Software Design Specifications

For

Faculty Project Matching Portal – Version 1.0

Prepared by:
Sri Spoorthi Vattem,
A.A. Vivek Showry,
Sidhanth Ranjan,
Adya Bahl,
Reddyshetty Kruthi

Table of Contents

1. Introduction

1.1 Purpose

This document details the Software Design for the Faculty Project Matching Portal—a web-based application developed to streamline the listing, application, and approval process for faculty projects at Mahindra University. It outlines the system architecture, module interfaces, data models, and quality requirements. This specification is intended for software developers, faculty members, students, and university administrators, guiding them in implementation and deployment.

1.2 Scope

The document covers all aspects of the design:

- **User Interfaces:** Based on HTML prototypes (homepage, projects view, and login page).
- **Backend Logic:** Developed in Django with custom user authentication and role management.
- **Data Storage:** Using SQLite for persistent storage.
- **Functional Components:** Allow faculty to post projects, students to browse and apply, and both to track application status.

The design influences the entire development process—from UI layout to database relationships and error handling.

1.3 Definitions, Acronyms, and Abbreviations

- **MU:** Mahindra University
- **UI:** User Interface
- **SRS:** Software Requirements Specification
- **SQLite:** Lightweight database engine
- **Django:** Python-based web framework
- **API:** Application Programming Interface
- **CRUD:** Create, Read, Update, Delete

1.4 References

- **Software Requirements Specification (FilledSRS_Team3.pdf):** Provides the overall product requirements.
- **Homepage Prototype (homepage.html):** Visual layout for the landing page.
- **Projects View Prototype (viewprojects.html):** Design for browsing available projects.
- **Login Interface (loginpagewithoutpics.html):** User authentication interface design.
- **Django Source Code:** Analysis from the extracted Django project (third working draft.zip) detailing models, views, forms, and URL interfaces.

---

2. Use Case View

2.1 Use Case

The system addresses multiple core functionalities, outlined briefly as follows:

- **User Registration and Authentication:**
    - **Actors:** New users (students, professors)
    - **Process:** Users sign up via a custom form; login via Django's authentication with custom enhancements.
- **Dashboard Navigation:**
    - **Actors:** Students and Professors
    - **Process:** A dashboard redirect view determines which dashboard (student or professor) is served based on user role.
- **Project Listing and Browsing:**
    - **Actors:** Students (primarily) and Faculty
    - **Process:** A project list view displays available projects; detailed views offer more information.
- **Project Management:**
    - **Actors:** Professors
    - **Process:** Professors add, edit, and delete projects using dedicated CRUD views.
- **Application Submission and Processing:**
    - **Actors:** Students
    - **Process:** Students apply to projects using a dedicated application form; professors review and update the status of applications.
- **Profile Management:**
    - **Actors:** All users
    - **Process:** Users view and edit their profiles, with separate forms tailored for student and professor details.
- **Contact and Communication:**
    - **Actors:** All users
    - **Process:** A contact view allows users to reach out for support or inquiries.

---

3. Design Overview

3.1 Design Goals and Constraints

- **Goals:**
    - Provide a user-friendly, reliable web portal for project matching.
    - Ensure seamless user interaction for both faculty and students.
    - Achieve responsive performance for up to 100 concurrent users.
- **Constraints:**
    - Implementation on a university-managed server.
    - Use of SQLite as the primary data store.
    - Django framework governs backend processing.
    - Basic security measures due to controlled access.

3.2 Design Assumptions

- The system is intended for internal use within Mahindra University.
- User traffic is moderate, with the possibility of scale if required.
- Faculty project listings and student applications drive most interactions.
- Basic scheduling between faculty and students will be implemented.

3.3 Significant Design Packages

The overall design is broken down into these packages:

- **Presentation Layer:**
  - Implements the UI using HTML, CSS, and minimal JavaScript.
  - Pages include the home page, project list, and login interface.
- **Business Logic Layer:**
  - Comprises Django views and custom decorators (e.g., role-based access controls).
  - Manages core workflows such as user registration, project CRUD operations, and application processes.
- **Data Access Layer:**
  - Uses Django ORM with models: Custom User, Profile, Project, and Application.
- **Security Module:**
  - Built on Django authentication mechanisms enhanced with custom user management.
- **Admin Interface:**
  - Utilizes Django admin with customizations (UserAdmin, ProjectAdmin, ApplicationAdmin) for administrative tasks.

3.4 Dependent External Interfaces

The design requires external interfaces to integrate with:

External Module/Application	Expected Module/Interface	Description
University Authentication Service	University Single Sign-On (SSO)	Validates user identities (if available)
Email Service Provider	SMTP Interface	Notifies users about application status updates
Hosting Environment	University Server Infrastructure	Provides deployment, backup, and maintenance support

3.5 Implemented Application External Interfaces (and SOA web services)

The following RESTful endpoints are made available:

Interface Name	Implementing Module	Functionality Description
User Authentication	Security Module	Manages login, signup, session management, and role-based access
Project API	Project Management Module	CRUD operations for project management
Application API	Application Management Module	Handles submission and status update of project applications
Profile API	User/Profile Management Module	Manages view and update actions for user profiles
Contact API	Communication Module	Supports feedback and support communication

---

4. Logical View

4.1 Design Model

The design model decomposes the system into the following significant layers:

- **Presentation Layer:**

  - **Components:**
    - *HomeView, Dashboard Views, Project List and Detail Views, Login and Signup Pages, Profile Pages*
  - **Responsibilities:**
    - Render dynamic content using Django templates.
    - Capture user input and route to appropriate views.
- **Business Logic Layer:**
  - **Components:**
    - Django view functions such as `home_view`, `signup_view`, `student_dashboard_view`, `professor_dashboard_view`, `project_list_view`, `apply_project_view`, and others.
  - **Responsibilities:**
    - Process incoming requests, enforce business rules, handle role differentiation, and trigger CRUD operations.
- **Data Access Layer:**
  - **Components:**
    - Django Models:
      - `User` (custom user with roles),
      - `Profile` (user details),
      - `Project` (faculty-listed projects),
      - `Application` (applications by students)
  - **Responsibilities:**
    - Map between Python objects and database tables using Django ORM.
- **Security Module:**
  - **Components:**
    - Custom user manager and authentication forms.
    - Decorators for role-based access control (e.g., `student_required`, `professor_required`).
  - **Responsibilities:**
    - Manage user credentials, permissions, and session life cycle.
- **Admin Interface:**
  - **Components:**
    - Django Admin classes (`UserAdmin`, `ProjectAdmin`, `ApplicationAdmin`).
  - **Responsibilities:**
    - Provide administrators with tools to manage system data.

4.2 Use Case Realization

Each use case from Section 2 is realized as follows:

- **User Registration & Login:**
  1. **Signup:** Users register via `signup_view` using `CustomUserCreationForm` with department choices.
  2. **Login:** Django's `LoginView` with a custom authentication form validates credentials.
  3. **Dashboard Redirect:** A controller (`dashboard_redirect_view`) routes users based on role (student or professor).
- **Project Interaction:**
  1. **Browse Projects:** `project_list_view` fetches and displays open projects.
  2. **Project Detail:** Detailed information about a project is shown via `project_detail_view`.
  3. **Application Submission:** Students use `apply_project_view` with `ApplicationForm` to apply for a project.
  4. **Project Management:** Professors manage projects using `add_project_view`, `edit_project_view`, and deletion views.

- **Application Processing:**
    1. **View Applications:** Professors review applications through `view_project_applications_view`.
    2. **Update Status:** Professors approve or reject applications using `update_application_status_view`.
- **Profile Management:**
    1. **View/Edit Profile:** Users update their details using appropriate forms (`UserProfileForm`, `StudentProfileForm`, `ProfessorProfileForm`).
- **Contact and Support:**
    1. **Contact View:** Users access the `contact_view` to send inquiries or feedback.

---

5. Data View

5.1 Domain Model

The persistent domain model comprises the following entities and relationships:

- **User:**
    - *Attributes:* email, name, usertype (student/professor), is_staff, is_active, date_joined
    - *Relationships:* One-to-One with Profile; can be associated with multiple Projects (as faculty) or Applications (as student).
- **Profile:**
    - *Attributes:* academic details (for students), faculty information (for professors)
    - *Relationship:* One-to-One with User.
- **Project:**
    - *Attributes:* title, description, status, created_at, faculty reference
    - *Relationship:* Many-to-One relationship with a professor user.
- **Application:**
    - *Attributes:* submission date, application status (Pending, Approved, Rejected)
    - *Relationships:* Associated with a student and a project.

5.2 Data Model (Persistent Data View)

The system uses SQLite as the backend, and the models map as follows:

- **Users Table:**
    - *Fields:* userID (PK), email, name, usertype, is_staff, is_active, date_joined.
- **Profiles Table:**
    - *Fields:* profileID (PK), user (FK), academicdetails, facultyinformation.
- **Projects Table:**
    - *Fields:* projectID (PK), title, description, status, created_at, faculty (FK).
- **Applications Table:**
    - *Fields:* applicationID (PK), student (FK), project (FK), submissionDate, applicationStatus.

5.2.1 Data Dictionary

- **User:**
    - email: Primary user identifier, unique.
    - name: Full name for display purposes.

    - usertype: Either "student" or "professor".
- **Profile:**
    - academicdetails: Additional info for students.
    - facultyinformation: Details relevant to faculty users.
- **Project:**
    - title: Short project title.
    - description: Detailed project explanation.
    - status: Indicator of project availability.
- **Application:**
    - submissionDate: Timestamp of application submission.
    - applicationStatus: Current state (Pending, Approved, Rejected).

---

6. Exception Handling

- **User Input Validation:**
    - Forms (e.g., `CustomUserCreationForm`, `ApplicationForm`) enforce rules such as password length and proper email format.
- **Authentication & Authorization:**
    - Role-specific decorators and error messages prevent unauthorized access.
- **Database Errors:**
    - Django's ORM exception handling mechanisms log and manage database transaction failures.
- **Business Logic Exceptions:**
    - Attempts to apply for closed projects or submit incomplete forms trigger user-friendly error messages and logging.

---

7. Configurable Parameters

The application employs several configurable settings that can be adjusted without code changes:

Configuration Parameter Name	**Definition and Usage**	**Dynamic?**
SESSION_TIMEOUT	Maximum time of inactivity before auto logout (in minutes)	Yes
RESULTS_PER_PAGE	Number of projects or applications displayed per page	Yes
DATABASE_PATH	File path for the SQLite database	No
EMAIL_SMTP_SERVER	SMTP server details for sending out notifications	No
LOGGING_LEVEL	Logging verbosity for monitoring application events	Yes

---

8. Quality of Service

8.1 Availability

- **Target:** Minimum uptime of 99%.
- **Mechanism:** Deployment on robust university servers with redundancy, regular backups, and scheduled maintenance practices.

8.2 Security and Authorization

- **Authentication:** Custom Django authentication with hashed passwords and secure login.
- **Authorization:** Role-based access control restricts sensitive functionality.
- **Data Protection:** Sensitive information is kept to a minimum and secured according to university policies.

8.3 Load and Performance Implications

- **Concurrent Users:** Designed to support up to 100 concurrent users.
- **Response Time:** Queries and interactions are optimized to complete within a 7-second window through efficient ORM usage and caching strategies.
- **Scalability:** Although initially designed for internal usage, the structure allows migration to a more robust database system if needed.

8.4 Monitoring and Control

- **Monitoring:**
  - Integrated logging via Django's logging framework.
  - Administrators can view critical system metrics through a dedicated dashboard.
- **Control:**
  - Administrative interfaces (Django Admin) enable real-time monitoring and configuration changes.
  - Scheduled maintenance procedures and automated alerts help maintain system integrity.