🧭 Django Request Object — Complete Developer Reference

🧱 What Is a Request Object?

In Django, every incoming HTTP request is represented by an instance of the `HttpRequest` class.
It holds **everything the server knows about the client's request** — including URL info, headers, cookies, files, and metadata.

🔹 1. `req.META` — Low-Level Metadata

Definition:
A dictionary containing **low-level metadata** — raw data from the web server, including environment variables, HTTP headers, and connection details.

🧩 What is Low-Level Metadata?

Low-level metadata means **"data about the request at the network or server level"** — unprocessed values like IP addresses, protocol, ports, and raw headers.

Difference from Headers:
Headers are part of metadata, but metadata also includes deeper system-level info.

Example Keys in `req.META`:

Key	Example Value	Meaning
`REMOTE_ADDR`	`'192.168.0.12'`	Client IP address
`HTTP_USER_AGENT`	`'Mozilla/5.0 (Windows NT 10.0)'`	Browser info
`SERVER_PORT`	`'8000'`	Port Django is running on

`REQUEST_METHOD`	`'POST'`	HTTP method
`PATH_INFO`	`'/shop/2/'`	Requested path
`QUERY_STRING`	`'search=laptop&page=2'`	Raw query parameters

Example:

```
def log_request(req):
    print("IP:", req.META.get("REMOTE_ADDR"))
    print("User-Agent:", req.META.get("HTTP_USER_AGENT"))
    return HttpResponse("Logged request info")
```

Real-World Example:
Used in **logging, analytics, and security** to capture client IPs, request origins, or devices.

🔹 2. `req.headers`

Definition:
A **high-level**, dictionary-like object that contains all HTTP headers sent by the client (browser, Postman, etc.).
It is **case-insensitive** and **user-friendly**, built from the low-level `req.META`.

Difference from `req.META`:

`req.headers`	`req.META`
High-level, readable	Low-level, raw server data
Keys look like `"User-Agent"`	Keys look like `"HTTP_USER_AGENT"`
Only contains headers	Contains headers + other environment info

Example Use Case:
Restrict your app from being accessed by Postman or cURL.

```
def secure_view(req):
    user_agent = req.headers.get("User-Agent", "")
    if "Postman" in user_agent:
        return HttpResponseForbidden("Requests from Postman are not
allowed.")
    return HttpResponse("Welcome, browser user!")
```

Real-World Example:
Allow only browser-originated requests while blocking testing tools or bots.

🔹 3. `req.path`

Definition:
Returns the **path part of the URL**, excluding the domain name and query parameters.

Example:
`https://myshop.com/products/15?ref=home`
➡️ `req.path = '/products/15'`

Real-World Example:
Track which product page a user is visiting.

🔹 4. `req.path_info`

Definition:
Similar to `req.path`, but it gives the **decoded URL path** (used internally for routing).
If the path contains URL-encoded characters, this shows the readable version.

Example:
`/product/%E2%9C%93` → `req.path_info = '/product/✓'`

Real-World Example:
Useful when writing middleware that intercepts raw URLs before Django resolves them.

🔹 5. `req.user`

Definition:
Represents the **authenticated user** associated with the request.
If not logged in, it defaults to an `AnonymousUser`.

Example:

```
if req.user.is_authenticated:
    return HttpResponse(f"Welcome, {req.user.username}")
else:
    return HttpResponse("Please log in first.")
```

Real-World Example:
Used in dashboards, access control, or personalization.

🔹 6. `req.COOKIES` — 🍪 Cookies

Definition:
A dictionary containing all cookies sent by the client's browser.

What are Cookies?
Cookies are **small pieces of data stored in the browser** to help websites remember user information between requests.
They solve the **stateless** nature of HTTP.

Example Cookie Data:

Key	Value	Meaning
`sessionid`	`3adf76b90e9d12c`	Django session identifier
`csrftoken`	`xYz123AbC`	Protects against CSRF
`theme`	`dark`	User's theme preference

Example (Access):

```
def get_theme(req):
    theme = req.COOKIES.get('theme', 'light')
    return HttpResponse(f"Theme: {theme}")
```

Example (Set/Delete):

```
def set_cookie(req):
    res = HttpResponse("Cookie set!")
    res.set_cookie('theme', 'dark', max_age=3600)
    return res

def clear_cookie(req):
    res = HttpResponse("Cookie cleared!")
    res.delete_cookie('theme')
    return res
```

Real-World Uses:

- Login sessions (`sessionid`)
- Preferences (language, dark mode)
- Analytics or tracking
- Cart persistence

🔹 7. `req.FILES`

Definition:
Contains all files uploaded via POST requests (`<input type="file">`).

Example:

```
from django.core.files.storage import FileSystemStorage

def upload_file(req):
    if 'pic' in req.FILES:
        fs = FileSystemStorage()
        filename = fs.save(req.FILES['pic'].name, req.FILES['pic'])
        image_url = fs.url(filename)
        return JsonResponse({'image_url': image_url})
    return JsonResponse({'error': 'No file uploaded'})
```

Real-World Example:
Profile picture uploads, document submission, etc.

🔹 8. `req.session`

Definition:
A dictionary-like object used to store **per-user session data** (stored on the server).

How it works with Cookies:
Django stores a small cookie (`sessionid`) in the browser, which links to data stored on the server.

Example:

```
req.session['cart'] = ['laptop', 'mouse']
```

Real-World Example:
Maintaining a shopping cart or keeping login info.

🔹 9. `req.scheme`

Definition:
Returns `'http'` or `'https'`, indicating which protocol the client used.

Example:

```
if req.scheme != 'https':
    return HttpResponseForbidden("Use HTTPS only.")
```

Real-World Example:
Redirecting all traffic to HTTPS in production.

🔹 10. `req.encoding`

Definition:
Specifies the character encoding (like `'utf-8'`) used to decode form data or the request body.

Real-World Example:
Ensures that non-English text or emojis are correctly decoded from a POST body.

🔹 11. `req.content_type` — MIME Type

Definition:
Shows the **MIME type** (or **Content-Type**) of the request body.

🧩 What is MIME?

Full form: Multipurpose Internet Mail Extensions
It defines **what type of data** is being sent over HTTP.

Structure:

```
type/subtype
```

Examples:

MIME Type	Meaning
`text/html`	HTML page
`text/plain`	Plain text
`application/json`	JSON data
`image/jpeg`	JPEG image

`multipart/form-data`	File upload data

Example (in Django):

```
def check_type(req):
    print(req.content_type)
    if req.content_type == 'application/json':
        data = json.loads(req.body)
        return JsonResponse({'received': data})
```

Real-World Example:
APIs check MIME type to know if they should parse `req.body` as JSON, form data, or file data.

🧾 URL Parameter Example

When you define:

```
path('sample/<int:id>/', views.sample)
```

Your view receives both `req` and `id`:

```
def sample(req, id):
    return HttpResponse(f"Product ID: {id}")
```

Real-World Example:
Dynamic routes for products, users, etc.

🛒 Example with a Fake Product Dictionary

```
# products.py
prod_list = {
    1: {"name": "Laptop", "price": 60000},
    2: {"name": "Mobile", "price": 25000},
    3: {"name": "Headphones", "price": 2000},
}

# views.py
from . import products

def sample(req, id):
    product = products.prod_list.get(id, {"error": "Product not
found"})
    return JsonResponse(product)
```

🔍 How to View All Request Attributes

Use this to explore everything Django provides:

```
def debug_request(req):
    print(dir(req))
    return HttpResponse("Check console for all request attributes.")
```