GKTCS INNOVATIONS

Form Validation

Forms validation

GKTCS INNOVATIONS

Form validation is the main reason that any developer has to use Forms for. The basic security principle of any computer application is "do not trust the user input". That could be because the user passes in malicious data to compromise the application or the user has a made a mistake in providing the needed data which in turn, unintentionally, breaks our application. What ever the case, every piece of user input has to be validated to keep our application secure and un-compromised.

GKTCS INNOVATIONS

Django validates a form when we run the is_valid() method and the validated data is placed in the cleaned_data attribute of the form. A coding example will clear all of our doubts caused by vague English statements.

GKTCS INNOVATIONS

We first define a registration form as follows:

```
from django.contrib.auth.models import User

class RegistrationForm(ModelForm):
    re_password = forms.CharField(max_length=128)

    class Meta:
        model = User
        fields = ['first_name', 'last_name', 'email', 'password', 're_password']
```

This form uses the Django's built in User model to build our model form. This is equivalent to following form in terms of validation except that it does not have a model associated with it or have the save() method.

```
class RegistrationForm(forms.Form):
    first_name = forms.CharField(max_length=30, required=False)
    last_name = forms.CharField(max_length=30, required=False)
    email = forms.EmailField(required=False)
    password = forms.CharField(max_length=128)
    re_password = forms.CharField(max_length=128)
```

Registration Form:

GKTCS INNOVATIONS

I have used the django bootstrap3 to render forms with Bootstrap. My HTML template (that uses django-bootstrap3) looks like this:

```
{% load bootstrap3 %}

<html>

<head>
    {% bootstrap_css %}
    {% bootstrap_javascript %}
    {% bootstrap_messages %}
</head>

<body>
    <div class="container">
        <div class="row">
            <div class="col-sm-offset-3 col-sm-6">
                <form action="" method="post">
                    {% csrf_token %}
                    {% bootstrap_form form %}
                    {% buttons %}
                    <button type="submit" class="btn btn-success btn-block">Register</button>
                    {% endbuttons %}
                </form>
            </div>
        </div>
    </div>
</body>

</html>
```

GKTCS INNOVATIONS

The view does nothing but validate the form data and show a message stating whether the form is valid or not. This is enough to demonstrate the most functionality of forms.

The only required fields in our form here are password and re_password as all the other included fields are allowed null values in the model.

GKTCS INNOVATIONS

Valid form:

Now that, I think, clears up everything that we need to change in our form now. Here they are as bullet points.

1. **Make all the fields necessary - This is possible with built-in validators.**
2. **Make both the password fields rendered as password input elements - This can be achieved with built-in widgets.**
3. **Check that the password field is at least 8 characters long - We use built-in validators again.**
4. **Finally make sure that the re-entered password is same as the password we entered first - We achieve it with a custom validator.**

Adding validators to model form

GKTCS INNOVATIONS

Task one. We first make all the fields as required. Before we do the changes in the form let us see what are the validators set for the fields. We can check them in the Django shell which we can start with:

```
python manage.py shell
```

GKTCS INNOVATIONS

Once the shell is up and running, do the following.

```
In [1]: from posts.forms import RegistrationForm

In [2]: rf = RegistrationForm()

In [3]: fn_field = rf.fields['first_name']

In [4]: fn_field
Out[4]: <django.forms.fields.CharField at 0x7f44b18dfdd8>

In [5]: fn_field.validators
Out[5]: [<django.core.validators.MaxLengthValidator at 0x7f44b228fcf8>]

In [6]: fn_field.max_length
Out[6]: 30

In [7]: fn_field.required
Out[7]: False
```

GKTCS INNOVATIONS

We have only checked the status of the first_name form field. All the other fields are also similar. There is only one validator associated with the first_name field and that is MaxLengthValidator. This has be auto-generated due the constraint of the field size in the User model. We can also observe that the required attribute of the field is set to False. To make them as required fields, we change our __init__ method of our form definition to the following:

```
def __init__(self, *args, **kwargs):
    super(RegistrationForm, self).__init__(*args, **kwargs)
    for field_name, field in self.fields.items():
        field.required = True
```

GKTCS INNOVATIONS

That is all we need to do to make the field compulsory. Before we check, we need to exit and restart the Django shell, because the changes in the code are reflected in the shell only after a restart. Now the result of our new form definition is:

```
In [1]: from posts.forms import RegistrationForm

In [2]: rf = RegistrationForm()

In [3]: fn_field = rf.fields['first_name']

In [4]: fn_field.validators
Out[4]: [<django.core.validators.MaxLengthValidator at 0x7f666a5baba8>]

In [5]: fn_field.required
Out[5]: True
```

The result when the form is submitted with empty field values is:
All are required fields

Adding additional validators to model fields

Our next requirement is to check whether the input password is at least 8 characters long or not. This is as simple as specifying the max_length parameter to the re_password field. Before making changes to our form let us examine our form in the django shell.

```
In [1]: from posts.forms import RegistrationForm

In [2]: rf = RegistrationForm()

In [3]: pwd_field = rf.fields['password']

In [4]: pwd_field.validators
Out[4]: [<django.core.validators.MaxLengthValidator at 0x7fe696cc3ac8>]
```

We only have a MaxLengthValidator for our password field. We can achieve the minimum length check by defining our password field of the RegistrationForm as follows

```
password = forms.CharField(max_length=40, min_length=8, widget=PasswordInput())
```

GKTCS INNOVATIONS

While this works what we are actually doing is re-defining the password field which is already defined in the model. A better way to achieve the same result is to add the validator in the __init__() method.

```
from django.core.validators import MinLengthValidator

def __init__(self, *args, **kwargs):
    super(RegistrationForm, self).__init__(*args, **kwargs)
    for field_name, field in self.fields.items():
        field.required = True
    # Our previous definition was till this line

    password_field = self.fields['password']
    password_field.validators.append(MinLengthValidator(limit_value=8))
```

GKTCS INNOVATIONS

Let us examine the result of this change through the Django shell.

```
In [1]: from posts.forms import RegistrationForm

In [2]: rf = RegistrationForm()

In [3]: pwd_field = rf.fields['password']

In [4]: pwd_field.validators
Out[4]:
[<django.core.validators.MaxLengthValidator at 0x7fcbfefac710>,
<django.core.validators.MinLengthValidator at 0x7fcbfefaf0f0>]
```

Validators that work on multiple fields

GKTCS INNOVATIONS

Our final requirement is to verify that both the passwords entered are same. The validation of individual field values is done by the clean_<field_name>() methods. The overall validation of the whole data is however, done by the clean() method. We are going to override this method. Define the clean() method of our RegistrationForm as follows:

```
def clean(self):
    super(RegistrationForm, self).clean()
    # This method will set the `cleaned_data` attribute

    password = self.cleaned_data.get('password')
    re_password = self.cleaned_data.get('re_password')
    if not password == re_password:
        raise ValidationError('Passwords must match')
```

GKTCS INNOVATIONS

The result of this in the HTML is:

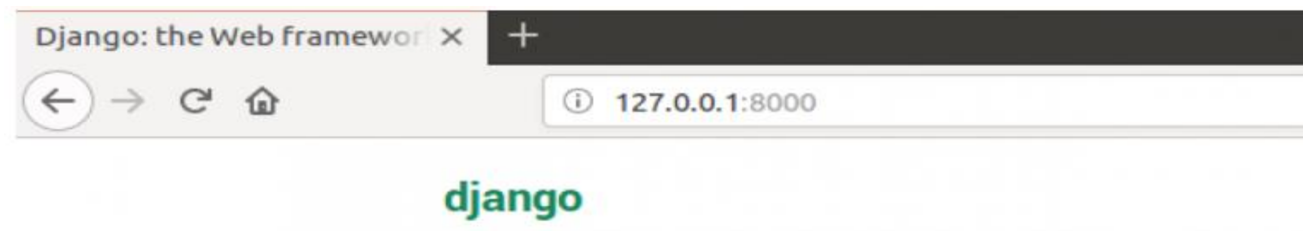
Password mis-match

URL Mapping

What is a URL?

GKTCS INNOVATIONS

A URL is a web address. You can see a URL every time you visit a website – it is visible in your browser's address bar.
(Yes! 127.0.0.1:8000 is URL and https://www.gktcs.com is also a URL)



How does Django Server interpret URLs?

Django interprets URLs in a rather different way, the URLs in Django are in the format of regular expressions, which are easily readable by humans than the traditional URLs of PHP frameworks.

Path() Function

GKTCS INNOVATIONS

In Django, the **path()** function is used to configure URLs. In its basic form, the path() function has a very simple syntax: **path(route, view)**

A practical example of the basic path() function would be: **path('mypage/', views.myview)**

In this example, a request to **http://example.com/mypage** would be routed to the **myview function** in the application's **views.py** file.

URL Function

some commonly used functions for URL handling and mapping.

01

path(route, view, kwargs=None, name=None)

It returns an element for inclusion in urlpatterns.

eg: path('index/', views.index, name='main-view')

02

re_path(route, view, kwargs=None, name=None)

It returns an element for inclusion in urlpatterns.

re_path(r'^index/$', views.index, name='index'),

GKTCS INNOVATIONS

03

include(module, namespace=None)

It is a function that takes a full Python import path to another URLconf module that should be "included" in this place.

04

register_converter(converter, type_name)

It is used for registering a converter for use in path() routes.

URL Patterns/Dispatcher

GKTCS INNOVATIONS

Root urls should be configured in settings .py

ROOT_URLCONF=‘app.urls’

Example

```
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

How do URLs work in Django?

GKTCS INNOVATIONS

Lets open up the mysite/urls.py file in your code editor of choice and see what it looks like:

mysite/urls.py

```
from django.contrib.import admin
from django.urls import path

urlpatterns = [
path(‘admin/’, admin.site.urls),

]
```

Your first Django URL !

GKTCS INNOVATIONS

- Add a line that will import blog.urls . You will also need to change the from Django.urls… Line because we are using the include function here, so you will need to add that import to the line.
- Your mysite/urls.py file should now look like this:

mysite/urls.py

```
from django.contrib.import admin
from django.urls import path, include

urlpatterns = [
path(‘admin/’, admin.site.urls),
path(‘ ’, include(‘blog.urls’)),

]
```

blog.urls

GKTCS INNOVATIONS

- Create a new empty file named urls.py in the blog directory, and open it in the code editor. Add these first two lines:

blog/urls.py

```
from django.urls import path
From .import views

urlpatterns = [
path(‘ ’, views.post_list,
name=‘post_list’),

]
```