**Section 2.Hello World**

**8. Intro**

* We are going to create first web application using Django.
* Django is written in Python programming language.
* It is very easy to create Django web application.
* It makes easy, more powerful and faster web application development.
* Django is very powerful web application framework.

1. **A Fresh Virtualenv**

* Create fresh virtual environment for project.
* Create virtual environment using following command,

**python –m venv NameOfVirtualEnvironment**

* Activate environment,

**NameOfVirtualEnvironment\Scripts\activate.bat**

* Deactivate environment,

**NameOfVirtualEnvironment\deactivate**

* After activation of environment run the command,

**where python**

* Get python interpreter path from above command and change workspace setting of **pythonPath** variable.
* Using **pip list** command in our virtual environment, it shows the current modules in environment.
* We need to install django version 1.11.17 in this virtual environment, so we will run,

**pip install django==1.11.17**

* Run **pip freeze** command, copy paste the output, in requirements.txt
* File requirement.txt contains our required module names , this is useful when you want to give requirements of your project to other person and uploading your module in cloud as well.

**10. Hello World**

* Use the Django version used in this course.
* Use Django 1.11.\*
* To create new project using Django as follows,

**django-admin startproject ProjectName**

* This creates brand new project folder with name ProjectName
* Go in this folder and run **python manage.py runserver**, it will run development server on our machine.
* You can give port number,

python manage.py runserver PortNumber

* In this way we have created, our django project.
* It creates package with project name(\_\_init\_\_.py, urls.py, settings.py, and wsgi.py ), sqlite-db and manage.py.
* Using manage.py we can do all the administrative tasks. For more details check <https://djangobook.com/mdj2-django-structure/>
* **Note:** Inside the outer ProjectName folder are two files:

**db.sqlite3**. The database created when you ran the migrate command.

**manage.py**. A command-line utility for executing Django commands from within your project.

The inner **ProjectName** folder is your Django website application. This is the one application that Django creates automatically for you. Because Django is a web framework, it assumes you are going to want a website app

* The **\_\_init\_\_.py** file tells Python that this folder (your Django app) is a Python package.
* **settings.py** contains the settings for your Django project. Every Django project must have a settings file. By convention, Django puts it in your website app, but it doesn’t have to live there.
* **urls.py** contains project-level URL configurations. By default, this contains a single URL pattern for the admin.
* **wsgi.py** enables WSGI compatible web servers to serve your project. This is used at the production time.

**11. Render HTML**

* After running above code, we get web page which is blank, we need to add something in this page.
* In our webapp package , we will create views.py
* **Note:** A view function, or “view” for short, is simply a Python function that takes a web request and returns a web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, etc. Example: You use view to create web pages, note that you need to associate a view to a URL to see it as a web page. In Django, views have to be created in the app views.py file.
* In views.py we have created function home\_page, which takes request as an argument. Basically happening is we go to URL and this function is going to call, which takes htttp request as an argument, and returns response (it is just request-response).
* We need to associate this view function to URL to see it as a web page. So we need to do relative import this function in our urls.py (it is only one file for our whole django project). Add this to urlpatterns list as **url(r’^$’,home\_page).**
* **Note:** Python offers two different primitive operations based on regular expressions: **match** checks for a match only at the beginning of the string, while **search** checks for a match anywhere in the string. **”^”** meansmatches beginning of line. **”$”** means matches end of line.
* **Note:** Raw strings are raw string literals that treat backslash (\) as a literal character. For example, if we try to print a string with a “\n” inside, it will add one line break. But if we mark it as a raw string, it will simply print out the “\n” as a normal character. Python raw strings are useful for writing regular expressions and for using with SQL parsers*.*
* If we refresh the server, it will render the content on the web page. In the same way we can render the html content, using html tags in content to render.

**12. Django Template**

* We will go to website [www.getbootstrap.com](http://www.getbootstrap.com) , we are going to use v4, we use this to style our ecommerce page.
* From this website , we will copy starter template, just paste in views.py’s home\_page function ,using multiline python string format, so now our html response will return template. Refresh the server , you can see changes.
* We have imported render, this **render allows us to render html file.** Writing html code in views.py itself is bad practice.
* So we will go in settings.py file, we will search for templates area,

there DIRS list variable, here we are going to setup a folder or directory in which we are going to hold our html, we will make following changes, ‘DIRS’:[os.path.join(BASE\_DIR,’templates’)], BASE\_DIR is the directory where manage.py exist. So we will create templates folder in that directory.

* We have created home\_page.html file in templates folder, and render it in views.py, **render takes two arguments request and template name**. <https://docs.djangoproject.com/en/3.0/topics/http/shortcuts/#render> , see this documentation for more information.
* URL is handled by function in views.py.
* When we go to the webpage, we ask for something on that webpage(i.e we are asking for something) and server has to know , what that something is , in our case that something is view(we asked something to url, that url looks for view), that view returns back something, in our case it is returning basic html page.

**13. Template Context**

* We have created three views functions for, home\_page, about\_page and contact\_page.
* Now we can render our template.
* We want to change the data that is displayed on the page.
* We can pass something (context) to our template, we can pass is dictionary.
* Context is some extra data, that we pass into our html document.
* We can pass that extra data using dictionary.
* We use context key is used on html page to display its value in html page. {{ title }}, in this way we use context key in html file, then it will show up its value.

**14. HTML Form**

* Rendering html with some data is important ,but we build web application have aspect of being smart, i.e it remembers the things , it is big reason behind building web application.
* We want to collect the data and remember the data.
* We know html form.
* Here we will create contact form.
* Django try to load html template by default from templates folder.
* In templates folder we create contact folder, in that we create view.html file.
* In view.html, we will create form using form tag.
* By default , for form there is GET method.
* **When we want to send data to backend , we use POST method.**
* When we refresh the webpage ,we get error i.e CSRF verification failed.<https://portswigger.net/web-security/csrf/tokens>, <https://stackoverflow.com/questions/5207160/what-is-a-csrf-token-what-is-its-importance-and-how-does-it-work> , this is security measure.
* To solve this error we need to add in html file is {% csrf\_token %}, our view does not have to handle this.
* It is like, we navigate to a page, when we submit form it navigates to a page, it is type of navigation ( it is different kind of request, here it is POST request). POST means some data is coming from that form.
* **request.POST**, it returns query dictionary.
* **<QueryDict:{'csrfmiddlewaretoken': ['tyRr3ij3bFG67Dh2F9pgMvCUDIhEzyKuv4nPGUbuSwY8iVBaeCJv3Wi7YPv1gFEb'], 'full\_name': ['John']}>**
* The name attribute of frontend data, we use at the backend to access data, using method **request.POST.get(“full\_name”),** using this we got particular key’s value. It gives **John**.
* This created form has problem, it doesn’t validate the data, so we will got for Django form.

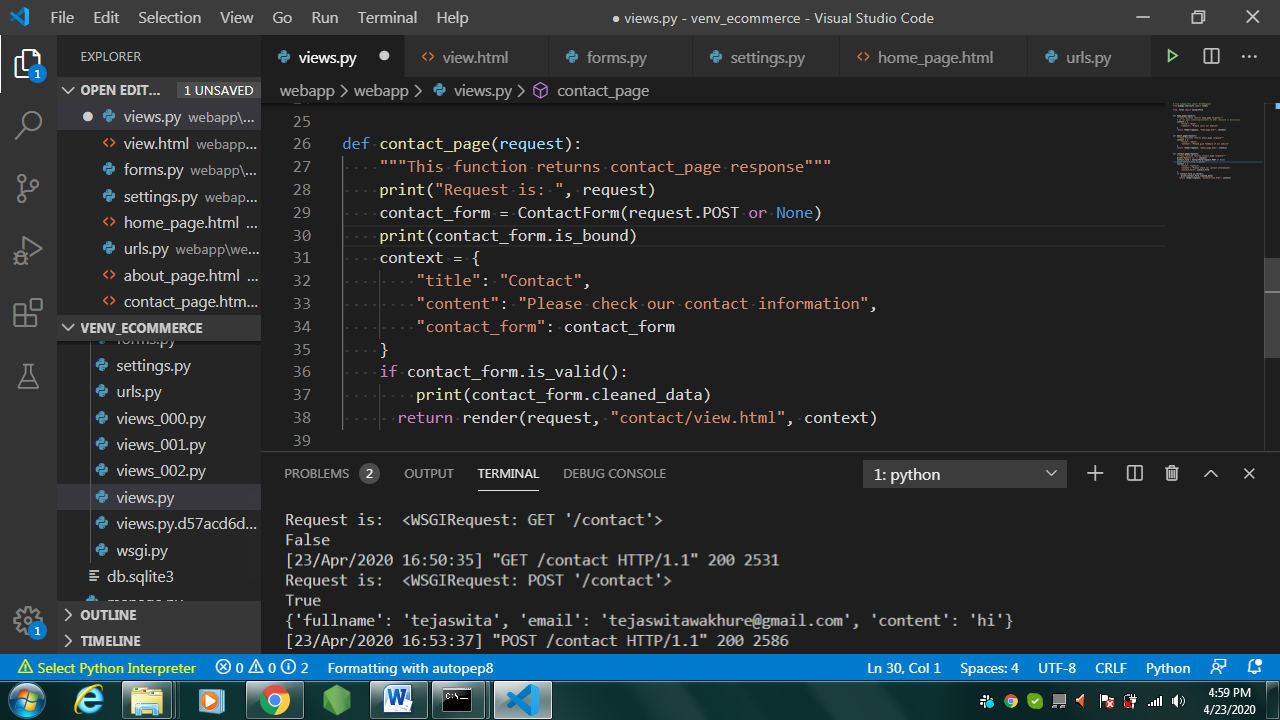
**15. Django Forms**

* In previous video, we used html form , in this we have written hardcoded form, as we have written everything for this form.
* But we want effective way to do this.
* Django provides, very robust and easy way to render out fields and also validate them to make sure no possible errors with it.
* As we created views.py module, same way we will create forms.py in our application.
* We created ContactForm class, which is inherited from forms.Form class , Form is built in class in Django for forms. Here we have designed ContactForm.
* This class creates form that we want with required fields.
* We need to import this class in our views.py. <https://realpython.com/absolute-vs-relative-python-imports/>
* We will create instance of this form in views.py, and will pass to html.
* As we need fields in proper format, so we will go for reference for the form built-in field classes <https://docs.djangoproject.com/en/3.0/ref/forms/fields/>
* **Note:** Naturally, the **forms** library comes with a set of **Field** classes that represent common validation needs. For each field, we describe the default widget used if you don’t specify **widget**.

A **widget** is **Django's** representation of an HTML input element. The **widget** handles the rendering of the HTML, and the extraction of data from a GET/POST dictionary that corresponds to the **widget**. <https://docs.djangoproject.com/en/3.0/ref/forms/widgets/>

Eg. default widget for CharField is TextInput, that means we can update this TextInput.(see code) (search django customizing widget instances)

* We have widgets and classes to look better in forms.py ContactForm, so we have designed frontend too. In views.py we have covered how to handle this data.
* forms.TextInput(), it means render out text input form.
* We will search Textarea class, it is rendered as Textarea, for content area , we want to this widget.
* We will use EmailInput widget for EmailField.(views\_002.py)
* We want to grab the frontend data and passing it through the contact form, so in this ContactForm(or request form) we are going to pass request.POST or None, it tells if there is POST data passed to this form or if not pass i.e None .



* **Note:**

Form validation happens when the data is cleaned. If you want to customize this process, there are various places to make changes, each one serving a different purpose. Three types of cleaning methods are run during form processing. These are normally executed when you call the **is\_valid()** method on a form. There are other things that can also trigger cleaning and validation (accessing the **errors** attribute or calling **full\_clean()** directly), but normally they won’t be needed.

<https://docs.djangoproject.com/en/3.0/ref/forms/validation/>

* So contact\_form.is\_valid() will always return False. You need to bind your form to the post data with form=ContactForm(request.POST)
* **Note:**

A [**Form**](https://docs.djangoproject.com/en/3.0/ref/forms/api/#django.forms.Form) instance is either **bound** to a set of data, or **unbound**.

If it’s **bound** to a set of data, it’s capable of validating that data and rendering the form as HTML with the data displayed in the HTML.

If it’s **unbound**, it cannot do validation (because there’s no data to validate!), but it can still render the blank form as HTML.

If you need to distinguish between bound and unbound form instances at runtime, check the value of the form’s **[is\_bound](https://docs.djangoproject.com/en/3.0/ref/forms/api/" \l "django.forms.Form.is_bound" \o "django.forms.Form.is_bound)** attribute

If you have a bound [**Form**](https://docs.djangoproject.com/en/3.0/ref/forms/api/#django.forms.Form) instance and want to change the data somehow, or if you want to bind an unbound [**Form**](https://docs.djangoproject.com/en/3.0/ref/forms/api/#django.forms.Form) instance to some data, create another [**Form**](https://docs.djangoproject.com/en/3.0/ref/forms/api/#django.forms.Form) instance. There is no way to change data in a [**Form**](https://docs.djangoproject.com/en/3.0/ref/forms/api/#django.forms.Form) instance. Once a [**Form**](https://docs.djangoproject.com/en/3.0/ref/forms/api/#django.forms.Form) instance has been created, you should consider its data immutable, whether it has data or not.

<https://docs.djangoproject.com/en/3.0/ref/forms/api/>

* In the process, **Django** creates an **attribute** called cleaned\_data , a dictionary which contains **cleaned data** only from the fields which have passed the validation tests. Note that cleaned\_data **attribute** will only be available to you after you have invoked the is\_valid() method.
* Observe above screenshot , it shows that when there is GET request, data is not bound to form and when there is POST request data get bound to the form.
* After running this code data will sit in the fields until we cleared out. Here we can see the data that is coming through very similar to request POST ,but it can stay there if there are errors, this data with error will not get accessed at the backend, only valid data get accessed at the backend.
* We need to add some validation or custom validation, we will define functions in class ContactForm.
* We will add validation for email.(see code)
* You should only do form validation it the request is a POST.
* You should always redirect after a post request, even if it’s back to the same page this will rest the values, else we have to clear out those values.

**16. User Login**

* Login to the actual service.
* For this we have to define new function login\_page, views.py to handle this functionality.
* We will create LoginForm class in forms.py
* We need to import this class in views.py to create its instance.
* We have to create auth directory in templates folder.