Software Design Specifications

for

Final Year Project Application

Prepared by:

SE22UCSE315 - Srija Lukka
SE22UCSE290 - Vidhitareddy Maddi
SE22UCSE196 - Parthavi K
SE22UCSE044 - BVeda sri
SE22UCSE030 - Ankita Sarkar

Document Information

Title: Projex

Prepared By:
Team 12

Preparation Date: 7th April 2024

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename

Table of Contents

1.Introduction

1.1Purpose

The purpose of this Software Design Specification (SDS) document is to provide a comprehensive design framework for the Final-Year Student Project Allocation System. This document serves as a guide for developers, designers, and stakeholders involved in the project, ensuring a structured approach to the system's architecture, components, and interactions.

The intended audience for this document includes:

- Developers – To understand system architecture, module interactions, and implementation details.
- Project Advisors – To ensure the system meets academic and administrative needs.
- QA Engineers – To reference system behavior for testing purposes.
- End-Users (Students & Faculty) – To gain an understanding of the system's functionalities.

1.2Scope

This Software Design Specification applies to the Final-Year Student Project Allocation System, which aims to streamline the process of assigning students to faculty-guided projects. The system will:

- Allow students to register and select project preferences.
- Enable faculty members to propose and manage projects.
- Implement an automated or semi-automated allocation algorithm based on predefined criteria.
- Provide an intuitive web-based frontend for seamless interaction.
- Ensure data security and user authentication.
- The system will impact students, faculty members, and administrative staff, making project assignment more efficient and transparent.

1.3Definitions, Acronyms, and Abbreviations

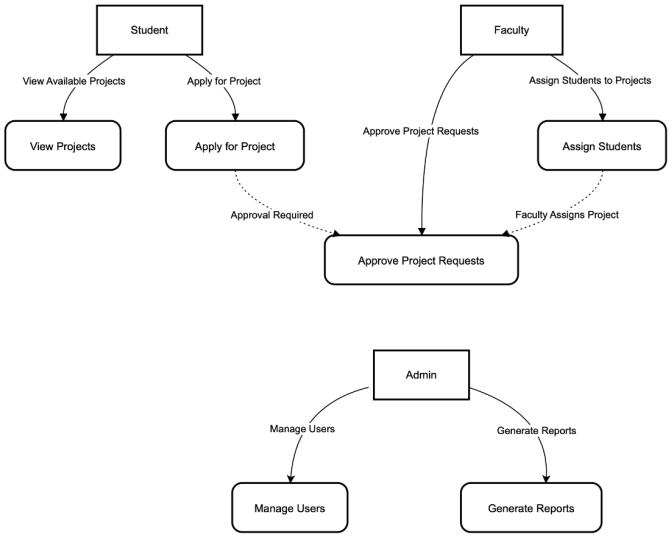
- UI – User Interface
- UX – User Experience
- DBMS – Database Management System
- API – Application Programming Interface
- Allocation Algorithm – The method used to assign students to projects based on preference, availability, and faculty constraints
- Frontend – The web-based user interface for interaction
- Backend – The server-side logic managing data processing and allocation

1.4References

IEEE Standard for Software Design Descriptions (IEEE 1016-2009)
Web Technologies (HTML, CSS, JavaScript, React.js)
Backend Frameworks (Node.js, Express.js)
Database Management (MongoDB / PostgreSQL)
Authentication Methods (OAuth, JWT-based authentication)
Allocation Algorithm – Based on faculty preferences, student choices, and system constraints

2.Use Case View

2.1Use Case



3. Design Overview

This section provides an overview of the entire software design for the *Final-Year Project Allocation System*. It references and complies with high-level design interface contracts, requirements, and the high-level module decomposition approach.

3.1Design Goals and Constraints

Design Goals:

- Provide an efficient and user-friendly platform for final-year students to view, apply for, and be assigned projects.
- Enable faculty members to approve and assign projects efficiently.
- Ensure a seamless admin interface for user and data management.

- Ensure scalability and maintainability for future enhancements.
- Secure authentication and role-based access control.

Constraints:

- The system must be accessible via web browsers.
- It must integrate with university databases for student and faculty authentication.
- The project selection process should be transparent and streamlined.
- The system must handle multiple concurrent users effectively.

3.2Design Assumptions

- The university provides an existing authentication system for students and faculty.
- Each faculty member is responsible for a fixed number of students.
- Projects are assigned based on faculty approval.
- The system will be hosted on a university or cloud server.
- Students can only apply for projects once per semester.

3.3Significant Design Packages

The system follows an MVC (Model-View-Controller) architecture, with distinct layers for ease of maintenance and scalability:

- Frontend (View Layer): Developed using React.js for dynamic and responsive UI.
- Backend (Controller & Model Layers): Implemented using Node.js and Express for handling requests, with MongoDB as the database.
- Authentication Module: Implements user login and role-based access control (RBAC).
- Project Management Module: Handles project posting, student applications, and approvals.
- Admin Dashboard: Provides features for user and system management.
- Notification System: Sends email and in-app notifications for updates on project applications.

3.4Dependent External Interfaces

External Application	Module Using the Interface	Functionality Description
University Auth System	Authentication Module	Authenticates users via university credentials
Email Service (SMTP)	Notification System	Sends emails to notify users about project status
Cloud Database (MongoDB)	Project Management, Authentication	Stores user and project-related data

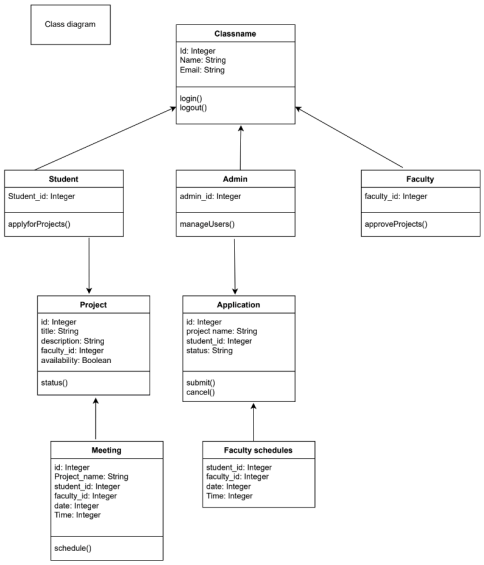
3.5Implemented Application External Interfaces (and SOA web services)

Interface Name	Module Implementing the Interface	Functionality Description
User Authentication API	Authentication Module	Handles login and role-based access
Project API	Project Management Module	Manages project CRUD operations
Notification API	Notification System	Sends email and in-app notifications

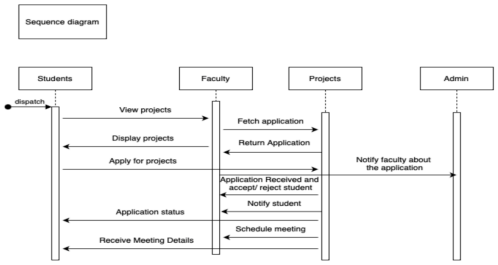
| Admin API | Admin Dashboard | Manages users, system settings, and logs |

4.Logical View

4.1Design Model



4.2Use Case Realization



5.Data View

5.1Domain Model

The core domain entities are:

- User
    - id
    - name
    - email
    - passwordHash
    - role (enum: STUDENT, FACULTY, ADMIN)
- Project
    - id
    - title
    - description
    - technologies (array)
    - keywords (array)
    - status (enum: OPEN, CLOSED, ONGOING)
    - screenshots (array of image URLs)
    - studentImages (array of image URLs)
    - contactName
    - contactEmail
    - facultyId (FK to User)
- Application
    - id
    - studentId (FK to User)
    - projectId (FK to Project)
    - status (enum: PENDING, APPROVED, REJECTED)
    - motivationLetter

Relationships:

- One Faculty can have multiple Projects
- One Student can apply to multiple Projects
- A Project can have many Applications

5.2Data Model (persistent data view)

This is implemented using Prisma ORM and MySQL. Here's a simplified data schema in SQL-like format:

- CREATE TABLE users (id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255), email VARCHAR(255) UNIQUE, password_hash TEXT, role ENUM('STUDENT', 'FACULTY', 'ADMIN'));
- CREATE TABLE projects (id INT PRIMARY KEY AUTO_INCREMENT, title VARCHAR(255), description TEXT, technologies JSON, keywords JSON, status ENUM('OPEN', 'ONGOING', 'CLOSED'), screenshots JSON, student_images JSON, contact_name VARCHAR(255), contact_email VARCHAR(255), faculty_id INT, FOREIGN KEY (faculty_id) REFERENCES users(id));
- CREATE TABLE applications (id INT PRIMARY KEY AUTO_INCREMENT, student_id INT, project_id INT, status ENUM('PENDING', 'APPROVED', 'REJECTED'), motivation_letter TEXT, FOREIGN KEY (student_id) REFERENCES users(id), FOREIGN KEY (project_id) REFERENCES projects(id));

5.2.1Data Dictionary

Field	Description
users.role	User role (STUDENT, FACULTY, ADMIN)
projects.status	Project lifecycle status
applications.status	Application workflow status
technologies	Tags indicating technologies used
keywords	Tags for search optimization
screenshots	URLs of project screenshots

6.Exception Handling

Exception	Cause	Handling Mechanism	Logging	Action
AuthenticationError	Invalid token/session	Return 401 Unauthorized	Yes	Redirected to login
ValidationError	Missing/Invalid form fields	Return 400 with field messages	No	Show form error
DatabaseConnectionError	DB is unreachable	Return 500 Internal Server Error	Yes	Retry or show error page
RecordNotFoundError	Project/User/Application not found	Return 404 Not Found	Yes	Display page not found
FileGenerationError	Project/User/Application not found	Return 500 and notify user	Yes	Login alert

7.Configurable Parameters

Configuration Parameter Name	Definition and Usage	Dynamic?
PORT	Port number for Express server	Yes
DATABASE_URL	MySQL connection string for Prisma	No
JWT_SECRET	Secret key for generating JWT tokens	No
PDF_STORAGE_PATH	Secret key for generating JWT tokens	Yes
MAX_UPLOAD_SIZE	Maximum allowed image upload size (MB)	Yes
SEARCH_INDEX_REFRESH_RATE	How often search index updates for new keywords	Yes

8.Quality of Service

8.1Availability

- The app is designed to be stateless and horizontally scalable.
- Downtime only during scheduled DB maintenance or deployment.
- Reports and assets (images, PDFs) are stored in cloud (S3 or Cloudinary) to avoid service disruption.

8.2Security and Authorization

- JWT-based authentication with role-based access control (RBAC).
- Faculty access is restricted to their own projects.
- Students can only see OPEN or assigned projects.
- Passwords hashed with bcrypt.
- Admin panel access guarded by role verification and CSRF tokens.
- Input validation + sanitization using Joi/Zod + helmet for HTTP headers.

8.3Load and Performance Implications

- Expected max load: 500+ users accessing simultaneously during peak season.
- Project and application tables projected to grow 1000+ records per year.
- Use of Prisma connection pooling and query optimization.

- PDF generation is cached or delayed using a queue (like BullMQ).
- Image loading is lazy-loaded on the frontend to reduce payload.

8.4Monitoring and Control

- Monitoring via tools like PM2, LogRocket, or Sentry for backend error tracking.
- Health check endpoint (/api/health).
- PDF generation logs for performance tracking.
- Admin dashboard to view system usage, application load, and report generation stats.
- Alerts configured for high CPU/memory or failed deployments.