FSD Project Report

Since 1962

PixelForge Nexus:
A Secure MERN Stack Project Management Platform

Submitted by
Kshitij Nangare

Submitted to
Ms. Pooja Prajapati

Vivekanand Education Society's Institute of Technology Chembur, Mumbai – 400 074
October 2025

Executive Summary: PixelForge Nexus

PixelForge Nexus is a state-of-the-art, full-stack project management and collaboration platform specifically designed for the secure environment of **game development studios** like Creative SkillZ LLC. It is built on the **MERN Stack** (MongoDB, Express.js, React, Node.js) and operates under a fundamental 'secure-by-design' philosophy.

The platform's core objective is to centralize project lifecycle management—from task assignment to real-time communication—while rigorously implementing **Role-Based Access Control (RBAC)** to protect sensitive intellectual property (IP) and proprietary game assets. By integrating features such as **secure authentication (JWT with bcrypt)**, granular authorization for **Admin, Lead, and Developer** roles, and **real-time chat (Socket.io)**, PixelForge Nexus ensures that security is an active layer of the platform, not an afterthought. The successful development was accelerated by the strategic, ethical use of Large Language Models (LLMs) like Google Gemini and ChatGPT for boilerplate generation and security review. The final product is a scalable, intuitive, and highly secure system ready for deployment in a professional, IP-sensitive environment.

---

1. Introduction

1.1 Project Background and Motivation

The digital transformation of industries has brought unprecedented collaboration, but for sectors handling high-value Intellectual Property (IP), such as **game development**, the need for **secure, controlled environments** is critical. Traditional, generic project management tools often fall short on granular security controls, leaving sensitive assets vulnerable.

PixelForge Nexus was motivated by the imperative to address this security gap. It provides a specialized, centralized platform where project assets, daily progress reports, and team communications are managed under a **secure-by-design** architecture, mitigating the risk of unauthorized access and data leakage, which is paramount for studios like Creative SkillZ LLC.

1.2 Project Objectives and Scope

The core objective was to deliver a functional and secure full-stack application prototype that fulfills the following goals:

1. **Implement Robust Security:** Establish state-of-the-art user authentication and granular, server-side **Role-Based Access Control (RBAC)**.
2. **Facilitate Core Management:** Provide a seamless interface for **Project Lifecycle Management** and advanced **Task Management** (Kanban-style).
3. **Enable Secure Collaboration:** Integrate features for **Real-Time Communication** and **Secure Document Handling** (with MIME validation).
4. **Enhance Accountability:** Introduce **Daily Progress Reporting** for developers, visible to Leads and Admins.

1.3 Problem Identified

The primary challenge in collaborative, high-IP environments like game development is the lack of project management tools that prioritize security and granular access control.

1. **IP Leakage Risk:** Generic platforms often provide flat access structures, exposing proprietary game assets, source code, and confidential designs to all team members, significantly increasing the risk of data breaches or leaks.
2. **Inefficient Workflow:** Developers and Leads often switch between multiple tools (e.g., chat apps, file storage, task boards), leading to fragmented data, context switching, and reduced productivity.

3. **Lack of Accountability:** Without integrated daily reporting and role-specific visibility, tracking true progress and identifying blockers across different seniority levels becomes cumbersome and reactive.

1.4 Our Solution

PixelForge Nexus offers a **"secure-by-design" centralized hub** that resolves these issues through three key pillars:

1. **Granular Security:** Implements **JWT authentication** and strict server-side **RBAC** (Admin, Lead, Developer), ensuring users only see and interact with data relevant to their role and assigned projects (Principle of Least Privilege).
2. **Integrated Management:** Consolidates **Kanban task boards**, **real-time Socket.io chat**, and **secure document storage** into a single application.
3. **Enhanced Transparency:** Mandates **Daily Progress Reporting** for developers, providing Leads and Admins with real-time, structured insights into team workload and project velocity.

1.5 Target Audience

The platform is designed for professional organizations with sensitive IP, primarily:

- **Small to Mid-Sized Game Development Studios (e.g., Creative SkillZ LLC):** Organizations needing enterprise-level security without the overhead of massive, complex systems.
- **Creative Agencies & Design Firms:** Any company managing proprietary creative assets and needing strict control over who can access draft work and final files.
- **Distributed Software Teams:** Teams that rely entirely on digital collaboration and require a single source of truth for all project communication and artifacts.

1.6 Impact and Benefits

Area	Impact/Benefit
Security & IP Protection	**Mitigated Risk:** RBAC and secure file handling drastically reduce the surface area for unauthorized access and IP leakage.

Productivity	**Reduced Context Switching:** Centralized platform increases focus and efficiency. Task boards and reports streamline daily work.
Team Communication	**Instant Collaboration:** Integrated, project-specific chat (Socket.io) ensures all discussion is contextualized and persistent.

1.7 Feasibility and Viability

Factor	Assessment	Rationale
Technical Feasibility	**High**	The project relies on the mature, well-documented **MERN stack**, augmented by industry-standard security practices (**bcrypt, JWT**) and established real-time technology (**Socket.io**).
Market Viability	**Strong**	Addresses a clear need in the project management market for a **secure, IP-focused alternative** to generic tools, making it particularly attractive to the game/creative sector.
Scalability	**Medium-High**	Built on Node.js/MongoDB, the system is inherently scalable. Future production deployment would involve scaling the database (MongoDB Atlas) and deploying the server on platforms like AWS or Azure.

2. System Overview and Architecture

2.1 Key Features at a Glance

Feature Category	Core Functionality	Security/Differentiation
Access Control	User Registration, Login, Logout.	**JWT Authentication, bcrypt Hashing, Role-Based Authorization (Admin, Lead, Developer).**
Project Management	CRUD operations for projects, status tracking.	Admin/Lead permissions only for creation/editing.
Task Management	Kanban board (To Do, In Progress, Review, Complete), Assignment, Priority.	Leads/Admins assign tasks; Developers update status.
Collaboration	Project-specific chat, Secure document/link uploads.	**Socket.io authentication**, **Multer with MIME Type Validation**.
Accountability	Developers submit daily reports.	Unique constraint (one report/user/project/day) for data integrity.

2.2 Technology Stack: The MERN Foundation

The project is built on the highly scalable and efficient **MERN** stack, augmented with specialized libraries for security and real-time functionality:

Component	Technology Used	Rationale/Benefit
Frontend	**React.js, Vite**	Fast, component-based UI, accelerated development via Vite.
Styling	**Tailwind CSS**	Utility-first for rapid, responsive, and consistent styling.
State Management	**Zustand**	Lightweight, fast, and simple global state management for auth/user data.
Backend	**Node.js, Express.js**	Non-blocking I/O for high performance and scalability.
Database	**MongoDB Atlas**	Flexible NoSQL schema for rapidly changing game development requirements.
Authentication	**JWT, bcryptjs**	Stateless and secure token-based authentication with salted password hashing.

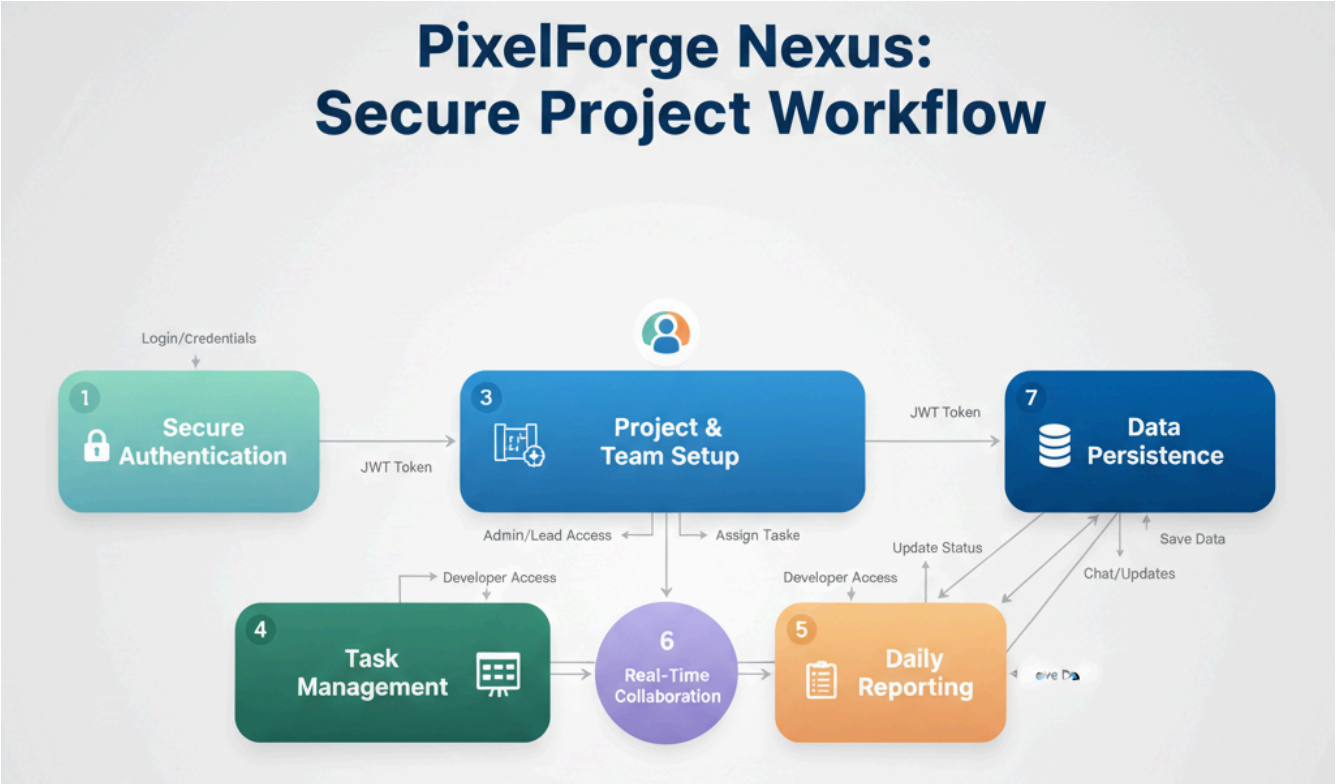
Real-Time	**Socket.io**	Efficient, low-latency communication for real-time chat.

2.3 Three-Tier Architecture

PixelForge Nexus follows a classic three-tier architectural model:

1. **Presentation Tier (Client):** The **React SPA** handles the user interface and logic, communicating with the backend via **Axios** for RESTful API calls and **Socket.io** for real-time data.
2. **Application Tier (Server):** The **Node.js/Express.js** server hosts the application logic, business rules, and security middleware. It processes requests, interacts with the database, and handles JWT token verification.
3. **Data Tier (Database): MongoDB Atlas** securely stores all application data, including user credentials, project specifications, and task statuses.

2.4 Workflow Diagram



---

3. Implementation and Secure Coding Practices

3.1 Security Implementation: Secure-by-Design

Security was the central design parameter, implemented at every layer of the application:

- **Password Storage:** All passwords are **hashed and salted using bcryptjs** before being stored in the database.
- **Authentication Flow: JSON Web Tokens (JWT)** provide a stateless, secure session mechanism. The token is stored securely (e.g., in an **HttpOnly cookie** or local storage) and validated via dedicated middleware on every protected server route.
- **Authorization Enforcement:** The `authorizeRoles(...)` middleware checks the user's role against the required permissions for an endpoint **on the server side**. This prevents unauthorized actions even if frontend checks are bypassed. The principle of **Least Privilege** is strictly applied.

- **Data Validation: Mongoose schema validation** and **Multer middleware** ensure all incoming data, including uploaded files, adheres to expected types and formats (e.g., MIME type checking for documents).

3.2 Core Feature Implementation

A. Role-Based Access Control (RBAC)

The system defines three core roles with distinct permissions:

- **Admin:** Full system oversight. Can manage users (CRUD), all projects, and all data.
- **Project Lead:** Can create/edit/delete their own projects, assign tasks, and manage project documents and team members.
- **Developer:** Can only view their assigned projects, update their assigned tasks' status, and submit daily progress reports.

B. Real-Time Collaboration (Socket.io)

Project-specific **real-time chat** is implemented using **Socket.io**. Crucially, the Socket connection is **authenticated using the JWT token**, and the server verifies that the user is an active member of the project before allowing them to join the project's chat room. This prevents unauthorized eavesdropping or messaging.

C. Secure Document Management

The Express backend utilizes the **Multer** middleware for file uploads. The implementation includes:

1. **File Type Restriction:** Explicitly whitelist allowed file MIME types (e.g., image/png, application/pdf) to prevent execution of malicious scripts.
2. **File Naming:** Files are renamed upon upload to prevent path traversal attacks.
3. **Storage:** Files are stored securely on the server or an S3-like bucket (for production scalability) and only served after an authorization check.

---

4. Development Methodology and Learning

4.1 Agile Development and Iterative Design

The project followed an **agile, iterative development cycle**. Core features (Authentication, Basic CRUD) were implemented first, followed by enhancements

(RBAC, Socket.io, Kanban board), allowing for continuous testing and refinement. The project's structure (Section 3.4 of original report) promoted clear separation of concerns (controllers, models, routes), ensuring a maintainable and scalable codebase.

4.2 Strategic Use of AI Assistance (LLMs)

The strategic use of Large Language Models (LLMs)—including Google Gemini, ChatGPT, and Grok—significantly enhanced the efficiency and quality of the project:

LLM Application	Impact on Development
Boilerplate Generation	Rapid creation of standard Express.js route structures and Mongoose schemas.
Security Best Practices	Assisted in reviewing JWT implementation and password hashing for adherence to industry standards.
Debugging & Error Resolution	Provided explanations and solutions for complex MERN stack integration issues.
Documentation	Aided in structuring and drafting the technical sections of this report.

This process reinforced the developer's understanding of secure coding practices and accelerated the timeline by focusing human effort on complex business logic and architecture.

Conclusion and Future Scope

The **PixelForge Nexus** project successfully delivered a secure, fully functional project management platform tailored for the sensitive requirements of game development studios. The commitment to a **secure-by-design** approach, powered by the **MERN stack**, has resulted in a robust system featuring:

- **Granular RBAC** protecting sensitive project data.
- **State-of-the-art authentication** via JWT and bcrypt.
- A comprehensive suite of collaboration tools, including a **real-time chat system**.

All primary objectives were met, and the final prototype provides a scalable and maintainable foundation.

Future Enhancements

Potential areas for future expansion include:

1. **Asset Versioning:** Integrating a more robust system for tracking changes to documents and game assets.
2. **Notification System:** Implementing email or in-app notifications for task assignments and due dates.
3. **Advanced Analytics:** Developing dashboards for Project Leads to analyze team productivity, task bottlenecks, and reporting compliance.

The success of PixelForge Nexus demonstrates the capability of modern full-stack development to create secure, customized, and efficient enterprise applications.

Code :

```
import mongoose from 'mongoose';  581.7k (gzipped: 145.3k)

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    trim: true,
  },
  email: {
    type: String,
    required: [true, 'Email is required'],
    unique: true,
    trim: true,
    lowercase: true,
  },
  password: {
    type: String,
    required: [true, 'Password is required'],
    minlength: [4, 'Password must be at least 4 characters'],
  },
  role: {
    type: String,
    enum: ['admin', 'lead', 'developer'],
    default: 'developer'
  },
  projects: [{
    projectId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Project'
    },
    roleInProject: {
      type: String,
      enum: ['lead', 'member']
    }
  }],
  mfaSecret: {
    type: String,
    select: false
  },
}, {
  timestamps: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});
```

Figure 1

```
import mongoose from 'mongoose';  581.7k (gzipped: 145.3k)

const projectSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Project name is required'],
    trim: true,
  },
  description: {
    type: String,
    required: [true, 'Description is required'],
    trim: true,
  },
  deadline: {
    type: Date,
    required: [true, 'Deadline is required'],
  },
  status: {
    type: String,
    enum: ['active', 'completed'],
    default: 'active'
  },
  lead: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: [true, 'Project must have a lead'],
  },
  team: [{
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    }
  }],
}, {
  timestamps: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});

// Indexes
projectSchema.index({ name: 'text' });
projectSchema.index({ status: 1, deadline: 1 });

const Project = mongoose.model('Project', projectSchema);
```

Figure 2

```
import mongoose from 'mongoose';  581.7k (gzipped: 145.3k)

const documentSchema = new mongoose.Schema({
  project: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Project',
    required: true
  },
  name: {
    type: String,
    required: true,
    trim: true
  },
  data: {
    type: Buffer,
    required: false // Optional for URL-based documents
  },
  contentType: {
    type: String,
    required: false // Optional for URL-based documents
  },
  size: {
    type: Number,
    required: false // Optional for URL-based documents
  },

  link: {
    type: String,
    required: false, // Optional for file-based documents
    trim: true
  },
  uploadedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  }
}, {
  timestamps: true
});
```

Figure 3

```
const messageSchema = new mongoose.Schema({
  projectId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Project',
    required: true
  },
  sender: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  content: {
    type: String,
    required: true,
    trim: true,
    maxlength: 1000
  },
  type: {
    type: String,
    enum: ['text', 'file', 'system'],
    default: 'text'
  },
  readBy: [{
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    },
    readAt: {
      type: Date,
      default: Date.now
    }
  }]
}, {
  timestamps: true
});

// Index for efficient querying
messageSchema.index({ projectId: 1, createdAt: 1 });

const Message = mongoose.model('Message', messageSchema);
export default Message;
```

Figure 4

```
import { Server } from 'socket.io';  311.2k (gzipped: 66.7k)
import Project from '../models/Project.js';
import Message from '../models/Message.js';

const setupSocket = (server) => {
  const io = new Server(server, {
    cors: {
      origin: process.env.FRONTEND_URL || "http://localhost:5173",
      methods: ["GET", "POST"],
      credentials: true
    },
    transports: ['websocket', 'polling']
  });


  // Store online users
  const onlineUsers = new Map();

  io.on('connection', (socket) => {

    // Handle join project
    socket.on('join-project', async (projectId) => {
      if (!projectId) {
        socket.emit('error', 'Project ID is required');
        return;
      }

      try {
        socket.join(projectId);

        // Send previous messages
        const messages = await Message.find({ projectId })
          .populate('sender', 'name email')
          .sort({ createdAt: 1 })
          .limit(50);

        socket.emit('previous-messages', messages);

      } catch (error) {
        console.error('Error fetching messages:', error);
        socket.emit('error', 'Failed to load messages');
      }
    });
```

Figure 5

```
socket.on('send-message', async (data) => {
  try {
    const { projectId, sender, content } = data;

    if (!projectId || !sender || !content) {
      socket.emit('error', 'Missing required fields');
      return;
    }

    // Verify user has access to project
    const project = await Project.findById(projectId);
    if (!project) {
      socket.emit('error', 'Project not found');
      return;
    }

    // Check if user is part of project team or lead
    const isTeamMember = project.team.some(member =>
      member.userId.toString() === sender._id
    );
    const isLead = project.lead.toString() === sender._id;

    const isAdmin = sender.role == 'admin';

    if (!isTeamMember && !isLead && !isAdmin) {
      socket.emit('error', 'Not authorized to send messages in this project');
      return;
    }

    // Save message to database
    const message = new Message({
      projectId,
      sender: sender._id,
      content: content.trim(),
      type: 'text'
    });

    await message.save();

    // Populate sender info
    await message.populate('sender', 'name email');
```

Figure 6

Output :

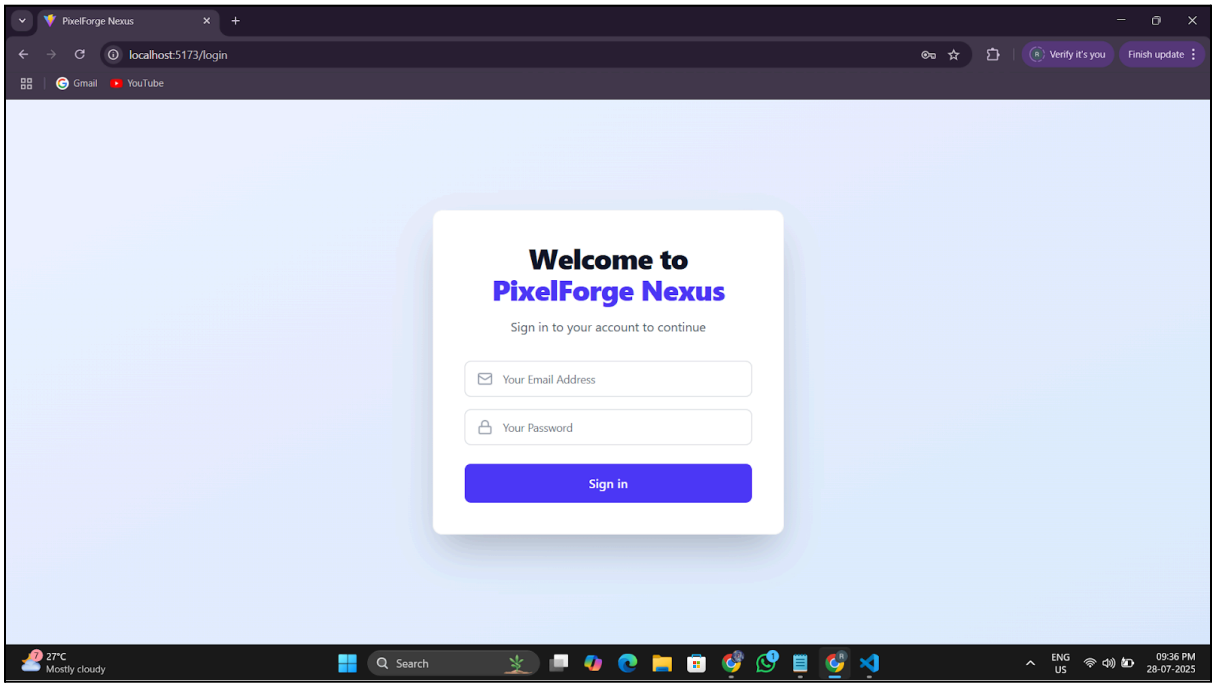


Figure 7

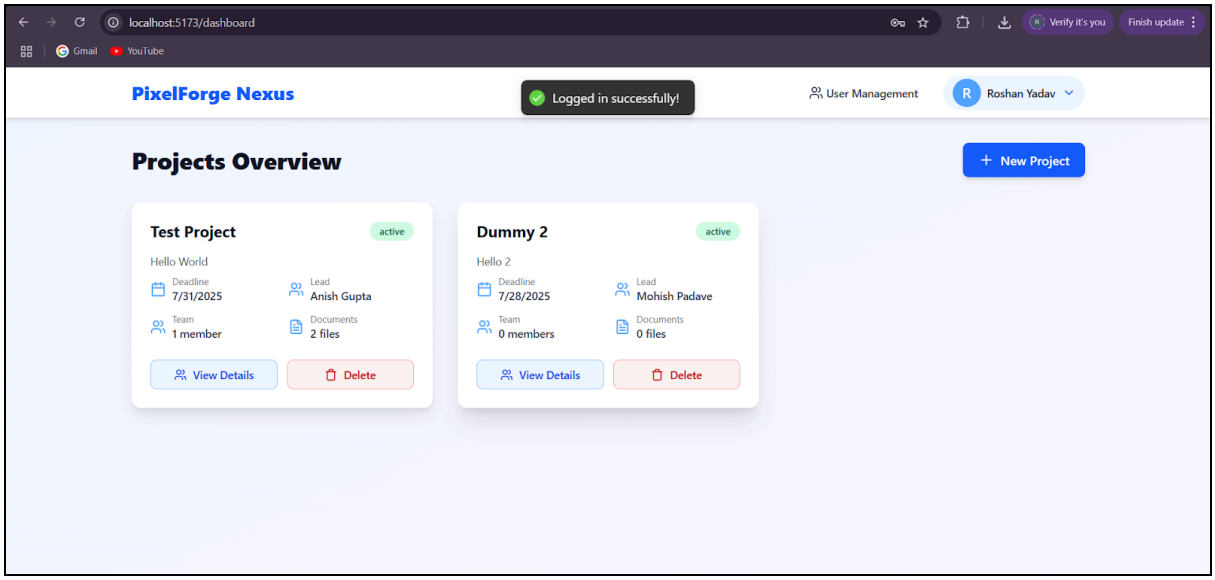


Figure 8

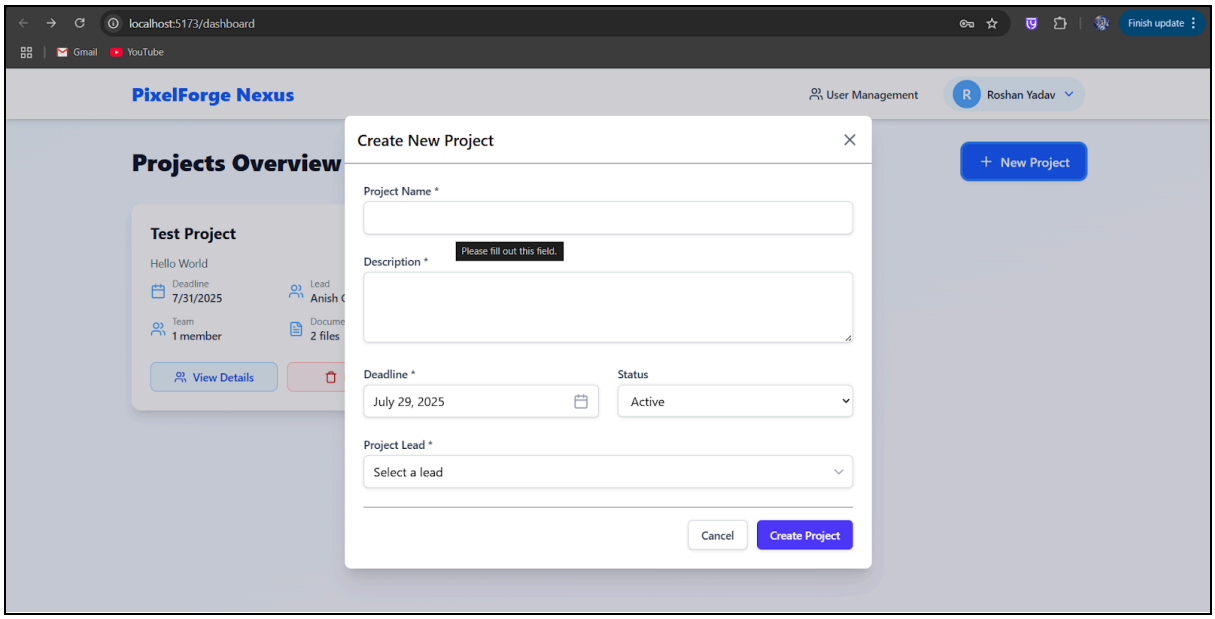


Figure 9

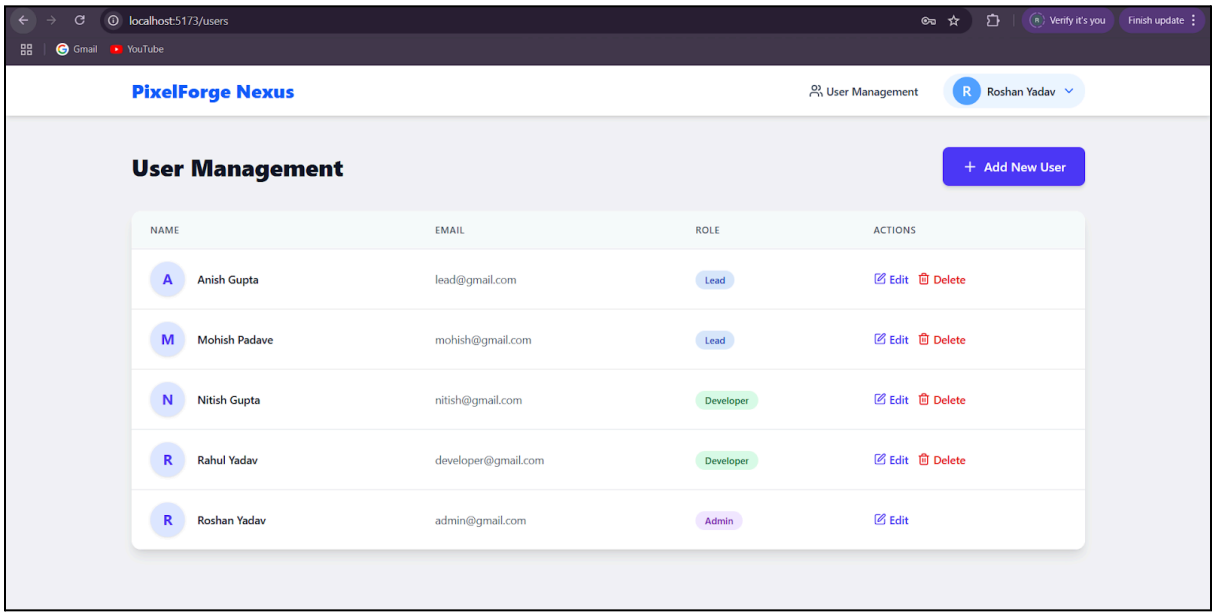


Figure 10

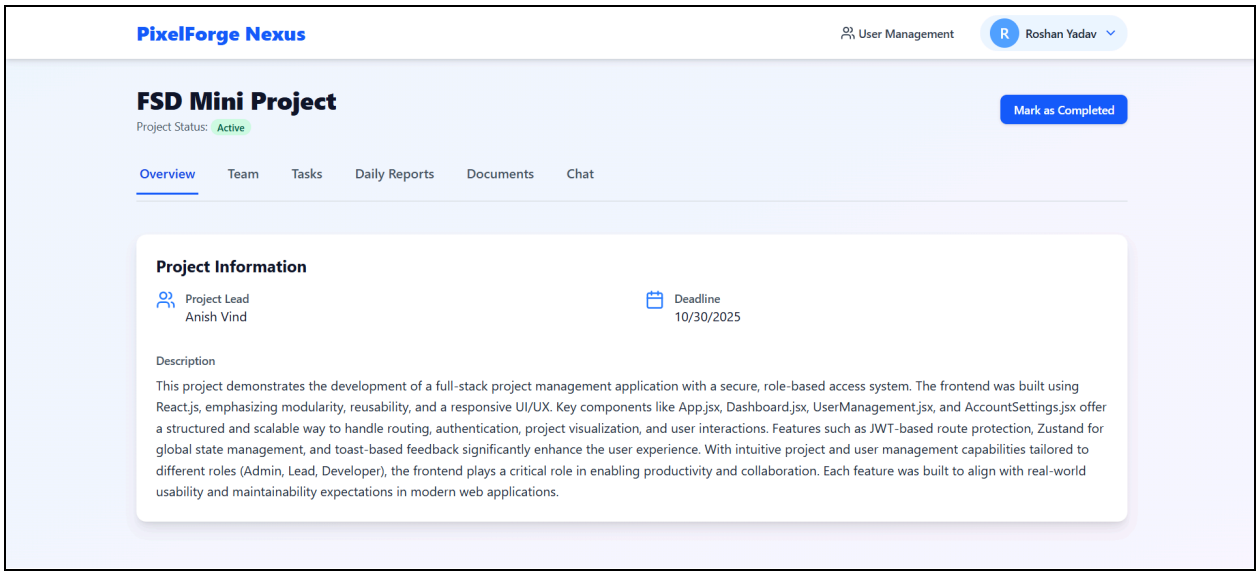


Figure 11

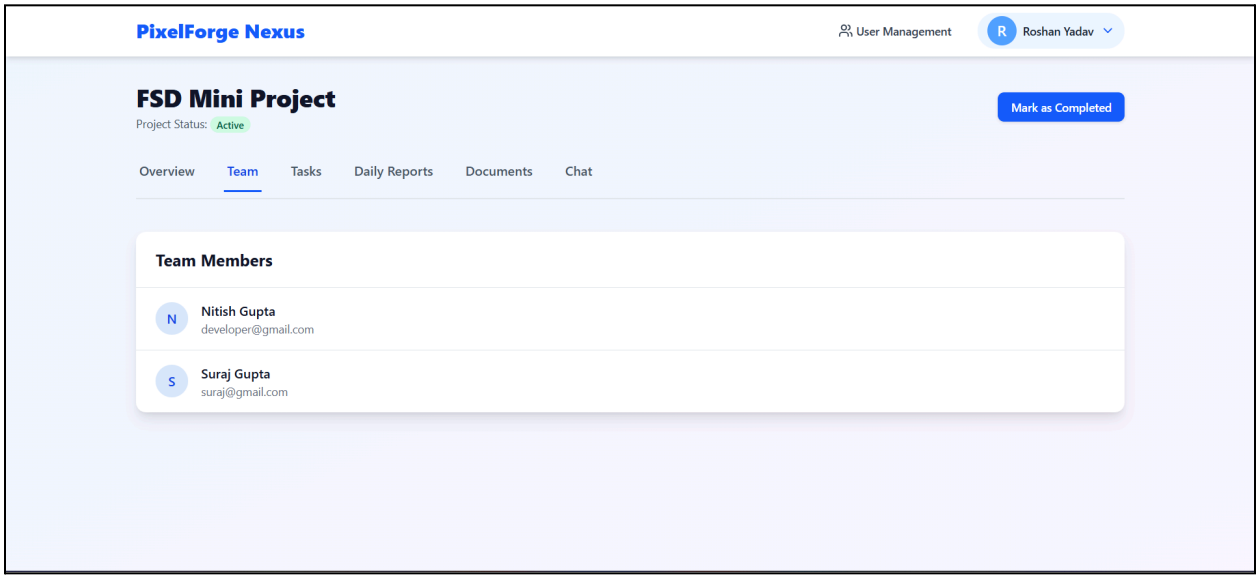


Figure 12

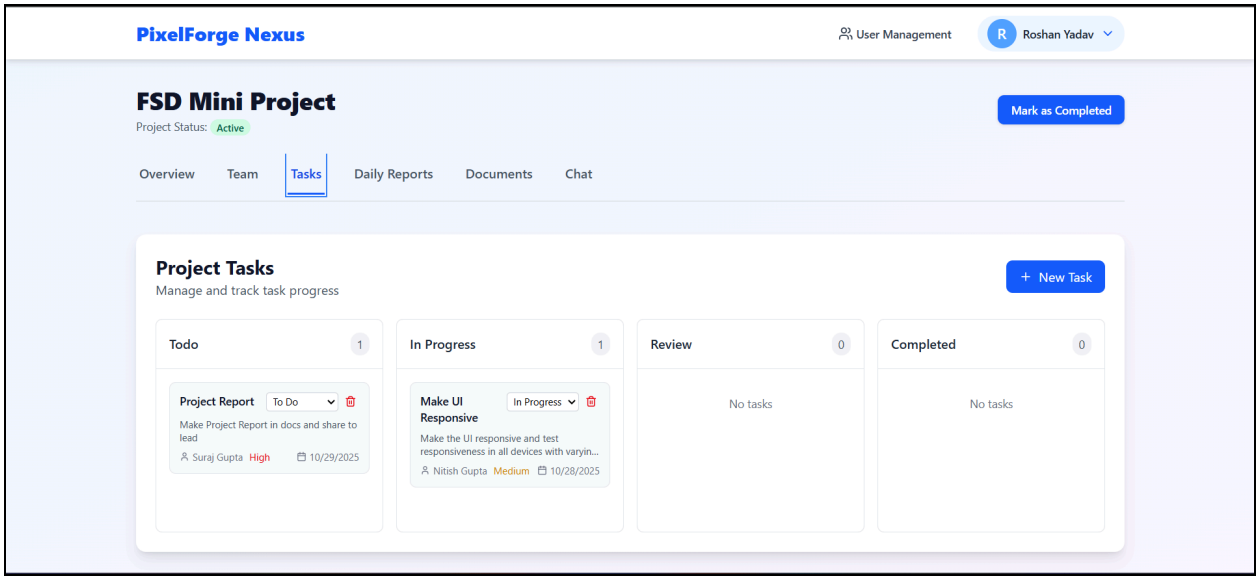


Figure 13

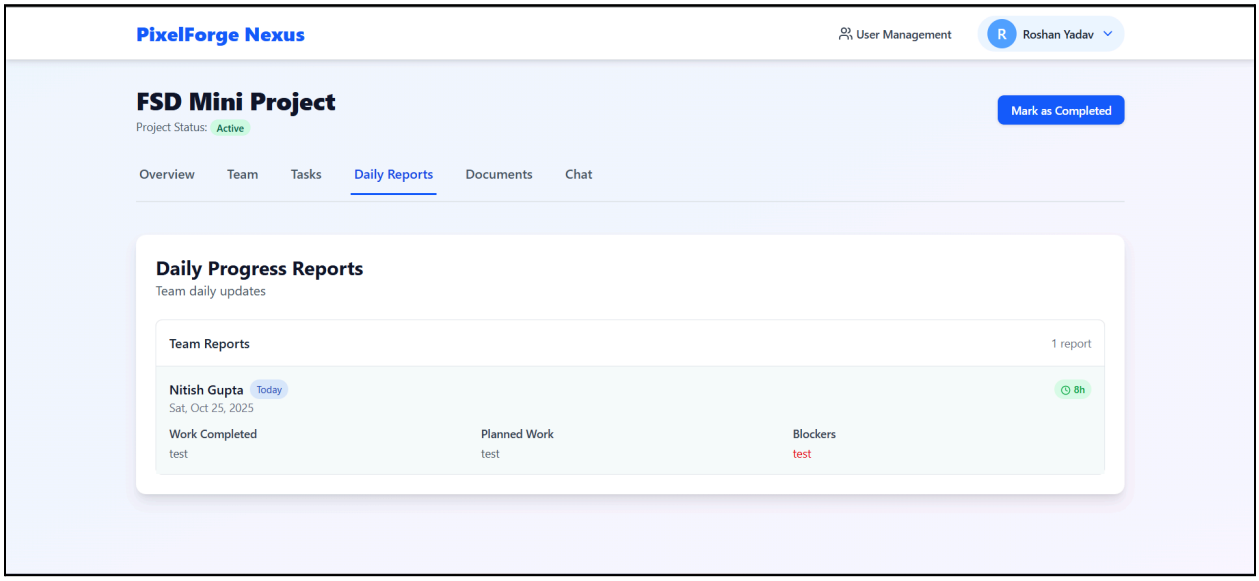


Figure 14

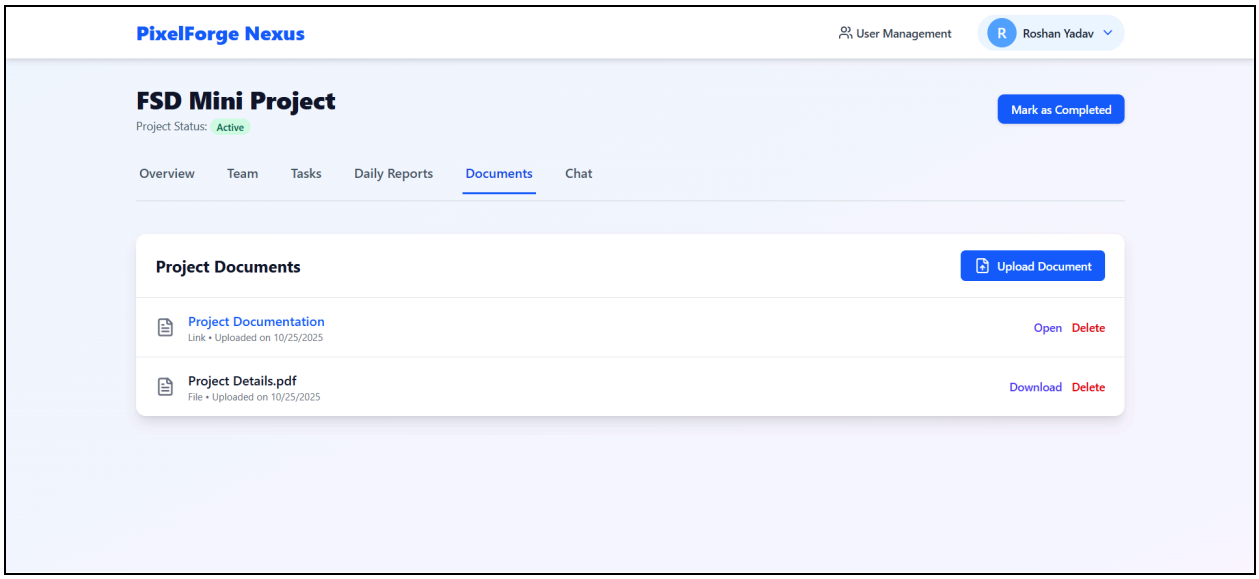


Figure 15

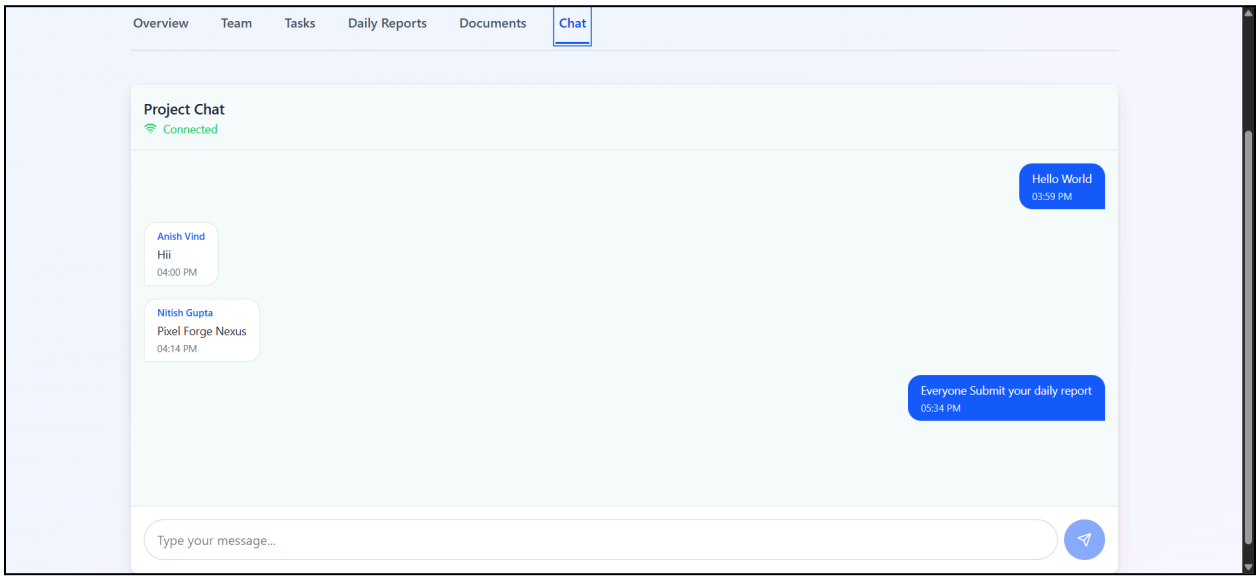


Figure 16

Conclusion

The PixelForge Nexus project successfully demonstrates the end-to-end development of a secure, modern full-stack web application. All primary objectives have been met, resulting in a functional system with:

- A robust and scalable MERN stack architecture.
- A comprehensive set of project management and collaboration features.
- A strong security posture with integrated authentication, authorization, and data protection.
- A responsive and intuitive user interface.

The implementation of granular **Role-Based Access Control (RBAC)** is central to the project's success, ensuring data integrity and confidentiality. The combination of a feature-rich React frontend and a secure Node.js backend provides a scalable and maintainable foundation. The final prototype is a functional, secure, and collaborative platform that effectively streamlines project, task, and team management, ready for further expansion and deployment.