Day 2

Django

Open Source - Alexandria

Agenda

- 404 Not Found
- Template Inheritance
- Form
- CRUD
- Static files
- Bootstrap
- Git

404 Not Found Error Handling

One of the basic and common error is (404 Not Found)
example: in the ***view.py***

```
from django.http import Http404
from .models import Student

def details(request, student_name):
    try:
        st = Student.objects.get(first_name = student_name)
    except Student.DoesNotExist:
        raise Http404("Student Not Exist!")
    else:
        return render(request, 'My_App/details.html', {'student': st})
```

404 Not Found continue

Considering that get a model object or 404 in a common and basic used job Django introduced a convenience method to speed it up So the previous code can be replaced with single method

get_object_or_404(Model, Condition)

```
from django.shortcuts import render, get_object_or_404
from .models import Student

def details(request, student_id):
    student = get_object_or_404 (Student, id = student_id)
    return render(request, 'App/detail.html', {'student': student} )
```

Template Organization

- **Include** ⟶ get another template and put it inside the current template.
- **extends** ⟶get another template defined blocks to be replaced in current.
- **block** ⟶ the template defined in a block tag can be extended.

Hello Re Usability , Hello Modularity

Note: include, and extends sees from templates directory So even if the template files are in the same dir you will need to write the inner folder under templates directory if exists when extends or include.

Template Organization example

create file ***base.html***

```
<html>
    <head> </head>
    <body>
        {% block main_block %}
        {% endblock %}
    </body>
</html>
```

in our *templates.html* we extends *base.html* for example in *details.html*

```
{% extends “innr/base.html” %}
{% block main_block %}
<h1>{{student.student_name}}</h1>
<ul>
    <li>{{student.age}}</li>
    <li>{{student.email}}</li>
</ul>
{% endblock %}
```

Template Organization example Continue

create file ***base.html***

```
<html>
    <head> </head>
    <body>
        {% include “inr/header.html” %}
        {% block main_block %}
        {% endblock %}
        {% include “innr/footer.html” %}
    </body>
</html>
```

in our templates.html we extends *base.html* for example in *details.html*

```
{% extends “innr/base.html” %}
{% block main_block %}
<h1>{{student.student_name}}</h1>
<ul>
    <li>{{student.age}}</li>
    <li>{{student.email}}</li>
</ul>
{% endblock %}
```

Navigation

We use links and buttons to navigate between our application templates.
putting the URL regex into the href.
example in *index.html*

```
{% for s in students %}
    <tr>
        <td>{{s.student_name}}</td>
        <td>{{s.track}}</td>
        <td>{{s.age}}</td>
        <td><a href="{{s.id}}/name">View Details</a></td>
    </tr>
```

and in details.html we can add back

Adding and Processing Forms

Steps to use Django Model Form:

1. create file *forms.py* under your application
2. **`from django import forms`** and **`from .models import Student`**

each form is a class that inherits from ModelForm

in the form class we defined class **Meta:** to set the form model and the form fields

3. in *urls.py* we define the view action:

```
url(r'^track/new$', views.track_new),
url(r'^student/new$', views.student_new),
```

Django form forms.py example

```
from django import forms
from .models import Track, Student

class TrackForm(forms.ModelForm):
    class Meta:
        model = Track
        fields = ('track_name',) #note the final comma to tell its tuple.

class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ('student_name', 'age', 'track',)
```

Django Forms Views

In *views.py* create the views action to First check if the form had been submitted (posted) and of the form is valid then save it to the model else thats means the form hadn’t been submitted yet so display it.

we **import the forms and model classes**

```
from .models import Student, Track

from .forms import TrackForm, StudentForm

from django.http import HttpResponseRedirect
```

Django Form View.py example

```
def student_new(request):
    form = StudentForm()
    if request.method == "POST":
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect('/opensource')

    return render(request, 'student/new.html', {'form':form})
```

Django Form Template

The last step is to call the model form into a temlate. To do so:

- we create a form tag with method = post
- we call the {% csrf_token %} **#Cross-Site Request Forgery**
- call the form as a parameter {{ form.as_p }}
- create submit button
- close the form tag

Django Form Template Example

under*/templates/student/new.html*

```
<h1>New Student</h1>

<form method="POST">{% csrf_token %}

    {{ form.as_p }}

    <button type="submit" >Save</button>

</form>
```

Django Form Validation

Try to Save an empty Form

;)

Edit Django Form

In *urls.py* we define the views:

```
url(r'^student/(?P<student_id>[0-9]+)/edit/$', views.student_edit),
```

Edit Django Form

In *views.py* create edit actions:

```
def student_edit(request, student_id):
    student = get_object_or_404 (Student, id=student_id)
    if request.method == "POST":
        form = StudentForm(request.POST, instance=student)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect('/opensource')
    else:
        form = StudentForm(instance=student)
    return render(request, 'student/new.html', {'form':form})
```

delete Action Django Form

in urls.py

```
url(r'^student/(?P<student_id>[0-9]+)/del/$ ', views.student_delete),
```

in views.py

```
def student_delete(request, student_id):
    obj = Student.objects.get(id = student_id)
    obj.delete()
    return HttpResponseRedirect('/opensource')
```

The Full CRUD activity

Listing in views.py

```
def index(request):
    all_students = Student.objects.all()
    context = {'students':all_students}
    return render(request, 'student/index.html', context)
```

in url.py

```
            url(r'^$',views.index, name='index'),
```

Full CRUD Template

in *index.html*

```
<h1> Students </h1>
<table border="1" width="100%">
<tr><th>Name</th><th>Track</th><th>Age</th><th>Details</th><th colspan="2">Actions</th></tr>
{% for s in students %}
     <tr>
          <td>{{s.student_name}}</td>
          <td>{{s.track}}</td>
          <td>{{s.age}}</td>
          <td><a href="{{s.id}}/name">View Details</a></td>
          <td><a href="student/{{s.id}}/edit">Edit</a></td>
          <td><a href="student/{{s.id}}/del">Delete</a></td>
     </tr>
{% endfor %}
</table>
```

Static Files

To add Static files we need to:

1- create sub folder in our application directory with name ***static***
2- in setting.py STATIC_URL = os.path.join(BASE_DIR, App_name/static/')
3- in our template file **{% load static %}** on the 1st line
4- <link rel="stylesheet" type="text/css" href="{% **static** 'css/style.css' %}" />

same as image src= "{% **static** 'image/1.jpeg' %}"

Work Together ... commit your effort

Git

&

Bootstrap

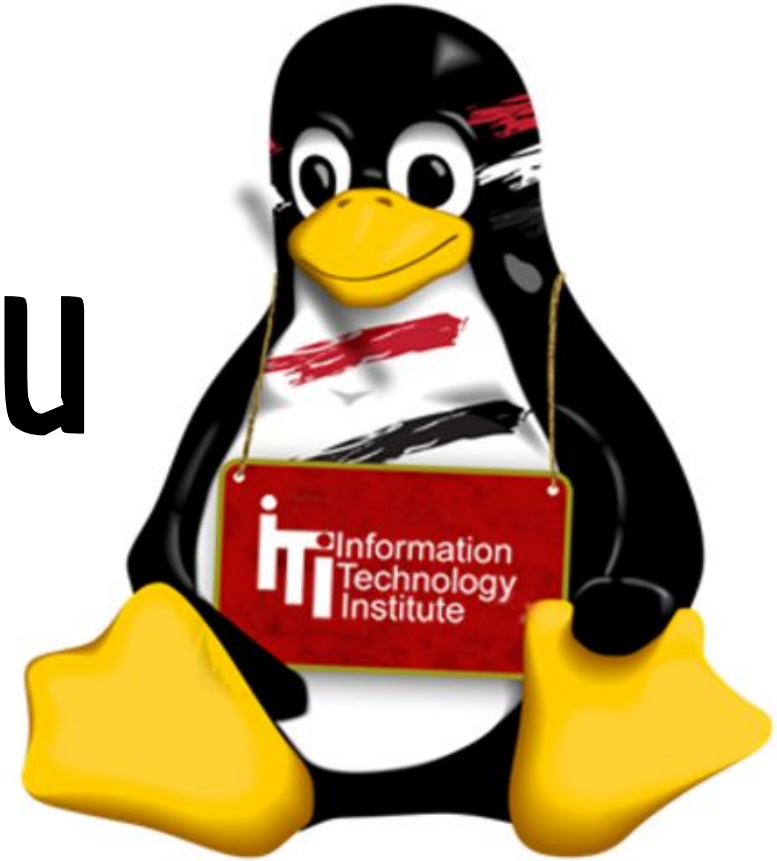
Give it a UI ... Make it a Responsive

Bootstrap -- add attributes on form elements

In forms.py

```
class student_form(forms.ModelForm):
    class Meta:
        model = student
        fields = ('st_name', 'st_age', 'st_track')
        widgets = {
        'st_name': forms.TextInput(attrs={'class': 'form-control'}),
        'st_age' : forms.TextInput(attrs={'class': 'form-control'}),
        'st_track': forms.TextInput(attrs={'class': 'form-control'})
        }
```

Thank You



Mohamed Ramadan