Django Form and models

INT 253

Django Models

- A Django model is the built-in feature that Django uses to create tables, their fields, and various constraints.
- In short, Django Models is the SQL Database one uses with Django.

Django Models

- A model is the single, definitive source of information about your data.
- It contains the essential fields and behaviors of the data you're storing.
- Generally, each model maps to a single database table.

The basics:

- Each model is a Python class that subclasses django.db.models.
- Model.Each attribute of the model represents a database field.
- With all of this, Django gives you an automatically-generated database-access API

- Django provides a Form class which is used to create HTML forms.
- It describes a form and how it works and appears.
- It is similar to the **ModelForm** class that creates a form by using the Model, but it does not require the Model.
- Each field of the form class map to the HTML form **<input>** element and each one is a class itself, it manages form data and performs validation while submitting the form.

Example

```
from django import forms
class StudentForm(forms.Form):
 firstname = forms.CharField(label="Enter first name",max_length
=50)
 lastname  = forms.CharField(label="Enter last name", max_lengt
h = 100)
```

Explanation

- StudentForm is created that contains two fields of CharField type. Charfield is a class and used to create an HTML text input component in the form.
- The label is used to set HTML label of the component and max_length sets length of an input value.

Rendered, it produces the following HTML to the browser.

```
<label for="id_firstname">Enter first name:</label>
 <input type="text" name="firstname" required maxlength="50" id="id_firstname" />
<label for="id_lastname">Enter last name:</label> <input type="text" name="lastname" required maxlength="100" id="id_lastname" />
```

Building a Form in Django Put this code into the **forms.py** file.

- Create a form to get Student information, use the following code.

```
from django import forms
class StudentForm(forms.Form):
firstname = forms.CharField(label="Enter first name",max_length=
50)   lastname  = forms.CharField(label="Enter last name", max_l
ength = 100)
```

we need to instantiate the form in **views.py** file

```
from django.shortcuts import render
from myapp.form import StudentForm
def index(request):
student = StudentForm()
return render(request,"index.html",{'form':student})
```

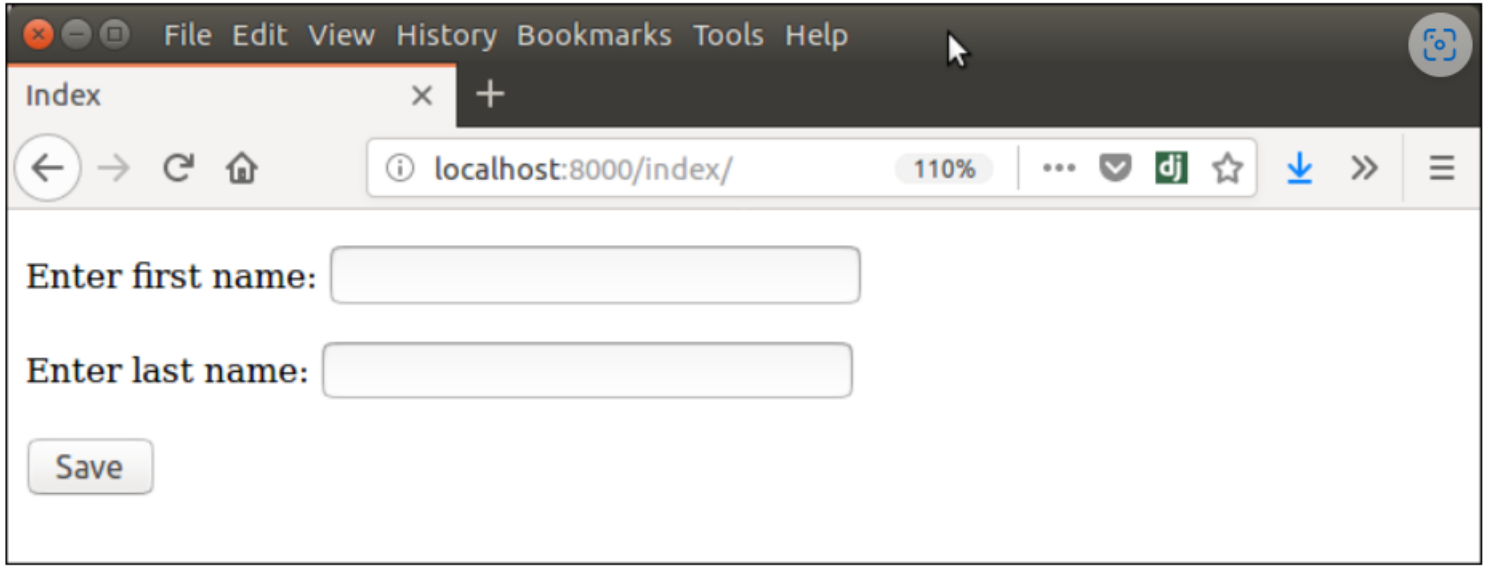
Passing the context of form into index template that looks like this:**index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Index</title>
</head>
<body>
<form method="POST" class="post-form">
 {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
</form>
</body>
</html>
```

Provide the URL in urls.py

```
from django.contrib import admin
from django.urls import path
from myapp import views
urlpatterns = [
path('admin/', admin.site.urls),
path('index/', views.index),
]
```

- Run Server and access the form at browser by **localhost:8000/index,** and it will produce the following output.


File Edit View History Bookmarks Tools Help
Index
localhost:8000/index/
110%
Enter first name:
Enter last name:
Save

OUTPUT OPTION

There are other output options though for the <label>/<input> pairs:

{{ form.as_table }} will render them as table cells wrapped in <tr> tags

{{ form.as_p }} will render them wrapped in <p> tags

{{ form.as_ul }} will render them wrapped in <li> tags

Django Form Validation

- Django provides built-in methods to validate form data automatically. Django forms submit only if it contains CSRF tokens.
- It uses a clean and easy approach to validate data.

-
- The **is_valid()** method is used to perform validation for each field of the form, it is defined in Django Form class.
- It returns True if data is valid and place all data into a cleaned_data attribute.

Metaclass

- A metaclass in Python is a class of a class that defines how a class behaves.
- A class is itself an instance of a metaclass.
- A class in Python defines how the instance of the class will behave.
- In order to understand metaclasses well, one needs to have prior experience working with Python classes.

Example

- **class TestClass():**
- **pass**

- **my_test_class = TestClass()**
- **print(my_test_class)**
- This code defines a class called **TestClass** using the **class** keyword in Python.
- The **pass** keyword is used to indicate that the class has no methods or attributes defined.
- Then, an instance of the **TestClass** class is created and assigned to the variable **my_test_class** using the parentheses **()** after the class name.
- Finally, the **print()** function is used to output the value of **my_test_class**, which will be a string representation of the object's memory location in memory.

Django Validation Example

- **models.py**

```
1.from django.db import models
2.class Employee(models.Model):
3.    eid = models.CharField(max_length=20)
4.    ename = models.CharField(max_length=100)
5.    econtact = models.CharField(max_length=15)
6.    class Meta:
7.        db_table = "employee"
```

Now, create a form which contains the below code.

- **forms.py**

```
1.from django import forms
2.from myapp.models import Employee
3.
4.class EmployeeForm(forms.ModelForm):
5.    class Meta:
6.        model = Employee
7.        fields = "__all__"
```

Instantiate the form, check whether request is post or not. It validate the data by using is_valid() method.

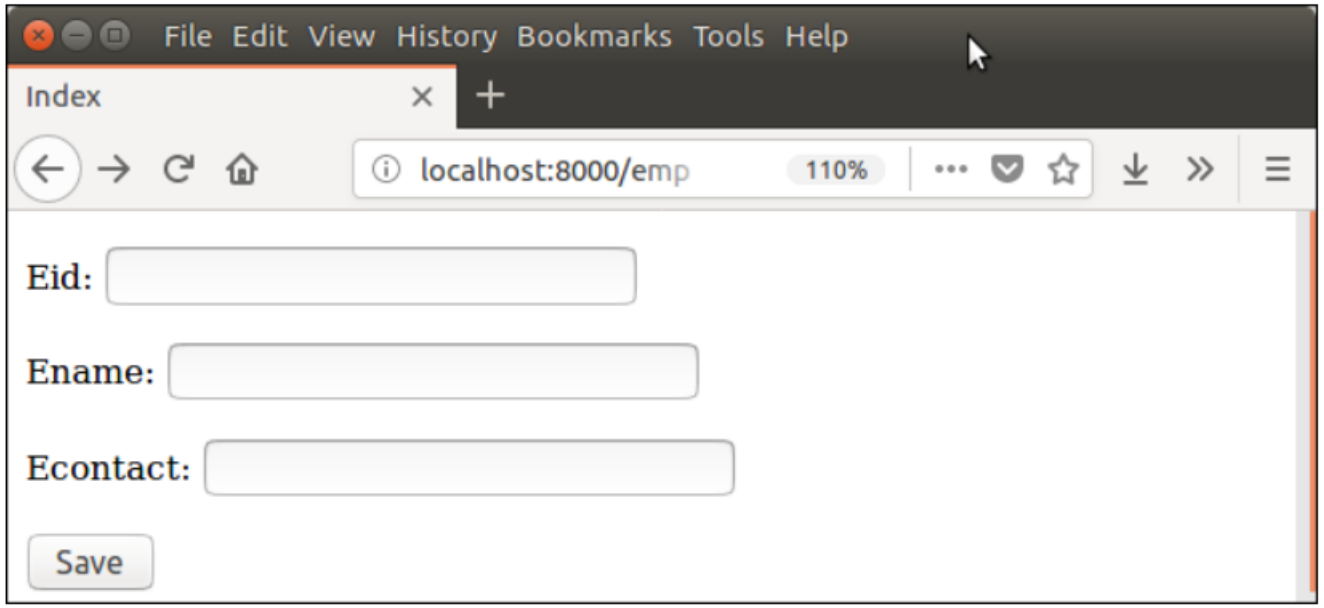
- **views.py**

```
def emp(request):
    if request.method == "POST":
        form = EmployeeForm(request.POST)
        if form.is_valid():
            try:
                return redirect('/')
            except:
                pass
    else:
        form = EmployeeForm()
    return render(request,'index.html',{'form':form})
```

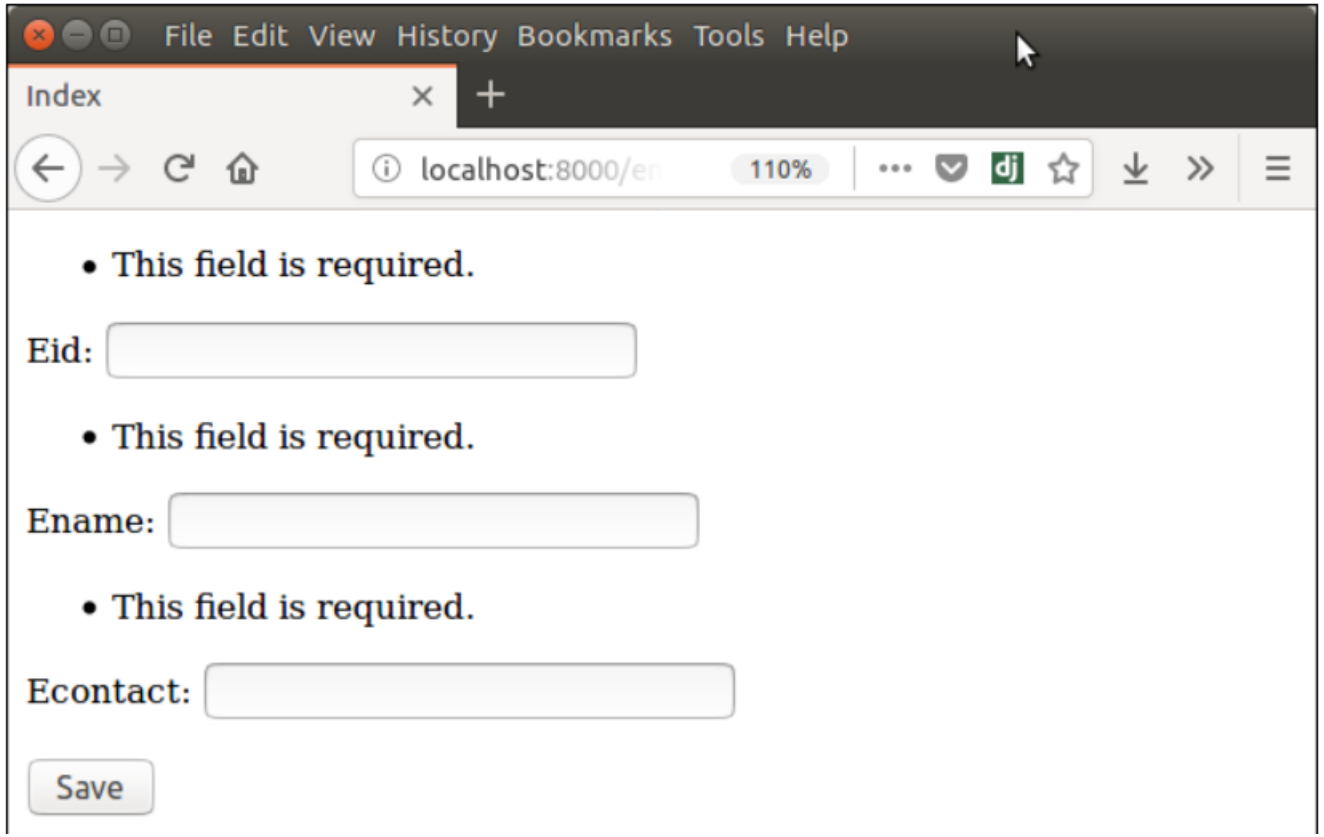
Index template that shows form and errors.
// index.html

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>Index</title>
6. </head>
7. <body>
8. <form method="POST" class="post-form" enctype="multipart/form-data">
9.         {% csrf_token %}
10.        {{ form.as_p }}
11.        <button type="submit" class="save btn btn-default">Save</button>
12.</form>
13.</body>
14.</html>
```

Start server and access the form.



It validates each field and throws errors if any validation fails.



Example
model.py

```
from django.db import models
class Service(models.Model):
    service_icon=models.CharField(max_length=50)
    service_title=models.CharField(max_length=50)
    service_des=models.TextField()
```

Make Migrations

- Python.manage.py startapp application_name
- Python.manage.py makemigration

```
Migrations for 'service':
  service\migrations\0001_initial.py
    - Create model Service
```

Initial.py created in migration folder

```
migrations.CreateModel(
    name='Service',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True,
        ('service_icon', models.CharField(max_length=50)),
        ('service_title', models.CharField(max_length=50)),
        ('service_des', models.TextField()),
    ],
),
```

Django template language

- Django’s template language is designed to strike a balance
- between power and ease. It’s designed to feel comfortable to those used to working with HTML
- It includes
- Templates
- Variable
- Filters
- Tags
- Comments
- Template Inheritance

Introduction

- DTL is a powerful and flexible language that allows you to create dynamic web pages using HTML templates.

```
{% extends "base.html" %}

{% block content %}
    <h1>{{ title }}</h1>
    <p>{{ body|linebreaks }}</p>
{% endblock %}
```

•

•{% if condition %}...{% endif %}: Allows you to conditionally render content based on a Boolean value.
•{% for item in list %}...{% endfor %}: Allows you to loop over a list or query set.
•{% block name %}...{% endblock %}: Defines a named block that can be overridden in a child template.
•{% include "template.html" %}: Includes another template within the current template.

Templates

- A template is a text file. It can generate any text-based format (HTML, XML, CSV, etc.).
- A template contains **variables**, which get replaced with values when the template is evaluated, and **tags**, which control the logic of the template.

```
{% extends "base_generic.html" %}

{% block title %}{{ section.title }}{% endblock %}

{% block content %}
<h1>{{ section.title }}</h1>

{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

Variables

- Variable look like
- {{ variable}}
- When the template engine encounters a variable ,it evaluates that variable and replaces it with result.
- Variable name consists of any combination of alphanumeric character and underscore(“ _“) but may not start with an underscore and may not be a number.
- The dot(“.”) also appears in variable sections
- Use a dot(.) to access attribute of a variable.

Variables

- {{section.title}} will be replced with the **title** attribute of the **section** object.
- {{foo.bar}} will be interpreted as a literal string and not using the value of “bar”,if one exists I the template context.
- Variable attribute that begin with underscore may not be accessed as they are generally considered private.

Technical View

- When the template system encounters a dot, it tries the following lookups
- Dictionary lookup
- Attribute or method lookup
- Numeric index lookup
- If the resulting value is callable, it is called with no argument.The result of the call becomes the template value.

Filter

- Modify variable for display by using **filters**
- **Filters look like {{ name|lower}}**
- **This display the value {{name}} variable after being filtere through the lower filter,which converts text to lowercase.Use a (|) to apply filter**
- **Filter can be chained**
- **The output of ine filter is applied to the next .{{text|escape|linebreaks}} is common idiom for escaping text contents,then converting line breaks to <p> tags**

•

•{{ variable|default:"Default Value" }}: Sets a default value for a variable if it is not defined.
•{{ variable|length }}: Returns the length of a list or string.
•{{ variable|title }}: Capitalizes the first letter of each word in a string.
•{{ variable|date:"D d M Y" }}: Formats a date according to a specified format.

Filter –take argument

- A filter argument look like
- {{bio|truncatewords:30}}
- 30 words of the bio variable
- Filter argument that contain spaces must be quoted
- Eg {{ list|join:”,”}}

Default variable

- If a variable is false or empty ,use given default
- {{value|default:”nothing”}}

length

- Return the length of the value.This works both for string and lists
- Eg
- {{value|length}
- If value [‘a’,’b’,’c’,’d’] ,output will be 4

Tags

- Tags look like:{% tag%}.
- More complex than variable
- Some create text in output, some control flow by performing loops or logic and some loads external information into templates to be used later variable
- {%tag%}..tag contents..{% endtag%})

For-Loop over each item in an array

- Eg to display the list of athletes provided in **athlete_list:**

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>
```

If,elif and else

- Evaluate a variable ,and if that variable is “true” then the content of block will displayed
- Athlete_list is not empty,the number of athletes will displayed by the {{athlete_list|length}} variable.
- If athlete_in_locker_room_list is not empty ,the message”Athlethes should be out of the locker room soon!”
- If both lists are empty ,”No athleyes will be displayed.

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|length }}
{% elif athlete_in_locker_room_list %}
    Athletes should be out of the locker room soon!
{% else %}
    No athletes.
{% endif %}
```

If tag

- Various operations can be used in if tag

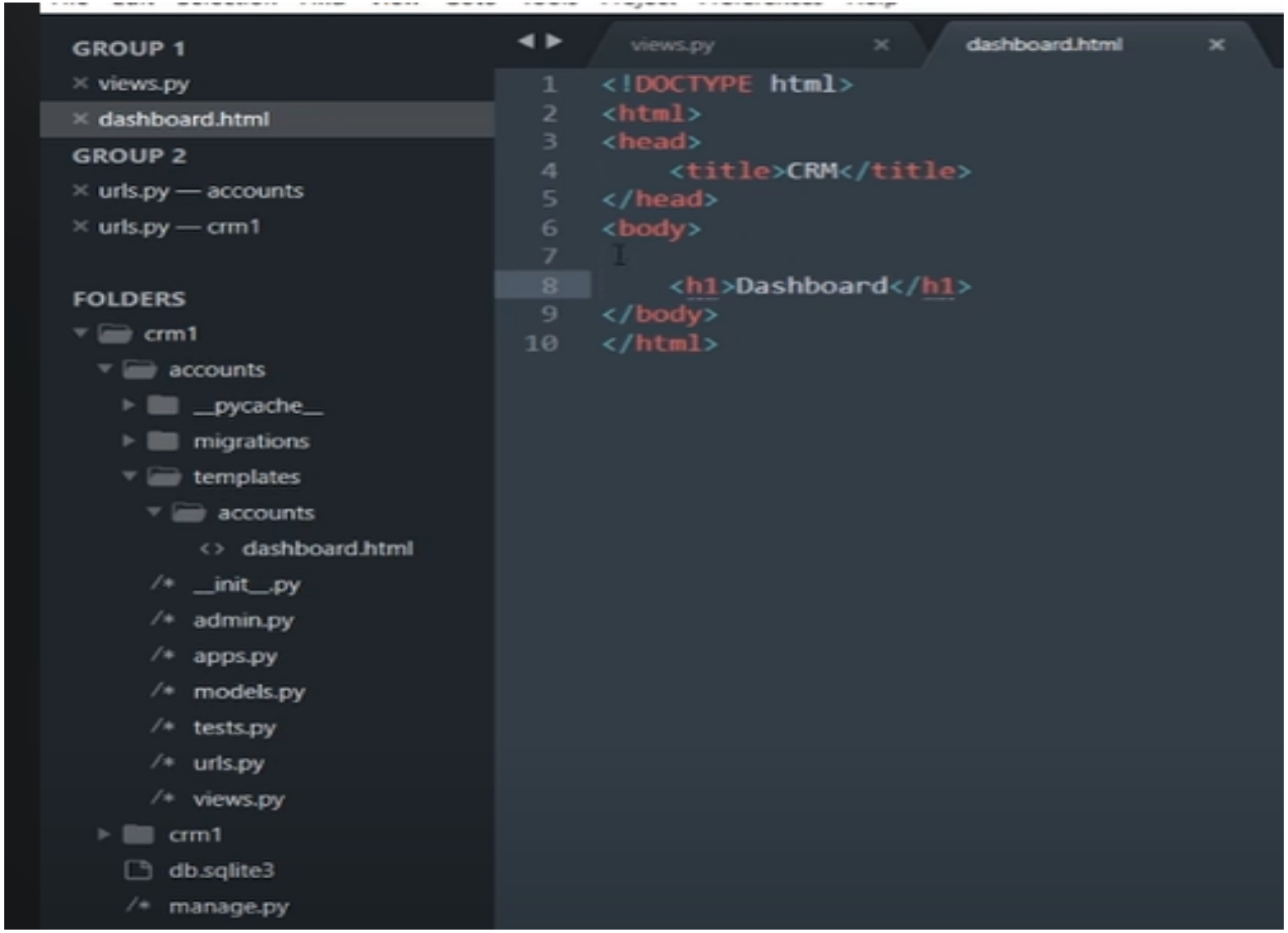
```
{% if athlete_list|length > 1 %}
   Team: {% for athlete in athlete_list %} ... {% endfor %}
{% else %}
   Athlete: {{ athlete_list.0.name }}
{% endif %}
```

Comment

- To comment –out part of a line in a template ,use the comment
- Syntax:
- {# #}.
- Eg template will render as ‘hello’
- *{#greeting#}*hello

Template Inheritance

Templates –account folder

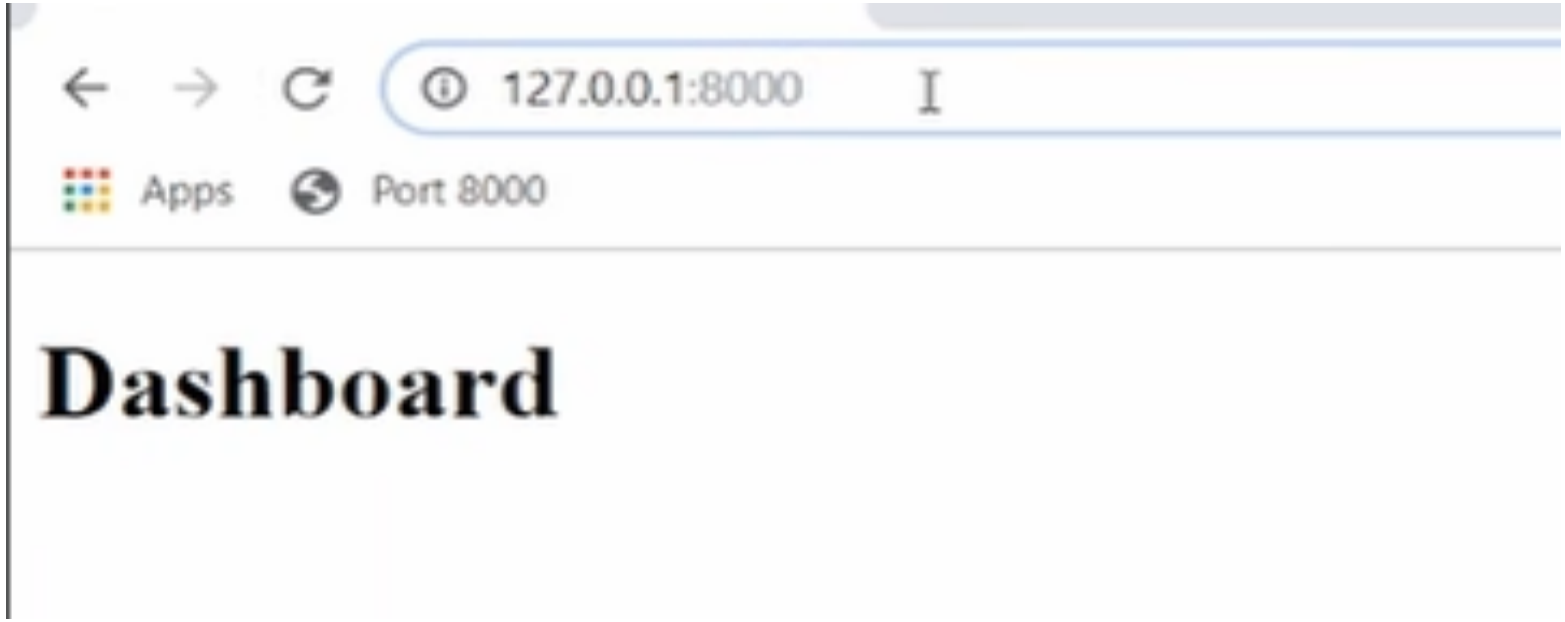


Rendering the page in view .py

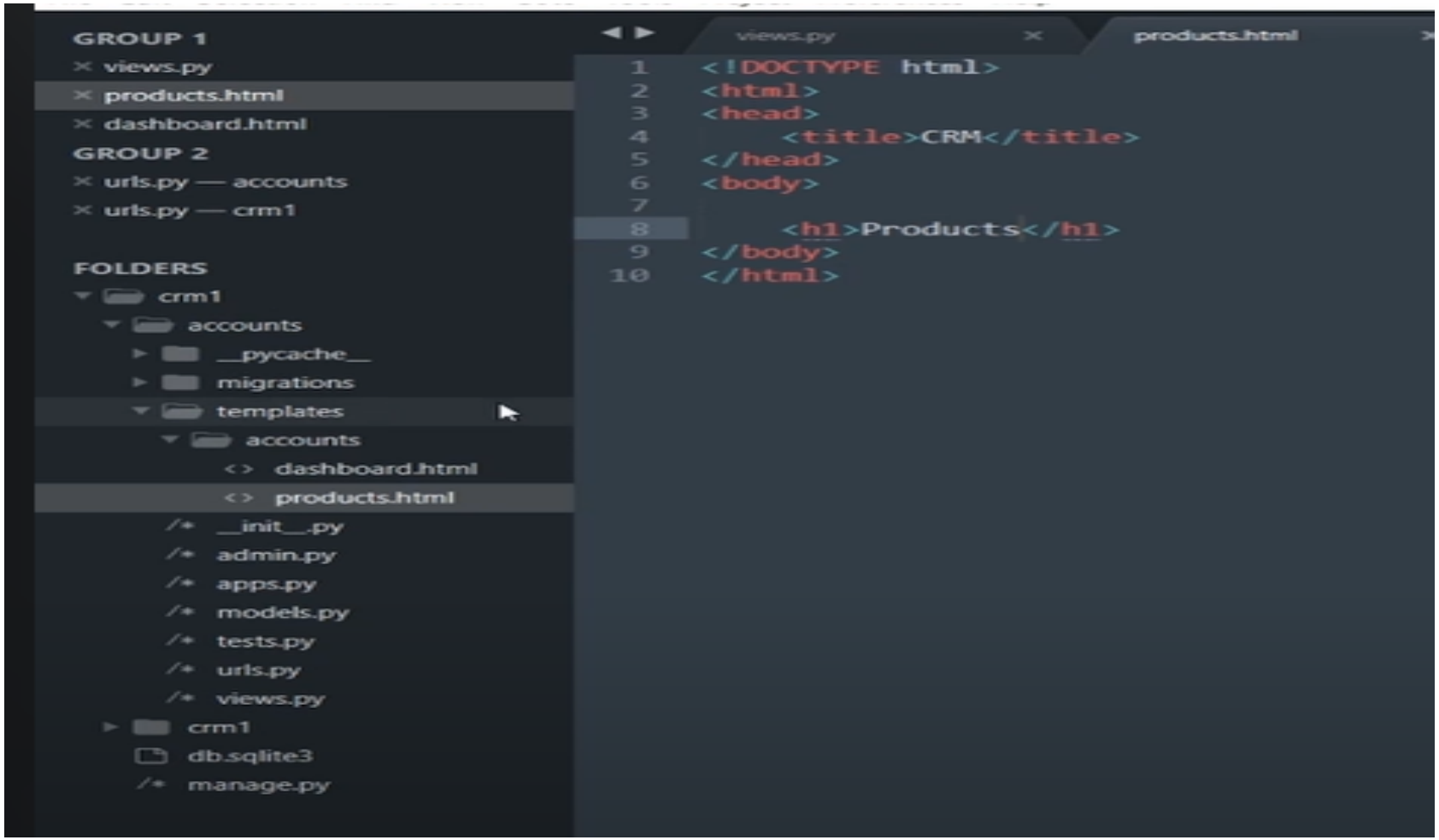
views.py dashboard.html

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

# Create your views here.

def home(request):
    return render(request, 'accounts/dashboard.html')
```

Output



Create a product.html page



Create a customer html page

views.py | products.html | customer.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>CRM</title>
</head>
<body>

    <h1>Customer</h1>
</body>
</html>
```

Views.py

views.py | products.html | customer.html | dashboa

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

# Create your views here.

def home(request):
    return render(request, 'accounts/dashboard.html')

def products(request):
    return render(request, 'accounts/products.html')

def customer(request):
    return render(request, 'accounts/customer.html')
```

Output