QUIZ USING DJANGO

END TERM REPORT

By

JASMINE KAUR SAINI, KRATI GARG, JYOTI SWAIN

REGISTRATION NO.: 11901155, 11913716, 11905266

ROLL NUMBERS:35, 48, 58

Section: K19JP

Department of Intelligent Systems,
School of Computer Science Engineering,
Lovely Professional University, Jalandhar

ABSTRACT

This report introduces the process of creating part of a quiz application which is a data-driven website used by the admin and students. This website has four major components: login, quiz creation, quiz taker, and results. The major part of the group component and minor part of marking component are implemented by us and the implementation details will be introduced in the report. The implementation uses a tool called Django Framework which is an excellent open source web application frame work for complex data-driven website development. The major part of this report will introduce how to use Django to create a web page user interface and inner logic to handle user request by going through the group component implementation process.

SUMMARY

This is a quiz application using DJANGO, PYTHON, HTML, CSS etc.

Current Features

----

*Login page

*Sign up page

*If anyone will try to login without signup then they will be directed to sign up page

*Types of quiz will be sorted

*Sorting of quiz results

*Quiz will be in the form of MCQ

*Custom message will be displayed after submission of each question

*Customise own quiz

*Scoreboard

ACKNOWLEDGEMENT

We would like to thank our Python teacher, Mr. Sagar Pandey for their guidance and support in completing our project. Though it's the time of pandemic, still we are able to complete it.

Special thanks to sources we learnt from like YouTube, Google, Bootstrap and My Class.

Date: 25/10/2020

Submitted by

Name: Jasmine Kaur Saini, Krati Garg, Jyoti Swain

Registration No.: 11901155, 11913716, 11905266

Roll Numbers:35, 48, 58

TABLE OF CONTENTS

1. INTRODUCTION

We created the quiz model to store the name of the quiz and virtually all the data related to the quiz. Each Question is linked to an individual Quiz in the sense that every question has its label i.e. the question, and the quiz with which it belongs to.

Description of requirements/features: Create quiz- user should be able to create quiz with objective type (multiple choice) and long answer type questions
can choose number of questions that should be in the quiz (which will be picked up randomly from a pool of questions from the same quiz category)
should be able to categorize quiz
add quiz title
manage created quiz(s).

There will be login and sign up page. We can create the quiz and giving them a particular quiz name. The quiz will be in the form of MCQ. A submit button will be there below every question. At the end of quiz there will be score board. The participant with highest score will be placed at the top.

2. PRELIMINARY

2.1 DJANGO FRAMEWORK

Django is an open source web application frame work written in Python. The primary goal of Django is to make the development of complex, data-based websites easier. Thus, Django emphasizes the reusability and pluggability of components to ensure rapid developments. Django consists of three major parts: model, view and template.

2.2 MODEL

Model is a single, definitive data source which contains the essential field and behaviour of the data. Usually one model is one table in the database. Each attribute in the model represents a field of a table in the database. Django provides a set of automatically-generated database application programming interfaces (APIs) for the convenience of users.

2.3 VIEW

View is short form of view file. It is a file containing Python function which takes web requests and returns web responses. A response can be HTML content or XML documents or a "404 error" and so on. The logic inside the view function can be arbitrary as long as it returns the desired response. To link the view function with a particular URL we need to use a structure called URLconf which maps URLs to view functions.

2.4 TEMPLATE

Django's template is a simple text file which can generate a text-based format like HTML and XML. The template contains variables and tags. Variables will be replaced by the result when the template is evaluated. Tags control the logic of the template. We also can modify the variables by using filters. For example, a lowercase filter can convert the variable from uppercase into lowercase.

2.5 PYTHON

Python is the language used to build the Django framework. It is a dynamic scripting language similar to Perl and Ruby . The principal author of Python is Guido van Rossum. Python supports dynamic typing and has a garbage collector for automatic memory management. Another important feature of Python is dynamic name solution which binds the names of functions and variables during execution.

3. WORKING OF PROJECT

3.1 LOGIN

To log a user in, from a view, use **login()**. It takes an **HttpRequest** object and a **User** object. **login()** saves the user's ID in the session, using Django's session framework.

```
From django.contrib.authimport authenticate, login
def my_view(request):
    username = request.POST.get('username')
    password = request.POST.get('password')
    user = authenticate(username=username, password=password)
if user  is not None:
login(request, user)
# Redirect to a success page.
        ...
else:
# Return an 'invalid login' error message.
```

This code helps to authenticate and fetch the username and password entered by the user.

3.2 MODELS

A model is the single, definitive source of information about your data. It contains the essential fields and behaviours of the data you're storing. Generally, each model maps to a single database table.

- Quiz
- Question
- Quiz Taker

3.2.1 QUIZ

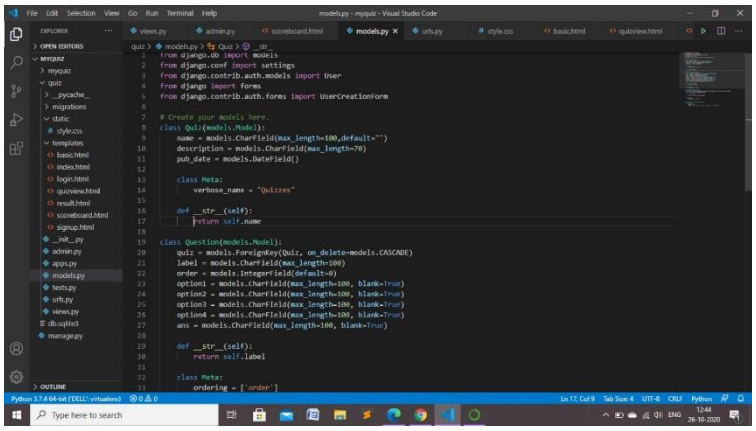
In this we are using the model to name the quiz like GK, History, Maths etc. Secondly it includes the description of the quiz. There will be a description box. Then one more feature of publishing date is available like when the quiz is published.

3.2.2 QUESTION

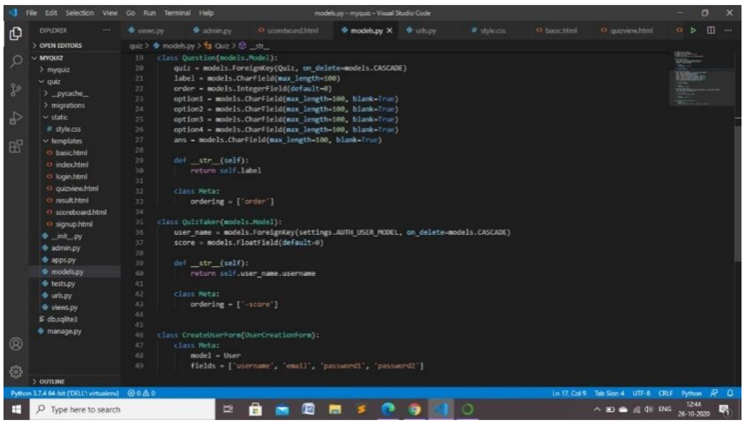
In quiz model, we are naming the category of the quiz but not including the questions by using a foreign key from quiz table.. So here using question model, we can draft our questions. First, we have to write the questions and enter the options and also store the right answer. Ordering feature is also available that in which questions are displayed according to the order.

3.2.3 QUIZ TAKER

In this the user who has taken the quiz, his information is stored . User’s name and Score are stored. - After the quiz the score board will be displayed and the one with the highest score will be displayed at the top.



```
from django.db import models
from django.conf import settings
from django.contrib.auth.models import User
from django import forms
from django.contrib.auth.forms import UserCreationForm

# Create your models here.
class Quiz(models.Model):
    name = models.CharField(max_length=100,default="")
    description = models.CharField(max_length=70)
    pub_date = models.DateField()

    class Meta:
        verbose_name = "Quizzes"

    def __str__(self):
        return self.name

class Question(models.Model):
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE)
    label = models.CharField(max_length=100)
    order = models.IntegerField(default=0)
    option1 = models.CharField(max_length=100, blank=True)
    option2 = models.CharField(max_length=100, blank=True)
    option3 = models.CharField(max_length=100, blank=True)
    option4 = models.CharField(max_length=100, blank=True)
    ans = models.CharField(max_length=100, blank=True)

    def __str__(self):
        return self.label

    class Meta:
        ordering = ['order']
```



```
class Question(models.Model):
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE)
    label = models.CharField(max_length=100)
    order = models.IntegerField(default=0)
    option1 = models.CharField(max_length=100, blank=True)
    option2 = models.CharField(max_length=100, blank=True)
    option3 = models.CharField(max_length=100, blank=True)
    option4 = models.CharField(max_length=100, blank=True)
    ans = models.CharField(max_length=100, blank=True)

    def __str__(self):
        return self.label

    class Meta:
        ordering = ['order']

class QuizTaker(models.Model):
    user_name = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    score = models.FloatField(default=0)

    def __str__(self):
        return self.user_name.username

    class Meta:
        ordering = ['-score']


class CreateUserForm(UserCreationForm):
    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']
```

3.3 TEMPLATES

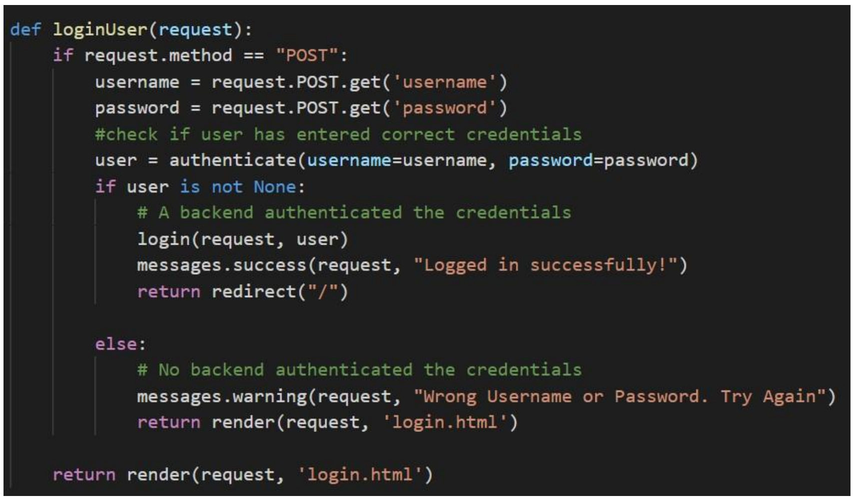
A template is a text file. It can generate any text-based format (HTML, XML, CSV, etc.).

A template contains **variables**, which get replaced with values when the template is evaluated, and **tags**, which control the logic of the template.

The most powerful – and thus the most complex – part of Django’s template engine is template inheritance. **Template inheritance** allows you to build a base “skeleton” template that contains all the common elements of your site and defines blocks that child templates can override. Bootstrap is the best website to get the codes of templets/buttons and thus making work little bit easier.

3.4 AUTHENTICATION

Django uses sessions and middleware to hook the authentication system into request objects. These provide a **request.user** attribute on every request which represents the current user. If the current user has not logged in, this attribute will be set to an instance of **AnonymousUser**, otherwise it will be an instance of User.
It helps to check whether the user is same who have registered in the quiz by matching their username and password.
Use **authenticate()** to verify a set of credentials. It takes credentials as keyword arguments, username and password for the default case, checks them against each authentication backend, and returns a User object if the credentials are valid for a backend. If the credentials aren't valid for any backend or if a backend raises **PermissionDenied**, it returns **None**.
In simple words, if a user tries to attempt a quiz by clicking on take quiz button who have not logged in my quiz then he will be directed to login page.

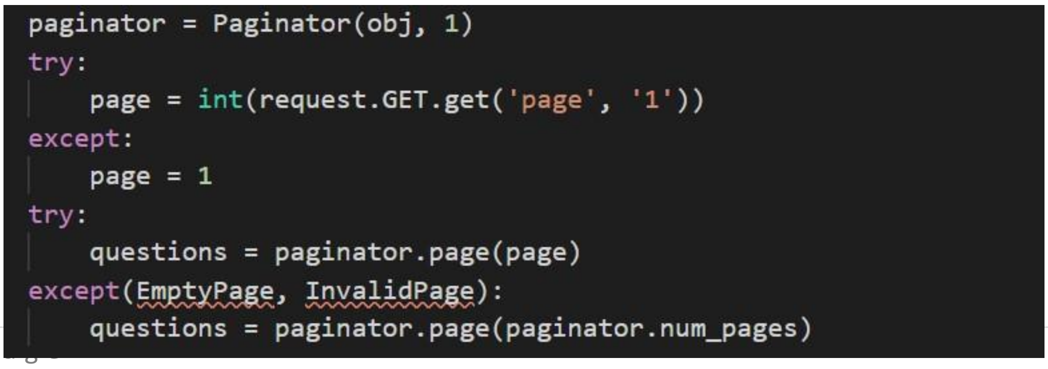


```
def loginUser(request):
    if request.method == "POST":
        username = request.POST.get('username')
        password = request.POST.get('password')
        #check if user has entered correct credentials
        user = authenticate(username=username, password=password)
        if user is not None:
            # A backend authenticated the credentials
            login(request, user)
            messages.success(request, "Logged in successfully!")
            return redirect("/")

        else:
            # No backend authenticated the credentials
            messages.warning(request, "Wrong Username or Password. Try Again")
            return render(request, 'login.html')

    return render(request, 'login.html')
```

3.5 PAGINATOR

Django provides high-level and low-level ways to help you manage paginated data – that is, data that's split across several pages, with "Previous/Next" links. It does all the heavy lifting of actually splitting a **QuerySet** into **Page** objects.

Here this code is displaying the questions in different pages.



```
paginator = Paginator(obj, 1)
try:
    page = int(request.GET.get('page', '1'))
except:
    page = 1
try:
    questions = paginator.page(page)
except(EmptyPage, InvalidPage):
    questions = paginator.page(paginator.num_pages)
```

3.6 LOGOUT

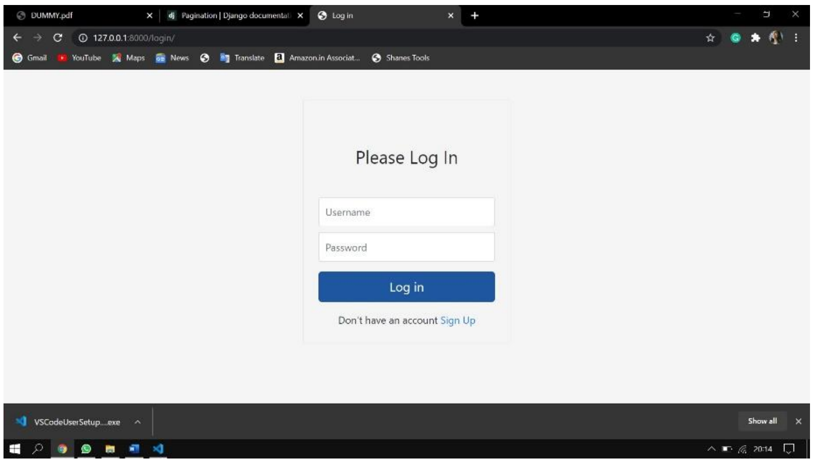
To log out a user who has been logged in via **django.contrib.auth.login()**, use **django.contrib.auth.logout()** within your view. It takes an HttpRequest object and has no return value.

```
from django.contrib.auth import logout

def logout_view(request):

logout(request)

# Redirect to a success page.
```

4. OUTPUT

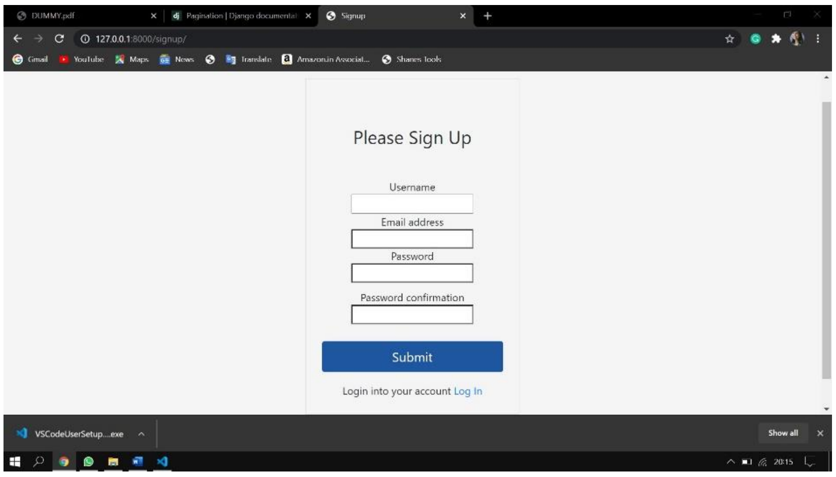
4.1 LOGIN PAGE

If anyone tries to attempt the quiz who haven't logged in, he will be directed to the login page. In views.py, there will be all the processing that the password and user name is in database or not.



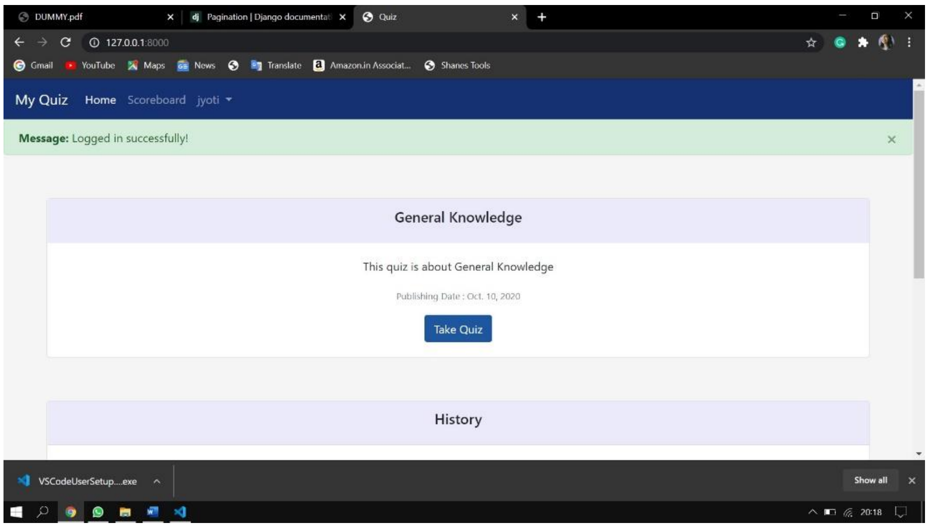
4.2 SIGN UP PAGE

If a user is new he have to sign up first.



4.3 HOME PAGE

After successful login the user is directed to this page.



Message: Logged in successfully!

General Knowledge

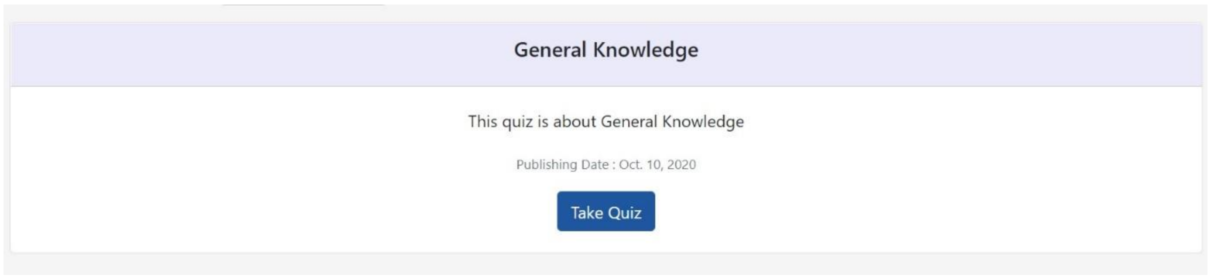
This quiz is about General Knowledge

Publishing Date : Oct. 10, 2020

History

4.4 QUIZ COLUMN

By clicking on the take quiz button of any quiz category he can attempt the quiz. Here information like quiz category, description of quiz and publishing date is mentioned here.

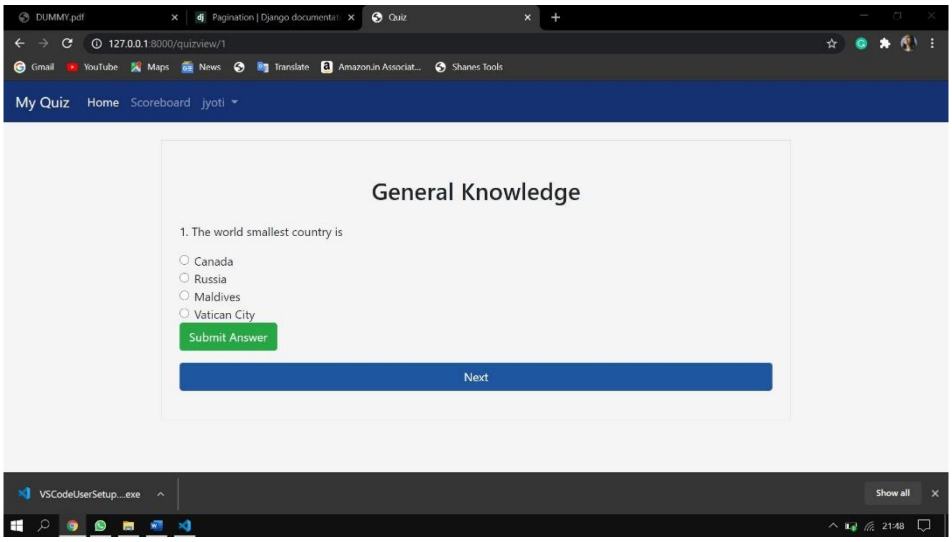

General Knowledge

This quiz is about General Knowledge

Publishing Date : Oct. 10, 2020

4.5 QUESTION

The questions will be displayed like this, and there are ratio buttons for selecting the options, after selecting the option the user had to click on submit, then on next. After this the next question will be displayed in next page.



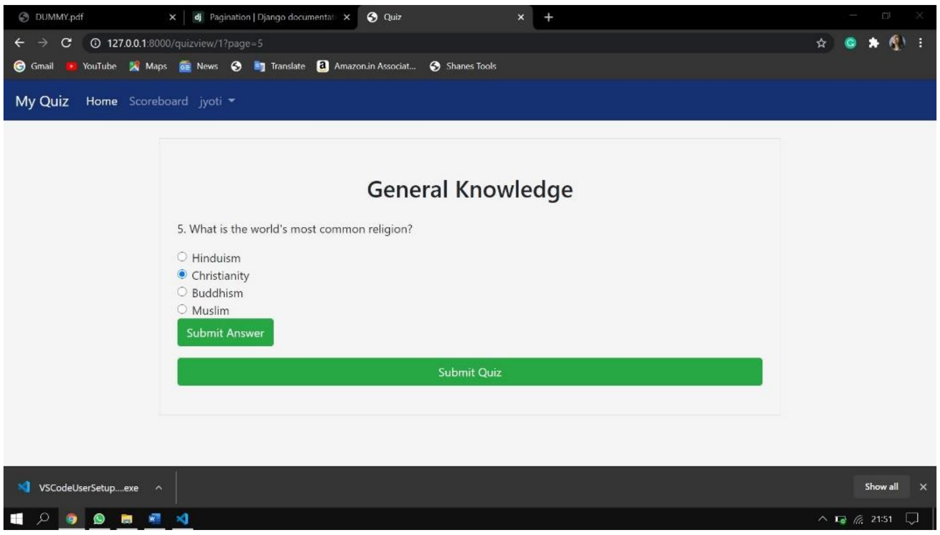
General Knowledge

1. The world smallest country is

- Canada
- Russia
- Maldives
- Vatican City

4.6 SUBMIT QUIZ

After attempting all the questions, there will be button of submit quiz.



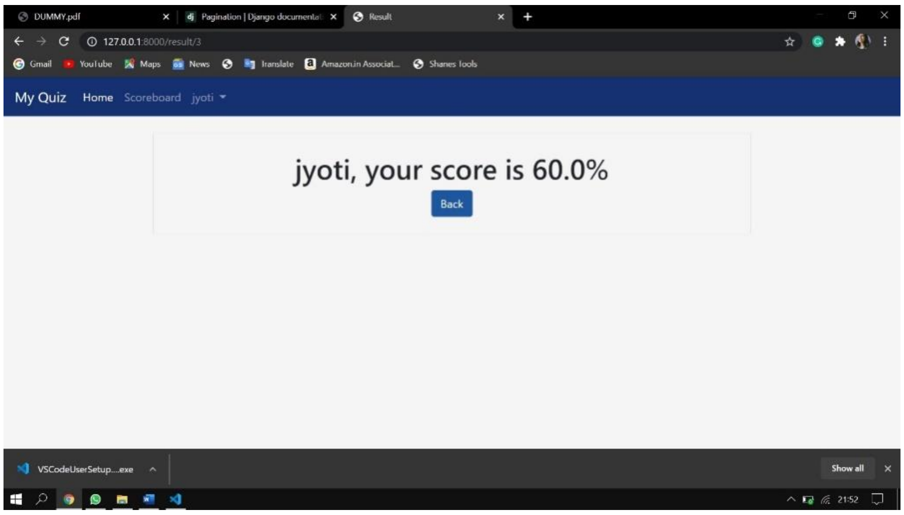
General Knowledge

5. What is the world's most common religion?

- Hinduism
- [✓ Christianity]
- Buddhism
- Muslim

4.7 SCORE

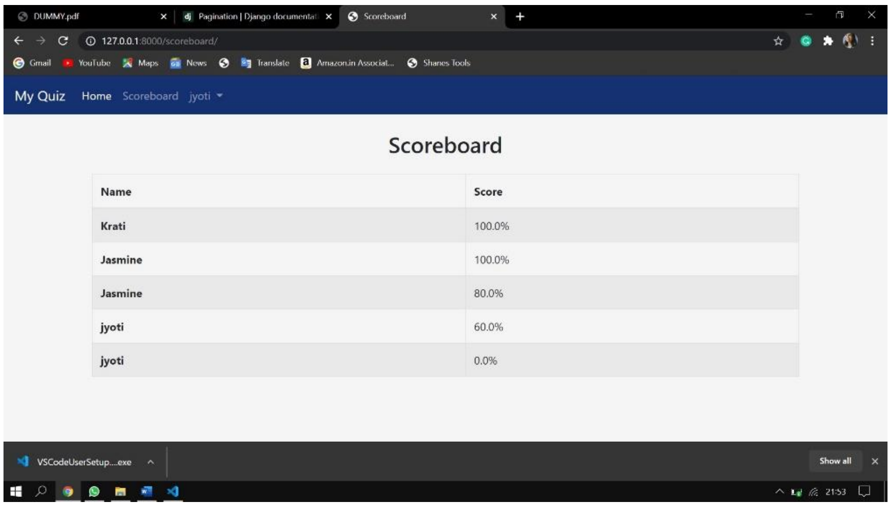
After submitting the quiz, the user can see his score. There is a particular id is given to a question category and the right answer so at the end of the quiz the id of submitted answers and the right answers is matched. Thus after all the processing the score is displayed.



jyoti, your score is 60.0%

4.8 SCOREBOARD

The users can see their names on the scoreboard, the one who scored the highest is displayed at the top.

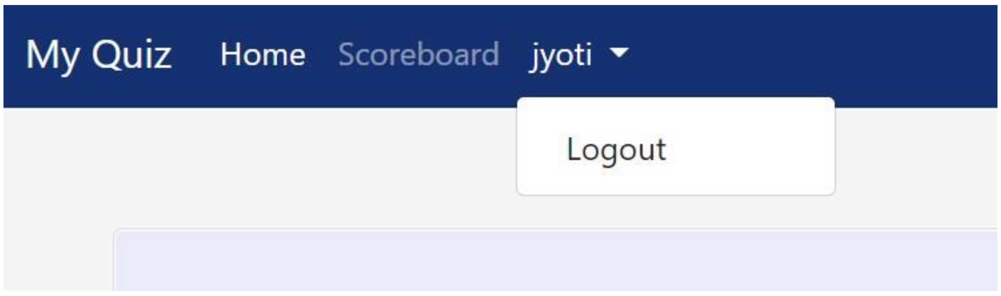


Scoreboard

Name	Score
Krati	100.0%
Jasmine	100.0%
Jasmine	80.0%
jyoti	60.0%
jyoti	0.0%

4.9 LOG OUT

At the top left there is a logout button, simply by clicking on that button, the user will be logged out and cannot attempt any quiz.



5. CONCLUSION

The Django framework gives us a simple and reliable way to create the course management system. It provides powerful functionalities and concise syntax to help programmers deal with the database, the web page and the inner logic. The experience of developing the group component in the system also helped me learning a lot of website development with Django. Within the Django framework, we have successfully accomplished the requirements of the system. It will make the work for instructors to manage the course much easier. In short, this system will bring great user experience to both admin and students. The only limitation for this course system is that although the developers have been testing it with various use cases, it may still encounter problems during real time use. However, even if that happens, the flexibility of Django would provide a simple way to fix the problem, as well as add new features into the system.

6. REFERENCES

[1] Django homepage. http://www.djangoproject.com/

[2] Python documentation. http://www.python.org/doc.

[3] Django (web framework). http://en.wikipedia.org/wiki/Django.

[4] Django documentation. http://docs.djangoproject.com.

[5] Authentication https://docs.djangoproject.com/en/3.1/topics/auth/default/

[6] Templates https://docs.djangoproject.com/en/3.1/ref/templates/language/

[7] Views https://docs.djangoproject.com/en/3.1/topics/http/views/

[8] Models https://docs.djangoproject.com/en/3.1/topics/db/models/