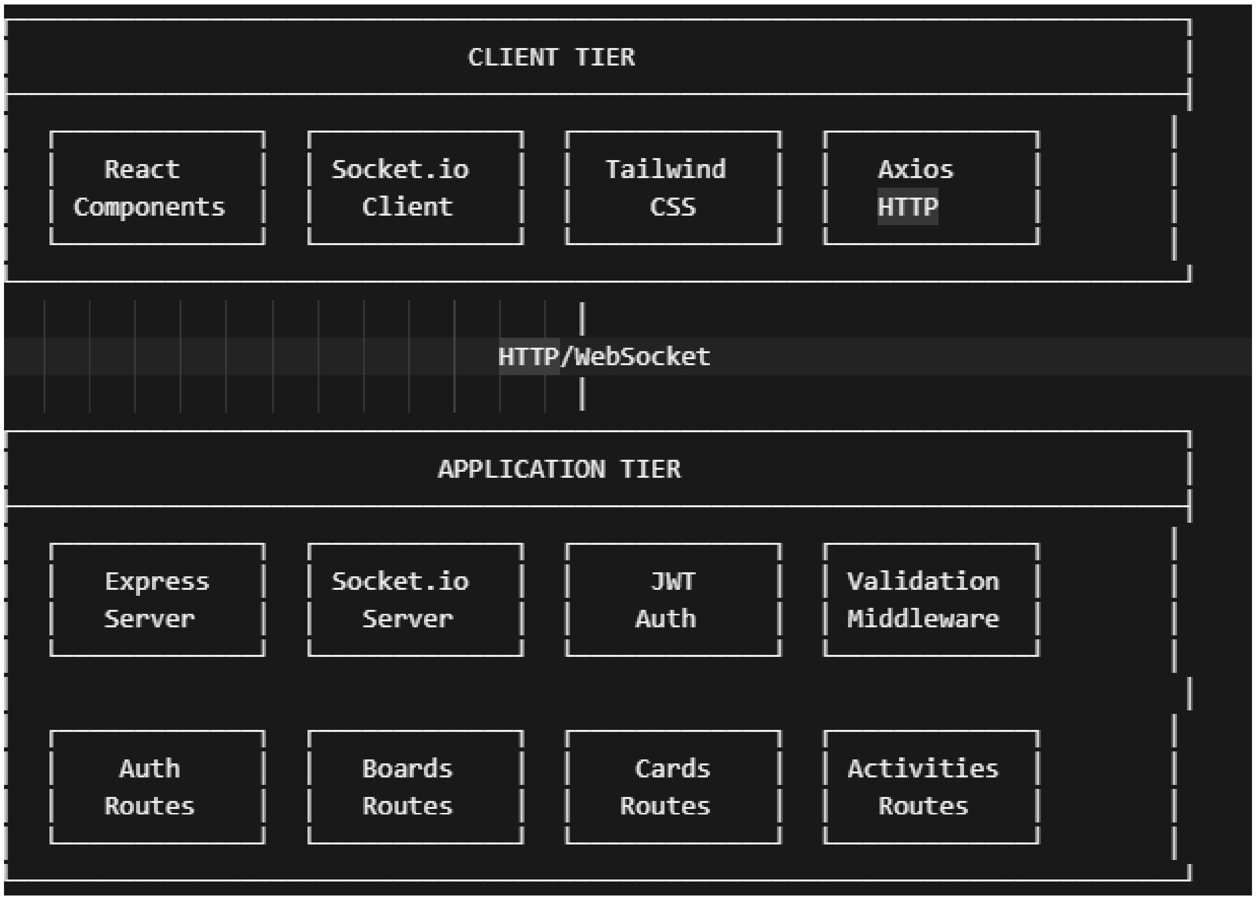
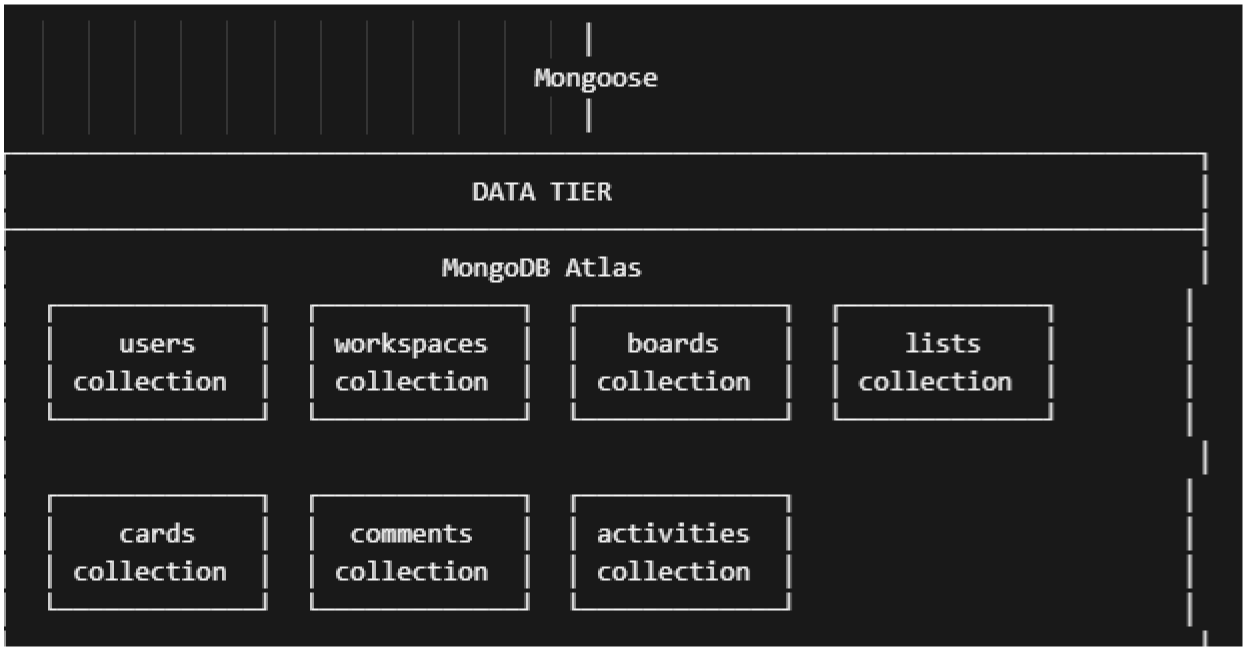
# High-Level Design (HLD) - Mini-Trello Kanban App

## System Architecture Overview

\`\`\`





```
                                    |
                                 Mongoose
                                    |
┌─────────────────────────────────────────────────────────────────────────┐
│                             DATA TIER                                   │
├─────────────────────────────────────────────────────────────────────────┤
│                           MongoDB Atlas                                 │
│  ┌────────────┐  ┌────────────┐  ┌────────────┐  ┌────────────┐         │
│  │   users    │  │ workspaces │  │   boards   │  │   lists    │         │
│  │ collection │  │ collection │  │ collection │  │ collection │         │
│  └────────────┘  └────────────┘  └────────────┘  └────────────┘         │
│                                                                         │
│  ┌────────────┐  ┌────────────┐  ┌────────────┐                         │
│  │   cards    │  │  comments  │  │ activities │                         │
│  │ collection │  │ collection │  │ collection │                         │
│  └────────────┘  └────────────┘  └────────────┘                         │
```

## Major Components

### 1. Frontend Layer (React SPA)

Core Components:
- **Authentication System**: Login/Register forms with JWT token management
- **Dashboard**: Board listing and workspace management
- **Kanban Board**: Drag-and-drop interface with real-time updates
- **Card Management**: Detailed card modals with comments and attachments
- **Real-time Client**: Socket.io integration for live collaboration

State Management:
- **React Context**: Global auth state and user information
- **Local State**: Board data, UI state, and temporary form data
- **Optimistic Updates**: Immediate UI feedback with server reconciliation

### 2. Backend Layer (Node.js/Express)

API Services:
- **Authentication Service**: JWT-based auth with bcrypt password hashing
- **Board Service**: CRUD operations for boards, lists, and cards
- **Real-time Service**: Socket.io server for live collaboration
- **Activity Service**: Audit logging and activity feed generation
- **Search Service**: Full-text search across cards and boards

Middleware Stack:
- **Security**: Helmet, CORS, rate limiting
- **Authentication**: JWT verification middleware
- **Validation**: Express-validator for input sanitization
- **Error Handling**: Centralized error handling and logging

### 3. Database Layer (MongoDB)

Data Models:
- **Users**: Authentication and profile information
- **Workspaces**: Organizational containers with member management
- **Boards**: Project boards with settings and member permissions
- **Lists**: Ordered columns within boards
- **Cards**: Task items with rich metadata
- **Comments**: Threaded discussions on cards
- **Activities**: Audit trail of all system actions

## Data Flow Architecture

### 1. Authentication Flow
\`\`\`
Client → Login Request → Express Auth Route → JWT Generation →
Client Storage → Authenticated Requests → JWT Verification → Protected Resources
\`\`\`

### 2. Real-time Collaboration Flow
\`\`\`
User Action → Optimistic UI Update → API Request → Database Update →
Socket.io Broadcast → Other Clients → UI Synchronization
\`\`\`

### 3. Drag & Drop Flow
\`\`\`
Drag Start → Calculate New Position → Optimistic UI Update →
API Request → Position Validation → Database Update →
Real-time Broadcast → Conflict Resolution
\`\`\`

## Real-time Communication Strategy

### WebSocket vs Server-Sent Events Decision

Chosen: WebSocket (Socket.io)

Rationale:
- **Bidirectional Communication**: Needed for typing indicators and presence
- **Room-based Broadcasting**: Efficient board-specific updates
- **Automatic Fallbacks**: Socket.io handles connection failures gracefully
- **Event-based Architecture**: Clean separation of different update types

Implementation:
- **Board Rooms**: Users join/leave rooms per board for isolated updates
- **Event Types**: card-moved, card-updated, comment-added, user-presence
- **Conflict Resolution**: Last-write-wins with optimistic UI updates
- **Connection Management**: Automatic reconnection with state synchronization

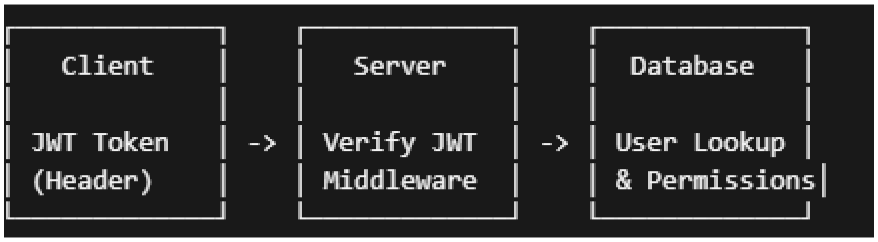
### Performance Optimizations

Database Level:
- **Indexing Strategy**: Compound indexes on frequently queried fields
- **Connection Pooling**: Mongoose connection pool optimization
- **Query Optimization**: Aggregation pipelines for complex queries
- **Data Archiving**: Automated archiving of old activities and boards

Application Level:
- **Caching Layer**: Redis for session storage and frequent queries
- **Response Compression**: Gzip compression for API responses
- **Request Batching**: Bulk operations for multiple card updates
- **Background Jobs**: Async processing for non-critical operations

Frontend Level:
- **Code Splitting**: Lazy loading of board components
- **Virtual Scrolling**: Efficient rendering of large card lists
- **Debounced Updates**: Reduced API calls for rapid user actions
- **Service Worker**: Offline capability and background sync

## Security Architecture

### Authentication & Authorization
\`\`\`

```
┌─────────────┐    ┌─────────────┐    ┌─────────────┐
│   Client    │    │   Server    │    │  Database   │
│             │    │             │    │             │
│ JWT Token   │ -> │ Verify JWT  │ -> │ User Lookup │
│ (Header)    │    │ Middleware  │    │ & Permissions│
└─────────────┘    └─────────────┘    └─────────────┘
```
\`\`\`

Security Measures:
- **Password Security**: bcrypt hashing with salt rounds
- **JWT Security**: Short expiration, secure signing algorithm
- **Input Validation**: Comprehensive validation and sanitization
- **Rate Limiting**: API endpoint protection against abuse
- **CORS Policy**: Strict origin validation
- **Security Headers**: Helmet.js for security headers

### Data Protection
- **Access Control**: Role-based permissions (owner, member, viewer)
- **Data Validation**: Server-side validation for all inputs
- **Audit Logging**: Comprehensive activity tracking
- **Error Handling**: Secure error messages without data leakage

Infrastructure Components:
- **Frontend**: Static hosting with CDN distribution
- **Backend**: Container-based deployment with auto-scaling
- **Database**: Managed MongoDB with automated backups
- **Monitoring**: Application performance monitoring and alerting
- **SSL/TLS**: End-to-end encryption for all communications

## Monitoring & Observability

### Key Metrics
- **Performance**: Response times, throughput, error rates
- **Business**: Active users, boards created, collaboration events
- **Infrastructure**: CPU, memory, database connections
- **Real-time**: WebSocket connections, message latency

### Logging Strategy
- **Application Logs**: Structured logging with correlation IDs
- **Access Logs**: Request/response logging with user context
- **Error Logs**: Comprehensive error tracking and alerting
- **Audit Logs**: Security-relevant events and data changes

This high-level design provides a scalable, secure, and maintainable architecture for the Mini-Trello application while supporting real-time collaboration and future growth requirements.