

Validating Forms



Built-in Validators



Form Level Validation



Custom Validator



Field Level Validation

Validating Forms - Form Level Validation

► In forms.py

```
from django import forms
from django.core.exceptions import ValidationError

class ContactForm(forms.Form):

    subject = forms.CharField(max_length=100, required=True)
    message = forms.CharField(widget=forms.Textarea)
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)

    def myclean(self):
        cleaned_data = super(ContactForm, self).clean()

        email = cleaned_data.get('sender')
        dom = 'gmail.com'

        if dom not in email:
            print('here')
            raise ValidationError('incorrect domain')
```

Validating Forms - Form Level Validation

► In views.py

```
from django.shortcuts import render
from django.http import HttpResponseRedirect

from .forms import ContactForm

# Create your views here.
def page(request):

    if request.method == 'POST':

        form = ContactForm(request.POST)
        if form.is_valid():
            form.myclean()
            return HttpResponseRedirect('thanks')

    else:

        form = ContactForm()
        context = {'form': form}

        return render(request, 'home.html', context)
```

Validating Forms - Form Level Validation

► In home.html

```
<form method='POST'>
{% csrf_token %}
{{ form }}

<input type="submit" value="submit">
</form>
```

Validating Forms - Field Level Validation

- In forms.py

forms.py

```
from django import forms
from django.core.exceptions import ValidationError

class ContactForm(forms.Form):

    subject = forms.CharField(max_length=100, required=True)
    message = forms.CharField(widget=forms.Textarea)
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)

    def field_validation(self):
        email = self.cleaned_data.get('sender')
        dom = 'gmail.com'
        if dom not in email:
            raise ValidationError('field validation failed')
```

Validating Forms - Field Level Validation

► In forms.py

views.py

```
from django.shortcuts import render
from django.http import HttpResponse

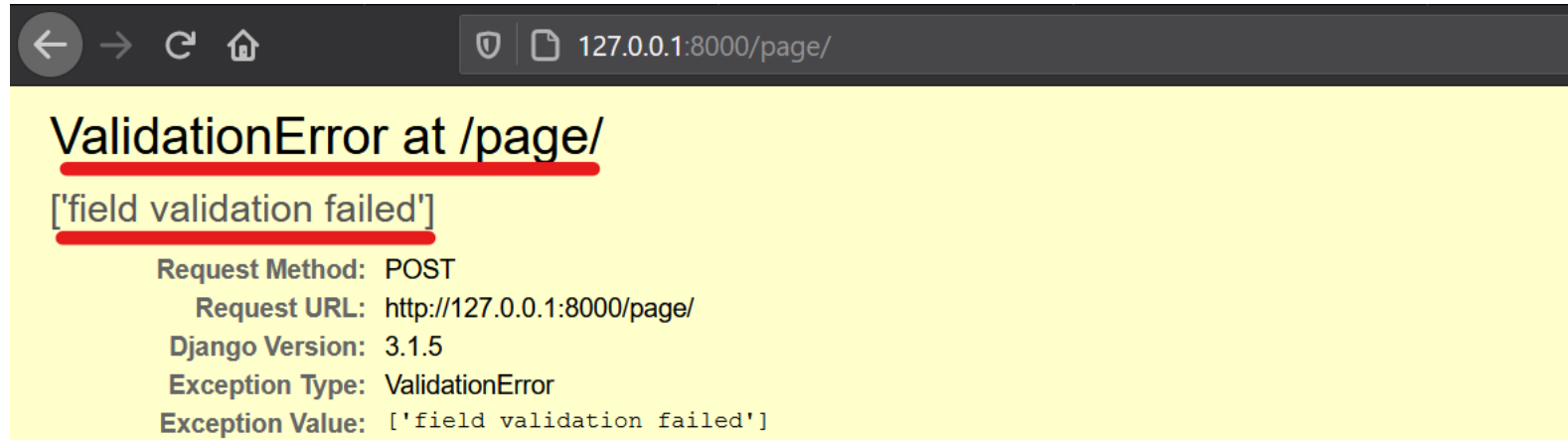
from .forms import ContactForm

def page(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            form.field_validation()
            return HttpResponse('thanks')

    else:
        form = ContactForm()
        context = {'form': form}
        return render(request, 'home.html', context)
```

Validating Forms - Field Level Validation

- ▶ If sender has any mail domain other than gmail.com



A screenshot of a web browser window showing a Django error page. The browser's address bar displays the URL `127.0.0.1:8000/page/`. The error message, displayed on a yellow background, is `ValidationError at /page/` with the value `['field validation failed']`. Below the error message, the following details are listed: Request Method: POST, Request URL: `http://127.0.0.1:8000/page/`, Django Version: 3.1.5, Exception Type: `ValidationError`, and Exception Value: `['field validation failed']`.

```
ValidationError at /page/  
['field validation failed']  
Request Method: POST  
Request URL: http://127.0.0.1:8000/page/  
Django Version: 3.1.5  
Exception Type: ValidationError  
Exception Value: ['field validation failed']
```

Forms - Built-in fields

Built-in Fields

- ▶ BooleanField
- ▶ CharField
- ▶ ChoiceField
- ▶ TypeChoiceField
- ▶ DateField
- ▶ DateTimeField
- ▶ DecimalField
- ▶ DurationField
- ▶ EmailField
- ▶ FileField
- ▶ FilePathField
- ▶ FloatField
- ▶ ImageField
- ▶ IntegerField
- ▶ JSONField
- ▶ URLField

More:

<https://docs.djangoproject.com/en/3.1/ref/forms/fields/#built-in-field-classes>

Forms - Built-in fields

forms.py

```
from django import forms

class myForm(forms.Form):
    x1 = forms.BooleanField(required=False)
    x2 = forms.CharField(min_length = 2, max_length = 10,
required=False)

    x = (('s', 'small'),
        ('m', 'medium'))
    x3 = forms.ChoiceField(choices=x)

    x = (('1', '10'),
        ('1', '20'))
    x4 = forms.TypedChoiceField(choices=x,coerce=int)
```

Forms - Built-in fields

forms.py

Contd..

```
x5 = forms.DateField(required=False)
x6 = forms.DateTimeField(required=False)
x7 = forms.DecimalField(required=False)
x8 = forms.DurationField(required=False)
x9 = forms.EmailField(required=False)
x10 = forms.FileField(required=False)
x11 =
forms.FilePathField(allow_folders=True, path=r'C:\Users\harulp663
\Desktop\practice\myproj', required=False)
x12 = forms.FloatField(required=False)
x14 = forms.IntegerField(required=False)
x15 = forms.JSONField(required=False)
x16 = forms.URLField(required=False)
```

Forms - Built-in fields

Home.html

```
<form method='POST' enctype="multipart/form-data">
{% csrf_token %}
{{ form }}

<input type="submit" value="submit">
</form>
```

Forms - Built-in fields

Views.py

```
from django.shortcuts import render
from django.http import HttpResponse

from .forms import myForm

def page(request):
    if request.method == 'POST':
        form = myForm(request.POST,request.FILES)
        if form.is_valid():
            values = form.clean()
            print(values)
            return HttpResponse('check ur command prompt')
        else:
            return HttpResponse('not valid form')

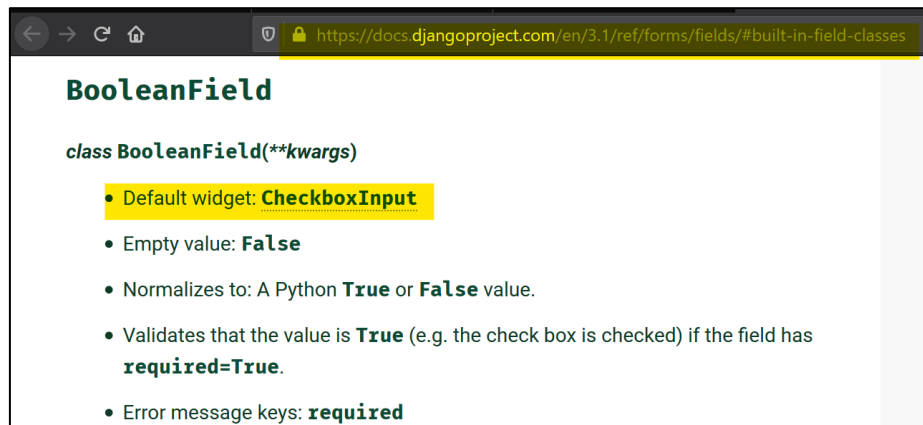
    else:
        form = myForm()
        context = {'form': form.as_ul}
        return render(request, 'home.html', context)
```

Forms - Built-in widgets

Built-in widgets

Highlighted some widgets in the diagram

- ▶ Whenever a built-in field is used, there is a default widget used



- X1: ☐
- X2:
- X3:
- X4:
- X5:
- X6:
- X7:
- X8:
- X9:
- X10: No file selected.
- X11:
- X12:
- X14:

Forms - Built-in widgets

- ▶ **Widgets handling input of text**

- ▶ TextInput
- ▶ NumberInput
- ▶ EmailInput
- ▶ URLInput
- ▶ PasswordInput
- ▶ HiddenInput
- ▶ DateTimeInput
- ▶ DateTextInput
- ▶ TimeInput
- ▶ TextArea

- ▶ **Selector and checkbox widgets**

- ▶ CheckBoxInput
- ▶ Select
- ▶ NullBooleanSelect
- ▶ SelectMultiple
- ▶ RadioSelect
- ▶ CheckboxSelectMultiple
- ▶ **File upload widgets**
 - ▶ File Input
 - ▶ ClearableFileInput

More:

<https://docs.djangoproject.com/en/3.1/ref/forms/widgets/#built-in-widgets>

Forms - Built-in widgets

forms.py

```
from django import forms

class myForm(forms.Form):
    x5 = forms.DateField( required=False,
                          widget=forms.SelectDateWidget)
```

• X1: ☐

• X2:

• X3:

• X4:

• X5:

• X6:

• X7:

• X8:

• X9:

• X10:

• X11:

• X12:

• X13:

• X14:

• null

Model form

- ▶ For database-driven app
- ▶ Using models to create a form

models.py

```
from django.db import models  
  
# Create your models here.  
class member(models.Model):  
    first_name= models.CharField(max_length=30)  
    last_name= models.CharField(max_length=30)
```


Model form

forms.py

```
from django.forms import ModelForm
from .models import member

class myModelForm(ModelForm):
    class Meta:
        model = member
        fields = ["first_name", "last_name"]
```

Model form

views.py

```
from django.shortcuts import render
from django.http import HttpResponseRedirect

from .forms import myModelForm
from .models import member

def page(request):
    if request.method == 'POST':
        form = myModelForm(request.POST)
        if form.is_valid():
            form.save()
            values = form.clean()
            print(values)
            return HttpResponseRedirect('check ur command prompt')
        else:
            return HttpResponseRedirect('not valid form')

    else:
        form = myModelForm()
        context = {'form': form}
        return render(request, 'home.html', context)
```

Model form

- ▶ You can save the form data to the DB

views.py

```
form = myModelForm(request.POST)
if form.is_valid():
    form.save()
```

- ▶ Suppose update an existing record in db?

views.py

```
x = member.objects.get(first_name="BEULAH S")
form = myModelForm(request.POST, instance=x)
if form.is_valid():
    form.save()
```

Form sets

- ▶ Suppose you need the same form to be displayed n times - use formset

views.py

```
from django.forms import formset_factory

myFormSet = formset_factory(myModelForm, extra=2)
context = {'form': myFormSet}
return render(request, 'home.html', context)
```

- ▶ Output

First name: Last name: First name: Last name:

File upload

- ▶ In forms you need `FileField`
- ▶ You need to bind the file
 - ▶ With post method add `request.files`
 - ▶ In template html file add `enctype` in form tag
- ▶ Write a function which reads that file and write to someother location

File upload

forms.py

```
from django import forms

class myForm(forms.Form):
    x = forms.FileField()
```

Index.html

```
<form method='POST' enctype="multipart/form-data">
{% csrf_token %}
{{ form }}

<input type="submit" value="submit">
</form>
```

File upload

forms.py

```
from django.shortcuts import render
from .forms import myForm

def handle_file(f):
    with open('myfile.pdf', 'wb+') as fp:
        for chunk in f.chunks():
            fp.write(chunk)

def page(request):
    if request.method == 'POST':
        form = myForm(request.POST, request.FILES)
        handle_file(request.FILES['x'])
    else:
        form = myForm()
        return render(request, 'home.html', {'form': form})
```

Output:

- ▶ A new file would have been created at your project directory

Models Layer

-

Accessing related objects

Relation

- ▶ One-to-many
- ▶ Many-to-many

How ?

- ▶ ForeignKey for one-to-many
- ▶ ManyToManyField for many-to-many

Models Layer

-

Accessing related objects

one-to-many

Models.py

```
from django.db import models

# Create your models here.
class Mother(models.Model):
    name = models.CharField(max_length=10, null=True)
    num_children = models.IntegerField(null=True)

    def __str__(self):
        return self.name

class Child(models.Model):
    mother = models.ForeignKey(Mother, on_delete=models.CASCADE,
                               null=True)

    name = models.CharField(max_length=10, null=True)
    age = models.IntegerField(null=True)
    def __str__(self):
        return self.name
```

Models Layer

-

Accessing related objects

one-to-many

Python manage.py shell

```
>>> from myapp.models import Mother
>>> from myapp.models import Child
>>>
>>>
>>> m = Mother(name="Angelina", num_children=6)
>>> m.save()
>>>
>>> c1 = Child(name="Maddox", age=10, mother=m)
>>> c1.save()
>>> c2 = Child(name="Zahara", age=12, mother=m)
>>> c2.save()
```

```
sqlite> select * from myapp_mother;
3|Angelina|6
sqlite>
sqlite>
sqlite> select * from myapp_child;
4|2|10|Krishnan
5|2|12|SriDevi
6|3|10|Maddox
7|3|12|Zahara
```

Models Layer

-

Accessing
related
objects

one-to-many

- ▶ Accessing mother through child

Python manage.py shell

```
>>> x = Child.objects.get(id=6)
>>> x
<Child: Maddox>
>>> x.mother
<Mother: Angelina>
>>> x.mother.name
'Angelina'
>>> x.name
'Maddox'
```

Models Layer

-

Accessing
related
objects

one-to-many

- ▶ Accessing child through mother

Python manage.py shell

```
>>> x = Mother.objects.get(id=3)
>>> x.child_set.all()
<QuerySet [<Child: Maddox>, <Child: Zahara>]>
```

Models Layer

-

Accessing related objects

one-to-many

- ▶ creating a relation without save() explicitly called

Python manage.py shell

```
>>> c3 = x.child_set.create(name="Shiloh", age=13)
>>> x.child_set.all()
<QuerySet [<Child: Maddox>, <Child: Shiloh>]>
```

Models Layer

-

Accessing related objects

one-to-many

- ▶ Removing a relation

Python manage.py shell

```
>>> x = Mother.objects.get(id=3)
>>> x.child_set.all()
<QuerySet [<Child: Maddox>, <Child: Zahara>]>
>>> x.child_set.remove(c2)
>>> x.child_set.all()
<QuerySet [<Child: Maddox>]>
```

Models Layer

- ▶ adding a relation

-

Accessing related objects

one-to-many

Python manage.py shell

```
>>> c2 = Child.objects.get(id=7)
>>> c2
<Child: Zahara>
>>> x
<Mother: Angelina>
>>> x.child_set.all()
<QuerySet [<Child: Maddox>, <Child: Shiloh>]>
>>> x.child_set.add(c2)
>>> x.child_set.all()
<QuerySet [<Child: Maddox>, <Child: Zahara>, <Child: Shiloh>]>
```

Models Layer

-

Accessing related objects

many-to-many

Models.py

```
# many-to-many
```

```
class Subject(models.Model):  
    name = models.CharField(max_length=10)  
    def __str__(self):  
        return self.name  
  
class Student(models.Model):  
    name = models.CharField(max_length=10)  
    subject = models.ManyToManyField(Subject)  
    def __str__(self):  
        return self.name
```


Models Layer

-

Accessing
related
objects

many-to-many

Models.py

```
>>> from myapp.models import Subject, Student
>>>
>>> s1 = Subject(name="Tamil")
>>> s1.save()
>>>
>>> s2 = Subject(name="ECA")
>>> s2.save()
>>>
>>> x = Student(name="Jessy")
>>> x.save()
>>> x.subject.set(s)
>>>
>>> x = Student(name="Peter")
>>> x.save()
>>> x.subject.set(s)
```

Authentication

Django built-in authentication

- ▶ Authentication - Checking if the user is valid
- ▶ Authorization - what the authenticated user is allowed to do

- ▶ APPS
 - ▶ `django.contrib.auth`
 - ▶ `django.contrib.contenttypes`
- ▶ MIDDLEWARE
 - ▶ `SessionMiddleware`
 - ▶ `AuthenticationMiddleware`

Authentication

Django built-in authentication

USER

- ▶ username
- ▶ password
- ▶ email
- ▶ first_name
- ▶ last_name

Authentication

Django built-in authentication

using *authenticate()*

views.py

```
from django.contrib.auth import authenticate

# Create your views here.
def page(request):

    # NOTE: passwords shouldn't be passed as plain text !
    # usually, we will get this value through a form
    # that it would be passed as a variable value

    user = authenticate(username='admin', password='admin')
    return HttpResponse(user)

    # also username admin and password admin
    # is a weak account as this is a default value
    # in most of the systems and people can easily guess it !!
```

Authentication

Django built-in authentication

- ▶ using *request.user.is_authenticated* to check if the user is logged-in
- ▶ Returns True if the user is logged in
- ▶ Returns False if not

Authentication

Django built-in authentication

Login

views.py

```
from django.contrib.auth import authenticate
from django.contrib.auth import Login

# Create your views here.
def page(request):

    user = authenticate(username='admin', password='admin')
    if user:
        Login(request, user)
```

Authentication

Django built-in authentication

logout

views.py

```
from django.contrib.auth import authenticate
from django.contrib.auth import login
from django.contrib.auth import Logout

# Create your views here.
def page(request):

    user = authenticate(username='admin', password='admin')
    if user:
        login(request, user)
        # ur logics etc, finally logout ?
        Logout(request)
```

Authentication

Password management

- ▶ In settings.py - AUTH_PASSWORD_VALIDATORS
 - ▶ UserAttributeSimilarityValidator
 - Checks whether there is similarity between username and password
 - ▶ MinimumLengthValidator
 - Minimum chars to be in a password
 - ▶ CommonPasswordValidator
 - Not a commonly used password, example: admin
 - ▶ NumericPasswordValidator
 - Checks whether the password contains only numbers

Authentication

Password management

Validate password

views.py

```
from django.shortcuts import render
from django.http import HttpResponse

from django.contrib.auth.password_validation import
validate_password

def page(request):
    mypassword = 'admin'
    return HttpResponse(validate_password(mypassword))
```

ValidationError at /page/

['This password is too short. It must contain at least 8 characters.', 'This password is too common.']

Authentication

customizing authentication

Try out -

<https://docs.djangoproject.com/en/3.1/topics/auth/customizing/#a-full-example>

Logging

- ▶ Using python inbuilt module logging

views.py

```
from django.shortcuts import render
from django.http import HttpResponse

import logging

# Create your views here.
def page(request):
    logging.basicConfig(level=logging.DEBUG)
    logging.info('page() is invoked')
    logging.error('some error occurred')

    return HttpResponse('check cmd')
```

Static file management

css, js, img

- ▶ Add `django.contrib.staticfiles` to `INSTALLED_APPS` in `settings.py`
- ▶ In `settings.py`

`settings.py`

```
STATICFILES_DIRS = [  
    os.path.join(BASE_DIR, 'static')  
]
```

`index.html`

```
{% load static %}  
...  
{% static '/css/main.css' %}
```

pagination

views.py

```
from django.shortcuts import render
from django.http import HttpResponse

from django.contrib.auth.models import User
from django.core.paginator import Paginator, EmptyPage

# Create your views here.
def foo(requests):
    users = User.objects.filter(groups__name='member')
    users = Paginator(users, 5)

    page_num = requests.GET.get('page', 1)
    try:
        users_in_page = users.page(page_num)
    except EmptyPage:
        users_in_page = users.page(1)

    context = {'users': users_in_page}

    return render(requests, 'indexx.html', context)
```

pagination

index.html

```
<table>
<thead>
  <tr>
    <th>Username</th>
    <th>Date Joined</th>
  </tr>
</thead>
<tbody>
  {% for user in users %}
    <tr>
      <td> {{ user.username }} </td>
      <td> {{ user.date_joined }} </td>
    </tr>
  {% endfor %}
</tbody>
</table>

{% if users.has_previous %}
  <a href="{% url 'trainer' %}?page={{ users.previous_page_number }}"
  class="ms-auto"> Prev </a>
{% endif %}

{% if users.has_next %}
  <a href="{% url 'trainer' %}?page={{ users.next_page_number }}"
  class="ms-auto"> Next </a>
{% endif %}
```

caching

- ▶ To cut the overhead of making queries to database

Views.py

```
from django.views.decorators.cache import cache_page

# cache page takes argument of num of seconds the cache needs to be saved

@cache_page(30)
def foo(requests):
    users = User.objects.filter(groups__name='member')
```

Site maps

- ▶ A way to tell search engines about your websites links and posts
- ▶ How?
 - Using a xml file
 - That says the urls with some meta data for the engines to understand the content
- ▶ How in Django?

In settings.py, add below to `INSTALLED_APPS`

settings.py

```
INSTALLED_APPS = [  
    'django.contrib.sites',  
    'django.contrib.sitemaps',  
]  
  
SITE_ID = 1
```


Site maps

models.py

```
from django.db import models
from django.urls import reverse

# Create your models here.
class blogs(models.Model):
    title = models.CharField(max_length=10)

    def get_absolute_url(self):
        return reverse("my_blogs", args=[str(self.id)])
```

urls.py

```
from django.contrib.sitemaps.views import sitemap
from common.sitemaps import mysitemap
from common.views import myview

sitemaps = {'blogs': mysitemap}

urlpatterns = [
    path('my_blogs/<int:id>', myview, name='my_blogs'),
    path('sitemap.xml', sitemap, {'sitemaps': sitemaps})
]
```

Site maps

views.py

```
from .models import blogs
from django.shortcuts import get_object_or_404

def myview(requests, post_id = id):
    item = get_object_or_404(blogs, id=post_id)

    return render(requests, 'sample.html', {'items': item})
```

Sample.html

```
{{ item }}
```

Admin.py

```
from django.contrib import admin
from .models import blogs

# Register your models here.
admin.site.register(blogs)
```

Site maps

Sitemaps.py

```
from django.contrib.sitemaps import Sitemap

from .models import blogs

class mysitemap(Sitemap):
    def items(self):
        return blogs.objects.all()
```



Message Framework

- ▶ For notification messages

In settings.py we have already have these

- ▶ `INSTALLED_APPS -- django.contrib.messages`
- ▶ `MIDDLEWARE --`
`django.contrib.sessions.middleware.SessionMiddleware,`
`django.contrib.messages.middleware.MessageMiddleware`
- ▶ `TEMPLATES --`
`django.contrib.messages.context_processors.messages`

You need view function, urls , and a html template

Message Framework

views.py

```
from django.contrib import messages

def my_message(requests):
    messages.add_message(requests, messages.INFO, 'hiiii')
    messages.success(requests, 'success message')
    messages.warning(requests, 'warning message')
    messages.error(requests, 'error occurred')
    context = {}
    return render(requests, 'my_message.html', context)
```

views.py

```
from trainer.views import my_message

urlpatterns = [
    path('my_message_site', my_message)
]
```

Message Framework

My_message.html

```
{% if messages %}
<ul>
  {% for message in messages %}
    <li>{{message.tags}}: {{ message }} </li>
  {% endfor %}
</ul>
{% endif %}
```

