Django

Getting Python in the web

Outline

Forms

Forms

- Provide a convenient way to handle form **validation, rendering, and data processing** in Django applications. They abstract away the **complexities** of HTML form handling and provide **a high-level API for working with forms**

- **Form class** is a Python class that **defines** the structure and behavior of a form. It is typically created by **subclassing django.forms.Form or django.forms.ModelForm** to define the fields, validation rules, and any additional behavior for the form.

- **Widget** is a representation of an HTML form input element. It defines how a form field is rendered and displayed in the user interface. **Widgets control the appearance, behavior, and input options** for a form field.

Field

Form Class

`Form.is_bound()`	Check if the Form has populated by data or not
`Form.is_valid()`	Check if the Form valid or not
`Form.errors`	The errors list for all fields
`Form.fields`	The field list

Fields Options

Field

- `required`
- `label`
- `initial`
- `error_messages`
- `disabled`
- `widget`

CharField

- `min_length`
- `max_length`

IntegerField

- `min_value`
- `max_value`

ChoiceField

- `choices`

Field – Widget Map

Field	Widget
CharField	TextInput
EmailField	EmailInput
IntegerField	NumberInput
BooleanField	CheckboxInput
ChoiceField	Select
DateField	DateInput

django.forms.Form

Form-Example

```
from django import forms

class MyForm(forms.Form):
    # Text field
    name = forms.CharField(max_length=100, label='Name', required=True)

    # Email field with custom validation
    email = forms.EmailField(label='Email', required=True, help_text='Please enter a valid email
address.')

    # Integer field with minimum and maximum value validation
    age = forms.IntegerField(label='Age', min_value=18, max_value=99)

    # Boolean field
    is_active = forms.BooleanField(label='Active', required=False)

    # Choice field with options
    GENDER_CHOICES = (        ('M', 'Male'),        ('F', 'Female'),        ('O', 'Other'),    )
```

Form-Example

```
gender = forms.ChoiceField(label='Gender', choices=GENDER_CHOICES)

# Multiple choice field with checkbox rendering
LANGUAGE_CHOICES = (
    ('en', 'English'),
    ('fr', 'French'),
    ('es', 'Spanish'),
)
languages = forms.MultipleChoiceField(label='Languages', choices=LANGUAGE_CHOICES,
                                      widget=forms.CheckboxSelectMultiple)

# Date field
birth_date = forms.DateField(label='Birth Date')

# File upload field
resume = forms.FileField(label='Resume', required=False)
```

Relationship Fields

- **Create** a form that is not directly tied to a model.
- **Define** form fields explicitly within the form class, specifying their types, validation rules, and any custom behavior.
- **form class** allows you to handle data **input and validation** without directly interacting with a database

Form-Example

```
# Password field with minimum length validation
   password = forms.CharField(label='Password', widget=forms.PasswordInput, min_length=8)

   # Hidden field
   secret_key = forms.CharField(widget=forms.HiddenInput)

   # Custom validation for the entire form
   def clean(self):
       cleaned_data = super().clean()
       # Perform additional validation across multiple fields
       name = cleaned_data.get('name')
       email = cleaned_data.get('email')

       if name and email and name.lower() == email.lower():
           raise forms.ValidationError("Name and email cannot be the same.")

       return cleaned_data
```

Form handle in views

```
from django.shortcuts import render, redirect
from .forms import MyForm
def my_view(request):
    if request.method == 'POST':
        form = MyForm(request.POST)
        if form.is_valid():
            # Process the form data
            # Access form field values using form.cleaned_data dictionary

            # Perform additional actions (e.g., save to database, send email)

            # Redirect to a success page or return an appropriate response
            return redirect('success')
    else:
        form = MyForm()

    # Render the form in the template
    return render(request, 'my_template.html', {'form': form})
```

Form handle in Template

```
<form method="post" action="{% url 'my_view' %}">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Submit</button>
</form>
```

ModelForm-Example

```
from django import forms
from .models import MyModel

class MyModelForm(forms.ModelForm):
    class Meta:
        model = MyModel
        fields = ['name', 'email', 'age', 'is_active']
        labels = {
            'name': 'Name',
            'email': 'Email',
            'age': 'Age',
            'is_active': 'Active',
        }
        widgets = {
            'name': forms.TextInput(attrs={'placeholder': 'Enter your name'}),
            'email': forms.EmailInput(attrs={'placeholder': 'Enter your email'}),
            'age': forms.NumberInput(attrs={'min': 18, 'max': 99}),
        }
```

ModelForm-Example

```
help_texts = {
            'email': 'Please enter a valid email address.',
        }
        error_messages = {
            'name': {
                'required': 'Name is required.',
                'max_length': 'Name should not exceed 100 characters.',
            },
            'age': {
                'required': 'Age is required.',
                'invalid': 'Age must be a valid number.',
                'min_value': 'Age should be at least 18.',
                'max_value': 'Age should not exceed 99.',
            },
        }
```

ModelForm-Example

```
def clean(self):
        cleaned_data = super().clean()
        # Perform additional validation across multiple fields
        name = cleaned_data.get('name')
        email = cleaned_data.get('email')

        if name and email and name.lower() == email.lower():
            raise forms.ValidationError("Name and email cannot be the same.")

        return cleaned_data
```

ModelForm handle in views

```
from django.shortcuts import render, redirect
from .forms import MyModelForm
def my_view(request):
    if request.method == 'POST':
        form = MyModelForm(request.POST)
        if form.is_valid():
            instance = form.save()
            # Process the form data or perform additional actions
            return redirect('success')
    else:
        form = MyModelForm()

    return render(request, 'my_template.html', {'form': form})
```

View Decorator

Decorator

- Modify the behavior of view functions.
- @decorator_name
- Commonly used view decorators in Django
- **@login_required**: This decorator ensures that the user must be authenticated to access the view. If the user is not authenticated, they will be redirected to the login page
- **@permission_required:** This decorator checks if the user has the specified permission to access the view
    - @permission_required('app_name.permission_name')

Decorator

- **@cache_page:** This decorator caches the rendered output of the view for a specified duration, improving performance by serving cached content instead of executing the view function.

    - @cache_page(60 * 15) # Cache for 15 minutes

- **@csrf_exempt:** This decorator disables CSRF protection for the view, allowing POST requests without requiring a CSRF token. Be cautious when using this decorator, as it removes an important security measure

Example

```
from django.views.decorators.http import require_http_methods

@require_http_methods(['GET', 'POST'])
def my_view(request):
    if request.method == 'GET':
        # Handle GET request
        return HttpResponse('This is a GET request.')
    elif request.method == 'POST':
        # Handle POST request
        return HttpResponse('This is a POST request.')
    else:
        # Return a 405 Method Not Allowed response for other HTTP methods
        return HttpResponseNotAllowed(['GET', 'POST'])
```

Example

```
from django.views.decorators.http import require_GET

@require_GET
def my_view(request):
    # Handle GET request
    return HttpResponse('This is a GET request.')
```

Class-based Views

Organizing and structuring view logic

Example

```
#views.py
from django.views import View
from django.http import HttpResponse

class MyView(View):
    def get(self, request):
        # Handle GET request
        return HttpResponse('This is a GET request.')

    def post(self, request):
        # Handle POST request
        return HttpResponse('This is a POST request.')

# urls.py
 urlpatterns = [ url(r'^about/$', MyView.as_view()) ]
```

Generic View

create reusable view classes for common tasks

Views - template Map

if app_name is library and model = Book

View	Template Name	Context
`ListView`	library /book_list.html	object_list
`DetailView`	library /book_detail.html	object
`CreateView`	library /book_form.html	form
`UpdateView`	library /book_update_form.html	form
`DeleteView`	library / book_confirm_delete.html	

Example

```
from django.views.generic import ListView, DetailView, CreateView, UpdateView, DeleteView
from django.urls import reverse_lazy
from myapp.models import MyModel

class MyModelListView(ListView):
    model = MyModel
    template_name = 'myapp/mymodel_list.html'
    context_object_name = 'mymodels'

class MyModelDetailView(DetailView):
    model = MyModel
    template_name = 'myapp/mymodel_detail.html'
    context_object_name = 'mymodel'
```

Example

```
class MyModelUpdateView(UpdateView):
    model = MyModel
    template_name = 'myapp/mymodel_update.html'
    fields = ['field1', 'field2', 'field3']
    success_url = reverse_lazy('mymodel-list')

class MyModelDeleteView(DeleteView):
    model = MyModel
    template_name = 'myapp/mymodel_confirm_delete.html'
    success_url = reverse_lazy('mymodel-list')
```

Shortcuts

simplify common tasks and reduce the amount of code you need to write

Redirect

```
from django.shortcuts import redirect

def my_view(request):
    return redirect('/myapp/some-url/')
```

Other

```
from django.shortcuts import get_object_or_404, get_list_or_404, get_object_or_None
from myapp.models import MyModel

def my_view(request, object_id):
    obj = get_object_or_404(MyModel, id=object_id)
    # Handle the retrieved object or raise a 404 error

Or

def my_view(request):
    objects = get_list_or_404(MyModel, some_condition=True)
    # Handle the retrieved list of objects or raise a 404 error

def my_view(request, object_id):
                   obj = get_object_or_None(MyModel, id=object_id)
                   # Handle the retrieved object or None if not found
```

Sessions

The session framework lets you store and retrieve arbitrary data on a per-site-visitor basis. It stores data on the server side and abstracts the sending and receiving of cookies.

settings.py

```
INSTALLED_APPS = [

         # Other Apps,

         “django.contrib.sessions”

]


MIDDLEWARE = [

          # Other Middleware Classes,

         “django.contrib.sessions.middleware.SessionMiddleware”

]
```

Settings

`SESSION_COOKIE_AGE`	Set the Session Cookie Age
`SESSION_EXPIRE_AT_BROWSER_CLOSE`	If True, Session Cookie Age will be deleted after browser closed
`SESSION_COOKIE_NAME`	Set the Session Cookie Name

request.session

views.py

```
def login(request,user_id):
     #Setting Session item
     request.session['member_id'] = user_id
     return HttpResponse("You are logged in")


def view_profile(req):
    #Getting Session item
    if 'member_id' in req.session and req.session['member_id']:
        return HttpResponse("View Profile")
    else:
        return HttpResponse("Please login to view ")
```

methods

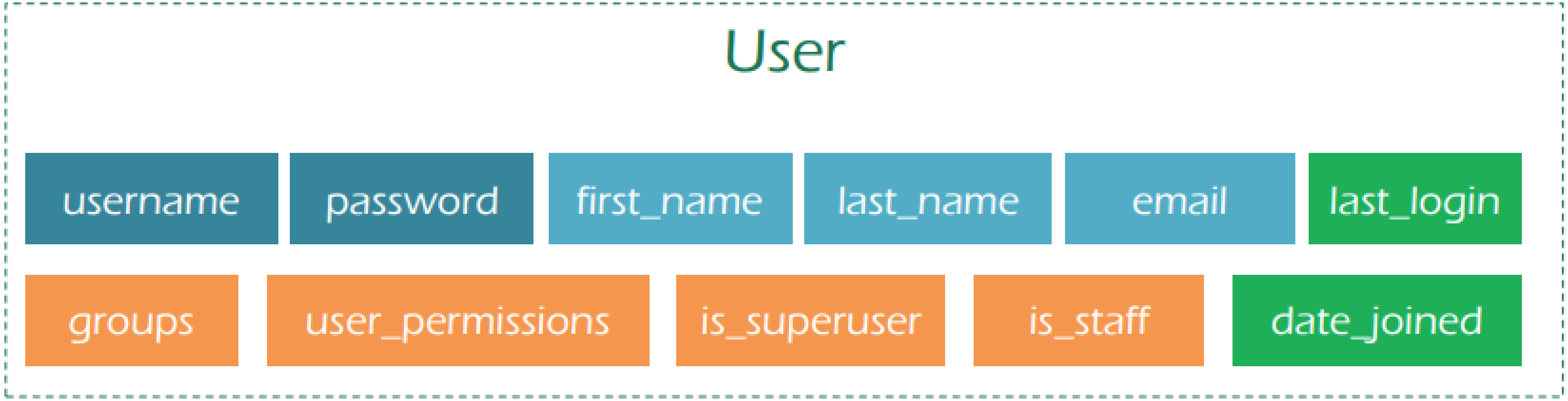
```
request.session.get(key, default=None)
```

```
request.session.pop(key, default=__not_given)
```

```
request.session.clear()
```

```
request.session.set_expiry(value)
```

Authentication



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

user = User.objects.create_user(‘Ahmed',’am@gmail.com’,‘120829')
```

User Authentication Example

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login, logout

def login_view(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('home')
    else:
        form = AuthenticationForm()
    return render(request, 'login.html', {'form': form})
```

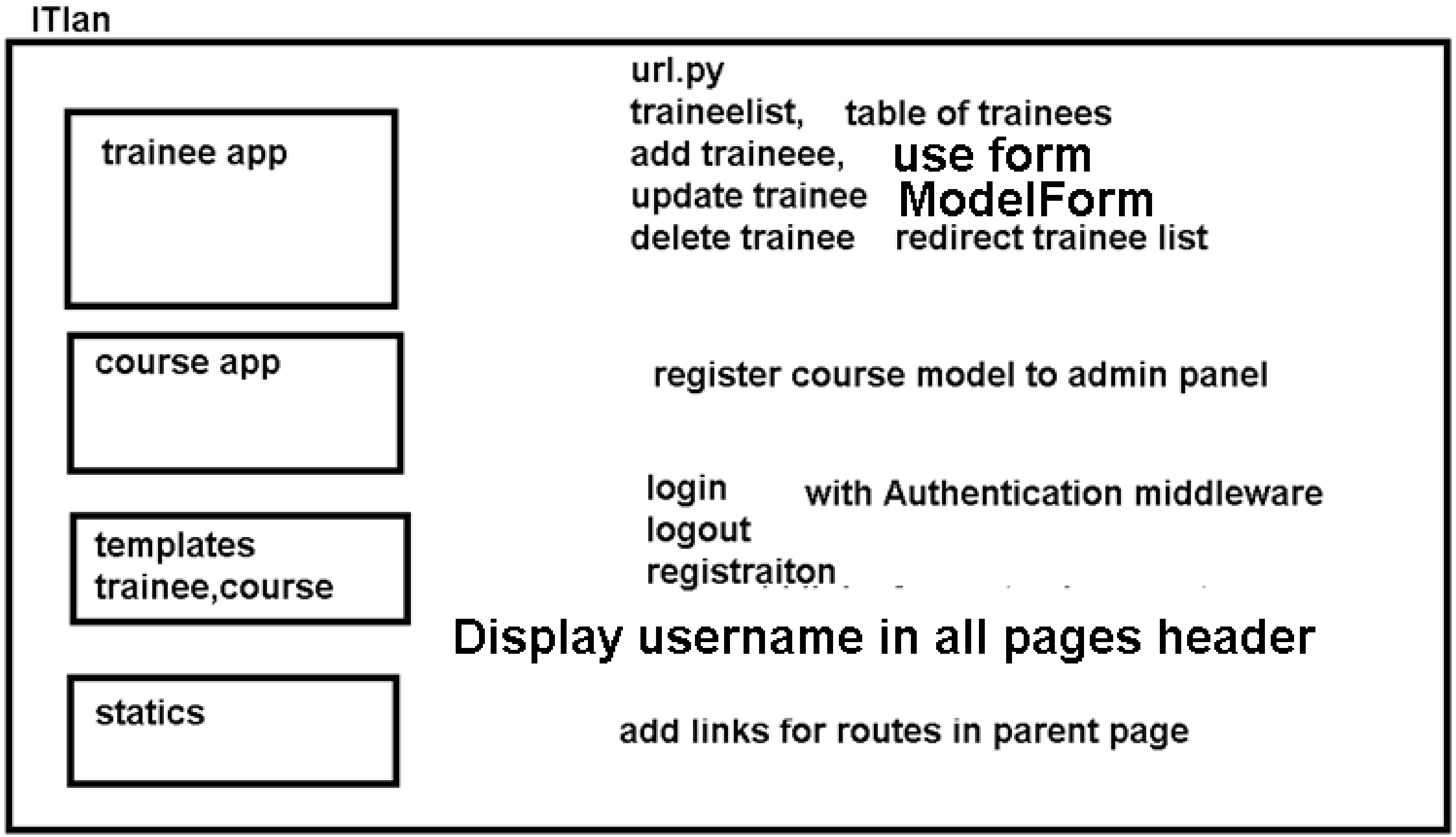
User Authentication Example

```
def any_view(request):
 if request.user.is_authenticated():
 # Do actions for Logged in Users
 else:
# Do actions for Guests


def logout_view(request):
    logout(request)
    return redirect('login')
```

Lab

template   inhertance



url.py
traineelist,   table of trainees
add traineee,   use form
update trainee   ModelForm
delete trainee   redirect trainee list

register course model to admin panel

login   with Authentication middleware
logout
registraiton

Display username in all pages header

add links for routes in parent page