

# WEB TECHNOLOGY PROJECT

## DOCUMENTATION



Submitted By:

VAIBHAV HARIT (2K16/SE/86)

SAHAJ BHALLA (2K16/SE/70)

SACHIN KHANDEWAL (2K16/SE/68)

# **TABLE OF CONTENTS**

## **1. Python**

- a. Built-in data types**
- b. Advantages**
- c. Embedded Python**

## **2. Django**

- a. Installation**
- b. Setting up MySQL**
- c. Widget tweaks**

## **3. HTML, CSS, Bootstrap, Javascript**

## **4. Django (MVT) Framework**

- a. Model**
  - i. Validation**
- b. View**
- c. Template**
- d. Routes**

## **5. Database**

# **PYTHON**

**Python** is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991.

An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java.

We've used python version 3 or above.

## **BUILT IN DATA TYPES**

Python comes with lots of things already baked in, and provides you with tons of tools to use and hit the road running.

Data types are types of “things” that are mainly used to represent data, such as numbers, text, and other values. This is basically the “stuff” that we as a Python programmers will work with.

The following are the “kinds of things” (objects) that cover like 98% of all built-in data types that you'll be using on a day to day basis, i.e. these are being used all over the place:

- Strings (texts)
- True, False, and Nil
- Symbols
- Arrays
- Hashes
- Numbers

There are more data types, but those are rather exotic, and used much less often.

### **Flow of Control:**

Python has following control flow structure

- if-else
- until
- while
- for
- unless

- times

## **ADVANTAGES OF PYTHON**

The Python language has diversified application in the software development companies such as in gaming, web frameworks and applications, language development, prototyping, graphic design applications, etc. This provides the language a higher plethora over other programming languages used in the industry. Some of its advantages are-

### **Extensive Support Libraries**

It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Most of the highly used programming tasks are already scripted into it that limits the length of the codes to be written in Python.

### **Integration Feature**

Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or Java via Jython. Python also processes XML and other markup languages as it can run on all modern operating systems through same byte code.

### **Improved Programmer's Productivity**

The language has extensive support libraries and clean object-oriented designs that increase two to ten fold of programmer's productivity while using the languages like Java, VB, Perl, C, C++ and C#.

### **Productivity**

With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications.

### **Sample Python Code**

Here is a sample Python code to add two numbers.

```
# This program adds two numbers
```

```
num1 = 1.5
num2 = 6.3

# Add two numbers
sum = float(num1) + float(num2)

# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1,
num2, sum))
```

**Output** – This will produce the following result –  
The sum of 1.5 and 6.3 is 7.8

## EMBEDDED PYTHON

Python can be used in embedded, small or minimal hardware devices, depending on how limiting the devices actually are.

Devices capable of running CPython

Some modern embedded devices have enough memory and a fast enough CPU to run a typical Linux-based environment, for example, and running CPython on such devices is mostly a matter of compilation (or cross-compilation) and tuning.

Devices which could be considered as "embedded" by modern standards and which can run tuned versions of CPython include the following:

- Gumstix
- Raspberry Pi
- BeagleBone Black
- FIC Neo1973 and Neo FreeRunner (Python on Openmoko)
- Telit GSM/GPRS modules (also available as AarLogic family GPRS/GPS QUAD Band Modules)

See also [PythonForArmLinux](#) and [OpenEmbedded](#).

Work to improve CPython for embedded applications

Various efforts have been made to make CPython more usable for embedded applications:

- Patches in the [OpenEmbedded](#) repository

- Cross-compilation issues: 1006238, 5404, 3871
- General interpreter startup costs: SpeedUpInterpreterStartup
- File access overhead on startup: Improving interpreter startup speed, Tons of stats/opens to non-existing files increases Python's startup on loaded NFS servers, Startup time
- Import-related costs: `__file__`
- Using a launcher process where "expensive" modules are required: Introducing python-launcher

## Reduced or reworked Python implementations

Some devices may be more restrictive in that the typical memory footprint of CPython may exceed the amount of memory available on the device. In such cases, a re-engineered or adapted version of CPython, perhaps even to the point where it can be considered a new implementation of Python, might be appropriate.

Examples of such implementations include the following:

- PyMite
- Tiny Python
- Zerynth formerly Viper

On the other hand, one can start with a full build, and simply remove unneeded modules, *e.g.*, Tkinter, etc., to realize a reduced-size Python with little effort.

## Python-based tools for developing embedded applications

Sometimes the embedded environment is just too restrictive to support a Python virtual machine. In such cases, various Python tools can be employed for prototyping, with the eventual application or system code being generated and deployed on the device.

Tools that support this kind of development include the following:

- MyHDL
- WhatOS

# DJANGO

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

## INSTALLING DJANGO

### Install Python

Being a Python Web framework, Django requires Python. See [What Python version can I use with Django?](#) for details.

Get the latest version of Python at <https://www.python.org/downloads/> or with your operating system's package manager.

### Install Apache and **mod\_wsgi**

If you just want to experiment with Django, skip ahead to the next section; Django includes a lightweight web server you can use for testing, so you won't need to set up Apache until you're ready to deploy Django in production.

### Get your database running

If you plan to use Django's database API functionality, you'll need to make sure a database server is running. Django supports many different database servers and is officially supported with PostgreSQL, MySQL, Oracle and SQLite.

### Install the Django code

### Installing an official release with **pip**

This is the recommended way to install Django.

1. Install **pip**. The easiest is to use the standalone pip installer. If your distribution already has **pip** installed, you might need to update it if it's outdated. If it's outdated, you'll know because installation won't work.

2. Take a look at `virtualenv` and `virtualenvwrapper`. These tools provide isolated Python environments, which are more practical than installing packages systemwide. They also allow installing packages without administrator privileges. The contributing tutorial walks through how to create a `virtualenv`.
3. After you've created and activated a virtual environment, enter the command **`pip install Django`** at the shell prompt.

The appropriate libraries are installed for all versions of Python that we support, so if you're not using a `virtualenv`, to access a MySQL database just `import MySQLdb`.

If you *are* using a `virtualenv`, you'll need to install the correct package yourself. Start a bash console inside the `virtualenv`, then:

For Python 2.7

```
pip install mysql-python
```

For Python 3.x

```
pip install mysqlclient
```

```
python manage.py migrate --run-syncdb
```

We've used Django version 2 or above in our site.



## WIDGET TWEAKS

When it comes to build forms, Django Forms can be really handy. If your application provide ways for the end-user to input data, it's strongly advised to do so through the built-in Django Forms. It will automate a good amount of work as well as providing a really stable and secure functionality.

In a nutshell, Django handles three distinct parts of the work involved in forms:

1. Preparing and restructuring data to make it ready for rendering;
2. Creating HTML forms for the data;
3. Receiving and processing submitted forms and data from the client.

The parts 1 and 3 are usually fine for the most cases. But when it comes to the actual HTML forms rendering, sometimes it lacks some options.

That's where the Django Widget Tweaks takes place.

Before we start to talk about the Django Widget Tweaks package itself, I wanted to elaborate a little bit more about the problem I usually face that motivated me to look for this solution.

Installation:

```
pip install django-widget-tweaks
```

Usage:

The `render_field` tag should be easier to use and should make form field customizations much easier for designers and front-end developers.

The template filters are more powerful than the `render_field` tag, but they use a more complex and less HTML-like syntax.

*render\_field*

This is a template tag that can be used as an alternative to aforementioned filters. This template tag renders a field using a syntax similar to plain HTML attributes.

*attr*

Adds or replaces any single html attribute for the form field.

### *add\_class*

Adds CSS class to field element. Split classes by whitespace in order to add several classes at once.

### *set\_data*

Sets HTML5 data attribute ( <http://ejohn.org/blog/html-5-data-attributes/> ). Useful for unobtrusive javascript. It is just a shortcut for 'attr' filter that prepends attribute names with 'data-' string.

### *append\_attr*

Appends attribute value with extra data.

### *add\_error\_class*

The same as 'add\_class' but adds css class only if validation failed for the field (field.errors is not empty).

### *field\_type and widget\_type*

'field\_type' and 'widget\_type' are template filters that return field class name and field widget class name (in lower case).

## HTML

First developed by Tim Berners-Lee in 1990, HTML is short for HyperText Markup Language. HTML is used to create electronic documents (called pages) that are displayed on the World Wide Web. Each page contains a series of connections to other pages called hyperlinks. Every web page you see on the Internet is written using one version of HTML code or another.

In our application in the views we have extensively used the HTML version 5. With HTML we have also used haml. Haml is short form which replace many HTML tags

## CSS

Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language.

We have used CSS with Sassy CSS in our application. Sassy CSS provides nesting of the CSS ids and classes and adds many features to the existing CSS for less typing and reusing the code.

## BOOTSTRAP

Build responsive, mobile-first projects on the web with the world's most popular front-end component library. *Bootstrap* is an open source toolkit for developing with HTML, CSS, and JS.

We have used the bootstrap in the navigations in our app. We have used Bootstrap version 3 and 4.

Also we included bootstrap-sass gem into our website to add many functionalities of bootstrapping without loading bootstraps classes into our app.

## JAVASCRIPT

**JavaScript** often abbreviated as **JS**, is a high-level, dynamic, weakly typed, prototype-based, multi-paradigm, and interpreted programming language. Alongside HTML and CSS, JavaScript is one of the three core technologies of World Wide Web content production. It is used to make webpages interactive and provide online programs, including video games. The majority of websites employ it, and all modern web browsers support it without the need for plug-ins by means of a built-in JavaScript engine. Each of the many JavaScript engines represent a different implementation of JavaScript, all based on the ECMAScript specification, with some engines not supporting the spec fully, and with many engines supporting additional features beyond ECMA.

## **DJANGO MVT FRAMEWORK**

As you already know, Django is a Python web framework. And like most modern framework, Django supports the MVC pattern. First let's see what is the Model-View-Controller (MVC) pattern, and then we will look at Django's specificity for the Model-View-Template (MVT) pattern.

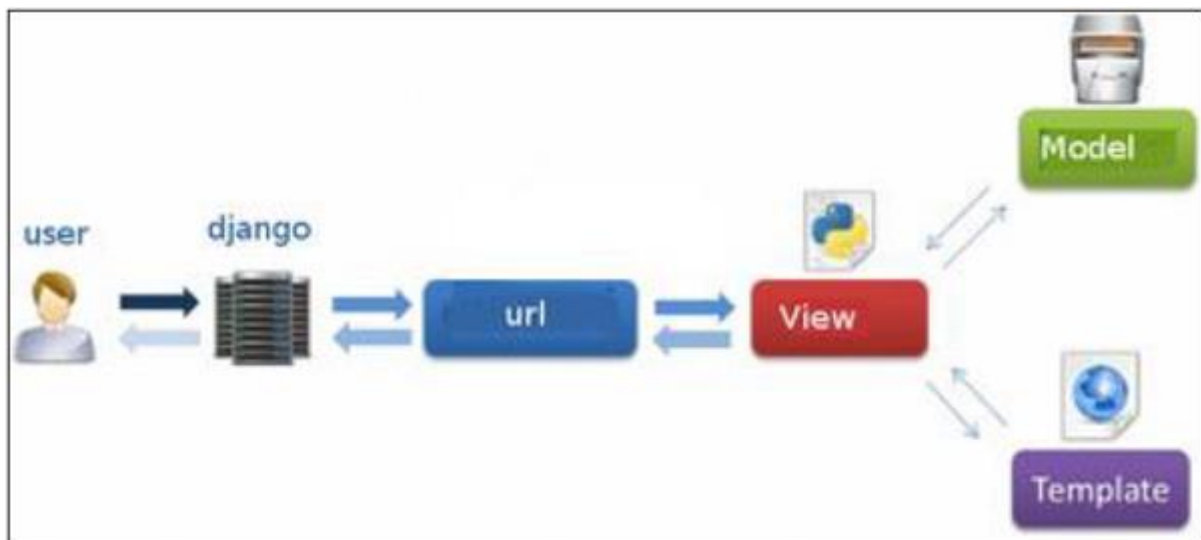
### MVC Pattern

When talking about applications that provides UI (web or desktop), we usually talk about MVC architecture. And as the name suggests, MVC pattern is based on three components: Model, View, and Controller.

### DJANGO MVC - MVT Pattern

The Model-View-Template (MVT) is slightly different from MVC. In fact the main difference between the two patterns is that Django itself takes care of the Controller part (Software Code that controls the interactions between the Model and View), leaving us with the template. The template is a HTML file mixed with Django Template Language (DTL).

The following diagram illustrates how each of the components of the MVT pattern interacts with each other to serve a user request –



The developer provides the Model, the view and the template then just maps it to a URL and Django does the magic to serve it to the user.

So, what exactly is MVT ?

- **M stands for “Model,”** the data access layer. A model is a class that represents table or collection in our DB, and where every attribute of the class is a field of the table or collection This layer contains anything and everything about the data: how to access it, how to validate it, which behaviors it has, and the relationships between the data. Models are defined in the app/models.py

## VALIDATION:

**The implementation of validations is done in a Django model. The data you are entering into the database is defined in the actual Django model, so it only makes sense to define what valid data entails in the same location.**

The validations are –

- The value of name shouldn't be empty.
- The value of message shouldn't be empty.
- The value of email shouldn't be empty.
- The value of email should be a valid e-mail.

## In our site:

Our form incorporates both models.py and forms.py (in the boards folder) using circular import. Hence, models are as given.

## Forms.py

```
from django import forms
from .models import Contact

class ContactForm(forms.ModelForm):
    message = forms.CharField(
        max_length=2000,
        widget=forms.Textarea(),
    )

    def clean(self):
        cleaned_data = super(ContactForm, self).clean()
        name = cleaned_data.get('name')
        email = cleaned_data.get('email')
        message = cleaned_data.get('message')
        if not name and not email and not message:
            raise forms.ValidationError('You have to write something!')

    class Meta:
        model = Contact
        fields = ('name', 'email', 'message',)

    def __init__(self, *args, **kwargs):
        super(ContactForm, self).__init__(*args, **kwargs)
```

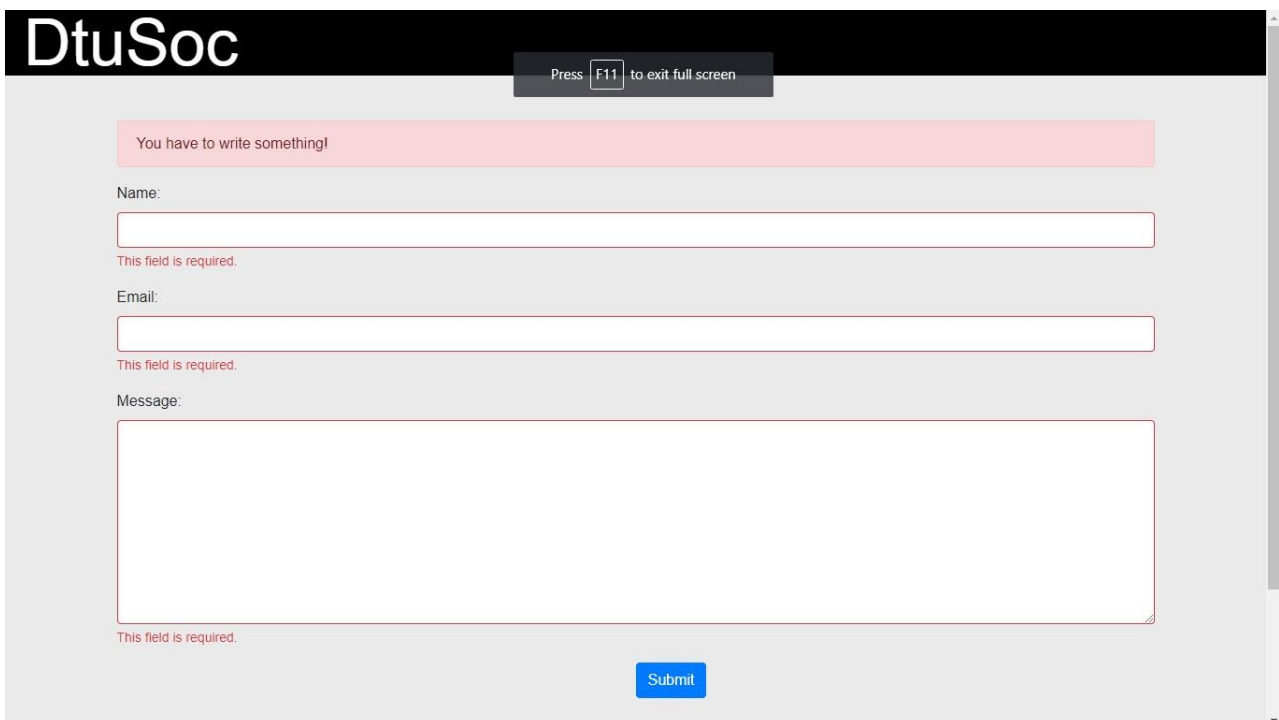
## Models.py

```
from django.db import models
from django import forms

class Contact(models.Model):
    name = models.CharField(max_length=30)
    email = models.EmailField(max_length=254)
    message = models.CharField(
        max_length=2000,
    )
```

The form model is displayed like this:

## Contact.html



The screenshot shows a web browser window with a black header bar containing the text "DtuSoc" in white. Below the header, a grey bar contains the text "Press F11 to exit full screen". The main content area has a light grey background. At the top of this area is a pink rectangular box with the text "You have to write something!". Below this are three form fields: a "Name:" field, an "Email:" field, and a "Message:" field. Each field is a white rectangle with a thin red border. Below each field is a small red text label that says "This field is required.". At the bottom of the form is a blue rectangular button with the text "Submit" in white.

DtuSoc

Press F11 to exit full screen

Name:

sahaj

Email:

This field is required.

Message:

This field is required.

Submit

DtuSoc

Press F11 to exit full screen

Name:

sahaj

Email:

sahaj

Enter a valid email address.

Message:

This field is required.

Submit

# DtuSoc

Press F11 to exit full screen

Name:

Email:

Message:

This field is required.

Submit

# DtuSoc

[f](#)[t](#)[i](#)[You](#)

# DtuSoc

Press F11 to exit full screen

Name:

Email:

Message:

Great site!

This field is required.

Submit

# DtuSoc

[f](#)[t](#)[i](#)[You](#)



- **V stands for “View,”** the business logic layer. This layer contains the logic that accesses the model and defers to the appropriate template(s). You can think of it as the bridge between models and templates.

The views can be found in views.py in boards folder.

```
from django.shortcuts import render_to_response, redirect, render
from .forms import ContactForm
from .models import Contact

def _form_view(request, template_name='basic.html', form_class=ContactForm):
    if request.method == 'POST':
        form = form_class(request.POST)
        if form.is_valid():
            f = form.save(commit=False)
            f.save()
            return render_to_response('thank.html', {'fa': f})
        else:
            form = form_class()
    return render(request, template_name, {'form': form})

def thank(request):
    return render(request, 'thank.html')

def index(request):
    return render(request, 'index.html')

def contact(request):
    return _form_view(request, template_name='contact.html')

def Rotract(request):
    return render(request, 'Rotract.html')

def d42(request):
    return render(request, 'Rotract.html')

def d42(request):
    return render(request, 'd42.html')

def CSI(request):
    return render(request, 'CSI.html')

def NSS(request):
    return render(request, 'NSS.html')

def Aahvaan(request):
    return render(request, 'Aahvaan.html')

def Ecell(request):
    return render(request, 'Ecell.html')

def Parchhayi(request):
    return render(request, 'Parchhayi.html')

def IEEE(request):
    return render(request, 'IEEE.html')
```

These requests are given using URL mapping in the urls.py in pro folder

```
from django.conf.urls import url
from django.contrib import admin
from boards import views

urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^index', views.index, name='index'),
    url(r'^thank', views.thank, name='thank'),
    url(r'^admin/', admin.site.urls),
    url(r'^Retract', views.Retract, name='Retract'),
    url(r'^d42', views.d42, name='d42'),
    url(r'^CSI', views.CSI, name='CSI'),
    url(r'^NSS', views.NSS, name='NSS'),
    url(r'^Ecell', views.Ecell, name='Ecell'),
    url(r'^IEEE', views.IEEE, name='IEEE'),
    url(r'^Parchhayi', views.Parchhayi, name='Parchhayi'),
    url(r'^Aahvaan', views.Aahvaan, name='Aahvaan'),
    url(r'^contact', views.contact, name='contact'),
]
```

**Our home page:**



Delhi Technological University, being one of the biggest universities in India, has a ton of different societies and everyone, especially the students, knows how hard it can be to track and explore the different options of societies and groups that exist which may interest you.



## Explore

DtuSoc is a place where you can find information such as what it's about, multiple photos, contact information, etc. about all the different societies at Delhi Technological University at one place so you can choose and contact the one that interests you the most!



## Our Societies



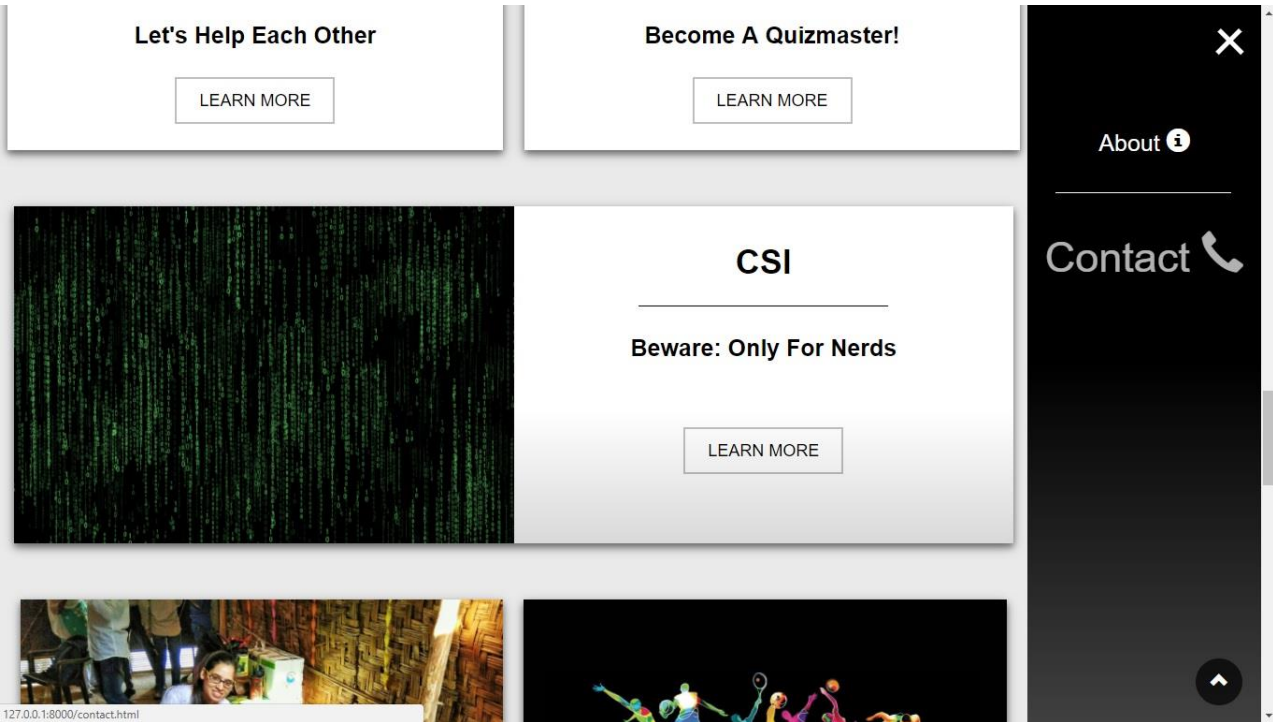
**Retract**



**D42**







## The society page:



**T stands for “Template,”** the presentation layer. This layer contains presentation-related decisions: how something should be displayed on a Web page or other type of document.

### **In our site:**

Our templates in the template folder are as follows:

- index (home page)
- contact (form for messages)
- thank (confirms successful message submission and redirects to home page)
- For different societies:
  - Aahvaan
  - Rotract
  - IEEE
  - CSI
  - NSS
  - ECell
  - D42
  - Parchhayi

The template has tags to display data automatically.  
Our form page looks like this:

```
<form method="post" novalidate>
  {% csrf_token %}

  {% for hidden_field in form.hidden_fields %}
  {{ hidden_field }}
  {% endfor %}

  {% if form.non_field_errors %}
  <div class="alert alert-danger" role="alert">
    {% for error in form.non_field_errors %}
    {{ error }}
    {% endfor %}
  </div>
  {% endif %}

  {% for field in form.visible_fields %}
```

```

<div class="form-group">
    {{ field.label_tag }}

    {% if form.is_bound %}
    {% if field.errors %}
    {% render_field field class="form-control is-invalid" %}
    {% for error in field.errors %}
    <div class="invalid-feedback">
        {{ error }}
    </div>
    {% endfor %}
    {% else %}
    {% render_field field class="form-control is-valid" %}
    {% endif %}
    {% else %}
    {% render_field field class="form-control" %}
    {% endif %}

    {% if field.help_text %}
    <small class="form-text text-muted">{{ field.help_text }}</small>
    {% endif %}
</div>
{% endfor %}

<button type="submit" class="btn btn-primary" style="margin-left: 50%;cursor:
pointer">Submit</button>
</form>

```

After Django receives the request it automatically replaces these tags with the useful information.

## ROUTES

We have following routes in our app with following http verbs

The POST verb is most-often utilized to post messages

The HTTP GET method is used to read (or retrieve) a representation of a resource.

```
Command Prompt - python manage.py runserver
Microsoft Windows [Version 10.0.16299.125]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\HP>cd myproject

C:\Users\HP\myproject>venv\Scripts\activate

(venv) C:\Users\HP\myproject>cd pro

(venv) C:\Users\HP\myproject\pro>python manage.py runserver
Performing system checks...

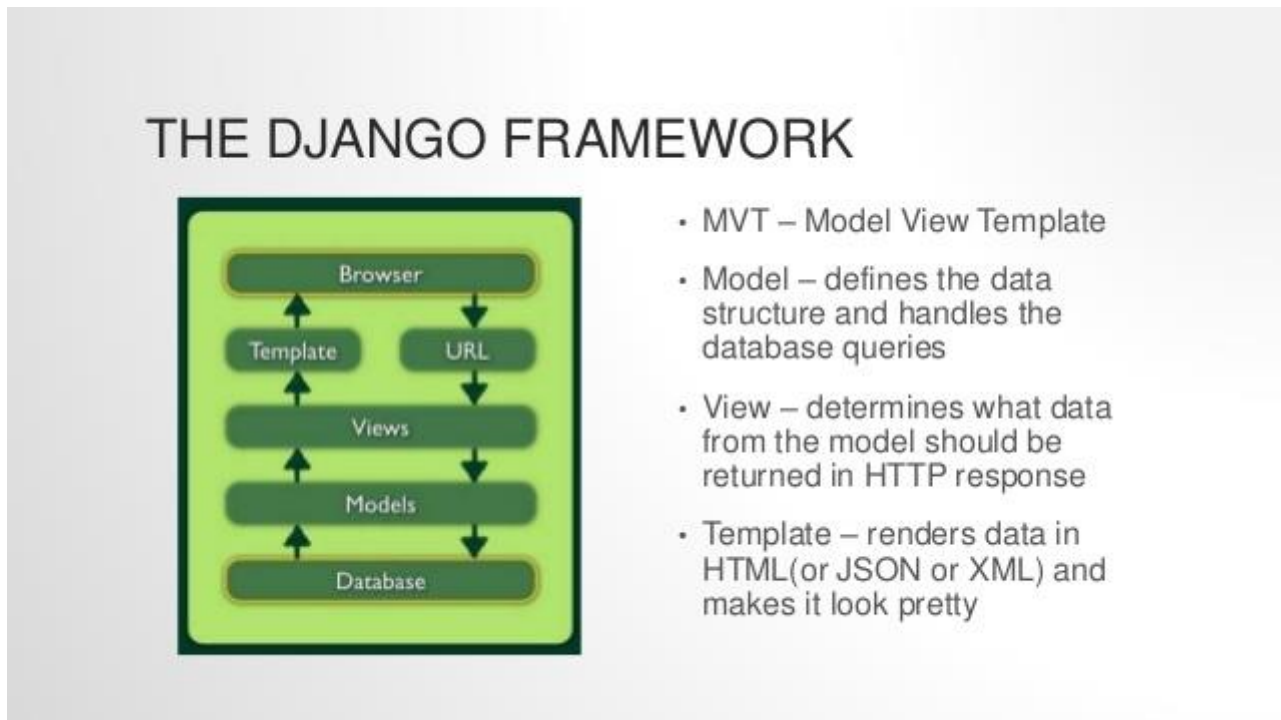
System check identified no issues (0 silenced).
December 25, 2017 - 18:43:35
Django version 2.0, using settings 'pro.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[25/Dec/2017 18:43:43] "GET / HTTP/1.1" 200 13851
[25/Dec/2017 18:43:43] "GET /static/index.js HTTP/1.1" 200 2501
[25/Dec/2017 18:43:43] "GET /static/IEEE.jpg HTTP/1.1" 200 84507
[25/Dec/2017 18:43:43] "GET /static/index.css HTTP/1.1" 200 9609
[25/Dec/2017 18:43:43] "GET /static/Parchhay19.jpg HTTP/1.1" 200 115389
[25/Dec/2017 18:43:43] "GET /static/Aahvaan6.jpg HTTP/1.1" 200 172983
[25/Dec/2017 18:43:43] "GET /static/CSI9.jpg HTTP/1.1" 200 64759
[25/Dec/2017 18:43:43] "GET /static/DTU1.jpg HTTP/1.1" 200 235529
[25/Dec/2017 18:43:43] "GET /static/RotractLogo.jpg HTTP/1.1" 200 608700
[25/Dec/2017 18:43:43] "GET /static/d1.jpg HTTP/1.1" 200 97997
[25/Dec/2017 18:43:43] "GET /static/NSS4.jpg HTTP/1.1" 200 139311
[25/Dec/2017 18:43:43] "GET /static/Aahvaan9.jpg HTTP/1.1" 200 37584
[25/Dec/2017 18:43:43] "GET /static/DTU2.jpg HTTP/1.1" 200 3910925
[25/Dec/2017 18:43:43] "GET /static/CSILogo.jpg HTTP/1.1" 200 1603046
[25/Dec/2017 18:43:43] "GET /static/ecell10.jpg HTTP/1.1" 200 262799
[25/Dec/2017 18:43:43] "GET /static/ieee10.jpg HTTP/1.1" 200 64072
[25/Dec/2017 18:43:43] "GET /static/Parchhay1Logo.jpg HTTP/1.1" 200 580114
[25/Dec/2017 18:43:44] "GET /static/slide01.jpg HTTP/1.1" 200 575353
[25/Dec/2017 18:43:44] "GET /static/slide04.jpg HTTP/1.1" 200 351032

[26/Dec/2017 20:07:49] "GET /favicon.ico HTTP/1.1" 404 3412
[26/Dec/2017 20:07:51] "GET /contact.html HTTP/1.1" 200 3650
[26/Dec/2017 20:08:07] "POST /contact.html HTTP/1.1" 200 4057
[26/Dec/2017 20:08:17] "POST /contact.html HTTP/1.1" 200 3952
[26/Dec/2017 20:08:21] "GET /index.html HTTP/1.1" 200 13851
[26/Dec/2017 20:08:23] "GET /contact.html HTTP/1.1" 200 3650
[26/Dec/2017 20:08:37] "POST /contact.html HTTP/1.1" 200 3233
[26/Dec/2017 20:08:47] "GET /index.html HTTP/1.1" 200 13851
[26/Dec/2017 20:08:58] "GET /Rotract.html HTTP/1.1" 200 11102
[26/Dec/2017 20:08:58] "GET /static/Rotract1.jpg HTTP/1.1" 200 113975
[26/Dec/2017 20:08:58] "GET /static/Rotract2.jpg HTTP/1.1" 200 109172
[26/Dec/2017 20:08:58] "GET /static/Rotract4.jpg HTTP/1.1" 200 97094
[26/Dec/2017 20:08:58] "GET /static/Rotract8.jpg HTTP/1.1" 200 100391
[26/Dec/2017 20:08:58] "GET /static/Rotract3.jpg HTTP/1.1" 200 117311
[26/Dec/2017 20:08:58] "GET /static/Rotract5.jpg HTTP/1.1" 200 71042
[26/Dec/2017 20:08:58] "GET /static/Rotract7.jpg HTTP/1.1" 200 87919
[26/Dec/2017 20:08:58] "GET /static/Rotract6.jpg HTTP/1.1" 200 102120
[26/Dec/2017 20:08:58] "GET /static/Rotract9.jpg HTTP/1.1" 200 41565
[26/Dec/2017 20:09:18] "GET /contact.html HTTP/1.1" 200 3650
[26/Dec/2017 20:09:30] "POST /contact.html HTTP/1.1" 200 3227
[26/Dec/2017 20:09:40] "GET /index.html HTTP/1.1" 200 13851

(venv) C:\Users\HP\myproject\pro>
```

## Pictorial Representation of MVT Framework

Given below is a pictorial representation of Django Framework –



## DATABASE

A database is a collection of information that is organized so that it can be easily accessed, managed and updated.

Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it.

Computer databases typically contain aggregations of data records or files, such as sales transactions, product catalogs and inventories, and customer profiles.

Typically, a database manager provides users with the ability to control read/write access, specify report generation and analyze usage. Some databases



offer ACID (atomicity, consistency, isolation and durability) compliance to guarantee that data is consistent and that transactions are complete.

Databases are prevalent in large mainframe systems, but are also present in smaller distributed workstations and midrange systems.