**SUNYANI TECHNICAL UNIVERSITY**

**FACULTY OF ENGINEERING**

**DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING**



**PROJECT REPORT**

**DESIGN AND IMPLEMENTATION OF PROJECT PORTFOLIO MANAGEMENT SYSTEM**

**A PROJECT REPORT PRESENTED TO THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING, SUNYANI TECHNICAL UNIVERSITY, IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF HIGHER NATIONAL DIPLOMA IN ELECTRICAL AND ELECTRONIC ENGINEERING**

**BY**

**ADJEI SYLVESTER BAFFOE O6220287**

**BIPUAH UMAR BABA 06220748**

**ILLIYASU MOHAMMED NASRULLAH 06220054**

**JULY**

**Chapter 3: Materials and Methods**

**3.1 Introduction**

The development of the Project Portfolio Management System (PPMS) follows a structured and methodical approach to ensure the delivery of a reliable, scalable, and user-centered solution. This chapter outlines the materials, tools, methodologies, and processes employed in the design and implementation of the system. It details the software and hardware used, the system architecture, database design, development methodology, and testing procedures.

The goal is to ensure that the PPMS is developed using modern, industry-standard practices while meeting the academic and operational needs of the department. The chosen technologies—Node.js, React/Next.js, and PostgreSQL—support rapid development, real-time functionality, and long-term maintainability.

3.2 **System Development Methodology**

The PPMS was developed using an Agile iterative approach, specifically adapted for academic project timelines. Agile was selected due to its flexibility, emphasis on user feedback, and ability to deliver functional increments early in the development cycle [1].

3.2.1 Why Agile?

* Allows continuous feedback from stakeholders (students, supervisors).
* Supports incremental delivery: core features (login, project submission) were built first.
* Enables quick identification and correction of issues.

The development was organized into four sprints, each lasting one week:

1. Sprint 1: Requirements gathering, UI wireframing, and environment setup.
2. Sprint 2: Backend API development (Node.js + Express) and database modeling.
3. Sprint 3: Frontend implementation (Next.js) and integration with backend.
4. Sprint 4: Testing, bug fixing, and documentation.

This iterative model ensured that the system evolved based on real user needs and technical feasibility.

3.3 Functional and Non-Functional Requirements

Before development, requirements were gathered through interviews and use case analysis.

3.3.1 Functional Requirements

|  |  |  |
| --- | --- | --- |
| FR1 | Role-Based Authentication | System shall support login for students, supervisors, and admins using JWT. |
| FR2 | Project Submission | Students shall submit project title, abstract, keywords, and supervisor. |
| FR3 | Searchable Project Database | Users shall search projects by title, keyword, year, or supervisor. |
| FR4 | Progress Tracking | Supervisors shall view milestones and student progress. |
| FR5 | File Upload | Students shall upload documents (PDF, DOCX) to their project. |
| FR6 | Messaging System | Students and supervisors shall exchange messages within the app. |
| FR7 | Admin Dashboard | Administrators shall view project statistics and manage users. |
| FR8 | Report Export | System shall generate reports in PDF and CSV formats. |

3.3.2 Non-Functional Requirements

|  |  |  |
| --- | --- | --- |
| NFR1 | Performance | Search results shall load within 2 seconds. |
| NFR2 | Security | Passwords shall be hashed; API shall use JWT for authentication. |
| NFR3 | Usability | Interface shall be intuitive for non-technical users. |
| NFR4 | Scalability | System shall support up to 500 users. |
| NFR5 | Availability | System shall be available 24/7 with 99% uptime. |
| NFR6 | Maintainability | Code shall be modular and well-documented. |

These requirements guided the design and implementation of the system.

**3.4 System Architecture**

The PPMS follows a three-tier web architecture:

1. Frontend (Presentation Layer)
   * Built with React and Next.js for dynamic, server-rendered pages.
   * Responsive design using Tailwind CSS for mobile and desktop compatibility.
   * Real-time updates via WebSocket (planned for future enhancement).
2. Backend (Application Layer)
   * Developed using Node.js with Express.js as the web framework.
   * RESTful API design for handling requests (e.g., /api/projects, /api/users).
   * JWT (JSON Web Tokens) for secure authentication and session management.
3. Database (Data Layer)
   * PostgreSQL used for structured storage of project metadata, user roles, files, and messages.
   * Prisma ORM used to interact with the database, ensuring type safety and clean queries.

This architecture ensures separation of concerns, easy debugging, and independent scalability of each layer.

**3.5 Software and Hardware Tools**

**3.5.1 Software Stack**

|  |  |
| --- | --- |
| Node.js v18+ | Backend JavaScript runtime |
| Express.js | Web framework for API routes |
| Next.js 13+ | Frontend React framework with SSR |
| PostgreSQL 15 | Relational database |
| Prisma | Database ORM and schema migration |
| JWT | Secure user authentication |
| Bcrypt | Password hashing |
| VS Code | Code editor |
| Postman | API testing |
| Git & GitHub | Version control |
| Docker | Containerization for deployment |

**3.5.2 Hardware Requirements**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Development | Intel i5 | 8 GB | 256 GB SSD | Windows/macOS/Linux |
| Production | Intel Xeon | 16 GB | 500 GB SSD | Ubuntu Server 22.04 |

The system can be deployed on university servers or cloud platforms like AWS EC2 or Google Cloud.

**3.6 Database Design**

The database schema is based on an Entity-Relationship (ER) model with the following core entities:

3.6.1 Entities and Attributes

1. User
   * id, name, email, password\_hash, role (student/supervisor/admin), department, createdAt
2. Project
   * id, title, abstract, keywords, studentId, supervisorId, year, status (draft, submitted, approved, completed), createdAt
3. Milestone
   * id, projectId, title, dueDate, completed, notes, createdAt
4. File
   * id, projectId, fileName, filePath, uploadDate
5. Message
   * id, senderId, receiverId, projectId, content, read, timestamp

3.6.2 Relationships

* One User (student) → One Project
* One User (supervisor) → Many Projects
* One Project → Many Milestones, Files, Messages

This relational design ensures data integrity and supports efficient querying for search and reporting.

3.7 Development Process

The implementation followed these steps:

1. Environment Setup: Node.js, PostgreSQL, and Next.js initialized.
2. Database Modeling: Prisma schema defined and migrated.
3. Backend Development: REST API endpoints created for:
   * User authentication
   * Project CRUD operations
   * File upload
   * Messaging
4. Frontend Development: Pages built in Next.js:
   * Login / Register
   * Student Dashboard
   * Supervisor View
   * Admin Panel
5. Integration: Frontend connected to backend via API calls.
6. Testing: Unit and integration tests performed.

**3.8 Testing Strategy**

A multi-level testing approach was adopted to ensure system reliability.

**3.8.1 Types of Testing**

|  |  |  |
| --- | --- | --- |
| Unit Testing | Test individual functions (e.g., login) | Jest, Supertest |
| Integration Testing | Verify API ↔ Database interaction | Postman, Jest |
| User Acceptance Testing (UAT) | Validate with real users | Feedback forms, observation |
| Security Testing | Check for vulnerabilities | Manual review, Helmet.js |
| Performance Testing | Measure response time | Lighthouse, Apache JMeter (planned) |

3.8.2 Sample Test Cases

|  |  |  |
| --- | --- | --- |
| TC01 | Valid student login | Redirect to student dashboard |
| TC02 | Invalid password | Show "Invalid credentials" error |
| TC03 | Search project by "AI" | List all matching projects |
| TC04 | Submit project proposal | Data saved; success message shown |
| TC05 | Supervisor views student project | Milestones and files displayed |

All critical bugs were resolved before final deployment.

**3.9 Chapter Summary**

This chapter has detailed the materials and methods used in developing the PPMS. An Agile methodology was adopted for flexibility and stakeholder engagement. The system was built using Node.js, Next.js, and PostgreSQL, following a three-tier architecture. Requirements were clearly defined, and the database was modeled to support efficient data management. A comprehensive testing strategy ensured functionality, security, and performance.

The next chapter, Chapter 4: System Design and Implementation, will present the user interface, core modules, API structure, and screenshots (to be added by you), demonstrating how the system meets the intended objectives.

**References**

[1] R. Pressman and B. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. New York: McGraw-Hill, 2020.  
[2] Node.js Foundation, *Node.js Documentation*, 2023. [Online]. Available: [https://nodejs.org](https://nodejs.org/)  
[3] Vercel, *Next.js Documentation*, 2023. [Online]. Available: <https://nextjs.org/docs>  
[4] PostgreSQL Global Development Group, *PostgreSQL Documentation*, 2023. [Online]. Available: <https://www.postgresql.org/docs/>  
[5] Prisma Labs, *Prisma ORM Documentation*, 2023. [Online]. Available: <https://www.prisma.io/docs>