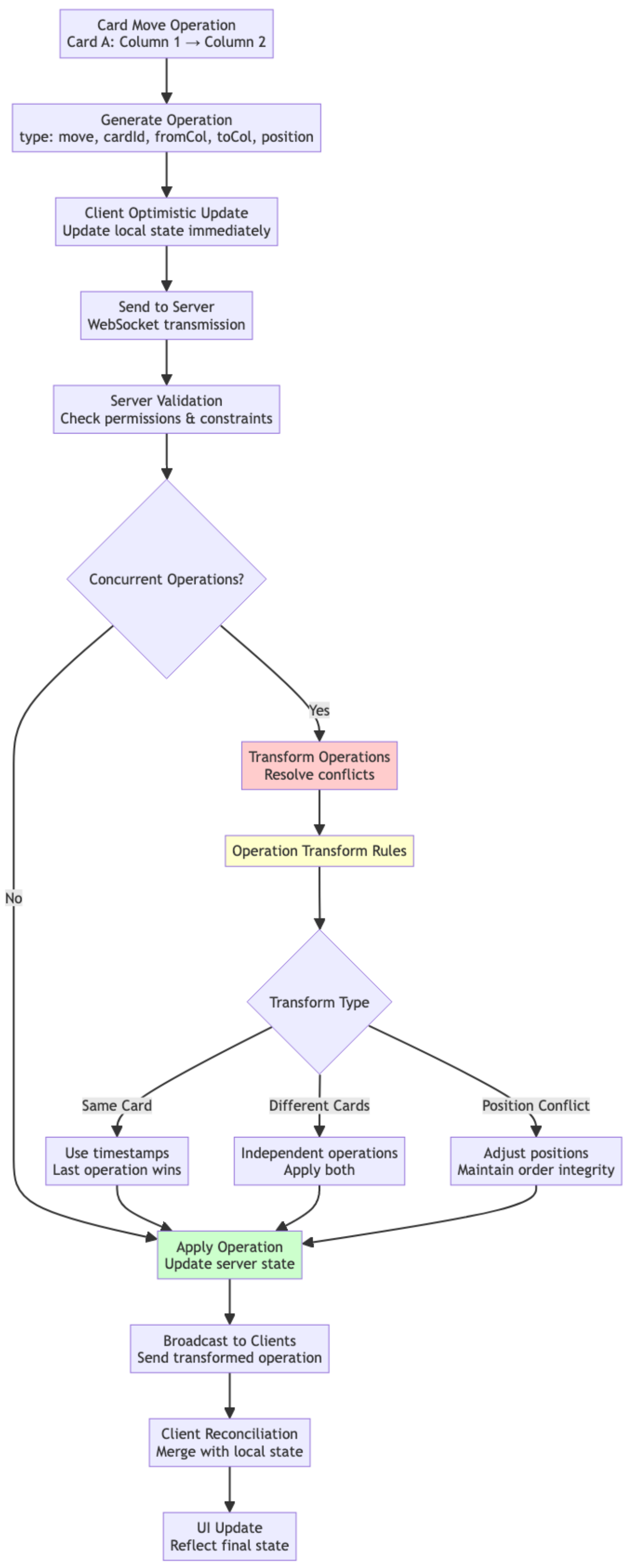
Real-Time Sync, Data Modeling & APIs

☐ Back to Top

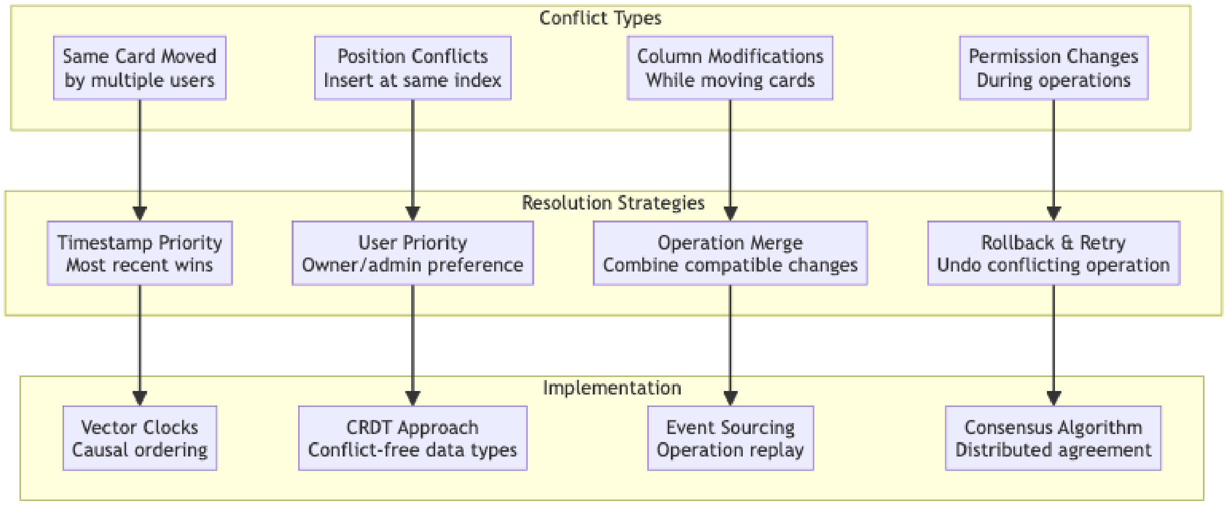
Operational Transform for Kanban Operations

☐ Back to Top

Card Movement Algorithm ☐ Back to Top


Card Move Operation
Card A: Column 1 → Column 2
Generate Operation
type: move, cardId, fromCol, toCol, position
Client Optimistic Update
Update local state immediately
Send to Server
WebSocket transmission
Server Validation
Check permissions & constraints
Concurrent Operations?
Yes
No
Transform Operations
Resolve conflicts
Operation Transform Rules
Transform Type
Same Card
Different Cards
Position Conflict
Use timestamps
Last operation wins
Independent operations
Apply both
Adjust positions
Maintain order integrity
Apply Operation
Update server state
Broadcast to Clients
Send transformed operation
Client Reconciliation
Merge with local state
UI Update
Reflect final state

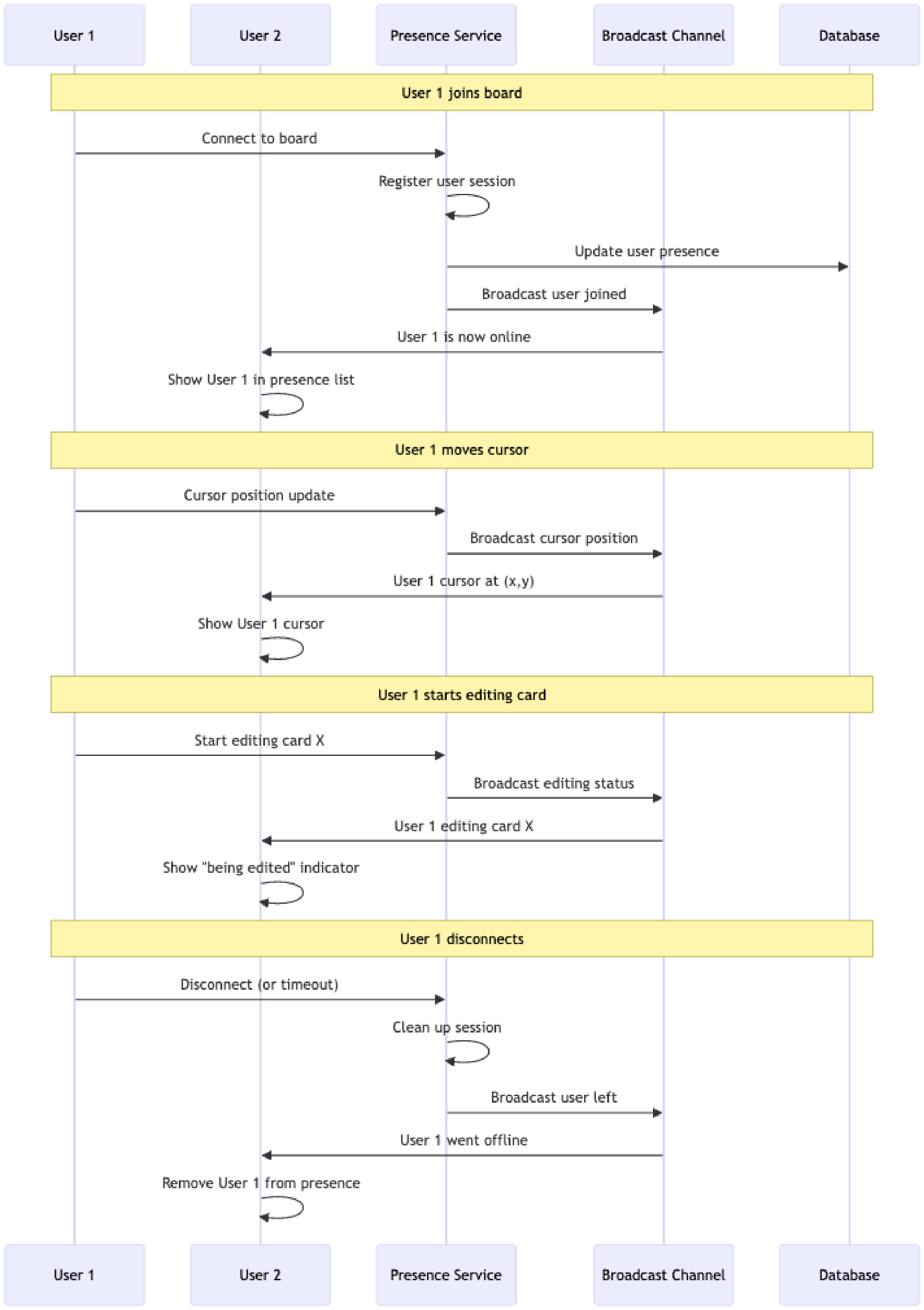
Conflict Resolution Strategy Back to Top


Conflict Types
Same Card Moved by multiple users
Position Conflicts Insert at same index
Column Modifications While moving cards
Permission Changes During operations
Resolution Strategies
Timestamp Priority Most recent wins
User Priority Owner/admin preference
Operation Merge Combine compatible changes
Rollback & Retry Undo conflicting operation
Implementation
Vector Clocks Causal ordering
CRDT Approach Conflict-free data types
Event Sourcing Operation replay
Consensus Algorithm Distributed agreement

Real-time Presence System

 Back to Top

User Activity Tracking Back to Top


User 1
User 2
Presence Service
Broadcast Channel
Database
User 1 joins board
Connect to board
Register user session
Update user presence
Broadcast user joined
User 1 is now online
Show User 1 in presence list
User 1 moves cursor
Cursor position update
Broadcast cursor position
User 1 cursor at (x,y)
Show User 1 cursor
User 1 starts editing card
Start editing card X
Broadcast editing status
User 1 editing card X
Show "being edited" indicator
User 1 disconnects
Disconnect (or timeout)
Clean up session
Broadcast user left
User 1 went offline
Remove User 1 from presence
User 1
User 2
Presence Service
Broadcast Channel
Database

Data Models

□ Back to Top

Board Schema □ Back to Top

```
interface Board {
  id: string
  name: string
  description?: string
  visibility: 'private' | 'team' | 'organization' | 'public'

  columns: Column[]
  members: BoardMember[]
  settings: BoardSettings

  created_at: Date
  updated_at: Date
  created_by: string

  // Collaboration
  version: number
  last_activity: Date

  // Configuration
  workflow_type: 'kanban' | 'scrum' | 'custom'
  labels: Label[]
  custom_fields: CustomField[]
}

interface Column {
  id: string
  board_id: string
  name: string
  position: number

  // Workflow
  wip_limit?: number
  column_type: 'backlog' | 'in_progress' | 'done' | 'custom'

  // Cards (ordered by position)
  card_ids: string[]
  // Styling
  color?: string
  collapsed: boolean

  created_at: Date
  updated_at: Date
}

interface Card {
  id: string
  board_id: string
  column_id: string
  position: number

  // Content
  title: string
  description?: string

  // Metadata
  labels: string[]
  assignees: string[]
  due_date?: Date

  // Rich content
  checklist: ChecklistItem[]
  attachments: Attachment[]
  comments: Comment[]

  // Tracking
  created_at: Date
  updated_at: Date
  created_by: string

  // Custom fields
  custom_field_values: Record<string, any>
}
```

Operation Schema ☐ Back to Top

```
interface Operation {
  id: string
  type: 'move_card' | 'create_card' | 'update_card' | 'delete_card' |
      'create_column' | 'update_column' | 'delete_column'

  board_id: string
  user_id: string
  timestamp: number

  // Operation data
  data: {
    card_id?: string
    column_id?: string
    from_column?: string
    to_column?: string
    from_position?: number
    to_position?: number
    changes?: Record<string, any>
  }

  // Conflict resolution
  vector_clock: Record<string, number>
  causally_ready: boolean

  // Status
  applied: boolean
  conflicts: string[]
}
```

TypeScript Interfaces & Component Props

☐ Back to Top

Core Data Interfaces

```
interface KanbanBoard {
  id: string;
  title: string;
  description?: string;
  columns: BoardColumn[];
  members: BoardMember[];
  settings: BoardSettings;
  permissions: BoardPermissions;
  createdAt: Date;
  updatedAt: Date;
  owner: string;
```

```
}

interface BoardColumn {
  id: string;
  title: string;
  position: number;
  wipLimit?: number;
  color?: string;
  cards: KanbanCard[];
  isCollapsed: boolean;
  rules?: ColumnRule[];
}

interface KanbanCard {
  id: string;
  title: string;
  description?: string;
  assignees: string[];
  labels: CardLabel[];
  dueDate?: Date;
  priority: 'low' | 'medium' | 'high' | 'urgent';
  attachments: Attachment[];
  checklist: ChecklistItem[];
  comments: Comment[];
  position: number;
  columnId: string;
  createdAt: Date;
  updatedAt: Date;
  estimatedHours?: number;
  timeSpent?: number;
}

interface BoardMember {
  userId: string;
  role: 'owner' | 'admin' | 'member' | 'viewer';
  permissions: MemberPermissions;
  joinedAt: Date;
  isActive: boolean;
}

interface DragDropState {
  isDragging: boolean;
  draggedItem?: {
    type: 'card' | 'column';
    id: string;
    sourceColumnId?: string;
    sourceIndex: number;
  };
  dropTarget?: {
    columnId: string;
    index: number;
  };
  ghostPosition?: {
    x: number;
    y: number;
  };
}

interface ActivityFeed {
  id: string;
  type: 'card_created' | 'card_moved' | 'card_updated' | 'member_added';
  actorId: string;
  targetId: string;
  metadata: Record<string, any>;
  timestamp: Date;
  boardId: string;
}
```

Component Props Interfaces

```
interface KanbanBoardProps {
  board: KanbanBoard;
  onCardMove: (cardId: string, targetColumnId: string, position: number) => void;
  onColumnMove: (columnId: string, newPosition: number) => void;
  onCardClick: (card: KanbanCard) => void;
  onCardCreate: (columnId: string, card: Partial<KanbanCard>) => void;
  onCardUpdate: (cardId: string, updates: Partial<KanbanCard>) => void;
  onCardDelete: (cardId: string) => void;
  enableVirtualization?: boolean;
  showActivityFeed?: boolean;
}

interface BoardColumnProps {
  column: BoardColumn;
  cards: KanbanCard[];
  onCardDrop: (cardId: string, position: number) => void;
  onCardCreate: (card: Partial<KanbanCard>) => void;
  onColumnUpdate: (updates: Partial<BoardColumn>) => void;
  onColumnDelete: () => void;
  isDragOver?: boolean;
  isCollapsed?: boolean;
  showWipLimit?: boolean;
}

interface KanbanCardProps {
  card: KanbanCard;
  onClick: (card: KanbanCard) => void;
  onUpdate: (updates: Partial<KanbanCard>) => void;
  onDelete: () => void;
  isDragging?: boolean;
  isSelected?: boolean;
  showLabels?: boolean;
  showAssignees?: boolean;
  showDueDate?: boolean;
  compact?: boolean;
}

interface CardDetailModalProps {
  card: KanbanCard;
  isOpen: boolean;
  onClose: () => void;
  onUpdate: (updates: Partial<KanbanCard>) => void;
  onDelete: () => void;
  boardMembers: BoardMember[];
  availableLabels: CardLabel[];
  showComments?: boolean;
  showChecklist?: boolean;
  showAttachments?: boolean;
}

interface BoardHeaderProps {
  board: KanbanBoard;
  onTitleUpdate: (title: string) => void;
  onMemberAdd: (userId: string) => void;
  onSettingsOpen: () => void;
  onFilterChange: (filters: BoardFilters) => void;
  showFilters?: boolean;
  showMembers?: boolean;
  showSearch?: boolean;
}
```

API Reference

☐ Back to Top

Board Management

- `GET /api/boards` - Get user's boards with access permissions and metadata
- `POST /api/boards` - Create new kanban board with initial columns and settings
- `GET /api/boards/:id` - Get board details with columns, cards, and members
- `PUT /api/boards/:id` - Update board title, description, or settings
- `DELETE /api/boards/:id` - Delete board and all associated data

Column Operations

- `POST /api/boards/:id/columns` - Add new column to board with position
- `PUT /api/columns/:id` - Update column title, WIP limit, or rules
- `DELETE /api/columns/:id` - Delete column and handle card reassignment
- `PUT /api/columns/:id/position` - Reorder column position in board
- `POST /api/columns/:id/duplicate` - Duplicate column with cards (optional)

Card Management

- `POST /api/boards/:id/cards` - Create new card in specified column
- `GET /api/cards/:id` - Get detailed card information with history
- `PUT /api/cards/:id` - Update card content, assignees, or metadata
- `DELETE /api/cards/:id` - Delete card and clean up references
- `POST /api/cards/:id/move` - Move card between columns with position

Drag & Drop Operations

- `POST /api/cards/:id/drag-start` - Initialize card drag operation
- `PUT /api/cards/:id/drag-move` - Update card position during drag
- `POST /api/cards/:id/drop` - Complete card drop with final position
- `POST /api/columns/:id/reorder` - Reorder multiple cards in column
- `POST /api/board/:id/bulk-move` - Move multiple cards in single operation

Real-time Collaboration

- `WS /api/boards/:id/connect` - WebSocket connection for real-time updates
- `WS CARD_UPDATED` - Broadcast card changes to board collaborators
- `WS MEMBER_CURSOR` - Share cursor position during card interactions
- `WS TYPING_INDICATOR` - Show typing indicators for card editing
- `WS PRESENCE_UPDATE` - Update member presence and activity status

Comments & Activity

- `POST /api/cards/:id/comments` - Add comment to card with mentions

- `GET /api/cards/:id/comments` - Get card comments with pagination
- `PUT /api/comments/:id` - Edit comment content (author only)
- `DELETE /api/comments/:id` - Delete comment with moderation rules
- `GET /api/boards/:id/activity` - Get board activity feed and audit log

Labels & Categories

- `GET /api/boards/:id/labels` - Get available labels for board
- `POST /api/boards/:id/labels` - Create new label with color and name
- `PUT /api/labels/:id` - Update label properties or color
- `DELETE /api/labels/:id` - Delete label and remove from cards
- `POST /api/cards/:id/labels` - Add or remove labels from card

Member & Permission Management

- `POST /api/boards/:id/members` - Invite member to board with role
- `PUT /api/boards/:id/members/:userId` - Update member role or permissions
- `DELETE /api/boards/:id/members/:userId` - Remove member from board
- `GET /api/boards/:id/permissions` - Get detailed permission matrix
- `PUT /api/boards/:id/permissions` - Update board permission settings

Search & Filtering

- `GET /api/boards/:id/search` - Search cards and comments within board
- `POST /api/boards/:id/filter` - Apply filters to board view
- `GET /api/cards/assigned` - Get cards assigned to current user
- `GET /api/cards/due-soon` - Get cards with upcoming due dates
- `POST /api/boards/:id/export` - Export board data in various formats

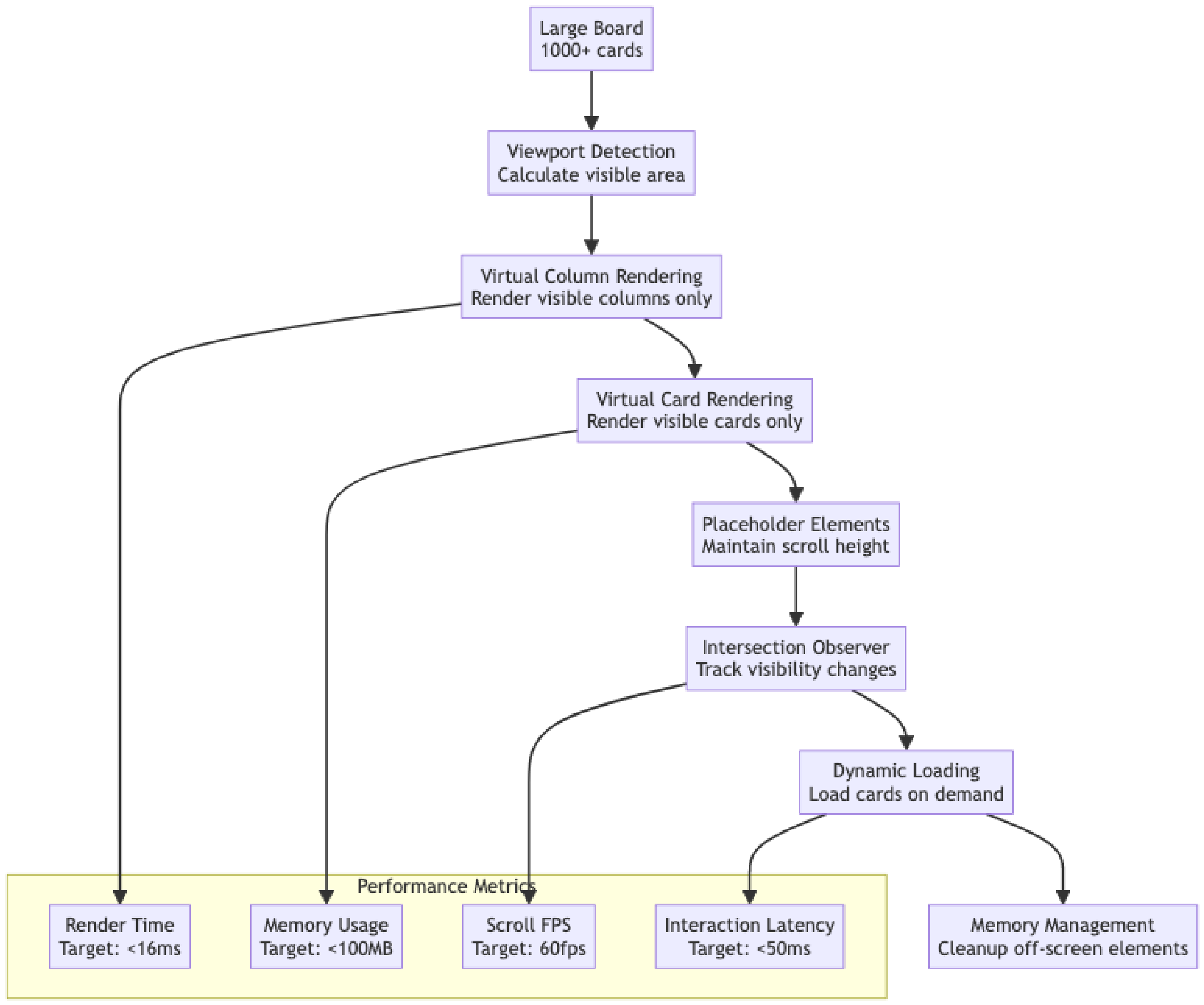
Performance and Scalability

☐ Back to Top

Client-Side Optimization

☐ Back to Top

Virtual Scrolling for Large Boards ☐ Back to Top

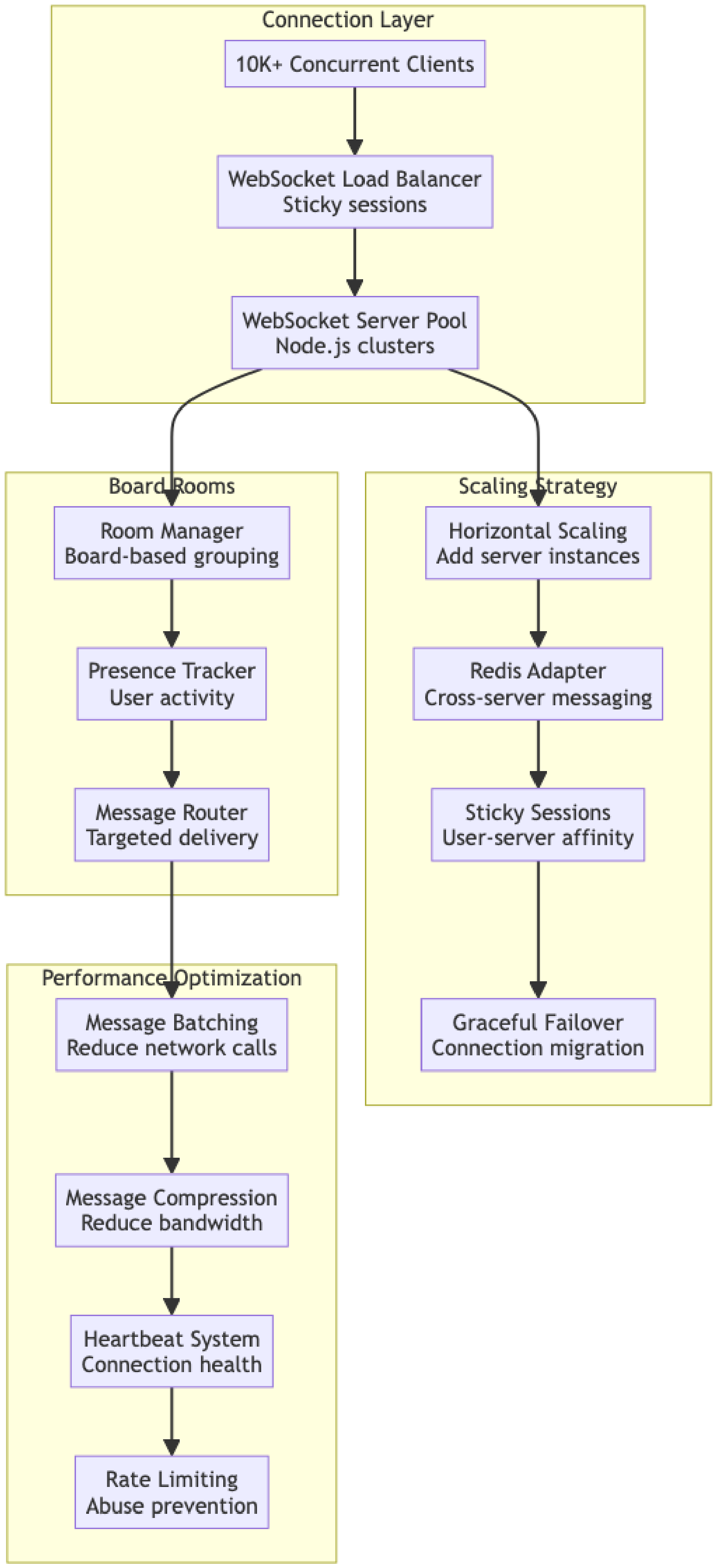


Real-time Scaling

☐ Back to Top

WebSocket Connection Management

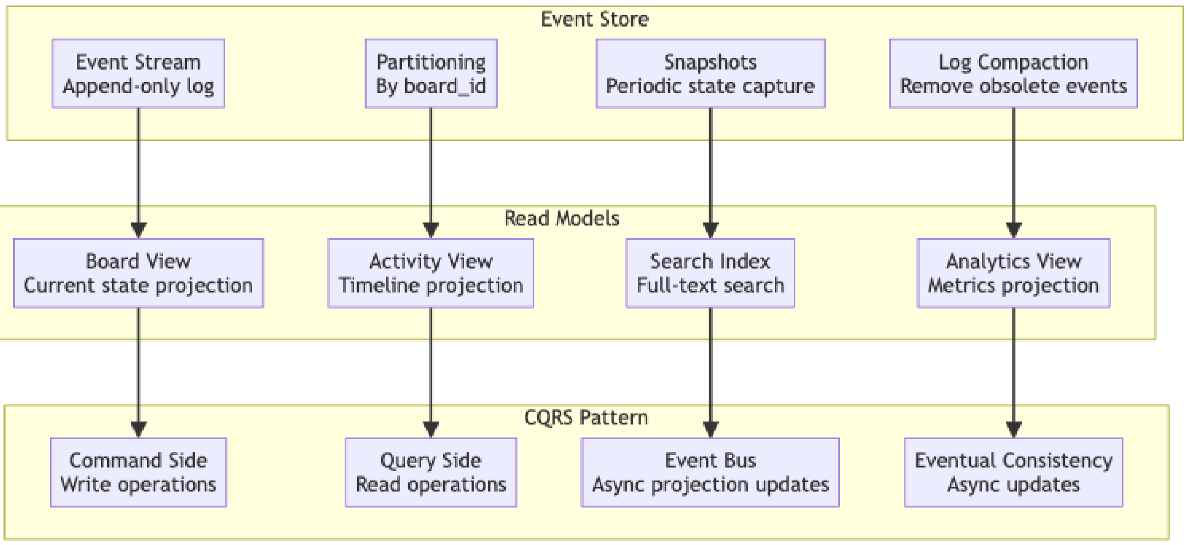
☐ Back to Top


Connection Layer
10K+ Concurrent Clients
WebSocket Load Balancer Sticky sessions
WebSocket Server Pool Node.js clusters
Board Rooms
Room Manager Board-based grouping
Presence Tracker User activity
Message Router Targeted delivery
Scaling Strategy
Horizontal Scaling Add server instances
Redis Adapter Cross-server messaging
Sticky Sessions User-server affinity
Graceful Failover Connection migration
Performance Optimization
Message Batching Reduce network calls
Message Compression Reduce bandwidth
Heartbeat System Connection health
Rate Limiting Abuse prevention

Database Optimization

□ Back to Top

Event Sourcing for Operations □ Back to Top


Event Store
Event Stream Append-only log
Partitioning By board_id
Snapshots Periodic state capture
Log Compaction Remove obsolete events
Read Models
Board View Current state projection
Activity View Timeline projection
Search Index Full-text search
Analytics View Metrics projection
CQRS Pattern
Command Side Write operations
Query Side Read operations
Event Bus Async projection updates
Eventual Consistency Async updates

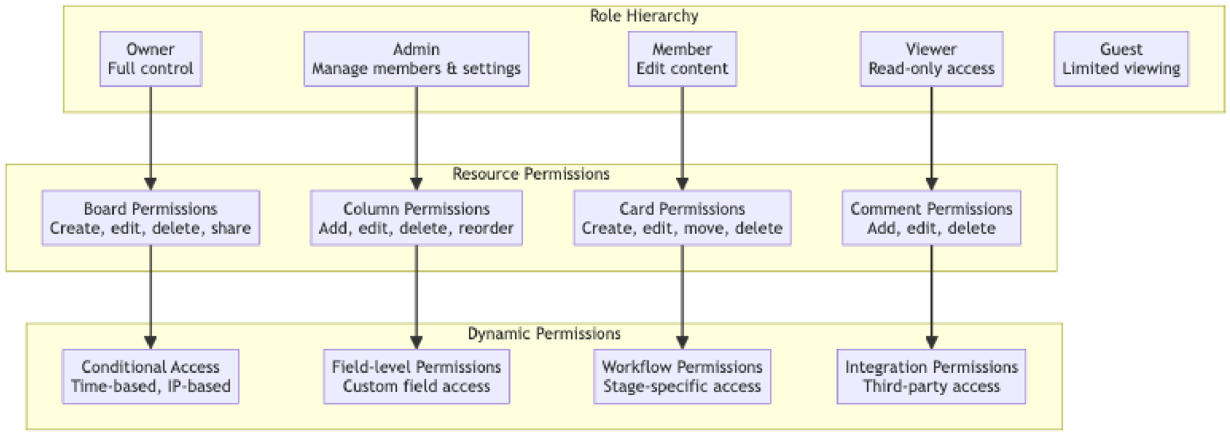
Security and Privacy

□ Back to Top

Collaborative Security Model

□ Back to Top

Permission System □ Back to Top

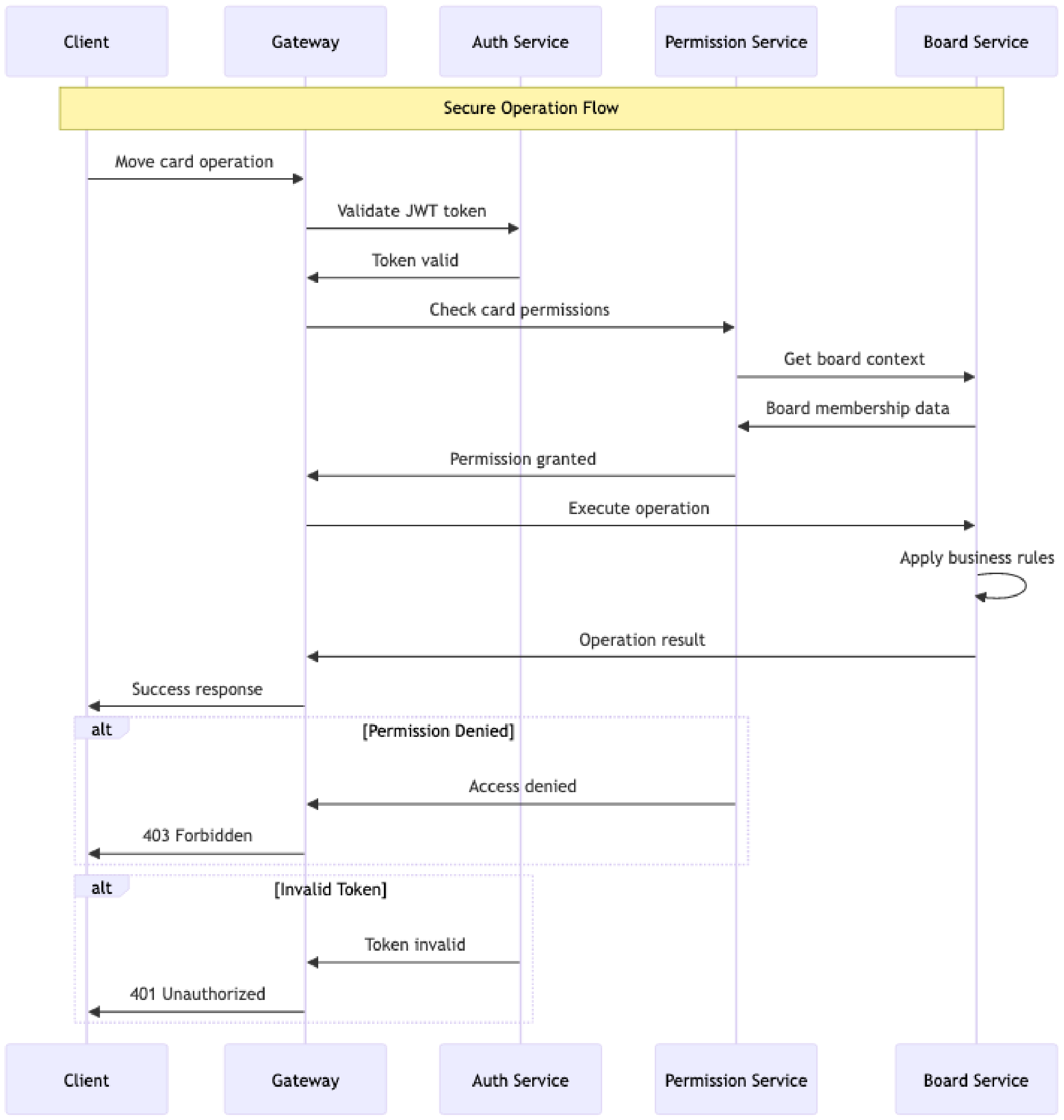


Data Protection

□ Back to Top

Real-time Data Security

□ Back to Top



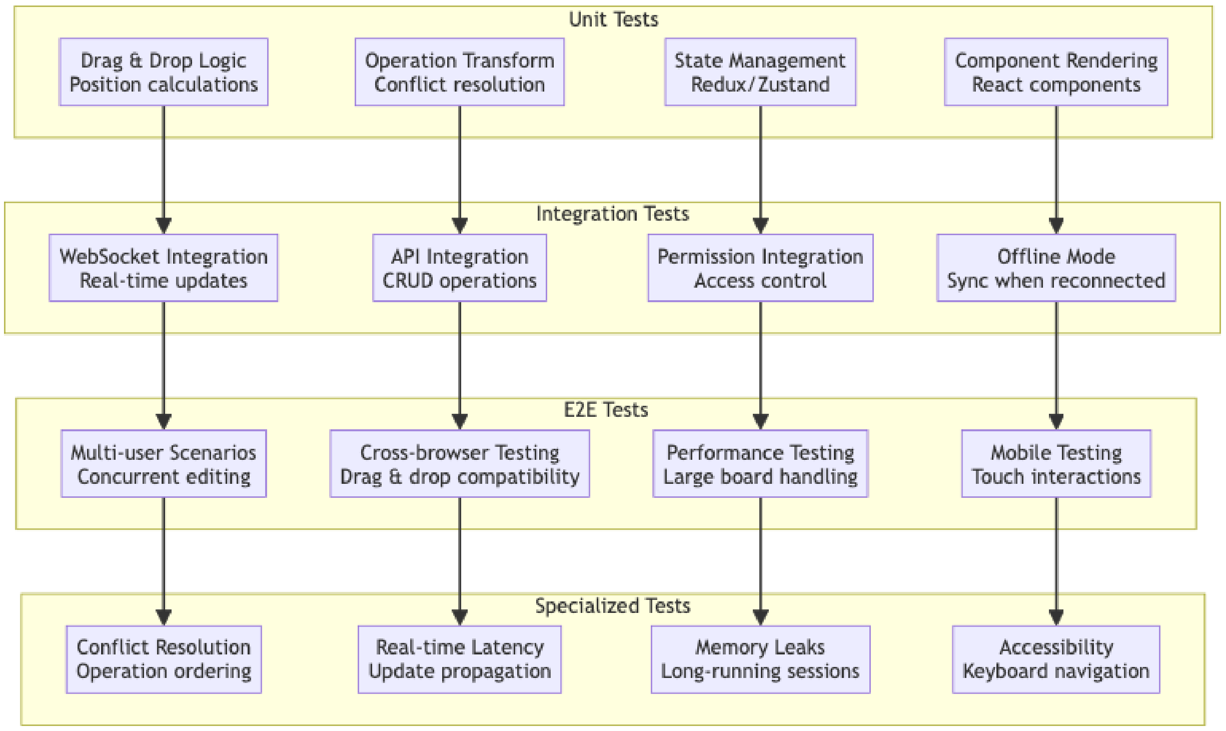
Testing, Monitoring, and Maintainability

☐ Back to Top

Testing Strategy

☐ Back to Top

Collaborative Feature Testing ☐ Back to Top


Unit Tests
Drag & Drop Logic Position calculations
Operation Transform Conflict resolution
State Management Redux/Zustand
Component Rendering React components
Integration Tests
WebSocket Integration Real-time updates
API Integration CRUD operations
Permission Integration Access control
Offline Mode Sync when reconnected
E2E Tests
Multi-user Scenarios Concurrent editing
Cross-browser Testing Drag & drop compatibility
Performance Testing Large board handling
Mobile Testing Touch interactions
Specialized Tests
Conflict Resolution Operation ordering
Real-time Latency Update propagation
Memory Leaks Long-running sessions
Accessibility Keyboard navigation

Trade-offs, Deep Dives, and Extensions

☐ Back to Top

Operational Transform vs CRDT

☐ Back to Top

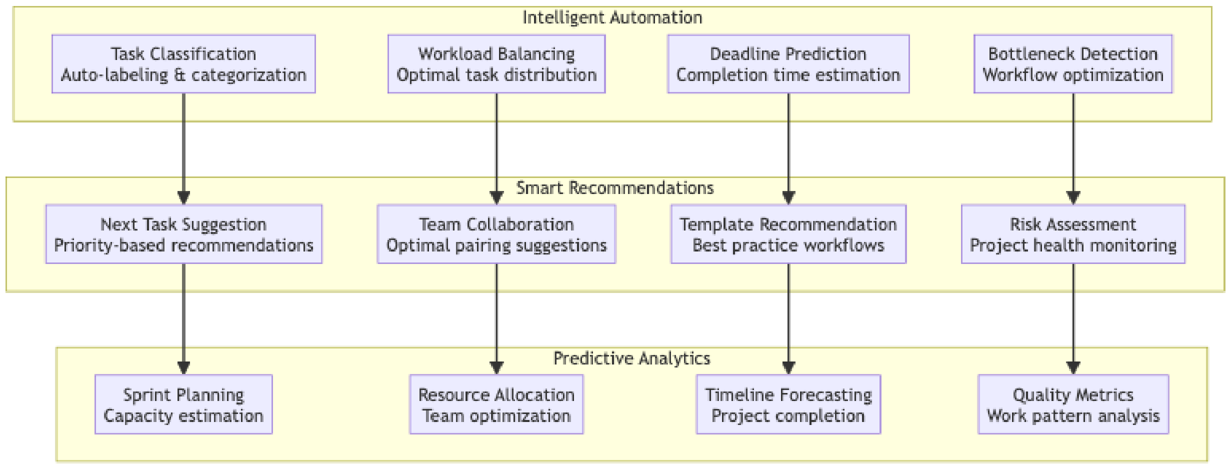
Aspect	Operational Transform	CRDT (Conflict-free Replicated Data Types)
Complexity	High implementation	Moderate implementation
Performance	Good for small ops	Excellent for concurrent ops
Memory Usage	Low overhead	Higher memory usage
Conflict Resolution	Manual transform logic	Automatic convergence

Aspect	Operational Transform	CRDT (Conflict-free Replicated Data Types)
Undo/Redo	Complex implementation	Very difficult
Network Usage	Efficient	Larger message size

Advanced Features

☐ Back to Top

AI-Powered Project Management ☐ Back to Top


Intelligent Automation
Task Classification Auto-labeling & categorization
Workload Balancing Optimal task distribution
Deadline Prediction Completion time estimation
Bottleneck Detection Workflow optimization
Smart Recommendations
Next Task Suggestion Priority-based recommendations
Team Collaboration Optimal pairing suggestions
Template Recommendation Best practice workflows
Risk Assessment Project health monitoring
Predictive Analytics
Sprint Planning Capacity estimation
Resource Allocation Team optimization
Timeline Forecasting Project completion
Quality Metrics Work pattern analysis

Future Extensions

☐ Back to Top

Next-Generation Collaboration Features ☐ Back to Top

---

1. **Immersive Collaboration**:
   - VR/AR board interfaces
   - 3D spatial organization
   - Gesture-based interactions
   - Voice-controlled operations
2. **Advanced AI Integration**:
   - Natural language task creation

    - Automated workflow optimization
    - Intelligent resource allocation
    - Predictive project analytics
3. **Enhanced Real-time Features**:
    - Live video collaboration
    - Shared cursors and annotations
    - Real-time co-editing
    - Synchronized presentations
4. **Integration Ecosystem**:
    - Deep tool integrations
    - Workflow automation
    - Custom app marketplace
    - API-first architecture

This comprehensive design provides a robust foundation for building a scalable, collaborative Kanban board system that handles real-time multi-user editing, maintains data consistency, and delivers excellent user experience across all platforms while supporting advanced project management workflows.