EpiTrello

Technical Project Specifications

Project Type: Collaborative Kanban Board Application

Development Period: October 20, 2025 – January 28, 2026

Team: Corentin Wolff & Basile Trebus-Hamann

Version: 1.0.0 | **Last Updated:** January 13, 2026

1 Project Overview

EpiTrello is a full-stack real-time collaborative project management application inspired by Trello, implementing the Kanban methodology. Built as a professional simulation project, it enables teams to organize work through boards containing ordered lists of task cards with drag-and-drop functionality, real-time synchronization via WebSockets, and comprehensive access control.

Project Context

Duration: 14 calendar weeks (October 20, 2025 – January 28, 2026) with development occurring 3 days per week (Monday–Wednesday), totaling approximately 39 working days and 312 planned development hours distributed across 7 two-week sprints.

Methodology: Agile/Scrum with Test-Driven Development (TDD), enforcing 90%+ code coverage thresholds via CI/CD pipelines, mandatory code reviews through pull requests, and continuous deployment to Fly.io cloud infrastructure.

Team Composition

Corentin Wolff – Full-stack development with focus on backend architecture, API design, database modeling, and WebSocket implementation.

Basile Trebus-Hamann – Full-stack development with focus on frontend architecture, React components, drag-and-drop system, and UI/UX implementation.

312h	7	90%+	45+
Development Hours	Sprint Cycles	Test Coverage	API Endpoints

2 Technical Architecture

System Architecture Overview

EpiTrello implements a modern three-tier client-server architecture with clear separation between presentation, business logic, and data persistence layers. The frontend React SPA communicates with the Express.js backend through RESTful HTTP endpoints for CRUD operations and Socket.IO WebSocket connections for real-time event propagation.

Technology Stack & Selection Rationale

UI Framework **React 19**	State Mgmt **Redux Toolkit**	Routing **React Router 7**	Styling **Tailwind CSS 3**
Drag & Drop **dnd-kit**	HTTP Client **Axios**	Build Tool **Vite 7**	WS Client **Socket.IO Client**
Runtime **Node.js (ESM)**	Framework **Express 4**	Real-time **Socket.IO 4**	Database **MongoDB Atlas**
ODM **Mongoose 8**	Auth **JWT + bcrypt**	Validation **Zod**	Email **Nodemailer**

Technology	Selection Rationale	Alternatives Considered
React 19	Component-based architecture enables reusable UI elements critical for Kanban boards (cards, lists, modals). Massive ecosystem with battle-tested libraries for routing, state management, and drag-and-drop. React 19's concurrent features improve perceived performance during complex drag operations. Strong TypeScript support and extensive documentation reduce onboarding time.	Vue.js (smaller ecosystem for complex DnD), Svelte (less mature tooling), Angular (steeper learning curve, overkill for project scope)
Redux Toolkit	Centralized state management essential for synchronizing board/list/card data across components and WebSocket events. createAsyncThunk simplifies API calls with loading/error states. Predictable state updates via reducers make debugging real-time sync issues straightforward. DevTools enable time-travel debugging critical for complex UI states.	Zustand (simpler but less structured for large apps), React Context (insufficient for complex cross-component state), MobX (reactive approach harder to debug)
@dnd-kit	Modern drag-and-drop library with first-class accessibility support (keyboard navigation, screen reader announcements). Modular architecture allows combining sortable lists with cross-container drops. Better performance than	react-beautiful-dnd (deprecated, no longer maintained), react-dnd (lower-level API requiring more boilerplate),

	alternatives through optimized rendering. Active maintenance and TypeScript-first design.	native HTML5 DnD (poor cross-browser support, no animations)
Tailwind CSS 3	Utility-first approach enables rapid UI prototyping without context-switching to CSS files. Purging removes unused styles resulting in minimal production bundles (~10KB). Built-in dark mode support via variant classes. Consistent design system through configuration. JIT compiler provides instant feedback during development.	CSS Modules (more boilerplate), Styled Components (runtime overhead), plain CSS (inconsistent patterns, harder maintenance)
Vite 7	Near-instant dev server startup via native ES modules (vs. bundling everything upfront). Hot Module Replacement preserves application state during edits. Optimized production builds with Rollup. First-class React support with @vitejs/plugin-react. Significantly faster than webpack-based alternatives.	Create React App (slow builds, maintenance mode), webpack (complex configuration), Parcel (less ecosystem support)
Express 4	Minimal, unopinionated framework providing flexibility for REST API design. Extensive middleware ecosystem (helmet, cors, compression, morgan). Easy integration with Socket.IO on same HTTP server. Low learning curve enables faster development. Production-proven at scale.	Fastify (faster but smaller ecosystem), Koa (less middleware availability), NestJS (TypeScript-heavy, over-engineered for project scope), Hono (newer, less battle-tested)
Socket.IO 4	Automatic WebSocket connection with fallback to HTTP long-polling for unreliable networks. Built-in room system perfect for board-based broadcasting. Automatic reconnection with configurable backoff. Client libraries for React integration. Handles connection state and heartbeats automatically.	ws (too low-level, manual room management), Pusher (third-party dependency, cost), Server-Sent Events (unidirectional only)
MongoDB Atlas	Document model naturally fits hierarchical board→list→card structure without complex joins. Flexible schema accommodates evolving card properties (labels, checklists, comments). Text indexes enable full-text search. Free M0 tier sufficient for project scope. Atlas provides managed backups, monitoring, and scaling.	PostgreSQL (rigid schema requires migrations for card property changes), MySQL (same issues), Firebase (vendor lock-in, less query flexibility)
Mongoose 8	Schema validation ensures data integrity at application layer. Middleware hooks enable pre/post save logic (password hashing, activity logging). Population simplifies reference resolution. TypeScript support via schema inference. Mature ecosystem with extensive documentation.	Native MongoDB driver (no schema validation), Prisma (better for SQL, MongoDB support less mature), TypeORM (primarily SQL-focused)
JWT + bcrypt	Stateless JWT authentication eliminates server-side session storage, simplifying horizontal scaling. bcrypt's adaptive hashing protects against brute-force attacks. Token-based auth works seamlessly with both REST API and WebSocket connections. Industry-standard approach with extensive security auditing.	Session-based auth (requires sticky sessions or shared storage), OAuth only (overkill for internal auth), Passport.js (abstraction adds complexity without benefit here)
Zod	TypeScript-first schema validation with excellent type inference. Single source of truth for request validation and TypeScript types. Composable schemas for complex nested objects (cards with labels, checklists). Better error messages than alternatives. Zero dependencies.	Joi (JavaScript-first, weaker TS support), Yup (less performant, API inconsistencies), express-validator (middleware-coupled, less reusable)

Full-Stack JavaScript Decision: Using JavaScript/Node.js across frontend and backend enables code sharing (validation schemas, type definitions), reduces context-switching, and allows both team members to work on any layer. The ecosystem's package availability (npm) and community support accelerate development within the 14-week timeline.

Frontend Architecture

The React 19 SPA uses a feature-based architecture where each domain (auth, boards, cards, lists, notifications) is encapsulated as a Redux Toolkit slice with its async thunks, selectors, and related components. The application implements optimistic UI updates for responsive interactions, with Socket.IO listeners syncing remote changes to local Redux state.

Key Patterns: Feature slices with createAsyncThunk, React Router 7 nested layouts with protected routes, custom hooks for socket management (useBoardSocket), dnd-kit sensors for accessible keyboard/pointer drag operations.

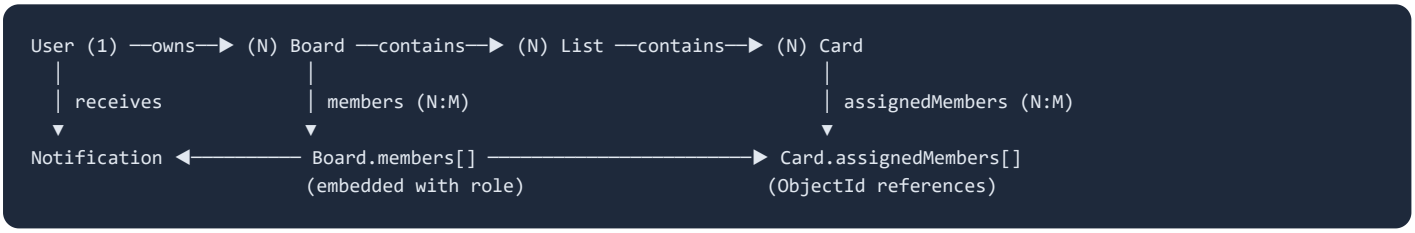
Backend Architecture

The Express.js server follows a layered MVC architecture with clear separation: Routes define endpoints and apply middleware, Controllers handle HTTP request/response cycles and orchestrate business logic, Models encapsulate Mongoose schemas with validation and indexing.

Middleware Pipeline: Helmet (security headers) → CORS → Compression → Body Parser → Morgan (logging) → JWT Authentication → Route Handlers → Error Handler. Socket.IO runs on the same HTTP server with JWT-authenticated connections and room-based broadcasting.

3 Data Model Design

Entity Relationships



Model	Key Fields	Indexes	Relationships
User	username (unique, 3-50 chars), email (unique, lowercase), password (bcrypt hash, min 12 chars pre-hash), avatarUrl (base64 data URL), passwordResetToken/Expires	email (unique), username (unique)	Owns boards, member of boards, receives notifications, assigned to cards
Board	title (max 120 chars), description, owner (User ref), members[] (embedded: user ref + role enum), background (type: color|image, value, thumbnail), activity[] (embedded log)	owner, members.user, text(title, description)	Contains lists, has owner (1:1), has members (N:M embedded)
List	title (max 120 chars), board (Board ref), position (Number, 0-indexed), archived (Boolean, soft delete)	board, compound {board, position}	Belongs to board (N:1), contains cards (1:N)
Card	title (max 120 chars), description, list (List ref), position, labels[] (color + text), dueDate, checklist[] (text + completed), assignedMembers[] (User refs), comments[] (embedded), activity[] (embedded), archived	list, compound {list, position}, text(title, description)	Belongs to list (N:1), assigned members (N:M)
Notification	recipient (User ref), type (card_assigned|mention|comment), title (max 200), message (max 500), board/card/actor (refs), read (Boolean)	recipient, read, compound {recipient, read, createdAt}	Sent to user (N:1), references board/card/actor

Design Decisions: Member roles are embedded in Board.members[] for atomic permission checks without joins. Activity logs are embedded for read performance but should consider TTL indexes for production scale. Card comments include author reference for @mention notifications. Text indexes enable full-text search across boards and cards.

4 REST API Reference

Base URLs: Development: `http://localhost:5000/api` | Production: `https://epitrello-backend.fly.dev/api`

Authentication: All protected endpoints require `Authorization: Bearer <jwt_token>` header. Tokens are valid for 7 days and obtained via /auth/login or /auth/register.

🔐 Authentication Endpoints

POST	`/auth/register`	Create account with username, email, password (min 12 chars, strength validated via zxcvbn)
POST	`/auth/login`	Authenticate with email/password, returns JWT token and user object
GET	`/auth/me`	Get current authenticated user profile from token
POST	`/auth/forgot-password`	Request password reset email (always returns success to prevent enumeration)
POST	`/auth/reset-password`	Reset password using token from email link

👤 User Endpoints

GET	`/users/profile`	Get current user's full profile
PUT	`/users/profile`	Update profile (multipart/form-data for avatar upload, max 2MB JPEG/PNG/GIF/WebP)
PUT	`/users/password`	Change password (requires current password verification)
GET	`/users/search?q=`	Search users by username/email (min 2 chars, max 10 results, excludes self)

📋 Board Endpoints

POST	`/boards`	Create board with title, description, background (color hex or image URL)
GET	`/boards`	List all boards where user is owner or member (includes membershipRole)
GET	`/boards/:id`	Get specific board with members populated
PATCH	`/boards/:id`	Update board (admin/owner only)
DELETE	`/boards/:id`	Delete board and all lists/cards (owner only)
GET	`/boards/:id/members`	Get detailed member list with user profiles and roles
POST	`/boards/:id/members`	Add member with role (admin/owner only, roles: admin|member|viewer)
PATCH	`/boards/:id/members/:userId`	Update member role (admin/owner only)
DELETE	`/boards/:id/members/:userId`	Remove member (admin/owner or self-removal)
GET	`/boards/:id/activity`	Get paginated activity history (limit, before cursor)

📝 List Endpoints

POST	`/lists`	Create list with title, board ID, optional position
GET	`/lists?board=`	Get all lists for a board, sorted by position
GET	`/lists/:id`	Get specific list

PATCH	`/lists/:id`	Update list title, position, or archived status
DELETE	`/lists/:id`	Delete list and all contained cards
POST	`/lists/reorder`	Batch reorder multiple lists by position

🎴 Card Endpoints

POST	`/cards`	Create card with title, list ID, optional description/position
GET	`/cards?list=`	Get all cards for a list, sorted by position
GET	`/cards/:id`	Get card with full details (comments, activity, members populated)
PATCH	`/cards/:id`	Update card fields (title, description, labels, dueDate, checklist, etc.)
DELETE	`/cards/:id`	Delete card
POST	`/cards/:id/move`	Move card to different list and/or position
POST	`/cards/:id/comments`	Add comment to card (triggers notifications for @mentions)
DELETE	`/cards/:id/comments/:commentId`	Delete comment (author only)
POST	`/cards/:id/members`	Assign member to card (triggers notification)
DELETE	`/cards/:id/members/:userId`	Unassign member from card

🔔 Notification Endpoints

GET	`/notifications`	Get user's notifications (supports ?unreadOnly=true)
PATCH	`/notifications/:id/read`	Mark notification as read
POST	`/notifications/read-all`	Mark all notifications as read

5 Real-Time WebSocket Events

Connection: Socket.IO client connects to `ws://localhost:5000` (dev) or `wss://epitrello-backend.fly.dev` (prod) with JWT token in auth object. Supports automatic reconnection with exponential backoff (1s–5s delay, max 10 attempts).

Room System: Users automatically join personal room (`user:{userId}`) for notifications. Board rooms (`board:{boardId}`) require explicit join via `board:join` event.

Board Room Events

- `board:join` / `board:leave` – Client joins/leaves board room
- `board:joined` – Server confirms join with activeUsers list
- `board:user-joined` / `board:user-left` – User presence changes
- `board:updated` – Board settings changed (title, background, etc.)
- `board:deleted` – Board was deleted, clients should navigate away
- `board:member-added` / `board:member-removed` – Membership changes

List & Card Events

- `list:created` / `list:updated` / `list:deleted` – List CRUD operations
- `list:reordered` – Multiple lists repositioned
- `card:created` / `card:updated` / `card:deleted` – Card CRUD operations
- `card:moved` – Card moved between lists or repositioned
- `card:comment-added` – New comment on card
- `card:member-assigned` / `card:member-unassigned` – Assignment changes

Event Flow Example: Card Move Operation

1. **Client A** drags card from List 1 to List 2 → dnd-kit fires onDragEnd → dispatch moveCard async thunk
2. **HTTP Request:** POST /api/cards/:id/move with {targetListId, position} → Server updates Card.list and Card.position in MongoDB
3. **Server broadcasts** `card:moved` event to board room with sourceListId, targetListId, updated card position
4. **Clients B, C** receive socket event → Redux listener dispatches action to update local card state in both lists
5. **Client A** receives HTTP response → optimistic update confirmed or rolled back on error

6 Functional Features & User Stories

User Stories by Epic

Epic	User Story	Priority	Sprint
Authentication	As a visitor, I want to register with email and password so that I can create my own boards and collaborate with others.	Must Have	Sprint 2
	As a user, I want to log in with my credentials so that I can access my boards and continue my work.	Must Have	Sprint 2
	As a user, I want to reset my password via email so that I can recover access if I forget my credentials.	Should Have	Sprint 2
	As a user, I want to update my profile (username, avatar) so that team members can identify me on shared boards.	Should Have	Sprint 2
	As a user, I want to change my password from settings so that I can maintain account security.	Should Have	Sprint 2
Board Management	As a user, I want to create a new board with a title and background so that I can organize a new project.	Must Have	Sprint 2
	As a user, I want to see all my boards (owned and shared) on a dashboard so that I can quickly access any project.	Must Have	Sprint 2
	As a board owner, I want to edit board title, description, and background so that I can customize the workspace.	Must Have	Sprint 2
	As a board owner, I want to delete a board so that I can remove completed or abandoned projects.	Must Have	Sprint 2
	As a board owner/admin, I want to invite users by searching their username/email so that I can build my team.	Must Have	Sprint 5
	As a board owner, I want to assign roles (admin, member, viewer) to members so that I can control access levels.	Should Have	Sprint 5
List Management	As a board member, I want to create lists (e.g., "To Do", "In Progress", "Done") so that I can organize workflow stages.	Must Have	Sprint 3
	As a board member, I want to rename lists inline so that I can adjust my workflow without interruption.	Must Have	Sprint 3
	As a board member, I want to drag lists horizontally to reorder them so that I can reflect my workflow sequence.	Must Have	Sprint 4
	As a board member, I want to archive/delete lists so that I can clean up unused workflow stages.	Should Have	Sprint 3
Card Management	As a board member, I want to create cards with a title so that I can add tasks to my lists.	Must Have	Sprint 3
	As a board member, I want to open a card modal to add description, due date, and labels so that I can provide task details.	Must Have	Sprint 3
	As a board member, I want to drag cards vertically within a list so that I can prioritize tasks.	Must Have	Sprint 4
	As a board member, I want to drag cards between lists so that I can move tasks through workflow stages.	Must Have	Sprint 4

	As a board member, I want to create checklists on cards so that I can break tasks into subtasks.	Should Have	Sprint 3
	As a board member, I want to add comments to cards so that I can discuss tasks with team members.	Should Have	Sprint 3
	As a board member, I want to assign members to cards so that responsibility is clear.	Must Have	Sprint 5
	As a board member, I want to @mention users in comments so that I can notify specific team members.	Should Have	Sprint 6
Real-Time Collaboration	As a board member, I want to see changes made by others instantly so that I'm always working with current data.	Must Have	Sprint 6
	As a board member, I want to see who else is viewing the board so that I know who's online.	Should Have	Sprint 6
	As a user, I want to receive notifications when I'm assigned to a card or @mentioned so that I don't miss important updates.	Must Have	Sprint 6
	As a user, I want to see a notification badge and center so that I can review all my alerts in one place.	Should Have	Sprint 6
Bonus Features	As a user, I want to search cards by title/description so that I can quickly find specific tasks.	Could Have	Sprint 6+
	As a board member, I want to view board activity history so that I can track what changed and when.	Could Have	Sprint 5
	As a user, I want to toggle dark mode so that I can reduce eye strain during night work sessions.	Could Have	Sprint 6+

Feature Implementation Summary

Feature Domain	Implemented Capabilities
User Management	Secure registration with password strength validation (zxcvbn library enforcing 12+ char passwords with complexity requirements), JWT-based stateless authentication with 7-day token expiry, email-based password reset flow with cryptographically secure tokens (SHA-256 hashed, 1-hour expiry), profile management with avatar upload (base64-encoded, max 2MB, JPEG/PNG/GIF/WebP formats), user search for board member invitations.
Board Management	Full CRUD operations on boards with customizable backgrounds (solid colors via hex codes or image URLs with thumbnails), role-based access control with four permission levels (owner: full control including deletion; admin: manage members and settings; member: create/edit content; viewer: read-only access), board activity history with paginated retrieval and cursor-based pagination, member invitation system with role assignment.
List Management	Create, rename, archive, and delete lists within boards, drag-and-drop horizontal reordering using dnd-kit with position persistence, batch reorder API for atomic multi-list position updates, soft-delete via archived flag for potential restoration.
Card Management	Full card lifecycle management with rich content: titles (max 120 chars), markdown-compatible descriptions, color-coded labels with optional text, due dates with visual indicators, checklists with completion tracking, member assignment (triggers notifications), comments with @mention support (parses usernames and creates notifications), card activity log tracking all modifications.
Drag & Drop	Accessible drag-and-drop powered by @dnd-kit with keyboard navigation support, card reordering within lists (vertical), card movement between lists (cross-list drops), list reordering (horizontal), visual feedback with drag overlays, placeholders, and smooth animations, optimistic UI updates with server reconciliation.
Real-Time Sync	Socket.IO-based bidirectional communication with JWT-authenticated connections, room-based event broadcasting (per-board rooms for collaborative updates, per-user rooms for notifications), active user

	presence indicators showing who's viewing the board, automatic reconnection with exponential backoff, live updates for all CRUD operations propagated to connected clients within 500ms.
Notifications	Push notifications for card assignments, @mentions in comments, and comments on assigned cards, real-time delivery via WebSocket to user's personal room, mark-as-read functionality (individual and bulk), unread count badge in navigation, notification center with links to relevant cards/boards.

Bonus Features (Implemented): Full-text search across cards and boards using MongoDB text indexes, board activity history with actor/action/entity tracking, dark mode support via Tailwind CSS dark variant classes, avatar upload with client-side preview and server-side validation.

7 Security & Performance

Authentication & Authorization

Password Security: Passwords hashed using bcrypt with salt rounds, minimum 12 characters enforced, strength validated via zxcvbn to reject weak/common passwords. Password reset tokens are SHA-256 hashed before storage with 1-hour expiration.

JWT Implementation: Stateless authentication using jsonwebtoken library with configurable secret (minimum 16 characters enforced via Zod), 7-day expiration, tokens validated on every protected request via auth middleware.

RBAC: Four-tier role system (owner > admin > member > viewer) enforced at controller level, permission checks before all mutating operations, self-removal allowed for members leaving boards.

Request Security

Helmet Middleware: Sets security headers including Content-Security-Policy, X-Content-Type-Options, X-Frame-Options, Strict-Transport-Security (HSTS in production).

CORS Configuration: Whitelist-based origin validation, credentials mode enabled for cookie/auth header transmission, configured via CLIENT_URL environment variable.

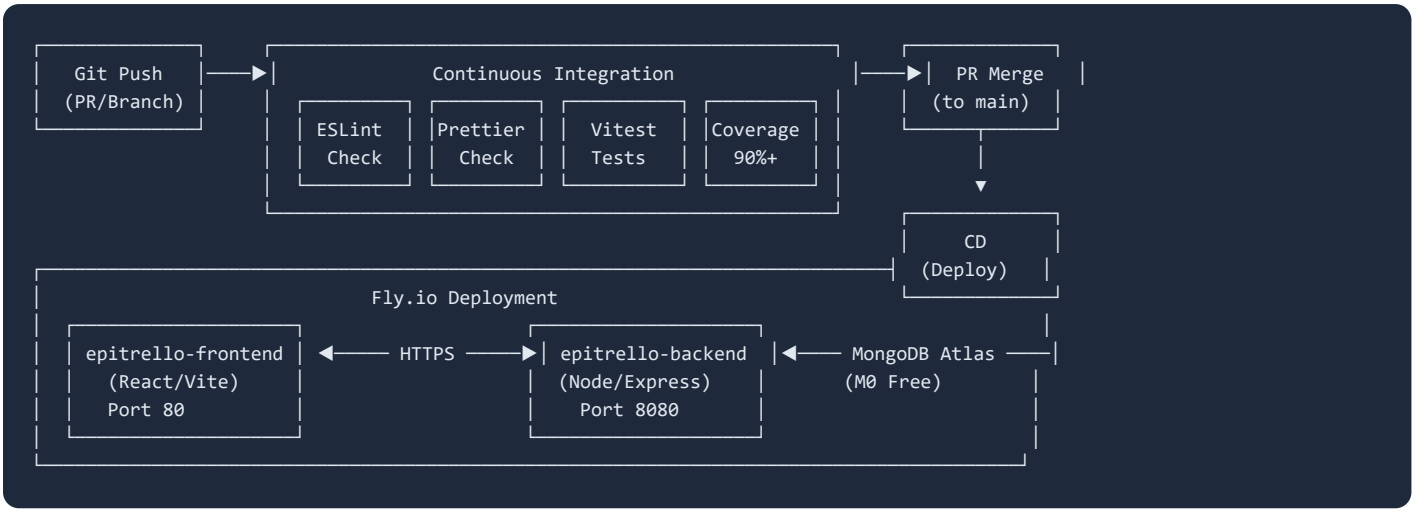
Input Validation: All request bodies validated using Zod schemas at API boundary, MongoDB injection prevented via Mongoose type coercion, file uploads restricted by type and size (multer configuration).

Performance Targets & Optimizations

Metric	Target	Implementation
Initial Page Load	< 3 seconds	Vite code splitting with React.lazy, Tailwind CSS purging (removes unused styles), gzip compression via Express middleware, CDN-ready static build
API Response Time	< 200ms (p95)	MongoDB compound indexes on frequently queried fields, Mongoose lean() queries where virtuals not needed, efficient projections limiting returned fields
Real-Time Latency	< 500ms	Socket.IO with WebSocket transport (fallback to polling), room-based broadcasting to minimize event scope, optimistic UI updates on client
Concurrent Users	10+ per board	Stateless JWT auth (no server-side sessions), MongoDB connection pooling, Fly.io auto-scaling machines

8 DevOps & Infrastructure

CI/CD Pipeline (GitHub Actions)



Local Development Environment

Docker Compose: Three-service orchestration with MongoDB container (localhost:27017), backend API container (localhost:5000 with hot reload via --watch flag), and frontend dev server (localhost:5173 with Vite HMR).

Environment Variables: Backend requires MONGODB_URI, JWT_SECRET (validated via Zod schema); optional SMTP_* vars for email functionality. Frontend uses VITE_API_URL (must be prefixed with VITE_ for client exposure).

Production Deployment (Fly.io)

Infrastructure: Paris region (cdg) deployment, auto-scaling machines, health checks on /api/health endpoint. Frontend served as static build via nginx-equivalent. Backend runs Node.js with production optimizations.

Secrets Management: Fly.io secrets for MONGODB_URI (Atlas connection string with 0.0.0.0/0 network access), JWT_SECRET (32+ chars), optional SMTP configuration for password reset emails.

Code Quality Tooling

Tool	Configuration	Enforcement
ESLint 9	Flat config (eslint.config.js), React hooks plugin, React refresh plugin for HMR compatibility	CI fails on any warning (--max-warnings=0), pre-commit hook via Husky
Prettier 3	Tailwind CSS plugin for class sorting, consistent formatting across JS/JSX/JSON/MD	CI format check, auto-format on staged files via lint-staged
Vitest 2	React Testing Library + MSW for frontend, Supertest + mongodb-memory-server for backend	90% coverage thresholds (statements/lines), 80% branches/functions
Husky	Pre-commit hooks running lint-staged (ESLint + Prettier on staged files)	Blocks commits failing lint/format checks

9 Development Schedule

Project Management: GitHub Projects board with Kanban workflow (Backlog → Sprint Backlog → In Progress → Review → Done). Issues labeled by sprint with assignees and linked pull requests.

Team: 2 developers | **Methodology:** Scrum with 2-week sprints | **Work Distribution:** Backend/Frontend split per sprint

14	39	312h	48h
Calendar Weeks	Working Days	Total Hours	Per Sprint

Sprint 1: Project Setup 48h

Oct 21-23 & Oct 28-30 (6 days)

Environment configuration (Node.js ESM, MongoDB, React 19 with Vite), Git repository initialization with monorepo structure, Docker Compose for local development, GitHub Actions CI/CD pipeline setup, ESLint/Prettier/Husky code quality toolchain, MongoDB schema design for Users/Boards/Lists/Cards/Notifications, REST API architecture planning with endpoint definitions, Redux Toolkit store configuration with feature slices.

Sprint 2: Authentication & Boards 48h

Nov 4-6 & Nov 11-13 (6 days)

Planned Tasks: User Login, User Logout, View User Profile, Create Board, View All Boards, View Board Details, Password security enforcement

User registration API with Zod validation and zxcvbn password strength checking, JWT-based login system with 7-day token expiry, authentication middleware for protected routes, React auth components (registration/login forms with validation feedback), user profile view and edit functionality with avatar upload, Board CRUD API endpoints with owner assignment, dashboard page with board listing and creation modal, board background customization (color picker and image URL).

Sprint 3: Lists & Cards 48h

Nov 18-20 & Nov 25-27 (6 days)

Planned Tasks: Lists CRUD Operations (Backend), Cards CRUD Operations (Backend), Board View Page (Frontend), Board View Tests, View Card Details

User Stories: Edit Board, Delete Board, Create List, Edit List, Delete List, Create Card, View Card Details, Edit Card, Add Card Description, Delete Card

List CRUD API with position-based ordering and board association, Card CRUD API with nested resources (labels, checklist, comments, activity), board view page with horizontal list layout, list component with inline title editing and card container, card component with preview (title, labels, due date indicators, member avatars), card detail modal with tabbed sections (description, checklist, comments, activity), rich text description editor with markdown preview.

Sprint 4: Drag & Drop 48h

Dec 2-4 & Dec 9-11 (6 days)

Planned Tasks: Lists DnD Backend, List DnD Frontend, Cards DnD within Lists (Backend + Frontend), Cards DnD between Lists (Backend + Frontend), DnD Tests

User Stories: Reorder Lists, Move Cards Between Lists, Reorder Cards Within Lists

@dnd-kit library integration with DndContext, sensors (pointer, keyboard), and collision detection, horizontal list drag-and-drop with SortableContext and position updates, vertical card drag within lists using sortable items, cross-list card movement with source/target list handling, visual feedback system (drag overlays, insertion placeholders, smooth animations), position persistence API endpoints with batch update support, optimistic UI updates with rollback on server error.

Sprint 5: Collaboration 48h

Dec 16-18 & Dec 23 + Jan 6-8 (6 days)

Planned Tasks: Password Recovery feature, Board Permission Levels implementation, Member Management system

User Stories: Edit User Profile, Share Board with Users, View Board Members, Remove Board Member, Assign Member to Card, Board Permission Levels

Board sharing API (add/remove/update members with role assignment), user search endpoint for member invitation with autocomplete UI, board members sidebar with role badges and removal capability, four-tier permission system (owner/admin/member/viewer) with controller-level enforcement, card member assignment API with notification triggers, member avatars display on card previews and detail modal, authorization middleware updates for role-based access control on all protected routes, password reset via email with secure tokens.

Sprint 6: Real-Time Features 48h

Jan 13-15 & Jan 20-22 (6 days)

User Stories: Real-Time Board Synchronization, Receive Notifications, See Active Users

WebSocket Tasks: Board UI updates automatically, No page refresh needed, Conflicts handled gracefully, Connection status indicator

Presence Tasks: Active user avatars displayed on board header, Users appear when they join, Users disappear when they leave/disconnect, Cursor/presence indicators (optional), Support for 10+ simultaneous users

Socket.IO server configuration with JWT authentication middleware, WebSocket connection management (user/board room join/leave events), real-time board synchronization events for all CRUD operations (board/list/card updates broadcast to room), active user presence system with join/leave notifications and avatar display, notification system backend (create notifications for assignments, mentions, comments) and frontend (notification center, unread badge, real-time delivery), connection status indicators with reconnection feedback.

Sprint 7: Testing & Delivery 24h

Jan 27-29 (3 days)

Unit test completion for critical components and utility functions (targeting 90%+ coverage), integration tests for all API endpoints using Supertest with mongodb-memory-server, E2E test scenarios for core user journeys (registration → board creation → card management), bug fixes and performance optimization based on test findings, technical documentation finalization (API reference, architecture guide, data models, WebSocket events), user guide creation with screenshots and workflow explanations, presentation preparation and project delivery.

Detailed Task Breakdown with Hour Estimates

Sprint	Task	Dev 1 (h)	Dev 2 (h)	Total
Sprint 1: Project Setup (48h)				
1	Git repo setup & monorepo structure	2	2	4
	Docker Compose configuration	3	3	6
	CI/CD pipeline (GitHub Actions)	4	4	8
	ESLint/Prettier/Husky setup	2	2	4
	MongoDB schema design (5 models)	6	6	12
	Backend/Frontend scaffolding & Redux store	7	7	14
Sprint 2: Authentication & Boards (48h)				
2	User registration API + Zod validation	4	–	4
	JWT login/logout + auth middleware	5	–	5
	React auth forms (login/register)	–	6	6

	User profile page + avatar upload	3	5	8
	Board CRUD API endpoints	6	–	6
	Dashboard + board creation modal	–	8	8
	Password strength validation (zxcvbn)	3	2	5
	Auth tests	3	3	6
Sprint 3: Lists & Cards (48h)				
3	List CRUD API + position ordering	6	–	6
	Card CRUD API + nested resources	8	–	8
	Board view page + horizontal list layout	–	8	8
	List component + inline editing	–	5	5
	Card component + preview display	–	6	6
	Card detail modal (description, checklist, comments)	4	7	11
	Lists & Cards tests	2	2	4
Sprint 4: Drag & Drop (48h)				
4	List reorder API (batch position update)	5	–	5
	Card move API (within + between lists)	6	–	6
	@dnd-kit integration + DndContext setup	–	6	6
	List horizontal DnD + SortableContext	–	7	7
	Card vertical DnD + cross-list movement	–	10	10
	Visual feedback (overlays, placeholders, animations)	–	6	6
	DnD integration tests	4	4	8
Sprint 5: Collaboration (48h)				
5	Board members API (add/remove/update role)	6	–	6
	Permission system (RBAC middleware)	8	–	8
	User search API for invitations	3	–	3
	Password reset flow (email + tokens)	–	8	8
	Member management UI (sidebar, avatars)	–	7	7
	Card assignment API + UI	4	4	8
	Collaboration tests	4	4	8
Sprint 6: Real-Time Features (48h)				
6	Socket.IO server + JWT auth middleware	6	–	6
	Room management (board/user rooms)	4	–	4
	CRUD event broadcasting	6	–	6

	Socket.IO client + Redux listeners	–	8	8
	Active users presence system	4	6	10
	Notification system (backend + frontend)	4	6	10
	Connection status + reconnection handling	2	2	4
Sprint 7: Testing & Delivery (24h)				
7	Unit test completion (90% coverage)	4	4	8
	Integration/E2E tests	3	3	6
	Bug fixes & performance optimization	2	2	4
	Documentation & presentation prep	3	3	6
TOTAL		**156h**	**156h**	**312h**

10 Deliverables & Success Criteria

Source Code & Repository

Complete GitHub repository with full commit history demonstrating incremental development, monorepo structure with /backend and /frontend directories, comprehensive README.md with installation instructions, environment setup, and development commands, Docker Compose configuration for one-command local deployment, GitHub Actions workflows for CI (lint/test/coverage) and CD (Fly.io deployment).

Technical Documentation

Architecture overview with system diagrams and layer descriptions, data model documentation with entity relationships and field specifications, REST API reference with all endpoints, request/response formats, and error codes, WebSocket events documentation with payloads and usage examples, environment variables reference for both development and production, deployment guide for Fly.io with MongoDB Atlas setup instructions.

Application Deliverables

Functional web application deployed to Fly.io (frontend: epitrello-frontend.fly.dev, backend: epitrello-backend.fly.dev), local Docker-based development environment, demo data seeding scripts for testing, test user accounts for evaluation purposes.

Test Suite

Automated test suite with 90%+ statement/line coverage and 80%+ branch/function coverage, backend tests using Vitest + Supertest + mongodb-memory-server, frontend tests using Vitest + React Testing Library + MSW, coverage reports generated via vitest --coverage with HTML lcov output.

Success Criteria

Criterion	Measurement
Core Feature Completion	All Kanban features functional: user auth, board/list/card CRUD, drag-and-drop, member management, real-time sync
Application Stability	No critical bugs, graceful error handling, responsive UI across desktop/tablet viewports
Code Quality	90%+ test coverage, zero ESLint warnings, consistent Prettier formatting, passing CI on all merges
Real-Time Reliability	WebSocket updates propagate within 500ms, automatic reconnection works, no state desync between clients
Documentation Completeness	All APIs documented, architecture explained, deployment reproducible from docs alone
Methodology Adherence	Sprint-based development with backlog tracking, TDD practices applied, code reviews on all PRs

Production URLs: Frontend: `https://epitrello-frontend.fly.dev` | Backend API: `https://epitrello-backend.fly.dev/api` | Health Check: `/api/health`