**Software Design Document**

**<Project Title>**



**Submitted by**

<Student Name (Roll No)>

**Submitted to**

<Supervisor Name>

**Department of Information Technology**

**Faculty of Computing**

**The Islamia University of Bahawalpur**

**Summery**

Here the author will write summery of the document. They can provide a brief of the background, their design considerations, application architecture and

**Table of Content**

1. **Application Architecture**

The **University Project Repository** application will be developed using the **Django** framework, which follows the **Model-View-Template (MVT)** architectural pattern. This architecture separates the application into three interconnected components: **Models**, **Views**, and **Templates**, allowing for efficient development, maintenance, and scalability. Below is a detailed explanation of the components and their roles in the system:

**MVT Architecture Components**

**Models:**

Models in Django define the database schema and act as the business logic layer of the application. For this project, the following models are proposed:

* User: To manage student and tutor accounts with roles and permissions.
* Project: To store project details, such as title, description, file uploads, linked repositories, submission dates, and plagiarism reports.
* Feedback: To handle tutor reviews, comments, and grades.
* PlagiarismReport: To store results from the Turnitin API integration.

**Views:**  
Views are responsible for handling user requests, retrieving data from models, and rendering templates. Django views will manage the application's workflows, such as uploading projects, grading submissions, and displaying feedback.

**Templates**:  
Templates handle the presentation layer of the application. These files will use Django's template language to dynamically render HTML content for features like project listings, submission forms, feedback views, and grade displays.

**Controllers**

In Django, the **View** component acts as the controller. Below is the list of the main views (controllers) and their respective functions:

1. **HomeView**:
   * index(request): Displays the homepage with a list of recently uploaded projects and top-graded projects.
2. **ProjectView**:
   * submit\_project(request): Handles the project submission process, including file uploads and repository links.
   * view\_project(request, project\_id): Displays detailed information about a specific project.
   * list\_projects(request): Lists all available projects for browsing by students and tutors.
3. **FeedbackView**:
   * add\_feedback(request, project\_id): Allows tutors to provide reviews, grades, and comments for a submitted project.
   * view\_feedback(request, project\_id): Displays feedback given by tutors for a specific project.
4. **TurnitinView**:
   * check\_plagiarism(request, project\_id): Calls the Turnitin API to verify the originality of submitted projects and stores the results in the database.
   * view\_plagiarism\_report(request, project\_id): Displays the plagiarism report for tutors and students.
5. **AuthenticationView**:
   * login(request): Authenticates users (students and tutors) and redirects them to their dashboard.
   * logout(request): Logs out users and clears session data.
   * register(request): Handles user registration and account creation.

External APIs and Libraries

Turnitin API:

The Turnitin API will be used to check the originality of student project submissions. It works by sending project files to Turnitin’s servers, where they are checked against a large database of academic resources and online content. The API will return a similarity score and a detailed report, which will be saved in the PlagiarismReport model.

Django Libraries:

django.contrib.auth: Used for user authentication and authorization.

django.db.models: Used for defining and interacting with database models.

django.forms: Used to handle form submissions for project uploads and feedback.

Bootstrap:

Bootstrap will be used as a front-end framework to ensure the platform is responsive and user-friendly.

PostgreSQL:

PostgreSQL will serve as the relational database, providing robust support for storing user data, project files, and feedback.

UML Diagrams

 **Class Diagram**: Illustrating the relationships between models such as **User**, **Project**, **Feedback**, and **PlagiarismReport**.

 **Sequence Diagram**: Explaining key workflows like project submission, grading, and plagiarism checks.

 **Use Case Diagram**: Outlining the interactions between users (students, tutors, admins) and the system.

**Coding Conventions**

The following coding conventions will be adhered to throughout the development process:

* **PEP 8**: The Python style guide will be followed for writing clean and maintainable code.
* **Function Naming**: Function names will be in snake\_case and describe their purpose clearly (e.g., submit\_project).
* **Commenting**: Inline and block comments will be used to explain code functionality.
* **Version Control**: Git will be used to manage the codebase with clear commit messages and branch naming conventions (e.g., feature/project\_submission).

Authors will explain the architecture of overall application. If they are following a pattern like MVC or MVVM then they will also explain the components of architecture (list of proposed Models, Views and Controllers). Further they explain the controllers and also provide the list of function of controllers but they will not explain Models and Views. If some APIs are developing then they will also explain these and if they are calling some external APIs or libraries then they also explain the working of these APIs and libraries.

They can provide class diagrams, sequence diagrams and other UML diagrams to explain the proposed working of their system. Furthermore, they will also provide the coding conventions they use to write the application.

1. **Data Model Schema**

Detail of database conceptual schema will be provided here. Each data model entity will be explained in details, its working properties and relationship with other entities. Authors will also provide ER diagram to explain the working entities.

1. **User Interface**

Overall design philosophy such as color scheme, font, design niche and screen constraints (if required then responsiveness or targeted screens size) will be discussed here. List of screens/pages will be provided and explain the data on each page/screen. List of controls (Textbox, Drop Down, Buttons etc.) on each page will be provided and explain the working of data. Also explain the input constraints.