Django Introduction

Django is a web application framework written in Python programming language. It is based on MVT (Model View Template) design pattern. The Django is very demanding due to its rapid development feature. It takes less time to build application after collecting client requirement.

This framework uses a famous tag line:The web framework for perfectionists with deadlines.

By using Django, we can build web applications in very less time. Django is designed in such a manner that it handles much of configure things automatically, so we can focus on application development only.

History

Django was design and developed by Lawrence journal world in 2003 and publicly released under BSD license in July 2005. Currently, DSF (Django Software Foundation) maintains its development and release cycle.

Django was released on 21, July 2005. Its current stable version is 2.0.3 which was released on 6 March, 2018.

Popularity

Django is widely accepted and used by various well-known sites such as:

1. Instagram

2. Mozilla

3. Disqus

4. Pinterest

5. Bitbucket

6. The Washington Times

Features of Django

1. Rapid Development

Django was designed with the intention to make a framework which takes less time to build web application. The project implementation phase is a very time taken but Django creates it rapidly.

2. Secure

Django takes security seriously and helps developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery etc. Its user authentication system provides a secure way to manage user accounts and passwords.

3. Scalable

Django is scalable in nature and has ability to quickly and flexibly switch from small to large scale application project.

4. Fully loaded

Django includes various helping task modules and libraries which can be used to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds etc.

5. Versatile

Django is versatile in nature which allows it to build applications for different-different domains. Now a days, Companies are using Django to build various types of applications like: content management systems, social networks sites or scientific computing platforms etc.

6. Open Source

Django is an open source web application framework. It is publicly available without cost. It can be downloaded with source code from the public repository. Open source reduces the total cost of the application development.

7. Vast and Supported Community

Django is an one of the most popular web framework. It has widely supportive community and channels to share and connect.

Django Project

To create a Django project, we can use the following command. projectname is the name of Django application.

$ django-admin startproject projectname

Django Project Example

Here, we are creating a project djangpapp in the current directory.

1. Create project

>django-admin startproject djangpapp

2. Locate into the Project

Now, move to the project by changing the directory. The Directory can be changed by using the following command.

>cd djangpapp

3. Running the Django Project

Django project has a built-in development server which is used to run application instantly without any external web server. It means we don't need of Apache or another web server to run the application in development mode.

To run the application, we can use the following command.

> python3 manage.py runserver

Django Virtual Environment

The virtual environment is an environment which is used by Django to execute an application. It is recommended to create and execute a Django application in a separate environment. Python provides a tool virtualenv to create an isolated Python environment.

Django Admin Interface

Django provides a built-in admin module which can be used to perform CRUD operations on the models. It reads metadata from the model to provide a quick interface where the user can manage the content of the application.

This is a built-in module and designed to perform admin related tasks to the user.

1. Run the project using runserver command

2. Run the command to apply basic tables to the project

    > python manage.py migrate

3. Run the following command to createsuperuser

    >python manage.py createsuperuser

Prompt will come to ask username , email id , password and confirm password.

4. After successful prompt message again open admin interface and insert the login credentials.

5. Login done successfully.

Django App

Django application consists of project and app, it also generates an automatic base directory for the app, so we can focus on writing code (business logic) rather than creating app directories.

The difference between a project and app is, a project is a collection of configuration files and apps whereas the app is a web application which is written to perform business logic.

Creating an App

To create an app, we can use the following command.

>python3 manage.py startapp appname

Django App Example

> python3 manage.py startapp myapp

See the directory structure of the created app, it contains the migrations folder to store

migration files and model to write business logic.

Initially, all the files are empty, no code is available but we can use these to implement business logic on the basis of the MVC design pattern.

First Program in Django

1. Open views.py file in any text editor and write the given code to it.

```
// views.py

from django.shortcuts import render

# Create your views here.

from django.http import HttpResponse

 def hello(request):

   return HttpResponse("<h2>Hello, Welcome to Django!</h2>")
```

2. Open urls.py file in any text editor and write the given code to it

```
// urls.py

from django.contrib import admin

from django.urls import path

from myapp import views

 urlpatterns = [

   path('admin/', admin.site.urls),

   path('hello/', views.hello),

]
```

3. Run the Application

```
$ python3 manage.py runserver
```

We have made changes in two files of the application. Now, let's run the it by using the following command. This command will start the server at port 8000.

Open any web browser and enter the URL localhost:8000/hello. It will show the output given below.

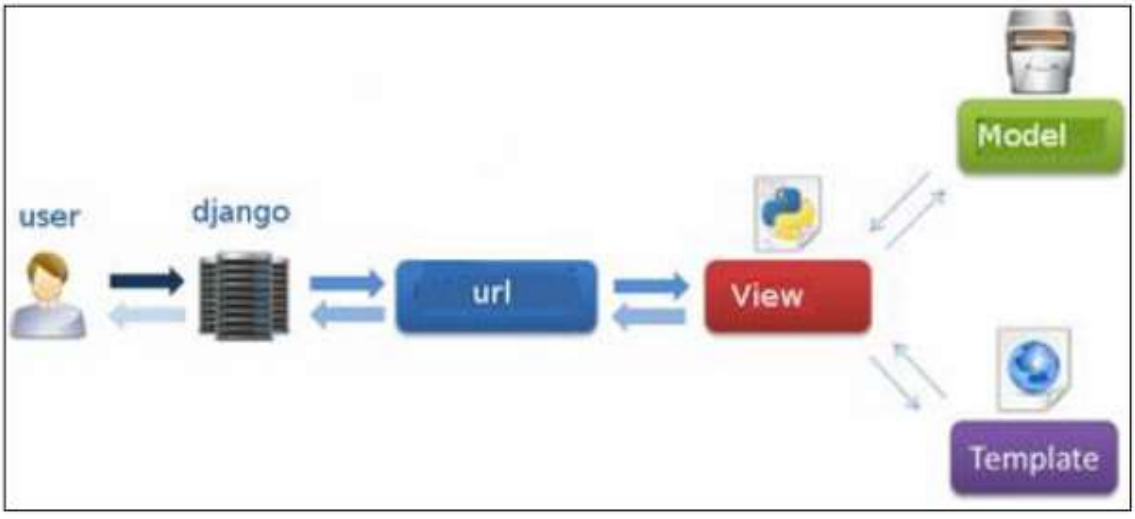
Django MVT

The MVT (Model View Template) is a software design pattern. It is a collection of three important components Model View and Template. The Model helps to handle database. It is a data access layer which handles the data.

The Template is a presentation layer which handles User Interface part completely. The View is used to execute the business logic and interact with a model to carry data and renders a template.

Although Django follows MVC pattern but maintains it?s own conventions. So, control is handled by the framework itself.

There is no separate controller and complete application is based on Model View and Template. That?s why it is called MVT application.



Django Model

In Django, a model is a class which is used to contain essential fields and methods. 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).

Django provides us a database-abstraction API which allows us to create, retrieve, update and delete a record from the mapped table.

Model is defined in Models.py file. This file can contain multiple models.

Let's see an example here, we are creating a model Employee which has two fields first_name and last_name.

```
from django.db import models

class Employee(models.Model):

   first_name = models.CharField(max_length=30)

   last_name = models.CharField(max_length=30)
```

The first_name and last_name fields are specified as class attributes and each attribute maps to a database column.

This model will create a table into the database that looks like below.

```
CREATE TABLE appname_employee (

   "id" INT NOT NULL PRIMARY KEY,

   "first_name" varchar(30) NOT NULL,

   "last_name" varchar(30) NOT NULL

);
```

The created table contains an auto-created id field. The name of the table is a combination of app name and model name that can be changed further.

Register / Use Model

After creating a model, register model into the INSTALLED_APPS inside settings.py.

For example,

```
INSTALLED_APPS = [
   #...
   'appname',
   #...
]
```

Django Model Fields Example

```
first_name = models.CharField(max_length=50) # for creating varchar column
release_date = models.DateField()            # for creating date column
num_stars = models.IntegerField()            # for creating integer column
```

Django Model Example

We created a model Student that contains the following code in models.py file.

```
//models.py
class Student(models.Model):
   first_name = models.CharField(max_length=20)
   last_name  = models.CharField(max_length=30)
   contact    = models.IntegerField()
   email      = models.EmailField(max_length=50)
   age        = models.IntegerField()
```

After that apply migration by using the following command.

python3 manage.py makemigrations myapp

Django Views

A view is a place where we put our business logic of the application. The view is a python function which is used to perform some business logic and return a response to the user. This response can be the HTML contents of a Web page, or a redirect, or a 404 error.

All the view function are created inside the views.py file of the Django app.

Django View Simple Example

```
//views.py

import datetime

# Create your views here.

from django.http import HttpResponse

def index(request):

   now = datetime.datetime.now()

   html = "<html><body><h3>Now time is %s.</h3></body></html>" % now

   return HttpResponse(html)    # rendering the template in HttpResponse
```

Let's step through the code.

First, we will import DateTime library that provides a method to get current date and time and HttpResponse class.

Next, we define a view function index that takes HTTP request and respond back.

View calls when gets mapped with URL in urls.py. For example

path('index/', views.index),

Returning Errors

Django provides various built-in error classes that are the subclass of HttpResponse and use to show error message as HTTP response. Some classes are listed below.

Class	Description
class HttpResponseNotModified	It is used to designate that a page hasn't been

	modified since the user's last request (status code 304).
class HttpResponseBadRequest	It acts just like HttpResponse but uses a 400 status code.
class HttpResponseNotFound	It acts just like HttpResponse but uses a 404 status code.
class HttpResponseNotAllowed	It acts just like HttpResponse but uses a 410 status code.
HttpResponseServerError	It acts just like HttpResponse but uses a 500 status code.

Django View Example

// views.py

```
from django.shortcuts import render

# Create your views here.

from django.http import HttpResponse, HttpResponseNotFound

def index(request):

   a = 1

   if a:

      return HttpResponseNotFound('<h1>Page not found</h1>')

   else:

      return HttpResponse('<h1>Page was found</h1>') # rendering the template in
HttpResponse
```

Django Templates

Django provides a convenient way to generate dynamic HTML pages by using its template

system.

A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

Why Django Template?

In HTML file, we can't write python code because the code is only interpreted by python interpreter not the browser. We know that HTML is a static markup language, while Python is a dynamic programming language.

Django template engine is used to separate the design from the python code and allows us to build dynamic web pages.

Django Template Configuration

To configure the template system, we have to provide some entries in settings.py file.

```
TEMPLATES = [
  {
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [os.path.join(BASE_DIR,'templates')],
    'APP_DIRS': True,
    'OPTIONS': {
      'context_processors': [
        'django.template.context_processors.debug',
        'django.template.context_processors.request',
        'django.contrib.auth.context_processors.auth',
        'django.contrib.messages.context_processors.messages',
      ],
    },
  },
```

]

Here, we mentioned that our template directory name is templates. By default, DjangoTemplates looks for a templates subdirectory in each of the INSTALLED_APPS.

Django Template Simple Example

First**, create a directory templates inside the project app** as we did below.

django templates

After that create a template index.html inside the created folder.

django templates 1

Our template index.html contains the following code.

// index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Index</title>
</head>
<body>
<h2>Welcome to Django!!!</h2>
</body>
```

```
</html>
```

Loading Template

To load the template, call get_template() method as we did below and pass template name.

```
//views.py
from django.shortcuts import render
#importing loading from django template
from django.template import loader
# Create your views here.
from django.http import HttpResponse
def index(request):
  template = loader.get_template('index.html') # getting our template
  return HttpResponse(template.render())      # rendering the template in HttpResponse
```

Set a URL to access the template from the browser.

```
//urls.py
path('index/', views.index),
```

Register app inside the INSTALLED_APPS

```
INSTALLED_APPS = [
   'django.contrib.admin',
   'django.contrib.auth',
```

```
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp'
]
```

Run Server

Execute the following command and access the template by entering localhost:8000/index at the browser.

```
$ python3 manage.py runserver
```

Django Template Language

Django template uses its own syntax to deal with variable, tags, expressions etc. A template is rendered with a context which is used to get value at a web page. See the examples.

Variables

Variables associated with a context can be accessed by {{}} (double curly braces). For example, a variable name value is rahul. Then the following statement will replace name with its value.

```
My name is {{name}}.
```

My name is rahul

Django Variable Example

//views.py

```
from django.shortcuts import render
#importing loading from django template
from django.template import loader
# Create your views here.
from django.http import HttpResponse
def index(request):
   template = loader.get_template('index.html') # getting our template
   name = {
      'student':'rahul'
   }
   return HttpResponse(template.render(name))      # rendering the template in HttpResponse
```

//index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Index</title>
</head>
<body>
<h2>Welcome to Django!!!</h2>
<h3>My Name is: {{ student }}</h3>
</body>
</html>
```

Tags

In a template, Tags provide arbitrary logic in the rendering process. For example, a tag can output content, serve as a control structure e.g. an "if" statement or a "for" loop, grab content from a database etc.

Tags are surrounded by {% %} braces. For example.

```
{% csrf_token %}


{% if user.is_authenticated %}
   Hello, {{ user.username }}.
{% endif %}
```

Django Static Files Handling

In a web application, apart from business logic and data handling, we also need to handle and manage static resources like CSS, JavaScript, images etc.

It is important to manage these resources so that it does not affect our application performance.

Django deals with it very efficiently and provides a convenient manner to use resources.

The django.contrib.staticfiles module helps to manage them.

Django Static (CSS, JavaScript, images) Configuration

1. Include the django.contrib.staticfiles in INSTALLED_APPS.

```
INSTALLED_APPS = [
   'django.contrib.admin',
   'django.contrib.auth',
```

```
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp'
]
```

2. Define STATIC_URL in settings.py file as given below.

```
STATIC_URL = '/static/'
```

3. Load static files in the templates by using the below expression.

```
{% load static %}
```

4. Store all images, JavaScript, CSS files in a static folder of the application. First create a directory static, store the files inside it

Django Image Loading Example

To load an image in a template file, use the code given below.

// index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Index</title>
     {% load static %}
```

```
</head>
<body>
<img src="{% static '/wallpaper.jpeg' %}" alt="My image" height="300px" width="700px"/>
</body>
</html>
```

//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),
]
```

//views.py

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

Run the server by using python manage.py runserver command.

Django Loading JavaScript

To load JavaScript file, just add the following line of code in index.html file.

```
{% load static %}
  <script src="{% static '/js/script.js' %}"
```

// index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Index</title>
    {% load static %}
   <script src="{% static '/js/script.js' %}" type="text/javascript"></script>
</head>
<body>
</body>
</html>
```

// script.js

```
alert("Hello, Welcome to 3RI Technologies");
```

Django Loading CSS Example

To, load CSS file, use the following code in index.html file.

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

After that create a directory CSS and file style.css which contains the following code.

// style.css

```
h1{
color:red;
}
```

// index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Index</title>
    {% load static %}
   <link href="{% static 'css/style.css' %}" rel="stylesheet">
</head>
<body>
<h1>Welcome to Javatpoint</h1>
</body>
</html>
```

Run the server by using python manage.py runserver command.

Django Model Form

Django Model Form

It is a class which is used to create an HTML form by using the Model. 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. For example, suppose we have a model containing various fields, we don't need to repeat the fields in the

form file.

For this reason, Django provides a helper class which allows us to create a Form class from a Django model.

Django ModelForm Example

First, create a model that contains fields name and other metadata. It can be used to create a table in database and dynamic HTML form.

// model.py

```
from __future__ import unicode_literals

from django.db import models

class Student(models.Model):

   first_name = models.CharField(max_length=20)

   last_name  = models.CharField(max_length=30)

   class Meta:

      db_table = "student"
```

This file contains a class that inherits ModelForm and mention the model name for which HTML form is created.

// form.py

```
from django import forms

from myapp.models import Student

class EmpForm(forms.ModelForm):

   class Meta:
```

```
    model = Student
    fields = "__all__"
```

Write a view function to load the ModelForm from forms.py.

//views.py

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

//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),
]
```

And finally, create a index.html file that contains the following code.

//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>
```

Run Server

Run the server by using python manage.py runserver command.

After that access the template by localhost:8000/index URL, and it will produce the following output to the browser.

Django Forms

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.

Building a Form in Django

Suppose we want to 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_length = 100)
```

Put this code into the forms.py file.

Instantiating Form in Django

Now, we need to instantiate the form in views.py file. See, the below code.

// views.py

```
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>
```

//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.

Django Database Connectivity

The settings.py file contains all the project settings along with database connection details. By default, Django works with SQLite, database and allows configuring for other databases as well.

Database connectivity requires all the connection details such as database name, user credentials, hostname drive name etc.

To connect with MySQL, django.db.backends.mysql driver is used to establishing a connection between application and database.

We need to provide all connection details in the settings file. The settings.py file of our project contains the following code for the database.

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'djangoApp',
      'USER':'root',
      'PASSWORD':'mysql',
      'HOST':'localhost',
      'PORT':'3306'
   }
}
```

After providing details, check the connection using the migrate command.

```
$ python3 manage.py migrate
```

Migrating Model

Well, till here, we have learned to connect Django application to the MySQL database. Next, we will see how to create a table using the model.

Each Django's model is mapped to a table in the database. So after creating a model, we need to migrate it. Let's see an example.

Suppose, we have a model class Employee in the models.py file that contains the following code.

// models.py

```
from django.db import models

class Employee(models.Model):
   eid      = models.CharField(max_length=20)
   ename    = models.CharField(max_length=100)
   econtact = models.CharField(max_length=15)
   class Meta:
      db_table = "employee"
```

Django first creates a migration file that contains the details of table structure. To create migration use the following command.

```
$ python3 manage.py makemigrations
```

The created migration file is located into migrations folder and contains the following code.

```
from django.db import migrations, models

class Migration(migrations.Migration):
   initial = True
   dependencies = [
```

```
    ]

    operations = [

        migrations.CreateModel(

            name='Employee',

            fields=[

                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),

                ('eid', models.CharField(max_length=20)),

                ('ename', models.CharField(max_length=100)),

                ('econtact', models.CharField(max_length=15)),

            ],

            options={

                'db_table': 'employee',

            },

        ),

    ]
```

Now, migrate to reflect the changes into the database.

```
$ python3 manage.py migrate
```

Django Session

A session is a mechanism to store information on the server side during the interaction with the web application.

Django Cookie

A cookie is a small piece of information which is stored in the client browser. It is used to

store user's data in a file permanently (or for the specified time).

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

The set_cookie() method is used to set a cookie and get() method is used to get the cookie.

Django with Bootstrap

Bootstrap is a framework which is used to create user interface in web applications. It provides css, js and other tools that help to create required interface.

In Django, we can use bootstrap to create more user friendly applications.

To implement bootstrap, we need to follow the following steps.

1. Download the Bootstrap

Visit the official site https://getbootstrap.com to download the bootstrap at local machine. It is a zip file, extract it and see it contains the two folder CSS and JS.

2. Create a Directory

Create a directory with the name static inside the created app and place the css and jss folders inside it. These folders contain numerous files.

3. Create a Template

First create a templates folder inside the app then create a index.htm file to implement (link) the bootstrap css and js files.

4. Load the Boostrap

load the bootstrap files resides into the static folder. Use the following code.

{% load staticfiles %}

And link the files by providing the file location (source). See the index.html file.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  {% load staticfiles %}
  <link href="{% static 'css/bootstrap.min.css' %}" >
  <script src="{% static 'bootstrap.min.js' %}"></script>
  <script>alert();</script>
</head>
<body>
</body>
</html>
```

In this template, we have link two files one is bootstrap.min.css and second is bootstrap.min.js. Lets see how to use them in application.

Suppose, if we don't use bootstrap, our html login for looks like this:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>login</title>
```

```
</head>
<body>
 <form action="/save" method="post">
  <div class="form-group">
   <label for="email">Email address:</label>
   <input type="email" class="form-control" id="email">
  </div>
  <div class="form-group">
   <label for="pwd">Password:</label>
   <input type="password" class="form-control" id="pwd">
  </div>
  <div class="checkbox">
   <label><input type="checkbox"> Remember me</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
</body>
</html>
```

After loading bootstrap files. Our code look like this:

```
// index.html

<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <title>login</title>
  {% load staticfiles %}
  <link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet">
  <script src="{% static 'js/bootstrap.min.js' %}"></script>
</head>
<body>
 <form action="/save" method="post">
  <div class="form-group">
   <label for="email">Email address:</label>
   <input type="email" class="form-control" id="email">
  </div>
  <div class="form-group">
   <label for="pwd">Password:</label>
   <input type="password" class="form-control" id="pwd">
  </div>
  <div class="checkbox">
   <label><input type="checkbox"> Remember me</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
</body>
</html>
```