# United University

Django

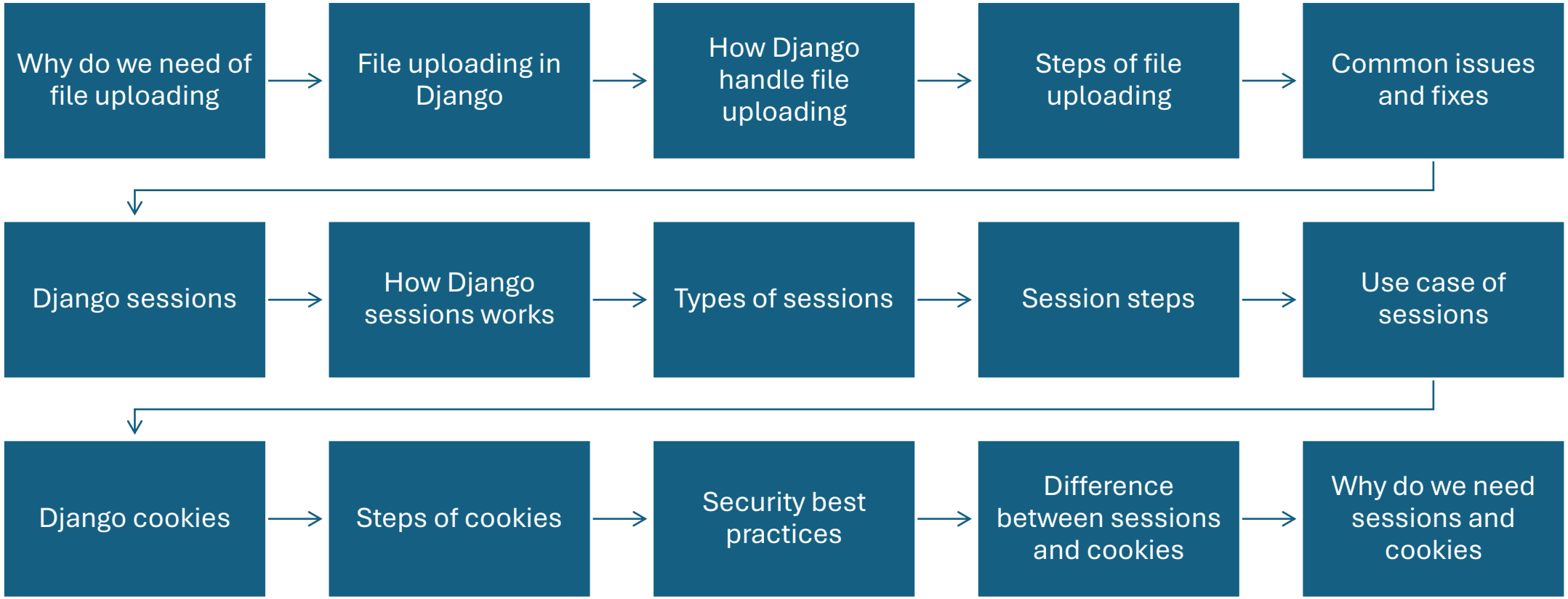
Topic : File uploading , cookies handling  and sessions

Presented by : Devansh Dwivedi(BTech 3rd NCS)

and Ricky Vishwakarma(BTech 3rd NCS)

Date : 11 April 2025

# Table of content



# Why do we need of file uploading

- File uploading in webpages is important because it allows user to share or send files to a server or another user through the web interface

- Common Reasons for File Uploading on Webpages
  - User submission
  - Media sharing
  - Document management
  - Profile customization
  - Online forms
  - Product or service support
  - Data import

# File uploading in Django

In Django, file uploading is handled through forms and models, and Django provides a built-in way to manage uploaded files

When Django handles a file upload, the file data ends up placed in request.FILES

Data can be upload in 2 form FileField  and ImageField

Date : 12 April 2025

# How Django handle file uploading

1 . User Submits file

Browser sends file via <form enctype = “multipart/form-data”>.

Without enctype, files won’t upload.

2 . Django Receives File

File lands in request.FILES(not request.POST)

3. File processing

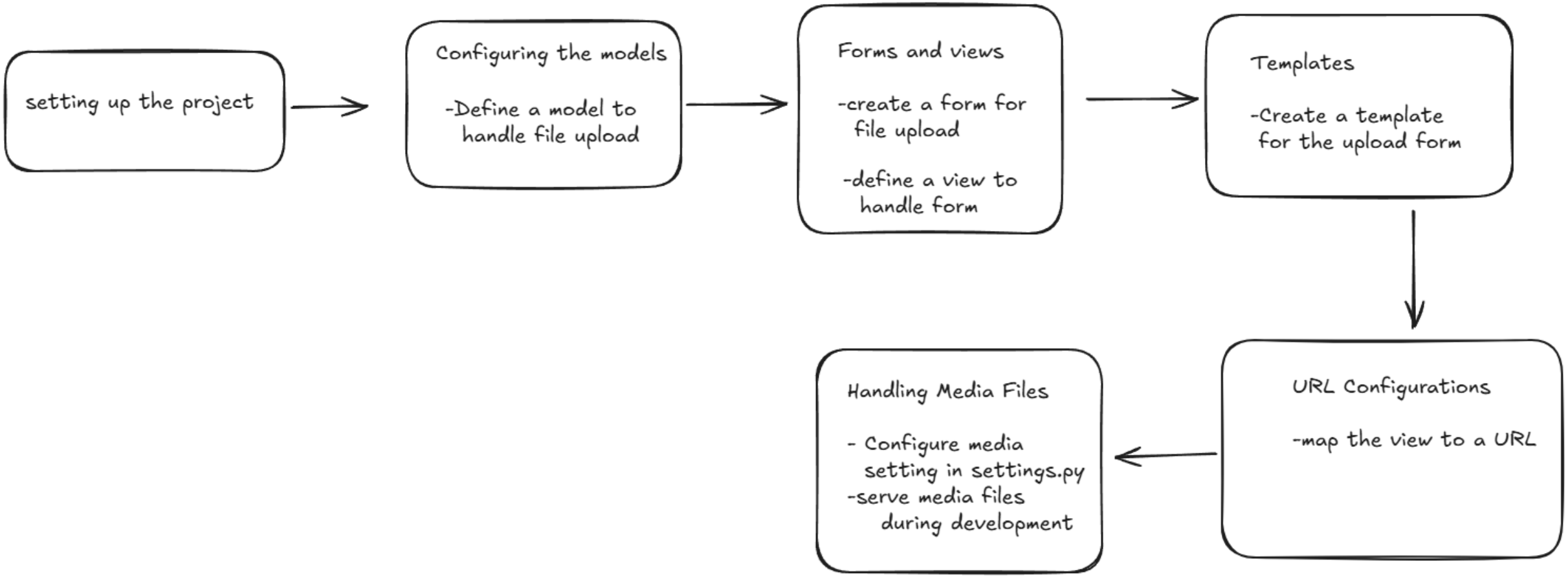
o1 : Save directly to disk

o2 : Use Django FileField

4. Storage

Development : Files saved to MEDIA_ROOT (configure in settings.py)

# Steps of file uploading



# Common issues and Fixes

| Issues | Solutions |
| --- | --- |
| FileNotFoundError | Ensure MEDIA_ROOT exists(mkdir media) |
| Missing enctype | Add enctype= “multipart/form-data” to <form> |
| File not saving | Check request.FILES and form.is_valid() |
| 403 forbidden | Verify csrf_token is include |
| | |

# Django Sessions

In Django, **sessions** are a way to store and manage data for individual users across requests — essentially, a mechanism to remember things about a user **between HTTP requests**.

**Why sessions?**

HTTP is **stateless**, meaning each request is independent. If a user logs in, Django needs a way to remember that they are logged in the next time they load a page. That's where sessions come in.

Uses a session ID cookie to track users .

# How Django sessions work

Session ID : When a user visit your site ,Django generates  a unique session ID and sends it to the client as a cookie (usually named as session ID)

Server-Side Storage : The actual session data (like the user's ID, cart items , etc) is  stored in the server. The client only holds the session ID.

Middleware : Django uses sessionMiddleware to handle the process of loading/storing sessions data.

# Types of seasons

1 .Database-backed sessions :

Require django.contrib.sessions in INSTALLED_APP

2. Cache-backed sessions :

faster than database sessions

Data may be lost if cache is cleared

3. Cache Database Sessions :

Best of both worlds: writes to cache and database

Reads form cache first , falls back to database

4. File-based Sessions:

slower than database or cache

Useful if you don’t want to use a database

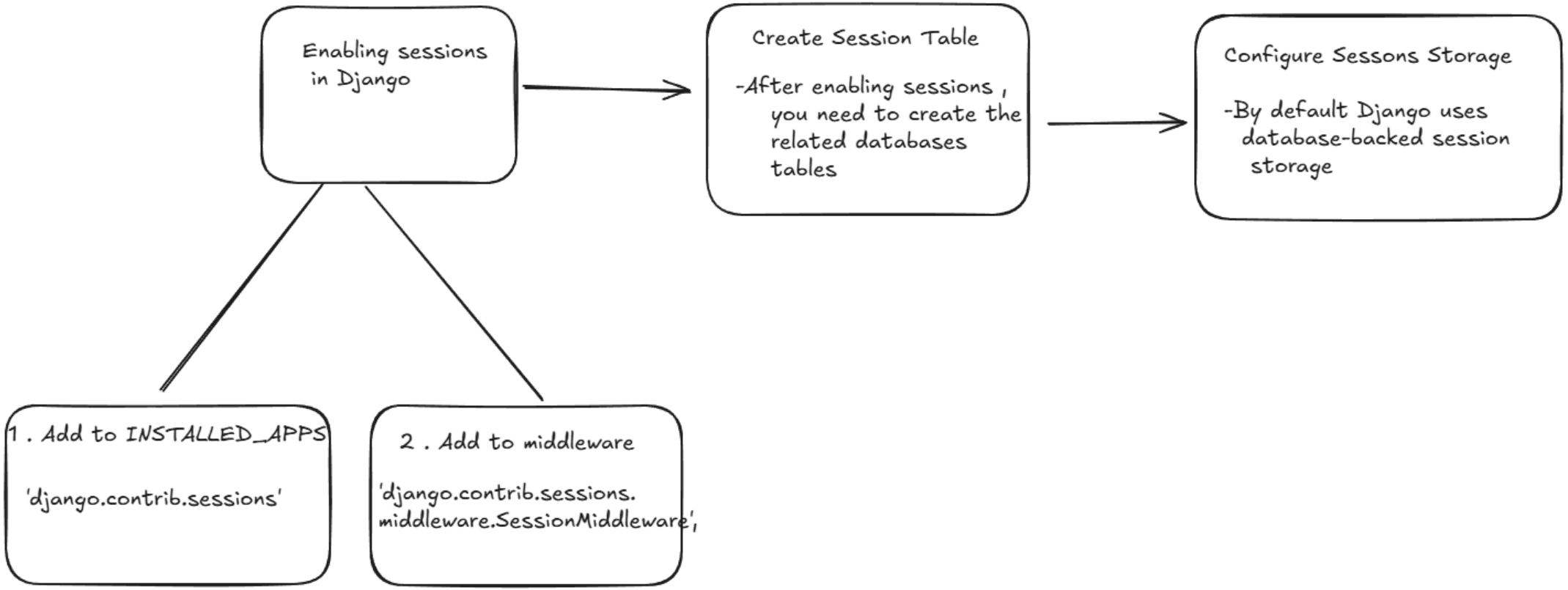
5. Signed Cookie Sessions:

Stores data in the cookie itself (signed to prevent)

Limited to 4kb of data

No server-side storage required

# Sessions Steps




python manage.py syncdb

python manage.py migrate

# Common use cases

- User authentications
- Shopping cart
- Form wizards
- User preferences
- Temporary messages

# Django Cookies 🍪

Cookies are small pieces of data that help store user-specific information on the client side. In Django, managing cookies is a simple task that can be done using a few straightforward commands.

Cookies are essential for maintaining user sessions and preferences. They are used to store data such as login information or user settings. Django provides easy methods to work with cookies, allowing to set and retrieve them as needed

To set a cookie in Django, we use the set_cookie method on an **HttpResponse** object. This method allows you to define the cookie's name, value, and optional parameters such as expiration time and domain.

# Steps for cookies

Setting Cookies in Django

To set a cookie, use the set_cookie() method on an HttpResponse object.

```
from django.http import HttpResponse

def set_cookie_view(request):
    response = HttpResponse("Cookie is set!")
    response.set_cookie('my_cookie', 'cookie_value')
    return response
```

Setting expiration and other options.

- -max_age: Lifetime of the cookies in second
- -secure= true : Ensure the cookies is sent only over HTTPS

Getting cookies in django

to retrieve a cookie, access the request.COOKIES dictionary

Complete example: Set and get Cookies

```
from django.http import HttpResponse

def set_and_get_cookie_view(request):
    # Setting the cookie
    response = HttpResponse("Cookie has been set and read!")
    response.set_cookie('my_cookie', 'cookie_value')

    # Getting the cookie value
    cookie_value = request.COOKIES.get('my_cookie', 'not set')
    response.content += f" Cookie value: {cookie_value}"

    return response
```

# Security Best Practices

1. Always use secure = True
2. Enable httponly = true
3. Set samesite = ‘Lax’ (CSRF protection)
4. Never store sensitive data (passwards , tokens)

# Difference between sessions and cookies

| feature | Cookies | Sessions |
|---|---|---|
| Stored in | Client browser | Server |
| Size  limit | ~4kb | Larger |
| Security | Less secure | More secure |
| Best for | Preferences, small info | Auth data, user sessions |

# Why do we need sessions and cookies

1 .Problem : HTTP forget everything

2 .Solution :

sessions : Remember users securely (server)

Cookies : Remember small things (browser)

3 .Combined : Cookies hold the ID -> Sessions hold data

Feel free to ask anything

# Thank you