Sub :	FULL STACK DEVELOPMENT					Sub Code:	21CS62	Branch : ISE
Date:	8/7/2024	Duration	90 min’s	Max Marks	50	Sem/Sec	VI/ A, B, C	OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1 a)	What is the purpose of a database schema in Django? Explain the steps to define a model in Django and How would you retrieve all objects from a Django model?	5M	CO2	L2
1 b)	Explain with an example the use of a model string representation for a Django model.	5M	CO2	L2
2	Develop a Django app that performs student registration to a course. It should also display list of students registered for any selected course. Create students and course as models	10M	CO3	L3
3	Create a feedback form that saves data to a database model and include validation for feedback	10M	CO2	L3
4	Compare and contrast the use of regular forms and ModelForms using examples in Django.	10M	CO2	L2
5	How would you extend the Django admin interface to include additional functionality for a specific model?	10 M	CO2	L2
6 a)	Demonstrate how to link a form submission to a model instance in Django.	5 M	CO3	L2
6 b)	Given a complex URL routing scenario, explain how you would organize URLConfs in a Django project	5 M	CO2	L2

1 a) Purpose of a Database Schema in Django, Steps to Define a Model, and Retrieving All Objects from a Model

Marks: 5M, CO2, L2

Evaluation Scheme:

- Purpose of a database schema: 1 mark
- Steps to define a model: 2 marks
- Retrieving all objects from a model: 2 marks

Answer:

Purpose of a Database Schema in Django: A database schema in Django defines the structure of the database, including the tables, fields, and relationships between the data. It ensures data integrity, enforces constraints, and allows Django to interact with the database in a consistent manner.

Steps to Define a Model in Django:

1. **Create the Model:** Define a class that inherits from `django.db.models.Model`. Each attribute of the class represents a database field.

   ```
   python
   Copy code
   from django.db import models
   ```

```
class Student(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    email = models.EmailField()
```

2. **Add the Model to `INSTALLED_APPS`:** Ensure your app is listed in the `INSTALLED_APPS` setting in `settings.py`.
3. **Create Migrations:** Run `python manage.py makemigrations` to create migration files.
4. **Apply Migrations:** Run `python manage.py migrate` to apply the migrations and create the database schema.

Retrieving All Objects from a Django Model: Use the `.all()` method on the model's manager.

```
python
Copy code
students = Student.objects.all()
```

1 b) Example of Model String Representation

Marks: 5M, CO2, L2

Evaluation Scheme:

- Explanation of string representation: 2 marks
- Example: 3 marks

Answer:

Explanation of String Representation: The `__str__` method in a Django model provides a human-readable representation of the model instances. This is especially useful for displaying meaningful information in Django admin or during debugging.

Example:

```
python
Copy code
from django.db import models

class Course(models.Model):
    name = models.CharField(max_length=100)
    code = models.CharField(max_length=10)

    def __str__(self):
        return f"{self.code}: {self.name}"

# When an instance of Course is printed, it will display as "CS101:
Introduction to Computer Science" for example.
```

2. Developing a Django App for Student Registration to a Course

Marks: 10M, CO3, L3

Evaluation Scheme:

- Model definitions: 3 marks
- Views for registration: 3 marks
- Template for displaying students: 2 marks
- URL configurations: 2 marks

Answer:

Model Definitions:

```
python
Copy code
from django.db import models

class Course(models.Model):
    name = models.CharField(max_length=100)
    code = models.CharField(max_length=10)

    def __str__(self):
        return f"{self.code}: {self.name}"

class Student(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    email = models.EmailField()
    course     =     models.ForeignKey(Course,     on_delete=models.CASCADE,
related_name='students')

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

Views for Registration:

```
python
Copy code
from django.shortcuts import render, redirect
from .models import Student, Course
from .forms import StudentForm

def register_student(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('student_list')
    else:
        form = StudentForm()
    return render(request, 'register_student.html', {'form': form})
```

```
def student_list(request):
    courses = Course.objects.all()
    return render(request, 'student_list.html', {'courses': courses})
```

Template for Displaying Students: `register_student.html`

```
html
Copy code
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Register</button>
</form>
```

`student_list.html`

```
html
Copy code
{% for course in courses %}
    <h2>{{ course.name }}</h2>
    <ul>
        {% for student in course.students.all %}
            <li>{{ student.first_name }} {{ student.last_name }}</li>
        {% endfor %}
    </ul>
{% endfor %}
```

URL Configurations:

```
python
Copy code
from django.urls import path
from . import views

urlpatterns = [
    path('register/', views.register_student, name='register_student'),
    path('students/', views.student_list, name='student_list'),
]
```

3. Creating a Feedback Form with Validation

Marks: 10M, CO2, L3

Evaluation Scheme:

- Model definition: 3 marks
- Form definition: 3 marks
- View for form handling: 2 marks
- Template with validation: 2 marks

Answer:

Model Definition:

```
python
Copy code
from django.db import models

class Feedback(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    comments = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

Form Definition:

```
python
Copy code
from django import forms
from .models import Feedback

class FeedbackForm(forms.ModelForm):
    class Meta:
        model = Feedback
        fields = ['name', 'email', 'comments']

    def clean_comments(self):
        comments = self.cleaned_data.get('comments')
        if len(comments) < 10:
            raise  forms.ValidationError("Comments  must  be  at  least  10
characters long.")
        return comments
```

View for Form Handling:

```
python
Copy code
from django.shortcuts import render, redirect
from .forms import FeedbackForm

def feedback_view(request):
    if request.method == 'POST':
        form = FeedbackForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('thank_you')
    else:
        form = FeedbackForm()
    return render(request, 'feedback.html', {'form': form})
```

Template with Validation: `feedback.html`

```
html
Copy code
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
</form>
{% if form.errors %}
    <ul>
        {% for field in form %}
            {% for error in field.errors %}
                <li>{{ error }}</li>
            {% endfor %}
        {% endfor %}
    </ul>
{% endif %}
```

4. Comparing Regular Forms and ModelForms in Django

Marks: 10M, CO2, L2

Evaluation Scheme:

- Explanation of Regular Forms: 2 marks
- Explanation of ModelForms: 2 marks
- Comparison: 4 marks
- Examples: 2 marks

Answer:

Regular Forms: Regular forms in Django are created by subclassing `forms.Form`. They provide a way to manually define fields and handle form validation.

ModelForms: ModelForms are created by subclassing `forms.ModelForm`. They automatically generate form fields based on the fields of a Django model, and they simplify the process of creating and updating model instances.

Comparison:

- **Ease of Use:** ModelForms are easier to use when the form closely matches the model, as they require less boilerplate code.
- **Flexibility:** Regular forms provide more flexibility, allowing for custom fields and validation logic that may not map directly to a model.
- **Integration:** ModelForms integrate seamlessly with Django's ORM, automatically handling instance creation and updating.

Examples:

Regular Form:

```
python
Copy code
from django import forms

class ContactForm(forms.Form):
    name = forms.CharField(max_length=100)
    email = forms.EmailField()
    message = forms.CharField(widget=forms.Textarea)
```

ModelForm:

```
python
Copy code
from django import forms
from .models import Feedback

class FeedbackForm(forms.ModelForm):
    class Meta:
        model = Feedback
        fields = ['name', 'email', 'comments']
```

5. Extending the Django Admin Interface

Marks: 10M, CO2, L2

Evaluation Scheme:

- Basic admin configuration: 3 marks
- Adding additional functionality: 5 marks
- Example: 2 marks

Answer:

Basic Admin Configuration: To extend the Django admin interface, you need to create a custom admin class for the model and register it with the admin site.

```
python
Copy code
from django.contrib import admin
from .models import Student

class StudentAdmin(admin.ModelAdmin):
    list_display = ('first_name', 'last_name', 'email')
    search_fields = ('first_name', 'last_name', 'email')

admin.site.register(Student, StudentAdmin)
```

Adding Additional Functionality: You can add various functionalities, such as custom actions, filters, and fieldsets.

```
python
Copy code
class StudentAdmin(admin.ModelAdmin):
    list_display = ('first_name', 'last_name', 'email')
    search_fields = ('first_name', 'last_name', 'email')
    list_filter = ('course',)
    fieldsets = (
        (None, {'fields': ('first_name', 'last_name')}),
        ('Contact Information', {'fields': ('email',)}),
        ('Course Information', {'fields': ('course',)}),
    )
    actions = ['make_inactive']

    def make_inactive(self, request,
```