Full Stack Application Development with Cloud Computing

Module 3 - Back-End Development & Integration

Lab - 1

Unit 2 - Back-End Frameworks - Django

Topic: Django

Ex. 1: To create an employee management system using Django

Solution

- py --version (To check the python version)
- py -m venv samplevenv (To create a virtual environment)
- samplevenv\Scripts\activate.bat (To activate the virtual environment)

First, make sure you have Django installed. You can install it using pip:

pip install django

Next, create a new Django project and navigate to its directory:

```
django-admin startproject employee_management
cd employee_management
```

Create a new Django app called "employees":

```
python manage.py startapp employees
```

Now, let's define the models for our employee management system. Open the employees/models.py file and add the following code:

```
from django.db import models

class Employee(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    department = models.CharField(max_length=100)
```

```
    position = models.CharField(max_length=100)
    salary = models.DecimalField(max_digits=8, decimal_places=2)

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

Next, we need to apply the migrations to create the corresponding database tables. Run the following commands:

```
python manage.py makemigrations
python manage.py migrate
```

Now, let's create the views and templates for managing employees. Open the employees/views.py file and add the following code:

```
from django.shortcuts import render, redirect
from .models import Employee
from .forms import EmployeeForm

def employee_list(request):
    employees = Employee.objects.all()
    return render(request, 'employees/employee_list.html', {'employees': employees})

def employee_create(request):
    if request.method == 'POST':
        form = EmployeeForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('employee_list')
    else:
        form = EmployeeForm()
    return render(request, 'employees/employee_form.html', {'form': form})

def employee_update(request, pk):
    employee = Employee.objects.get(pk=pk)
    if request.method == 'POST':
        form = EmployeeForm(request.POST, instance=employee)
        if form.is_valid():
            form.save()
            return redirect('employee_list')
    else:
        form = EmployeeForm(instance=employee)
    return render(request, 'employees/employee_form.html', {'form': form, 'employee': employee})

def employee_delete(request, pk):
```

```
    employee = Employee.objects.get(pk=pk)
    employee.delete()
    return redirect('employee_list')
```

In the employees/views.py file, we have defined several views for listing, creating, updating, and deleting employees.

Now, let's create the forms for employee creation and update. Create a new file called employees/forms.py and add the following code:

```
from django import forms
from .models import Employee

class EmployeeForm(forms.ModelForm):
    class Meta:
        model = Employee
        fields = ('name', 'email', 'department', 'position', 'salary')
```

Next, we need to create the corresponding HTML templates. Create a new directory called templates in the project root directory, and within it, create a directory called employees. In the templates/employees directory, create the following HTML templates:

1. employee_list.html:

```
html
<!DOCTYPE html>
<html>
<head>
    <title>Employee List</title>
</head>
<body>
    <h1>Employee List</h1>
    <a href="{% url 'employee_create' %}">Add Employee</a>
    <ul>
        {% for employee in employees %}
        <li>
            {{ employee.name }} - {{ employee.email }}
            <a href="{% url 'employee_update' employee.id %}">Edit</a>
            <a href="{% url 'employee_delete' employee.id %}">Delete</a>
        </li>
        {% empty %}
        <li>No employees found.</li>
        {% endfor %}
    </ul>
</body>
</html>
```

2. employee_form.html:

```
html
<!DOCTYPE html>
<html>
<head>
    <title>{% if employee %}Update Employee{% else %}Create Employee{% endif
%}</title>
</head>
<body>
    <h1>{% if employee %}Update Employee{% else %}Create Employee{% endif
%}</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">{% if employee %}Update{% else %}Create{% endif
%}</button>
    </form>
</body>
</html>
```

Finally, let's define the URLs for our views. Open the employee_management/urls.py file and update it with the following code:

```
from django.contrib import admin
from django.urls import path
from employees import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('employees/', views.employee_list, name='employee_list'),
    path('employees/create/', views.employee_create, name='employee_create'),
    path('employees/update/<int:pk>/',views.employee_update,
name='employee_update'),
    path('employees/delete/<int:pk>/', views.employee_delete, name='employee_delete'),
]
```

That's it! You've now implemented a simple employee management system using Django. You can run the development server using the following command:

```
python manage.py runserver
```

You can access the employee management system by visiting http://localhost:8000/employees/ in your web browser.

Please note that this is a basic implementation, and you may need to add more features such as authentication, validation, and error handling based on your requirements.

Practice Question

Ex. 2: To create a library management system using Django

Objective: The objective of the project is to create a library management system to list the books, insert books, update book details, and delete books.