

EXPERIMENT 11

AIM: Django Web Framework

Program 1: Creating web application using Django web framework

- Installing Django
- Creating project
- Creating App and Views
- Creating and activating model
- Admin interface -Modify database from admin interface

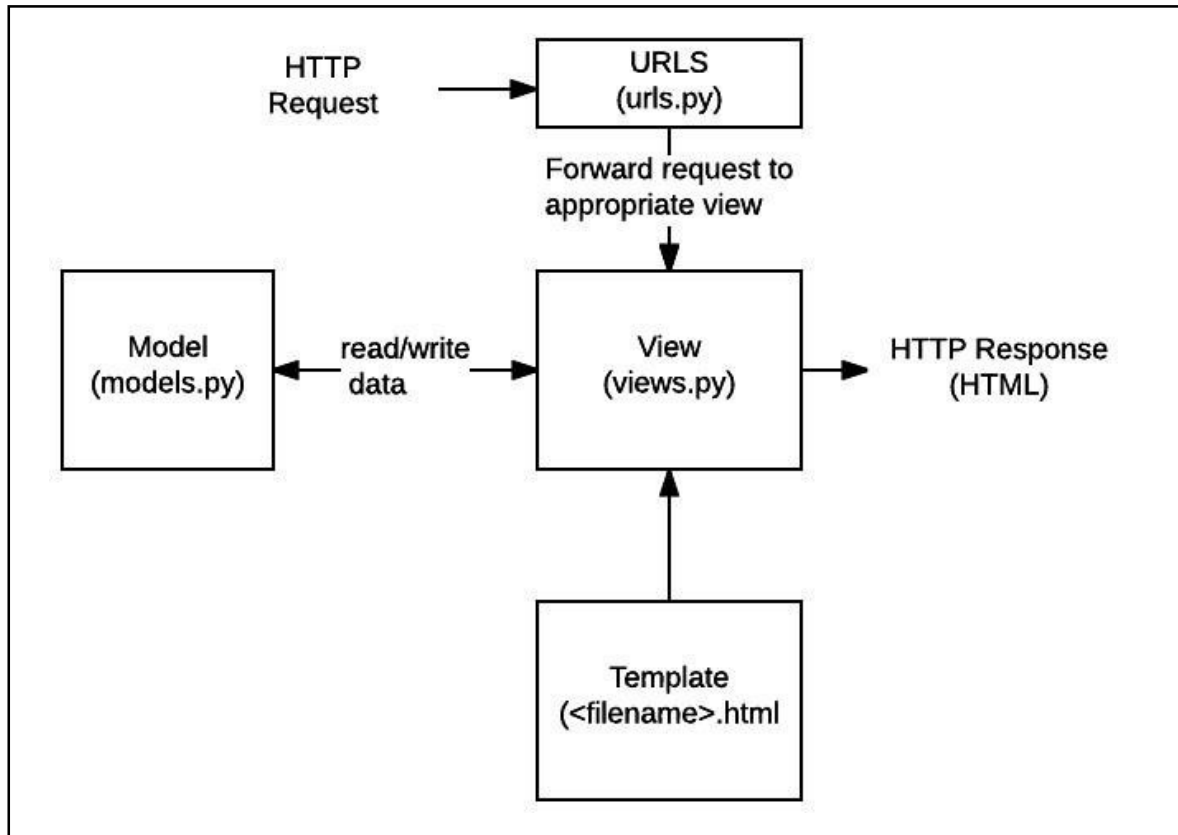
Functions used:

1. Django:

Django is a high-level Python Web framework that encourages rapid development and clean pragmatic design. A Web framework is a set of components that provide a standard way to develop websites fast and easily. Django's primary goal is to ease the creation of complex database-driven websites. Some well known sites that use Django include PBS, Instagram, Disqus, Washington Times, Bitbucket and Mozilla.

In a traditional data-driven website, a web application waits for HTTP requests from the web browser (or other client). When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.

Django web applications typically group the code that handles each of these steps into separate files:



- **URLs:** While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.
- **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via models, and delegate the formatting of the response to templates.
- **Models:** Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
- **Templates:** A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A view can dynamically create an HTML page using an HTML template, populating it with data from a model. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

CODE -

#manage.py

import os

import sys

def

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'todo.settings')try:

from django.core.management import execute_from_command_lineexcept

ImportError as exc:

raise ImportError(

"Couldn't import Django. Are you sure it's installed and "

"available on your PYTHONPATH environment variable? Did you ""forget

to activate a virtual environment?"

) from exc

execute_from_command_line(sys.argv)if__

name__=='__main__': main()

#admin.py from django.contrib import

admin from .models import *

admin.site.register(Task)

#apps.py from django.apps import

AppConfig class

TasksConfig(AppConfig):

name = 'tasks' #views.py from

django.shortcuts import render, redirect from

django.http import HttpResponse from

.models import * from .forms import * def

index(request): tasks = Task.objects.all()

```

form = TaskForm() if request.method
=='POST':
    form = TaskForm(request.POST)if
    form.is_valid():
        form.save()
    return redirect('/')
context = {'tasks':tasks, 'form':form} return
render(request, 'tasks/list.html', context) def
updateTask(request, pk): task =
Task.objects.get(id=pk) form =
TaskForm(instance=task) if request.method ==
'POST':
    form = TaskForm(request.POST,
instance=task) if form.is_valid():
        form.save() return
        redirect('/')
context = {'form':form} return render(request,
'tasks/update_task.html', context) def deleteTask(request,
pk): item = Task.objects.get(id=pk) if request.method ==
'POST':
    item.delete() return redirect('/') context =
{'item':item}
return render(request, 'tasks/delete.html', context)
#models.py from django.db
import models class Task(models.Model):
    title = models.CharField(max_length=200) complete =
    models.BooleanField(default=False)created =
models.DateTimeField(auto_now_add=True)
def
__str__(self):
    return self.title

```

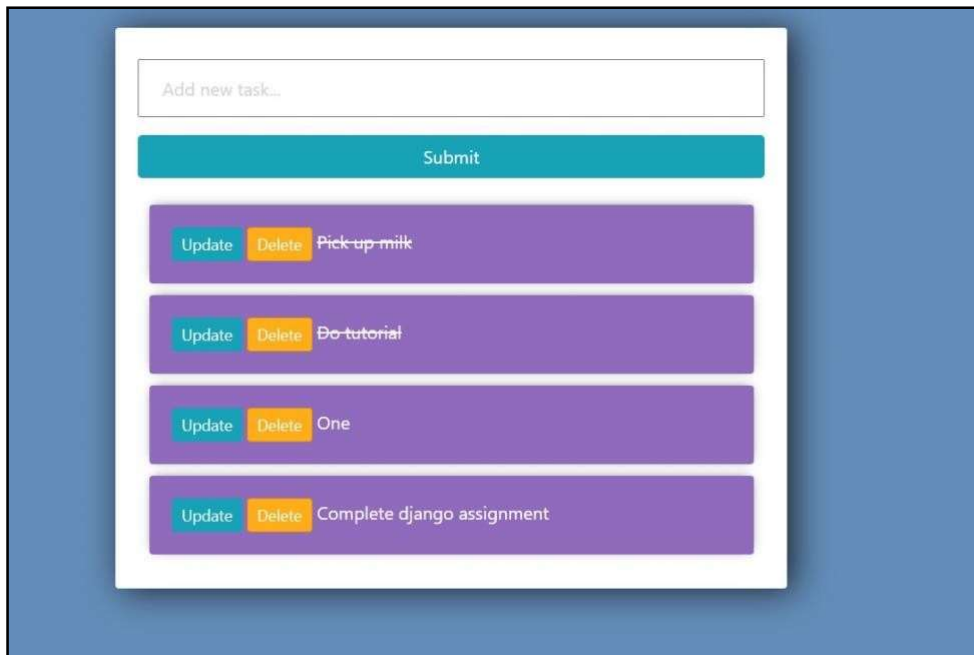
```

#forms.py from django import forms
from django.forms import ModelForm
from .models import * class
TaskForm(forms.ModelForm):
    title= forms.CharField(widget= forms.TextInput(attrs={'placeholder':'Add new
task...'})) class Meta:
        model = Task
        fields = '_all_'

#urls.py from django.urls import path from . import views urlpatterns = [
    path("", views.index, name="list"),    path('update_task/<str:pk>/',
views.updateTask, name="update_task"),    path('delete/<str:pk>/',
views.deleteTask, name="delete"),
]

```

OUTPUT –



The screenshot shows a web application interface for task management. At the top, there is a text input field with the placeholder text "Add new task...". Below the input field is a teal "Submit" button. Underneath the submit button, there is a list of four tasks, each displayed in a purple box. Each task box contains two buttons: a teal "Update" button and an orange "Delete" button, followed by the task description. The tasks listed are:

- Pick-up milk
- Do tutorial
- One
- Complete django assignment

Submit

Update>Delete

Pick up milk

Update>Delete

Do tutorial

Update>Delete

One

Update>Delete

Complete django assignment

Update>Delete

dance practice

Update Task

Title: Complete: ☒

Are you sure you want to delete "One"?

Cancel

Add new task...

Submit

Update

Delete

Pick up milk

Update

Delete

Do tutorial

Update

Delete

Complete django assignment

Update

Delete

dance practice