Mini Project Report

Student Database Management System

- N.V.V. GANESH(RA1911027010109)
- VARUN KHAREEDU(RA1911027010118)
- PHANI SWAROOP(RA1911027010114)

Table of Contents:

1.Abstract:

The Student Management System provides a simple interface for maintenance of student information. It can be used by educational institutes or colleges to maintain the records of students easily. The creation and management of accurate, up-to-date information regarding a students' academic career is critically important in the university as well as colleges. Student information system deals with all kinds of student details, academic related reports, college details, course details, curriculum, batch details, and other resource related details too. Achieving this objective is difficult using a manual system as the information is scattered, can be redundant and collecting relevant information may be very time consuming. All these problems are solved using this project.

2.Introduction:

Purpose:

The objective of Student management System is to allow the administrator of any organisation to edit and find out the personal details of a student and allows the student to keep up to date his profile .It'll also facilitate keeping all the records of students, such as their id, name, phone number, DOB etc. So all the information about a student will be available in a few seconds. Overall, it'll make Student Information Management an easier job for the administrator and the student of any organisation. The main purpose of this document is to illustrate the requirements of the project Student information System and is intended to help any organisation to maintain and manage its student's personal data.

Scope:

Without a Student information System, managing and maintaining the details of the student is a tedious job for any organisation. The Student Information system will store all the details of the students including their background information, educational qualifications, personal details and all the information related to their resume .

Login module: Login module will help in authentication of user accounts .Users who have valid login id and password can only login into their respective accounts.

Search module: Suppose there are hundreds of students and from this we have to search a particular student and we know the name of the student .In manual system it is a tedious task though we know the name of the student, but using this module we can easily search the student by specifying the name of the student in the search criteria. Thus this module will help the administrator in searching the student with various criteria easily.

3.Functionality:

There are two different users who will be using this product:

- Administrator who can view and edit the details of any students.
- Students who can view their details as well as they can edit their details.

The features that are available to the Administrator are:

- An Administrator can login into the system and perform any of the available operations.
- Can enable/disable students.
- Can edit student information to the database.
- Can make a search for a specific student.
- Can access all the details of the student.

The features that are available to the student are:

- Students can login into the system and can perform any of the available options.
- Can view his/her personal details.
- Can edit his/her personal details.
- Can upload his/her official documents.
- Can upload his/her image.

4. s/wand h/w requirements:

Software Requirements:

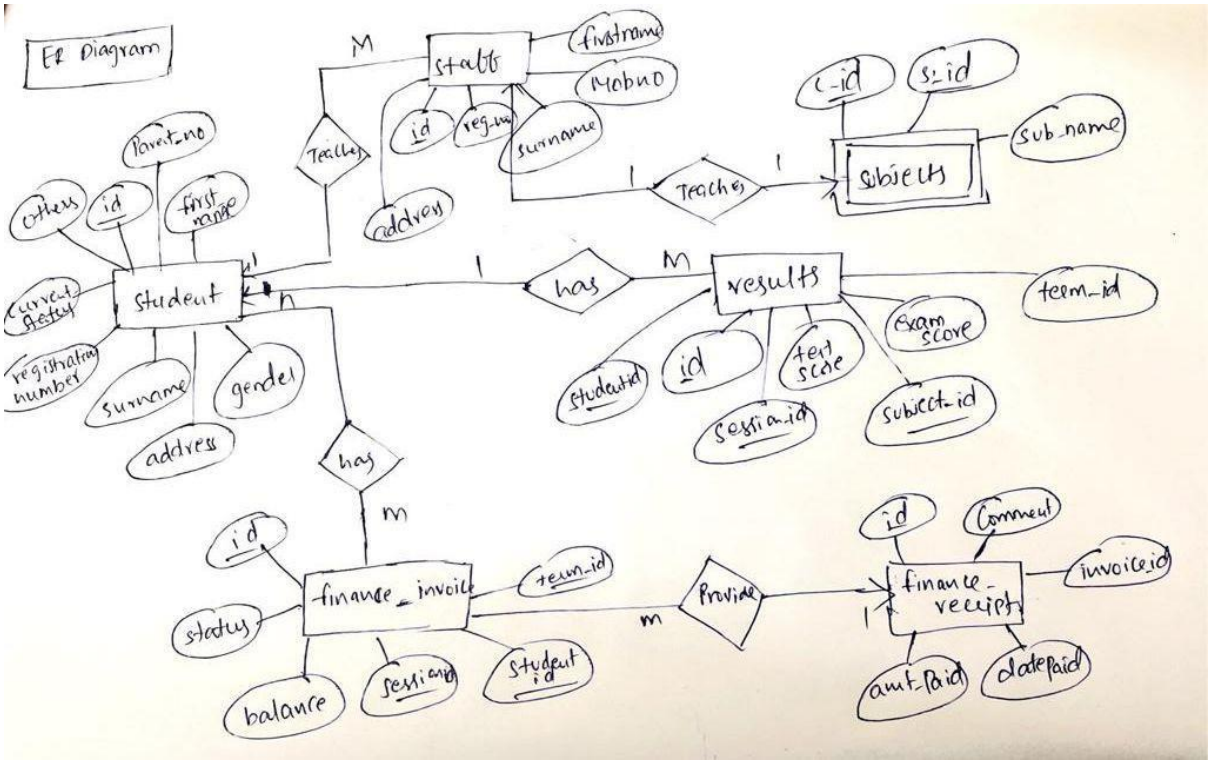
- Django
- SQLITE DBbrowser
- VS code editor
- Microsoft windows

Hardware Requirements:

- 512 MB Ram or Higher
- 20 GB HDD or Higher
- Intel Pentium IV processor or equivalent or higher
- Network Connectivity

5.Database Design:

ER Diagram:



Schemes used:

User_group_permissions table:

Fields	Data Type	Relationships
id	int	Primary Key
user_id	int	Candidate key
permission_id	int	Candidate key

Student Table:

Fields	Data Type	Relationships
id	Varchar(10)	Primary Key
Current_status	Varchar(200)	Not Null
registration_number	Varchar(200)	Not Null
surname	Varchar(20)	Not Null
firstname	Varchar(20)	Not null
gender	Varchar(10)	Not Null
address	Varchar(50)	Not Null
parent_mobile_number	Varchar(13)	Not Null
others	text	Not Null

Staff Table:

Fields	Data Type	Relationships

id	Varchar(10)	Primary Key
Current_status	Varchar(200)	Not Null
registration_number	Varchar(200)	Not Null
surname	Varchar(20)	Not Null
firstname	Varchar(20)	Not null
gender	Varchar(10)	Not Null
address	Varchar(50)	Not Null
parent_mobile_number	Varchar(13)	Not Null
others	text	Not Null

Finance_invoice table:

Fields	**Data Type**	**Relationships**
id	int	Candidate Key

status	varchar(20)	Not Null
balance_from_previous_term	int	Not Null
class_for_id	int	Candidate Key
session_id	int	Candidate Key
student_id	int	Candidate Key
term_id	int	Candidate Key

Finance_receipt table:

Fields	**Data Type**	**Relationships**
id	int	Primary Key
amount_paid	int	Not Null
date_paid	**date**	**Not Null**
comment	**Varchar(200)**	**Not Null**
invoice_id	**int**	**Not Null**

Result_result table:

Fields	Data Type	Relationships
id	int	Primary Key
test_score	int	Not Null
exam_score	int	Not Null
current_class_id	int	Candidate key
session_id	int	Candidate key
student_id	int	Candidate key
subject_id	int	Candidate key
term_id	int	Candidate key

6.DBMS concepts mapped in project

- We used the 'Inner Join' Project to combine two tables by mapping the Foreign key of 1st table to the primary key of 2nd table.
- We used a SELECT statement to extract the Attributes FROM the Table.
- We used WHERE statement to specify the clause given by user through frontend
- We used an AND statement to combine two conditions.

- We used an 'In' statement to check students from a Group of specified students.

7.Front End and Backend technologies:

Frontend	Web designing	Backend
● Html ● Css ● Javascript ● python	● Django	● sqlite

8.Normalisation level:

As we are managing multiple tables, so when we were trying to combine the course table with the student table we were getting errors as some values of the field were NULL which were intended not to be NULL during the process of making the structure. So we decomposed the table into 3 parts making the student, couse, instructor as separate tables. We are getting a lossy composition while joining the tables. We solved it by mapping the keys correctly with the respective database.

9.Sample Code:

Student views.py:

```
import csv


from django.contrib.auth.mixins import LoginRequiredMixin

from django.contrib.messages.views import SuccessMessageMixin

from django.forms import widgets

from django.http import HttpResponse

from django.urls import reverse_lazy

from django.views.generic import DetailView, ListView, View
```

```
from django.views.generic.edit import CreateView, DeleteView, UpdateView

from apps.finance.models import Invoice

from .models import Student, StudentBulkUpload



class StudentListView(LoginRequiredMixin, ListView):
    model = Student
    template_name = "students/student_list.html"



class StudentDetailView(LoginRequiredMixin, DetailView):
    model = Student
    template_name = "students/student_detail.html"

    def get_context_data(self, **kwargs):
        context = super(StudentDetailView, self).get_context_data(**kwargs)
        context["payments"] = Invoice.objects.filter(student=self.object)
        return context



class StudentCreateView(LoginRequiredMixin, SuccessMessageMixin, CreateView):
    model = Student
    fields = "__all__"
    success_message = "New student successfully added."

    def get_form(self):
```

```
        """add date picker in forms"""
        form = super(StudentCreateView, self).get_form()
        form.fields["date_of_birth"].widget = widgets.DateInput(attrs={"type": "date"})
        form.fields["address"].widget = widgets.Textarea(attrs={"rows": 2})
        form.fields["others"].widget = widgets.Textarea(attrs={"rows": 2})
        return form


class StudentUpdateView(LoginRequiredMixin, SuccessMessageMixin, UpdateView):
    model = Student
    fields = "__all__"
    success_message = "Record successfully updated."

    def get_form(self):
        """add date picker in forms"""
        form = super(StudentUpdateView, self).get_form()
        form.fields["date_of_birth"].widget = widgets.DateInput(attrs={"type": "date"})
        form.fields["date_of_admission"].widget = widgets.DateInput(
            attrs={"type": "date"}
        )
        form.fields["address"].widget = widgets.Textarea(attrs={"rows": 2})
        form.fields["others"].widget = widgets.Textarea(attrs={"rows": 2})
        # form.fields['passport'].widget = widgets.FileInput()
        return form


class StudentDeleteView(LoginRequiredMixin, DeleteView):
    model = Student
```

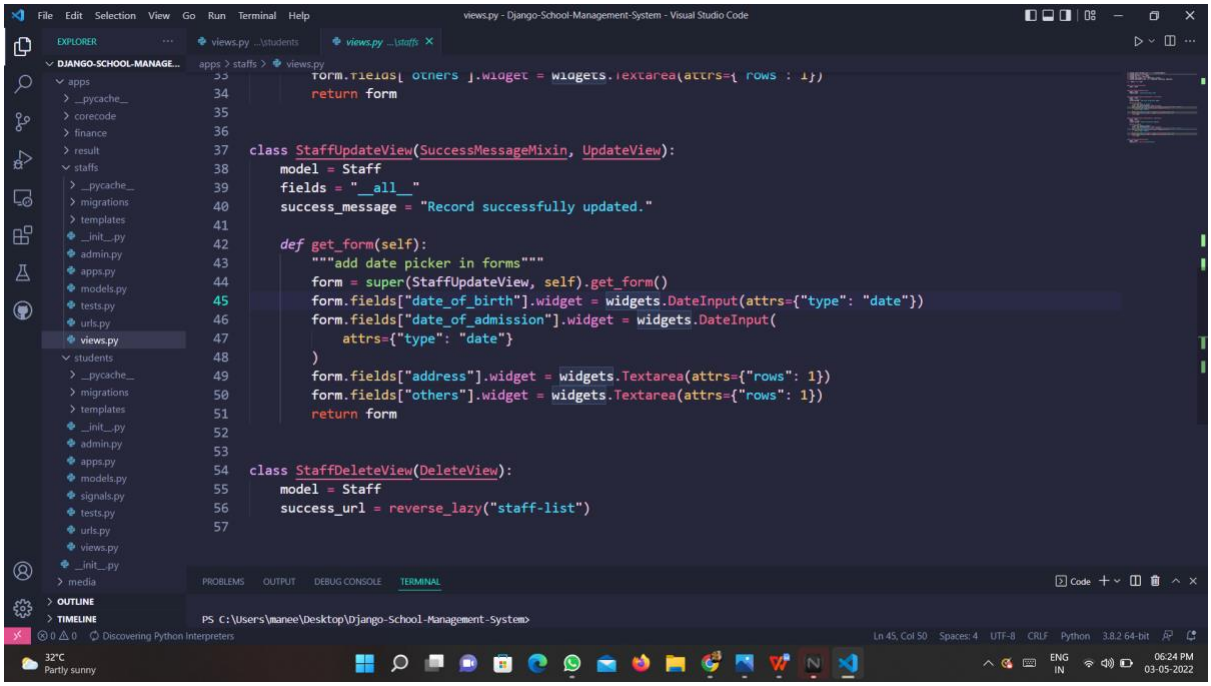
```
    success_url = reverse_lazy("student-list")



class StudentBulkUploadView(LoginRequiredMixin, SuccessMessageMixin, CreateView):
    model = StudentBulkUpload
    template_name = "students/students_upload.html"
    fields = ["csv_file"]
    success_url = "/student/list"
    success_message = "Successfully uploaded students"
class DownloadCSVViewdownloadcsv(LoginRequiredMixin, View):
    def get(self, request, *args, **kwargs):
        response = HttpResponse(content_type="text/csv")
        response["Content-Disposition"] = 'attachment; filename="student_template.csv"'
        writer = csv.writer(response)
        writer.writerow(
            [
                "registration_number",
                "surname",
                "firstname",
                "other_names",
                "gender",
                "parent_number",
                "address",
                "current_class",
            ]
        )
        return response
```

Staff views.py:

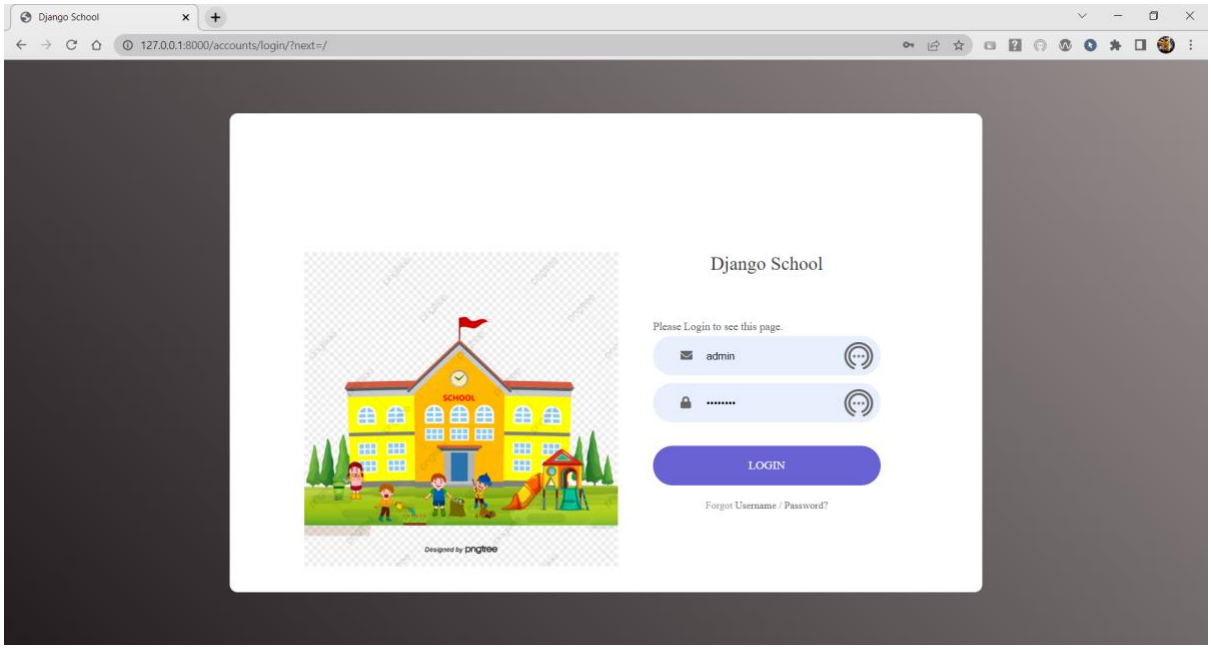
```
from django.contrib.messages.views import SuccessMessageMixin
from django.forms import widgets
from django.shortcuts import render
from django.urls import reverse_lazy
from django.views.generic import DetailView, ListView
from django.views.generic.edit import CreateView, DeleteView, UpdateView
from .models import Staff
class StaffListView(ListView):
    model = Staff
class StaffDetailView(DetailView):
    model = Staff
    template_name = "staffs/staff_detail.html"
class StaffCreateView(SuccessMessageMixin, CreateView):
    model = Staff
    fields = "__all__"
    success_message = "New staff successfully added"
    def get_form(self):
        """add date picker in forms"""
        form = super(StaffCreateView, self).get_form()
        form.fields["date_of_birth"].widget = widgets.DateInput(attrs={"type": "date"})
        form.fields["date_of_admission"].widget = widgets.DateInput(
            attrs={"type": "date"}
        )
        form.fields["address"].widget = widgets.Textarea(attrs={"rows": 1})
        form.fields["others"].widget = widgets.Textarea(attrs={"rows": 1})
        return form
class StaffUpdateView(SuccessMessageMixin, UpdateView):
```

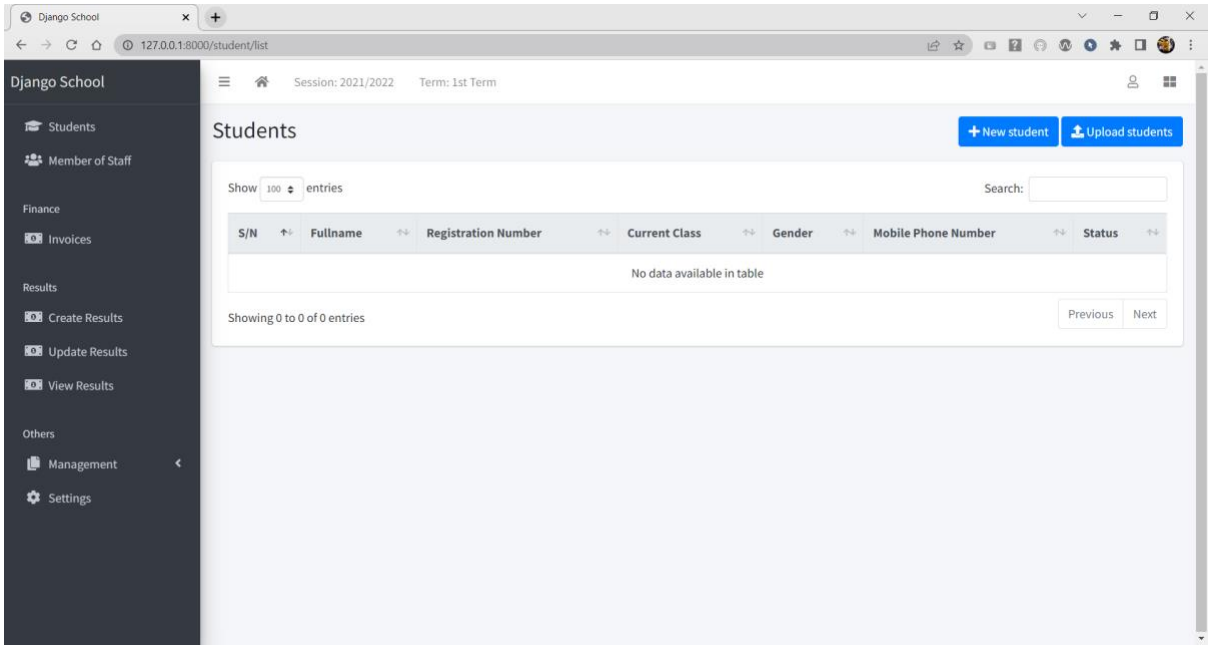
```
    model = Staff
    fields = "__all__"
    success_message = "Record successfully updated."
    def get_form(self):
DECLARE
    user_rec user_table%rowtype;
    pass_rec passbooking%rowtype;
    ticket_rec busticket%rowtype;
BEGIN
    SELECT * into user_rec
    FROM user_table
    WHERE user_id = 1;
    SELECT * into pass_rec
    FROM passbooking
    WHERE user_id = 1;
    SELECT * into ticket_rec
    FROM busticket
    WHERE user_id = 1;
    dbms_output.put_line('User ID: ' || user_rec.user_id);
    dbms_output.put_line('User First Name: ' || user_rec.first_name);
    dbms_output.put_line('User Last Name: ' || user_rec.last_name);
    dbms_output.put_line('User DOB: ' || user_rec.dob);
    dbms_output.put_line('Pass ID: ' || pass_rec.pass_id);
END;
CREATE OR REPLACE TRIGGER display_price_changes
BEFORE DELETE OR INSERT OR UPDATE ON passbooking
FOR EACH ROW
BEGIN
```

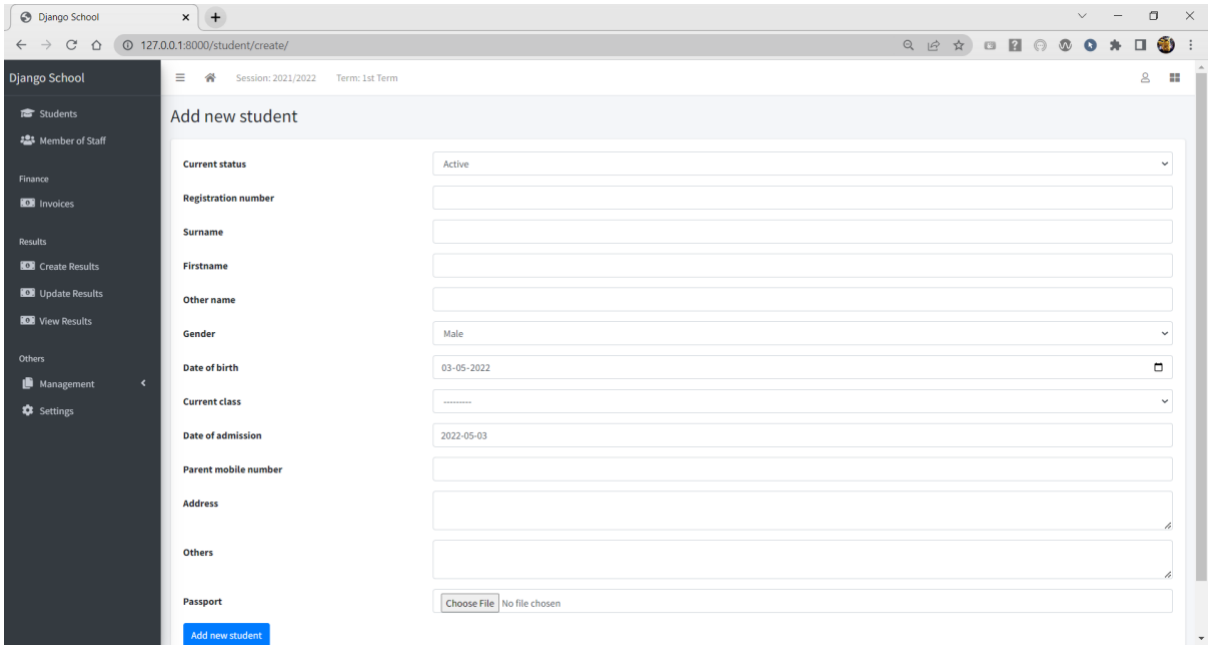
```
  price_diff := :NEW.pass_price  - :OLD.pass_price;
  dbms_output.put_line('Old salary: ' || :OLD.pass_price);
  dbms_output.put_line('New salary: ' || :NEW.pass_price);
  dbms_output.put_line('Salary difference: ' || price_diff);
  total_rows number(2);
BEGIN
  UPDATE passbooking
  SET pass_price = pass_price + 100;
  IF sql%notfound THEN
    dbms_output.put_line('No customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' price(s) updated ');
  END IF;
END;
        """add date picker in forms"""
        form = super(StaffUpdateView, self).get_form()
        form.fields["date_of_birth"].widget = widgets.DateInput(attrs={"type": "date"})
        form.fields["date_of_admission"].widget = widgets.DateInput(
            attrs={"type": "date"}
        )
        form.fields["address"].widget = widgets.Textarea(attrs={"rows": 1})
        form.fields["others"].widget = widgets.Textarea(attrs={"rows": 1})
        return form
class StaffDeleteView(DeleteView):
    model = Staff
    success_url = reverse_lazy("staff-list")
```

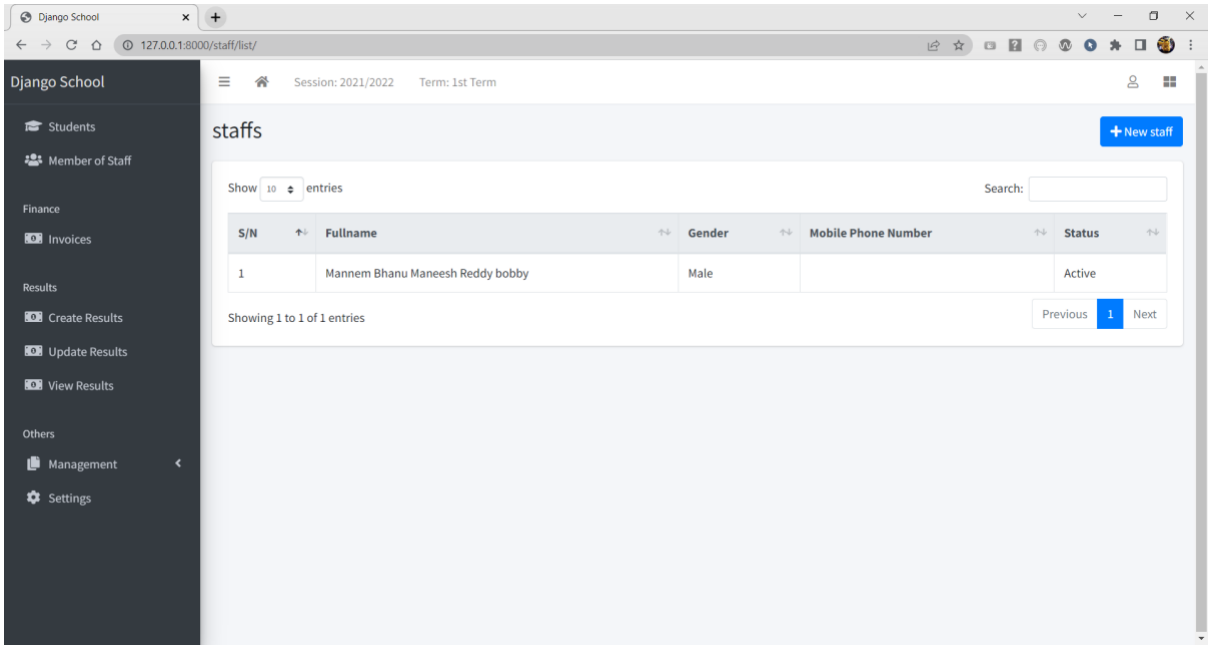


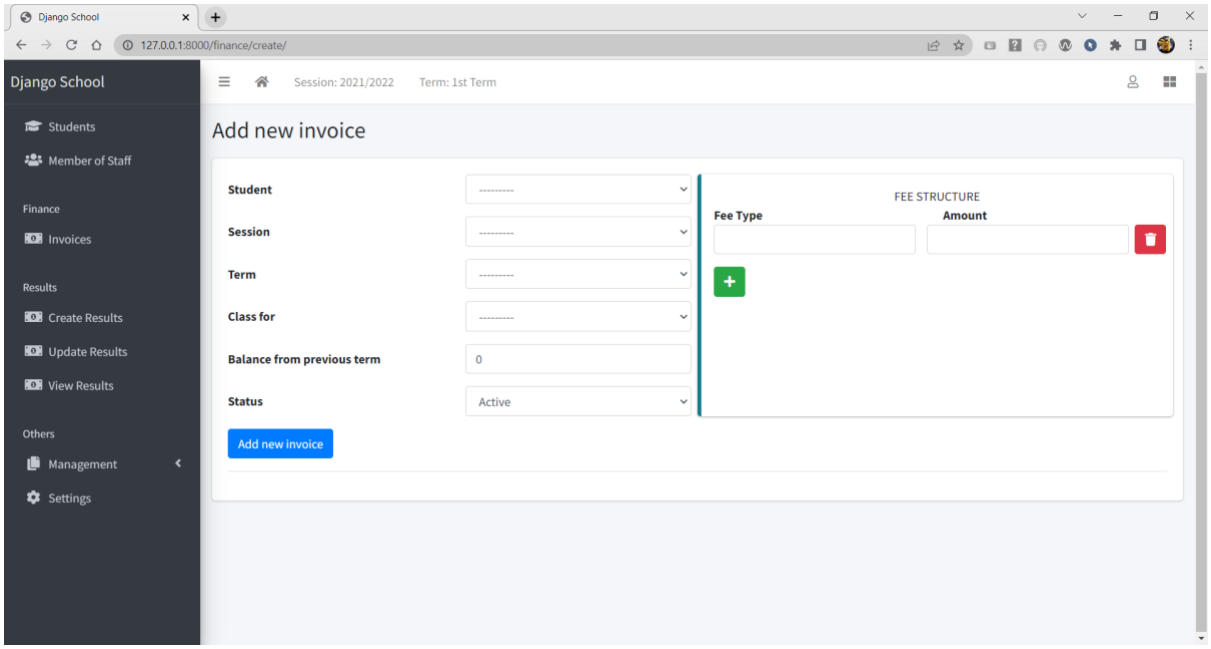
10.Screenshots:

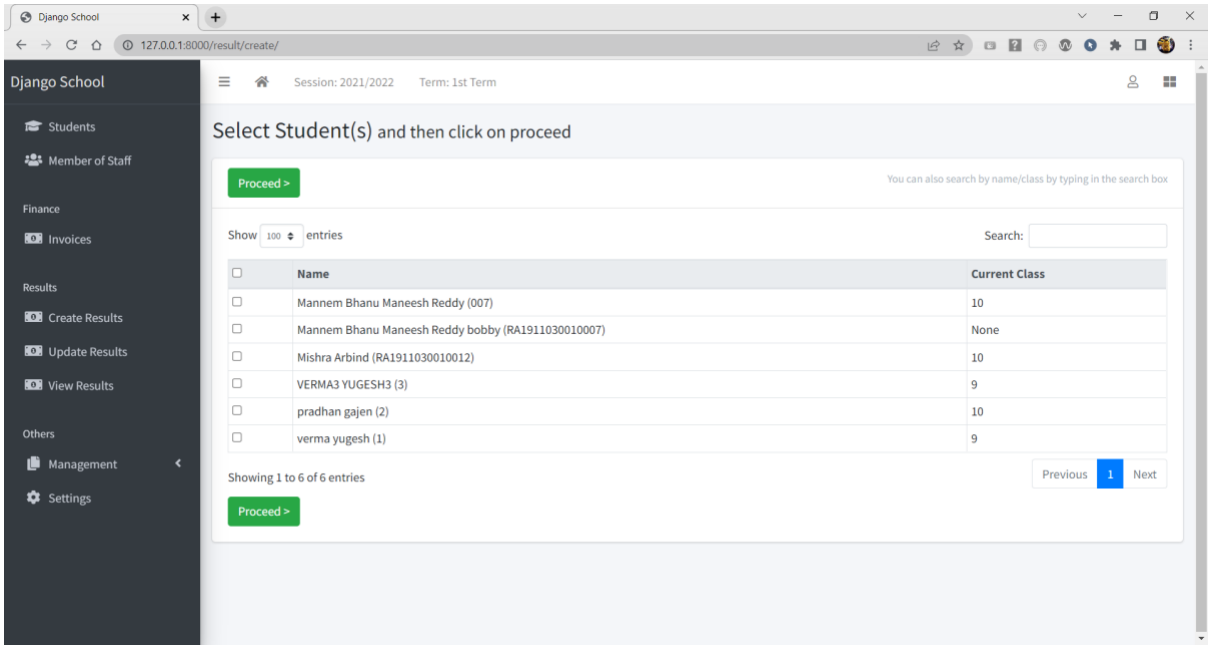


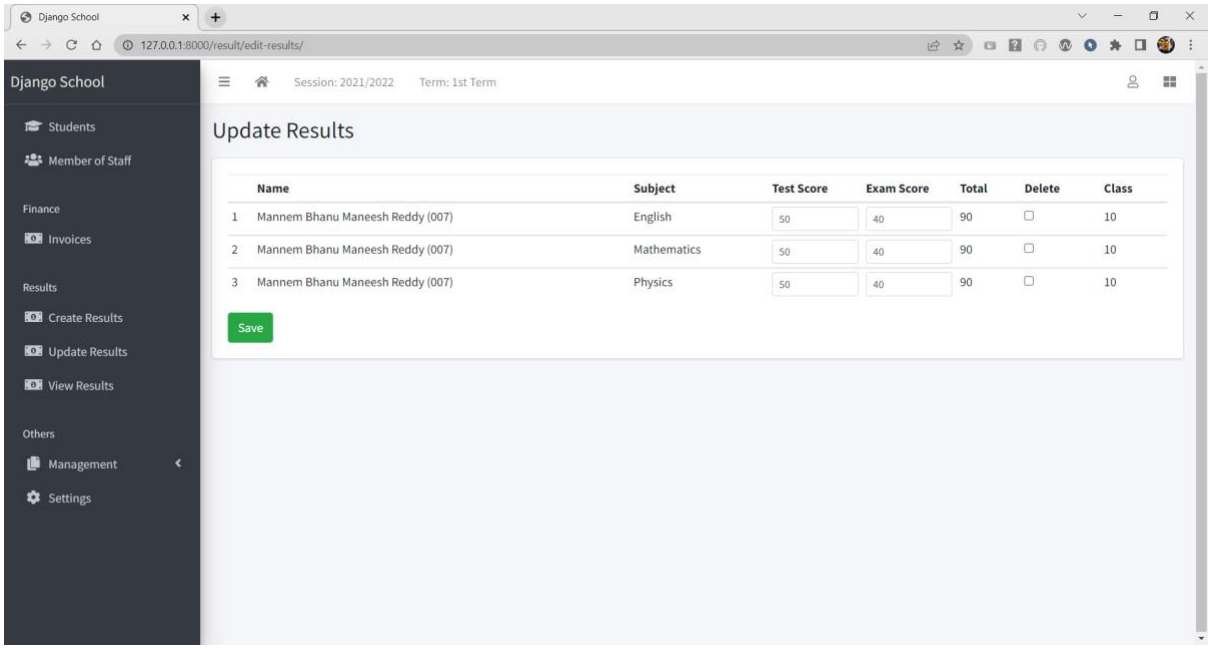

Django School
127.0.0.1:8000/student/list
Django School
Students
Member of Staff
Finance
Invoices
Results
Create Results
Update Results
View Results
Others
Management
Settings
Session: 2021/2022
Term: 1st Term
Students
+ New student
Upload students
Show 100 entries
Search:
S/N
Fullname
Registration Number
Current Class
Gender
Mobile Phone Number
Status
No data available in table
Showing 0 to 0 of 0 entries
Previous
Next

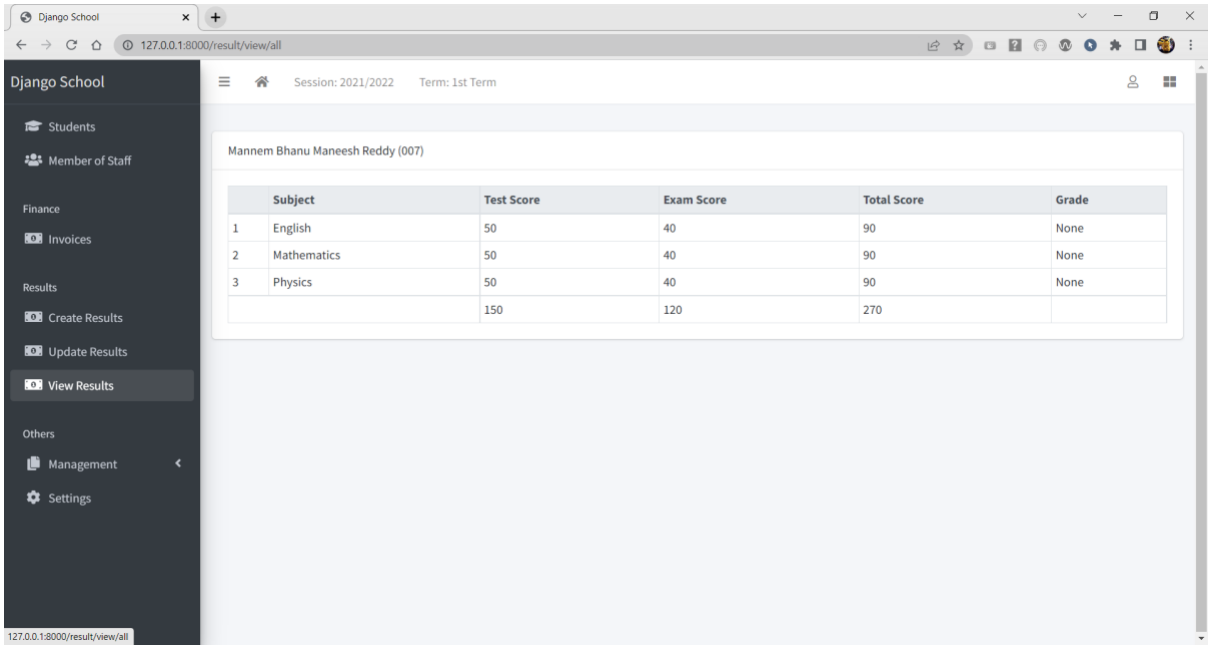

Django School
127.0.0.1:8000/student/create/
Django School
Students
Member of Staff
Finance
Invoices
Results
Create Results
Update Results
View Results
Others
Management
Settings
Session: 2021/2022
Term: 1st Term
Add new student
Current status
Active
Registration number
Surname
Firstname
Other name
Gender
Male
Date of birth
03-05-2022
Current class
Date of admission
2022-05-03
Parent mobile number
Address
Others
Passport
Choose File
No file chosen
Add new student


Django School
127.0.0.1:8000/staff/list/
Django School
Students
Member of Staff
Finance
Invoices
Results
Create Results
Update Results
View Results
Others
Management
Settings
Session: 2021/2022
Term: 1st Term
staffs
New staff
Show 10 entries
Search:
S/N
Fullname
Gender
Mobile Phone Number
Status
1
Mannem Bhanu Maneesh Reddy bobby
Male
Active
Showing 1 to 1 of 1 entries
Previous
1
Next


Django School
127.0.0.1:8000/finance/create/
Django School
Students
Member of Staff
Finance
Invoices
Results
Create Results
Update Results
View Results
Others
Management
Settings
Session: 2021/2022
Term: 1st Term
Add new invoice
Student
Session
Term
Class for
Balance from previous term
0
Status
Active
Add new invoice
FEE STRUCTURE
Fee Type
Amount


Django School
127.0.0.1:8000/result/create/
Django School
Students
Member of Staff
Finance
Invoices
Results
Create Results
Update Results
View Results
Others
Management
Settings
Session: 2021/2022
Term: 1st Term
Select Student(s) and then click on proceed
Proceed >
You can also search by name/class by typing in the search box
Show 100 entries
Search:
Name
Current Class
Mannem Bhanu Maneesh Reddy (007)
10
Mannem Bhanu Maneesh Reddy bobby (RA1911030010007)
None
Mishra Arbind (RA1911030010012)
10
VERMA3 YUGESH3 (3)
9
pradhan gajen (2)
10
verma yugesh (1)
9
Showing 1 to 6 of 6 entries
Previous
1
Next
Proceed >


Django School
127.0.0.1:8000/result/edit-results/
Django School
Students
Member of Staff
Finance
Invoices
Results
Create Results
Update Results
View Results
Others
Management
Settings
Session: 2021/2022
Term: 1st Term
Update Results
Name
Subject
Test Score
Exam Score
Total
Delete
Class
1
Mannem Bhanu Maneesh Reddy (007)
English
50
40
90
10
2
Mannem Bhanu Maneesh Reddy (007)
Mathematics
50
40
90
10
3
Mannem Bhanu Maneesh Reddy (007)
Physics
50
40
90
10
Save



11.Conclusion:

The project titled “Student Management System” is developed using Html, css and javascript as front end and Django, sqlite database in back end to computerise the process of management of student records.

- Helpful to work paperless work and manage all the data.
- Provides easy, accurate, unambiguous and faster data access.

12.Future Enhancements:

- This software can be made for all OS.
- Higher Security features can be included in this software.
- Program scheduling can also be included in this software.
- This software can be implemented with an OS to reduce the overhead of installing and running the interface of each and every tool at different places.
- Automatic Shutdown through SMS can be implemented in this.
- Other possible modules like attendance, hostel, lab, library can be included.

Contributed by **Faculty**

N.V.V. GANESH (109) **HEMAVATHI**

P.V. PHANI SWAROOP (114)

VARUN KHAREEDU (118)