Role-Based Project Collaboration Full-Stack System (Team Tasker)

Prepared by:
SIRAJUDEEN G
sirajudeenghousebasha@gmail.com

Role-Based Project Collaboration Full-Stack System (Team Tasker)

Prepared for: Sirajudeen G
Date: 12 Jun 2025

I. Project Summary

TeamTasker is a comprehensive full-stack task and project management system built to streamline workflow coordination, user access control, and real-time updates across collaborative teams. It empowers different team members based on their roles to create, manage, and track tasks and projects through a robust and secure interface.

The application focuses on simplifying the management of projects by introducing role-based access, real-time socket-driven updates, and efficient data handling with PostgreSQL and Redis. Each user interacts only with functionalities relevant to their role, such as Admin, Project Manager, Developer, Tester, or Viewer. Security and modularity are deeply embedded in both frontend and backend design.

II. Project Overview

A. Project Objectives

TeamTasker aims to provide a secure, scalable, and real-time task management solution that aligns responsibilities based on user roles. The system enables streamlined planning, coordination, and execution of projects, where every team member has defined access boundaries and role-specific responsibilities.

B. Project Scope

TeamTasker is designed for teams that require structured collaboration. The scope includes user registration, login with JWT authentication, project and task management, role-based access control (RBAC), commenting, real-time updates, and robust session management using Redis. The app is ideal for organizations managing multiple ongoing projects across different departments or roles.

C. Key Deliverables

TeamTasker allows teams to plan, assign, collaborate, and monitor tasks under various projects with role-specific access levels. Tasks support assignees, comments, and progress tracking. Admins manage users and roles, while project managers oversee deliverables. Developers and testers focus on execution and quality assurance respectively.

Key functionalities include real-time task visibility, live project boards, and secure login/auth workflows using JWTs. The backend ensures only authorized actions via permissions middleware, while the frontend adapts dynamically to each user's permissions. Redis optimizes session management, and Socket.IO delivers instant feedback on task events.

Built with React on the frontend and Node.js with TypeScript on the backend, TeamTasker adopts clean architecture principles, Sequelize ORM, and a modular role-permission model. This design ensures performance, extensibility, and security for growing teams.

III. Features Implemented

The following are the key functional modules and features implemented in TeamTasker, each contributing to secure, collaborative, and real-time task management:

- **Role-Based Access Control (RBAC):** Enables permission-based operations for roles such as Admin, Project Manager, Developer, Tester, and Viewer.
- **User Authentication:** JWT-based login and registration flow with access and refresh tokens.
- **Session Management with Redis:** Redis handles session storage and caching for faster response and better scalability.
- **Admin Role Management:** Admin users can assign and update roles for other users.
- **Project Management:** Authorized users can create, update, and delete projects with full task integration.
- **Task Management:** Tasks can be created, edited, assigned to users, and have status updates reflecting progress.
- **Real-time Dashboard Sync:** Socket.IO broadcasts live task and project updates to all connected clients.
- **Commenting System:** Tasks support threaded comments with user context to aid collaboration.
- **Secure Routing:** Backend APIs are protected using `verifyJWT`, `authorizeRoles`, and `authorizePermissions` middleware.

- **Dynamic Frontend UI:** React components are conditionally rendered based on the authenticated user's role and permissions.
- **Persistent Data Layer:** All core data—users, tasks, roles, permissions, and projects—is stored and queried via PostgreSQL using Sequelize ORM.
- **Efficient Error Handling:** Graceful fallback for unauthorized (401), forbidden (403), and server (500) errors.
- **Empty State UI:** Intuitive messaging when task lists or project data are not yet available.

IV. Tech Stack

A. Frontend

- Axios for API calls
- Tailwind css for styling
- Context API for auth state management
- Socket.IO client for real-time updates
- Role-based UI control

B. Backend

- Node.js + Express
- TypeScript for static type safety
- Socket.IO server for real-time notifications
- Redis (ioredis) for caching and token/session management
- JWT authentication system.

C. Database

- PostgreSQL
- Sequelize ORM
- Models: User, Role, Permission, Task, Project, Comment
- Associations: User <-> Role, Role <-> Permission, Task <-> Project, Task <-> User (assignee), Task <-> Comment

D. Database

- Nodemon for dev

- Git for version control

V. Challenges and Solutions

Challenge 1: Role mismatch causing permission checks to fail

- **Problem:** `user.roles[0]` returned an object like `{ name: "Admin" }`, but permission checks expected strings.
- **Solution:** Normalized roles in frontend `AuthContext.js` by mapping to `roles.map(r => r.name)` ensuring compatibility with `rolePermissions` map.

Challenge 2: Tasks not showing after page refresh despite being created

- **Problem:** `projectId` was missing from the backend `listTasks` response, so frontend filtering failed.
- **Solution:** Added `attributes: ['projectId', ...]` in Sequelize query to include it explicitly in the API response.

Challenge 3: Server crashing on `/projects/:id` due to undefined user object

- **Problem:** `req.user` was missing if `verifyJWT` was skipped or failed silently.
- **Solution:** Ensured all protected routes include `verifyJWT` and added null checks inside `getProjectById` to avoid crashes.

Challenge 4: Real-time updates not syncing between clients

- **Problem:** Socket.IO emit calls were scoped incorrectly or never fired.
- **Solution:** Scoped emits by project ID and ensured `io.emit("taskCreated", task)` was triggered after DB persistence.

Challenge 5: Blank pages for empty task lists

- **Problem:** UI did not handle empty array returns from API.
- **Solution:** Added fallback rendering in React (`tasks.length === 0 ? <EmptyState /> : <TaskList />`).

Challenge 6: Duplicate users or roles during seed process

- **Problem:** Manual re-seeding caused duplicate constraints.
- **Solution:** Added `findOrCreate` logic in RBAC seeding utility to safely upsert roles and permissions.

Challenge 7: Managing refresh token security

- **Problem:** Refresh token could be reused or tampered with.

VI. Repository and Media

A. Github Repository

- **GitHub Repository:** https://github.com/Siraj004/Task_Manager

B. Demo video link

- **Demo Video:** Watch Demo Video

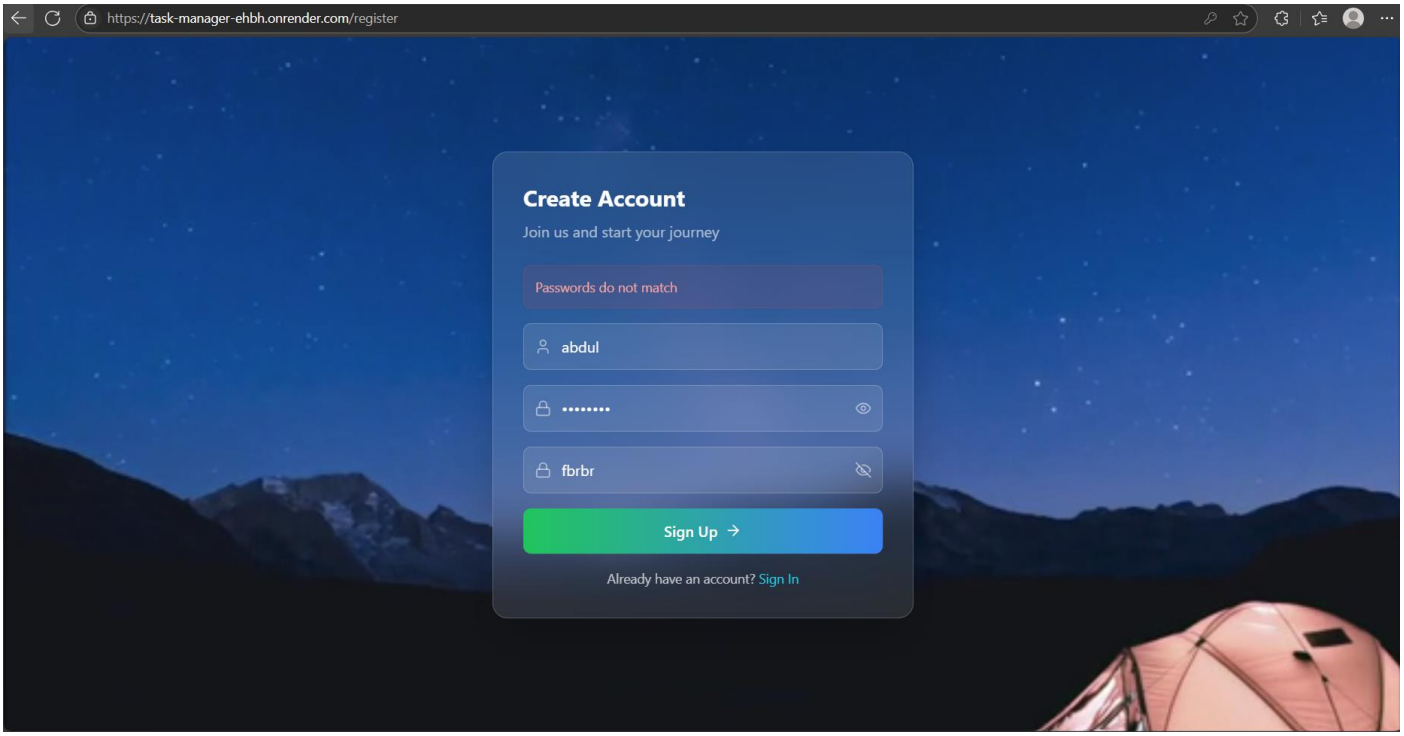
C. Live Deployment Link

- **Frontend + Backend Live URL:** https://task-manager-ehbh.onrender.com

D. Screenshots of Project

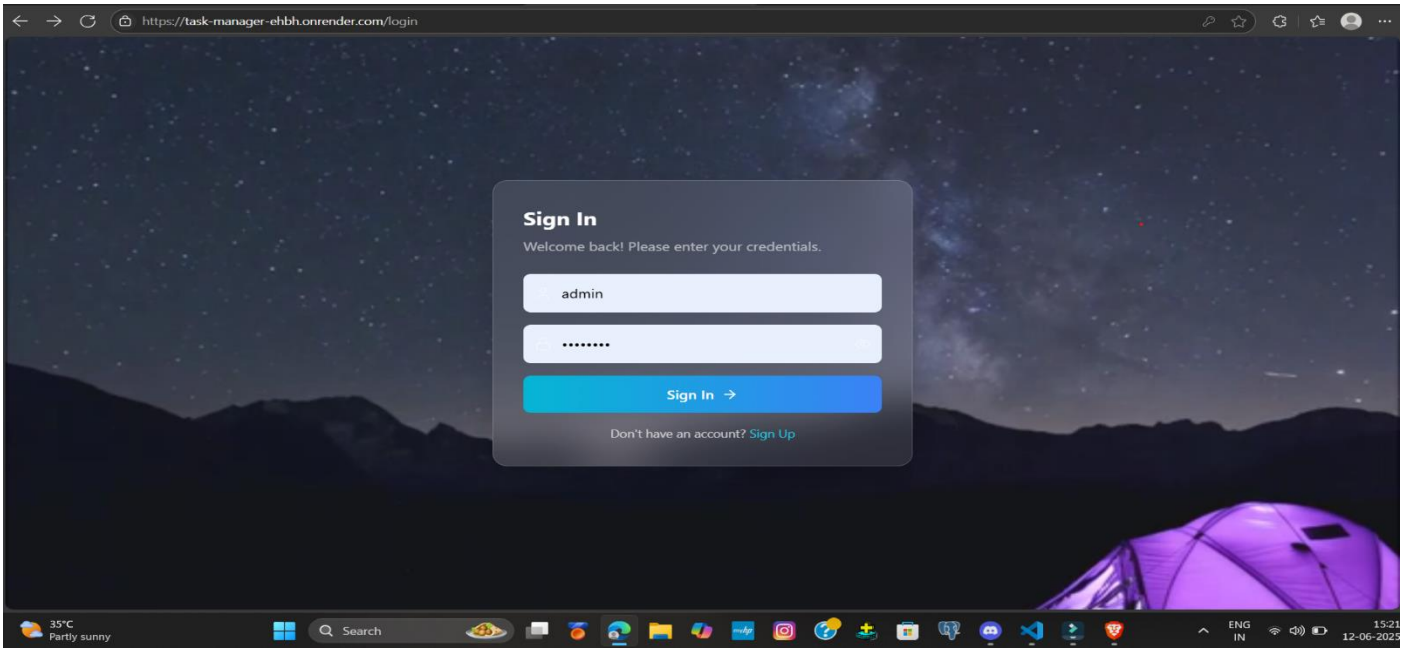
- **Screenshot Gallery:** View on Google Drive

Sign up page
New users can Sign up => they will be assigned for viewer role by default



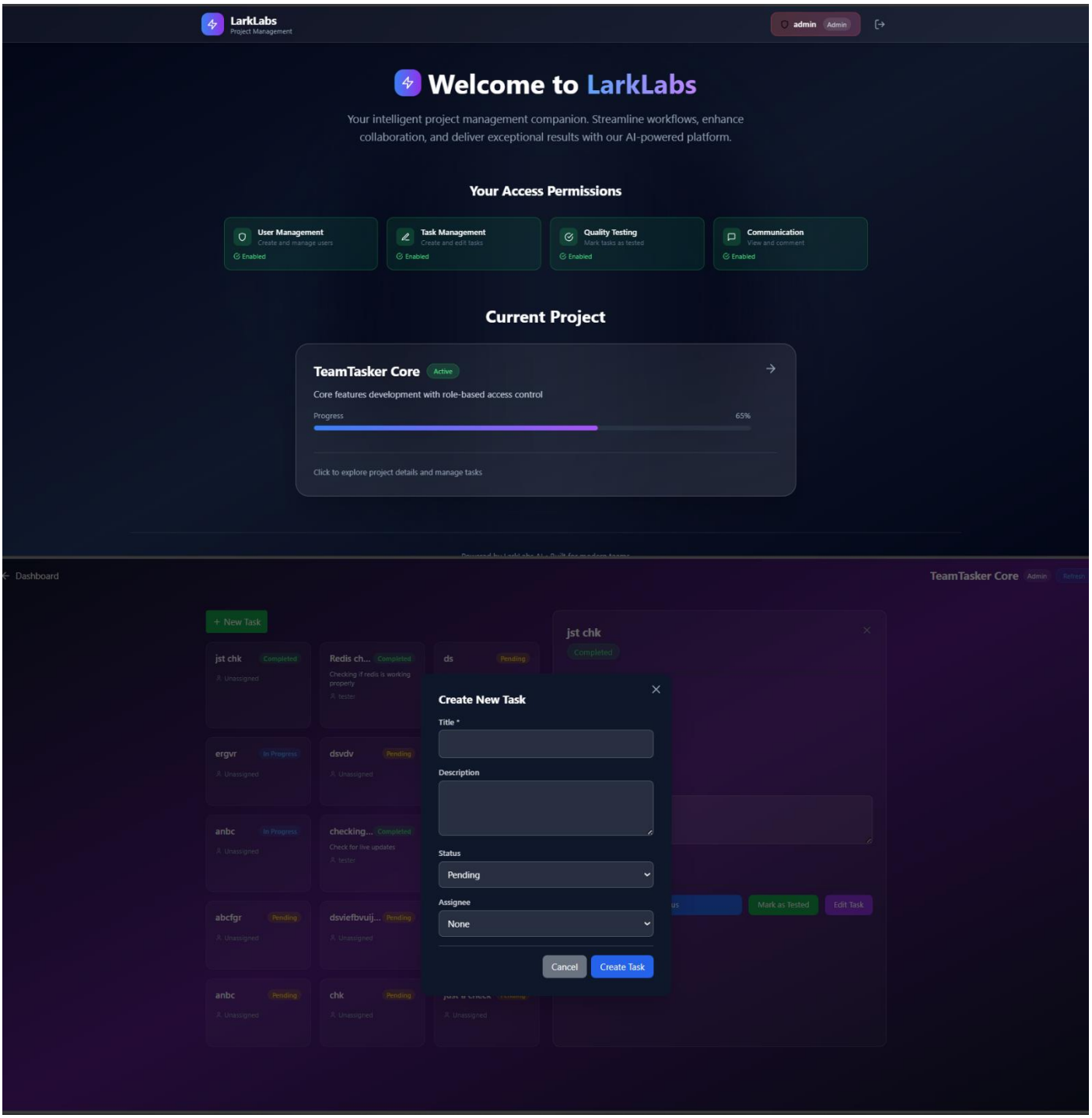
Sign in page=>Using RBAC I've assigned for five role

- admin admin123
- manager manager123
- tester tester123
- dev dev123
- viewer viewr123
- also newly Signed up person can also log in



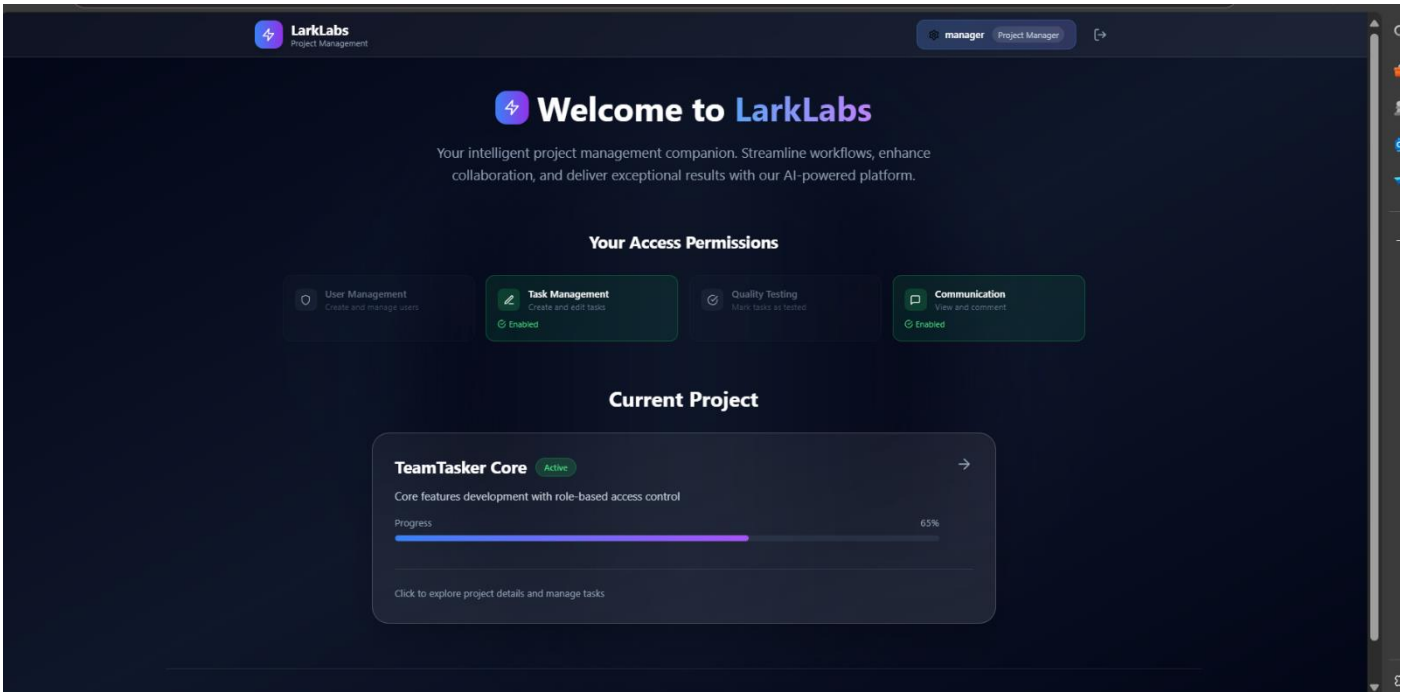
Admin Login page and DashBoard page

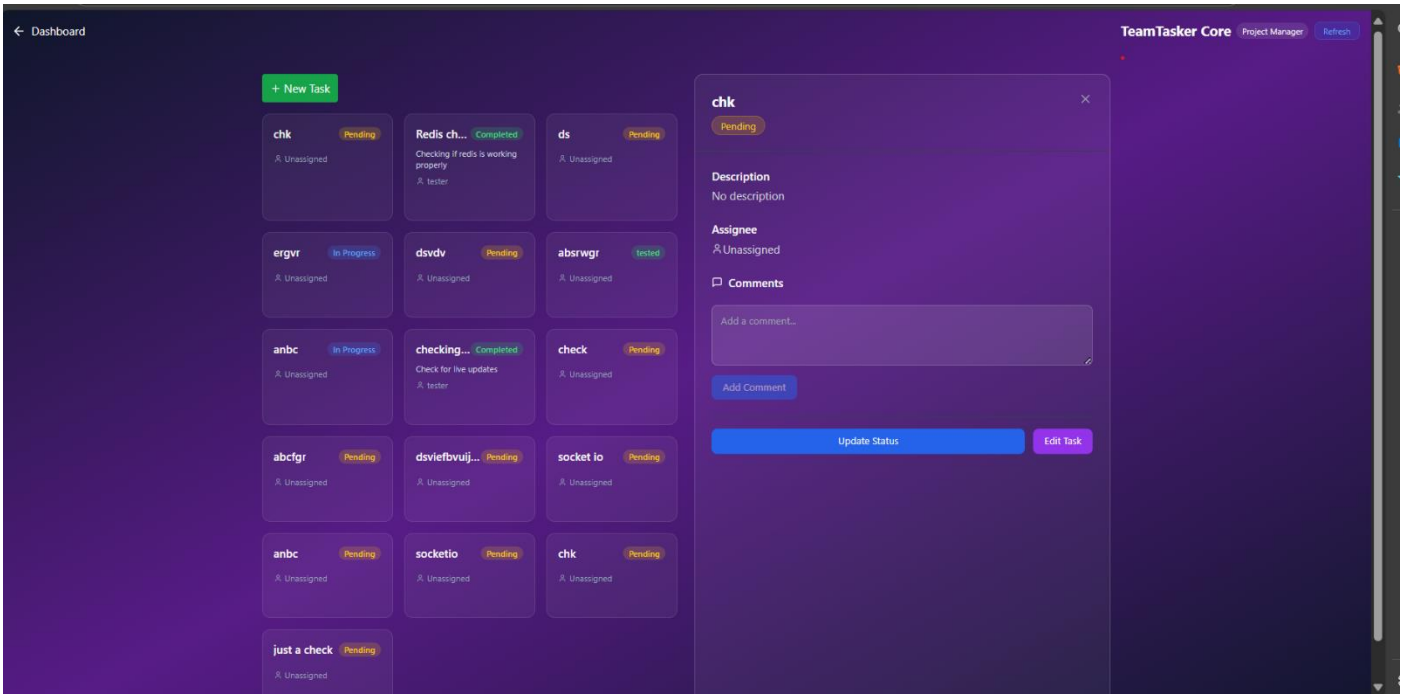
- Able create/delete task
- User management
- Task Management
- Quality management
- Communication



Manager Login page and DashBoard page

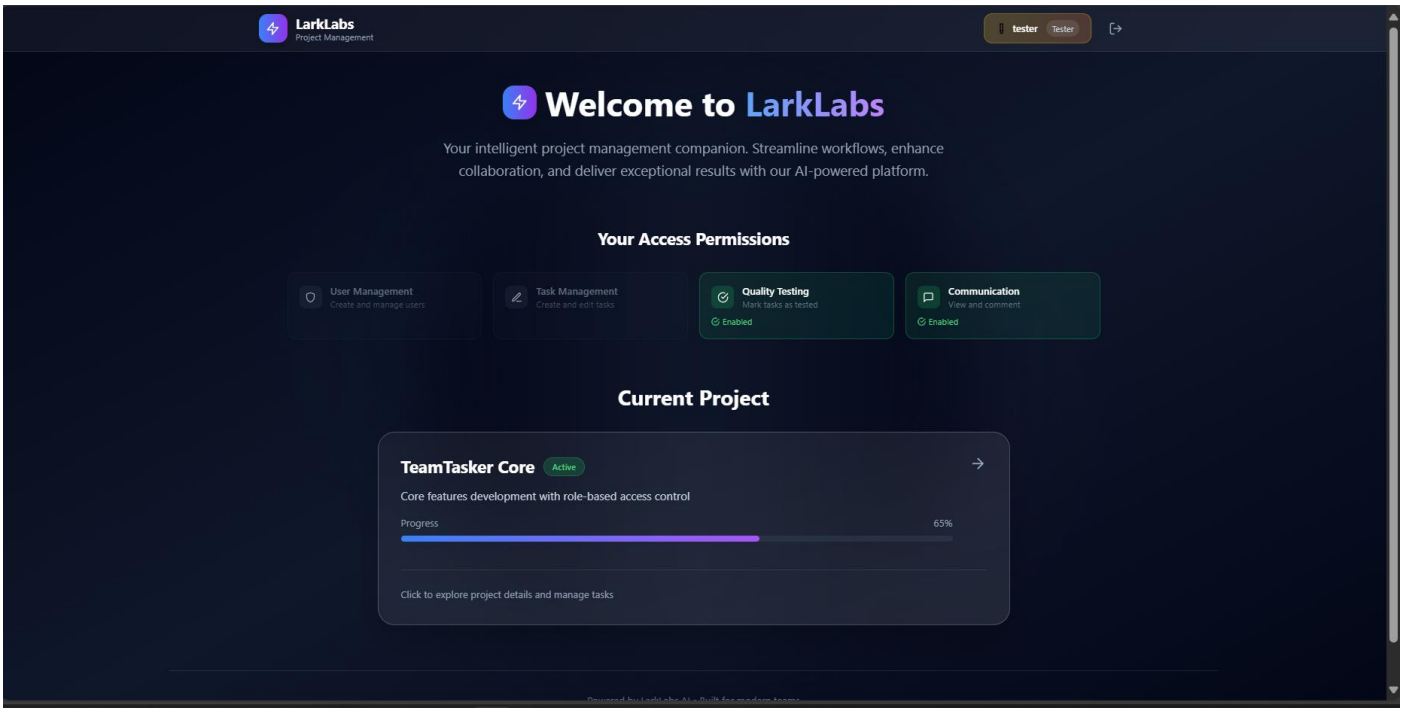
- Able create/delete task
- Task Management
- Communication

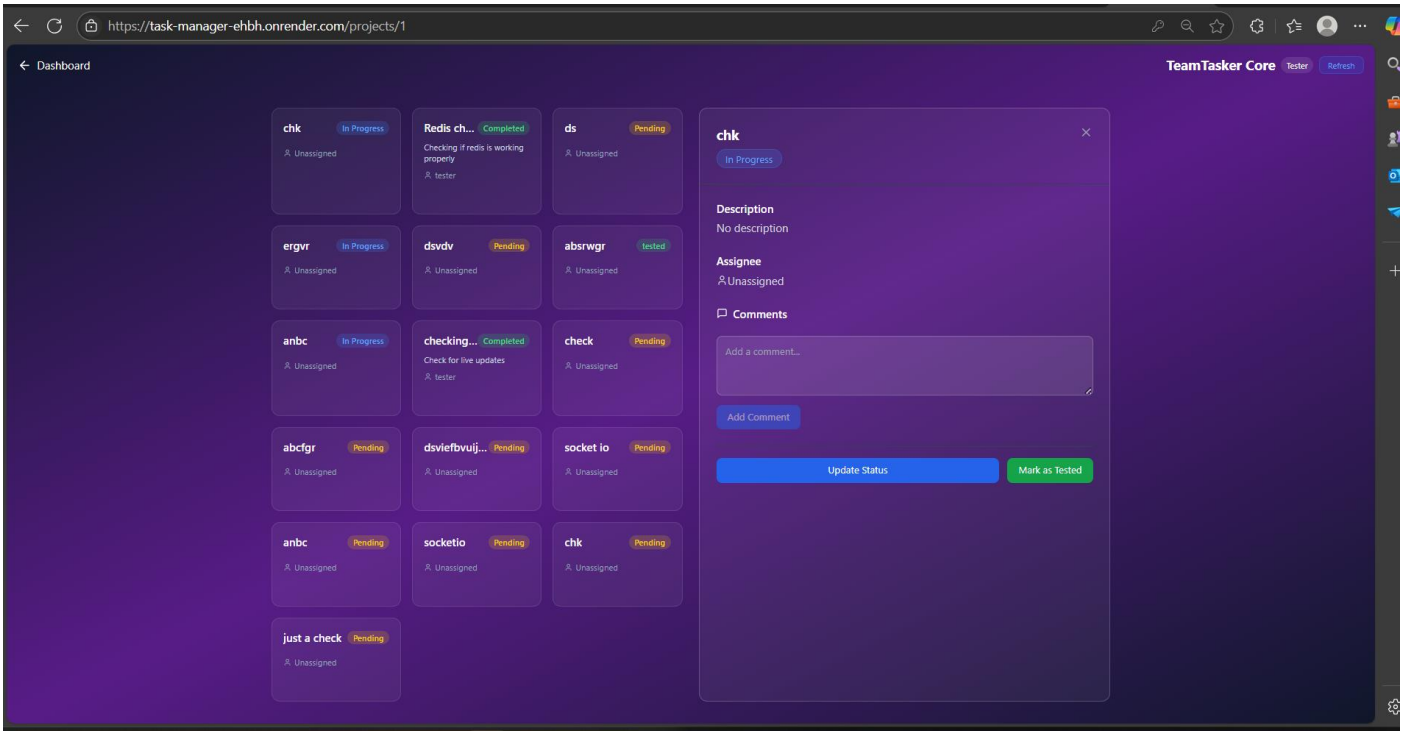




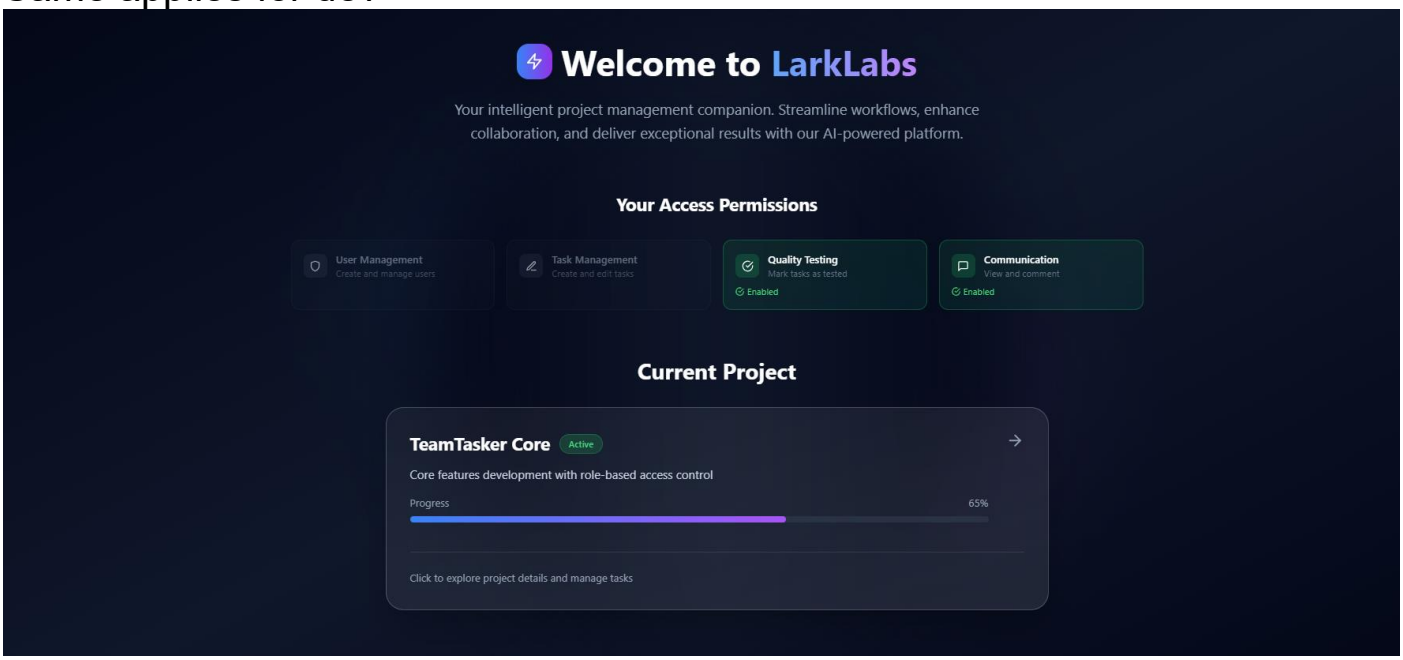
Tester Login page and DashBoard page

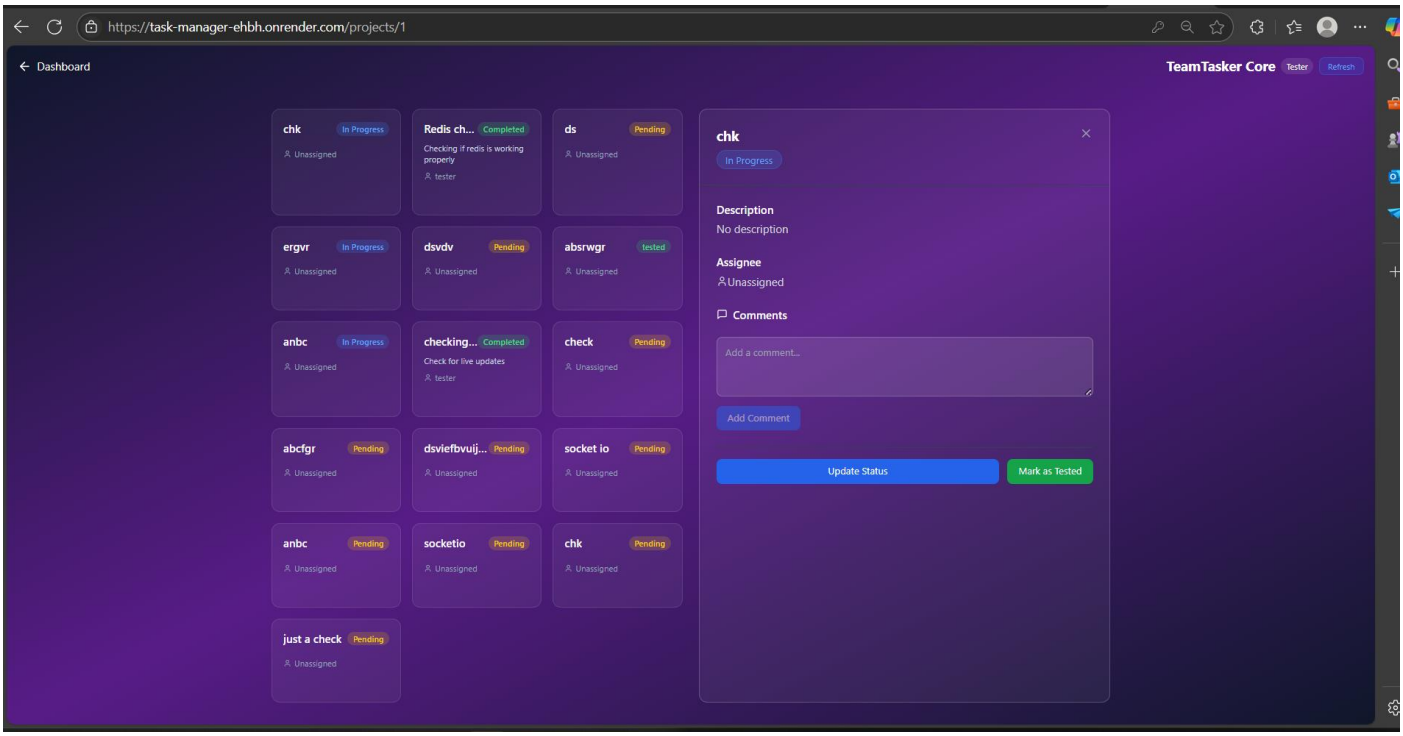
- Quality management
- Communication





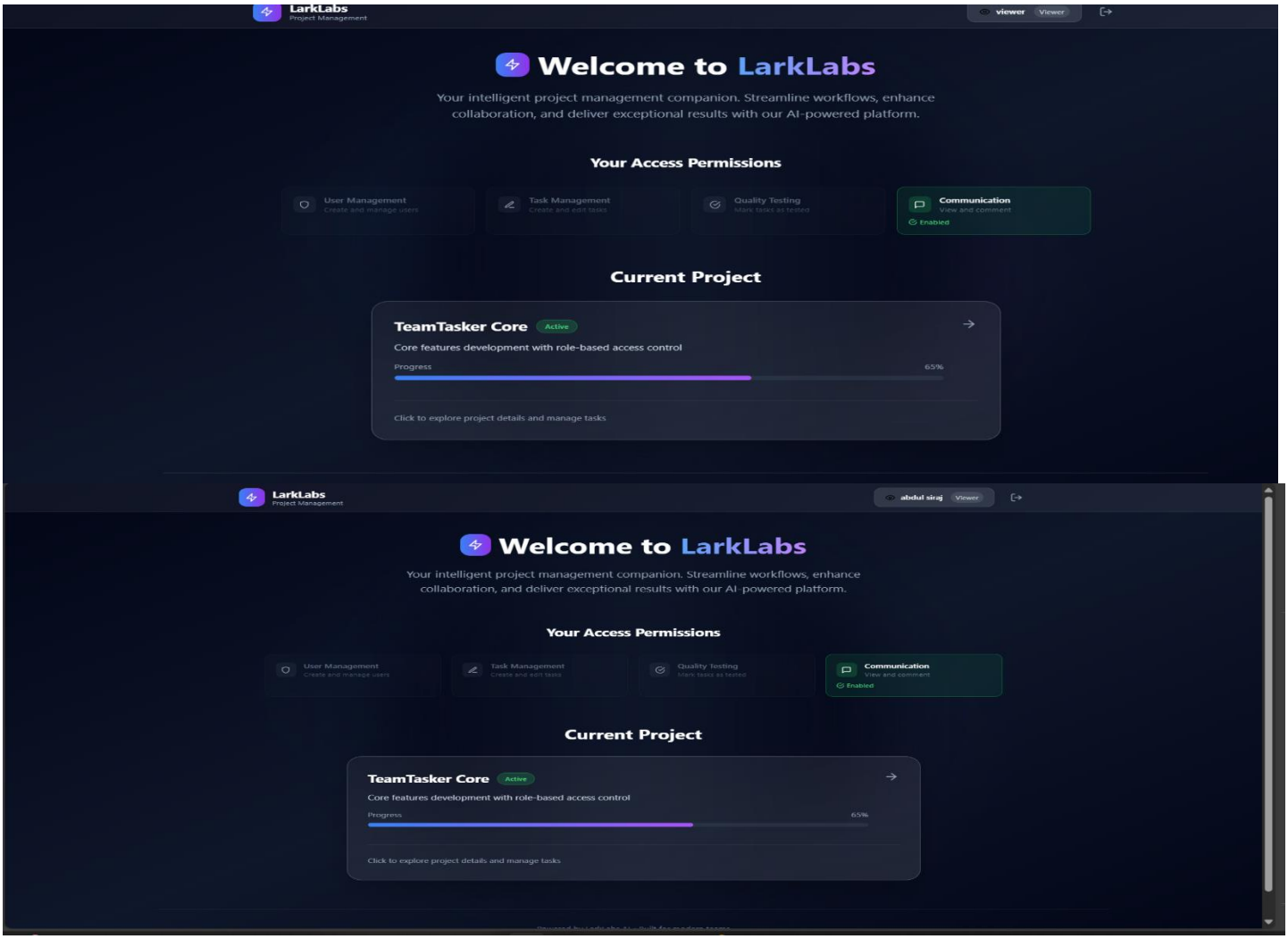
Same applies for dev

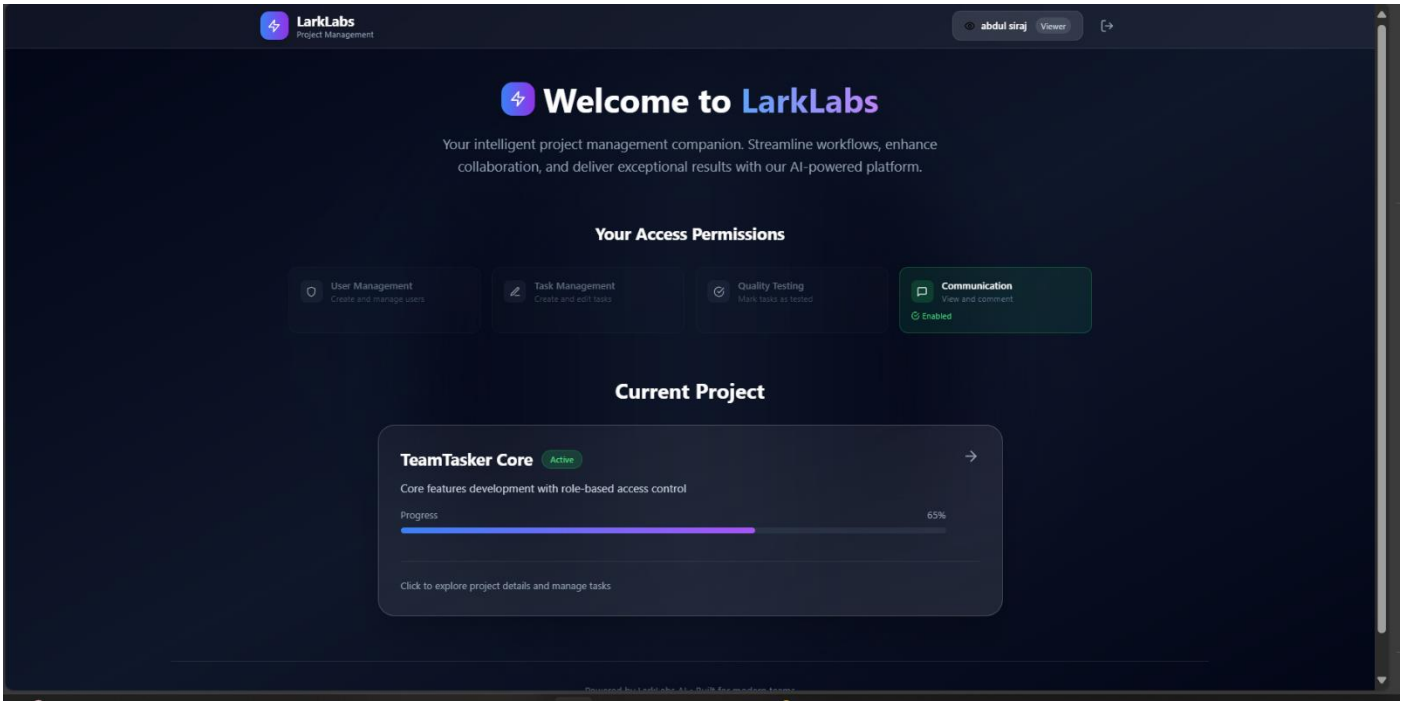




Signedup user ,viewer login page and DashBoard page

- Quality management
- Communication





Realtime Socket Io images