VESIT V.E.S. Since 1962 VIVEKANAND EDUCATION SOCIETY INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Real-Time Collaborative Task Management System

Submitted in partial fulfilment of the requirements of the

Full Stack Development

in Semester - V

By

58- Prakash Thapa

70- Chirag Poornamath

under the guidance of

Mrs. Pooja Prajapati

VESIT V.E.S. Since 1962

Department of Information Technology

Vivekanand Education Society's Institute of Technology-2025-2026

VESIT V.E.S. Since 1962 VIVEKANAND EDUCATION SOCIETY INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

Abstract

The Real-Time Collaborative Task Management System is a full-stack web application developed as the integrative mini project for the Full Stack Development laboratory. The application consolidates ten laboratory experiments into a single production-grade system that supports user authentication, role-based authorization, RESTful APIs, real-time collaboration through WebSockets, responsive UI design, and DevOps deployment pipelines. The frontend is implemented in React with Tailwind CSS for rapid, utility-driven responsive styling. The backend uses Node.js and Express to expose secure REST endpoints and to host Socket.io event handlers. MongoDB with Mongoose provides persistent storage. Authentication is based on JSON Web Tokens (JWT) with role enforcement. CI/CD pipelines are implemented using GitHub Actions and deployments are hosted on Vercel (frontend) and Render (backend). Containerization with Docker ensures environment parity. This report documents the project objectives, system design, the mapping of lab experiments into modules of the final system, implementation details, testing, deployment, challenges, results, and future enhancements.

Table of Contents

1. Introduction

Real-time collaboration and transparent task tracking are essential for efficient team workflows. This mini project builds a web application that supports collaborative task management in a Kanban board layout, enabling multiple users to create, assign, move, and comment on tasks with immediate synchronization across clients. The application is designed and implemented as an agglomeration of ten laboratory experiments conducted during the course. Each experiment introduces a specific technology or practice; the final system integrates them to demonstrate practical competency in full-stack development.

---

2. Objectives

- Consolidate theoretical and lab exercises into a working, deployable web application.
- Implement a responsive, accessible UI using Tailwind CSS.
- Apply React Hooks and Context/Redux for component and application state.
- Design secure RESTful APIs with proper validation and error handling.
- Persist application data using MongoDB and Mongoose models.
- Implement JWT-based authentication and role-based authorization.
- Validate APIs using Postman test collections.
- Enable real-time bi-directional communication with Socket.io.
- Establish CI/CD pipelines and deploy the application using containerized artifacts.
- Demonstrate ability to debug, profile, and optimize across the stack.

---

3. Scope of the Project

The scope of this project covers the following functional and non-functional requirements:

Functional:

- User registration, login, and profile management.
- Project creation and membership management.
- Task creation, editing, assigning, deleting, and moving across columns.
- Real-time updates for task operations and presence status.
- Notifications (in-app) for relevant events.
- Role-based access control (Admin/Member).

Non-Functional:

- Responsive UI across desktop and mobile.
- Secure authentication and authorization.
- Reasonable performance under moderate concurrent load.
- Continuous integration and deployment workflows.
- Reproducible deployment using Docker.

Out of scope: advanced third-party calendar sync, real push notifications to mobile OS (only in-app and email placeholders), and multi-region scaling.

---

4. System Overview

The system is a three-tier application combining client, server, and database layers. The client is a single page React application that renders the Kanban board, handles drag-and-drop interactions, maintains local state, and communicates with the server via REST and WebSockets. The server exposes REST endpoints for CRUD operations, implements authentication middleware, and manages Socket.io rooms for per-project real-time messaging. The database stores normalized documents for users, projects, tasks, and notifications.

Key flows:

- Authentication flow: credentials → REST login endpoint → JWT issued → stored client-side → used for subsequent REST and Socket authentication.
- Task flow: create/update/move actions → server persists to DB → server emits Socket.io events to project room → clients update state and UI.
- Presence flow: client connects/disconnects → server broadcasts online/offline events → presence indicators update.

---

5. Technologies and Tools

Frontend:

- React 18 (functional components + hooks)
- Vite (dev server and build tool)
- Tailwind CSS (utility-first styling)
- React Beautiful DnD (drag and drop)
- Socket.io Client

Backend:

- Node.js (v18+)
- Express.js
- Socket.io Server
- MongoDB (Atlas for production or local for development)
- Mongoose (schema modeling)
- Bcrypt (password hashing)
- JSON Web Tokens (JWT) for auth

DevOps & Testing:

- Docker & docker-compose
- GitHub Actions for CI/CD
- Vercel (frontend) and Render (backend) for hosting
- Postman for API validation

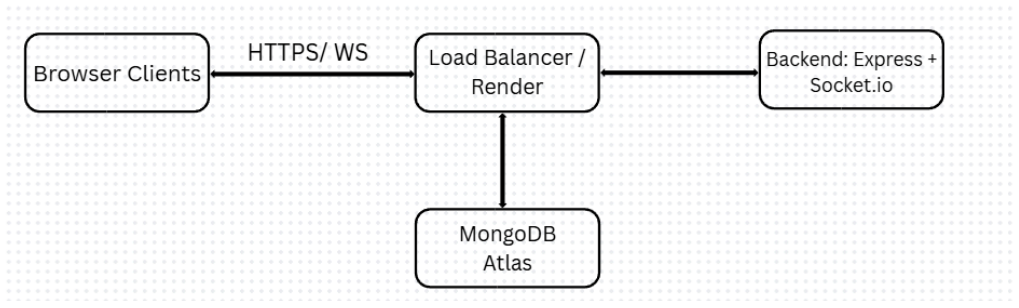
Other Utilities:

- ESLint, Prettier for code quality
- PM2 (optional) for production process management

---

6. System Architecture

High-Level Diagram (logical):



Architectural Highlights:

- The server separates REST endpoints and WebSocket event handlers but shares business logic modules (services) and models to maintain a single source of truth.
- Each project has a Socket.io "room". When users join a project page, the client emits join-project; the server adds the socket to that room and begins broadcasting project-scoped events.
- JWT tokens are verified both in REST middleware and at Socket.io connection time (via token handshake) to ensure only authenticated users open WebSocket connections.

---

7. Database Design

Collections and Key Fields:

Users

- _id, name, email, passwordHash, role, avatarUrl, lastSeen, createdAt, updatedAt

Projects

- _id, title, description, adminId, members [userId], columns [{ id, title, order }], createdAt, updatedAt

Tasks

- _id, title, description, status (column id), projectId, assignedTo [userId], position, comments [{ author, text, createdAt }], dueDate, createdAt, updatedAt

Notifications

- _id, userId, type, payload, read (boolean), createdAt

Indexing and Performance:

- Indexes on users.email (unique), tasks.projectId, tasks.assignedTo, and projects.adminId.
- TTL index for temporary activity logs if implemented.
- Use projection and pagination for large task lists.

---

8. Experiment Integration — Mapping Experiments to Modules

Each lab experiment was intentionally designed to map onto a module or capability of the final system:

1. **Tailwind CSS (Responsive UI)** — Styles, responsive breakpoints, and utility classes used for the board, cards, and administrative UI.
2. **React Hooks** — useEffect for lifecycle and subscriptions, useContext for auth and socket context, custom hooks for API calls (useApi), debounced search, and form handling.
3. **Redux / Context API** — Global state for user, projects, and real-time event handling; Redux used when state complexity increased (e.g., multiple reducers for projects/tasks).
4. **REST API Design & Mongoose** — Structured endpoints for users/projects/tasks, schema validation, and relational references implemented via Mongoose.
5. **Production-Ready APIs** — Validation, sanitization, logging middleware, and rate limiting applied across routes.
6. **JWT Authentication & Roles** — Secure login, refresh token strategy, and middleware enforcing Admin/Member permissions.
7. **Postman Validation** — Postman collections document endpoints and include tests for status codes, response payloads, and auth flows.
8. **WebSockets (Socket.io)** — Real-time updates, presence, and notifications built on project rooms and event broadcasting.
9. **CI/CD (GitHub Actions)** — Linting, tests, build, and deploy jobs on push/PR to main branch.
10. **Docker Deployment** — Dockerfiles for client and server; docker-compose for local orchestration and parity.

---

9. Frontend Implementation

Component Structure

- App: top-level routing, auth provider, socket provider.
- Auth components: Login, Register, Profile.
- Dashboard: lists projects, quick stats.
- Board: main Kanban component, coordinates columns and cards.
- TaskCard, TaskModal: show and edit task details.
- PresenceIndicator, Notifications: real-time UI widgets.

Styling & Responsiveness

- Tailwind utility classes compose reusable components. Breakpoints (sm, md, lg) ensure the board degrades gracefully on smaller screens: on narrow viewports columns scroll horizontally; on mobile, the project page stacks with collapsible columns.

Drag & Drop

- React Beautiful DnD manages drag context, droppable columns, and draggable cards. OnDragEnd computes the new status and position and calls an API endpoint followed by a socket emit to propagate change.

Socket & State Integration

- A SocketContext provider wraps the app to allow components to subscribe to events: task-created, task-updated, task-moved, user-online, user-offline.
- Local caching and optimistic UI updates: the UI assumes success and rolls back on error responses.

Accessibility

- Focus management within modals, ARIA attributes for droppables, keyboard support for navigation, and sufficient color contrast for readability.

---

10. Backend Implementation

Folder Structure

```
server/
├── controllers/
├── models/
├── routes/
├── middleware/
├── socket/
└── utils/
```

REST Endpoints

- Auth routes: /api/auth/register, /api/auth/login, /api/auth/me
- Project routes: /api/projects, /api/projects/:id/members
- Task routes: /api/tasks, /api/tasks/:id/move
- Notification routes: /api/notifications

Middleware

- auth.js: verifies JWT and attaches user to request.
- roles.js: checks user roles before sensitive operations.
- errorHandler.js: centralized error handling and logging
- validateRequest.js: request schema validation using Joi or express-validator.

Socket Handlers

- Authentication of socket handshake.
- Join/leave rooms per project.
- Event handlers: task-created, task-updated, task-deleted, task-moved, comment-added.
- Broadcast strategies to minimize duplication (emit to room excluding origin if needed).

Logging & Monitoring

- Winston or Morgan used for request logging.
- Sentry (optional) for error telemetry in production.

---

11. Real-time Collaboration and Socket.io Integration

Connection Flow

- Client connects to Socket.io with token: io(VITE_SOCKET_URL, { auth: { token }}).
- Server validates the token in io.use middleware.
- On join-project, the server socket.join(projectId) and emits user-online to the room.

Event Patterns

- **Command**: client emits an action (e.g., task-move-request) with minimal payload.
- **Server**: validates, persists change to DB, then emits authoritative task-moved to project room.
- **Client**: updates local store when receiving task-moved.

Conflict Resolution

- Server is authoritative: any conflicting simultaneous moves are serialized by server persistence time. Clients reconcile the server state when receiving events. To minimize user confusion, transient “syncing” indicators are shown.

---

12. Authentication, Authorization and Security

Authentication

- Passwords hashed with Bcrypt before storage.
- JWTs signed with a secure secret and expiry (JWT_EXPIRE=30d in development only; production uses shorter token life + refresh tokens).
- Tokens stored in secure, HTTP-only cookies or localStorage depending on requirements; secure cookie preferred for production.

Authorization

- Role enforcement middleware ensures only Admins can add/remove members.
- Resource ownership checks (e.g., task editing limited to members of the project).

API Security Best Practices Implemented

- HTTPS enforced in production.
- Rate limiting to reduce brute force.
- Input sanitization to prevent injection attacks.
- CORS constraints configured to allow frontend origin only.
- Helmet used to secure HTTP headers.

---

13. API Design and Validation (Postman)

A Postman collection documents all the API flows with example requests and environments. Key tests:

- Successful authentication returns status 200 and a token.
- Protected routes return 401 without token.
- CRUD endpoints return correct status codes (201 for create, 200 for read/update, 204 for delete).
- Race/error scenarios tested by concurrent Postman runners to examine behavior on conflicting updates.

Automated tests using Postman/Newman can be tied into the CI pipeline to run on each push.

---

14. State Management and React Hooks

Local vs Global State

- Local state handles component-level UI (modal open/close, form inputs).
- Global state (Redux or Context + reducer) stores user profile, active project, and tasks. Redux is chosen when middleware (redux-saga or thunk) was needed to coordinate asynchronous flows with socket events.

Hooks Utilized

- useEffect for subscriptions/unsubscriptions to socket events and for data fetching on mount.
- useMemo for memoizing sorted and filtered lists of tasks.
- Custom hooks: useAuth(), useSocket(projectId), useApi(endpoint) to standardize error handling and retries.

---

15. Testing Strategy and Results

Unit & Integration Tests

- Backend: Jest + Supertest test REST endpoints, middleware, and error responses.
- Frontend: React Testing Library tests component rendering, DnD behaviors, and state updates.

Manual Integration Tests

- Multi-browser (Chrome + Firefox + mobile emulation) tests to verify real-time synchronization.
- Simulated multiple users creating and moving tasks simultaneously to observe conflict resolution and latency.

Test Coverage & Findings

- Critical endpoints covered; test coverage aimed for 70%+.
- Observed edge case: occasional UI desynchronization when optimistic updates were applied and server later rejected changes; fixed by stricter validation and clearer rollback UI flows.

---

16. Deployment and DevOps

Dockerization

- Two Dockerfiles (frontend, backend). The backend image installs dependencies, copies code, and runs node server.js; the frontend builds static assets served by Vercel (or nginx if self-hosting).

docker-compose (local)

- docker-compose.yml orchestrates client, server, and a local MongoDB for development.

CI/CD (GitHub Actions)

- Workflows:
  - ci.yml: install → lint → run tests → build artifacts.
  - deploy.yml: on main push, build and trigger Vercel/Render deploy via their GitHub integration or via API.

Production Hosting

- Frontend deployed to Vercel for instant global CDN distribution.
- Backend deployed to Render with environment variables for MongoDB URI, JWT secret, and allowed origins.

Environment Configuration

- Secrets managed by deployment platform environment variables; no secrets in codebase.

---

17. Performance Considerations and Optimization

- Use of socket rooms reduces broadcast scope and network traffic.
- Task lists paginated or lazily loaded to avoid rendering large DOM lists.
- Memoization prevents unnecessary re-renders of task lists.
- Database queries tuned with appropriate indexes; projections used to limit returns.
- Compression (gzip) enabled in production server; static CDN used for frontend assets.

---

18. Challenges and Solutions

1. **Real-time Consistency** — Race conditions resolved by server-authoritative updates and atomic database operations.
2. **Drag-and-Drop Persistency** — Ordering maintained by position fields and recalculated on move; concurrent moves reconciled by server.
3. **Socket Authentication** — Token handshake implemented; invalid sockets rejected with clear client feedback.
4. **CORS & Socket Origins** — Identified mismatches during deployment; fixed by aligning CORS_ORIGIN and socket URL environment variables.
5. **Deployment Limits** — Free tiers on Render/Vercel required optimization like smaller instance size, fewer logging outputs, and efficient image compression.

---

19. Future Work and Enhancements

- Add calendar integration (Google Calendar) and milestone timelines.
- Implement file attachments and rich comments with markdown support.
- Extend notifications to email and mobile push (via Firebase).
- Analytics dashboard with project metrics and user productivity insights.
- Implement offline support and background sync using service workers.
- Introduce multi-tenant architecture for enterprise deployments.
- Add end-to-end encryption for sensitive task data at rest.

---

20. Conclusion

This project successfully integrates the learning objectives covered in the lab experiments into a functional and deployable real-time collaborative application. It demonstrates practical mastery of frontend responsiveness, modern React practices, state management, secure backend API design, real-time WebSocket communication, and modern DevOps practices. The system is suitable as a capstone mini project and forms a robust foundation for feature expansion or production hardening.

---

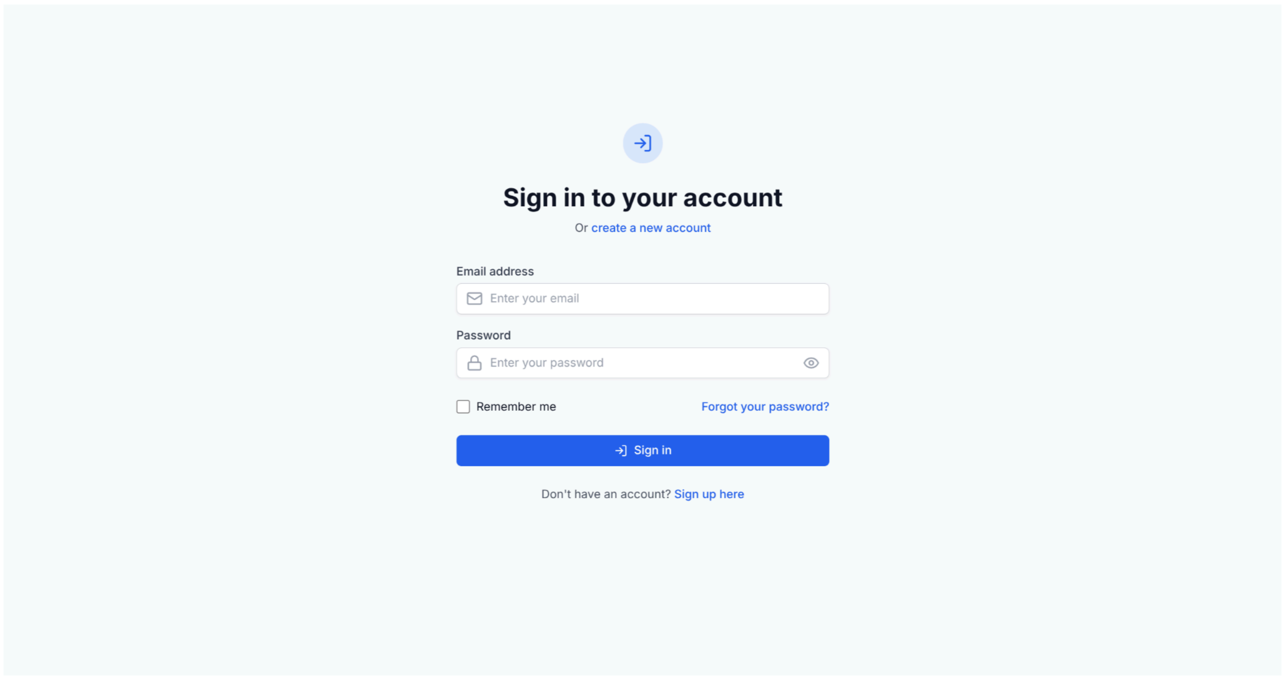
21. References

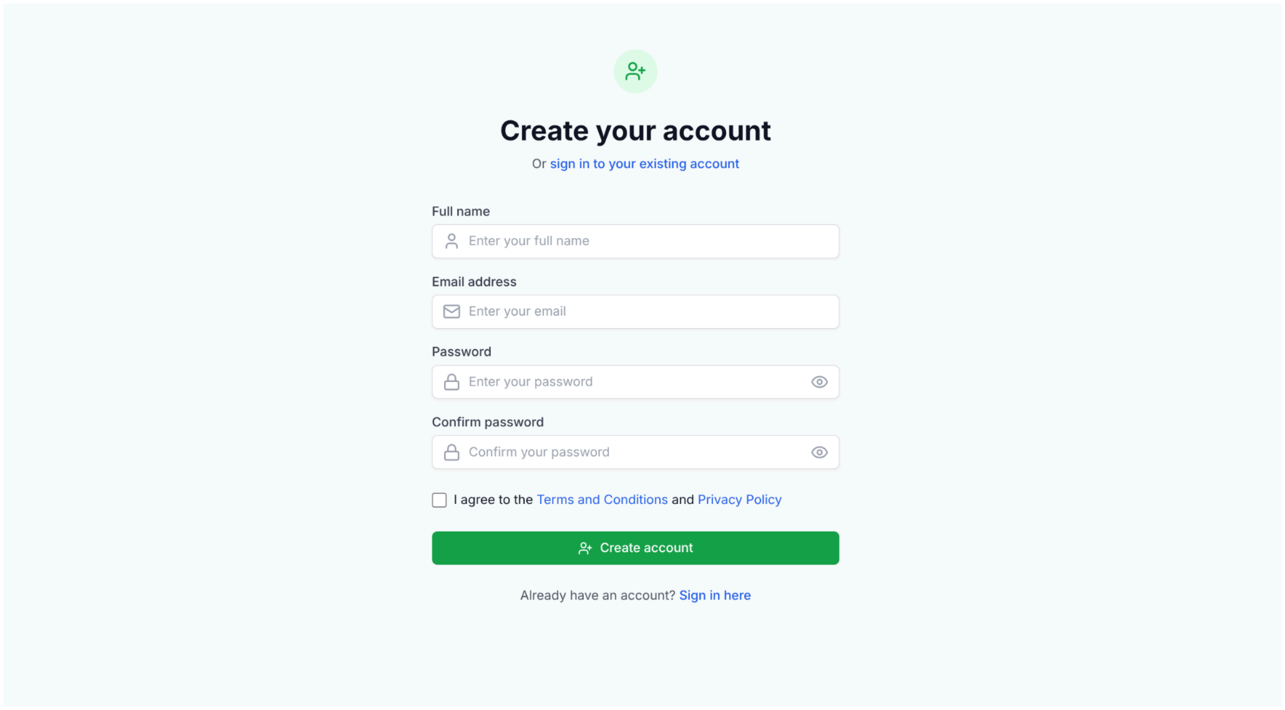
- React — https://react.dev/
- Express — https://expressjs.com/
- Socket.io — https://socket.io/
- MongoDB — https://www.mongodb.com/
- Mongoose — https://mongoosejs.com/
- Tailwind CSS — https://tailwindcss.com/
- Vercel docs — https://vercel.com/docs
- Render docs — https://render.com/docs
- JWT — https://jwt.io/
- React Beautiful DnD — https://github.com/atlassian/react-beautiful-dnd

---

22. Appendix: Screenshots

Figure 1 — Login / Registration Page




Create your account

Or sign in to your existing account

Full name

Enter your full name

Email address

Enter your email

Password

Enter your password

Confirm password

Confirm your password

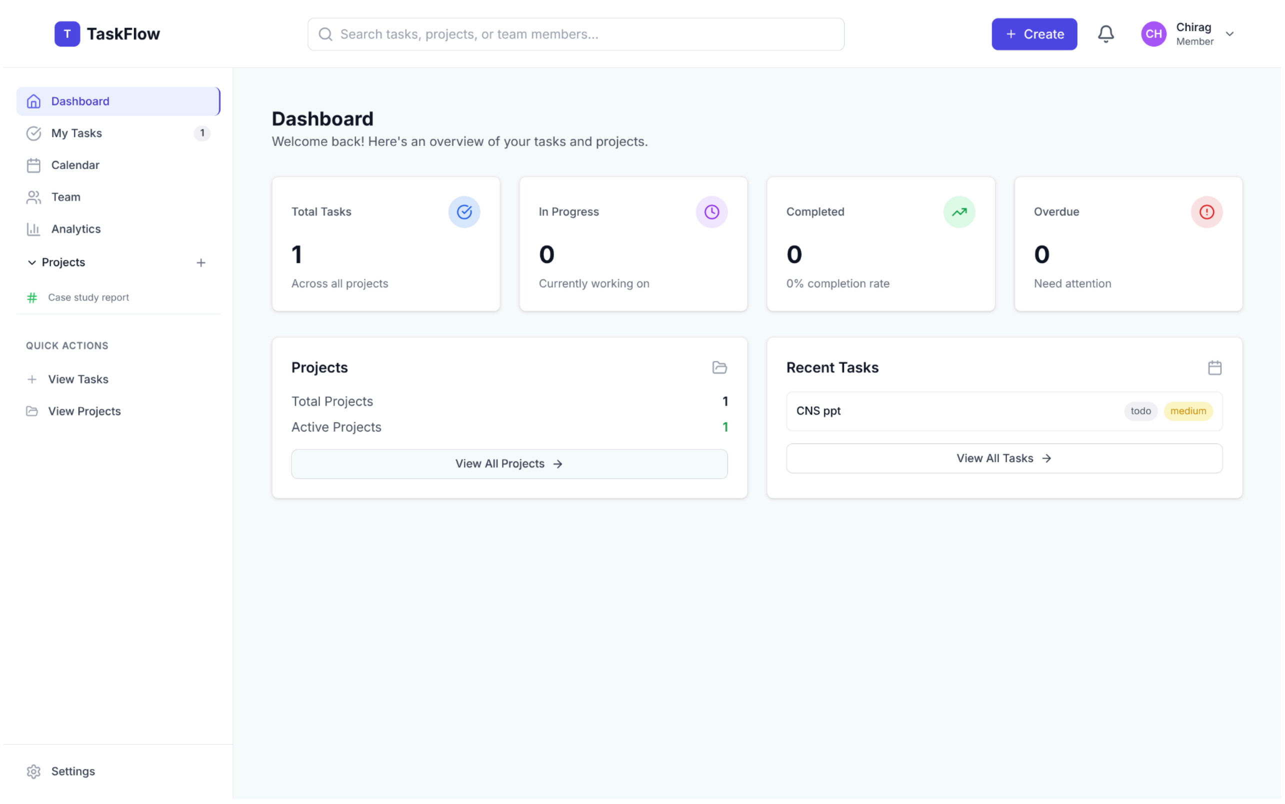
I agree to the Terms and Conditions and Privacy Policy

Create account

Already have an account? Sign in here

Login and Signup page showing authentication form and validation messages.

Figure 2 — Dashboard / Projects List


TaskFlow

Search tasks, projects, or team members...

Create

Chirag
Member

Dashboard
My Tasks 1
Calendar
Team
Analytics
Projects
Case study report

QUICK ACTIONS
View Tasks
View Projects

Settings

Dashboard

Welcome back! Here's an overview of your tasks and projects.

Total Tasks	In Progress	Completed	Overdue
1	0	0	0
Across all projects	Currently working on	0% completion rate	Need attention

Projects

Total Projects	1
Active Projects	1

View All Projects

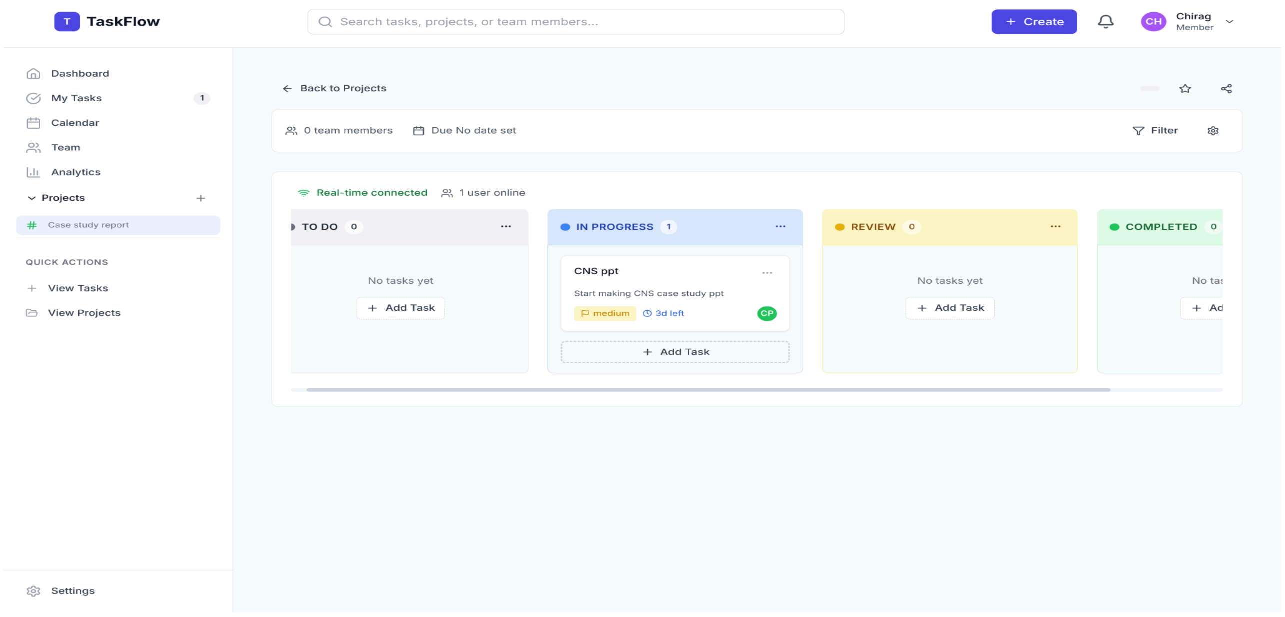
Recent Tasks

CNS ppt todo medium

View All Tasks

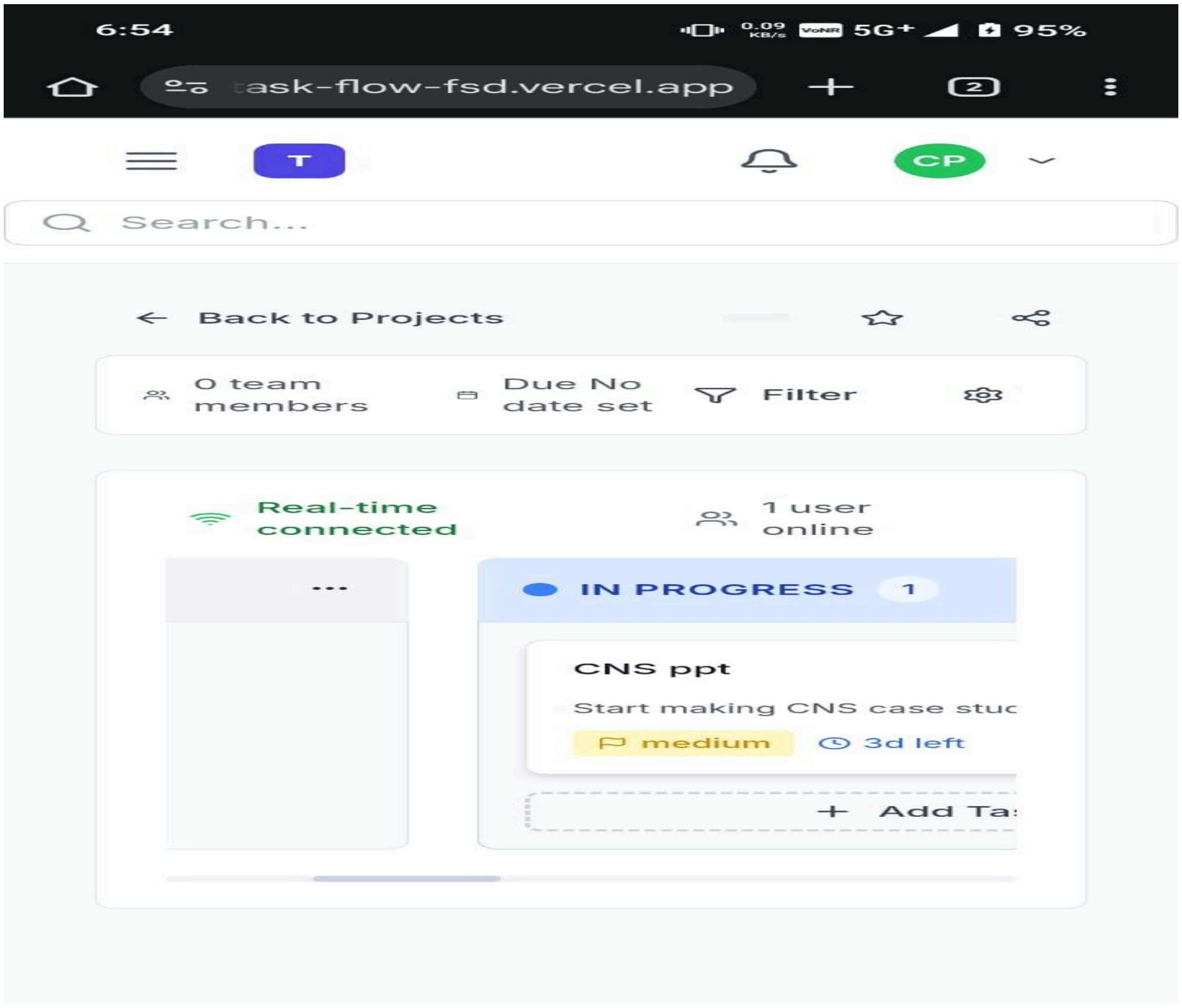
The dashboard listing available projects with quick stats.

Figure 3 — Kanban Board (Desktop)



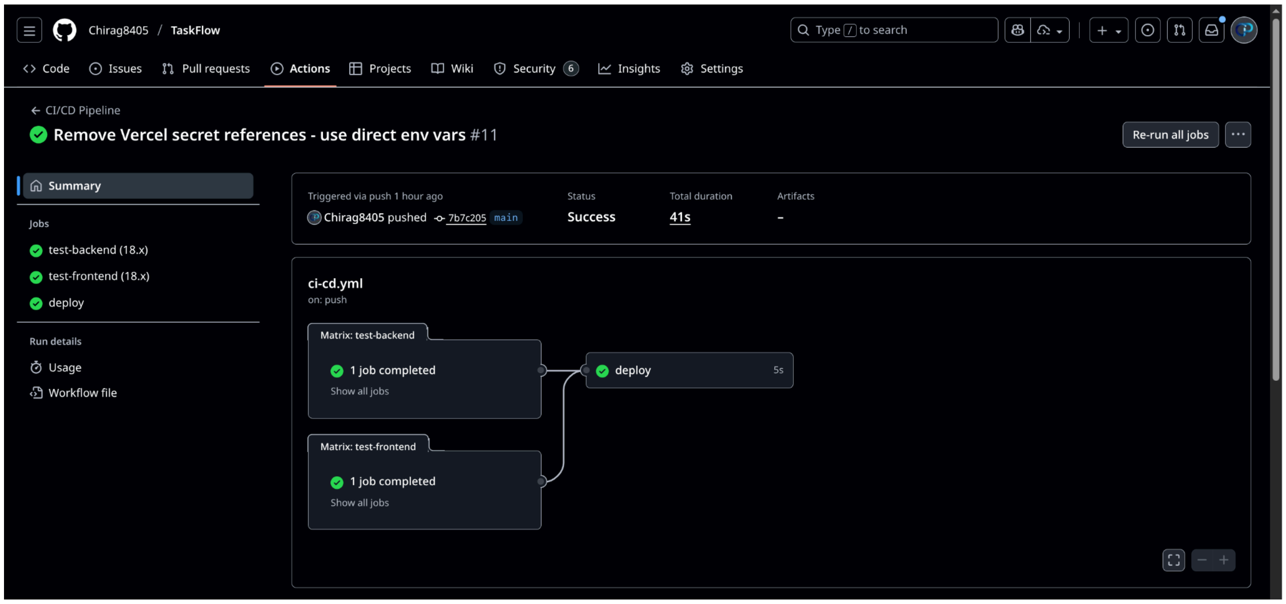
The main board view with columns (To Do, In Progress, Done) and draggable cards.

Figure 4 — Kanban Board (Mobile)



Mobile responsive behavior showing columns stacked and cards collapsed.

Figure 5 — Deployment Logs / Render Panel



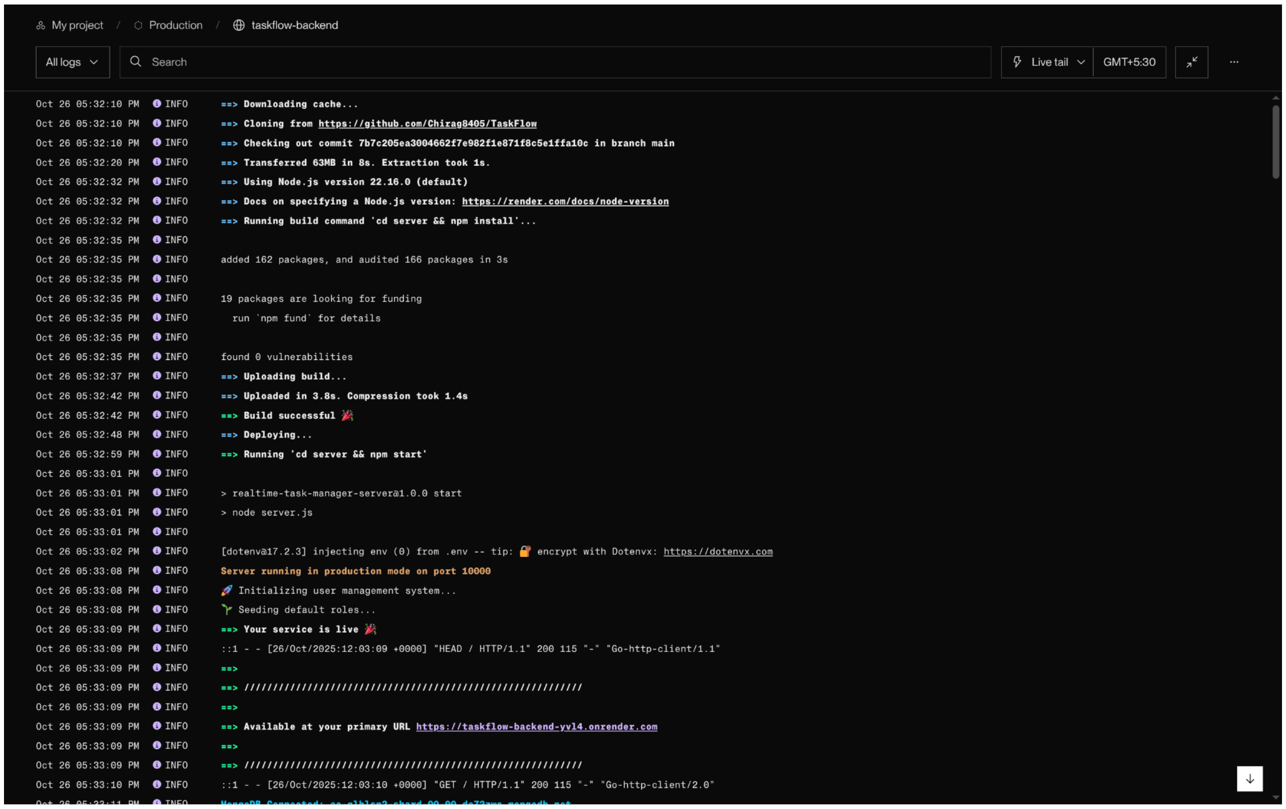
Remove Vercel secret references - use direct env vars #11

Triggered via push 1 hour ago: Chirag8405 pushed 7b7c205 main
Status: Success
Total duration: 41s
Artifacts: –

Jobs:
- test-backend (18.x)
- test-frontend (18.x)
- deploy

ci-cd.yml
on: push

Matrix: test-backend — 1 job completed
Matrix: test-frontend — 1 job completed
deploy 5s



```
Oct 26 05:32:10 PM  INFO   ==> Downloading cache...
Oct 26 05:32:10 PM  INFO   ==> Cloning from https://github.com/Chirag8405/TaskFlow
Oct 26 05:32:10 PM  INFO   ==> Checking out commit 7b7c205ea3004662f7e982f1e871f8c5e1ffa10c in branch main
Oct 26 05:32:20 PM  INFO   ==> Transferred 63MB in 8s. Extraction took 1s.
Oct 26 05:32:32 PM  INFO   ==> Using Node.js version 22.16.0 (default)
Oct 26 05:32:32 PM  INFO   ==> Docs on specifying a Node.js version: https://render.com/docs/node-version
Oct 26 05:32:32 PM  INFO   ==> Running build command 'cd server && npm install'...
Oct 26 05:32:35 PM  INFO
Oct 26 05:32:35 PM  INFO   added 162 packages, and audited 166 packages in 3s
Oct 26 05:32:35 PM  INFO
Oct 26 05:32:35 PM  INFO   19 packages are looking for funding
Oct 26 05:32:35 PM  INFO     run `npm fund` for details
Oct 26 05:32:35 PM  INFO
Oct 26 05:32:35 PM  INFO   found 0 vulnerabilities
Oct 26 05:32:37 PM  INFO   ==> Uploading build...
Oct 26 05:32:42 PM  INFO   ==> Uploaded in 3.8s. Compression took 1.4s
Oct 26 05:32:42 PM  INFO   ==> Build successful 🎉
Oct 26 05:32:48 PM  INFO   ==> Deploying...
Oct 26 05:32:59 PM  INFO   ==> Running 'cd server && npm start'
Oct 26 05:33:01 PM  INFO
Oct 26 05:33:01 PM  INFO   > realtime-task-manager-server@1.0.0 start
Oct 26 05:33:01 PM  INFO   > node server.js
Oct 26 05:33:01 PM  INFO
Oct 26 05:33:02 PM  INFO   [dotenv@17.2.3] injecting env (0) from .env -- tip: 🔐 encrypt with Dotenvx: https://dotenvx.com
Oct 26 05:33:08 PM  INFO   Server running in production mode on port 10000
Oct 26 05:33:08 PM  INFO   🚀 Initializing user management system...
Oct 26 05:33:08 PM  INFO   🌱 Seeding default roles...
Oct 26 05:33:09 PM  INFO   ==> Your service is live 🎉
Oct 26 05:33:09 PM  INFO   ::1 - - [26/Oct/2025:12:03:09 +0000] "HEAD / HTTP/1.1" 200 115 "-" "Go-http-client/1.1"
Oct 26 05:33:09 PM  INFO   ==>
Oct 26 05:33:09 PM  INFO   ==> ///////////////////////////////////////////////////////////
Oct 26 05:33:09 PM  INFO   ==>
Oct 26 05:33:09 PM  INFO   ==> Available at your primary URL https://taskflow-backend-yvl4.onrender.com
Oct 26 05:33:09 PM  INFO   ==>
Oct 26 05:33:09 PM  INFO   ==> ///////////////////////////////////////////////////////////
Oct 26 05:33:10 PM  INFO   ::1 - - [26/Oct/2025:12:03:10 +0000] "GET / HTTP/1.1" 200 115 "-" "Go-http-client/2.0"
```

Live URL: https://task-flow-fsd.vercel.app
Successful backend deployment and live URL.