Lab 8 – Forms in Django

Marks = 2

Submission =

- **On Brightspace. Submit views.py, forms.py, urls.py and any template you create in this lab or update it from the previous lab.**

PART 2: Work with Form class

Update your app so that users can place orders for courses they would like to purchase.

1. Update **Course** and **Order** models.

   a. Add a new *PositiveIntegerField* called *interested* to the **Course** model. This field indicates how many people are interested in this course. The default value for the field is **0**.

   b. Add a *DecimalField* called *order_price* to the **Order** model. This field indicates the price of the ordered course.

   c. Add another *PositiveIntegerField* called *levels* to the **Order** model. This field indicates how many levels of a course the student wants to order.

   d. Create a method `discount(self)` for the **Order** model. When `discount(self)` is called, 10% off is applied to the price of the ordered course. The discount value is saved in the field *order_price* of the Order model.

   e. Run **makemigrations**, **sqlmigrate** and **migrate** to update the database with the new changes. In the admin page, create up to 10 orders for different courses by different students.

2. Create new forms.

   a. Create a new file *myappF23/forms.py* and add the following two lines:

```
from django import forms

from myappF23.models import Order
```

b. Create new form class **`class InterestForm(forms.Form):`**

The form should have 3 fields:

i) *interested*: Should display **RadioSelect** buttons (Yes/No). The value returned will be 1 for Yes and 0 for No.

ii) *levels*: Will accept an integer value of 1 or higher, indicating how many levels of a course the student is interested in. Initial value is set to **1**.

iii) *comments*: An optional input using **Textarea** widget and label = '**Additional Comments**'

3. Create new ***courses*** **view.** This view will send a list of courses in the database to *courses.html*. The template will display the list of the courses and also have a link (**url** */myappF23/place_order*) to place a new order.

a. Edit your *views.py* file as follows:

```
# Import necessary classes and models

# Create your views here.


def courses(request):
    courlist = Course.objects.all().order_by('id')
    return render(request, 'myappF23/courses.html', {'courlist':
courlist})
```

b. Update *myappF23/urls.py* so that going to **url** *myappF23/courses/* will call the view function **`views.courses`**.

c. Create the template *courses.html* in *myappF23/templates/myappF23* dir. This template should display the list of available courses.

d. Update *base.html* to add a link to **url** *myappF23/courses/* in addition to the main (index) and about page.

PART 2: Work with ModelForm class

1. Add new form: Create a new form class: **`class OrderForm(forms.ModelForm):`**
   a. *OrderForm* should be a form based on the **Order** model.
   b. The form fields should include *student*, *course*, *levels*, and *order_date.*
   c. Set the widget for *student* field to **RadioSelect**.
   d. Set the widget of *order_date* field to '**SelectDateWidget'**

2. Define another view function `place_order(request)` in your *views.py* file.
   a. When the user goes to url `myappF23/place_order` it should display a list of courses in the database along with their prices and provide a form for the user to place an order for a *course*.
   b. Update *myappF23/urls.py* with the suitable url pattern for `place_order` view.
   c. Here is an initial version of `place_order(request)` view function, you'll update it in the following questions.

```
def place_order(request):
    msg = ''
    courlist = Course.objects.all()
    if request.method == 'POST':
        form = OrderForm(request.POST)
        if form.is_valid():
            order = form.save()
            msg = 'Your course has been ordered successfully.'
        else:
            msg = 'You exceeded the number of levels for this course.'
            return render(request, 'myappF23/order_response.html', {'msg': msg})
    else:
        form = OrderForm()
```

```
    return render(request, 'myappF23/placeorder.html', {'form':form, 'msg':msg,
'courlist':courlist})
```

d. Create the template *placeorder.html* in *myappF23/templates/myappF23* dir. This template should display the list of courses and render the form you created in *place_order* view. Also, create the *order_response.html t*o display the above message *msg.*

e. Update the `place_order(request)` function in *views.py*, so that after an order has been successfully placed and saved in db, if the course price is greater than $150.00, the function `discount()` should be called to update the price of the corresponding order.

f. Update the `place_order(request)` function to check if the students ordered a course levels that exceeds the number of levels for that course in the Course model. This may involve changing the types of the levels field and values in the Course and Order Models.

g. Update *courses.html* to add a link to *place_order* page using **url namespacing** tag.

3. ***coursedetail* view**. Define another function `coursedetail(request, course_id)` in your *views.py* file. When the user goes to **url** `myappF23/courses/<cour_id>` it should take you to *coursedetail.html* to display *name*, the number of people interested in the course so far (*interested*), and the price of the course (*price*). It should also use **InterestForm** to allow users to indicate their interest in the course.

a. Update *courses.html* so that by clicking on a course name, it should bring the user to **url** `myappF23/courses/<course_id>`

b. Update *myappF23/urls.py* with the suitable url pattern for `coursedetail` view.

c. If the page is called using GET display an unbound **InterestForm**. If it is called using POST, validate form. If form is valid and *interested* is 1, increment *interested* field for the course then save the updated *course* object and redirect to the main *index* page.