Django CRUD on record model with modelform setp by step guide and code explanation of each line of code.

Here's a complete step-by-step guide to building a CRUD app in Django using:

- a `Record` model
- `ModelForm`
- function-based views
- templates
- SQLite (default DB)

You'll learn:

1. Create Model
2. Create ModelForm
3. Create CRUD Views
4. Configure URLs
5. Build Templates
6. Run and Test
7. Explanation of every important line

---

1. Create Django Project

Install Django:

```
pip install django
```

Create project:

```
django-admin startproject myproject
```

Go inside project:

```
cd myproject
```

Create app:

```
python manage.py startapp records
```

---

2. Register App

Open:

`myproject/settings.py`

Find:

```
INSTALLED_APPS = [
```

Add:

```
'records',
```

Result:

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

Explanation

- `INSTALLED_APPS` tells Django which apps are active.
- `'records'` enables our CRUD app.

Project Structure

```
myproject/
│
├── manage.py
│
├── myproject/
│
├── records/
│   ├── migrations/
│   ├── templates/
│   │   └── records/
│   │       ├── create.html
│   │       ├── list.html
│   │       ├── update.html
│   │       └── delete.html
│   │
│   ├── models.py
│   ├── forms.py
│   ├── views.py
│   └── urls.py
```

---

3. Create Record Model

Open:

`records/models.py`

Write:

```
from django.db import models


class Record(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    age = models.IntegerField()
    created_at = models.DateTimeField(auto_now_add=True)

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

Explanation of Every Line

Import

```
from django.db import models
```

Imports Django's model classes.

Create Model Class

```
class Record(models.Model):
```

- Creates a database table named `Record`.
- `models.Model` is Django's base model class.

Name Field

```
name = models.CharField(max_length=100)
```

- Stores text.
- Maximum 100 characters.

Email Field

```
email = models.EmailField()
```

- Stores email addresses.
- Includes email validation.

Age Field

```
age = models.IntegerField()
```

- Stores integers.

Created Date

```
created_at = models.DateTimeField(auto_now_add=True)
```

- Automatically saves creation timestamp.

String Representation

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

- Shows object name in admin panel and shell.

---

4. Create Database Table

Run migrations:

```
python manage.py makemigrations
```

Then:

```
python manage.py migrate
```

Explanation

- `makemigrations` creates migration files.
- `migrate` creates actual database tables.

---

5. Create ModelForm

Open:

`records/forms.py`

Create file if it doesn't exist.

Write:

```
from django import forms
from .models import Record


class RecordForm(forms.ModelForm):

    class Meta:
        model = Record
        fields = ['name', 'email', 'age']
```

Explanation

Import Forms

```
from django import forms
```

Imports Django form system.

Import Model

```
from .models import Record
```

Imports `Record` model from current app.

Create ModelForm

```
class RecordForm(forms.ModelForm):
```

- Automatically creates form fields from model fields.

Meta Class

```
class Meta:
```

Used to configure the form.

Link Model

```
model = Record
```

Tells form to use `Record` model.

Select Fields

```
fields = ['name', 'email', 'age']
```

Only these fields appear in the form.

---

6. Create CRUD Views

Open:

```
records/views.py
```

Write:

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Record
from .forms import RecordForm


# CREATE
def create_record(request):

    if request.method == 'POST':
        form = RecordForm(request.POST)

        if form.is_valid():
            form.save()
            return redirect('record_list')

    else:
        form = RecordForm()

    return render(request, 'records/create.html', {'form': form})


# READ
def record_list(request):

    records = Record.objects.all()

    return render(request, 'records/list.html', {'records': records})
```

```
# UPDATE
def update_record(request, pk):

    record = get_object_or_404(Record, pk=pk)

    if request.method == 'POST':
        form = RecordForm(request.POST, instance=record)

        if form.is_valid():
            form.save()
            return redirect('record_list')

    else:
        form = RecordForm(instance=record)

    return render(request, 'records/update.html', {'form': form})


# DELETE
def delete_record(request, pk):

    record = get_object_or_404(Record, pk=pk)

    if request.method == 'POST':
        record.delete()
        return redirect('record_list')

    return render(request, 'records/delete.html', {'record': record})
```

Explanation of CRUD Views

Imports

```
from django.shortcuts import render, redirect, get_object_or_404
```

render

Displays HTML template.

---

redirect

Redirects to another URL.

`get_object_or_404`

Gets object or returns 404 page if not found.

CREATE View

`def create_record(request):`

Handles adding new record.

Check POST Request

`if request.method == 'POST':`

Checks if form was submitted.

Bind Form Data

`form = RecordForm(request.POST)`

Loads submitted form data.

Validate Form

`if form.is_valid():`

Checks validation rules.

Save Data

`form.save()`

Inserts data into database.

Redirect

```
return redirect('record_list')
```

Redirects after successful creation.

READ View

```
records = Record.objects.all()
```

Fetches all records from database.

UPDATE View

```
record = get_object_or_404(Record, pk=pk)
```

Gets specific record using primary key.

Load Existing Data

```
form = RecordForm(instance=record)
```

Pre-fills form with current values.

Update Existing Object

```
form = RecordForm(request.POST, instance=record)
```

Updates same record instead of creating new one.

DELETE View

```
record.delete()
```

Deletes record from database.

---

7. Configure URLs

Create:

`records/urls.py`

Write:

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

urlpatterns = [
    path('', views.record_list, name='record_list'),
    path('create/', views.create_record, name='create_record'),
    path('update/<int:pk>/', views.update_record, name='update_record'),
    path('delete/<int:pk>/', views.delete_record, name='delete_record'),
]
```

Explanation

Root URL

```
path('', views.record_list, name='record_list')
```

Homepage shows all records.

Dynamic URL

```
<int:pk>
```

Captures record ID from URL.

Example:

```
/update/5/

pk = 5
```

Include App URLs

Open:

`myproject/urls.py`

Write:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('records.urls')),
]
```

---

8. Create Templates

Create folders:

```
records/
    templates/
        records/
```

---

create.html

```
<h1>Create Record</h1>

<form method="POST">
    {% csrf_token %}

    {{ form.as_p }}

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

<a href="{% url 'record_list' %}">Back</a>
```

Explanation

CSRF Token

```
{% csrf_token %}
```

Protects form from CSRF attacks.

Render Form

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

Displays form fields inside `<p>` tags.

---

list.html

```
<h1>All Records</h1>

<a href="{% url 'create_record' %}">Create New Record</a>

<hr>

{% for record in records %}

    <p>
        {{ record.name }}
        {{ record.email }}
        {{ record.age }}

        <a href="{% url 'update_record' record.id %}">Edit</a>

        <a href="{% url 'delete_record' record.id %}">
            Delete
        </a>
    </p>

{% endfor %}
```

Explanation

Loop Through Records

```
{% for record in records %}
```

Iterates through all records.

Access Fields

```
{{ record.name }}
```

Displays model field values.

---

update.html

```
<h1>Update Record</h1>

<form method="POST">
    {% csrf_token %}

    {{ form.as_p }}

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

<a href="{% url 'record_list' %}">Back</a>
```

---

delete.html

```
<h1>Delete Record</h1>

<p>Are you sure you want to delete "{{ record.name }}"?</p>

<form method="POST">
    {% csrf_token %}

    <button type="submit">Yes Delete</button>
</form>

<a href="{% url 'record_list' %}">Cancel</a>
```

---

9. Run Server

```
python manage.py runserver
```

Open browser:

```
http://127.0.0.1:8000/
```

CRUD Flow Summary

Operation	URL	Function
Create	`/create/`	Add record
Read	`/`	Show records
Update	`/update/1/`	Edit record
Delete	`/delete/1/`	Remove record

Final Project Structure

```
myproject/
│
├── manage.py
│
├── myproject/
│
├── records/
│   ├── migrations/
│   ├── templates/
│   │   └── records/
│   │       ├── create.html
│   │       ├── list.html
│   │       ├── update.html
│   │       └── delete.html
│   │
│   ├── models.py
│   ├── forms.py
│   ├── views.py
│   └── urls.py
```

What You Learned

You now know how to:

- Create Django models
- Use ModelForms
- Handle GET and POST requests
- Save data to database
- Update existing rows
- Delete records
- Use templates
- Configure URLs
- Render forms dynamically