

A Major Project Report on

IMPLEMENTATION OF RESULT MANAGEMENT SYSTEM

Submitted in partial fulfillment of the requirements

For the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

SUBMITTED BY

D BHASKAR REDDY

20JK1A0508

M KIRAN KUMAR

20JK1A0522

P SRIKANTH

20JK1A0529

Under The Esteemed Guidance of

Baditha.Anusha



Department of Computer Science Engineering

GUNTUR ENGINEERING COLLEGE

(Approved by AICTE, NEW DELHI, Affiliated to JNTU KAKINADA)

NH-5, Yanamadala, Guntur, Andhra Pradesh - 522019

2020-2024

GUNTUR ENGINEERING COLLEGE

YANAMADALA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that D BHASKAR REDDY, M KIRAN, P SRIKANTH has completed the Internship in **Codegnan IT Solution Pvt. Ltd** on **PYTHON FULL STACK Internship** under my supervision as a part of partial fulfilment of the requirement for the Degree of **B.Tech.** in the Department of **COMPUTER SCIENCE AND ENGINEERING, GUNTUR ENGINEERING COLLEGE** YANAMADALA, during the academic year 2022-23.

ANUSHA BADITHA

Internship Guide

Dr . R. BULLIBABU., M. Tech.

Ph.D.

Professor & Head

**Department of Computer Science and
Engg.**

Dr . RAMA KOTAIAH

Principal

External Examiner

IMPLEMENTATION OF RESULT MANAGEMENT SYSTEM USING PYHTON

ACKNOWLEDGEMENTS

I would like to express my special thanks to my teacher **ESWAR SIR** who gave me the golden opportunity to do this wonderful project of “**IMPLEMENTATION OF RESULT MANAGEMENT SYSTEM**”.

I would also like to extend my gratitude to the HOD sir **K. BULLI BABU** for providing me with all the facilities that were required.

D BHASKAR REDDY 20JK1A0508

M KIRAN KUMAR 20JK1A0522

P SRIKANTH 20JK1A0529

ABSTRACT

An Online student Result Management System in an EDUCATIONAL INSTITUTIONS.

- It serves the students for Result management system. And this can be use full to the student and also the college management .

To implement a result management system using Python, you can follow these general steps:

Define the data structure: Determine how you want to store and organize the results. For example, you might use a dictionary or a list of dictionaries to store information about students and their respective results.

Implement CRUD operations: Create, Read, Update, and Delete operations are fundamental for managing results. Implement functions or methods for each of these operations.

Create: Allow users to input new results and store them in the appropriate data structure.

Read: Provide functionality to display results, either for individual students or for the entire class.

Update: Allow users to modify existing results based on student ID or any other identifier.

Delete: Provide a mechanism to remove results from the system.

Implement validation: Validate user input to ensure it meets the required format and constraints. For example, you might check if the entered marks are within the valid range.

Add additional features: Enhance your system with additional functionality as needed. Some possible features could include calculating average marks, generating reports, or sorting results based on specific criteria.

TABLE OF CONTENTS

S.NO CHAPTERS

1. Introduction
 - 1.1 project aims and objectives
 - 1.2 background of project
2. Existing system and Demerits
3. Proposed system
 - 3.1 advantages of proposed system
 - 3.2 available features
4. Hardware and software requirements
5. Data Dictionary
6. System Requirement Specification
 - 6.1 SRS Document
 - 6.2 Functional requirements
 - 6.3 Non functional requirements
7. System Design
 - 7.1 Introduction to design
 - 7.2 Unified Modeling Language Diagrams
8. UML diagrams
9. Coding and implementation
10. Output Screens

11. Technologies Used

12. Software Testing

12.1 Types of testing

12.2 Test cases

Conclusion

Future scope

Bibliography

CHAPTER- 1

INTRODUCTION

A result management system is a software application designed to efficiently manage and process results for educational institutions or organizations. It allows administrators and teachers to input, store, analyze, and generate reports on student performance. Python, being a versatile and widely-used programming language, provides a robust platform for building such a system.

In this introduction, we'll outline the key components and steps involved in implementing a result management system using Python.

Design and Planning:

Before diving into coding, it's essential to plan and design the system. Identify the requirements and functionalities needed, such as student registration, result entry, report generation, and data analysis. Define the data structures and relationships, user interfaces, and any additional features required.

Database Design:

A result management system typically relies on a database to store and retrieve student information and results. Choose a suitable database management system (e.g., SQLite, MySQL, PostgreSQL) and design the database schema. Define tables for students, courses, results, and any other relevant entities. Establish relationships and define constraints to ensure data integrity.

Set up the Development Environment:

Install Python and any necessary frameworks or libraries for building the result management system. Popular choices include Flask or Django for web development, and SQLAlchemy or Django ORM for database interactions. Ensure you have a text editor or an integrated development environment (IDE) to write your code.

User Interface Development:

Create a user-friendly interface to interact with the system. This could be a web-based application, a desktop GUI, or a command-line interface (CLI), depending on your requirements. Use HTML, CSS, and JavaScript for web interfaces, or libraries like Tkinter for desktop applications.

Implement Core Functionality:

Begin implementing the core features of the result management system. This includes student registration, result entry, result calculation, and result retrieval. Write functions or classes to handle these operations, ensuring they interact correctly with the database.

Data Analysis and Reporting:

Implement functions or modules to analyze student performance data and generate reports. You may use data visualization libraries such as Matplotlib or Plotly to create charts and graphs representing students' results over time. Consider including features like filtering results by class, subject, or date range to provide flexible reporting options.

Security and Access Control:

Implement security measures to ensure data privacy and access control. Implement user authentication and authorization mechanisms to restrict access to sensitive information. Use encryption techniques to protect sensitive data, such as student personal details or exam results.

Testing and Debugging:

Thoroughly test your result management system to identify and fix any issues or bugs. Perform unit testing on individual components and integration testing to ensure the system works as a whole. Solicit feedback from users to gather insights and make necessary improvements.

Deployment and Maintenance:

Once the result management system is stable and tested, deploy it to a production environment. Choose a suitable hosting platform or server to make the system accessible to the intended users. Regularly maintain and update the system to address any security vulnerabilities or add new features as required.

Remember to document your code and provide user manuals or documentation for system administrators and users to refer to

1.1 PROJECT AIMS AND OBJECTIVES

The aim of implementing the Result Management System using Python is to create a comprehensive and efficient system that automates and streamlines the management of student results in educational institutions.

The system should provide the following objectives:

Automation of Result Entry: Develop a user-friendly interface to allow teachers or administrators to easily enter and update student results for various subjects and exams.

Storage and Retrieval of Results: Implement a database system to securely store and organize student result data, including subject grades, marks, and overall performance.

Result Calculation and Analysis: Develop algorithms and functions to automatically calculate students' overall performance based on their individual subject grades. Implement analysis features to generate reports on student performance, such as class-wise or subject-wise comparisons.

Reporting and Visualization: Create a reporting module that allows for the generation of customized reports and data visualization, including graphs and charts, to provide a clear representation of student performance trends.

Student and Course Management: Develop functionalities for student registration and course management, allowing administrators to add or remove students and assign courses or subjects to them.

Security and Access Control: Implement robust security measures to protect sensitive student information and ensure appropriate access control. This includes user authentication, authorization, and encryption of sensitive data.

User-Friendly Interface: Design an intuitive and user-friendly interface that simplifies the process of data entry, result retrieval, and report generation. Consider usability principles to ensure ease of use for both administrators and teachers.

Scalability and Flexibility: Build a system that can handle a large volume of student data and accommodate future growth. Ensure that the system is scalable and flexible to adapt to changes in educational requirements or institutional policies.

Error Handling and Validation: Implement error handling mechanisms and data validation checks to minimize errors and ensure data integrity. Provide meaningful error messages to assist users in troubleshooting issues.

Documentation and Maintenance: Create comprehensive documentation, including system manuals and guides, to assist system administrators and users in understanding and utilizing the system. Plan for regular maintenance and updates to address any issues or enhancements in the future.

By achieving these aims and objectives, the Result Management System implemented using Python will streamline the process of managing student results, improve efficiency, and provide valuable insights into student performance for educational institution

1.2 Background of project

In educational institutions, managing and processing student results is a critical task that requires efficiency and accuracy. Traditional manual methods of result management can be time-consuming, error-prone, and lack the ability to provide insightful analysis. To address these challenges, implementing a Result Management System using Python can significantly streamline the process and improve overall efficiency.

The project aims to leverage the power of Python, a versatile and widely-used programming language, to develop a robust and user-friendly system for managing student results. Python offers a range of libraries, frameworks, and tools that make it well-suited for building such a system. Its simplicity, readability, and extensive community support contribute to its popularity in various application domains.

By implementing a Result Management System using Python, educational institutions can automate and simplify tasks related to result entry, storage, analysis, and reporting. The system will allow administrators and teachers to efficiently manage student information, record exam scores, calculate overall performance, and generate comprehensive reports. It will provide valuable insights into student progress, enabling informed decision-making and personalized interventions to support student success.

The use of Python enables the project to leverage its rich ecosystem of libraries and frameworks. For instance, web development frameworks like Flask or Django can be utilized to build a web-based interface for easy access and management of student results. Data analysis libraries such as Pandas, NumPy, or Matplotlib can be employed to perform statistical analysis and generate visualizations to aid in understanding student performance trends.

The implementation of the Result Management System using Python will contribute to the digital transformation of educational institutions. It will reduce manual effort, minimize errors, and improve the overall efficiency of result management processes. Additionally, the system will provide a centralized platform for accessing and analyzing student results, promoting data-driven decision-making and fostering collaboration among teachers, administrators, and other stakeholders.

Throughout the project implementation, considerations such as security, scalability, and user-friendliness will be prioritized. Security measures will be implemented to protect student data, access control mechanisms will ensure data privacy, and error handling and validation will be incorporated to maintain data integrity. The system will be designed to accommodate future growth and adapt to changing educational requirements, making it a sustainable solution for result management in the long term.

In summary, the implementation of a Result Management System using Python aims to revolutionize the way student results are managed and analyzed in educational institutions. By leveraging the strengths of Python and its ecosystem, the project seeks to deliver an efficient, scalable, and user-friendly solution that enhances decision-making, promotes data-driven education, and supports student success.

CHAPTER 2

2. Existing system and Demerits

Before implementing a new Result Management System using Python, it is important to evaluate the shortcomings of the existing system. The existing system may vary across educational institutions, but common issues and demerits include:

Manual Data Entry: Many institutions still rely on manual data entry processes, where teachers or administrators input student results and information into physical registers or spreadsheets. This method is time-consuming, error-prone, and lacks efficiency.

Lack of Centralized Data: In the absence of a centralized system, student result data may be scattered across multiple sources, such as paper-based records, spreadsheets, or local databases. This makes it challenging to access, manage, and analyze data effectively.

Limited Accessibility: Physical records or localized databases restrict access to student results. Teachers or administrators may face difficulties in retrieving and sharing result information, especially when they need to access it remotely or collaborate with other stakeholders.

Data Inconsistencies and Errors: Manual data entry increases the chances of human errors, such as typos, calculation mistakes, or misplaced entries. Inconsistent data formats or lack of validation checks further contribute to data inaccuracies, affecting the reliability of the results.

Time-Consuming Result Calculation: Without an automated system, calculating overall performance or aggregating results for analysis can be time-consuming. Manual computations may lead to delays and hinder prompt decision-making or interventions for students.

Limited Result Analysis and Reporting: Existing systems often lack robust analysis and reporting capabilities. Generating meaningful reports or visualizations to identify performance trends, compare results, or provide actionable insights becomes challenging or cumbersome.

Lack of Security Measures: Physical records or spreadsheets offer minimal security for student data. Unauthorized access or loss of records can lead to privacy breaches, compromising the confidentiality of student information.

Difficulty in Scaling and Adaptability: Traditional systems may struggle to accommodate the growing volume of student data or adapt to changes in educational policies or reporting requirements. They may lack the flexibility to incorporate new features or functionalities.

These demerits highlight the limitations and challenges faced by educational institutions using existing addressed, leading to increased efficiency, accuracy, accessibility, and data-driven decision-making in managing student

CHAPTER 3

PROPOSED SYSTEM

The proposed Result Management System aims to address the limitations of the existing system by leveraging the capabilities of Python and modern technologies. The system will provide an automated, centralized, and user-friendly platform for managing and analyzing student results. Key features of the proposed system include:

Automated Result Entry: The system will offer a user-friendly interface for teachers or administrators to enter and update student results electronically. This eliminates the need for manual data entry, reducing errors and saving time.

Centralized Data Storage: Student result data will be stored in a centralized database, ensuring easy access, organization, and management of information. This eliminates the reliance on scattered paper-based records or localized databases.

Web-Based Accessibility: The system will be accessible through a web-based interface, enabling authorized users to access and manage student results from anywhere with an internet connection. This promotes remote access, collaboration, and real-time updates.

Result Calculation and Analysis: The system will include algorithms and functions to automatically calculate students' overall performance based on their subject grades. It will provide analysis features such as performance trends, subject-wise comparisons, and statistical insights to support data-driven decision-making.

Customizable Reporting: The system will allow users to generate customized reports, including graphical visualizations, based on specific criteria such as class, subject, or exam. This enables comprehensive reporting and meaningful representation of student performance data.

Enhanced Security Measures: The proposed system will incorporate robust security measures to protect student data. User authentication and access control mechanisms will ensure data privacy, while encryption techniques will safeguard sensitive information.

Scalability and Adaptability: The system will be designed to handle a large volume of student data and accommodate future growth. It will have the flexibility to incorporate new features, adapt to changing educational requirements, and integrate with other systems or technologies.

Efficient Communication and Collaboration: The proposed system will facilitate effective communication and collaboration among teachers, administrators, and stakeholders. It will provide features for sharing information, discussing student performance, and making collaborative decisions within the system.

Historical Data Management: The system will maintain an organized archive of past results, allowing for historical analysis, audits, and comparison of student performance across multiple academic sessions.

User-Friendly Interface: The proposed system will prioritize a user-friendly interface, ensuring ease of use for teachers, administrators, and other system users. It will offer intuitive navigation, clear data input forms, and interactive visualizations for a seamless user experience.

By implementing the proposed Result Management System, educational institutions can benefit from improved efficiency, accuracy, accessibility, and data-driven decision-making. The system will streamline result management processes, provide valuable insights into student performance, and support personalized interventions to enhance student success.

CHAPTER 4

HARDWARE REQUIREMENTS

- Processor: Pentium ○ RAM: 4GB
- Hard Disk: 1TB
- Speed: 1.1GHz

SOFTWARE REQUIREMENTS

- Operating System: Windows
- Scripting Language: JSP
- Back-End: MYSQL
- Front-End: HTML5 and CSS3 ○ Supporting Tools: NetBeans IDE, JQUERY ○ Type: Web Application.
- Server: TOMCAT 8.0(cross platform, Apache, MYSQL, JSP)
- Java Version : J2SDK1.5

CHAPTER 5

Data Dictionary

A data dictionary provides a detailed description of the data elements used in a system or database. Here's an example of a data dictionary for the implementation of a Result Management System:

Table: Students

Student id:	Varchar(10)	Primary key
Frist name:	Varchar(10)	Primary key
Last name:	Varchar(10)	Primary key
DOB:	Int	
Gender:	Varchar(10)	Primary key
Grade level:	Int	

Table: Subjects

Student id:	Varchar(10)	Primary key
Subject name:	Varchar(10)	Primary key

Table: Results

Result id:	Int	Primary key
Student id:	Int	Foreign key
Subject id:	Int	Foreign key
Exam id:	Int	Foreign key
Marks obtained:	Int	Foreign key

Table: Grades

Grade_id:	Foreign key
Grade_name:	Foreign key
Grade_range:	Foreign key

Table: Performance

Performace id:	Varchar(10)
Student id:	Varchar (20)
Overall marks	Varchar (20)
Exam_id:	Int
Grade id:	Varchar(10)

The above data dictionary provides an overview of the tables and their respective fields that would be required for implementing a Result Management System. It includes tables for storing student information, subjects, exams, results, grades, and overall performance records. The primary keys and foreign keys establish the relationships between the tables for data integrity and retrieval.

Note that this is a basic example, and depending on the specific requirements of your Result Management System, you may need to expand or modify the data dictionary to include additional fields or tables to accommodate other relevant information, such as teacher details, class information, or attendance records.

CHAPTER 6

6.1 SRS DOCUMENT

The SRS typically contains the brief description of the project. The purpose of the requirement document is to specify all the information required to design, develop and test the software. The purpose of this project is to provide a friendly environment to maintain all the details of books.

6.2 FUNCTIONAL REQUIREMENTS

Functional requirements describe the specific functionalities or features that the Result Management System should possess. These requirements outline what the system should do to meet the needs of its users. Here are some example functional requirements for a Result Management System:

User Registration and Login

- 1.1 Users should be able to register an account with the system.
- 1.2 Registered users should be able to log in using their credentials.
- 1.3 The system should validate user credentials and provide appropriate access rights based on user roles (e.g., administrator, teacher, student).

Student Management

- 2.1 Administrators should be able to add, edit, and delete student information (e.g., name, date of birth, gender, grade level).
- 2.2 Teachers should be able to view and update student information for their assigned classes.
- 2.3 The system should generate unique student IDs and assign them to new students automatically.

Subject Management

- 3.1 Administrators should be able to add, edit, and delete subjects.
- 3.2 Teachers should be able to assign subjects to their classes.
- 3.3 The system should maintain a list of subjects offered and their corresponding subject codes.

Exam Management

- 4.1 Administrators should be able to create, schedule, and manage exams.
- 4.2 Teachers should be able to enter and update exam details (e.g., name, date, duration) for their assigned classes.
- 4.3 The system should generate unique exam IDs and assign them to new exams automatically.

Result Entry and Calculation

- 5.1 Teachers should be able to enter student results for each exam and subject.
- 5.2 The system should calculate the overall marks obtained by each student in each exam and subject.
- 5.3 The system should calculate the overall grade for each student based on predefined grade criteria.

Result Analysis and Reporting

- 6.1 Teachers and administrators should be able to generate result reports for individual students, classes, or entire grade levels.
- 6.2 The system should provide statistical analysis of student performance (e.g., average marks, standard deviation).
- 6.3 The system should allow for comparisons of results across exams, subjects, or academic sessions.

Communication and Notifications

- 7.1 Teachers should be able to communicate result-related information to students and parents through the system (e.g., messaging, email notifications).
- 7.2 The system should send notifications to relevant stakeholders (e.g., teachers, administrators) for important result-related events or deadlines.

Data Backup and Restore

- 8.1 The system should provide a backup mechanism to regularly backup student data and system configurations.
- 8.2 Administrators should be able to restore the system to a previous backup if needed.

These functional requirements provide an overview of the desired functionalities for a Result Management System. It's important to further refine and prioritize these requirements based on the specific needs and priorities of your educational institution

6.3 Non Functional Requirements

Non-functional requirements specify the qualities or attributes that the Result Management System should possess. These requirements focus on aspects such as performance, usability, security, and maintainability. Here are some examples of non-functional requirements for a Result Management System:

Usability:

- 1.1 The system should have an intuitive and user-friendly interface for easy navigation and interaction.
- 1.2 The system should provide clear and concise error messages to assist users in resolving issues.
- 1.3 The system should support multiple languages to accommodate users from diverse linguistic backgrounds.

Performance:

- 2.1 The system should respond to user actions within an acceptable response time (e.g., less than 2 seconds).

2.2 The system should handle a large number of concurrent users without significant degradation in performance.

2.3 The system should efficiently process and calculate result data to ensure timely generation of reports.

Security:

3.1 User authentication should be implemented to ensure that only authorized individuals can access the system.

3.2 The system should employ encryption techniques to secure sensitive data, such as passwords and student information.

3.3 Access to specific functionalities and data within the system should be controlled based on user roles and permissions.

3.4 The system should maintain a log of user activities and provide audit trails for accountability and security monitoring.

Scalability:

4.1 The system should be designed to accommodate a growing number of students, exams, and result data without significant performance degradation.

4.2 The system should support the addition of new features or modules as the needs of the educational institution evolve.

Maintainability:

5.1 The system should be modular and well-documented to facilitate future enhancements, bug fixes, and system updates.

5.2 The system should follow coding best practices and design patterns to enhance maintainability and code reusability.

5.3 The system should have a clear separation of concerns and adhere to standard software engineering principles.

Compatibility:

6.1 The system should be compatible with commonly used web browsers, operating systems, and devices.

6.2 The system should be designed to integrate with other existing educational systems, such as student information systems or learning management systems, if required.

Documentation:

7.1 The system should have comprehensive documentation, including user manuals, installation guides, and system administration guides.

7.2 The documentation should provide clear instructions on system configuration, maintenance, and troubleshooting.

These non-functional requirements help ensure that the Result Management System meets the necessary quality attributes, usability standards, and operational requirements. It is essential to consider the specific needs and priorities of the educational institution while defining and prioritizing these requirements.

CHAPTER 7

SYSTEM DESIGN

System design involves the process of creating an architectural plan and design for the Result Management System. It includes defining the system's structure, components, modules, and their interactions. Here's an overview of the key aspects of system design for a Result Management System:

System Architecture:

Define the overall structure and high-level components of the system, such as the user interface, application logic, and database.

Choose an appropriate architectural style, such as a layered architecture, client-server architecture, or a microservices architecture, based on the requirements and scalability needs.

Determine the communication protocols and interfaces between system components.

User Interface Design:

Design the user interface (UI) to provide an intuitive and user-friendly experience.

Create wireframes or prototypes to visualize the layout, navigation, and interaction flow of the UI.

Consider accessibility guidelines and best practices for designing inclusive interfaces. Incorporate branding elements and visual design principles to create a cohesive and appealing UI.

Database Design:

Define the database schema and tables based on the data requirements identified in the data dictionary.

Determine relationships between tables (e.g., one-to-many, many-to-many) and establish appropriate primary and foreign key constraints.

Optimize the database design for efficient data retrieval, storage, and integrity.

Consider data normalization techniques to eliminate data redundancy and improve data consistency.

System Components and Modules:

Identify and define the functional modules or components of the system based on the identified requirements.

Determine the responsibilities and interactions of each component.

Define the APIs or interfaces between components to facilitate communication and data exchange.

Data Flow and Processing:

Define the flow of data and processing within the system.

Identify key data inputs, transformations, and outputs for each module or component.

Determine the sequence and dependencies of data processing operations.

Error Handling and Exception Handling:

Identify potential error scenarios and exceptions that may occur during system operation.

Define strategies for handling and reporting errors, such as displaying error messages to users or logging errors for system administrators.

Implement appropriate error handling mechanisms, such as try-catch blocks, to gracefully handle exceptions and prevent system crashes.

Security Design:

Identify potential security risks and vulnerabilities in the system.

Implement security measures, such as user authentication, authorization, and data encryption, to protect sensitive data and ensure system integrity.

Apply security best practices, such as input validation and protection against common security threats (e.g., SQL injection, cross-site scripting).

Integration and External Systems:

Identify any external systems or APIs that need to be integrated with the Result Management System.

Define the integration mechanisms, such as API endpoints or data exchange formats.

Implement necessary data transformations or mappings to ensure seamless integration between systems.

Performance and Scalability:

Consider performance requirements and design the system to handle expected

workloads efficiently.

Optimize database queries, implement caching mechanisms, or use indexing to improve data retrieval performance.

Design the system to scale horizontally or vertically to accommodate increasing user loads or data volumes, if required.

Documentation:

Create technical documentation that describes the system design, including architectural diagrams, component descriptions, and data flow diagrams.

Document any assumptions, constraints, or design decisions made during the design process.

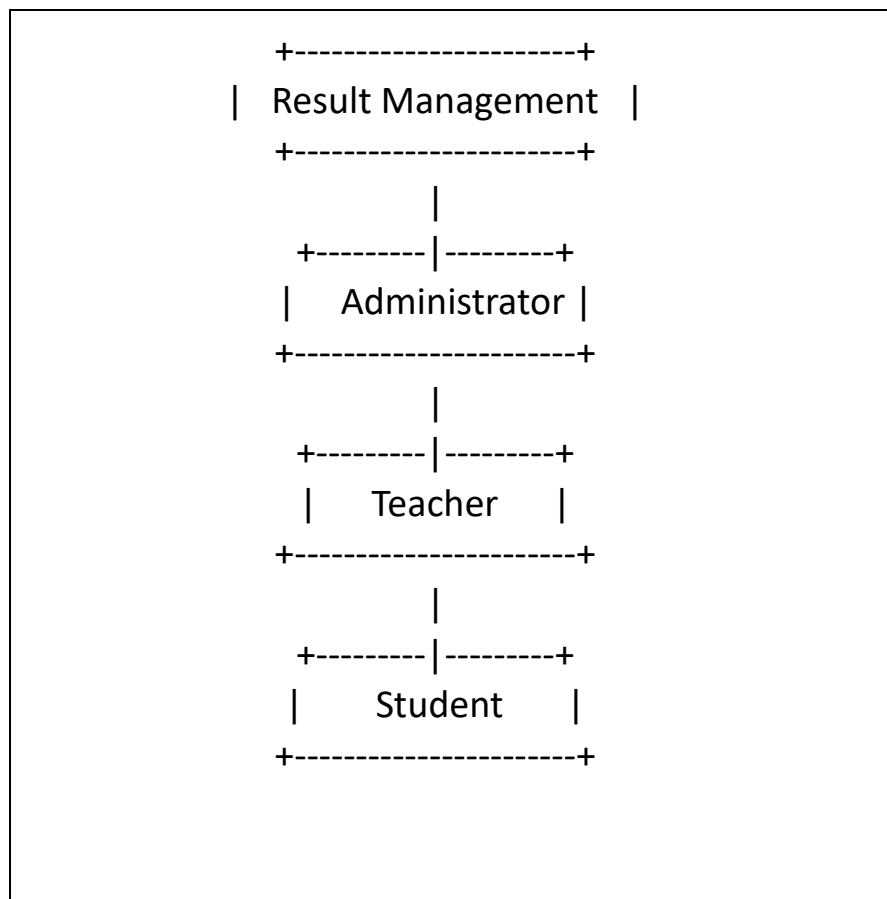
Update the documentation as the system evolves to reflect any changes or enhancements.

System design is an iterative process that may involve collaboration with stakeholders, feedback from users, and regular refinement as the project progresses. It's important to balance design considerations with the project's time and resource constraints while ensuring the Result Management System meets the functional and non-functional requirements.

CHAPTER 8

Use Case Diagram:

The use case diagram depicts the interactions between the users and the system. It identifies the different functionalities provided by the system and the actors involved. Here's an example of a use case diagram for a result management system.



Class Diagram:

The class diagram represents the static structure of the system, including classes, their attributes, and their relationships. Here's an example of a class diagram for a result management system.

```
+-----+
| ResultSystem |
+-----+
| - students: List<Student> |
| - teachers: List<Teacher> |
| - administrators: List<Administrator> |
+-----+
| + addStudent(student: Student): void |
| + addTeacher(teacher: Teacher): void |
| + addAdministrator(admin: Administrator): void |
| + getStudents(): List<Student> |
| + getTeachers(): List<Teacher> |
| + getAdministrators(): List<Administrator> |
+-----+

+-----+
| Student |
+-----+
| - studentId: int |
| - name: string |
| - grades: List<Grade> |
+-----+
| + getStudentId(): int |
| + getName(): string |
| + getGrades(): List<Grade> |
+-----+
```

```
+-----+
```

```

+-----+
| - teacherId: int |
| - name: string  |
+-----+
| + getTeacherId(): int |
| + getName(): string  |
+-----+

```

```

+-----+
| Administrator |
+-----+
| - adminId: int |
| - name: string |
+-----+
| + getAdminId(): int |
| + getName(): string |
+-----+

```

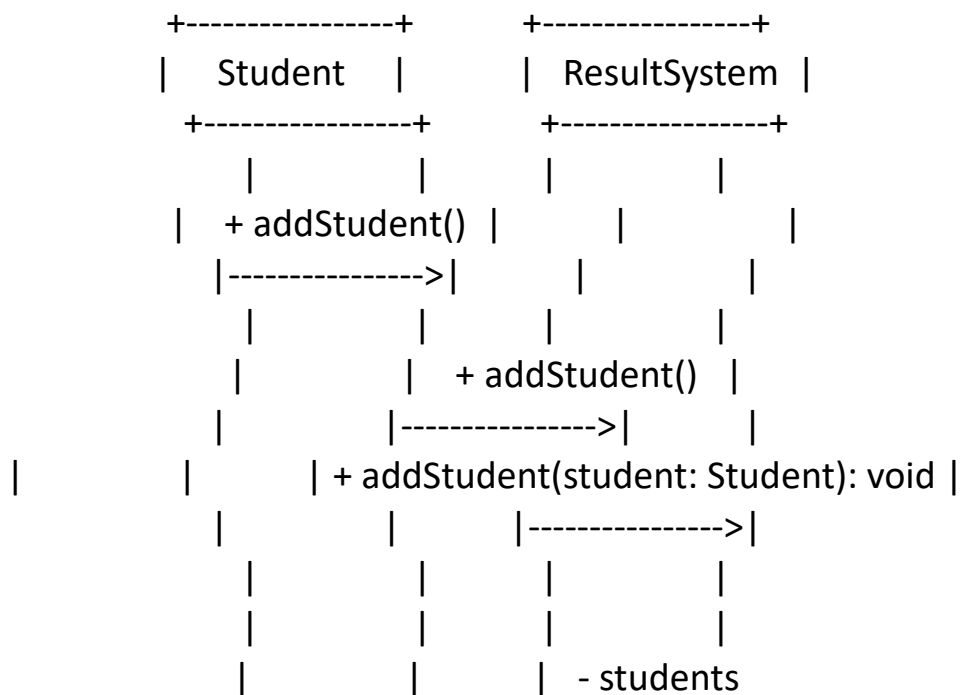
```

+-----+
|   Grade   |
+-----+
| - subject: string |
| - score: float   |
+-----+
| + getSubject(): string |
| + getScore(): float   | +-----+
--+
```


Sequence Diagram:

The sequence diagram shows the interactions between objects over time, demonstrating the flow of control and the order of method invocations.

Here's an example of a sequence diagram for a result management system, specifically for adding a student.



CHAPTER 9

CODING AND IMPLEMENTATION Sample

SOURCE CODE:

```
import tkinter as tk from tkinter
import messagebox import
sqlite3

# Function to create the database and tables

def create_database():    conn =

sqlite3.connect("results.db")    cursor =

conn.cursor()

    # Create the tables

cursor.execute("""

    CREATE TABLE IF NOT EXISTS semesters (

id INTEGER PRIMARY KEY AUTOINCREMENT,

sem_number INTEGER NOT NULL,        subject1 TEXT

NOT NULL,        subject1_internal INTEGER NOT

NULL,        subject1_external INTEGER NOT NULL,

subject2 TEXT NOT NULL,        subject2_internal

INTEGER NOT NULL,        subject2_external

INTEGER NOT NULL,        subject3 TEXT NOT NULL,

subject3_internal INTEGER NOT NULL,
```

```

subject3_external INTEGER NOT NULL,      subject4
TEXT NOT NULL,      subject4_internal INTEGER
NOT NULL,      subject4_external INTEGER NOT
NULL
    )
    """)

```

```

    conn.commit()

conn.close()

```

```

# Function to add a new semester def

```

```

add_semester(sem_number, subjects):

```

```

conn = sqlite3.connect("results.db")

```

```

cursor = conn.cursor()

```

```

    # Insert the semester into the database

```

```

cursor.execute("""      INSERT INTO semesters (
sem_number,      subject1, subject1_internal,
subject1_external,      subject2, subject2_internal,
subject2_external,      subject3, subject3_internal,
subject3_external,      subject4, subject4_internal,
subject4_external
    )
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    """, (
    sem_number,

```

```

        subjects[0][0], subjects[0][1], subjects[0][2],
subjects[1][0], subjects[1][1], subjects[1][2],      subjects[2][0],
subjects[2][1], subjects[2][2],      subjects[3][0], subjects[3][1],
subjects[3][2]
    ))

```

```

        conn.commit()

conn.close()

```

Function to update marks for a specific semester

```

def update_marks(sem_number, subjects):    conn
= sqlite3.connect("results.db")    cursor =
conn.cursor()

```

Update the marks in the database

```

cursor.execute("""

        UPDATE semesters        SET

subject1_internal=?, subject1_external=?,
subject2_internal=?, subject2_external=?,
subject3_internal=?, subject3_external=?,
subject4_internal=?, subject4_external=?

        WHERE sem_number=?

        """, (        subjects[0][1],
subjects[0][2],        subjects[1][1],
subjects[1][2],        subjects[2][1],
subjects[2][2],        subjects[3][1],
subjects[3][2],        sem_number

```

```
))
```

```
conn.commit()
```

```
conn.close()
```

```
# Function to handle the submit button click event def
```

```
submit():
```

```
    sem_number = sem_entry.get()
```

```
    subject1_name = subject1_entry.get()
```

```
subject1_internal = subject1_internal_entry.get()
```

```
subject1_external = subject1_external_entry.get()
```

```
    subject2_name = subject2_entry.get()
```

```
subject2_internal = subject2_internal_entry.get()
```

```
subject2_external = subject2_external_entry.get()
```

```
    subject3_name = subject3_entry.get()
```

```
subject3_internal = subject3_internal_entry.get()
```

```
subject3_external = subject3_external_entry.get()
```

```
    subject4_name = subject4_entry.get()
```

```
subject4_internal = subject4_internal_entry.get()
```

```
subject4_external = subject4_external_entry.get()
```

```
subjects = [
```

```
    (subject1_name, subject1_internal, subject1_external),
```

```
from flask import Flask,redirect,url_for

app=Flask(__name__)

@app.route('/')

def home():

    return 'hello world!'

@app.route('/second/<name>/<int:score>')

def second(name,score):    if score>=35:

    return redirect(url_for('score',var2=score,name=name))

    else:

    return redirect(url_for('fail'))

@app.route('/score/<name>/<int:var2>')

def score(name,var2):    if

35<=var2<=50:

    return f'{name},you are passed in third class'

elif 51<=var2<=75:

    return f'{name},you are passed in third class'

else:

    return f'{name},you are passed in first class'

@app.route('/third') def

third():

    return 'i am at third page'

@app.route('/pass') def

pass1():
```

```

    return 'u are passed'

@app.route('/fail') def
fail():

    return 'you are failed'

app.run(use_reloader=True,debug=True)

from flask import Flask,redirect,url_for

app=Flask(__name__) @app.route('/') def
home():

    return 'hello world!'

@app.route('/second/<name>/<int:score>')

def second(name,score):    if score>=35:

    return redirect(url_for('score',var2=score,name=name))

    else:

    return redirect(url_for('fail'))

@app.route('/score/<name>/<int:var2>')

def score(name,var2):

    if 35<=var2<=50:

        return f'{name},you are passed in third class'

    elif 51<=var2<=75:

        return f'{name},you are passed in third class'

    else:

        return f'{name},you are passed in first class'

@app.route('/third')

def third():

```

```

    return 'i am at third page'

@app.route('/pass')
def pass1():
    return 'u are passed'

@app.route('/fail') def
fail():
    return 'you are failed'

app.run(use_reloader==True,debug=True)

import tkinter as tk from tkinter
import messagebox import
sqlite3

# Function to create the database and tables

def create_database():    conn =
sqlite3.connect("results.db")    cursor =
conn.cursor()

# Create the tables

cursor.execute("""

    CREATE TABLE IF NOT EXISTS semesters (

id INTEGER PRIMARY KEY AUTOINCREMENT,

sem_number INTEGER NOT NULL,          subject1 TEXT

NOT NULL,          subject1_internal INTEGER NOT

NULL,          subject1_external INTEGER NOT NULL,

subject2 TEXT NOT NULL,

```



```

        subject2_internal INTEGER NOT NULL,
subject2_external INTEGER NOT NULL,
subject3 TEXT NOT NULL,        subject3_internal
INTEGER NOT NULL,        subject3_external
INTEGER NOT NULL,        subject4 TEXT NOT
NULL,        subject4_internal INTEGER NOT
NULL,        subject4_external INTEGER NOT
NULL
    )
    """)

```

```

conn.commit()

conn.close()

```

```

# Function to add a new semester def

```

```

add_semester(sem_number, subjects):

```

```

conn = sqlite3.connect("results.db")

```

```

cursor = conn.cursor()

```

```

    # Insert the semester into the database

```

```

cursor.execute("""        INSERT INTO semesters (
sem_number,        subject1, subject1_internal,
subject1_external,        subject2, subject2_internal,
subject2_external,        subject3, subject3_internal,
subject3_external,        subject4, subject4_internal,
subject4_external        )

```

```
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

```
""", (
```

```
    sem_number,    subjects[0][0],
```

```
subjects[0][1], subjects[0][2],    subjects[1][0],
```

```
subjects[1][1], subjects[1][2],    subjects[2][0],
```

```
subjects[2][1], subjects[2][2],    subjects[3][0],
```

```
subjects[3][1], subjects[3][2]
```

```
))
```

```
conn.commit()
```

```
conn.close()
```

```
# Function to update marks for a specific semester
```

```
def update_marks(sem_number, subjects):    conn
```

```
= sqlite3.connect("results.db")    cursor =
```

```
conn.cursor()
```

```
# Update the marks in the database
```

```
cursor.execute("""
```

```
    UPDATE semesters    SET
```

```
subject1_internal=?, subject1_external=,
```

```
subject2_internal=?, subject2_external=,
```

```
subject3_internal=?, subject3_external=,
```

```
subject4_internal=?, subject4_external=
```

```
    WHERE sem_number=
```

```
""", (    subjects[0][1],
```

```
subjects[0][2],    subjects[1][1],
```

```
subjects[1][2],    subjects[2][1],  
subjects[2][2],    subjects[3][1],  
subjects[3][2],    sem_number  
    ))
```

```
    conn.commit()  
  
conn.close()
```

```
# Function to handle the submit button click event def
```

```
submit():
```

```
    sem_number = sem_entry.get()
```

```
    subject1_name = subject1_entry.get()
```

```
subject1_internal = subject1_internal_entry.get()
```

```
subject1_external = subject1_external_entry.get()
```

```
    subject2_name = subject2_entry.get()
```

```
subject2_internal = subject2_internal_entry.get()
```

```
subject2_external = subject2_external_entry.get()
```

```
    subject3_name = subject3_entry.get()
```

```
subject3_internal = subject3_internal_entry.get()
```

```
subject3_external = subject3_external_entry.get()
```

```
subject4_name = subject4_entry.get()    subject4_internal
```

```
= subject4_internal_entry.get()    subject4_external =
```

```
subject4_external_entry.get()
```

```
subjects = [  
    (subject1_name, subject1_internal, subject1_external),  
  
    from itsdangerous import URLSafeTimedSerializer  
    from key import secret_key def token(email,salt):  
        serializer=URLSafeTimedSerializer(secret_key)  
    return serializer.dumps(email,salt=salt)  
  
import smtplib  
from email.message import EmailMessage def  
sendmail(to,subject,body):  
    server=smtplib.SMTP_SSL('smtp.gmail.com',465)  
    server.login('bodapatitejaswipratap8@gmail.com','jnmvxfxwvvozcoza')  
    msg=EmailMessage()  
    msg['From']='bodapatitejaswipratap8@gmail.com'  
    msg['Subject']=subject    msg['To']=to  
    msg.set_content(body)    server.send_message(msg)  
    server.quit()  
  
secret_key=b'\xa0S\xa5\x89\x06A\xea\x076\xad\xc32\x8e\xce\x03\xaf\x0e  
VE\xa0\xeeX\xa7\xc3RL\xc0\x96\xb1A\xad\xcc\xdc\xdd\xads\xe2\xce5\x0f'  
salt1='confirmation'  
key='4349fb1160ac40a2ba632ffd9b9294cf'  
  
body { font-family: Arial, sans-  
    serif; background-color:  
    #f2f2f2;  
}  
  
h2 {  text-align:  
center;  color:  
#333;  margin-  
top: 20px;  
}
```

```
.login-form { max-  
width: 300px; margin:  
0 auto; background-  
color: #fff; padding:  
20px; border-radius:  
5px;  
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);  
}
```

```
.login-form label {  
display: block; margin-  
bottom: 10px; color:  
#333; font-weight:  
bold;  
}
```

```
.login-form input[type="text"], .login-form  
input[type="password"] {  
    width: 100%;  
padding: 8px; border:  
1px solid #ccc; border-  
radius: 4px; font-size:  
14px;  
}
```

```
.login-form input[type="submit"] { background-  
color: #4CAF50;  
    color: #fff;  
    border: none;  
    padding: 10px 20px;  
border-radius: 4px;  
cursor: pointer; font-size:  
14px;  
    margin-top: 10px;  
}
```

```
.signup-link {  text-align: center;
margin-top: 10px;
}
```

```
.signup-link a {  color: #333;  text-decoration: none;  font-size: 14px;
}
```

```
.signup-link a:hover {
  text-decoration: underline;
}
```

```
body {  font-family: Arial, sans-serif;
background-color: #f2f2f2;
}
```

```
h2 {  text-align: center;  color: #333;
margin-top: 20px;
}
```

```
.profile {
  max-width: 400px;
  margin: 0 auto;
  background-color: #fff;
padding: 20px;  border-radius: 5px;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}
```

```
.profile label {
```

```
display: block; margin-  
bottom: 10px; color: #333;  
font-weight: bold;  
}
```

```
.profile input[type="text"],  
.profile input[type="email"],  
.profile input[type="password"] {  
width: 100%; padding: 8px;  
border: 1px solid #ccc; border-  
radius: 4px; font-size: 14px;  
}
```

```
.profile input[type="submit"] { background-  
color: #4CAF50;  
color: #fff;  
border: none;  
padding: 10px 20px;  
border-radius: 4px;  
cursor: pointer;  
font-size: 14px;  
margin-top: 10px;  
}
```

```
.login-link { text-align:  
center; margin-top:  
20px;  
}
```

```
.login-link a { color:  
#333; text-decoration:  
none; font-size: 14px;  
}
```

```
.login-link a:hover {  
text-decoration: underline;  
}
```

```
/* Global Styles */
@import
url('https://fonts.googleapis.com/css2?family=Poppins&display=swap');

* {
padding: 0;
margin: 0;
font-family: 'Poppins', 'monospace';
}

body {
background: url('https://wallpaperset.com/w/full/d/3/b/28666.jpg');
background-repeat: no-repeat;
background-size: cover;
}

/* Navigation Styles */
ul {
list-style: none;
background: linear-gradient(135deg, grey, white);
}
ul li
{
display: inline-block;
position: relative;
}

ul li a {
display: block;
padding: 25px 25px;
color: #fff; text-
decoration: none; text-
align: center; font-size:
20px;
```



```
}
```

```
ul li ul.dropdown li {  
display: block;  
}
```

```
ul li ul.dropdown {  
width: 100%;  
background: linear-gradient(135deg, grey, white);  
position: absolute;  
z-index: 999; display:  
none;  
}
```

```
ul li a:hover {  
background: grey;  
}
```

```
ul li:hover ul.dropdown {  
display: block;  
}
```

```
/* Logo Styles */  
.logo {  
padding: 0%;  
margin: 0%; align-  
items: center;  
padding-left: 40%;  
color: #fff;  
}
```

```
/* Container Styles */ .container {  
position: absolute; left: 48%;  
top: 45%; transform: translate(-  
50%, -50%); padding: 10px;  
}
```

```
input { outline: none;
box-sizing: border-box;
height: 60px; width: 0px;
padding: 0 20px; color:
black; border-radius:
50px; font-size: 20px;
border: 1px solid #c9c0cc;
transition: all .7s ease;
}
```

```
::placeholder {
color: grey;
}
```

```
.container:hover input {
width: 350px;
}
```

```
.container:hover i {
transform: rotate(-360deg);
}
```

```
.btn:hover {
background: grey;
}
```

```
.credentials { padding-
top: 0%; padding-left:
75%; padding-bottom:
0%;
}
```

```
.credentials li a {
color: black;
}
```

```
body { font-family: Arial,
sans-serif; margin: 20px;
}
h2 { color:
#333;
}
label { display:
block; margin-
top: 10px;
}
input[type="text"], textarea {
width: 300px; padding:
5px;
}
textarea {
height: 100px;
}
input[type="submit"] { background-
color: #4CAF50;
color: white;
border: none;
padding: 10px 20px;
text-align: center;
text-decoration: none;
display: inline-block;
font-size: 14px;
margin-top: 20px;
cursor: pointer;
}
```

@import

url('https://fonts.googleapis.com/css2?family=Poppins&display=swap');

* {

padding: 0;

margin: 0;

```
    font-family: 'Poppins', 'monospace';
}
body{  background-image:
url('https://i.ytimg.com/vi/UEEXWq5kQRk/maxresdefault.jpg');
background-size: cover;  background-position: center;
}

ul { list-style:
none;
background: linear-gradient(135deg, #71b7e6, #9b59b6);
}
ul li
{
    display: inline-block;
    position: relative;
}

ul li a {  display: block;
padding: 25px 25px;
color: #FFF;  text-
decoration: none;
text-align: center;
font-size: 20px;
}

ul li ul.dropdown li {
display: block;
}

ul li ul.dropdown { width: 100%; background: linear-
gradient(135deg, #71b7e6, #9b59b6); position: absolute;
z-index: 999; display: none;
}

ul li a:hover {
background: #6d30e9;
}
```

```
ul li:hover ul.dropdown { display:
block;
}
.logo{ padding: 0%;
margin: 0%; align-
items: center;
padding-left: 50%;
color: #9b59b6;
}
h5{
padding-left: 47%;
}
.container{ position: absolute;
left: 48%; top: 45%; transform:
translate(-50%, -50%); padding:
10px;
}
input{ outline: none;
box-sizing: border-box;
height: 60px; width: 0px;
padding: 0 20px; color:
black; border-radius: 50px;
font-size: 20px; border:
1px solid #9b59b6;
transition: all .7s ease;
}
::placeholder{
}
.btn{ position:
absolute; right: 0px;
top: 0px; width: 80px;
height: 80px;
background: #9b59b6;
line-height: 80px;
border-radius: 50%;
```

```
text-align: center;
cursor: pointer;
  transition: .5s;
}
.btn i{ font-size:
25px; color: white;
line-height: 80px;
transition: all .7s ease;
}
.container:hover input{
width: 350px;
}
.container:hover i{
transform: rotate(-360deg);
}
.btn:hover{
background: #9b59b6;
}
.slogan{ justify-
content: center;
padding-top: 5.5%;
}
.slogan p { box-sizing:
border-box; justify-
content: center; align-
items: center; text-
align: center; padding-
top: 13%;
}
p{ align-self: center;
padding: 10px 27px 30px 40px;
}
.credentials{ padding-
top: 0%; padding-left:
75%; padding-bottom:
0%;
```

```
}  
.button{  
    background: #9b59b6 ;    text-decoration-  
color: #9b59b6;  
}
```

```
.newsletter {  
position: relative;  
}
```

```
.newsletter::after {  
    content: "Click me for your newsletter";  
    position: absolute;  
top: 100%;  
    left: 50%;    transform:  
translateX(-50%);  
    opacity: 0;  
    transition: opacity 0.3s ease;  
}
```

```
.newsletter:hover::after {  
opacity: 1;  
}
```

```
.container {    max-  
width: 800px;  
margin: 0 auto;  
padding: 20px;  
}
```

```
h1 {    text-align: center;  
font-size: 24px;  
margin-bottom: 20px;  
}
```

```
h2 {    font-size:  
18px;
```

```
    margin-bottom: 10px;
}
```

```
table {    width: 100%;
border-collapse: collapse;
margin-bottom: 20px;
}
```

```
th, td {
padding: 8px;
text-align: left;
    border-bottom: 1px solid #ddd;
}
```

```
th {
    background-color: #f2f2f2;
}
```

```
tr:hover {
    background-color: #f5f5f5;
}
```

```
/* Add any additional styles or modifications as needed */
```

```
/* style.css */
```

```
body {
    font-family: Arial, sans-serif;    background-
color: #f1f1f1;
    margin: 0;
padding: 0;
}
```

```
.container {    max-
width: 600px;
margin: 0 auto;
```



```
padding: 20px;
background-color: #fff;
border-radius: 5px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

```
h1 { text-align: center;
margin-bottom: 20px;
}
```

```
form {
display: grid;
gap: 10px;
}
```

```
label { font-
weight: bold;
}
```

```
input[type="text"],
input[type="number"] {
width: 100%; padding:
10px; border: 1px
solid #ccc; border-
radius: 3px;
}
```

```
h3 { margin-top:
20px;
}
```

```
input[type="submit"] { background-
color: #4CAF50;
color: #fff;
padding: 10px 20px;
border: none;
border-radius: 3px;
```

```
    cursor: pointer;
}

input[type="submit"]:hover { background-
color: #45a049;
}
```

```
* styles.css */
```

```
body { font-family: Arial,
sans-serif; background-
color: #f2f2f2; margin: 0;
padding: 0;
}
```

```
h1 {
color: #333;
}
```

```
form { width: 300px;
margin: 0 auto;
padding: 20px;
background-color: #fff;
border: 1px solid #ccc;
border-radius: 5px;
}
```

```
label { display: block;
margin-bottom: 10px;
color: #333;
}
```

```
input[type="text"] { width:
100%; padding: 8px;
font-size: 16px;
border: 1px solid #ccc;
border-radius: 5px;
```

```
}
```

```
input[type="submit"] {  
width: 100%; padding:  
8px; font-size: 16px;  
background-color: #333;  
color: #fff; border:  
none; border-radius:  
5px;  
    cursor: pointer;  
}
```

```
input[type="submit"]:hover { background-  
color: #555;  
}
```

```
.container {  
width: 80%;  
margin: 0 auto;  
text-align: center;  
}
```

```
h1 { margin-top:  
20px;  
}
```

```
.results {  
    margin-top: 30px;  
}
```

```
h2 {  
    margin-bottom: 10px;  
}
```

```
table { width: 100%;  
border-collapse: collapse;  
}
```

```
th, td { padding: 8px;
border: 1px solid #ddd;
}
```

```
th {
background-color: #f2f2f2;
}
```

```
tr:nth-child(even) {
background-color: #f9f9f9;
}
```

```
body { font-family: Arial,
sans-serif; background-
color: #f2f2f2;
padding: 20px;
}
```

```
h2 { color:
#333;
}
```

```
p { color:
#666;
}
```

```
a { color:
#337ab7;
text-decoration: none;
}
```

```
a:hover {
text-decoration: underline;
}
```

```
import sys
import os
```

```
is_pypy = '_pypy_' in sys.builtin_module_names
```

```
def warn_distutils_present():    if
'distutils' not in sys.modules:
    return    if is_pypy and
sys.version_info < (3, 7):
    # PyPy for 3.6 unconditionally imports distutils, so bypass the warning
    # https://foss.heptapod.net/pypy/pypy/-
/blob/be829135bc0d758997b3566062999ee8b23872b4/libpython/3/site.py#
L250
    return
import warnings

warnings.warn(
    "Distutils was imported before Setuptools, but importing Setuptools "
    "also replaces the `distutils` module in `sys.modules`. This may lead "
    "to undesirable behaviors or errors. To avoid these issues, avoid "
    "using distutils directly, ensure that setuptools is installed in the "
    "traditional way (e.g. not an editable install), and/or make sure "
    "that setuptools is always imported before distutils."
)
```

```
def clear_distutils():    if
'distutils' not in sys.modules:
return
import warnings

warnings.warn("Setuptools is replacing distutils.")
mods = [    name
    for name in sys.modules
    if name == "distutils" or name.startswith("distutils.")
]
for name in mods:
    del sys.modules[name]
```

```
def enabled():
```

```
    """
```

```
    Allow selection of distutils by environment variable.
```

```
    """
```

```
    which = os.environ.get('SETUPTOOLS_USE_DISTUTILS', 'local')
```

```
    return which == 'local'
```

```
def ensure_local_distutils():
```

```
    import importlib
```

```
    clear_distutils()
```

```
    # With the DistutilsMetaFinder in place, #
```

```
    perform an import to cause distutils to be #
```

```
    loaded from setuptools._distutils. Ref #2906.
```

```
    with shim():
```

```
        importlib.import_module('distutils')
```

```
    # check that submodules load as expected
```

```
    core = importlib.import_module('distutils.core')
```

```
    assert 'distutils' in core.file, core.file_ assert
```

```
    'setuptools._distutils.log' not in sys.modules def
```

```
    do_override():
```

```
        """
```

```
        Ensure that the local copy of distutils is preferred over stdlib.
```

```
        See https://github.com/pypa/setuptools/issues/417#issuecomment-392298401
```

```
        for more motivation.
```

```
        """ if
```

```
enabled():
```

```
    warn_distutils_present()
```

```
    ensure_local_distutils()
```

```
class _TrivialRe:
    def
    _init_(self, *patterns):
    self._patterns = patterns
```

```
    def match(self, string):
        return all(pat in string for pat in self._patterns)
```

```
class DistutilsMetaFinder:
    def find_spec(self,
fullname, path, target=None):
        # optimization: only consider top level modules and those
        # found in the CPython test suite.      if path is not None and
        not fullname.startswith('test.'):      return

        method_name = 'spec_for_{fullname}'.format(**locals())
        method = getattr(self, method_name, lambda: None)      return
        method()
```

```
    def spec_for_distutils(self):
if self.is_cpython():
    return
    import importlib
    import importlib.abc
    import importlib.util
```

```
try:
    mod = importlib.import_module('setuptools._distutils')
except Exception:
    # There are a couple of cases where setuptools._distutils
    # may not be present:
    # - An older Setuptools without a local distutils is
    # taking precedence. Ref #2957.
    # - Path manipulation during sitecustomize removes
    # setuptools from the path but only after the hook      #
    # has been loaded. Ref #2980.
    # In either case, fall back to stdlib behavior.
    return
```

```

class DistutilsLoader(importlib.abc.Loader):
def create_module(self, spec):
mod.__name__ = 'distutils'
    return mod

    def exec_module(self, module):
        pass

    return importlib.util.spec_from_loader(
        'distutils', DistutilsLoader(), origin=mod.__file__
    )

    @staticmethod
def is_cpython():
    """
    Suppress supplying distutils for CPython (build and tests).
    Ref #2965 and #3007.
    """
    return os.path.isfile('pybuilddir.txt')
def spec_for_pip(self):
    """
    Ensure stdlib distutils when running under pip.
    See pypa/pip#8761 for rationale.
    """
    if self.pip_imported_during_build():
return    clear_distutils()
        self.spec_for_distutils = lambda: None

    @classmethod    def
pip_imported_during_build(cls):
    """
    Detect if pip is being imported in a build script. Ref #2355.
    """
    import traceback

    return any(

```



```
        cls.frame_file_is_setup(frame) for frame, line in
traceback.walk_stack(None)
    )
```

```
@staticmethod    def
frame_file_is_setup(frame):
    """
    Return True if the indicated frame suggests a setup.py file.
    """
    # some frames may not have _file_ (#2940)
    return frame.f_globals.get('_file_', '').endswith('setup.py')
```

```
def spec_for_sensitive_tests(self):
    """
    Ensure stdlib distutils when running select tests under CPython.
```

```
python/cpython#91169
    """
```

```
clear_distutils()
self.spec_for_distutils = lambda: None
```

```
sensitive_tests = (
    [
        'test.test_distutils',
        'test.test_peg_generator',
        'test.test_importlib',
    ]
    if sys.version_info < (3, 10)
else [
    'test.test_distutils',
]
)
```

```
for name in DistutilsMetaFinder.sensitive_tests:
    setattr(
```

```
        DistutilsMetaFinder,
f'spec_for_{name}',
        DistutilsMetaFinder.spec_for_sensitive_tests,
    )
```

```
DISTUTILS_FINDER = DistutilsMetaFinder()
```

```
def add_shim():
    DISTUTILS_FINDER in sys.meta_path or insert_shim()
```

```
class shim:
    def
    _enter_(self):
    insert_shim()

    def _exit_(self, exc, value, tb):
        remove_shim()
```

```
def insert_shim():
    sys.meta_path.insert(0, DISTUTILS_FINDER)
```

```
def remove_shim():
    try:
        sys.meta_path.remove(DISTUTILS_FINDER)
    except ValueError:
        pass
```

```
import sys
import os
```

```
is_pypy = '_pypy_' in sys.builtin_module_names
```

```
def warn_distutils_present():
    if
    'distutils' not in sys.modules:
```

```

        return if is_pypy and
sys.version_info < (3, 7):
    # PyPy for 3.6 unconditionally imports distutils, so bypass the warning
    # https://foss.heptapod.net/pypy/pypy/-
/blob/be829135bc0d758997b3566062999ee8b23872b4/libpython/3/site.py#
L250
    return
import warnings

warnings.warn(
    "Distutils was imported before Setuptools, but importing Setuptools "
    "also replaces the `distutils` module in `sys.modules`. This may lead "
    "to undesirable behaviors or errors. To avoid these issues, avoid "
    "using distutils directly, ensure that setuptools is installed in the "
    "traditional way (e.g. not an editable install), and/or make sure "
    "that setuptools is always imported before distutils."
)

def clear_distutils():
    if
'distutils' not in sys.modules:
        return
    import warnings

    warnings.warn("Setuptools is replacing distutils.")
    mods = [
        name
        for name in sys.modules
        if name == "distutils" or name.startswith("distutils.")
    ]
    for name in mods:
        del sys.modules[name]

def enabled():
    """
    Allow selection of distutils by environment variable.
    """

```

```
    which = os.environ.get('SETUPTOOLS_USE_DISTUTILS', 'local')
    return which == 'local'
```

```
def ensure_local_distutils():
```

```
    import importlib
```

```
    clear_distutils()
```

```
    # With the DistutilsMetaFinder in place, #
    perform an import to cause distutils to be #
    loaded from setuptools._distutils. Ref #2906.
```

```
    with shim():
```

```
        importlib.import_module('distutils')
```

```
    # check that submodules load as expected
```

```
    core = importlib.import_module('distutils.core')
```

```
    assert 'distutils' in core.file, core.file_
```

```
    assert 'setuptools._distutils.log' not in sys.modules
```

```
def do_override():
```

```
    """
```

```
    Ensure that the local copy of distutils is preferred over stdlib.
```

```
    See https://github.com/pypa/setuptools/issues/417#issuecomment-392298401
```

```
    for more motivation.
```

```
    """    if
```

```
enabled():
```

```
    warn_distutils_present()
```

```
    ensure_local_distutils()
```

```
class _TrivialRe:    def
```

```
    _init_(self, *patterns):
```

```
    self._patterns = patterns
```

```
def match(self, string):
    return all(pat in string for pat in self._patterns)
```

```
class DistutilsMetaFinder:
    def find_spec(self, fullname, path, target=None):
        # optimization: only consider top level modules and those
        # found in the CPython test suite.      if path is not None and
        not fullname.startswith('test.'):      return

        method_name = 'spec_for_{fullname}'.format(**locals())
        method = getattr(self, method_name, lambda: None)
        return method()
```

```
    def spec_for_distutils(self):
        if self.is_cpython():
            return
```

```
        import importlib
        import importlib.abc
        import importlib.util
```

```
    try:
        mod = importlib.import_module('setuptools._distutils')
    except Exception:
        # There are a couple of cases where setuptools._distutils
        # may not be present:
        # - An older Setuptools without a local distutils is
        # taking precedence. Ref #2957.
        # - Path manipulation during sitecustomize removes
        # setuptools from the path but only after the hook      #
        # has been loaded. Ref #2980.
        # In either case, fall back to stdlib behavior.
        return
```

```

class DistutilsLoader(importlib.abc.Loader):
def create_module(self, spec):
mod._name_ = 'distutils'
    return mod

    def exec_module(self, module):
        pass

    return importlib.util.spec_from_loader(
        'distutils', DistutilsLoader(), origin=mod._file_
    )

    @staticmethod
def is_cpython():
    """
    Suppress supplying distutils for CPython (build and tests).
    Ref #2965 and #3007.
    """
    return os.path.isfile('pybuilddir.txt')

def spec_for_pip(self):
    """
    Ensure stdlib distutils when running under pip.
    See pypa/pip#8761 for rationale.
    """
    if self.pip_imported_during_build():
return
        clear_distutils()
        self.spec_for_distutils = lambda: None

    @classmethod    def
pip_imported_during_build(cls):
    """
    Detect if pip is being imported in a build script. Ref #2355.
    """
    import traceback

```

```
        return any(
            cls.frame_file_is_setup(frame) for frame, line in
            traceback.walk_stack(None)
        )
```

```
    @staticmethod    def
    frame_file_is_setup(frame):
        """
        Return True if the indicated frame suggests a setup.py file.
        """
        # some frames may not have _file_ (#2940)
        return frame.f_globals.get('_file_', '').endswith('setup.py')
```

```
    def spec_for_sensitive_tests(self):
        """
        Ensure stdlib distutils when running select tests under CPython.
        python/cpython#91169
        """
```

```
    clear_distutils()
    self.spec_for_distutils = lambda: None
```

```
    sensitive_tests = (
        [
            'test.test_distutils',
            'test.test_peg_generator',
            'test.test_importlib',
        ]
        if sys.version_info < (3, 10)
    else [
        'test.test_distutils',
    ]
    )
```

```
    for name in DistutilsMetaFinder.sensitive_tests:
        setattr(
```

```
        DistutilsMetaFinder,
f'spec_for_{name}',
        DistutilsMetaFinder.spec_for_sensitive_tests,
    )
```

```
DISTUTILS_FINDER = DistutilsMetaFinder()
```

```
def add_shim():
    DISTUTILS_FINDER in sys.meta_path or insert_shim()
```

```
class shim:
    def
    _enter_(self):
    insert_shim()
    def
    _exit_(self, exc,
    value, tb):
        remove_shim()
```

```
def insert_shim():
    sys.meta_path.insert(0, DISTUTILS_FINDER)
```

```
def remove_shim():
    try:
        sys.meta_path.remove(DISTUTILS_FINDER)
    except ValueError:
        pass
```

```
_import_('__distutils_hack').do_override()
```

```
from blinker.base import ANY from
blinker.base import NamedSignal from
blinker.base import Namespace from
blinker.base import receiver_connected from
```



```
blinker.base import Signal from blinker.base
import signal from blinker.base import
WeakNamespace
```

```
_all_ = [
    "ANY",
    "NamedSignal",
    "Namespace",
    "Signal",
    "WeakNamespace",
    "receiver_connected",
    "signal",
]
```

```
_version_ = "1.6.2"
```

```
# extracted from Louie, http://pylouie.org/
```

```
# updated for Python 3
```

```
#
```

```
# Copyright (c) 2006 Patrick K. O'Brien, Mike C. Fletcher,
```

```
#         Matthew R. Scott
```

```
#
```

```
# Redistribution and use in source and binary forms, with or without #
# modification, are permitted provided that the following conditions are #
# met:
```

```
#
```

```
# * Redistributions of source code must retain the above copyright #
# notice, this list of conditions and the following disclaimer.
```

```
#
```

```
# * Redistributions in binary form must reproduce the above
```

```
# copyright notice, this list of conditions and the following
```

```
# disclaimer in the documentation and/or other materials provided #
# with the distribution.
```

```
#
```

```
# * Neither the name of the <ORGANIZATION> nor the names of its #
# contributors may be used to endorse or promote products derived #
# from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
# CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR
# A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT
# OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
# LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
# OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
# ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
# THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
# DAMAGE.
#
# """Refactored 'safe reference from dispatcher.py"""
import operator import sys import traceback
import weakref

get_self = operator.attrgetter("_self_") get_func
= operator.attrgetter("_func_")

def safe_ref(target, on_delete=None):
    """Return a safe weak reference to a callable target.
```

- ``target``: The object to be weakly referenced, if it's a bound method reference, will create a BoundMethodWeakref, otherwise creates a simple weakref.
- ``on_delete``: If provided, will have a hard reference stored to the callable to be called after the safe reference goes out of scope with the reference object, (either a weakref or a BoundMethodWeakref) as argument.

```

"""
try:
    im_self = get_self(target)
except AttributeError:
    if
callable(on_delete):
        return weakref.ref(target, on_delete)
    else:
        return
weakref.ref(target) else:
    if
im_self is not None:
        # Turn a bound method into a BoundMethodWeakref instance.
        # Keep track of these instances for lookup by disconnect().
assert hasattr(target, "im_func") or hasattr(target, "_func_"), (
f"safe_ref target {target!r} has im_self, but no im_func, "
    "don't know how to create reference"
)
reference = BoundMethodWeakref(target=target,
on_delete=on_delete)
return reference

```

class BoundMethodWeakref:

"""'Safe' and reusable weak references to instance methods.

BoundMethodWeakref objects provide a mechanism for referencing a bound method without requiring that the method object itself (which is normally a transient object) is kept alive. Instead, the BoundMethodWeakref object keeps weak references to both the object and the function which together define the instance method.

Attributes:

- `__key__`: The identity key for the reference, calculated by the class's `calculate_key` method applied to the target instance method.
- `__deletion_methods__`: Sequence of callable objects taking single argument, a reference to this object which will be called when either the target object or target function is garbage collected (i.e. when this object becomes invalid). These are specified as the `on_delete` parameters of `safe_ref` calls.
- `__weak_self__`: Weak reference to the target object.
- `__weak_func__`: Weak reference to the target function.

Class Attributes:

- `__all_instances__`: Class attribute pointing to all live `BoundMethodWeakref` objects indexed by the class's `calculate_key(target)` method applied to the target objects. This weak value dictionary is used to short-circuit creation so that multiple references to the same (object, function) pair produce the same `BoundMethodWeakref` instance.

"""

```
_all_instances = weakref.WeakValueDictionary() # type: ignore[varannotated]
```

```
def _new_(cls, target, on_delete=None, *arguments, **named):  
    """Create new instance or return current instance.
```

Basically this method of construction allows us to short-circuit creation of references to already-referenced instance methods. The key corresponding to the target is calculated, and if there is already an existing reference, that is returned, with its `deletion_methods` attribute updated. Otherwise the new instance is created and registered in the table of already-referenced methods.

```

        """
        key = cls.calculate_key(target)
        current = cls._all_instances.get(key)      if
        current is not None:
            current.deletion_methods.append(on_delete)
        return current      else:
            base = super().new_(cls)
        cls._all_instances[key] = base
            base._init_(target, on_delete, *arguments, **named)
        return base

```

```

def _init_(self, target, on_delete=None):

```

```

    """Return a weak-reference-like instance for a bound method.

```

- ``target``: The instance-method target for the weak reference, must have `im_self` and `im_func` attributes and be reconstructable via the following, which is true of built-in instance methods::

```

        target.im_func._get_( target.im_self )

```

- ``on_delete``: Optional callback which will be called when this weak reference ceases to be valid (i.e. either the object or the function is garbage collected). Should take a single argument, which will be passed a pointer to this object.

```

        """

```

```

def remove(weak, self=self):

```

```

    """Set self.isDead to True when method or instance is destroyed."""

```

```

    methods = self.deletion_methods[:]      del self.deletion_methods[:]
    try:

```

```

        del self._class._all_instances[self.key]
    except KeyError:      pass      for
        function in methods:      try:      if
        callable(function):

```

```

        function(self)
except Exception:
try:
        traceback.print_exc()
except AttributeError:
e = sys.exc_info()[1]
        print(
                f"Exception during saferef {self} "
f"cleanup function {function}: {e}"
        )

        self.deletion_methods = [on_delete]
self.key = self.calculate_key(target)
im_self = get_self(target)    im_func =
get_func(target)
        self.weak_self = weakref.ref(im_self, remove)
self.weak_func = weakref.ref(im_func, remove)
        self.self_name = str(im_self)
self.func_name = str(im_func._name_)

    @classmethod    def
calculate_key(cls, target):
        """Calculate the reference key for this reference.

        Currently this is a two-tuple of the id()'s of the target
object and the target function respectively.
        """

        return (id(get_self(target)), id(get_func(target)))

    def _str_(self):
        """Give a friendly representation of the object."""
        return "{}({}.{})".format(
self._class.name_,
self.self_name,    self.func_name,
        )

    _repr_ = _str_

```

```
def _hash_(self):
    return hash((self.self_name, self.key))
```

```
def _nonzero_(self):
    """Whether we are still a valid reference."""
    return self() is not None
```

```
def _eq_(self, other):
    """Compare with another reference."""
    if not isinstance(other, self._class_):
        return
    operator.eq(self._class_, type(other))
    return operator.eq(self.key, other.key)
```

```
def _call_(self):
    """Return a strong reference to the bound method.
```

If the target cannot be retrieved, then will return None, otherwise returns a bound instance method for our object and function.

Note: You may call this method any number of times, as it does not invalidate the reference.

```
    """
    target = self.weak_self()
    if target is not None:
        function = self.weak_func()
        if function is not None:
            return function._get_(target)
    return None
```

```
"""Activate virtualenv for current interpreter:
```

```
Use exec(open(this_file).read(), {'_file_': this_file}).
```

This can be used when you must use an existing Python interpreter, not the virtualenv bin/python.

```
"""
```

```
from _future_ import annotations
```

```
import os
import sys
```

```
try:
```

```
    abs_file = os.path.abspath(_file_) except NameError: raise
    AssertionError("You must use exec(open(this_file).read(), {'_file_':
    this_file}))")
```

```
bin_dir = os.path.dirname(abs_file)
```

```
base = bin_dir[: -len("Scripts") - 1] # strip away the bin part from the _file_,
plus the path separator
```

```
# prepend bin to PATH (this file is inside the bin directory)
```

```
os.environ["PATH"] = os.pathsep.join([bin_dir] + os.environ.get("PATH",
    "").split(os.pathsep))
```

```
os.environ["VIRTUAL_ENV"] = base # virtual env is right above bin directory
```

```
# add the virtual environments libraries to the host python import mechanism
```

```
prev_length = len(sys.path) for lib in "..\\Lib\\site-
packages".split(os.pathsep): path =
```

```
os.path.realpath(os.path.join(bin_dir, lib))
```

```
site.addsitedir(path.decode("utf-8") if "" else path)
```

```
sys.path[:] = sys.path[prev_length:] + sys.path[0:prev_length]
```

```
sys.real_prefix = sys.prefix
sys.prefix = base
```

```
from _future_ import annotations
```

```
import asyncio
```

```
import sys
```

```
import typing as t from functools
```

```
import partial from weakref
```

```
import ref
```

```
from blinker._saferef import BoundMethodWeakref
```



```
IdentityType = t.Union[t.Tuple[int, int], str, int]
```

```
class _symbol:    def
_init_(self, name):
    """Construct a new named symbol."""
self._name_ = self.name = name
```

```
    def _reduce_(self):
        return symbol, (self.name,)
```

```
    def _repr_(self):
return self.name
```

```
symbol.name_ = "symbol"
```

```
class symbol:
    """A constant symbol.
```

```
>>> symbol('foo') is symbol('foo')
True
>>> symbol('foo')
foo
```

A slight refinement of the `MAGICCOOKIE=object()` pattern. The primary advantage of `symbol()` is its `repr()`. They are also singletons.

Repeated calls of `symbol('name')` will all return the same instance.

```
"""
```

```
symbols = {} # type: ignore[var-annotated]
```

```
    def _new_(cls, name):
try:
```

```

        return cls.symbols[name]
except KeyError:
    return cls.symbols.setdefault(name, _symbol(name))

def hashable_identity(obj: object) -> IdentityType:
    if hasattr(obj, "_func_"):
        return (id(obj._func), id(obj.self_)) # type: ignore[attr-defined]
    elif hasattr(obj, "im_func"):
        return (id(obj.im_func), id(obj.im_self)) # type: ignore[attr-defined]
    elif isinstance(obj, (int, str)):
        return obj
    else:
        return id(obj)

```

```

WeakTypes = (ref, BoundMethodWeakref)

```

```

class annotatable_weakref(ref):
    """A weakref.ref that supports custom instance attributes."""

    receiver_id: t.Optional[IdentityType]
    sender_id: t.Optional[IdentityType]

```

```

def reference( # type: ignore[no-untyped-def]
    object, callback=None, **annotations ) ->
    annotatable_weakref:
    """Return an annotated weak ref."""
    if callable(object):
        weak = callable_reference(object, callback)
    else:
        weak = annotatable_weakref(object, callback)
    for key, value in annotations.items():
        setattr(weak, key, value)
    return weak # type: ignore[no-any-return]

```

```

def callable_reference(object, callback=None):
    """Return an annotated weak ref, supporting bound instance methods."""
    if hasattr(object, "im_self") and object.im_self is not None:
        return BoundMethodWeakref(target=object, on_delete=callback)
    elif hasattr(object, "_self") and object._self is not None:
        return BoundMethodWeakref(target=object, on_delete=callback)
    return annotatable_weakref(object, callback)

```

```

class lazy_property:
    """A @property that is only evaluated once."""

```

```

    def __init__(self, deferred):
self._deferred = deferred      self._doc_
= deferred._doc_

    def __get__(self, obj, cls):
if obj is None:
return self
    value = self._deferred(obj)
    setattr(obj, self.deferred.name_, value)
return value

```

```

def is_coroutine_function(func: t.Any) -> bool:
    # Python < 3.8 does not correctly determine partially wrapped
    # coroutine functions are coroutine functions, hence the need for
    # this to exist. Code taken from CPython.    if sys.version_info >= (3,
    8):
        return asyncio.iscoroutinefunction(func)
    else:
        # Note that there is something special about the AsyncMock
        # such that it isn't determined as a coroutine function    #
        without an explicit check.    try:
            from unittest.mock import AsyncMock # type: ignore[attr-defined]

            if isinstance(func, AsyncMock):

```

```
        return True
except ImportError:
    # Not testing, no async test to import
    pass

    while inspect.ismethod(func):
        func = func._func_
    while isinstance(func, partial):
        func = func.func    if not
    inspect.isfunction(func):
        return False

    if func._code_.co_flags & inspect.CO_COROUTINE:
        return True

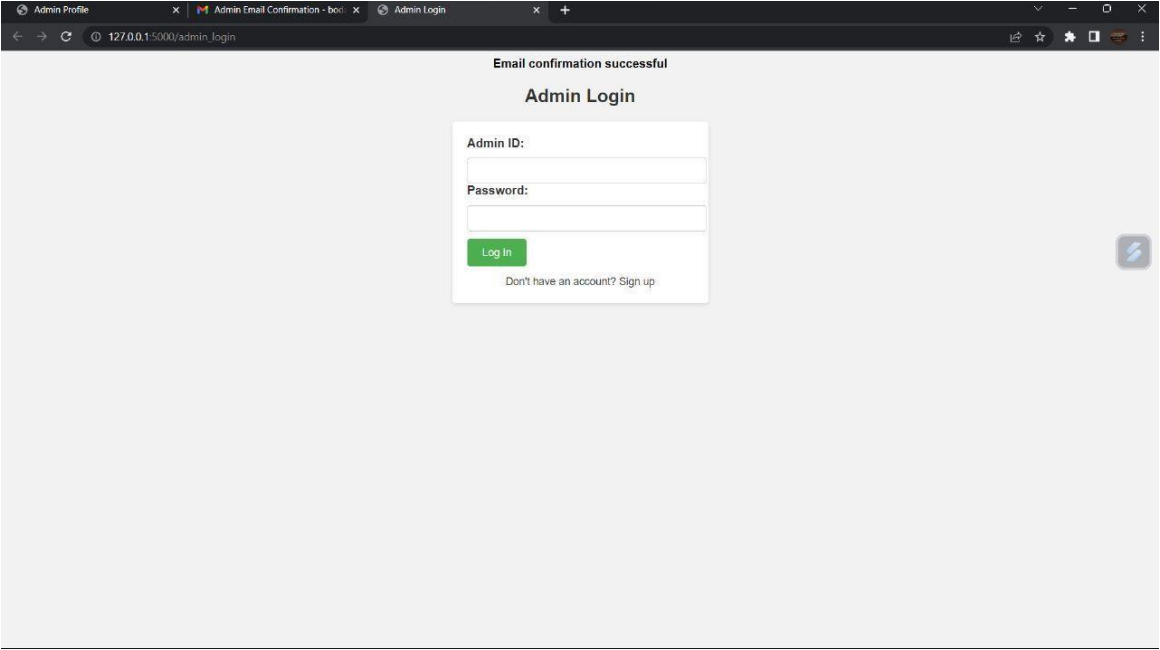
    acic = asyncio.coroutines._is_coroutine # type: ignore[attr-defined]
    return getattr(func, "_is_coroutine", None) is acic
```

CHAPTER 10

OUTPUT SCREENS:

Login Screen: The initial screen where users can enter their credentials to log into the result management system.






Student Sign In

127.0.0.1:5000/student_login

Student Login

Student ID:

Sign In



Student Sign In


127.0.0.1:5000/student_signup

Student Signup

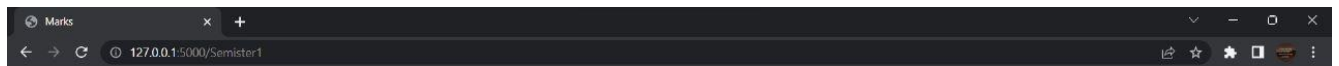
Student Name:

Student ID:

Sign In



[illegible]

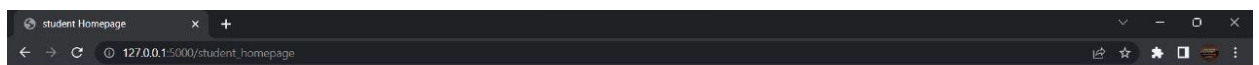


Marks

Bhaskar

Semester 1

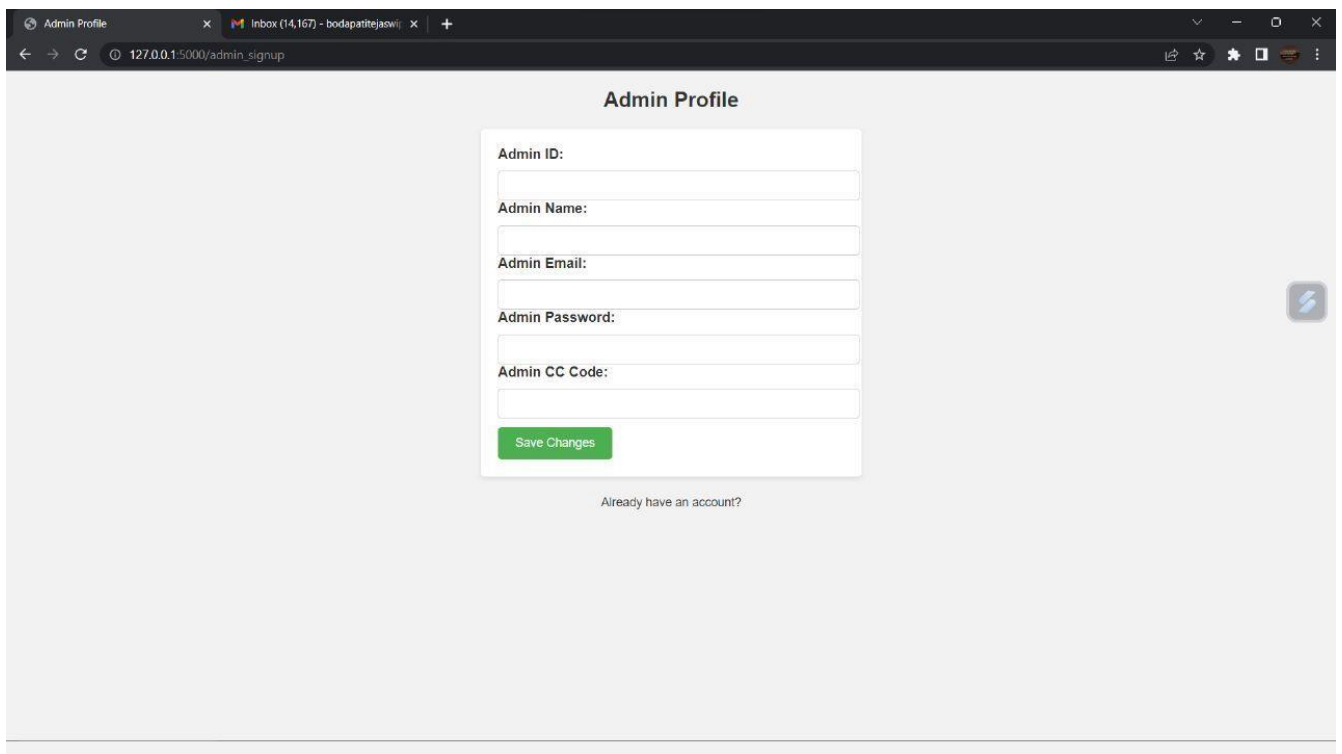
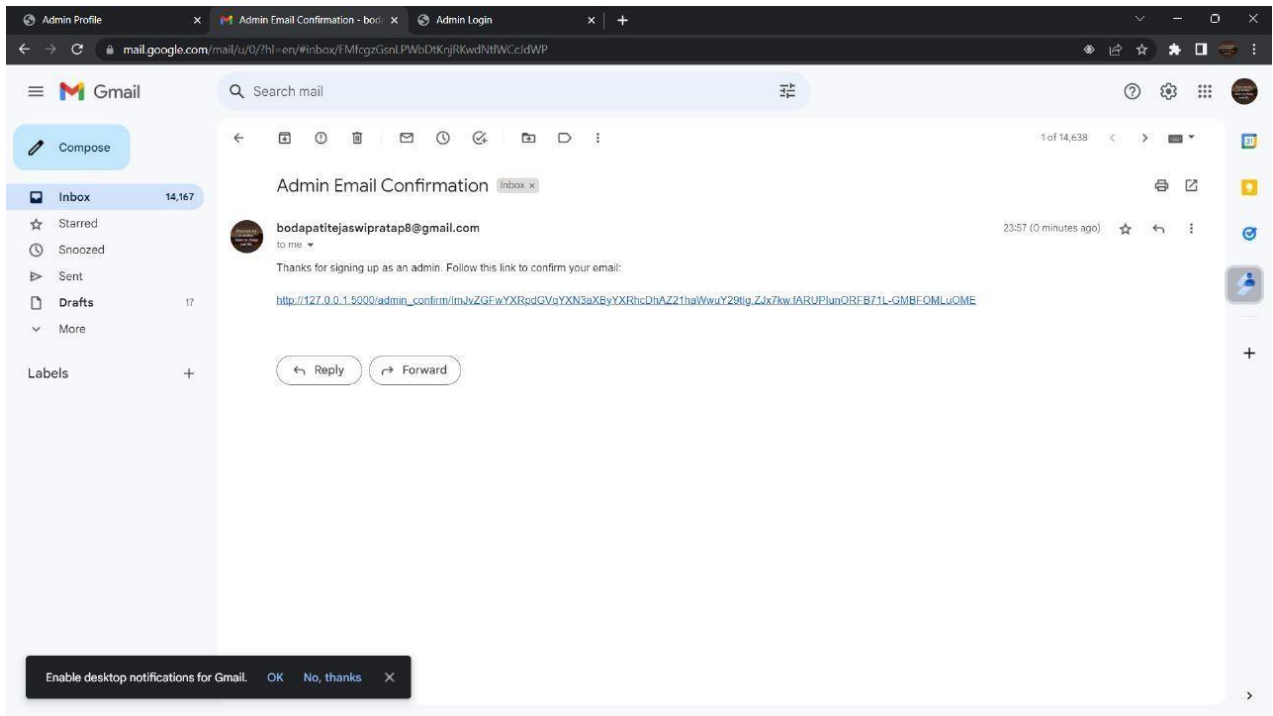
Subject	Marks
Physics	78
maths	88
chemistry	87
english	89
M2	97

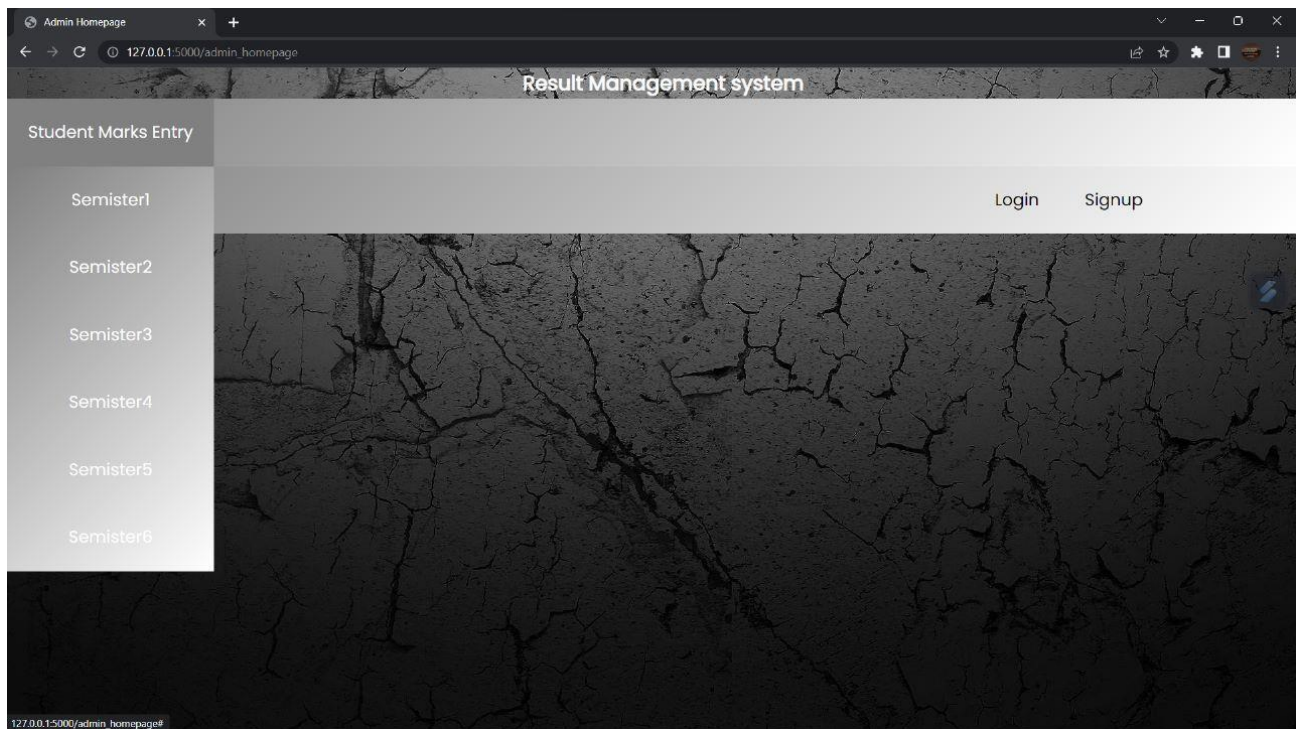


View Results

- [Semester1](#)
- [Semester2](#)
- [Semester3](#)
- [Semester4](#)
- [Semester5](#)
- [Semester6](#)
- [overall_percentage](#)
- [contact_admin](#)
- [student_Login](#)
- [student_Signup](#)
- [student_Logout](#)







CHAPTER 11

TECHNOLOGIES USED

There are many technologies that can be used to implement a result management system using Python. Some of the most common technologies include:

Python: Python is a general-purpose programming language that is well-suited for developing web applications. It is also a popular choice for developing educational software, such as result management systems.

Django: Django is a web framework that is written in Python. It provides a number of features that make it well-suited for developing result management systems, such as a built-in database abstraction layer, a user authentication system, and a template engine.

MySQL: MySQL is a relational database management system (RDBMS) that is commonly used with Python projects. It provides a reliable and scalable way to store and manage the data for a result management system.

Tkinter: Tkinter is a Python GUI toolkit that can be used to create graphical user interfaces (GUIs) for result management systems. It is a simple and easy-to-use toolkit that can be used to create functional and visually appealing GUIs.

Bootstrap: Bootstrap is a CSS framework that can be used to style the GUI of a result management system. It provides a number of pre-made style components that can be used to quickly and easily create a modern and responsive GUI.

In addition to these technologies, there are a number of other technologies that can be used to implement a result management system using Python. The specific technologies that are used will depend on the specific requirements of the system.

Here are some examples of result management systems that have been implemented using Python:

Student Result Management System (SRMS): This system is a desktop application that is used to manage student results. It is built using Python and Tkinter.

Online Result Management System (ORMS): This system is a web application that is used to manage student results. It is built using Python, Django, and MySQL.

Mobile Result Management System (MRMS): This system is a mobile application that is used to manage student results. It is built using Python, Django, and React Native.

These are just a few examples of the many result management systems that have been implemented using Python. The popularity of Python for this purpose is due to its flexibility, power, and ease of use.

CHAPTER 12

SOFTWARE TESTING

12.1 Types of testing

Unit testing

Module testing

Integration testing

Validation testing

Output testing

Unit Testing

- In this online examination system unit testing check login form. Unit testing focuses verification efforts on the smallest unit of the software design, the module.
- Unit testing focuses verification efforts on the smallest unit of the software design, the module. This is also known as “Module Testing”.
- The modules are tested separately.
- This testing was carried out during the programming stage itself. In this testing each module is found to be working satisfactory as regards to the expected output from the module.

Module Testing

This is also known as unit testing. Unit testing focuses on the verification of the smallest unit of software design of the module. In this each module was found to be working satisfactory as per the expected output of the module.

Integration Testing

- In this integration testing the modules are checked one by one up to the end module. Data can be grossed across an interface; one module can have adverse efforts on another.
- Integration testing is to take unit testing for construction of the program structure while at the same time conducting tests to uncover errors associated with the interface.
- The object is to take unit tested modules and build a program structure. All the modules are combined and tested as a whole.
- Here correction is difficult because the isolation of cause is complicated by the vast expense of the entire program.
- Thus, in the integration testing stop, all the errors uncovered are corrected for the text testing steps.

Validation Testing

- At the conclusion of integration testing software is completely assembled as a package, interfacing errors have been uncovered and corrected and a final series of software tests begins, validation test begins.
- Validation tests can be defined in many ways. But the simple definition is that validation succeeds when the software functions in a manner that can be reasonably expected by the examiners.
- After the validation test has been conducted one of two possible conditions exists.
- One is the function or performance characteristics confirm to specifications and are accepted and the other is deviation from specification is uncovered and a deficiency list is created.
- Proposed system under consideration has been tested by using validation testing and found to be working satisfactorily.

Output Testing

- In output testing each test has a different purpose: all the work should verify that all the system elements have been properly integrated and perform allocated functions. No system could be useful if it does not produce the required output.
- Here, output testing is done by checking whether the data being updated through the database is coming in the correct portion and position of the website template.

CONCLUSION

Flexibility: Python is a flexible language that can be used to implement a result management system in a variety of ways. This makes it a good choice for organizations with specific requirements or constraints.

Power: Python is a powerful language that can be used to create complex and sophisticated result management systems. This makes it a good choice for organizations with large or demanding workloads.

Ease of use: Python is an easy-to-learn language that makes it a good choice for organizations with limited development resources.

Community support: Python has a large and active community of developers who can provide support and resources for implementing result management systems.

Cost-effectiveness: Python is a cost-effective language to use for developing result management systems. This is due to the availability of free and open-source resources, as well as the relatively low cost of Python developers.

Overall, Python is a good choice for implementing a result management system. It is flexible, powerful, easy to use, and has a large and active community of developers.

Here are some of the challenges that may be encountered when implementing a result management system using Python:

Security: Python is a general-purpose language that can be used to create malicious code. This makes it important to take steps to secure any result management system that is implemented using Python.

Performance: Python is not as fast as some other languages, such as C++ or Java. This may be a concern for organizations with large or demanding workloads.

Scalability: Python can be scaled to handle large workloads, but it may not be as scalable as some other languages. This is something to consider for organizations that expect to grow their result management system in the future.

Overall, the benefits of implementing a result management system using Python outweigh the challenges.

Python is a flexible, powerful, and easy-to-use language that can be used to create cost-effective result management systems.