UNIT-III

1. Introduction to URLs and Regex.

Ans:

URL-Patterns:

URL-patterns act as a bridge between browser requests and views in web applications.

- These patterns are defined using regular expressions (regex) and help route requests to the correct views.
- Incorrect patterns can result in "page not found" errors.

Regular Expressions (Regex):

Regex, short for regular expressions, are patterns used for text matching and manipulation.

- Learning regex is essential for creating accurate URL-patterns.
- Regex allows you to specify complex patterns for expected URLs.

Models.py File:

- The models.py file in a web project typically contains import statements and model classes.
- Model classes define fields, subclasses, and methods for data structures.
- Multiple model classes can exist in a single models.py file, each with its methods and subclasses.

2. Functions available in URLconfs

Ans: Django uses functions to define URL patterns in urls.py.

Available Functions:

- In Django, you use the url() function to define URL patterns.
  Syntax: `url(route, view, kwargs=None, name=None)`
  - route is the regex pattern to match expected URLs.
  - view specifies the view (class-based or function-based) to invoke.
  - kwargs is for passing additional data to the view.
  - name provides a name for the URL pattern, useful for URL reversing.
- In newer versions, the re_path() function is introduced, which is similar to url().
  Syntax: `re_path(route, view, kwargs=None, name=None)`
- Note that the url() function will be deprecated in future Django versions.
- include() function is used to include URL patterns defined elsewhere into another URL pattern.
  Syntax: `include(module, namespace=None)` or
  `include(pattern_list)` or
  `include((pattern_list, app_namespace), namespace=None)`
- All these URL pattern functions are available in the django.conf.urls module.

Ex:

```
# myapp/urls.py
from django.urls import path, re_path
from . import views
urlpatterns = [
    path('', views.home, name='home'),  # Using path()
    re_path(r'^article/(?P<article_id>\d+)/$', views.article_detail,
name='article_detail'),]

# myproject/urls.py
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),  # Using include() with module]
```

3. Regex

Ans: A regular expression or regex is a string containing symbols and characters which are the representation of a certain string pattern and it is used to find or search for a string with a pattern. Raw strings (prefixed with 'r') are used for regex patterns.

Regular Expression	Description
`.`	Matches any single character except a newline.
`^`	Matches the start of a string.
`$`	Matches the end of a string.
`*`	Matches 0 or more occurrences of the preceding character or g
`+`	Matches 1 or more occurrences of the preceding character or g
`?`	Matches 0 or 1 occurrence of the preceding character or group
`[]`	Defines a character class; matches any one character in the bra example, `[aeiou]` matches any vowel.
`[^]`	Defines a negated character class; matches any one character brackets. For example, `[^0-9]` matches any non-digit charac
`()`	Groups expressions together. For example, `(abc)+` matches occurrences of "abc".
`	`
`\`	Escapes a special character to match it literally. For example, ` single backslash.
`\d`	Matches any digit (0-9).
`\D`	Matches any non-digit character.

4. Writing a regex for different url-functions.

Ans:

url/re_path() Function: URL patterns in Django use regular expressions.

Ex:

For URL "http://www.myinstitute.com/student/14/course/77":

- pattern: **r'^student/(?P<student_id>\d+)/course/(?P<course_id>\d+)/$'**

path() Function: No complex regex is needed and variables are specified directly in the URL pattern.

Ex:

For URL "http://www.myinstitute.com/student/14/course/77":

- pattern: **'student/<int:student_id>/course/<int:course_id>/'**

The path() method is easier and more readable for URL patterns.

5. Introduction to Forms in Django.

Ans: HTML forms can be used for user input, but they require manual HTML coding and validation.

- Django forms simplify the process by automatically generating HTML form elements.
- Django forms handle data validation, ensuring user input is correct and safe.
- They process user input into Python data structures for easy manipulation.
- Django forms can be tied to models, making it efficient to create forms based on database fields.
- This simplifies the process of collecting and storing user data in the database.

6. **Building basic forms (a form where users can enter their name, email, and inquiries for a bookstore.)**

Ans:

Step 1: Create a Django Project and App:

```
django-admin startproject basic_project
cd basic_project
python manage.py startapp basic_app
```

Step 2: Create a templates directory and html files:

Inside your app's folder, you should create a folder named "templates" and inside this "templates" folder, you should create another folder with the same name as your app. Inside this app create two html files named ‘index.html’ and ‘form_page.html’.

Step 3: Open index.html and code as below:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home</title>
</head>
<body>
    <h1>Welcome to Homepage</h1>
    <h2>Go to /formpage to fill out the form</h2>
</body>
</html>
```

Step 4: Open form_page.html and code as below:

form_page.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Forms</title>
</head>
<body>
    <h1>Fill out the form</h1>
    <div class="container">
        {{form}}
    </div>
</body>
</html>
```

Step 5: Open the settings.py file of the project and update it:

Create a TEMPLATE_DIR for your templates in the settings.py file and update as follows:

settings.py

```
TEMPLATE_DIR = os.path.join(BASE_DIR,'basic_app/templates')
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR,],
        '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',
            ],
        },
    },
]
```

Step 6: Create a Python file in the app folder with the name forms.py and code as follows:

forms.py

```python
from django import forms

class FormName(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    text = forms.CharField(widget=forms.Textarea)
```

Step 7: Create views for the homepage in views.py file:

views.py

```python
from django.shortcuts import render
from . import forms

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

def form_name_view(request):
    my_form = forms.FormName()
    return render(request, 'basic_app/form_page.html', {'form' : my_form})
```

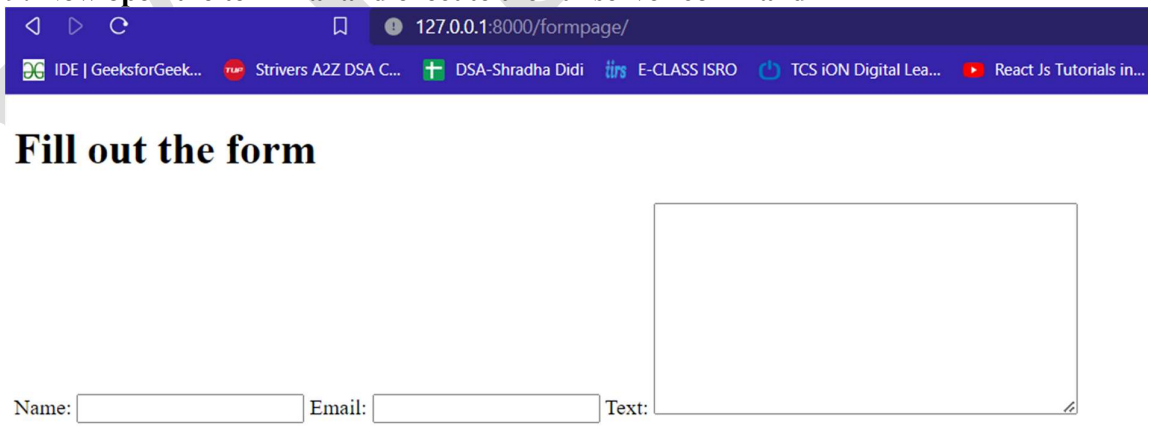
Step 8: Updating the urls.py file

urls.py

```python
from django.urls import path
from basic_app import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index, name = 'index'),
    path('index/', views.index, name = 'index'),
    path('formpage/', views.form_name_view, name = 'form_name'),
]
```

Step 9: Now open the terminal and execute the runserver command



7. Fetching data entered in the forms

Ans:

Step 10: Open the views.py file in the 'basic_app' folder and update the form_name_view as below

views.py

```python
from django.shortcuts import render
from . import forms

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

def form_name_view(request):
    if request.method == 'POST':
        my_form_with_data = forms.FormName(request.POST)
        if my_form_with_data.is_valid():
            # Picking data entered by user.
            print('Form Validated')
            print("Name: " + my_form_with_data.cleaned_data['name'])
            print("email: " + my_form_with_data.cleaned_data['email'])
            print("text: " + my_form_with_data.cleaned_data['text'])
    my_form = forms.FormName()
    return render(request, 'basic_app/form_page.html', {'form' : my_form})
```

Step 11: Open the index.html file and update as below

index.html

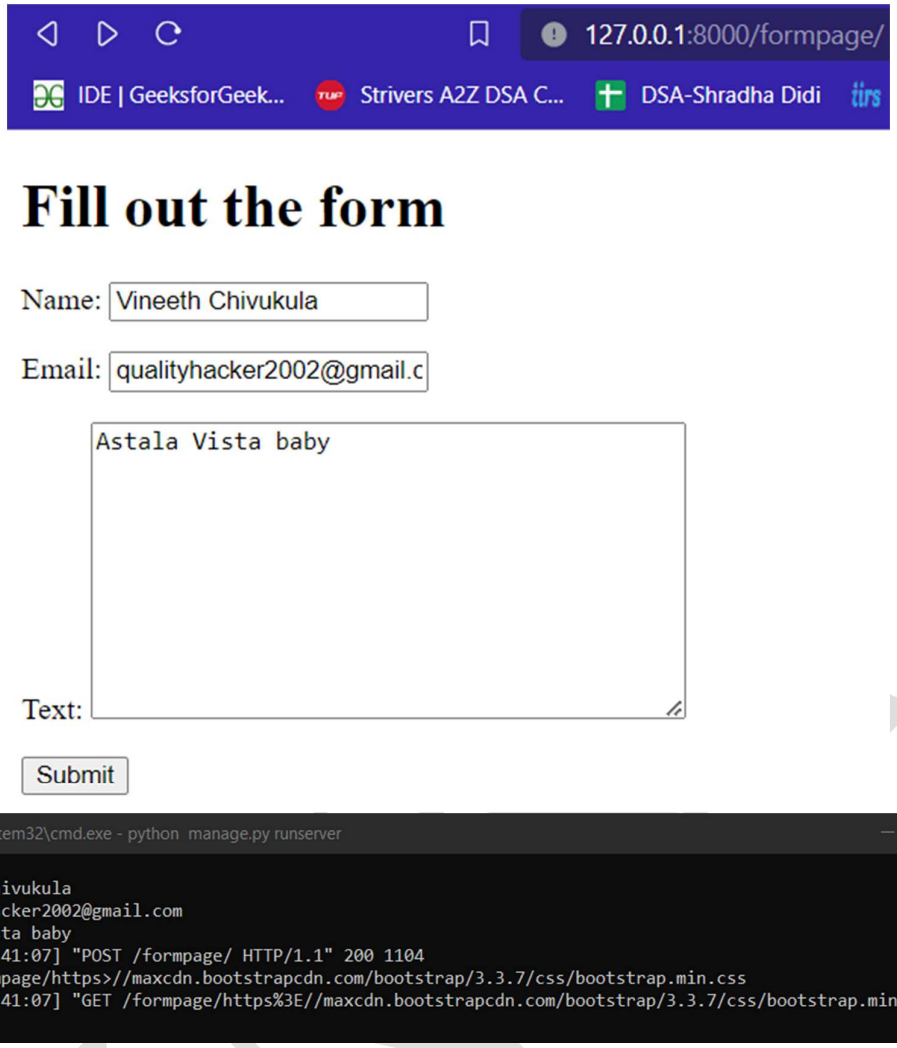
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home</title>
    <link rel="stylesheet"
    href="https>//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous">
</head>
<body>
    <div class="container">
        <div class="jumbotorn">
            <h1>Welcome to Homepage</h1>
            <h2>Go to /formpage to fill out the form</h2>
        </div>
    </div>
</body>
</html>
```

Step 11: Open the form_page.html file and update as below

form_page.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Forms</title>
    <link rel="stylesheet"
    href="https>//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous">
</head>
<body>
    <h1>Fill out the form</h1>
    <div class="container">
        <form method="post">
            {{form.as_p}}
            {% csrf_token %}
            <input type="submit" class="btn btn-primary" value="Submit">
        </form>
    </div>
</body>
</html>
```

Step 12: Now reload the page and enter the details.
Click the submit button and observe the terminal



8. Form fields and arguments

Ans: The following are the common form field types:

1. **CharField:** Used for short text fields like names and titles.
2. **EmailField:** Specifically designed for email addresses.
3. **IntegerField:** For storing integer values.
4. **BooleanField:** Represents a True/False or Yes/No choice.
5. **DateField:** For date input.
6. **DateTimeField:** For date and time input.
7. **ChoiceField:** Represents a dropdown list of choices.
8. **ModelChoiceField:** A choice field populated with options from a database model.
9. **ModelMultipleChoiceField:** Like ModelChoiceField but for multiple selections.
10. **FileField:** Allows users to upload files.

The following arguments allow you to customize and control how form fields behave and appear:

1. **required:** Determines if a field is mandatory (required=True by default).
2. **label:** Allows you to set a custom label for a form field.
3. **label_suffix:** Sets a label suffix, such as ':' or '='.
4. **initial:** Pre-fills a field with a default value that users can modify.
5. **widget:** Defines the input block type, like text boxes or radio buttons.
6. **help_text:** Provides user guidance and instructions for the field.
7. **error_messages:** Customizes error messages displayed for incorrect data.

8. **Validators:** A list of validation functions to check data validity.
9. **Localize:** Handles internationalization settings for the field.
10. **Disabled:** If set to True, the field is greyed out and cannot be edited.

Ex:

```python
from django import forms

class FormName(forms.Form):
    name = forms.CharField()
    email = forms.EmailField(disabled=True)
    text = forms.CharField(widget=forms.Textarea,max_length=100)
    rob_catcher = forms.CharField(required=False, widget=forms.HiddenInput)

    def clean_rob_catcher(self):
        rob_catcher = self.cleaned_data['rob_catcher']
        if len(rob_catcher)>0:
            raise forms.ValidationError('Robot Found')
        return rob_catcher
```

9. Form Validation

Ans:

Step 13: Update the form.py file as follows:

```python
from django import forms
from django.core import validators

def start_with_a(value):
    if value[0].lower() != 'a':
        raise forms.ValidationError('Name should start with "A"')

class FormName(forms.Form):
    name = forms.CharField(validators=[start_with_a])
    email = forms.EmailField()
    text = forms.CharField(widget=forms.Textarea,max_length=100)
    rob_catcher = forms.CharField(required=False,
                                  widget=forms.HiddenInput,
                                  validators=[validators.MaxLengthValidator(0)])
```

10. Model forms

Ans:

Step 14: Update models.py file as follows:

```python
from django.db import models

# Create your models here.
class Enquiry(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    text = models.TextField()
```

Step 15: Update the form.py file as follows:

```python
from django import forms
from basic_app.models import Enquiry

class EnquiryForm(forms.ModelForm):
    class Meta():
        model = Enquiry
        fields = '__all__'
```

Step 16: Update the views.py file as follows:

views.py

```python
from django.shortcuts import render
from basic_app.forms import EnquiryForm

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

def enquiry_form_view(request):
    form_empty = EnquiryForm()
    if request.method == 'POST':
        form_filled = EnquiryForm(request.POST)
        if form_filled.is_valid():
            form_filled.save(commit=True)
            print('Form Validated')
            print("Name: " + form_filled.cleaned_data['name'])
            print("email: " + form_filled.cleaned_data['email'])
            print("text: " + form_filled.cleaned_data['text'])
            return render(request, 'basic_app/form_page.html', {'form' : form_empty})
    else:
        print('Form Invalid')
    return render(request, 'basic_app/form_page.html', {'form' : form_empty})
```

Step 17: Update the urls.py file as follows:

urls.py

```python
from django.contrib import admin
from django.urls import path
from basic_app import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index, name = 'index'),
    path('index/', views.index, name = 'index'),
    path('formpage/', views.enquiry_form_view, name = 'form_name'),
]
```

Step 18: Go to settings.py file and update the INSTALLED_APPS list:

settings.py

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'basic_app',
]
```

Step 18: Execute the makemigrations, migrate and runserver commands:

You have now created a small application for inquiry forms and this application is currently taking inquiries and saving it on the database.