FortuneCloud
We Understand Technology

Django Framework

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.

Django is a web application framework written in Python programming language. It is based on MVT (Model View Template) design pattern.

Advantages

- Object-Relational Mapping (ORM) Support
- Multilingual Support
- Framework Support
- Administration GUI
- Development Environment
- Rapid Development
- Secure
- Scalable
- Fully loaded
- Versatile
- Open Source
- Vast and Supported Community

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).

Steps to create django project

1. pip install django
2. django-admin startproject mysite
3. cd mysite
4. python manage.py runserver
5. python manage.py startapp myfirstproject
6. create templates folder
7. create index.html in templates folder
8. open settings.py

```
INSTALLED_APPS = [
        'myfirstproject.apps.MyfirstprojectConfig',
]
```

9. settings.py

```
'DIRS': ['templates'],
```

10. open urls.py

```
path('',include('myfirstproject.urls')),
```

11. create urls.py in myfirstproject folder

```
from .import views
urlpatterns = [
        path('',views.index,name="index")
]
```

12. open views.py and create index method

```
def index(request):
        return render(request,"index.html")
```

13. python manage.py runserver

Steps to create django admin

1. python manage.py makemigrations
2. python manage.py migrate
3. python manage.py createsuperuser

   Enter username: admin

   Enter email : admin@gmail.com

   Password : admin

   Confirm password: admin

4. python manage.py runserver

URL Mapping

Django has his own way for URL mapping and it's done by editing your project urls.py file **(myproject/urls.py)**.

it gets user requests by URL locater and responds back.

// urls.py

```
from django.urls import path,include
from .import views

urlpatterns = [
   path('',views.index,name="index")
]
```

Views

A view function, or "view" for short, is simply a Python function that takes a web request and returns a web response.

A view is a place where we put our business logic of the application.

//views.py

```
def index(request):
    return render(request,"index.html")
```

Models

A model is a class that represents table or collection in our database, and where every attribute of the class is a field of the table or collection.

Each model class maps to a single table in the database.

Django Model is a subclass of **django.db.models.Model** and each field of the model class represents a database field (column).

```
from django.db import models

class Person(models.Model):

  fullname = models.CharField(max_length = 50)
  email = models.CharField(max_length = 50)
  phonenumber = models.IntegerField()
```

After that apply migration by using the following command.

python manage.py makemigrations

python manage.py migrate

Django Routing

1. Create paths in **urls.py**

```
 path('',views.index,name="index"),
path('about',views.about,name="about"),
path('contact',views.contact,name="contact")
```

2. Create functions in **views.py**

```
def index(request):
    return render(request,"index.html")

def about(request):
    return render(request,"about.html")

def contact(request):
    return render(request,"contact.html")
```

3. Create index.html, about.html & contact.html pages in **templates** folder

Django Addition Task

1. Urls.py

```
        path('',views.index,name="index"),
```

2. Views.py

```
        def index(request):
                return render(request,"index.html")
```

3. Index.html

```
<form action="add" method="post">
   {% csrf_token %}
   <input type="text" name="num1" placeholder="Number 1"><br><br>
   <input type="text" name="num2" placeholder="Number 2"><br><br>
```

```
    <button type="submit">submit</button>
</form>
```

4. Urls.py

```
        path('add',views.add,name="add"),
```

5. Views.py

```
def add(request):
    if request.method=="POST":
        n1 = request.POST['num1']
        n2 = request.POST['num2']

        result = int(n1) + int(n2)
        print("Addition=",result)

        return HttpResponse("Done")
    else:
        return HttpResponse("Fail")
```

Django registration form using model

1. urls.py

```
        path('',views.index,name="index"),
```

2. views.py

```
    def index(request):
        return render(request,"index.html")
```

3. index.html

```
<form action="registration" method="post">

    {% csrf_token %}

    <input type="text" name="fname" placeholder="First Name"><br><br>
    <input type="text" name="lname" placeholder="Last Name"><br><br>
    <input type="email" name="email" placeholder="Email"><br><br>
    <input type="password" name="password"
placeholder="Password"><br><br>
```

```
   <button type="submit">submit</button>
</form>
```

4. models.py

```
class student(models.Model):
    fname = models.CharField(max_length=20)
    lname = models.CharField(max_length=20)
    email = models.CharField(max_length=50,default="0")
    password = models.CharField(max_length=20,default="0")
```

5. **python manage.py makemgrations**
6. **python manage.py migrate**
7. urls.py

```
        path('registration',views.registration,name="registration"),
```

8. views.py

```
from .models import student

def registration(request):
    if request.method=="POST":
        fname = request.POST['fname']
        lname = request.POST['lname']
        email = request.POST['email']
        password = request.POST['password']

        s =
student(fname=fname,lname=lname,email=email,password=password)
        s.save()  #insert

        return HttpResponse("Registration successfully completed")
    else:
        return HttpResponse("Fail")
```

9. admin.py

```
from .models import student

admin.site.register(student)
```

10. python manage.py runserver

11.login admin panel & check registered student data

Redirect

1. views.py

```
from django.shortcuts import render, redirect

return redirect("welcome")
```

Data display

1. urls.py

```
        path('welcome',views.welcome,name="welcome"),
```

2. views.py

```
def welcome(request):
    data = student.objects.all()  #select
    return render(request,"welcome.html",{'data':data})
```

3. welcome.html

```
<table border="1">
    <tr>
        <th>Id</th>
        <th>Fname</th>
        <th>Lname</th>
        <th>Email</th>
        <th>Password</th>
    </tr>

    {% for i in data %}
    <tr>
        <td>{{i.id}}</td>
        <td>{{i.fname}}</td>
        <td>{{i.lname}}</td>
        <td>{{i.email}}</td>
        <td>{{i.password}}</td>
    </tr>
    {% endfor %}

</table>
```

Data delete – using id

1. welcome.html

```
  <td>
      <a href="delete_stud?id={{i.id}}">Delete</a>
  </td>
```

2. urls.py

```
  path('delete_stud',views.delete_stud,name="delete_stud"),
```

3. views.py

```
def delete_stud(request):
    id = request.GET['id']
    student.objects.filter(id=id).delete()  #delete
    return redirect("welcome")
```

Data Edit (Update profile)

1. welcome.html

```
  <td>
    <a href="edit_stud?id={{i.id}}">Edit</a>
  </td>
```

2. urls.py

```
        path('edit_stud',views.edit_stud,name="edit_stud"),
```

3. views.py

```
 def edit_stud(request):
     id = request.GET['id']
     data = student.objects.all().filter(id=id)
     return render(request,"edit.html",{'data':data})
```

4. edit.html

```
{% for i in data %}
    <form action="update_data" method="post">

        {% csrf_token %}
        <input type="hidden" name="id" value="{{i.id}}">
        <input type="text" name="fname" placeholder="First Name" value="{{i.fname}}"><br><br>
        <input type="text" name="lname" placeholder="Last Name" value="{{i.lname}}"><br><br>
        <input type="email" name="email" placeholder="Email" value="{{i.email}}"><br><br>
        <input type="password" name="password" placeholder="Password"
value="{{i.password}}"><br><br>

        <button type="submit">Update</button>
    </form>
{% endfor  %}
```

5. urls.py

```
        path('update_data',views.update_data,name="update_data"),
```

6. views.py

```
def update_data(request):
    if request.method=="POST":
        id = request.POST['id']
        fname = request.POST['fname']
        lname = request.POST['lname']
        email = request.POST['email']
        password = request.POST['password']


student.objects.filter(id=id).update(fname=fname,lname=lname,email=email,password=p
assword)
```

```
        return redirect("welcome")
    else:
        return redirect("welcome")
```

Custom login using session

Session:

Sessions are the mechanism used by Django (and most of the Internet) for keeping track of the "state" between the site and a particular browser. Sessions allow you to store arbitrary data per browser, and have this data available to the site whenever the browser connects.

1. Urls.py

```
path('login',views.login,name="login"),
```

2. Views.py

```
def login(request):
    return render(request,"login.html")
```

3. login.html

```
<form action="login_check" method="post">

    {% csrf_token %}

    <input type="email" name="email" placeholder="Email"><br><br>
    <input type="password" name="password" placeholder="Password"><br><br>

    <button type="submit">Login</button>
</form>
```

4. urls.py

```
        path('login_check',views.login_check,name="login_check"),
```

5. views.py

```
def login_check(request):
    if request.method=="POST":
        email = request.POST['email']
        password = request.POST['password']

        data = student.objects.all().filter(email=email,password=password)

        if len(data)==1:

            request.session["username"]=email  #session start

            return redirect('dashboard')
        else:
            return redirect('login')
```

6. urls.py

```
        path('dashboard',views.dashboard,name="dashboard"),
```

7. views.py

```
def dashboard(request):
    if request.session.get('username') is not None:
        return render(request,"dashboard.html")
    else:
        return redirect('login')
```

8. dashboard.html

```
<h2>Dashboard Page</h2>
```

```
{{request.session.username}}

<a href="logout">Logout</a>
```

9. urls.py

```
        path('logout',views.logout,name="logout"),
```

10. views.py

```
def logout(request):
    del request.session["username"]  #session end
    return redirect('login')
```

Cookie

Technically, cookies are text files with a small piece of data that the web server sends to a web browser. The web browser may store the cookie and send it back to the web server in subsequent requests.

Cookie has its expiry date and time and removes automatically when gets expire. Django provides built-in methods to set and fetch cookie.

How to add cookie

1. urls.py

```
        path('add_cookie',views.add_cookie,name="add_cookie"),
```

2. views.py

```
def add_cookie(request):
    res = HttpResponse("Cookie set")
    res.set_cookie("student_name","test")
    return res
```

3. Right click in your browser
4. Goto **Inspect**

5. Goto **Application** tab
6. Check in **cookies** section

How to get cookie

1. urls.py

```
        path('view_cookie',views.view_cookie,name="view_cookie"),
```

2. views.py

```
def view_cookie(request):
    name = request.COOKIES["student_name"]
    return HttpResponse("Name="+name)
```

File upload with media folder

1. create **media** folder
2. create **images** folder under **media** folder
3. settings.py

```
   MEDIA_URL = '/media/'
   MEDIA_ROOT = os.path.join(BASE_DIR,'media')
```

4. urls.py

```
        path('file',views.file,name="file"),
```

5. views.py

```
  def file(request):
       return render(request,"file.html")
```

6. file.html

```
<form action="file_upload" method="post" enctype="multipart/form-data">
   {% csrf_token %}
   <input type="text" name="name" placeholder="Name"><br><br>
   <input type="file" name="photo"><br><br>

   <button type="submit">Submit</button>
</form>
```

7. models.py

```
class employee(models.Model):
    name = models.CharField(max_length=20)
    photo = models.FileField(upload_to='images')
```

8. perform migrations
9. urls.py

```
        path('file_upload',views.file_upload,name="file_upload"),
```

10. views.py

```
def file_upload(request):
    if request.method=="POST":
        name = request.POST['name']
        photo = request.FILES['photo']

        e = employee(name=name, photo=photo)
        e.save()

        return HttpResponse("file uploaded successfully")
    else:
        return HttpResponse("Fail")
```

Static files

1. create **static** folder
2. settings.py

```
STATIC_URL = '/static/'
STATICFILES_DIRS=[
   os.path.join(BASE_DIR,'static')
]
STATIC_ROOT = os.path.join(BASE_DIR,'assets')
```

3. use this command to collect static files : **python manage.py collectstatic**

4. Please check **assets** folder is automatically created. In that folder you can now see admin panels css, js, etc.

5. create style.css file in static folder

```
h2{
   color:orange;
   background-color:black;
}
```

6. index.html

```
<link href="{% static 'style.css' %}" rel="stylesheet" type="text/css">
```

7. same as js & imags.

Django ModelForm

ModelForm is a regular Form which can automatically generate certain fields. The fields that are automatically generated depend on the content of the Meta class

Django ModelForm is a class that is used to directly convert a model into a Django form.

It is an efficient way to create a form without writing HTML code. Django automatically does it for us to reduce the application development time.

1. urls.py

```
        path('form',views.form,name="form"),
```

2. views.py

```
def form(request):
    return render(request,"form.html")
```

3. form.html

```
<form action="" method="post">

    {% csrf_token %}



    <button>submit</button>
</form>
```

4. models.py

```
class customer(models.Model):
    fname = models.CharField(max_length=20)
    lname = models.CharField(max_length=20)
```

5. **python manage.py makemgrations**
6. **python manage.py migrate**
7. create **forms.py** in your application folder

```
from django.forms import ModelForm

from .models import customer

class customer_form(ModelForm):
    class Meta:
        model=customer
        fields='__all__'
```

8. views.py

```
from .forms import customer_form

def form(request):
    form = customer_form()
    return render(request,"form.html",{'form':form})
```

9. form.html

```
<form action="" method="post">

    {% csrf_token %}

    {{form}}

    <button>submit</button>
</form>
```

OR

```
        {{form.as_p}}
```

OR

```
<table>
    {{form.as_table}}
</table>
```

Flash messages

Quite commonly in web applications, you need to display a one-time notification message (also known as “flash message”) to the user after processing a form or some other types of user input.

A flash message is a one-time notification message. To display the flash message in Django, you use the messages from django.contrib module

1. settings.py

```
from django.contrib.messages import constants as messages

MESSAGE_TAGS={
   messages.SUCCESS:'alert-success',
   messages.WARNING:'alert-warning'
}
```

2. views.py

```
from django.contrib import messages


def registration(request):
    if request.method=="POST":

        .........

        messages.success(request,"Registration successfully completed")
        return redirect("welcome")
    else:
        messages.warning(request, "Registration fail")
        return redirect("/")
```

3. create new html file **message.html** & include on the page.

```

{% if messages %}
    {% for i in messages %}
        <div class="alert {{i.tags}}" id="msg">
            {{i}}
        </div>

    {% endfor %}
{% endif %}
```

4. welcome.html page (**1. Add bootstrap links in head section 2. Include message.html page 3. Write script for alert remove after some seconds.**)

```
<head>
   <meta charset="UTF-8">
   <title>Title</title>
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</head>
```

```
    {% include 'message.html' %}
```

```
<script>

    setTimeout(function(){
      if($("#msg").length > 0)
      {
         $("#msg").remove();
      }

   },2000);

</script>
```

Authentication

It handles user accounts, groups, permissions and cookie-based user sessions.

The Django authentication system handles both authentication and authorization. Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do. Here the term authentication is used to refer to both tasks.

The auth system consists of:

- Users
- Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
- Groups: A generic way of applying labels and permissions to more than one user.
- A configurable password hashing system
- Forms and view tools for logging in users, or restricting content
- A pluggable backend system

Registration form using **User** model (**create_user**)

1. urls.py

```
        path('signup',views.signup,name="signup"),
```

2. views.py

```
def signup(request):
    return render(request,"signup.html")
```

3. signup.html

```
<form action="user_register" method="post">

   {% csrf_token %}

   <input type="text" name="first_name" placeholder="First Name"><br><br>
   <input type="text" name="last_name" placeholder="Last Name"><br><br>
   <input type="email" name="username" placeholder="Email"><br><br>
   <input type="password" name="password" placeholder="Password"><br><br>

   <button type="submit">submit</button>
</form>
```

4. urls.py

```
        path('user_register',views.user_register,name="user_register"),
```

5. views.py

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


def user_register(request):
    if request.method=="POST":
        fname = request.POST['first_name']
        lname = request.POST['last_name']
        email = request.POST['username']
        password = request.POST['password']
        us =
User.objects.create_user(first_name=fname,last_name=lname,username
=email,password=password)
        us.save()

        return HttpResponse("User added successfully")
    else:
        return HttpResponse("Fail")
```

6. Open admin panel & check in **User** table

Login using **User** model (**authenticate, login, logout**)

1. Urls.py

```
        path('signin',views.signin,name="signin"),
```

2. Views.py

```
def signin(request):
    return render(request,"signin.html")
```

3. signin.html

```
<form action="signin_check" method="post">

   {% csrf_token %}

   <input type="email" name="username" placeholder="Email"><br><br>
   <input type="password" name="password" placeholder="Password"><br><br>

   <button type="submit">Login</button>
</form>
```

4. Urls.py

```
        path('signin_check',views.signin_check,name="signin_check"),
```

5. Views.py

```
def signin_check(request):
    if request.method=="POST":
        username = request.POST['username']
        password = request.POST['password']

        us = authenticate(username=username,password=password)

        if us:
            login(request,us)    #session start
            return HttpResponse("Login successfully")
        else:
            return HttpResponse("Login fail")
```

6. urls.py

```
    path('user_logout',views.user_logout,name="user_logout"),
```

7. views.py

```
def user_logout(request):
    logout(request) #session end
    return HttpResponse("Logout successfully")
```

Reset password

1. urls.py

```
    path('reset',views.reset,name="reset"),
```

2. views.py

```
def reset(request):
    return render(request,"reset.html")
```

3. reset.html

```
<form action="reset_password" method="post">

    {% csrf_token %}

    <input type="email" name="username" placeholder="Username"><br><br>
    <input type="password" name="old_password" placeholder="Old Password"><br><br>
    <input type="password" name="new_password" placeholder="New Password"><br><br>

    <button type="submit">Submit</button>
</form>
```

4. urls.py

```
        path('reset_password',views.reset_password,name="reset_password")
```

5. views.py

```
def reset_password(request):
    if request.method=="POST":
        username = request.POST['username']
        old_password = request.POST['old_password']
        new_password = request.POST['new_password']

        us = authenticate(username=username, password=old_password)

        if us:
            us.set_password(new_password)    #reset password
            us.save()
            return HttpResponse("Password reseted successfully")
        else:
            return HttpResponse("Fail")
```

Django Relationships

Django offers three types of relational fields: ForeignKey, OneToOneField, and ManyToManyField.

OneToOneField Relationship

A OneToOneField is used to define a one-to-one relationship between two models. In other words, a OneToOneField is used when each instance of one model is associated with exactly one instance of another model.

eg. **student have an only one permanent address**

1. create two models in models.py

```python
class Student(models.Model):
    name = models.CharField(max_length=50)
    mobile = models.CharField(max_length=10)
    def __str__(self):
        return self.name


class Address(models.Model):
    student = models.OneToOneField(Student, on_delete=models.CASCADE)
    state = models.CharField(max_length=50)
    city = models.CharField(max_length=50)
```

on_delete=models.CASCADE

- The on_delete CASCADE option can be a powerful tool when you want to ensure that all related data is automatically removed when the parent record is deleted.

2. register in admin.py

```python
from .models import Student,Address

admin.site.register([Student,Address])
```

3. Check – admin panel
4. first - add record in Student model
5. second - add record in Address model
6. Try to delete record from Student model
7. Address is also deleted because of one to one relation

ManyToManyField Relationship

A ManyToManyField is used to define a many-to-many relationship between two models. In other words, a ManyToManyField is used when each instance of one model can be associated with one or more instances of another model, and vice versa.

eg. **Many candidates have many skills**

1. create two models in models.py

```
class Skills(models.Model):
    title = models.CharField(max_length=50)

    def __str__(self):
        return self.title

class Candidate(models.Model):
    name = models.CharField(max_length=50)
    email = models.CharField(max_length=100)
    skills = models.ManyToManyField(Skills)

    def __str__(self):
        return self.name
```

2. register in admin.py

```
from .models import Candidate,Skills

admin.site.register([Candidate,Skills])
```

3. Check admin panel
4. First – add more than one skills in skills model
5. Second – add candidate in candidate model with more than one skills

Foreign Key – (Many To One Relationship)

A ForeignKey field is used to define a many-to-one relationship between two models.

eg. **A single author can write a number of books**

1. create two models in models.py

```
class author(models.Model):
   name = models.CharField(max_length=100)

   def __str__(self):
      return self.name

class book(models.Model):
   title = models.CharField(max_length=200)
   author = models.ForeignKey(author,on_delete=models.CASCADE)

   def __str__(self):
      return self.title
```

2. register in admin.py

```
from .models import author,book

admin.site.register([author,book])
```

3. Check admin panel
4. first- add author in author model
5. second – add multiple books for author in book model
6. Try to delete record from author model
7. All books are deleted because of many to one relation

Django Model Inheritance

Model inheritance is a Django ORM feature that allows developers to create hierarchical relationships between database models. It enables code reuse, extensibility, and a cleaner code

Three types of model inheritance:

1. Abstract Base Classes.
2. Multi-table Inheritance.
3. Proxy Models.

1. Abstract Base Classes

   Abstract base classes provide a way to define common fields and methods that multiple models can inherit.

   a. create **user_information** model for common fields for all users.

```
class user_information(models.Model):
    name = models.CharField(max_length=100)
    email = models.CharField(max_length=100)

    class Meta:
        abstract = True
```

   b. create your models eg. **customer, seller, etc.**

```
class customer(user_information):
    cid = models.IntegerField()

class seller(user_information):
    sid = models.IntegerField()
```

   c. perform migrations & register in admin.py

```
from .models import customer,seller
admin.site.register([customer,seller])
```

   d. Check your admin panel

2. Multi-table Inheritance

   You can use multi-table inheritance when the parent model also needs to exist as a table in the database alongside the child model.

   a. create **Person** model with common fields for all users

```
class Person(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)

    class Meta:
        abstract = True
```

b. create **Employee** model & using inheritance with **Person**

```
class Employee(Person):
    employee_id = models.CharField(max_length=20)
```

c. create **Manager** model & using inheritance with **Employee**

```
class Manager(Employee):
    title = models.CharField(max_length=100)
```

d. register only **Manager** model in admin.py

```
from .models import Manager
admin.site.register([Manager])
```

e. Check admin panel

3. Proxy Models

A proxy model helps you create a new model that extends from an existing model without creating a new database table.
In this kind of model inheritance, the proxy and original models will share the same table.

a. create two models in models.py

```
class jobpost(models.Model):
    title = models.CharField(max_length=100)
    designation = models.CharField(max_length=100)
    no_of_openings = models.IntegerField()

    def __str__(self):
        return self.title

class proxy_jobpost(jobpost):
    class Meta:
        proxy = True
```

b. register both models py admin.py

```
from .models import jobpost,proxy_jobpost
admin.site.register([jobpost,proxy_jobpost])
```

c. Check admin panel
d. Add data in **jobpost** model
e. Data is also visible in **proxy_jobpost**.