# # Development Plan: Course Management System

# ## Table of Contents

# 1. [Introduction](#introduction)

# 2. [Objectives](#objectives)

# 3. [System Architecture](#system-architecture)

# 4. [Features and Functionalities](#features-and-functionalities)

# 5. [Technology Stack](#technology-stack)

# 6. [Development Phases](#development-phases)

# 7. [Timeline](#timeline)

# 8. [Testing Plan](#testing-plan)

# 9. [Risks and Mitigation](#risks-and-mitigation)

# 10. [Conclusion](#conclusion)

# ---

# ## Introduction

# The \*\*Course Management System\*\* is designed to streamline course, student, and faculty management, automate grade recording, and provide a robust reporting system. This plan outlines the development strategy, technologies, and schedule.

# ---

# ## Objectives

# - Simplify course, student, and faculty management processes.

# - Provide secure role-based authentication for users.

# - Automate grade recording and generate reports efficiently.

# ---

# ## System Architecture

# The system will follow a \*\*3-tier architecture\*\*:

# 1. \*\*Frontend\*\*: User interface built with React.js.

# 2. \*\*Backend\*\*: Business logic using Django/Flask.

# 3. \*\*Database\*\*: PostgreSQL for data storage.

# ---

# ## Features and Functionalities

# - \*\*Course Management\*\*: Add, edit, and delete courses.

# - \*\*Student Management\*\*: Enroll students, manage records, and record grades.

# - \*\*Faculty Management\*\*: Assign courses and manage workloads.

# - \*\*Authentication System\*\*: Secure login with role-based access.

# - \*\*Grade Recording\*\*: Record grades and compute averages.

# - \*\*Reporting Tools\*\*: Generate class lists, grades, and workload reports.

# ---

# ## Technology Stack

# - \*\*Frontend\*\*: React.js

# - \*\*Backend\*\*: Django or Flask

# - \*\*Database\*\*: PostgreSQL or MySQL

# - \*\*Version Control\*\*: Git and GitHub

# ---

# ## Development Phases

# | Phase | Duration | Tasks |

# |------------------------|----------|-------------------------------------------|

# | Planning and Setup | 1 week | Initialize repo, setup tech stack |

# | Core Features | 4 weeks | Develop Course, Student, Faculty modules |

# | Authentication & Grades| 2 weeks | Implement authentication and grade system |

# | Reporting System | 2 weeks | Create dynamic reporting tools |

# | Testing and Deployment | 2 weeks | Perform testing and deploy system |

# ---

# ## Timeline

# - \*\*Week 1\*\*: Initial setup and planning.

# - \*\*Weeks 2-5\*\*: Development of core features.

# - \*\*Weeks 6-7\*\*: Add authentication and grading system.

# - \*\*Weeks 8-9\*\*: Build reporting tools.

# - \*\*Week 10\*\*: Final testing and deployment.

# ---

# ## Testing Plan

# - \*\*Unit Testing\*\*: Verify individual components.

# - \*\*Integration Testing\*\*: Ensure modules work together.

# - \*\*System Testing\*\*: Test the entire system end-to-end.

# ---

# ## Risks and Mitigation

# - \*\*Delays in development\*\*: Use Agile methodology for incremental progress.

# - \*\*Data loss\*\*: Regular backups and version control.

# ---

# ## Conclusion

# This plan ensures a structured, step-by-step development of the Course Management System, delivering a secure, efficient, and user-friendly solution.Technical Documentation

## 1. Overview

This document provides a detailed technical explanation of the Course Management System, including system architecture, components, API endpoints, database schema, and deployment instructions. It serves as a reference for developers, system administrators, and technical stakeholders.

## 2. System Architecture

The Course Management System follows the Model-View-Controller (MVC) architecture for maintainability, scalability, and separation of concerns. The system is built using the following components:

* - Frontend: React.js provides a dynamic and interactive user interface for end-users.
* - Backend: Django (Python) handles the business logic, user authentication, and data processing.
* - Database: PostgreSQL is used for structured data storage and efficient querying.
* - API: RESTful APIs are implemented to facilitate communication between the frontend and backend.

## 3. API Endpoints

The system exposes the following API endpoints for interaction between the frontend and backend:

|  |  |  |
| --- | --- | --- |
| Endpoint | Method | Description |
| /api/auth/login | POST | Authenticate users and generate a token. |
| /api/auth/register | POST | Register new users with role-based access. |
| /api/courses | GET | Retrieve a list of all available courses. |
| /api/courses | POST | Add a new course (admin-only). |
| /api/students | GET | Retrieve a list of enrolled students. |
| /api/grades | POST | Submit grades for a specific student and course. |

## 4. Database Schema

The database schema consists of the following tables to handle core functionality:

|  |  |
| --- | --- |
| Table | Description |
| users | Stores user information including roles (admin, student, faculty). |
| courses | Stores course details such as name, description, and credits. |
| enrollments | Links students to courses and stores enrollment details. |
| grades | Stores grades for each student and course. |
| faculty\_assignments | Tracks which faculty members are assigned to which courses. |

## 5. Deployment Instructions

Follow these steps to deploy the system:

1. Clone the repository from GitHub.

2. Install required dependencies using `pip install -r requirements.txt` for the backend and `npm install` for the frontend.

3. Configure the PostgreSQL database and update the settings in the backend configuration file.

4. Run database migrations using `python manage.py migrate`.

5. Start the backend server using `python manage.py runserver`.

6. Start the frontend server using `npm start`.

7. Access the system via the configured URL.