

Title Page

Result Tracking System for University Students and Parents

Certification

I hereby declare that this work was done by me and not any third party. Should it be proven otherwise I shall accept the verdict of the Institution.

Ndukwe Peter Ifejikochukwu

2020/241163

Name

Reg No

Signature

Approval

This project titled Result Tracking System for University Students and Parents was carried out by Ndukwe Peter Ifejikochukwu with Registration Number 2020/241163 and has been approved to meet the requirements for the award of a Bachelor of Science Degree (B.Sc) in Computer Science.

.....

Dr. C. N. Asogwa

Project Supervisor

.....

Date

.....

External Examiner

.....

Date

Dedication

This project is dedicated to almighty God and to all Parents and Guardians sponsoring their children through school.

Acknowledgment

I acknowledge the Head of the Department of Computer Science in the University of Nigeria, Nsukka Prof. C.N. Udanor. I sincerely acknowledge and appreciate my project supervisor Dr. C.N. Asogwa for her kind guidance towards the realization of this project. I appreciate and acknowledge the entirety of the Department's Staff. I extend my appreciation to Dr. Greg Anichebe and Mr. Agbo Jonathan who were very helpful during the preparation of this project. I also acknowledge my Family for their continuous support during this time.

Abstract

Parents have no direct access to their children results. They have to wait for their children to send it to them and that is not guaranteed as the children may not want to because of their performance. Students have to see their Exam officers directly anytime they need to check their results. Such process is time consuming and hectic for both the Students and the Exam officers. This project developed a result tracking system to solve these problems. Object Oriented Analysis and Design Methodology was used to analyze and design the system using various UML diagrams. The project was implemented on a Windows 10 operating system using Visual Studio Code as the IDE. MySQL was used to handle the database. HTML, CSS, Python and the Django framework were the languages used to develop the system. The system would be beneficial to Students, Parents or Guardians and the Examination Officer.

Table of Contents

Title Page	1
Certification	2
Approval	3
Dedication	4
Acknowledgement	5
Abstract	6
Table of Contents	7
List of Figures	10
List of Tables	12

Chapter 1: **Introduction**

1.0 Background	13
1.1 Statement of the problem	13
1.2 Aim and Objectives of the Study	14
1.2. 1 Aim	14
1.2. 2 Objectives	14
1.3 Significance of the Problem	14

Chapter 2: **Literature Review**

2.0	Introduction	15
2.1	Theoretical Background	15
2.2	Review of Related Literature	16

Chapter 3: **System Analysis and Design**

3.0	Introduction	19
3.1	Description of the existing system	19
3.2	Analysis of the proposed system	20
3.3	Design of the proposed system	24
3.3.1	Input Design	24
3.3.2	Output Design	26
3.3.3	Database Design	29
3.3.4	System Architecture	31
3.3.5	Data Source	31

Chapter 4: **System Implementation**

4.0	Introduction	33
4.1	Choice of the Development Environment	33
4.2	Implementation Architecture	33
4.3	Software Testing	34

4.4	Documentation	40
4.4.1	User Manual	40
4.4.2	Source code listing	40
Chapter 5: Summary and Conclusion		
5.0	Summary	42
5.1	Conclusion	42
5.2	Recommendations	43
	References	44
	Appendices	46

List of Figures

Figure 3.1	Use Case diagram	21
Figure 3.2	Class diagram	22
Figure 3.3	Activity diagram	23
Figure 3.4	Sign up page input design	24
Figure 3.5	Login page input design	25
Figure 3.6	Student search page input design	25
Figure 3.7	Upload result page input design	26
Figure 3.8	Student home page output design	26
Figure 3.9	Exam officer home page output design	27
Figure 3.10	Student search results output design	27
Figure 3.11	Result upload success page output design	28
Figure 3.12	Select semester result page output design	28
Figure 3.13	View semester result page output design	29
Figure 3.14	System Architecture design	32
Figure 4.1	Implementation Architecture diagram	33
Figure 4.2	Sign up page implementation	34
Figure 4.3	Login page implementation	35
Figure 4.4	Search student page implementation	35

Figure 4.5	Upload result page implementation	36
Figure 4.6	Student home page implementation	36
Figure 4.7	Exam officer home page implementation	37
Figure 4.8	Student search result page implementation	37
Figure 4.9	Result upload successful page implementation	38
Figure 4.10	Select semester result page implementation	38
Figure 4.11	View semester result page implementation	39
Figure 4.12	Update result page implementation	39

List of Tables

Table 3.1	Exam Officer	29
Table 3.2	Course	30
Table 3.3	Student	30
Table 3.4	Result	30
Table 3.5	Parent	31

Chapter One

Introduction

1.0 Background to the study

The traditional approach for parents or guardians and university students to see their results is to either ask the lecturers concerned or exam officers directly or come to the school premises directly to request for it. In most occasions students may trick their parents or guardians and feed them with lies such as, that the result is not published in order to keep them from seeing their academic performance. In a situation like that, parents may not know of the academic performance of their children until it is too late; may be when such students are expected to graduate. This project addresses this need by developing a web application to bridge that gap, giving parents and guardians direct access to their wards results and academic performance as soon as they are ready. A number of works have been done to address the problem of results processing and how they can be accessed. In work [6] a result management system is developed that allows students to send their results to their Parents as pdf files. The limitation to that system lies in the fact that Students still hold the power to send or not to send to the results to their Parents. This project aims to solve that limitation by giving the Parents direct access to results without any need or interference from the students.

1.1 Statement of the Problem

1. Parents and Guardians may have to go physically to the university to ask lecturers or administrators for their children's results.
2. Parents and Guardians have no direct access to their children's results.
3. Students may send their parents forged results.
4. Students may lie that their results have not been published.

1.2 Aim and Objectives

1.2.1 Aim

The aim of the Project is to develop a Result Tracking System for University Students and Parents.

1.2.2 Objectives

1. Create an interface to enable Parents, Guardians and Students to login and access the system.
2. Design the result tracking system.
3. Provide an authenticated admin interface for uploading results.
4. Develop the tracking system with a subsystem (link) to enable the system send emails to parents anytime a new result is published.

1.3 Significance of the Problem

This system will be of benefit to Parents, Guardians and Students.

1. Parents and Guardians will be able to monitor their wards academic performance, ensuring the school fees paid for the students are not wasted.
2. As parents and guardians are able to see their wards results as they please, it will also provide avenue for counselling a student not doing well on time.
3. Students will be able to see their published results online without going physically to their departments.
4. Students and Parents can access results independently without having to see the Examination Officers. This will reduce the pressure on their work.

Chapter Two

Literature Review

2.0 Introduction

This chapter will look at the theoretical background with emphasis on the concepts and technologies used to build the project. It also reviews related literature on the project topic to gain insight on what others have done, how they did it and the limitations encountered so as to find the gap in literature and fill it.

2.1 Theoretical Background

Student tracking systems have been discussed, researched and developed in the past couple of years. It may appear in papers with names such as Student Result Management System (SRMS), Student Information Systems (SIS), School Management Systems and many more. A Student Result Management System helps schools manage, store and process results. The primary function is to streamline the result collation process. Hence, schools that use it can distribute accurate and timely information directly to students and parents [6]. The result tracking system to be built will be in form of a website application. The application will be developed using certain web development technologies. Web development covers the building of static sites, dynamic sites, blogs and complex web application systems. Web development can be broken into the frontend and backend parts respectively. The frontend describes the interface the user interacts with and everything that can be viewed from a screen. It may be referred to as the client side. The backend describes the part of the website that handles all processing of data as regards the website and gives responses to requests from the user. It is the part that handles server side operations. Frontend technologies that are used to build websites include HTML5, CSS, JavaScript, React, Next.js, Angular, Vue and many more. Backend technologies that are used to build

functional websites include PHP, Python, Java, Django, Flask, Spring Boot and many more. SQL is used in handling databases. The understanding of domain, hosting and version control is also important in the deployment phase of web development. These technologies among others can be used for the building of functional web applications. This project would be developed using HTML, CSS, JavaScript, Django and MySQL. HTML, CSS, and JavaScript will be used to develop the frontend of the application. These technologies for years have been the foundation for building user interfaces for web applications. HTML is used to create the layout and structure of the website. Cascading Style Sheets (CSS) is used to style the website and give it a good feel and design enabling the user to interact with the application seamlessly. JavaScript is a language used to make the website responsive. It will allow for various responses to user actions such as clicking, scrolling, hovering and more. Such responses would make the application responsive and interactive. Django is framework built on the Python programming language and it was chosen for the project because of the powerful features that comes with it for building web applications such as an authentication library, a prebuilt admin interface, Emails handling library, its Database API and templating engine. These features make solving the problems statements of the project very possible and easy. MySQL will be used to manage the database of the project. MySQL is strong and has good record for working on related projects. It also has good support and interactivity with Django.

2.2 Review of Related Literature

In work [1] the process of developing a result management system is presented with the department of Electrical/Electronic and Computer Engineering University of Uyo used as a case study. The system made use of the participatory software development methodology. It makes provision for three types of users namely the system administrator, staff and student. Each type of user can perform a set of roles

in the system. The student can register courses, view and print results. The staff can be the exam officer or a course lecturer. The exam officer can generate transcripts, view and print results. The course lecturer can record tests, exam scores and manage courses. The system administrator manages semesters and sessions. All users are required to register before access is given. The system was developed using HTML, CSS, PHP and MYSQL.

Work [2] refers to the design and implementation of a result processing and transcript system using the University of Ibadan as a case study. The system really takes into cognizance the process flow for student results to be compiled and approved in a university setting. It tries to reduce the burden on administrative staff for the generation of transcripts. With the system transcripts can be generated much easier without having to manually compile each student's result. The system was designed following the experimental approach and it adopted the use of the Software Development Life Cycle. The system makes use of forms to take in user data such as grades and stores them in a database. The data can be retrieved from the database and processed to produce valuable reports. The system was developed using PHP and MySQL and makes use of computer MAC address for authentication.

The objective in work [3] was to design and implement an examination result processing system using the department of Computer Science of the Kano University of Science and Technology, Wudil. The system allows students to login online to view their examination results providing relief for those outside school vicinity as at the release time of the results. The system was built using Microsoft Visual Studio, C#, ASP.NET and SQL_Server_2008 for generating database tables. An administrator in the system is able to create accounts for students, head of department, exams officer, level coordinator and lecturers. The Head of department has right to assign the exam officer, register courses, staff, students,

and assign courses to staff. The Exams Officer has the right to process and view student results. The level Coordinators have the right to process student transcripts and enter student scores. Lecturers can view student scores.

The aim of the work in [4] is to model the process of processing and managing students result in a result processing system. The system was implemented using PHP and MySQL. The system provides a robust database that is able to generate various reports in PDF format. The system used the incremental development methodology and follows the Software System Life Cycle (SSLC). The frontend of the system was implemented using HTML, CSS, JavaScript and Macromedia Dreamweaver 8 IDE. This system provides a more simplified design of a result processing system that is easier to make use of and understand.

The purpose of Student Result Management System is to create a better method for storing the result of students in a computerized form so that the result can be stored for a longer period with easy accessing and manipulation of the same [5]. The objective in work [5] is to build a Student result management system to reduce the manual work in managing students result in a college. The system manages the students results, provides an improved process for editing, adding, deleting and updating of records, integrates all the records of a semester. Only the administrator is granted access to this system.

Many efforts have been made towards the development of result management systems and past researchers and contributors must be commended but these works do not seem to make provision for easy and direct access to those results by Students Parents and Guardians. This is where this project's system comes in providing an online and remote access to upload and view students' results by both Parents and Students. It will develop a tracking system that allows parents know anytime a new result is published.

Chapter Three

System Analysis and Design

3.0 Introduction

The chapter discusses the steps taken to understand the workings of the existing system and the problems in the existing system. The design steps and the design tools for the system are presented. The chapter also covers the research methodology adopted in the work.

3.1 Description of the existing system

The process students would take in order to check their results include the following:

- Student goes to the Exam officer's Office to request for his or her result.
- Exam officer tells the student the grades of the result available and allows the student to copy it.
- Student may or may not send the received result to his or her Parent or Guardian.

After careful analysis the following problems were identified in the system:

- Parents and Guardians may have to go physically to the university to ask lecturers or administrators for their children's results.
- Parents and Guardians have no direct access to their children's results.
- Students may send their parents forged results.
- Students may lie that their results have not been published.

3.2 Analysis of the proposed system

The methodology used to work in this project is the Object Oriented Analysis and Design. Object-oriented analysis and design (OOAD) is a technical approach for analyzing and designing an application, system, or business by applying object-oriented programming, as well as using visual modeling throughout the software development process to guide stakeholder communication and product quality [7]. Grady Booch has defined OOA as, “Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain” [8]. According to [8] the primary tasks involved in object oriented analysis include identifying the objects, organizing the objects by creating objects model diagram, defining the object attributes, defining the object actions and describing how the objects interact. The system would be analyzed using various UML diagrams.

In work [12], a use case diagram is described as a visual representation of user interactions with a system, showing various use cases and different actors involved in the system. In work [13], it is noted that a use case diagram captures the functional requirements of a system and helps define how external actors interact with system functionalities.

The use case diagram below clearly shows what each actor has the capability and right to do. Students and Parents can only view results. Exam Officer can view results, add courses, upload, update, and delete results.



Figure 3.1: Use Case Diagram

Through OOAD the system can be broken down into classes that represent its components in a simplistic way. In work [10], a class diagram is defined as a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. In work [11], it is stated that class diagrams are used to visually

describe the structure of a system by representing its classes, attributes, operations, and the relationships between objects.

Upon analysis the classes that make up the system are:

- Student
- Parent
- Exam Officer
- Course
- Result

These classes are represented in the class diagram below that shows clearly the class attributes and actions. The attributes refers to the internals of the objects and actions refers to its behavior [8].

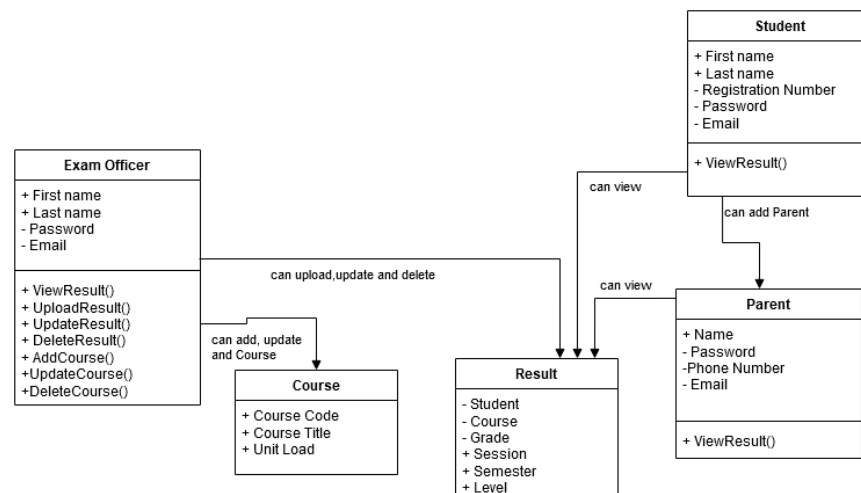


Figure 3.2: Class Diagram

In work [14], an activity diagram is defined as a type of flowchart that models the workflow of a system, describing the sequence of actions and decision points. In work [15], it is stated that activity diagrams illustrate the dynamic aspects of a system by representing processes, workflows, and the interactions between system

components. The activity diagram below shows the activities and process flow involved when using the system. The view of what the student, parent and exam officer can do is expressed clearly.

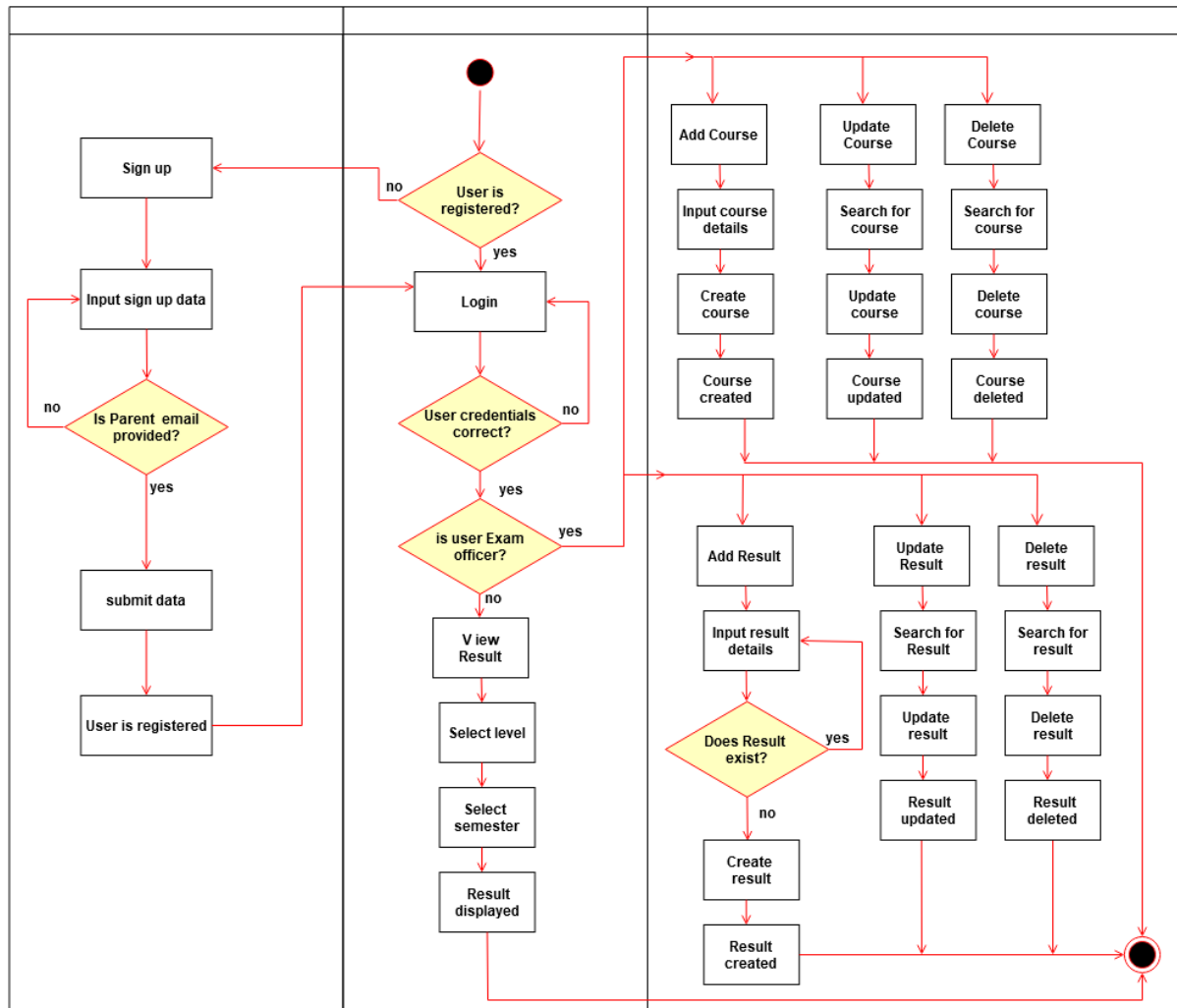


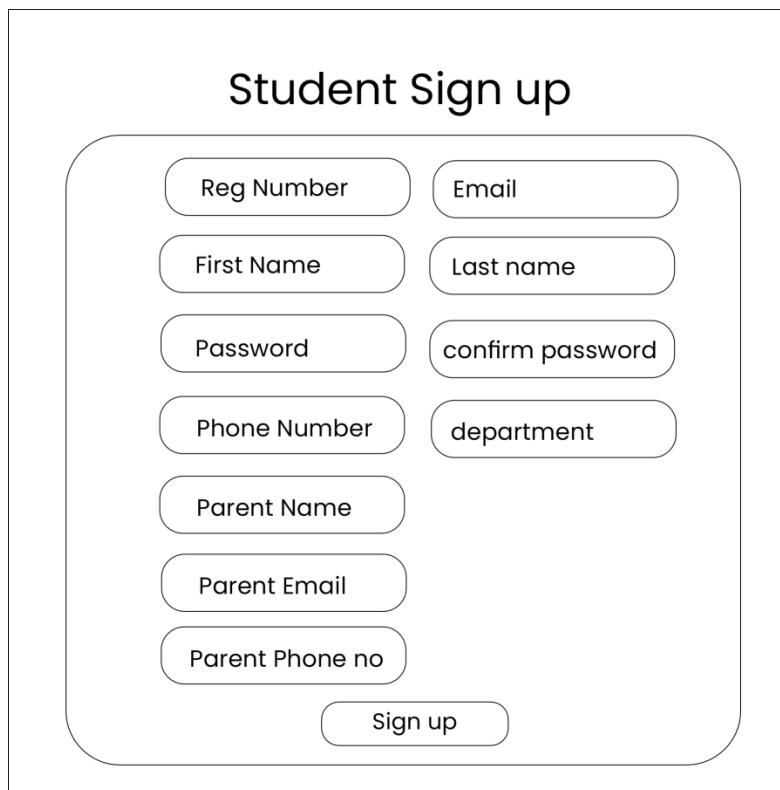
Figure 3.3: Activity Diagram

3.3 Design of the proposed system

3.3.1 Input Design

Find below input designs that shows the data fields required for the system to present an output.

Students are able to register on the system by inputting the details in the input design below.

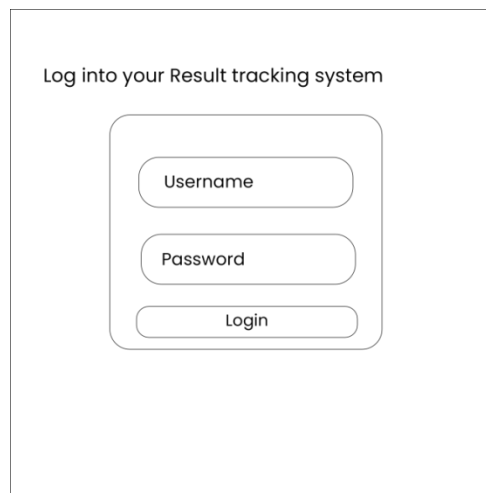


The image shows a 'Student Sign up' form. It is a rounded rectangle with a light gray border. Inside, there are several input fields represented by rounded rectangles with light gray borders. The fields are arranged in two columns. The left column contains: 'Reg Number', 'First Name', 'Password', 'Phone Number', 'Parent Name', 'Parent Email', and 'Parent Phone no'. The right column contains: 'Email', 'Last name', 'confirm password', and 'department'. At the bottom center of the form is a 'Sign up' button. The title 'Student Sign up' is centered at the top of the form.

Student Sign up	
Reg Number	Email
First Name	Last name
Password	confirm password
Phone Number	department
Parent Name	
Parent Email	
Parent Phone no	
Sign up	

Figure 3.4: Sign up page input design

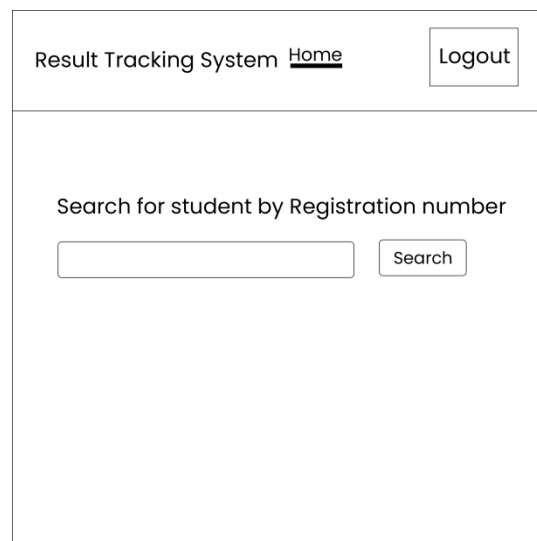
Below is the input design for students, exam officer, and parents to log into the system.



Log into your Result tracking system

Figure 3.5: Login page input design

Below is the input design for the search page where the exam officer can search for students in order to upload their results.



Result Tracking System [Home](#)

Search for student by Registration number

Figure 3.6: Student search page input design

Below is the input design for the upload page where the exam officer can upload results.

Result Tracking System [Home](#) [Logout](#)

Upload Student Result

Student Reg NO
Student Name

Course:
 Level:
 Exam score:
 CA score:
 Grade:
 Session:
 Semester:

[Upload](#)

Figure 3.7: Upload result page input design

3.3.2 Output Design

When input is given to the system it produces an output page. Find below all the output designs to the input designs of the system.

Find below the output design page for when a student logs into the system.

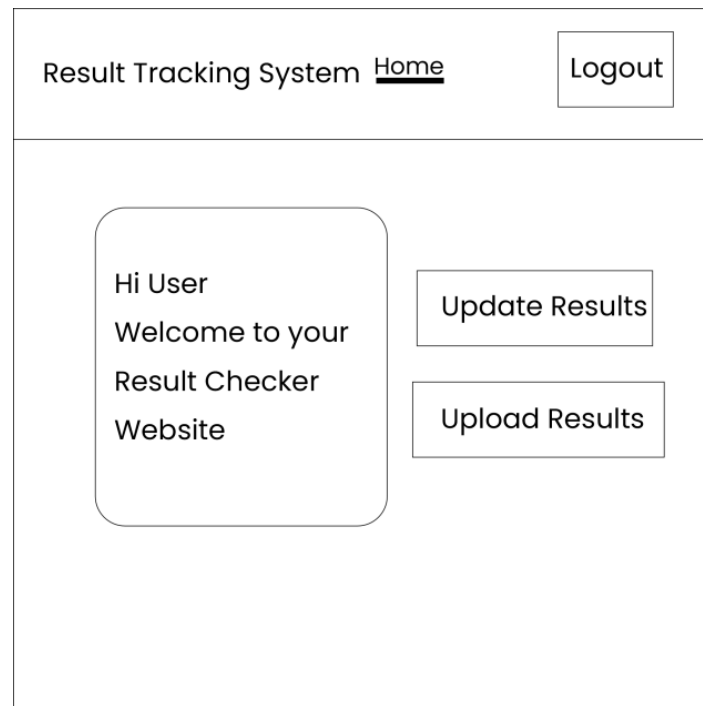
Result Tracking System [Home](#) [Logout](#)

Hi User
 Welcome to your
 Result Checker Website

[View Results now](#)

Figure 3.8: Student home page output design

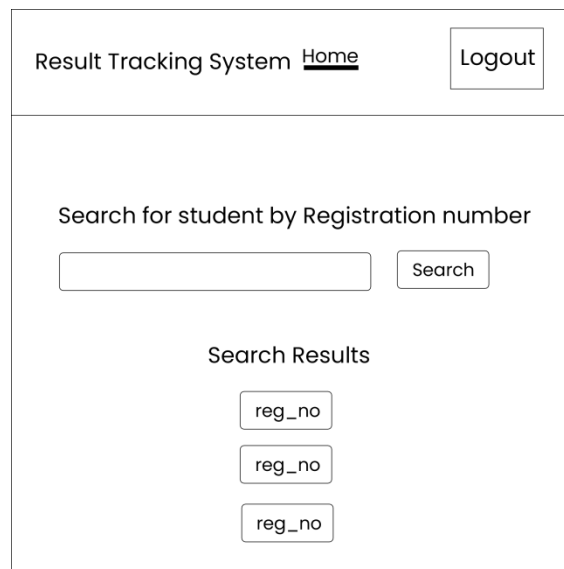
Find below the output design page for when the exam officer logs into the system.



The image shows a web page layout for an exam officer. At the top, there is a header bar containing the text "Result Tracking System" followed by a link labeled "Home" with a red underline, and a "Logout" button on the right. Below the header, the main content area is divided into two sections. On the left, a rounded rectangular box contains the text "Hi User", "Welcome to your", "Result Checker", and "Website". On the right, there are two rectangular buttons stacked vertically, labeled "Update Results" and "Upload Results".

Figure 3.9: Exam officer home page output design

Find below the output design page for when the exam officer searches for a student.



The image shows a web page layout for displaying student search results. At the top, there is a header bar containing the text "Result Tracking System" followed by a link labeled "Home" with a red underline, and a "Logout" button on the right. Below the header, the main content area contains a search section. It starts with the text "Search for student by Registration number" above a text input field and a "Search" button. Below this, the text "Search Results" is displayed. Underneath, there are three rectangular boxes, each containing the text "reg_no", representing the search results.

Figure 3.10: Student search results page output design

Find below the output design page for when the exam officer uploads a student result.

Result Tracking System [Home](#) [Logout](#)

Result upload succesful

Search for student by Registration number

[Search](#)

Figure 3.11: Result upload success page output design

Find below the output design page for when students and parents want to view results.

Result Tracking System [Home](#) [Logout](#)

Select result to view

Student Reg NO
Student Name

Level:	Semester:	
100	1st	2nd
200	1st	2nd
300	1st	2nd
400	1st	2nd

Figure 3.12: Select semester result page output design

Find below the output design page for when students select the semester of the result they want to view.

Result Tracking System [Home](#) Logout

Result for chosen level and semester

Student Reg NO
Student Name

Course code	Course Title	Unit load	Grade
Course code 1	Course Title 1	Unit load 1	Grade 1
Course code 2	Course Title 2	Unit load 2	Grade 2
Course code 3	Course Title 3	Unit load 3	Grade 3

Figure 3.13: View result page output design

3.3.3 Database Design

The classes mentioned in figure 3.2 each is an entity that will be represented as a table in a database. This system would be designed using the MySQL database management system. Find below the database tables that would be used to implement the system.

Table 3.1: ExamOfficer

This table defines all the attributes the Exam Officer can have.

Field	Data type	Size	Null	Description	Action	Extra
Exam officer id	Int	11	No	Unique exam officer id	Primary Key	Auto increment
First Name	Varchar	20	No	Exam officer's first name		
Last Name	Varchar	20	No	Exam officer's last name		
Password	Varchar	30	No	Exam officer's password		
Email	Email	40	No	Exam officer's email		

Table 3.2: Course

This table defines all the attributes a Course can have.

Field	Data type	Size	Null	Description	Action	Extra
Course id	Int	11	No	Unique course id	Primary Key	Auto increment
Course code	Varchar	7	No	Code for the course		
Course title	Varchar	100	No	Title of the course		
Unit load	Int	2	No	Unit_load of the course		

Table 3.3: Student

This table defines all the attributes a Student can have

Field	Data type	Size	Null	Description	Action	Extra
Student Id	Int	11	No	Unique Student Id	Primary Key	Auto increment
First Name	Varchar	20	No	Student first name		
Last Name	Varchar	20	No	Student last name		
Registration No	Varchar	11	No	Student registration no		
Password	Varchar	30	No	Student password		
Email	Email	40	No	Student email		

Table 3.4: Result

This table defines all the attributes a Student's result can have.

Field	Data type	Size	Null	Description	Action	Extra
Result id	Int	11	No	Unique result Id	Primary Key	Auto increment
Student id	Int	11	No	Unique Student Id	Foreign Key	Auto increment
Course id	Int	11	No	Unique course Id	Foreign Key	Auto increment
Grade	Varchar	1	No	Grade for the result		
Session	Varchar	9	No	Session of the result		
Semester	Varchar	11	No	Semester of the result		
Level	Varchar	3	No	Level result is for		

Table 3.5: Parent

This table defines all the attributes a Parent can have.

Field	Data type	Size	Null	Description	Action	Extra
Parent id	Int	12	No	Unique Parent id	Primary Key	Auto increment
Student Id	Int	11	No	Unique Student Id	Foreign Key	Auto increment
Name	Varchar	20	No	Parent name		
Password	Varchar	30	No	Parent password		
Phone number	Int	13	Yes	Parent phone number		
Email	Email	40	No	Parent email		

3.3.4 System Architecture

The system design will follow the 3 tier architecture. Three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where application data is stored and managed [9]. The presentation tier represents the user interface where data is collected from the user as input and displayed for the user as output. It will be developed using HTML and CSS. The application or middle tier connects the presentation and data tier. The application tier is where data from the presentation tier is processed following a set of rules as defined from the requirements specifications. It will be developed using python. The data tier is where the data and information the system works with is stored.

3.3.5 Data Source

The data source used for the system will be the results of students in the department of Computer Science, University of Nigeria, Nsukka.

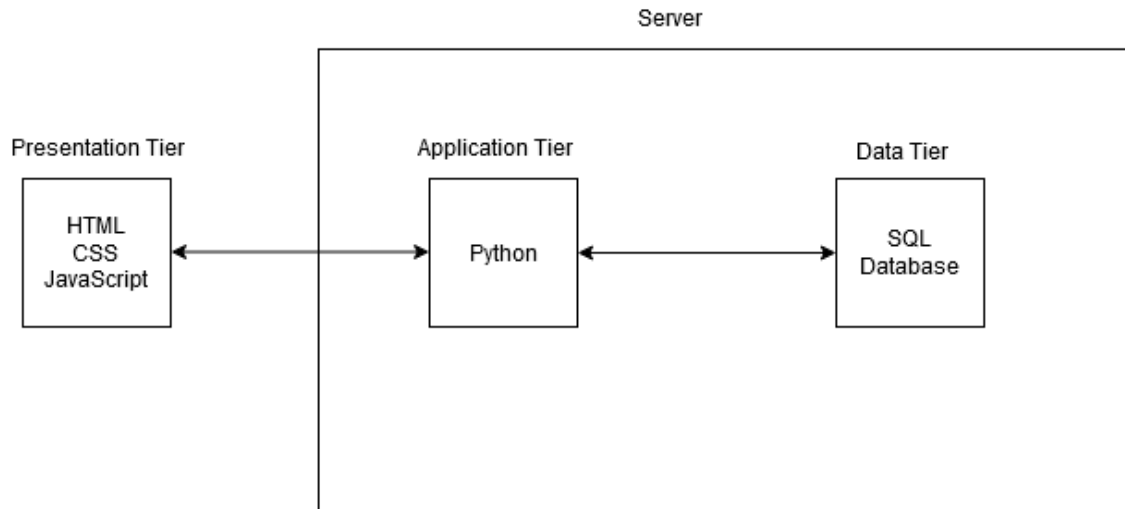


Figure 3.14: System Architecture design

Chapter Four

System Implementation

4.0 Introduction

This chapter will discuss the implementation of the result tracking system following the analysis and design defined in the last chapter.

4.1 Choice of Development Environment

The system will be built on a computer with windows 10 operating system and a 4 gigabyte RAM. It will be built using the Visual Studio Code IDE. The system will be implemented using HTML and CSS for the presentation tier, Python and Django framework for the application logic, and MySQL for the database.

4.2 Implementation Architecture

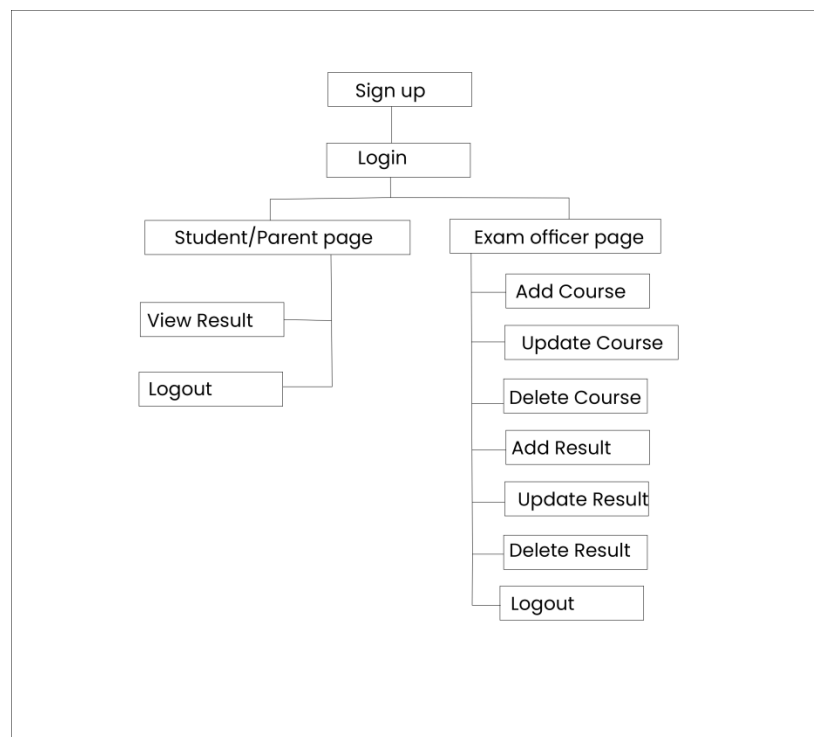
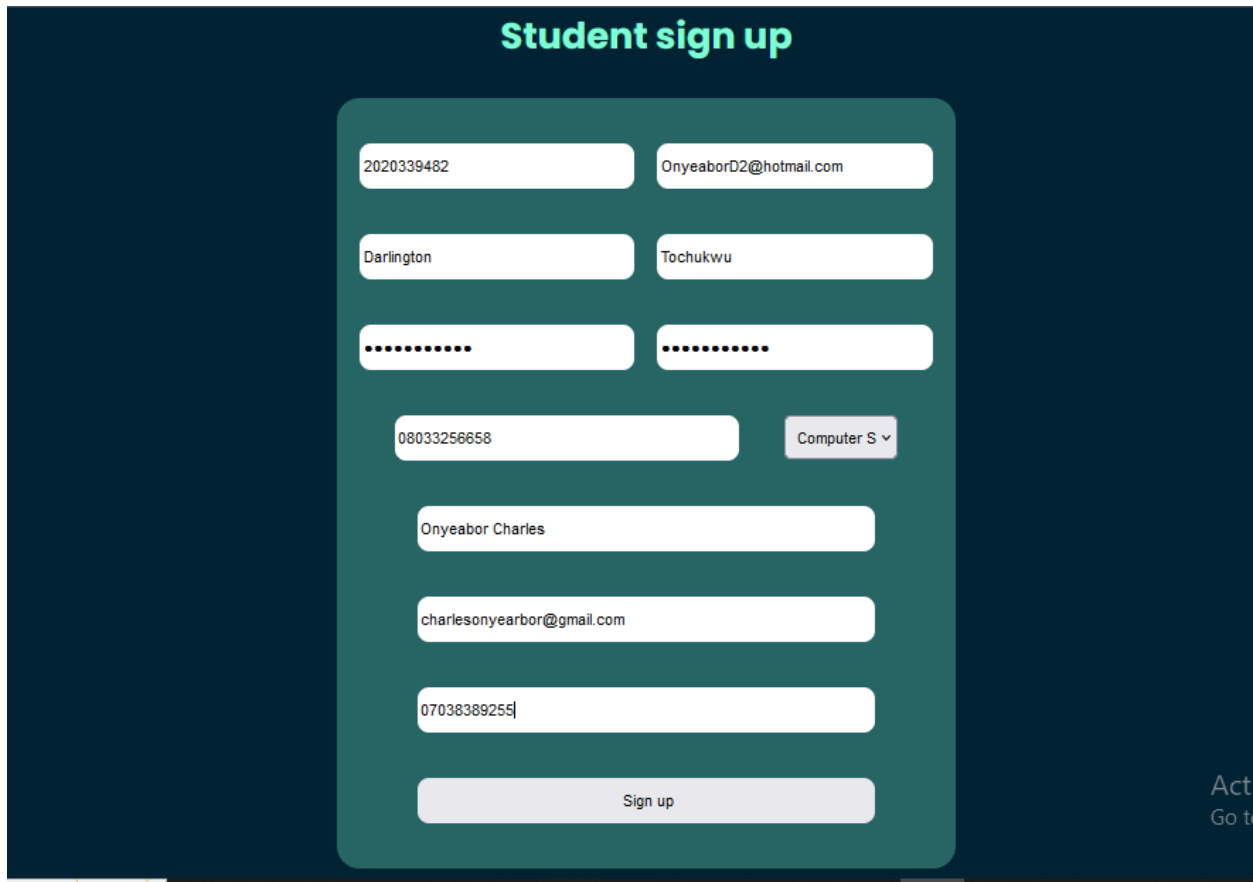


Figure 4.1: Implementation Architecture Diagram

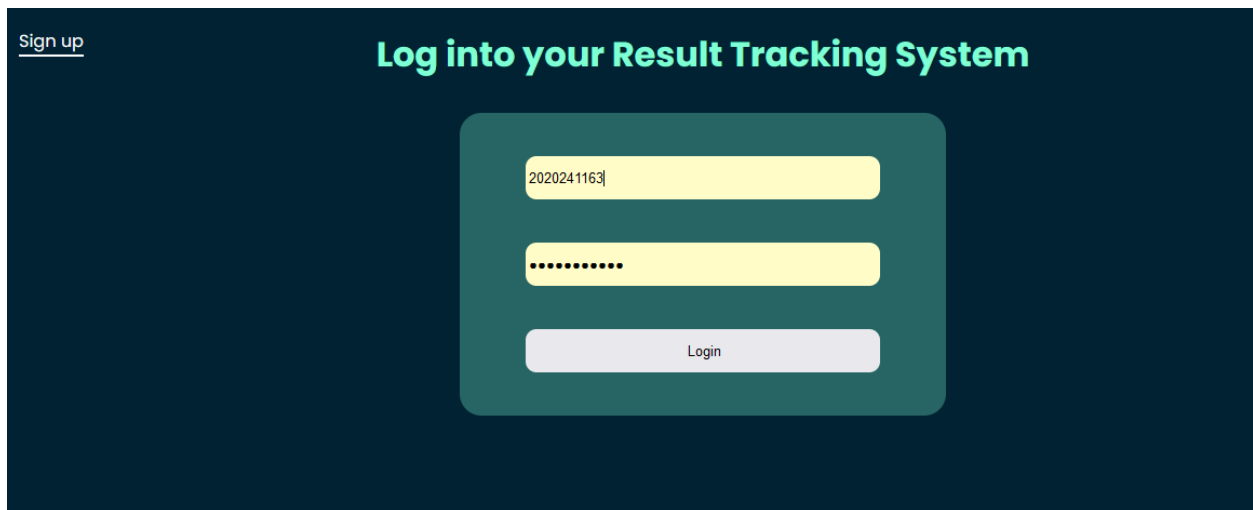
4.3 Software Testing

A unit test is a way of testing the smallest piece of code that can be logically isolated in a software application. Integration testing is the process of testing the interface between multiple software units or modules in order to expose faults in them and solve it. The system was duly tested and is works properly on the localhost. Find below the screenshots of the implemented system.



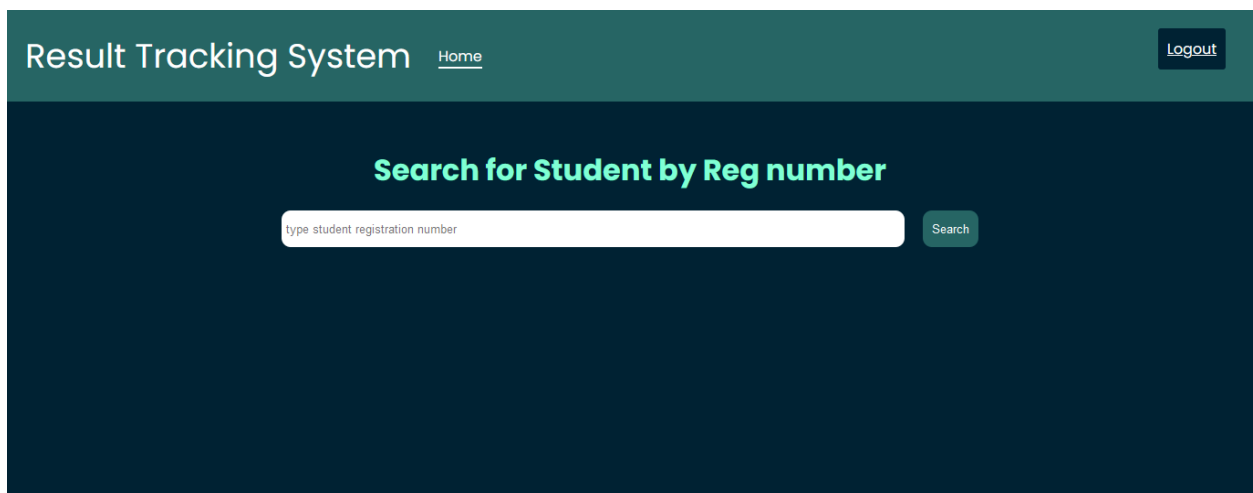
The screenshot displays a 'Student sign up' form on a dark blue background. The form is a light green rounded rectangle containing several input fields. At the top, the title 'Student sign up' is written in green. The form fields are arranged in two columns. The first row contains a text field with '2020339482' and a text field with 'OnyeaborD2@hotmail.com'. The second row contains a text field with 'Darlington' and a text field with 'Tochukwu'. The third row contains two text fields, both filled with ten dots. The fourth row contains a text field with '08033256658' and a dropdown menu showing 'Computer S' with a downward arrow. The fifth row contains a text field with 'Onyeabor Charles'. The sixth row contains a text field with 'charlesonyeabor@gmail.com'. The seventh row contains a text field with '07038389255'. At the bottom of the form is a light blue button labeled 'Sign up'. On the right side of the dark blue background, there is partially visible text: 'Act' and 'Go t'.

Figure 4.2: The user is requested to provide the listed details in order to register



The image shows a login page for a 'Result Tracking System'. The background is dark blue. In the top left corner, there is a link 'Sign up' in white. The main heading 'Log into your Result Tracking System' is centered at the top in a light green font. Below the heading is a light green rounded rectangle containing three input fields: the first contains the text '2020241163', the second contains ten dots representing a password, and the third is a light blue button labeled 'Login'.

Figure 4.3: The user is requested to input his/her credentials to login to the system.



The image shows a search page for the 'Result Tracking System'. The top header is light green and contains the text 'Result Tracking System' on the left, a link 'Home' in the center, and a dark blue button labeled 'Logout' on the right. The main area has a dark blue background with the heading 'Search for Student by Reg number' in light green. Below this heading is a white search bar with the placeholder text 'type student registration number' and a light green button labeled 'Search' to its right.

Figure 4.4: The Exam officer is requested to search for students via registration numbers in order to upload their results.

The screenshot shows a web interface for a 'Result Tracking System'. At the top, there is a header with the system name and a 'Home' link. A 'Logout' button is in the top right corner. The main heading is 'Upload Student Result'. Below this, a form contains the following fields: 'Registration Number' (2020/247852), 'Name' (Bright Chris), 'Course' (STA 205), 'Level' (200), 'Exam Score' (35), 'CA Score' (20), 'Grade' (C), 'Session' (2023/2024), and 'Semester' (1st). An 'upload' button is at the bottom of the form. A Windows watermark is visible in the bottom right corner.

Result Tracking System [Home](#) [Logout](#)

Upload Student Result

Registration Number: 2020/247852
Name: Bright Chris

Course: STA 205
Level: 200
Exam Score: 35
CA Score: 20
Grade: C
Session: 2023/2024
Semester: 1st

upload

Activate Windows
Go to Settings to activate Windows.

Figure 4.5: The Exam officer is requested to input the result details of the searched student.

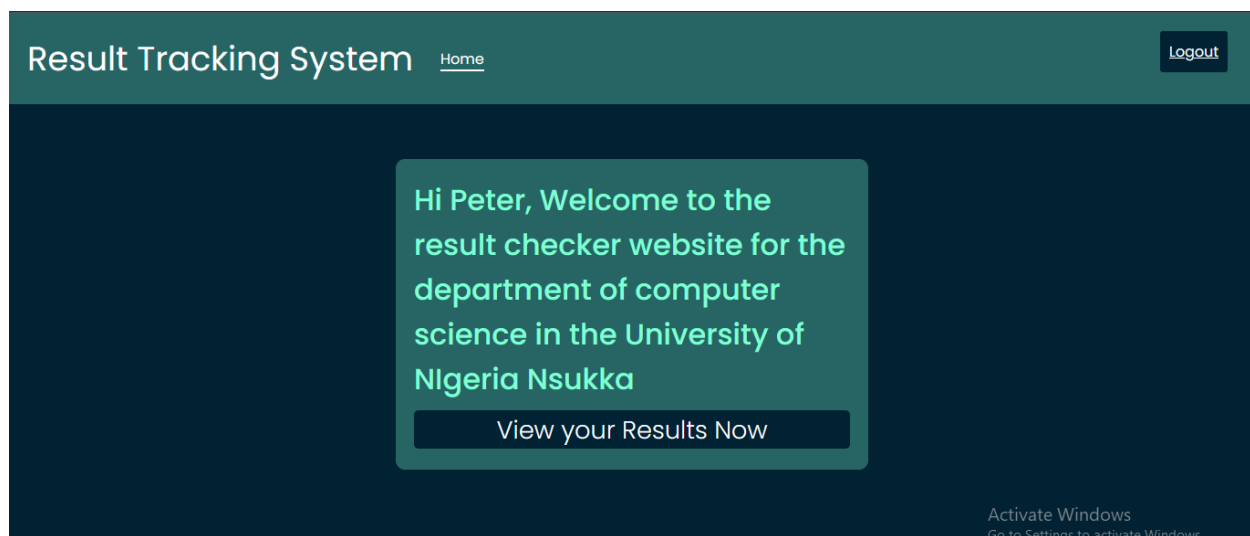


Figure 4.6: This is the output home page a student or Parent sees when logged in.

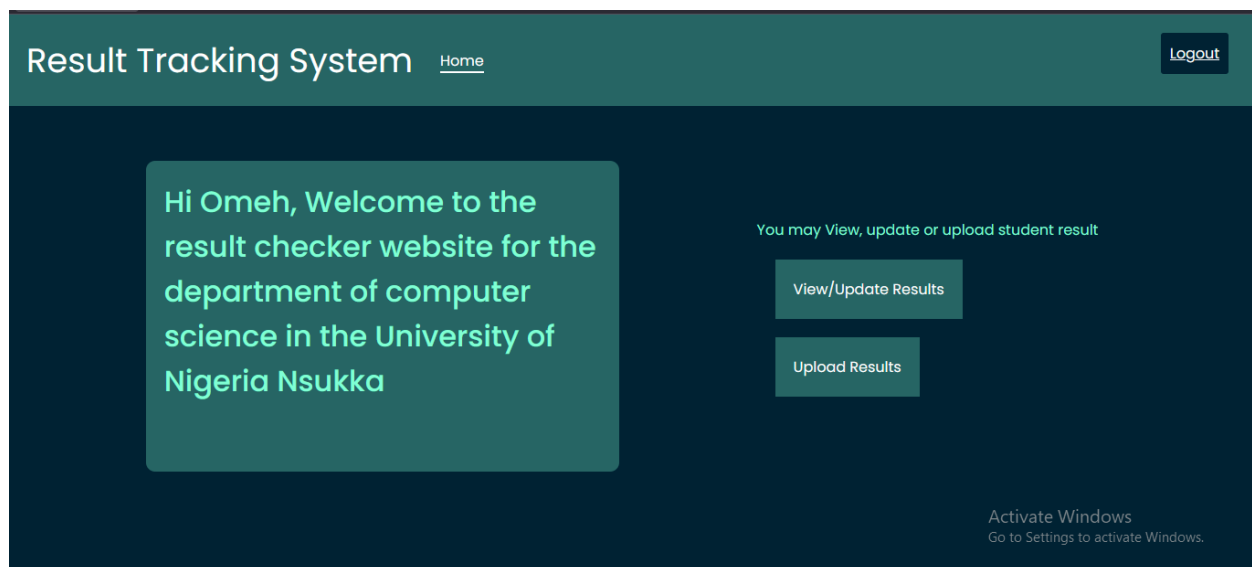


Figure 4.7: This is the output home page the Exam officer sees when logged in.

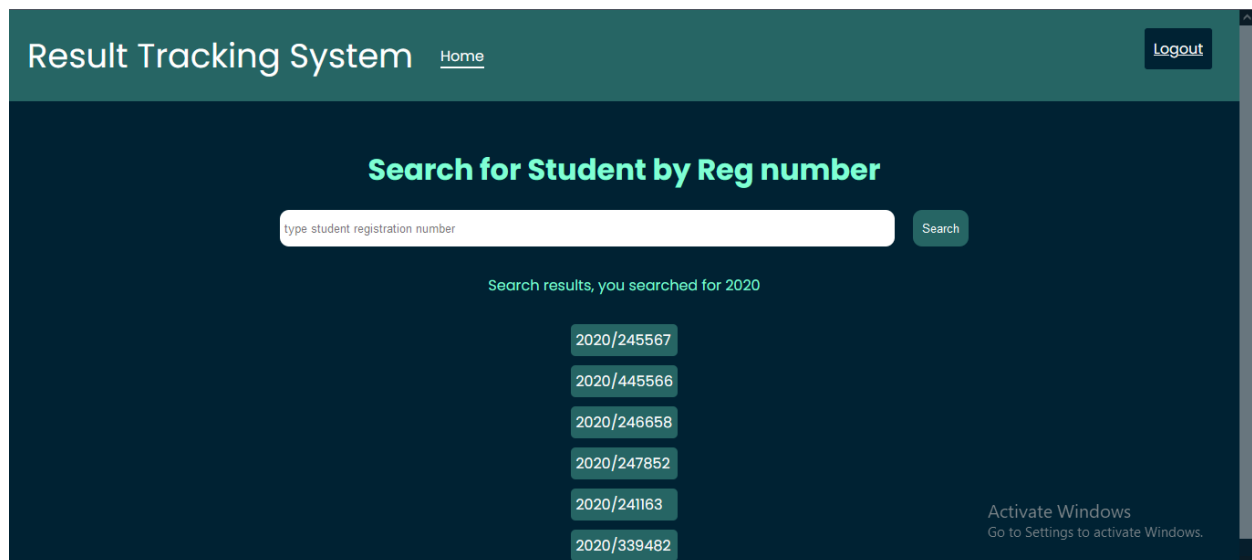


Figure 4.8: This is the output search results page when the Exam officer searches for a student.

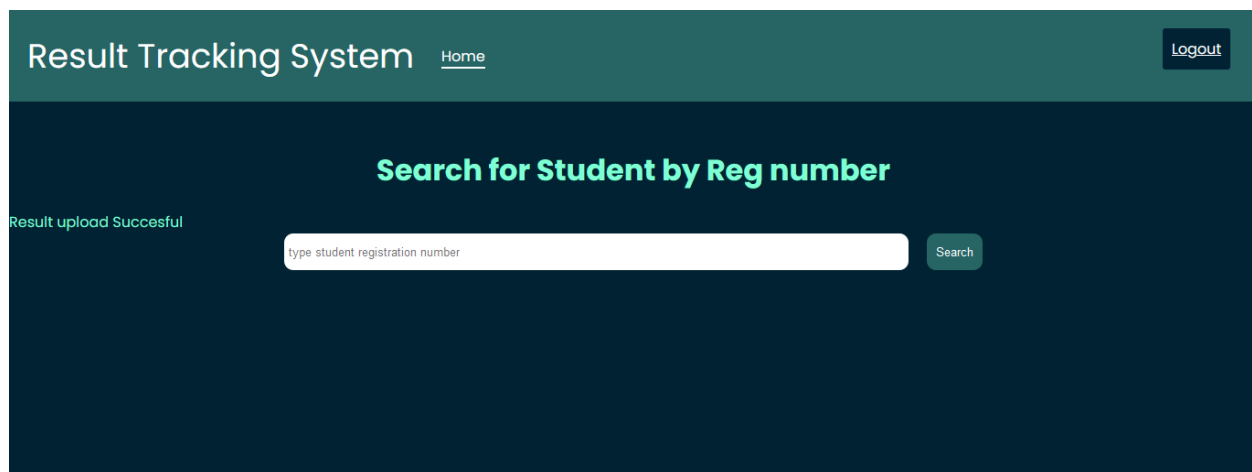


Figure 4.9: This is the output page that shows after the exam officer successfully uploads a student's result.

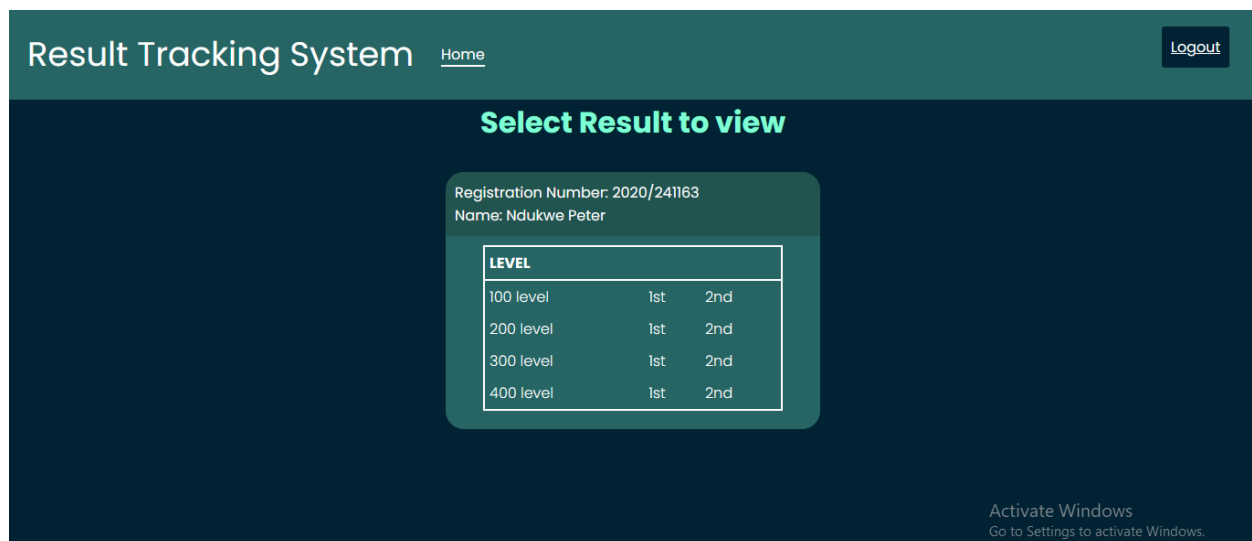


Figure 4.10: This is the output page that shows when a Student or Parent clicks the button to view results.

Result Tracking System
Home
Logout

Result for 100 level

Registration Number: 2020/241163
Name: Ndukwe Peter
Session: 2020/2021
Semester: 1st

COURSE CODE	COURSE TITLE	UNIT LOAD	GRADE
COS 101	Introduction to Computer Science	2	A
COS 124	Introduction to Database systems	3	A
GSP 101	Use of English	2	A
GSP 111	Use of Library	2	A
MTH 121	Elementary Mathematics II	3	F
PHY 115	General Physics for Physical Sciences I	2	B
STA 131	Inference	2	B

Figure 4.11: This is the results output page that shows the available results for any semester the Student or Parent selects.

Django administration
WELCOME, OMEH. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home - Rst - Results

Start typing to filter...

RST

Courses + Add
Grades + Add
Levels + Add
Parents + Add
Results + Add
Semesters + Add
Sessions + Add
Students + Add

Select result to change

Search

Action: ----- Go 0 of 37 selected

<input type="checkbox"/>	STUDENT	COURSE	GRADE
<input type="checkbox"/>	2020/339482	COS 232	B
<input type="checkbox"/>	-	STA 205	C
<input type="checkbox"/>	2020/241163	STA 131	B
<input type="checkbox"/>	2020/241163	PHY 115	B
<input type="checkbox"/>	2020/241163	MTH 121	F
<input type="checkbox"/>	2020/241163	GSP 111	A
<input type="checkbox"/>	2020/241163	GSP 101	A
<input type="checkbox"/>	2020/241163	COS 124	A
<input type="checkbox"/>	2020/241163	COS 101	A
<input type="checkbox"/>	2020/241163	COS 384	B
<input type="checkbox"/>	2020/241163	COS 386	A
<input type="checkbox"/>	2020/241163	COS 382	A

ADD RESULT +

FILTER

Show counts

By code

All
CED 341
CED 342
COS 101
COS 102
COS 104
COS 124
COS 201
COS 202
COS 203
COS 204
COS 231
COS 232
COS 242
COS 311
COS 331
COS 333
COS 335
COS 337

Figure 4.12: This is the output page that shows when the exam officer clicks the button to update a result.

4.4 Documentation

4.4.1 User Manual

To get started using the system follow these steps:

- Ensure you have python version 3 on your computer. If you do not install it by going to the official website.
- Activate a virtual environment. You can do this through this process:
 - Open your command prompt.
 - Navigate to the location of the project folder on your computer using the “cd” command.
 - Type this code and press enter “python –m venv env”
 - The virtual environment has been created in a folder “env”
 - Type this code and press enter to activate it “env\scripts\activate”.
- Install the Django framework by typing this code “pip install Django”
- Run the server for the project by typing this code “python manage.py runserver”
- Open a web browser and paste the url gotten from the server launch above

Once the browser is open a Student should register through the sign page. He or she would be required to provide his or her registration number, password, first name, last name, a password, phone number, his or her parent’s name, email and phone number. If all the credentials provided are of correct type then both the Student and Parent will be registered on the system. The Parent would receive his or her password via email. After signing up the user would be directed to the login page. The Student or Parent or Exam officer should input his or her username and password in order to login. If the credentials are correct the user will be logged in and directed to the home page. If the user is the exam officer, the home page would have two buttons that leads to other pages for result updating and result uploading respectively. If the Exam officer wants to upload student results he or she should click the “upload result” button. He or she will be directed to a search page where student’s registration numbers can be searched. The Exam officer can search for the student that he or she wants to upload a result for. Upon searching a list of registration numbers that matches the search input is returned. The Exam officer can now select the number he or she wants to upload result for by clicking on it. After clicking on the registration the exam officer would be directed to the upload result page. The Exam officer would be required to provide the details of the result such as the course, grade, score, semester and session of the result. Once provided the exam officer should click the upload button to upload the results. Once the upload is successful an email would be sent to the Student’s Parent informing him

or her about the new result published and the exam officer would be redirected to the search page to make another upload. If the Exam officer wants to update a result he or she should click on the home page menu. The Exam officer would now be able to see the “update results” button. If clicked the exam officer would be redirected to the update page where he or she can search for a particular result and make changes.

If it is a Student or Parent that logs in, the home page would have a “view results” button. When clicked the user would be directed to a page that requests them to choose the level and semester result to be viewed. If the level and semester is selected the user would be directed to a page that shows all the available results of the student for that semester.

4.4.1 Source code listing

The source code used to implement the system can be found in the appendices.

Chapter Five

Summary and Conclusion

5.0 Summary

The Result Tracking System for University Students and Parents was designed to streamline and enhance the process of tracking Students academic performance for themselves and their Parents or Guardians. The system allows students and their parents to access results online, offering a seamless user experience. An authenticated admin interface was implemented to facilitate the uploading of student results; ensuring only authorized personnel can manage the system's data. Additionally, an automated email notification system was integrated to notify a student's parent whenever a new result is published, keeping parents informed about their child's academic progress in real-time.

5.1 Conclusion

In conclusion, the Result Tracking System successfully addresses the challenges of Parents getting access to their ward's results and result management in universities. By providing an easy-to-use interface for both students and parents, the system enhances transparency and communication regarding academic performance. The inclusion of an email notification feature offers real-time updates to parents, improving their involvement in their children's education. The project effectively demonstrates the potential of web applications in modernizing traditional university administrative systems.

5.2 Recommendations

While the Result Tracking System meets the objectives of the project, there are several areas where improvements can be made:

- **User Interface Enhancements:**

Future development could focus on improving the user interface for students and parents. Adding features such as Grade point calculation, and downloadable reports would further improve the user experience.

- **Mobile Application:**

Developing a mobile version of the system could make it more accessible to users, particularly students and parents, who may prefer accessing results on their mobile devices.

- **Result Analysis and Visualization:**

Adding features for result analysis, such as graphical representations of academic performance (e.g., progress charts, comparison graphs), would provide students and parents with deeper insights into academic trends over time.

- **Notification System Expansion:**

The email notification system could be expanded to include multiple notification channels, such as SMS, or in-app notifications, for users who may not have regular access to email.

References

- [1] O. Oluwaseyitanfunmi, A.B Isaiah and A. O. Olalekan, "Design and Implementation of a Centralized University Result Processing and Transcript System: Case study of University of Ibadan," *International Journal of Computing Sciences Research*, [S.l.], vol. 2, no. 3, p. 89-101, Jan. 2019. ISSN 2546-115X. [Online] Available:<[//www.stepacademic.net/ijcsr/article/view/86](http://www.stepacademic.net/ijcsr/article/view/86)>. [Accessed: Jan 04, 2025].
- [2] A. J. Ekanem, S. Azuomba, and A. J. Jimoh, "Development of Result Management System: A case study of the University of Uyo", *Mathematical and Software Engineering*, Vol. 3, No. 1, 26-42, 2017. [Online] Available:https://www.academia.edu/40426148/Development_of_Students_Result_Management_System_A_case_study_of_University_of_Uyo [Accessed: Jan 04, 2025]
- [3] S. Udegbu, *Examination Result Processing System: a Case Study of Kano University of Science and Technology*, 2018. [E-book] Available: <https://afribary.com/works/examination-result-processing-system-a-case-study-of-kano-university-of-science-and-technology-udegbu-sunday>
- [4] O. Oluwadamilare and A. Bernard, "Modelling and Implementation of a Result Processing System," *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, vol. 8, no. 7, pp. 32-40, 2021. [Online]. Available: <https://pdfs.semanticscholar.org/1cbf/2297e9bc3f0148a160640fa79483cefecad6.pdf>. [Accessed: Jan 04, 2025]
- [5] K. Sarkar, K. Malviya, S. Rajput and N. P. S. Rathore, "Student Result Management System", *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, Volume 9 Issue XI , ISSN: 2321-9653, IC Value: 45.98, SJ Impact Factor: 7.429, Nov 2021. [Online]. Available: https://www.academia.edu/99854138/Student_Result_Management_System. [Accessed: Jan 04, 2025]
- [6] O. Marvis, *Design and Implementation of Student Result Management System-a Case Study of Wellspring University*, 2022. [E-book]. Available: https://www.researchgate.net/publication/383261998_DESIGN_AND_IMPLME

NTATION OF STUDENT RESULT MANAGEMENT SYSTEM- A CASE STUDY OF WELLSPRING UNIVERSITY.

[7] “Object-Oriented Analysis and Design,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Object-oriented_analysis_and_design.

[8] “Object-Oriented Paradigm,” *TutorialsPoint*. [Online]. Available: https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_paradigm.htm.

[9] “Three-Tier Architecture,” *IBM*. [Online]. Available: <https://www.ibm.com/think/topics/three-tier-architecture>.

[10] “What is Class Diagram?,” *Visual Paradigm*. [Online]. Available: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/>.

[11] “Class Diagram in UML,” *GeeksforGeeks*. [Online]. Available: <https://www.geeksforgeeks.org/class-diagram-in-uml/>.

[12] “Use case diagram,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Use_case_diagram.

[13] “Introduction to UML Use Case Diagrams,” *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/315023158_Introduction_to_UML_Use_Case_Diagrams.

[14] “Activity Diagrams,” *UML Diagrams*. [Online]. Available: <https://www.uml-diagrams.org/activity-diagrams.html>.

[15] “UML - Activity Diagram.” *TutorialsPoint*. [Online]. Available: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm.

Appendices

The entire source code can be found at <https://github.com/ndigital04/result-tracking-system>

Settings file

```
from pathlib import Path
from config.config import secret_key, database_password, email_password
BASE_DIR = Path(__file__).resolve().parent.parent
SECRET_KEY = secret_key
DEBUG = True
ALLOWED_HOSTS = []
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rst',
]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
ROOT_URLCONF = 'school_project.urls'
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
WSGI_APPLICATION = 'school_project.wsgi.application'
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'rts',
        'HOST': '127.0.0.1',
        'PORT': '3306',
        'USER': 'root',
```

```

        'PASSWORD': database_password,
    }
}
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True
STATIC_URL = 'static/'
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

```

Base urls.py code

```

from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('rst.urls')),
]

```

forms.py code

```

from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
from django import forms
class RegisterUserForm(UserCreationForm):
    class Meta:
        model = User
        fields= ['username', 'email', 'first_name', 'last_name', 'password1',
'password2']
        widgets = {
            'username': forms.TextInput({'placeholder': 'Reg Number'}),
            'email': forms.EmailInput({'placeholder': 'Email'}),
            'first_name': forms.TextInput({'placeholder': 'First Name'}),
            'last_name': forms.TextInput({'placeholder': 'Lastname'}),
        }
    def __init__(self, *args, **kwargs):
        super(RegisterUserForm, self).__init__(*args, **kwargs)

```

```

        self.fields['password1'].widget =
forms.PasswordInput(attrs={'placeholder': 'Password'})
        self.fields['password2'].widget =
forms.PasswordInput(attrs={'placeholder': 'Confirm Password'})
models.py code
from django.db import models
from django.contrib.auth.models import User
class Department(models.Model):
    name = models.CharField(max_length=50, null=False)
    def __str__(self):
        return self.name
class Exam_Officer(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, null=True)
    department = models.ForeignKey(Department, on_delete=models.CASCADE,
null=False)
    email = models.EmailField(null=False)
    def __str__(self):
        return self.user.first_name
class Course(models.Model):
    code = models.CharField(max_length=7, null=False)
    title = models.CharField(max_length=100, null=False)
    unit_load = models.IntegerField(null=False, blank=False)
    def __str__(self):
        return self.code
class Parent(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, null=True)
    phone_no = models.CharField(max_length=13, null=False)
    email = models.EmailField(null=False)
    def __str__(self):
        return self.user.username
class Student(models.Model):
    user= models.OneToOneField(User, on_delete=models.CASCADE, null=False)
    reg_no= models.CharField("Registration Number", max_length=11)
    department = models.ForeignKey(Department, on_delete=models.SET_NULL,
null=True)
    email = models.EmailField(null=False)
    parent = models.ForeignKey(Parent, on_delete=models.SET_NULL, null=True)
    def __str__(self):
        return self.reg_no
class Grade(models.Model):
    grade = models.CharField(max_length=1)
    def __str__(self):
        return self.grade
class Semester(models.Model):
    semester = models.CharField(max_length=11)
    def __str__(self):

```



```

        return self.semester
class Session(models.Model):
    session = models.CharField(max_length=9)
    def __str__(self):
        return self.session
class Level(models.Model):
    level = models.CharField(max_length=3)
    def __str__(self):
        return self.level
class Result(models.Model):
    student = models.ForeignKey(Student, on_delete=models.SET_NULL, null=True)
    course = models.ForeignKey(Course, on_delete=models.SET_NULL, null=True)
    grade = models.ForeignKey(Grade, on_delete=models.SET_NULL, null=True)
    semester = models.ForeignKey(Semester, on_delete=models.SET_NULL, null=True)
    session = models.ForeignKey(Session, on_delete=models.SET_NULL, null=True)
    level = models.ForeignKey(Level, on_delete=models.SET_NULL, null=True)
    exam_score = models.IntegerField(null=True)
    ca_score = models.IntegerField(null=True)
    def __str__(self):
        return self.student.reg_no if self.student else "unassigned result"

```

admin.py code

```

from django.contrib import admin
from .models import Student, Parent, Result, Department, Course, Grade, Semester,
Session, Level
@admin.register(Result)
class ResultAdmin(admin.ModelAdmin):
    list_display = ['student', 'course', 'grade']
    search_fields = ['student__reg_no', 'course__code']
    list_filter= ['course__code', 'session__session']
admin.site.register(Student)
admin.site.register(Parent)
admin.site.register(Department)
admin.site.register(Course)
admin.site.register(Grade)
admin.site.register(Semester)
admin.site.register(Session)
admin.site.register(Level)

```

send_email.py code

```

from email.mime.multipart import MIMEMultipart
import smtplib
from email.utils import formataddr
from email.header import Header
from string import Template
from pathlib import Path
from email.mime.text import MIMEText
from config.config import email_password

```

```

def send_email_to_registered_users(message_to,username,password):
    template = Template(
        Path("rst\\templates\\rst\\email.html").read_text())
    message = MIMEMultipart()
    message["from"] = formataddr(
        (str(Header('Result Checker', 'utf-8')), 'peterkingdbx@gmail.com'))
    message["to"] = message_to
    message["subject"] = "Result checker registration for your ward"
    body = template.substitute(
        {"username": username, "password": password})
    message.attach(MIMEText(body, "html"))
    print("about to open SMTP server")
    with smtplib.SMTP(host="smtp.gmail.com", port=587) as smtp:
        print("just opned smtp server")
        smtp.ehlo()
        smtp.starttls()
        print("About to login")
        smtp.login("peterkingdbx@gmail.com", email_password)
        print("logged in")
        smtp.send_message(message)
        print("message sent")
def send_email_for_newly_published_result(message_to,student_name,course,grade):
    template = Template(
        Path("rst\\templates\\rst\\new_result.html").read_text())
    message = MIMEMultipart()
    message["from"] = formataddr(
        (str(Header('New Published result', 'utf-8')), 'peterkingdbx@gmail.com'))
    message["to"] = message_to
    message["subject"] = "New results have been uploaded for your ward"
    body = template.substitute(
        {"student_name": student_name, "course": course, "grade":grade})
    message.attach(MIMEText(body, "html"))
    print("about to open SMTP server")
    with smtplib.SMTP(host="smtp.gmail.com", port=587) as smtp:
        print("just opned smtp server")
        smtp.ehlo()
        smtp.starttls()
        print("About to login")
        smtp.login("peterkingdbx@gmail.com", email_password)
        print("logged in")
        smtp.send_message(message)
        print("message sent")

```

urls.py code

```

from django.urls import path
from . import views
app_name = "rst"

```

```
urlpatterns = [
    path('', views.index, name="index"),
    path('admin-dashboard/', views.adminDashboard, name="adminPage"),
    path('signup/', views.signup, name="signup"),
    path('login/', views.login_view, name="login"),
    path('logout/', views.logout_view, name="logout"),
    path('admin-dashboard/upload-result/', views.search, name="search"),
    path('admin-dashboard/upload-result/<int:student_id>/', views.upload_result,
name="upload"),
    path('view_result/', views.view_result, name="view_result"),
    path('view_result/<level>/<semester>/', views.view_level_result,
name="view_level_result"),
]
```

views.py code

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import RegisterUserForm
from django.contrib import messages
from django.urls import reverse
from django.core.mail import send_mail
from .models import Student, Parent, Department, Course, Grade, Semester,
Session, Level, Result
from django.contrib.auth.models import User
from .send_email import send_email_to_registered_users,
send_email_for_newly_published_result
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.decorators import login_required
from django.contrib.admin.views.decorators import staff_member_required
@login_required(login_url= "rst:login")
def index(request):
    if request.user.is_staff:
        return HttpResponseRedirect(reverse("rst:adminPage"))
    context= {
        "user": request.user.first_name,
    }
    return render(request, "rst/index.html", context)
@staff_member_required
def adminDashboard(request):
    context= {
        "user":request.user.first_name
    }
    return render (request, "rst/adminpage.html", context)
def login_view(request):
    if request.user.is_authenticated:
        if request.user.is_staff:
            return HttpResponseRedirect(reverse("rst:adminPage"))
```

```

        return HttpResponseRedirect(reverse("rst:index"))
    if request.method == "POST":
        username = request.POST.get("username")
        password = request.POST.get("password")
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            if request.user.is_staff:
                return HttpResponseRedirect(reverse("rst:adminPage"))
            return HttpResponseRedirect(reverse("rst:index"))
    context = {}
    return render(request, "rst/login.html", context)
def logout_view(request):
    logout(request)
    return HttpResponseRedirect(reverse("rst:login"))
def signup(request):
    form = RegisterUserForm()
    departments = Department.objects.all()
    context = {'form': form,
               'departments': departments
    }
    if request.method == "POST":
        print("Post data received")
        form = RegisterUserForm(request.POST)
        parent_email = request.POST.get("parent-email")
        if not parent_email:
            form = RegisterUserForm()
            context = {
                "form": form,
                "error": "Parent email must be present",
                "departments": departments
            }
        return render(request, "rst/signup.html", context)
    if form.is_valid():
        print(form)
        form.save()
        messages.success(request, ("Sign up succesful"))
        username = request.POST.get("username")
        reg_number = username[0:4] + "/" + username[4:]
        department = request.POST.get("department")
        print(request.POST)
        print(department, request.POST["department"])
        department = Department.objects.get(pk=department)
        user = User.objects.get(username=username)
        student = Student.objects.create(user=user,
        reg_no=reg_number,email=user.email, department=department)

```

```

        student.save()
        parent_phone_no = request.POST.get("parent-number")
        parent_user = User.objects.create_user(
            username="p" + username, password=reg_number, email=parent_email)
        parent_user.save()
        parent = Parent.objects.create(
            user=parent_user, phone_no =parent_phone_no
, email=parent_user.email)
        parent.save()
        student.parent = parent
        student.save()
        send_email_to_registered_users(parent_user.email,parent_user.username
, reg_number)
        return HttpResponseRedirect(reverse("rst:login"))
    if form.errors:
        print("Form has errors")
        context = {
            'form': RegisterUserForm(),
            'errors': str(form.errors),
            'departments':departments
        }
        print(form.errors, form.error_messages)
        return render(request, 'rst/signup.html', context)
    return render(request, "rst/signup.html", context)
@staff_member_required
def search(request):
    if request.method == "POST":
        searched = request.POST.get("searched")
        students = Student.objects.filter(reg_no__contains=searched)
        context = {
            "searched":searched,
            "students":students
        }
        return render(request, "rst/search.html", context)
    context = {}
    return render(request, "rst/search.html", context)
@staff_member_required
def upload_result(request, student_id):
    student = Student.objects.get(id=student_id)
    courses = Course.objects.all().order_by("code").values()
    sessions = Session.objects.all()
    semesters = Semester.objects.all()
    levels = Level.objects.all()
    grades = Grade.objects.all()
    if request.method == "POST":
        print(request.POST)

```

```

student = Student.objects.get(id=student_id)
course_id = request.POST.get("course")
course = Course.objects.get(id=course_id)
grade_id = request.POST.get("grade")
grade = Grade.objects.get(id=grade_id)
semester_id = request.POST.get("semester")
semester = Semester.objects.get(id=semester_id)
session_id = request.POST.get("session")
session = Session.objects.get(id=session_id)
level_id = request.POST.get("level")
level = Level.objects.get(id=level_id)
exam_score = request.POST.get("exam-score")
ca_score = request.POST.get("ca-score")
try:
    result = Result.objects.get(student=student, course=course,
session=session)
    if result:
        context = {
            "student": student,
            "courses": courses,
            "sessions": sessions,
            "semesters": semesters,
            "levels": levels,
            "grades": grades,
            "exists": "The student already has this result. You may go
and update it"
        }
        return render(request, "rst/upload.html", context)
except (Result.DoesNotExist):
    result = Result.objects.create(
        student=student, course=course, grade=grade, semester=semester,
        session=session, level=level, exam_score=exam_score,
ca_score=ca_score)
    result.save()
    parent_email = student.parent.email
    send_email_for_newly_published_result(
        parent_email, student.user.first_name, course.code, grade)
    context = {
        "student": student,
        "courses": courses,
        "sessions": sessions,
        "semesters": semesters,
        "levels": levels,
        "grades": grades,
        "success": "Result upload Succesful"
    }

```

```

        return render(request, "rst/search.html", context)
context = {
    "student": student,
    "courses": courses,
    "sessions": sessions,
    "semesters": semesters,
    "levels": levels,
    "grades": grades
}
return render(request, "rst/upload.html", context)
@login_required(login_url="rst:login")
def view_result(request):
    if request.user.username[0] == "p":
        parent = Parent.objects.get(user=request.user)
        student = parent.student_set.all()[0]
    else:
        student = Student.objects.get(user=request.user)
    results = Result.objects.filter(student=student)
    levels = Level.objects.all()
    semesters = Semester.objects.all()
    print(results)
    context = {
        "student": student,
        "levels": levels,
        "semesters": semesters
    }
    return render(request, "rst/view_result.html", context)
@login_required(login_url="rst:login")
def view_level_result(request, level, semester):
    if request.user.username[0] == "p":
        parent = Parent.objects.get(user=request.user)
        student = parent.student_set.all()[0]
    else:
        student = Student.objects.get(user=request.user)
    level = Level.objects.get(level=level)
    semester = Semester.objects.get(semester=semester)
    results = Result.objects.filter(student=student, level=level,
semester=semester)
    print(results)
    try:
        result = results[0]
        context = {
            "student": student,
            "level": level,
            "results": results,
            "result": result,

```

```

        "semester": semester
    }
    return render(request, "rst/view_level_result.html", context)
except(IndexError):
    context ={
        "unavailable": "Results for this semester are unavailable"
    }
    return render(request, "rst/view_level_result.html", context)

```

index.html template code

```

{% extends "rst/base.html" %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Index {% endblock %}</title>
</head>
<body>
    {% block home %}
    <a href="{% url 'rst:index' %}"> Home </a>
    {%endblock%}
    {% block content%}
    <div class="container">
        <div class="main-message">
            <p>Hi {{request.user.first_name}}, Welcome to the result checker
website for the department of computer science in the University of Nigeria
Nsukka</p>
            <div class="view-result">
                <a href= "{% url 'rst:view_result' %}"> <p>View your Results
Now</p></a>
            </div>
        </div>
    </div>
    {% endblock %}
</body>
</html>

```

search.html template code

```

{% extends 'rst/base.html' %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %} Search Student {% endblock %}</title>

```



```

</head>
<body>
    {% block home %}
    <a href="{% url 'rst:adminPage' %}"> Home </a>
    {%endblock%}
    {% block content %}
    <h1 class="search">Search for Student by Reg number</h1>
    {% if success %}
        <p> {{success}}</p>
    {% endif %}
    <form class="search" action="{% url 'rst:search' %}" method="post">
        {% csrf_token %}
        <input class="search" type="search" placeholder="type student
registration number" name="searched">
        <input class="submit" type="submit" value="Search">
    </form>
    {% if searched %}
    <div class="search-container">
    <p class="search"> Search results, you searched for {{searched}} </p>
    <ul class="search">
    {% for student in students %}
    <a href="{% url 'rst:upload' student.id %}"><li
class="search">{{student}}</li></a>
    {% endfor %}
    </ul>
    </div>
    {% endif %}
    {% endblock %}
</body>
</html>

```

upload.html template code

```

{% extends 'rst/base.html' %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %} Upload Student Result {% endblock %}</title>
</head>
<body>
    {% block home %}
    <a href="{% url 'rst:adminPage' %}"> Home </a>
    {%endblock%}
    {% block content %}
    <h1 class="upload">Upload Student Result</h1>

```

```

<div class="response">
    {% if success %}
    <p> {{success}} </p>
    {% endif %}
    {% if exists %}
    <p>{{exists}}</p>
    {% endif %}
</div>
<div class="container upload-container">
    <form action="{% url 'rst:upload' student.id %}" method="post">
        {% csrf_token %}
        <div class="student-data">
            <p>Registration Number: {{student.reg_no}}</p>
            <p>Name: {{student.user.last_name}}
            {{student.user.first_name}}</p>
        </div>
        <div class="input">
            <label for="course">Course:</label>
            <select id="course" name="course">
                {% for course in courses %}
                <option value= {{course.id}}>{{course.code}}</option>
                {%endfor%}
            </select>
        </div>
        <div class="input">
            <label for="level">Level:</label>
            <select id="level" name="level">
                {% for level in levels %}
                <option value= {{level.id}}>{{level}}</option>
                {%endfor%}
            </select>
        </div>
        <div class="input exam-score">
            <label for="exam-score">Exam Score:</label>
            <input type="number" name="exam-score" id="exam-score">
        </div>
        <div class="input ca-score">
            <label for="ca-score">CA Score:</label>
            <input type="number" name="ca-score" id="ca-score">
        </div>
        <div class="input">
            <label for="grade">Grade:</label>
            <select id="grade" name="grade">
                {% for grade in grades %}
                <option value= {{grade.id}}>{{grade}}</option>
                {%endfor%}
            </select>
        </div>
    </form>
</div>

```

```

        </select>
    </div>
    <div class="input">
        <label for="session">Session:</label>
        <select id="session" name="session">
            {% for session in sessions %}
                <option value= {{session.id}}>{{session}}</option>
            {%endfor%}
        </select>
    </div>
    <div class="input">
        <label for="semester">Semester:</label>
        <select id="semester" name="semester">
            {% for semester in semesters %}
                <option value= {{semester.id}}>{{semester}}</option>
            {%endfor%}
        </select>
    </div>
    <input type="submit" value="upload" class="upload">
</form>
</div>
{% endblock %}
</body>
</html>

```

view_level_result.html template code

```

{% extends 'rst/base.html' %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %} Result for {{level}} level{% endblock %}</title>
</head>
<body>
    {% block home %}
    <a href="{% url 'rst:index' %}"> Home </a>
    {%endblock%}
    {% block content %}
    <h1>Result for {{level.level}} level</h1>
    <div class="container">
        <div class="results-container">
            <div class="student-data">
                <p>Registration Number: {{student.reg_no}}</p>
                <p>Name: {{student.user.last_name}}
                {{student.user.first_name}}</p>
            </div>
        </div>
    </div>
    {%endblock%}
    </body>
</html>

```

```

        <p>Session: {{result.session}}</p>
        <p>Semester: {{result.semester}}</p>
    </div>
    {% if unavailable %}
    {{unavailable}}
    {% endif %}
    <table>
        <tr>
            <th> Course Code</th>
            <th>Course Title</th>
            <th> Unit load </th>
            <th> Grade </th>

        </tr>
        {% for result in results %}
        <div class="results-row">
            <tr class="count">
                <td> {{result.course.code}}</td>
                <td> {{result.course.title}}</td>
                <td> {{result.course.unit_load}}</td>
                <td> {{result.grade}}</td>
            </tr>
        </div>
        {% endfor %}
    </table>
    </div>
</div>
{% endblock %}
</body>
</html>
styles.css css code
* {
    left: 50px;
    margin: 0;
    padding: 0;
}
body {
    background-color: #023;
    color: aquamarine;
    font-family: poppins;
    overflow-x: hidden;
}
/* Login page */
a.signup{
    position: absolute;
    top: 30px;
}
a.signup:visited{
    color: #fff;
}
h1{
    text-align: center;
    margin-top: 40px;
}
h1.login{
    text-align: center;
    font-family: 'poppins';
    margin-top: 30px;
}

```

```

div.container{
    width: 100%;
    display: flex;
    justify-content: center;
}

div.container form, div.results-
container{
    width: 30%;
    height: fit-content;
    margin: 30px auto;
    padding: 20px;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    border-radius: 20px;
    background-color:
rgba(127,255,212,.3);
}

div.container form input{
    width: 80%;
    margin: 20px 0;
    height: 40px;
    outline: none;
    border: none;
    border-radius: 10px;
    text-indent: 3px;
}

#login:hover, input.signup:hover,
input.upload:hover{
    background-color: #023;
    color: #fff;
}

/* Nav bar */
nav{
    width: 100%;
    position: absolute;
    top: 0;
    background-color:
rgba(127,255,212,.3);
    display: flex;
    height: 60px;

    padding: 20px;
}
main{
    margin-top: 100px;
}

div.logo-container{
    width: 50%;
    display: flex;
    justify-content: flex-start;
    align-items: center;
    gap: 30px;
}

div.logo-container div{
    height: fit-content;
}

div.logo-container div a, a.signup{
    color: #fff;
    border-bottom: 2px solid #fff;
    text-decoration: none;
}

div.logo-container div a:hover{
    color: #023;
    border-bottom: 2px solid #023;
}

div.unn-logo{
    background-image:
url("../images/unn logo.jpg");
    width: 50px;
    height: 50px;
    border-radius: 50px;
    background-size: cover;
    background-repeat: no-repeat;
    margin-right: 10px;
}

div.logo-container p{
    font-size: 2.3rem;
    font-family: "poppins";
    color: #fff;
}

div.logout{
    position: absolute;
    right: 70px;

```

