**Project Report**

**Project Title:** PixelForge Nexus

**Name:** Anuj Namdev

**Roll Number:** 0105CD221012

**Submission Date:** 27/07/2025

**Institute:** Oriental Institute of Science and Technology (OIST) Bhopal

**Project Submission Links**

* Source Code Link: <https://github.com/Anujcodes2627/PexelForge-Nexus>
* Video Report Link: <https://drive.google.com/file/d/1Ycfwj8w8y2SOPh_hD12BERiZgYBcWq4X/view?usp=drive_link>

**Table of Contents**

1. Abstract

2. Introduction

3. System Overview

4. System Architecture

5. Key Modules

6. Security Measures

7. Technology Stack

8.Interaction Flow

**1. Abstract**

PixelForge Nexus is a secure, full-stack game development project management system designed to manage projects, users, and project-related documents with a strong focus on role-based access control. The system supports multiple user roles—Admin, Project Lead, and Developer—each having limited, clearly defined access. It incorporates secure login mechanisms, document upload capability, and project lifecycle management, making it a practical prototype for real-world secure development practices.

**2. Introduction**

In modern software development, especially in collaborative and distributed environments, **security**, **access control**, and **user role separation** are essential components of any robust application. As software teams grow and projects become more complex, enforcing clear boundaries between what each user can see or do becomes critical to avoid misuse, data leaks, or accidental errors.

**PixelForge Nexus** is developed as part of a *Secure Design and Development* academic assignment to demonstrate the **practical application of secure software engineering principles**. It is designed to simulate a realistic game development project management system, allowing different team members to interact based on their responsibilities, while maintaining strict access policies.

This system showcases how role-based access control (RBAC), protected routes, and secure authentication can be implemented in a real-world scenario. It also includes document handling features to mimic real project workflows, such as file sharing, deadline tracking, and project completion.

PixelForge Nexus addresses the **specific needs of distributed game development teams**, where multiple users work under different roles and need a centralized, secure space to manage progress. By applying concepts like **JWT-based authentication**, **frontend and backend route guarding**, and **data validation**, this project serves as a foundational prototype for scalable, secure team-based applications.

Moreover, the use of the **MERN stack** (MongoDB, Express, React, Node.js) ensures that the project is both modern and aligned with current industry practices, providing a full-stack demonstration of how secure systems are built and deployed.

**3. System Overview**

The PixelForge Nexus system follows a **Role-Based Access Control (RBAC)** model with three distinct user roles—**Admin**, **Project Lead**, and **Developer**—each having well-defined permissions and limitations to ensure secure and efficient operation.

**Authentication and Authorization**

* All users must log in using valid credentials.
* Upon login, users receive a **JWT token** which is used for all secure requests.
* The frontend uses **React Protected Routes** to prevent unauthorized access.
* The backend applies role-specific middleware to guard sensitive endpoints.

**Admin Role – Superuser**

Admins act as system managers with the highest level of authority. They are responsible for:

* **User Management**
  + Add new users (Lead, Developer) via the registration form.
  + Manage user roles and ensure correct designation.
* **Project Management**
  + Create new projects with name, description, and deadline.
  + Assign a **Project Lead** during project creation.
  + Remove existing projects when no longer needed.
  + Mark projects as "Completed" or revert to "Active".
* **Access & Permissions**
  + View all users and all projects in the system.
  + Access every project’s document list and uploads.
  + Change project status or delete it entirely if needed.
* **Security Management**
  + Admins have access to password update functionality.
  + Can monitor project completion or team activity indirectly.

**Project** **Lead Role – Team Coordinator**

Project Leads are responsible for managing and coordinating projects assigned to them:

* **Team Assignment**
  + View all developers in the system.
  + Assign developers to their respective projects.
* **Project Interaction**
  + View all details of projects assigned to them.
  + Upload and manage project-specific documents.
  + View all documents uploaded by other developers or leads.
* **Restrictions**
  + Cannot delete or create projects.
  + Cannot add users to the system.
  + Can only assign developers to projects that are still “Active”.
* **Visibility**
  + Can only view projects assigned to them by Admin.

**Developer Role – Task Executor**

Developers are the most restricted users in the system. Their access is limited to their assigned tasks:

* **Project Access**
  + View only the projects they are assigned to.
  + View project name, description, deadline, and current status (Active/Completed).
* **Document Interaction**
  + View and download project documents.
  + Cannot upload or modify documents.
* **Account Management**
  + Can update their own password from the account settings page.
* **Restrictions**
  + Cannot create or delete projects.
  + Cannot view other users or projects not assigned to them.
  + Cannot assign roles or teams.

**4. System Architecture**

The architecture of **PixelForge Nexus** is built using the popular **MERN stack**—MongoDB, Express.js, React.js, and Node.js—designed for full-stack JavaScript-based development. This architecture allows for seamless integration between the frontend and backend, along with efficient data flow, state management, and secure operations.

**Architectural Design Overview**

* The system is organized into **client-side (frontend)** and **server-side (backend)** components, connected via secure **RESTful APIs**.
* **React Router DOM** is used to manage client-side routing while ensuring only authenticated users can access restricted pages.
* On the server side, **Express.js** handles API routing and middleware pipelines, enforcing security policies like JWT authentication and role verification.

**Core Components and Technologies Used:**

* **Frontend**:
  + Built with **React.js** for component-based UI development.
  + **Redux** is used for global state management (authentication, role handling).
  + **Tailwind CSS** provides responsive and modern UI styling.
  + Features **Protected Routes** to prevent unauthorized users from accessing sensitive views like /dashboard or /register.
* **Backend**:
  + Developed using **Node.js** and **Express.js**.
  + Implements a modular route-controller-model structure for clean organization.
  + Includes **role-based access control** using middleware (protect, adminOnly, leadOnly).
* **Database**:
  + Utilizes **MongoDB**, a NoSQL database, to store users, projects, and documents.
  + **Mongoose** ODM is used to define schemas and handle relationships (e.g., developers assigned to a project).
* **Authentication & Authorization**:
  + JWT (JSON Web Token) is used for secure user sessions.
  + Tokens are stored in Redux and passed in the Authorization header for each request.
  + Server-side middleware validates the token and attaches the user to req.user.
* **File Handling**:
  + **Multer** middleware is used to handle document uploads securely.
  + Uploaded files are stored in the uploads/ directory and linked to the corresponding project in the database.

**Interaction Flow:**

1. User logs in → JWT issued → Token stored in Redux
2. Frontend sends requests with token in headers → Backend verifies via middleware
3. Authorized APIs respond with project/user data based on role
4. UI updates accordingly with access to view/edit/upload only if role matches