Teacher Parent Interactive LMS

Submitted by:

Fahad Anyit 2017-EE-177

Azhak Anwar 2017-EE-186

Supervised by: Mr. Umer Shahid

Department of Electrical Engineering

University of Engineering and Technology Lahore

Teacher Parent Interactive LMS

Submitted to the faculty of the Electrical Engineering Department
of the University of Engineering and Technology Lahore
in partial fulfillment of the requirements for the Degree of

Bachelor of Science

in

Electrical Engineering.

Internal Examiner

External Examiner

Director
Undergraduate Studies

Department of Electrical Engineering

University of Engineering and Technology Lahore

Declaration

I declare that the work contained in this thesis is my own, except where explicitly stated otherwise. In addition, this work has not been submitted to obtain another degree or professional qualification.

Signed: _____

Date: _____

Acknowledgments

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

For/Dedicated to/To my...

Contents

List of Figures

List of Tables

Abbreviations

LMS	**L**earning **M**anagement **S**ystem
ERD	**E**ntity **R**elationship **D**iagram
DFD	**D**ata **F**low **D**iagram
IDE	**I**ntegrated **D**evelopment **E**nvironment
JS	**J**ava**S**cript

Abstract

The main idea of this project is learning the implementation and working of the database in a real-world application. For this purpose, we developed a website using *the Django* web development framework of *Python* titled as **Teacher Parent Interactive LMS**. The idea of this project is to make a communication bridge for teachers of a school with their students' parents. The system is developed in such a way that parents and students can view their result anywhere and anytime, and contact the teacher via the phone number provided by teachers. Moreover, it is also convenient for teachers to update marks of the students anywhere and anytime remotely. Different views implemented for a different type of users and a high level of authorization is also implemented for the security of users and integrity of the system ...

Follow this link to watch demonstration video: video
Follow this link to visit the website: https://lms.pythonanywhere.com/
Follow this link to visit the code: https://github.com/AzhakAnwar/db-project

Chapter 1

INTRODUCTION

1.1 Problem Statement

The robust relation of a teacher with his pupil's parents is a great starting point for handling problems that come up at school. In this busy world, it is quite hard for parents to contact their children's teacher and taking the current situation of a pandemic due to **covid-19** into account, it has become more difficult for parents to visit the school physically. In some cases, parents do not know the actual performance of their children in school due to the bluffing of children. As a result, parents seem overconfident about their children's condition.

1.2 Proposed Solution

Keeping the above situation in view, a lite solution to all these problems is the need of the hour. Since it is the era of digitalization, so we provide an online Learning Management System (LMS) which comes in handy and easily accessible by parents and teachers so they can communicate comfortably and discuss the progress of students. This system will not only allow easy communication between parents and teachers but also informs parents of their children's performance which will motivate children to perform well in school. Parents can see their children's result at any place without having to worry about their busy schedule.

Chapter 2

PROPOSED DESIGN

2.1 ER Diagram

Before starting the development of the website, it is necessary to define the back-end first. Term backend refers to the database design which stores the information of users and their records and ER diagram is the starting point of any database design. Purposed ER diagram of this project is as under:

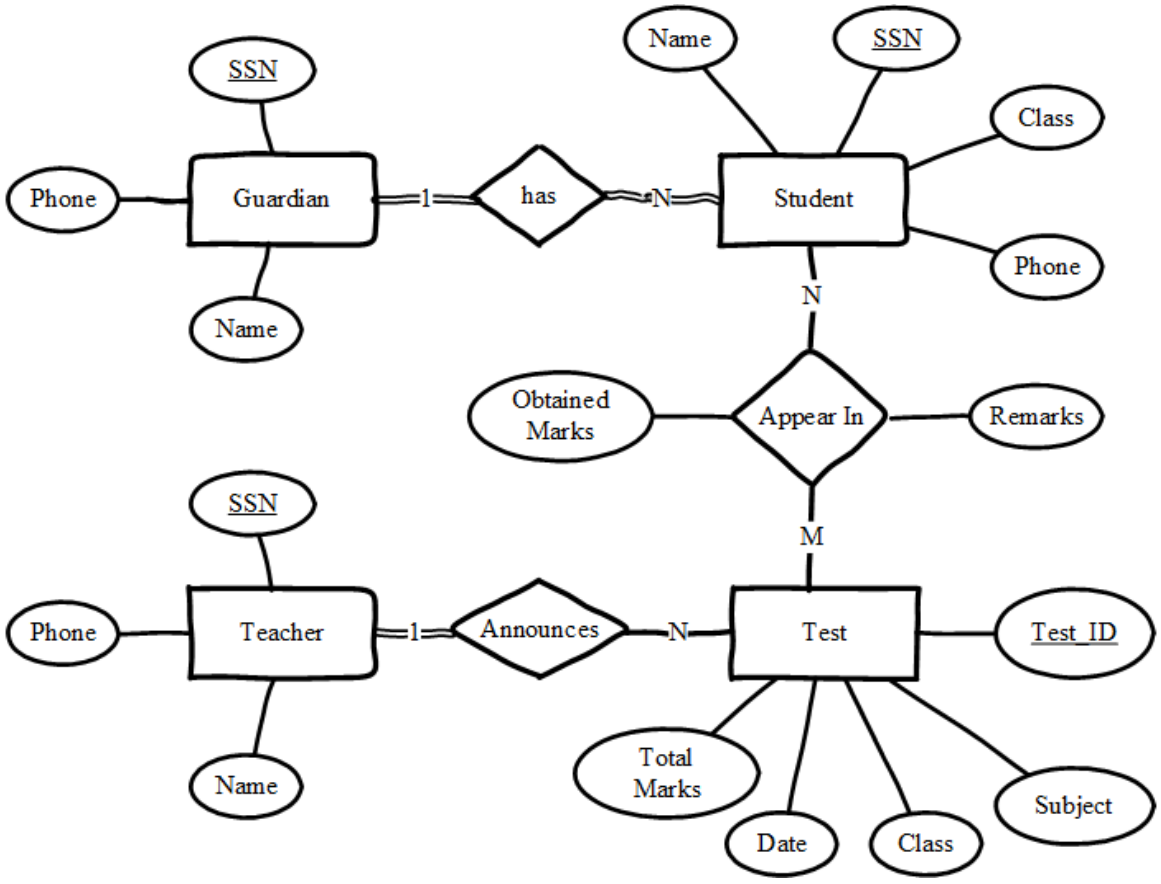

FIGURE 2.1: Proposed ER Diagram

In the ERD shown in2.1, it can be seen that there are four entities. Three of them are for users(Student, Parent, Teacher) while one is for record-keeping that is Test.
ER Diagram can be explained by starting from a teacher who announces a test and updates the marks of students who appeared in the test. Each student is enrolled with the SSN of his/her guardian so the teacher can get the contact details of his student's guardian by querying the details of the student. Similarly, the guardian can get the contact details of the teacher who marked the test of his child from test details. Each test has a unique ID which is the key of the Test entity. This key is helpful to fetch the details of each test and the teacher who announced the test. The relationship "Appears in" is a many to many relationship between student and test containing details of obtained marks and remarks for each test of each student.
Following constraints are applied to design the system:

- The teacher can only update marks and reviews of the tests he/she announced.
- A student can see only his marks and reviews.
- The guardian can see the result and reviews of all his children.
- Only the teacher has the right to update marks and reviews.
- The guardian can see the contact details of the only teachers who marked his children's test.
- The teacher can see the contact details of the students and their parent who appeared in his/her test.
- Not everyone can create a teacher account. Only a superuser can authorize the teacher.

2.2 Data Flow Diagram

After a successful ER diagram design, the next step is to analyze the flow of information within different users. For this purpose, a data flow diagram is designed which shows graphically the flow of data or information and represents the necessary steps to perform transactions. The DFD designed for this project is as shown:

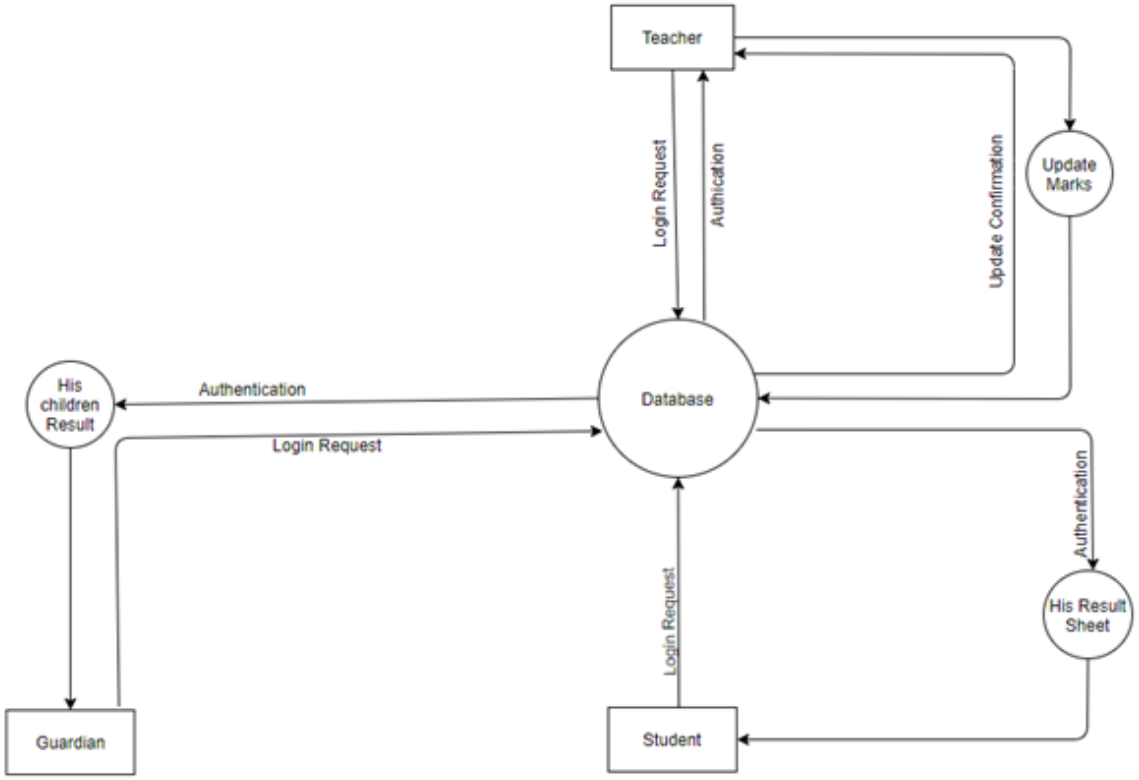

FIGURE 2.2: Data Flow Diagram

Since each type of user has a different view, moreover, each user of the same type has different data so it requires an authentication system i.e. each user will have his own user account. So before using the service, the user has to register an account and log into the account to view the result. Firstly, the user sends a login request to the database which checks the username and password, if it matches, the user will be logged in no matter he is a student, teacher or guardian. Each will have the same login form but after they have logged in, they have their corresponding view as shown in DF Diagram 2.2

Chapter 3

DESIGN IMPLEMENTATION

3.1 Software and Tools

It is necessary to select and set up the right software and tools to implement a design. Following are the tools and software used, and the reason why they are used for this project:

3.1.1 Framework

There are numerous frameworks available for web development like Django, ReactJS, WordPress, PHP etc. but we preferred Django due to the following reasons:

3.1.1.1 Easy to Use

Django is very easy to use because it uses Python programming language for coding and HTML for templates. It gives a very convenient method to render the HTML template with Jinja support by providing a lot of built-in functions and libraries. Moreover, Python is a high-level Object Oriented Programming language that is very easy to understand and learn. Nowadays, there is hardly any developer who does not code in python. Python is the most popular and most used language so by using it, the developer attaches himself to the largest community of developers.

3.1.1.2 Firm Control Over Database

Unlike other web frameworks like WordPress, Django provides a firm grip over database design as a Model class of Django where a developer can design and query the database according to his own will, refer Foreign Keys to other tables, set primary key according to the design and many more.

3.1.1.3 Easy to Find Help

Django has very brief and concise documentation where one can easily find a solution to his problem. Moreover, since it uses Python so there is a great community available to take help and increase our knowledge. A lot of help is already available on Stack Overflow.

3.1.1.4 Default Authentication System

Django comes with a very awesome and secure authentication system which makes account creation, validation, and authentication quite handy. It automatically manages the hashing of a password and matching it against a specific user. It comes with good support of cookies which when stored in a browser, automatically logs a user into his account no matter the session is cleared.
Moreover, this authentication system is very easy to customize if a developer needs more functionality with authentication.

3.1.1.5 Default Admin Section

Another reason to use Django is its default admin section where a developer can easily view, manage and manipulate database contents without having to worry about creating the templates.

3.1.2 Integrated Development Environment (IDE)

The IDE used for this project is *Visual Studio Code* also called *VS Code* is one of the best code editor, debugger integrated with IntelliSense and support for almost all the languages with the help of a vast collection of extensions supporting the language. VS Code is the best IDE for Python programming because of its unmatched extension of *Pylnace* which detects the patterns and behaviour of code, auto imports the required package, suggests auto-complete of variables, and displays recommended functions and attributes of a variable by detecting its data type and class with its extraordinary IntelliSense.

3.2 Customizing the User Model

Since Django comes with its default user authentication system, so it becomes very easy for developers to make an authentication system with almost no effort. But in this project, each user has the same login page so customizing the default Django authentication system makes the design very efficient and powerful to implement. This project requires somehow add another field/attribute to the default user's model which will differentiate the user types.
There are two ways to add a field to the authentication table of Django. One is to extend the "AbstractBaseUser" class and creating our own custom user. Second is to extend the "AbstractUser" class which keeps the default fields as they are, and adds more fields customized by the developer. The only precaution to use these both types of customization is to make it at the beginning of the project. Once migration is made, then there is no chance to apply this customization. The only way to work out is by deleting the previous migrations and database and customize the user model.

3.2.1 Custom User Model

The second type of customization is used in this project where all the default fields of the user model are kept and a new field with the data type of "Small Positive Integer" is added which is mapped to three user types as follows:

Integer Value	User Type
1	Student
2	Parent/Guardian
3	Teacher

TABLE 3.1: Lookup table of user type field in Custom User

Python code to customize the user model:

```
from django.db import models
from django.contrib.auth.models import AbstractUser
# Customizing User Model

class CustomUser(AbstractUser):
    USER_TYPE_CHOICES = (   # choice type field can have one value at a time
        (1, 'student'),
        (2, 'parent'),
        (3, 'teacher')
    )
    user_type = models.PositiveSmallIntegerField(
        verbose_name='Profession', choices=USER_TYPE_CHOICES,
        blank=True, null=True) # blank=True means field can be left null
```

Here, blank=True means the field can have a NULL value in the database, it is so because when a teacher signs up, his data will be stored in the database but he will not get privileges because this field will be NULL. As soon as the admin approves the teacher, the user_type variable will be set to 3, i.e. Teacher and he will get his privileges.

3.2.2 Custom Admin Section

Although the field has been added to the table, yet it will not display in the admin section of Django because the admin view of Django has default fields. So this section also needs modification to display additional field as shown:

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .customuser import CustomUser


class CustomUserAdmin(UserAdmin):
    model = CustomUser
    fieldsets = (
        *UserAdmin.fieldsets,  # original form fieldsets, expanded
        (                      # new fieldset added on to the bottom
            # group heading of ``Professional Info''
            'Professional Info', {'fields': ('user_type',)}
        )
    )
admin.site.register(CustomUser, CustomUserAdmin)
```

After registering the Custom User with an additional field, here is how this field will look like:

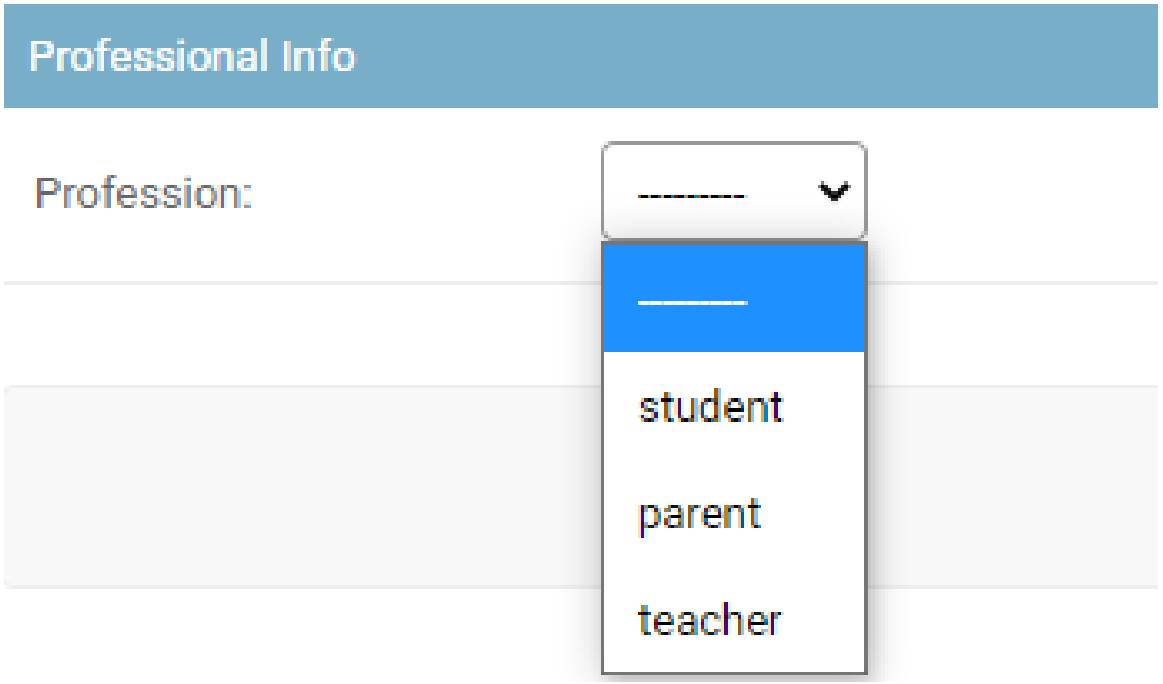

FIGURE 3.1: Custom User with additional field

Furthermore, this customization also helps the system to keep attackers away from accessing private information and differentiates quite strongly between student, parent and teacher account.

3.3 Models and Their Fields

A good ER diagram design can be translated into a relational schema which is very useful to create database tables and their attributes. In Django, the Model class takes place of the table and the field takes place of an attribute.
Two apps are created to make this project, one named as users which handles the details of users (students, parents, teachers) and the other named as test_conducts which handles the details of tests conducted by teachers. According to relational schema, there were a total of five tables that are

- Student (an entity): 4 attributes, 1 foreign key
- Parent (an entity): 3 attributes, 0 foreign key
- Teacher (an entity): 3 attributes, 0 foreign key
- Tests (an entity): 5 attributes, 1 foreign key
- Conduct (an N-to-M relation): 2 attributes, 2 foreign keys

All the user tables (Student, Parent, Teacher) has primary key "ssn" and each tuple of these tables are supposed to be referring to their corresponding credential information in the Custom

User table via a foreign key. This table is created by Django itself to store information related to user accounts and their credentials. The key attribute of this table is "id" which is an auto incremental integer, hence unique for each user.
In order to make things simple, the "ssn" of each user is referred to as the "id" of Custom User as a foreign key and this is made the primary key in each table. In this way, the problem of attaching a foreign key and the primary key is solved in one step. Declaration of Models is as under.

For Student:

```
class Student(models.Model):
    name = models.CharField(verbose_name='Full Name', max_length=30)
    parent_id = models.ForeignKey(to=Parent, verbose_name='Parent/Guardian',
        on_delete=models.CASCADE, to_field='ssn', null=True)
        # null=True to make field empty in the beginning
    ssn = models.OneToOneField(to=CustomUser, on_delete=models.CASCADE,
        primary_key=True, verbose_name='Student ID', to_field='id')
    standard = models.IntegerField(verbose_name='Class')
    phone = models.CharField(
        max_length=12, verbose_name='Phone No.', blank=True)

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

For Parent:

```
class Parent(models.Model):
    name = models.CharField(verbose_name='Full Name', max_length=30)
    ssn = models.OneToOneField(to=CustomUser, on_delete=models.CASCADE,
        verbose_name='Parent/Guardian ID', primary_key=True, to_field='id')
    phone = models.CharField(max_length=12, verbose_name='Phone No.')

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

For Teacher:

```
class Teacher(models.Model):
    name = models.CharField(verbose_name='Full Name', max_length=30)
    ssn = models.OneToOneField(to=CustomUser, on_delete=models.CASCADE,
        verbose_name='Teacher ID', primary_key=True, to_field='id')
    phone = models.CharField(max_length=12, verbose_name='Phone No.')

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

For Tests:

```
class StudentTests(models.Model):
    test_id = models.AutoField(verbose_name='Test ID', primary_key=True)
    date = models.DateField(verbose_name='Test Date', auto_now_add=True)
```

```
    total_marks = models.IntegerField(verbose_name='Max Marks', default=100)
    subject = models.CharField(verbose_name='Subject Name', max_length=20)
    teacher_id = models.ForeignKey(to='users.Teacher',
        verbose_name='Teacher ID', to_field='ssn', on_delete=models.CASCADE)
    standard = models.IntegerField(verbose_name='Class', blank=True)

    def __str__(self):
        return 'Test # ' + str(self.test_id)
```

For Conduct:

```
class Conduct(models.Model):
    obtained_marks = models.IntegerField()
    student_id = models.ForeignKey(to='users.Student',
        verbose_name='Student ID', to_field='ssn', on_delete=models.CASCADE)
    test_id = models.ForeignKey(to=StudentTests,
        verbose_name='Test ID', to_field='test_id', on_delete=models.CASCADE)
    remarks = models.TextField(verbose_name='Remarks', null=True, blank=True)

    @property
    def percentage(self): # derived attribute
        return (self.obtained_marks)/(self.test_id.total_marks) * 100

    class Meta:
        # Django model CANNOT have two primary keys, so need to be declared
        unique separately
        unique_together = (('test_id', 'student_id'),)

    def __str__(self):
        return f'{self.test_id} of {self.student_id}'
```

In all these Models, class word is replaced with standard because class is a keyword of Python. ForeignKey command is used for 1-to-N relation and OneToOne command is used for 1-to-1 relation. The phone number for a student is not compulsory so, blank is set True for this field.

Percentage attribute in the Conduct table is derived attribute whose need was aroused later on while working with student's tests report.

3.4 Django Views and Templates

Since Django follows the model-template-views architectural pattern, so it comes with built-in view functionalities where content can be shown, manipulated and styled on the front-end with the help of templates which can be rendered very easily by just calling render function. The argument of this function is the request, followed by the template name and followed by the dictionary of data to be used in the template.
Since there are two apps, so two view files are implemented. One for users app and the other for test conduct app. View file in the "users" app handles the

- Sign up

- Login
- Logout
- Approving or deleting newly signed up teachers (Superuser's job)

of user accounts.
View file in test conduct app handles

- Registering a test (for teacher)
- Adding marks of students for their tests (for teacher)
- Viewing all the tests with obtained marks. But these views are different for each type of user. For example:
  - Teacher will see the only tests he conducted
  - Student will see only his tests
  - Parent/Guardian will see the tests of only his children
- Viewing the complete record of a student. But it also has restrictions like:
  - Student can see his report only
  - Parent can see the report of all of his children
  - Teacher can see the report of a student only for the tests he marked
- Viewing contacts. The constraints on contact info are:
  - Student can see contact details of only those teachers who conducted his tests.
  - Parent can see contact details of the teachers who conducted at least one test of any of their children.
  - Teacher can see the contact details of the only student and his parent whose test is marked by him.

3.5 Database Queries

As we know in Django, tables are replaced with Models and attributes are replaced with fields, similarly, the query is replaced with an equivalent Python statement. In order to apply a query in Django, general syntax is:

```
# to retrieve all the elements of a table
<Model>.objects.all()

# to make a conditional query
<Model>.objects.filter(<condition>)

# to apply IN statement
<Model>.objects.filter(<field>__in=[<list>])
```

The return type of these queries is "QuerySet" and a specific field can be retrieved by using a for loop on QuerySet followed by a dot(.), followed by name of the field. E.g. in order to print the names of all the students from the student table, we can write a query like this:

```
students = Student.objects.all()    # retrieves all students

for student in students: # for loop on QuerySet will iterate through each tuple
    print(student.name)
```

Following major queries are made in this project:

3.5.1 Approve Teachers

Since the teacher has the authority to register and mark tests, moreover he has access to everyone's contact information, so not everyone should be allowed to make a teacher's account. However, we can apply a check and balance system, where only a superuser can give privileges to a teacher by approving him after verifying his details. The following query is written to retrieve the unauthorized teachers:

```
# retrieve the users from CustomUser table whose user_type is None
un_auth_users = CustomUser.objects.filter(user_type=None)

# retrieve the teachers whose ssn is matching with the id of users above
teachers = Teacher.objects.filter(
    ssn__in=un_auth_users.values_list('id', flat=True))
```

In the second query statement, the values_list function is used which retrieves a specific field of all the results passed as an argument to the function and "flat=True" is getting elements from all the tuples and making a flat list. These elements should be in a straight list because it is the requirement of ¡field¿__in statement.

3.5.2 Get Results

Before making a query to fetch the results of the student tests, we need to understand the structure of Django request for logged in users and logged out users.
When a user is logged in, every request sent by that user will contain the details of that user which can be accessed as an attribute of request and "request.user.is_authenticated" will yield True. However, when a user is not logged in, the request object will use "AnonymousUser" which is set default by Django and "request.user.is_authenticated" will yield False.

Since we have different views for different type of users, so we first check the type of user sending request and send data accordingly. Here is its implementation:

```
if request.user.user_type == 1:      # for student
    tests = Conduct.objects.select_related(
        # select_related is equivalent to joining on foreign keys
        'test_id').filter(student_id=request.user.id)

elif request.user.user_type == 2:    # for parent
    tests = Conduct.objects.select_related('test_id', 'student_id').filter(
        student_id__parent_id=request.user.id)
```

```
9
10 elif request.user.user_type == 3:   # for teacher
11     all_tests = Conduct.objects.select_related('test_id').all()
12     tests = all_tests.filter(test_id__teacher_id_id=request.user.id)
```

In these queries, the select_related function is used to join Models on foreign keys where its argument is the name of the field which is a foreign key to the other Model. This joining not only helps extend the Model to get details of other model but also reduces the number of database hits and therefore, reduces the execution time of the query as described in the article [1]. The query for a student is straight forward where we match the id of logged in user with the student id, i.e. id present in the custom user table and retrieve the details of tests.
The query is a little complicated when Parent/Guardian is logged in. The query is applied on Conduct Model which is then joined with the Student model and Tests model. After that, the condition is applied to the Student model to retrieve the details of all the students whose parent_id is matching with the id of the logged-in user.
Lastly, a query for a teacher is written where the Conduct model is joined with the Tests model and the record is saved in a temporary variable named "all_tests". After that, a condition is made on this QuerySet to select only tests which are registered by the logged-in teacher.

Every time query is made, final result is stored in variable named "tests" which is then passed to template as Python dictionary after ordering them by test_id as shown:

```
1 return render(request, 'tests.html', {'tests': tests.order_by('test_id')})
```

3.5.3 Result Details

When tests are shown to the user, there are three links present on each test. One for test details, the second for Student Report and the third for Teacher's details who marked that test.

- Test ID shows details related to the test like date, teacher, total marks and subject.
- Student's report can be divided into two parts.
  - In the first part, personal details and overall performance is displayed.
  - In the second part, all the tests with marks and remarks are displayed.
- Teacher details contain name and contact of teacher which will be displayed to the student and his guardian.

Chapter 4

FRONT-END AND HOSTING

4.1 Front-End

After the backend is designed, the next challenge is the frontend. For a developer, the frontend might not be as important as the backend but it does matter when the website is intended to be hosted live on the globe. Every type of person will be visiting the site and a good frontend comes before the service. That is something the customer will be going to judge the developer on. If the interface is not proper, it becomes hard to attract customer. So, a good frontend is as necessary as a good backend.
Generally, the frontend of a website contains three major parts. i.e. HTML, CSS and JavaScript. In this project, JavaScript and CSS are selected with the help of bootstrap v5. Follow the URL to study more about bootstrap: https://getbootstrap.com/
Bootstrap is used for this project due to the following reasons:

4.1.1 Easy to Implement

Styling with bootstrap is the easiest way to design the frontend. All we had to do was adding a URL that automatically adds and loads a CSS file to the template. Moreover, URLs are available for JS also which can be simply pasted to the template and they will be downloaded automatically.
Furthermore, bootstrap makes it very easy for a developer to do styling like using a different colour, using tables or making elegant forms just by adding a class in the tag of HTML. For example, if a developer wants to make a striped hovering table on the frontend, he only needs to add *class="table table-striped table-hover"* in the table tag and boom! it will be styled automatically. Now if the user wants the same styling and change the colour to green, all he needs to do is write a class like *class="table table-striped table-hover table-success"* and that's all.

4.1.2 Security

The first negative thought that comes to mind after going live is the protection and integrity of data. No developer will desire his system to be attacked or hacked by any means. While using

built-in templates, there is always a chance that the designer of that template has a back door and can read the private information of a user. But in the case of bootstrap, the developer does everything himself so he is sure that his templates do not have any back door where his private information can be accessed.

4.1.3 User-Friendly Documentation

Another reason to use bootstrap is its vast collection of documentation which is documented so nicely that any layman can style a website elegantly.

4.2 Hosting the Website Live

After the successful design of the website on localhost, it becomes ready to be hosted live so other people can see the appreciate the hard work and effort done for the project. There is no reason to work so hard on a project which will be abandoned in future.
There are many hosting services available for deploying website but most of them are quite costly. As a student, it is our first priority to be in the limits of the economy. So PythonAnywhere is chosen to deploy the project. Following are the good reasons to choose PythonAnywhere:

- It is **free** with limited services. But as far as this project concerns, its functionalities can be summed up in free service
- Automatic settings for Python projects. PythonAnywhere is based on the Python programming language, so they provide a lot of built-in python functionalities which may become hectic and difficult to set up manually.
- Already installed different versions of Python along with pip having path is already added to environmental variables
- Easy debugging is also a good feature of PythonAnywhere. Since PythonAnywhere is dedicated to Python projects, so it comes with built-in tools to debug Python apps

4.2.1 Challenges Faced in Deployment

Generally speaking, the project is full of challenges but those challenges come with their solution already. Hosting the project was something new so the challenges faced in it are:

- **Static Files**: Static files in Django are separable so their path is required to be set and added to the corresponding URL of static files. Statics are kept separable in Django because putting the database and static files on the same server may slow down the service for the queries which require a large number of database hits. So Django gives the option to put static files somewhere on another server so it can load contents in parallel which will reduce the processing time.
- **Setting up WSGI File**: WSGI stands for *Web Server Gateway Interface* which is essential to deploy a project live. For a sophomore developer, it is difficult to understand its purpose and make its settings according to the project requirement.

Chapter 5

CONCLUSIONS AND FUTURE PLANS

5.1 Conclusion

It was a very fruitful project which not only taught us the applications of the database in the market but also increased our interest in the subject. This project demonstrated all the theoretical concepts with practical real-world examples and applications strengthen our concepts further by teaching us different ways of storing and manipulating the data in the database.

5.1.1 Data Is the New Oil

"Data is the new oil". Data has become so powerful resource that is as expensive as oil. As the world is moving towards technology, the demand for data is increasing so and so forth. With the increase of data and the broadening of technology, comes the challenges of attacks and threats. In fact, attackers are always one step ahead of the designer. So the market is always in need of talented database engineers who can come up with new ideas and ways to secure the data.

5.2 Future Directions

For the time being, Teacher-Parent Interactive LMS has been launched successfully and working fine with the design constraints but *"There is always room for improvement"*. As long as design concerns, the following are the ideas to work on:

- Teacher can only add marks of tests. Once added, he cannot edit those marks. So the system can be designed to have edits and store update the database accordingly.
- Currently, the student cannot see all the registered tests of his class. He can only see the marked tests. So, we can add the functionality that students can see their upcoming tests with their dates.

- Guardian and student can only see the contact number of teacher, but cannot contact him on the website. Rather, they are forced to use an external communication channel. We can integrate messaging service inside the website so they can communicate easily on one platform.
- Student and Guardian don't get a notification if a new test is registered or marked. So we can implement a notification system or integrate a text messaging service so they will receive a text message every time a new test is registered or marked.

There can be a lot of suggestions but they all are pointless until this platform is used by some service. So, it is our plan to implement this system in local schools and colleges where old-fashioned and traditional ways are used for communication or sometimes teachers and parents cannot find time in their busy routine for a meeting. This platform will be handy for them for the better future of their students and children.

References

[1] Goutom Roy. Django select_related and prefetch_related. https://betterprogramming.pub/django-select-related-and-prefetch-related-f23043fd635d, 2019.