## **📄 PixelForge Nexus – Secure System Report**

**Prepared by:**

Rohan Mandal

Full-Stack Java Developer

Karjat, Maharashtra, India

**Email**: rohanmandal913@gmail.com

**LinkedIn**: https://www.linkedin.com/in/rohan-mandal-3596a9257

**GitHub**: https://github.com/rohanmandal341

**Course**: B.Tech in Computer Science and Engineering, Sandip University

**Submission Date**:28 July 2025

**Prototype Deliverables:**

- Source Code Drive Folder: [PixelForge\_Nexus](https://drive.google.com/drive/folders/1VT9evKEdgrjUQeppong2TfoSJy5A3Kx9?usp=sharing)

- Video Walkthrough: [PixelForge\_Video.mov](https://drive.google.com/file/d/1AlBnzOyDnux5O37mYVNS2wEhBWTy4xm5/view?usp=sharing)

- Test Credentials:

• Admin: rohanmandal130@gmail.com / 12345678

• Project Lead: rohanmandal9860@gmail.com / 12345678

• Developer: mandalrohan650@gmail.com / 12345678

**Declaration:**

I confirm that this report, source code, and video demonstration are my original work prepared for the secure system prototype as per the given brief. This submission demonstrates secure design, development, and testing principles following best practices in full-stack web application security.

## 

## **1️⃣ System Design**

### **1.1 Overview**

PixelForge Nexus is an integrated, secure online project management platform developed for Creative SkillZ LLC. Its core purpose is to provide a streamlined system for managing projects, assigning tasks, and handling project documents, while ensuring **strict security controls** are enforced throughout.

The system was built using a **full-stack architecture**, with a ReactJS frontend, Spring Boot backend, and a MySQL database. Security was embedded at each layer, with **role-based access control (RBAC)**, **multi-factor authentication (MFA)**, and **JSON Web Tokens (JWT)** for secure stateless sessions.

The design aims to prevent unauthorized access, data leaks, and misuse of admin functions, while being simple enough for team members to use daily without friction.

### **1.2 Security Principles and Architecture**

To protect data and ensure correct privileges, several key principles guided the design:

**Role-Based Access Control (RBAC)** Every user is assigned one of three roles: **Admin**, **Project Lead**, or **Developer**. This guarantees that only users with the correct permissions can access sensitive functionality. For example, only Admins can create or delete projects or users. Project Leads can assign Developers to projects but cannot delete projects. Developers have the least privileges and can only view projects they are assigned to.

**Multi-Factor Authentication (MFA)** Passwords alone can be weak or stolen, so the system uses a second layer of identity verification. After logging in with a valid email and password, users must enter a **One-Time Password (OTP)** sent to their email. OTPs expire after five minutes and cannot be reused.

**JWT-Based Authentication** Instead of server-side sessions, PixelForge Nexus uses JWTs for stateless authentication. This means once a user logs in and verifies their OTP, a signed token is generated and sent to the frontend. The frontend stores the token temporarily and attaches it to every secured API request. JWTs are signed and contain user roles so RBAC can be enforced.

**Secure Password Storage** User passwords are hashed using **BCrypt**, an adaptive hashing algorithm that makes brute force attacks computationally expensive. Plain-text passwords are never stored or logged.

**Strict CORS and CSRF Controls** CORS is configured to allow only known frontend domains. CSRF protection is disabled on the backend because the system uses stateless JWTs, which don’t rely on cookies or server sessions.

**Input Validation** All user input — project names, descriptions, uploaded file names — is validated to prevent malicious data from entering the system. This reduces the risk of injection attacks.

### **1.3 Threat Model**

The following threats were considered during design and development:

* **SQL Injection (SQLi):** One of the most common risks for web apps. Mitigated by using Spring Data JPA, which uses prepared statements.
* **Cross-Site Scripting (XSS):** Handled by React’s built-in escaping for rendering input and by validating all user inputs server-side.
* **Brute Force Attacks:** Limited by enforcing strong password rules and using MFA to add another barrier.
* **Broken Access Control:** One of the top OWASP risks. Prevented through strict @PreAuthorize checks on all controller methods, ensuring only users with correct roles can perform sensitive actions.
* **Session Hijacking:** JWTs expire after 24 hours and are signed. If stolen, the attacker cannot alter the payload or extend its lifetime.
* **Weak Document Upload Controls:** Only Project Leads and Admins can upload files. Files are stored with unique UUIDs to prevent filename collisions or malicious overwrites.

### **1.4 Mitigations and Safeguards**

* **Hashed Passwords:** Even if the database were breached, password hashes would resist rainbow table attacks.
* **Short-lived Tokens:** JWTs have expiry times to limit damage if compromised.
* **No Self-Registration:** Only Admins can create users, so attackers cannot create fake accounts to escalate privileges.
* **Minimal Attack Surface:** Endpoints not in use are closed off. Error messages are generic to avoid information leaks.
* **Verified File Uploads:** Files are restricted by type and size on the backend.

## **2️⃣ Development Details**

### **2.1 Tools and Tech Stack**

* **Frontend:** ReactJS with TailwindCSS for styling. Axios is used for HTTP requests with automatic JWT headers.
* **Backend:** Spring Boot (Java 17) with Spring Security, Spring Data JPA for ORM, and JJWT for JWT operations.
* **Database:** MySQL, auto-managed with Hibernate schema generation. Test data used a local instance.
* **Email:** Spring JavaMailSender used for sending OTP emails. During development, a local SMTP server was simulated.
* **Version Control:** Git and GitHub were used for source management.
* **IDE:** Visual Studio Code (frontend) and IntelliJ IDEA (backend).

### **2.2 Development Process**

The project was built incrementally:

* First, the secure login system with hashed passwords and JWT authentication was implemented.
* Next, the MFA feature was added and tested with dummy email accounts.
* After verifying core auth, role-based dashboards were built for Admin, Project Lead, and Developer.
* Document upload and role-specific project assignment logic were added last.

### **2.3 Challenges and Assumptions**

A few challenges arose during development:

* Integrating email OTP delivery needed local SMTP configuration and testing with Gmail SMTP.
* JWT signing keys had to be managed securely; in production these would be environment variables.
* Role checks needed careful testing to ensure no bypass was possible.
* File uploads had to be validated on both frontend and backend to avoid injection or malware risks.

Assumptions made:

* No public user registration — only Admins add new users.
* No version control for uploaded files.
* Localhost used for SMTP during development.
* Password resets are manual (no self-service forgot-password implemented).

### 

### 

### **2.4 Source Code Structure**

/backend/src/main/java/com.pixelforge.controller → Auth, User, Project, Document Controllers

/backend/src/main/java/com.pixelforge.security → JWT Filter, Config, Mail Config

/backend/src/main/java/com.pixelforge.service → Business logic for Auth, MFA, Projects

/backend/src/main/java/com.pixelforge.entity → Entities: User, Project, Document, MFAOtp

/frontend/src/pages → Login, OTPVerify, Dashboards, Settings

/frontend/src/components → Navbar, Forms, Uploads, Tables

/frontend/src/api → Axios config for JWT

## **3️⃣ Security Testing and Results**

### **3.1 Manual Testing**

Manual testing was done using Postman:

* Verified login flow rejects wrong credentials.
* OTP must match and must not be expired.
* Checked that trying to POST/DELETE projects without a valid JWT fails.
* Tested unauthorized document uploads — these fail as expected.

### **3.2 Automated Checks**

JUnit tests covered:

* BCrypt hashing works correctly and rejects wrong passwords.
* JWT signing and parsing with tampered tokens.
* Upload endpoints reject disallowed file types.

### **3.3 Security Issues & Future Improvements**

* **Brute Force:** Would benefit from login rate limiting and CAPTCHA.
* **Email:** Using a real SMTP with SPF/DKIM would stop spoofing.
* **Token Revocation:** No current refresh/revoke; tokens expire naturally.
* **Password Recovery:** Should add a secure forgot-password flow with tokenized reset links.

## **4️⃣ Formal Methods**

A simplified flow diagram was used to verify expected behavior:

User → Login → Verify Password → Send OTP → User Enters OTP → Verify OTP → Issue JWT → Access Dashboard

Role-based checks:

* If role is Admin → Full CRUD
* If role is Project Lead → Only own projects, can assign Developers
* If role is Developer → View only assigned projects

Verified by testing each endpoint to confirm no bypass is possible.

## 

## **Conclusion**

PixelForge Nexus achieves its goal of providing a simple, secure, and robust project management solution with:

* Strong authentication
* MFA as default for all users
* Clear role-based access controls
* Safe file upload
* Stateless JWT sessions

The system meets the practical brief and demonstrates secure full-stack development principles for a modern web app.

## **Deliverables**

* **Source Code Drive Link:** [PixelForge\_Nexus](https://drive.google.com/drive/folders/1VT9evKEdgrjUQeppong2TfoSJy5A3Kx9?usp=sharing)
* **Video Walkthrough Link:** [PixelForge\_Video.mov](https://drive.google.com/file/d/1AlBnzOyDnux5O37mYVNS2wEhBWTy4xm5/view?usp=sharing)
* **Test Credentials:**
  + Admin: rohanmandal130@gmail.com / 12345678
  + Project Lead: rohanmandal9860@gmail.com / 12345678
  + Developer: mandalrohan650@gmail.com / 12345678