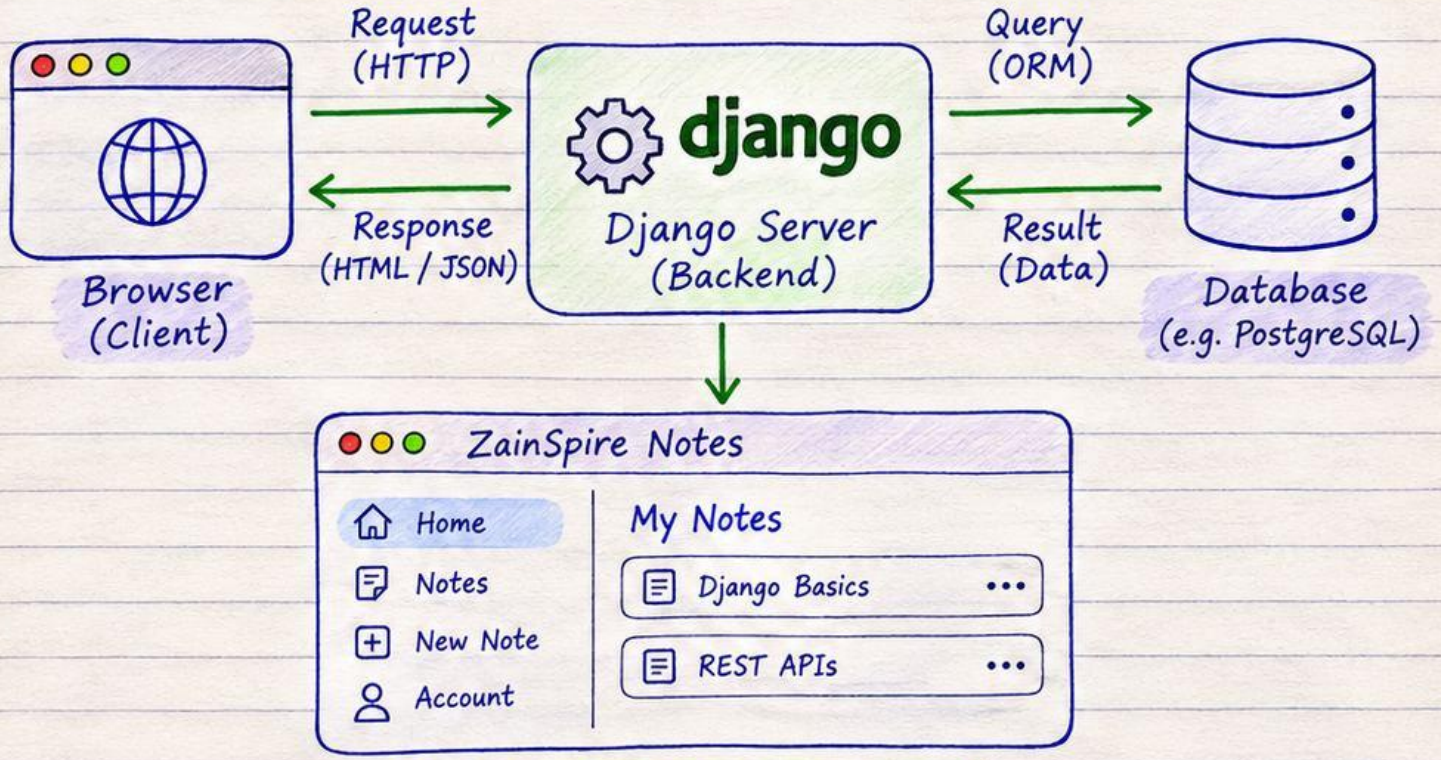
ZainSpire

# BACKEND DEVELOPMENT WITH DJANGO

Python • Django 5.2 • REST APIs

19 essential lessons. One connected project.




ZainSpire Notes

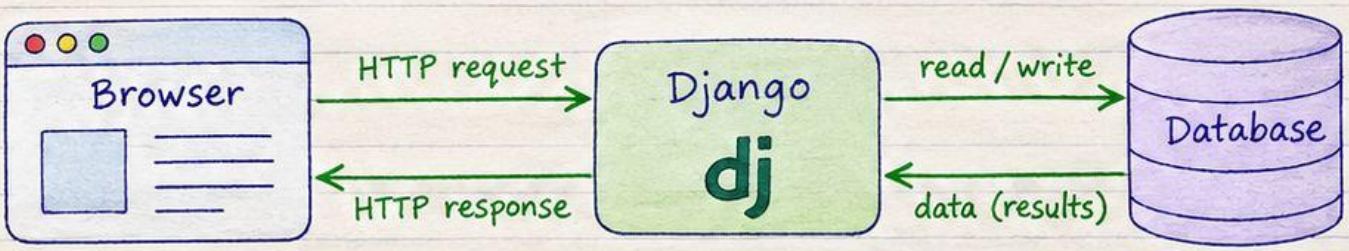
Models & ORM | Forms & Validation | Login & Permissions | REST APIs | Testing & Deployment

Understand it. Build it. Test it.

# WHAT THE BACKEND DOES

## WHAT RUNS WHERE

- **Frontend:** the interface people see and use.
- **Backend:** receives requests, applies rules and returns responses.
- **Database:** stores records between requests.



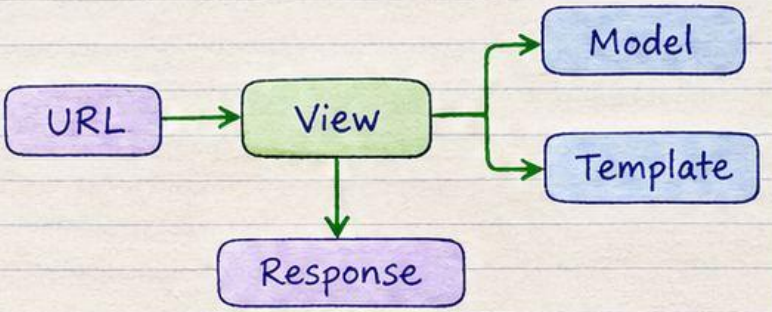

## A REAL EXAMPLE

Zain clicks "Save note".

1. Browser sends the title and body.
2. Django checks login, ownership and input.
3. Valid data is saved in the database.
4. Server returns a page redirect or JSON.

## DJANGO'S MAIN PARTS

- **Model** = data structure and database behavior.
- **View** = Python code that handles a request.
- **Template** = HTML rendered with data.
- **URLconf** = routes paths to views.



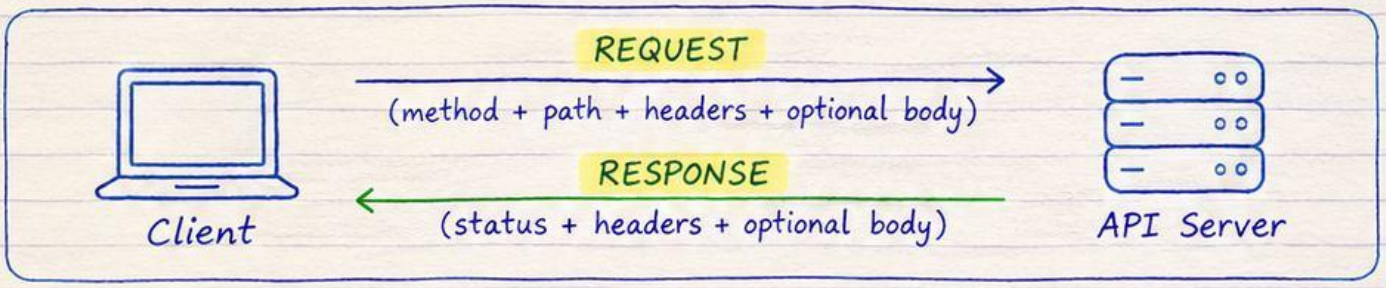

## OUR PROJECT

- ZainSpire Notes: each account owns private notes.
- Learn to create, list, edit and delete your own notes.
- Django 5.2 examples; API pages also use Django REST framework.

## REMEMBER

- A hidden button is not access control.
- Backend checks must run on every protected request.
- Python basics are assumed; these pages focus on backend work.

# HTTP, JSON & API DESIGN




**Example Request**

```
GET /api/notes/7/
```

**Example JSON Response**

```
{"id": 7, "title": "Learn Django"}
```

## METHODS WITH A PURPOSE

| Method & Path | Purpose |
|---|---|
| GET /api/notes/ | → list notes. |
| POST /api/notes/ | → create a note. |
| GET /api/notes/7/ | → read one. |
| PUT /api/notes/7/ | → update with a complete representation. |
| PATCH /api/notes/7/ | → update selected fields. |
| DELETE /api/notes/7/ | → delete one. |

## STATUS CODES

- `200 OK` : successful read or update.
- `201 Created` : new record.
- `204 No Content` : successful response with no body.
- `400 Bad Request` : invalid input.
- `401 Unauthorized` : authentication required or invalid.
- `403 Forbidden` : access refused; session auth can also use it for anonymous requests.
- `404 Not Found` : unavailable resource.
- `500` : unexpected server failure.

## CLEAR CONTRACTS

- Use resource nouns and consistent paths.
- JSON uses double quotes and `true` / `false` / `null`.
- `Content-Type` describes the request/response format.
- GET should not change application data.
- Return useful field errors, never secret debug details.

# CREATE YOUR DJANGO PROJECT

## ONE PROJECT, ONE ENVIRONMENT

Commands from a new project folder:

```
python -m venv .venv
```

Windows PowerShell:

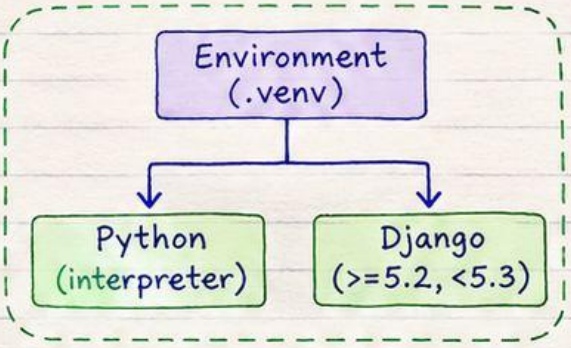
```
.\.venv\Scripts\Activate.ps1
```

macOS / Linux:

```
source .venv/bin/activate
```

Then:

```
python -m pip install "Django>=5.2,<5.3"
python -m django --version
```




⚠ Keep one project per environment.

## START THE PROJECT

```
django-admin startproject config .
python manage.py startapp notes
```

In config/settings.py, add "notes" to the existing INSTALLED_APPS list.

```
python manage.py migrate
python manage.py runserver
```

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

### Project Structure (after setup)

```
myproject/
├── .venv/
├── manage.py
├── config/
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── notes/
    ├── models.py
    ├── views.py
    └── templates/
```

## PROJECT VS APP

- Project = configuration for the whole site.
- App = a focused feature, such as notes or accounts.

## FILE MAP

| | |
|---|---|
| manage.py | → project commands |
| config/settings.py | → configuration |
| config/urls.py | → main routes |
| notes/models.py | → data |
| notes/views.py | → request handlers |
| notes/templates/ | → HTML |

## GOOD HABITS

- Use a supported Python version compatible with Django 5.2.
- Commit code and migrations; ignore .venv and secrets.
- runserver is for local development.
- Keep installed patch releases up to date within your chosen series.

# 05 / 20 — URLS, VIEWS & RESPONSES

## WRITE A SMALL VIEW — notes/views.py

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Salam, Zain!")
```

## WIRE THE APP — notes/urls.py

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

urlpatterns = [
    path("", views.home, name="home"),
]
```

## INCLUDE IT — config/urls.py

→ Add this route to the existing urlpatterns:

```
path("", include("notes.urls")),
```

→ Import include alongside path from django.urls.

## REQUEST FLOW

Browser GET / → config.urls → notes.urls → home(request) → 200 text response.




## USEFUL REQUEST VALUES

- `request.method` → "GET" or "POST"
- `request.GET` → query-string values
- `request.POST` → submitted form fields
- `request.user` → current user, with auth middleware

## JSON OPTION

```
from django.http import JsonResponse

return JsonResponse({"name": "Zain"})
```

→ This return belongs inside a view.

## IMPORTANT

- request.POST is not a JSON-body parser.
- Name routes and use reverse()/redirect() instead of hardcoding paths.
- A URL converter such as `<int:pk>` passes a Python integer to the view.
- A matching route does not prove that the user owns its object.

# 06 / 20 — DESIGN YOUR NOTE MODEL

notes/models.py

```python
from django.conf import settings
from django.db import models

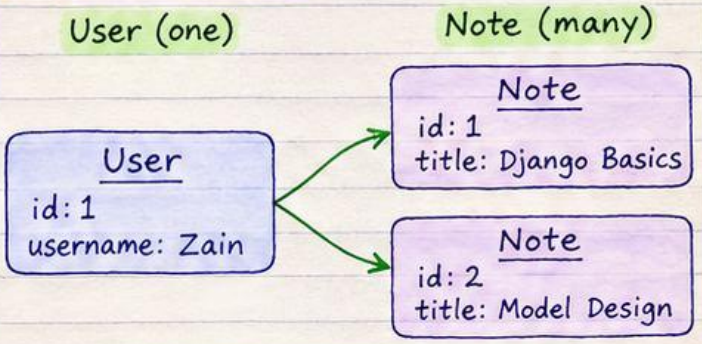
class Note(models.Model):
    owner = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name="notes",
    )
    title = models.CharField(max_length=120)
    body = models.TextField(blank=True)
    created_at = models.DateTimeField(
        auto_now_add=True,
    )

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

## WHAT THE FIELDS MEAN

- **id**: primary key added automatically.
- **owner**: required link to the account.
- **title**: short text, maximum 120 characters.
- **body**: may be empty during validation.
- **created_at**: timestamp set when first saved.

## RELATIONSHIP




⚠ **CASCADE** means deleting the user also deletes their notes.

✓ **blank** controls validation; **null** controls database NULL storage.

## CHOOSE DATA RULES EARLY

- Use settings.AUTH_USER_MODEL for user relationships.
- Use constraints for rules that the database must enforce.
- Pick on_delete behavior deliberately.
- Model save() does not automatically call full_clean().
- Validate incoming input through forms or serializers.

# MIGRATIONS & DJANGO ADMIN

## TURN MODELS INTO TABLES

```
python manage.py makemigrations notes
python manage.py migrate
python manage.py showmigrations
```

→ makemigrations records schema changes.

→ migrate applies unapplied migrations.

→ [X] means a migration is applied.

**Model**
**(Python class)**

```
class Note(models.Model):
    title = models.CharField(...)
    owner = models.ForeignKey(...)
```

→

**Migration file**
**(e.g. 0001_initial.py)**



→

**Database table**
**(e.g. notes_note)**

| id | title | owner_id |
|---|---|---|
| 1 | My Note | 1 |

Model change → migration file → database change.

## REGISTER YOUR MODEL — notes/admin.py

```
from django.contrib import admin
from .models import Note

@admin.register(Note)
class NoteAdmin(admin.ModelAdmin):
    list_display = ("title", "owner")
    search_fields = ("title",)
```




## CREATE AN ADMIN ACCOUNT

```
python manage.py createsuperuser
python manage.py runserver
```

→ Open /admin/ and sign in.

⚠ Use your own strong password; never store it in notes or source code.




## WHAT ADMIN IS FOR

- Trusted staff manage records.
- It is not your public customer interface.
- Staff access and model permissions matter.

## MIGRATION HABITS

- Commit migration files with the model changes.
- Review generated migrations before production.
- Back up data before risky schema changes.
- Do not delete migration history to fix a routine error.
- "No such table" often means migrations were not applied to the active database.

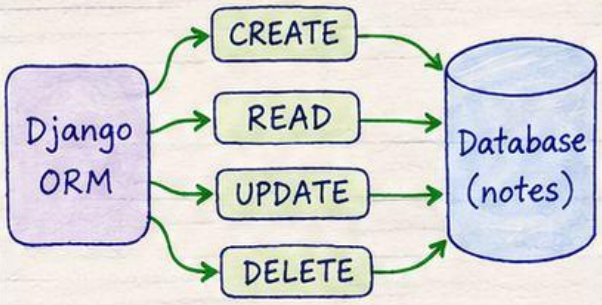
# ORM: CREATE, READ, UPDATE, DELETE

## START A SHELL

```
python manage.py shell
from notes.models import Note
from django.contrib.auth import get_user_model
User = get_user_model()
zain = User.objects.get(username="Zain")
```

→ Assumption: this account already exists.

## ORM CRUD (Note)




## CREATE

```
note = Note.objects.create(
    owner=zain, title="Learn Django"
)
```

→ Creates and saves a new Note row.

## IN A WEB VIEW

```
get_object_or_404(Note, pk=pk,
 owner=request.user)
```

→ This checks existence within the current user's records.

## READ

```
mine = Note.objects.filter(owner=zain)
mine = mine.order_by("-created_at", "-id")
note = mine.get(pk=note.pk)
```

→ filter() returns a QuerySet, possibly empty.

→ get() expects one object; it can raise DoesNotExist or MultipleObjectsReturned.

## QUERY HABITS

- QuerySets are usually lazy until evaluated.
- Chain filters before fetching large results.
- Use ORM parameters instead of string-built SQL.
- Keep ownership filters on reads, edits and deletes.
- Use transaction.atomic() when related writes must succeed together.

## UPDATE

```
note.title = "Practise Django"
note.save(update_fields=["title"])
```

→ Updates only the title field in the database.

## DELETE

```
note.delete()
```

⚠ Delete only after the user confirms the intended action.

# PAGES WITH TEMPLATES

## PRIVATE LIST VIEW — notes/views.py

```
from django.contrib.auth.decorators import login_required
from django.shortcuts import render
from .models import Note

@login_required
def note_list(request):
    notes = Note.objects.filter(
        owner=request.user
    ).order_by("-created_at", "-id")
    return render(
        request, "notes/list.html",
        {"notes": notes},
    )
```

## ROUTE — notes/urls.py

```
path("notes/", views.note_list, name="note-list"),
```

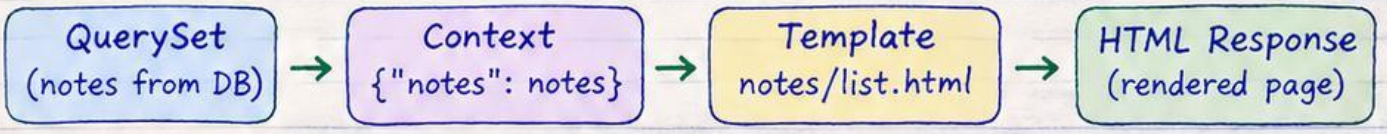
→ Add it to urlpatterns.

## TEMPLATE — notes/templates/notes/list.html

```
<h1>ZainSpire Notes</h1>
{% for note in notes %}
    <h2>{{ note.title }}</h2>
    <p>{{ note.body }}</p>
{% empty %}
    <p>No notes yet.</p>
{% endfor %}
```

## HOW IT FITS

→ QuerySet → context dictionary → template → HTML response.




- Template variables use {{ ... }}.
- Template logic uses {% ... %}.

## KEEP IT CLEAN

- Put database and business logic in Python.
- Use a base template and blocks for shared layout.
- Django escapes template variables by default.
- Do not mark untrusted user text as safe.
- For large lists, paginate; this first example is deliberately small.

# 10 / 20 — FORMS & SERVER-SIDE VALIDATION

notes/forms.py

```
from django import forms
from .models import Note

class NoteForm(forms.ModelForm):
    class Meta:
        model = Note
        fields = ["title", "body"]

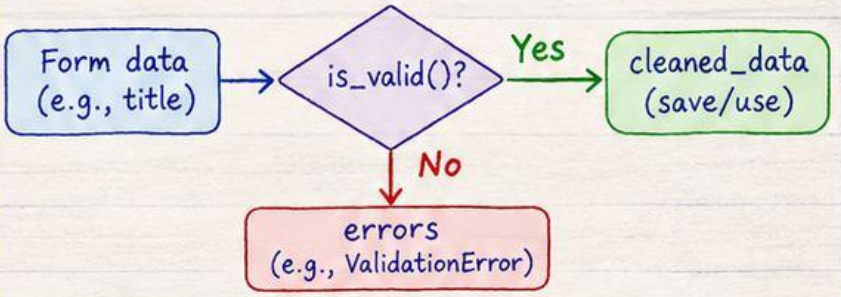
    def clean_title(self):
        title = self.cleaned_data["title"].strip()
        if len(title) < 3:
            raise forms.ValidationError(
                "Use at least 3 characters."
            )
        return title
```

## WHY THIS MATTERS

- Browser validation improves usability.
- The server must validate every submission.
- ModelForm reuses model field rules.
- Explicit fields prevent unwanted input fields.
- owner is assigned by the server, not the visitor.

## VALIDATION FLOW

Raw form data → is_valid() → cleaned_data or errors.




Valid title: "Learn Django"

Invalid title: "Hi" → Result: "Use at least 3 characters."

## RENDER ERRORS

- {{ form.as_p }} includes fields and field errors.
- Keep the bound form on failure so the user can correct it.

## CHOOSE THE RIGHT VALIDATOR

- clean_<field>() → one field.
- clean() → rules involving multiple fields.
- Database constraints → rules enforced for every write path.

**REMEMBER** Never save before checking is_valid(). Browser input can be missing, modified or forged.

# CREATE & EDIT NOTES SAFELY

## ONE VIEW FOR CREATE OR EDIT

- Use a single view to handle both creating a new note and editing an existing note.
- If pk is given → edit that note (after checking ownership).
- If pk is not given → create a new note.
- The same form (NoteForm) is used in both cases.

## FLOW (SAME VIEW)

GET (no save) → Show form (create or edit)

POST (with data) → Validate → Save (set owner) → Redirect to list

## notes/views.py — imports:

```
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib.auth.decorators import login_required
from .models import Note
from .forms import NoteForm
```

```
@login_required
def note_save(request, pk=None):
    note = None
    if pk is not None:
        note = get_object_or_404(
            Note, pk=pk, owner=request.user
        )

    form = NoteForm(
        request.POST if request.method == "POST" else None,
        instance=note,
    )
    if request.method == "POST" and form.is_valid():
        note = form.save(commit=False)
        note.owner = request.user
        note.save()
        return redirect("note-list")
    return render(request, "notes/form.html", {"form": form})
```

## ROUTES TO ADD

```
path("notes/new/", views.note_save, name="note-create"),
path("notes/<int:pk>/edit/", views.note_save, name="note-edit"),
```

## TEMPLATE — notes/templates/notes/form.html

```
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Save</button>
</form>
```

## WHY THESE LINES MATTER

- `instance=note` updates an existing row.
- `commit=False` lets the server set owner before saving.
- `POST → validate → save → redirect` avoids resubmitting on refresh.
- `GET` displays the form; it does not save data.

# 12 / 20 — LOGIN, LOGOUT & SESSIONS

## USE DJANGO'S BUILT-IN AUTH VIEWS

- In config/urls.py, add:

```
path("accounts/", include("django.contrib.auth.urls")),
```

- In config/settings.py:

```
LOGIN_URL = "/accounts/login/"
LOGIN_REDIRECT_URL = "/notes/"
LOGOUT_REDIRECT_URL = "/accounts/login/"
```

## LOGIN TEMPLATE

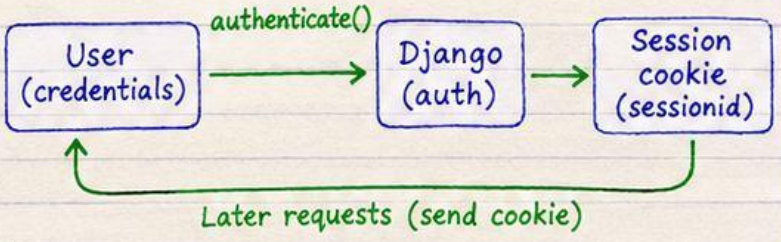
- Create notes/templates/registration/login.html:

```
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <input type="hidden" name="next" value="{{ next }}">
  <button type="submit">Log in</button>
</form>
```

## POST TO LOG OUT

```
<form method="post" action="{% url 'logout' %}">
  {% csrf_token %}
  <button type="submit">Log out</button>
</form>
```

## SESSION FLOW




→ Credentials → authenticate → session cookie.

→ Later requests send the cookie; Django resolves request.user.

→ Passwords are hashed, not stored as plain text.

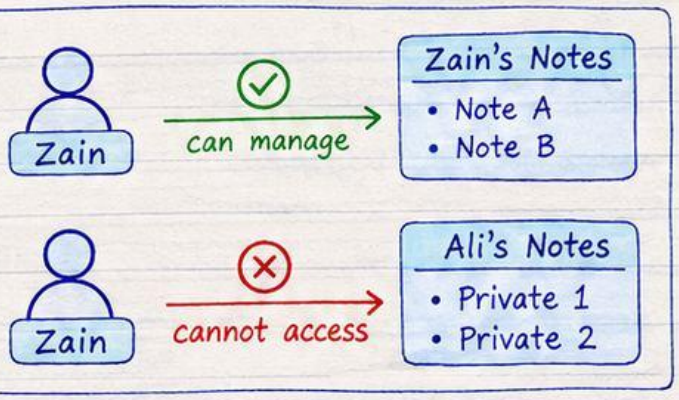
## ACCOUNTS & ACCESS

- Use create_user() or UserCreationForm; never assign raw passwords.
- Built-in auth URLs do not provide a signup page.
- @login_required redirects anonymous visitors to login.
- request.user.is_authenticated checks login state.
- Authentication = who you are.
- Authorization = what you may do.
- Add login rate limiting in a real deployment.

# 13 / 20 — OWNERSHIP & SAFE DELETE

## LOGIN IS ONLY THE FIRST CHECK

- Zain must not read or change Ali's private notes.
- Never trust an owner ID supplied by the browser.
- Scope database lookups to the signed-in user.




## POST-ONLY DELETE — notes/views.py

Uses the imports from pages 9 and 11.

```
from django.views.decorators.http import require_POST

@login_required
@require_POST
def note_delete(request, pk):
    note = get_object_or_404(
        Note, pk=pk, owner=request.user
    )
    note.delete()
    return redirect("note-list")
```

## ROUTE TO ADD

```
path("notes/<int:pk>/delete/", views.note_delete, name="note-delete"),
```

## DELETE BUTTON IN A NOTE LOOP

```
<form method="post" action="{% url 'note-delete' note.pk %}">
    {% csrf_token %}
    <p>Delete note: <strong>{{ note.title }}</strong>?</p>
    <p>Are you sure you want to delete this note?</p>
    <button type="submit">Delete note</button>
</form>
```

Delete note: **Project Plan**?

Are you sure you want to delete this note?

Delete note

## THREE DIFFERENT CHECKS

1. Logged in? Use authentication.
2. Allowed action? Use permissions.
3. Owns this object? Filter by owner.

## EXPECTED RESULTS

→ Owner + valid POST → delete and redirect.

→ Other user's note → 404 in this design.

→ GET on delete endpoint → 405 Method Not Allowed.

→ Missing CSRF token → rejected.

⚠ A hidden button cannot replace any server-side check.

# 14/20 — REST APIs & SERIALIZERS

## ADD DJANGO REST FRAMEWORK

- python -m pip install "djangorestframework>=3.16,<3.17"
- Add "rest_framework" to INSTALLED_APPS.

→ Django serves pages; DRF adds tools for structured web APIs.

## notes/serializers.py

```
from rest_framework import serializers
from .models import Note

class NoteSerializer(serializers.ModelSerializer):
    class Meta:
        model = Note
        fields = ["id", "title", "body", "created_at"]
        read_only_fields = ["id", "created_at"]

    def validate_title(self, value):
        value = value.strip()
        if len(value) < 3:
            raise serializers.ValidationError(
                "Use at least 3 characters."
            )
        return value
```

## TWO DIRECTIONS

Model object → serializer.data → response data

Request data → is_valid() → validated_data → save()

⚠ API validation and HTML form validation are separate paths.

## EXAMPLE REQUEST BODY

```
{"title": "Learn Django", "body": ""}
```

## INVALID INPUT

```
{"title": "Hi"}
```

→ Field error:

```
{"title": ["Use at least 3 characters."]}
```

## KEEP CONTROL

- List allowed fields explicitly.
- Do not accept owner from the client.
- Use partial updates deliberately for PATCH.
- Authentication and permissions belong on the API views too.

# API: LIST & CREATE YOUR NOTES

## notes/api.py

```
from rest_framework import generics, permissions
from rest_framework.authentication import SessionAuthentication
from .models import Note
from .serializers import NoteSerializer

class NoteListCreate(generics.ListCreateAPIView):
    serializer_class = NoteSerializer
    authentication_classes = [SessionAuthentication]
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        return Note.objects.filter(
            owner=self.request.user
        ).order_by("-created_at", "-id")

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)
```

## ROUTE — notes/urls.py

```
from . import api
```

Add to urlpatterns:

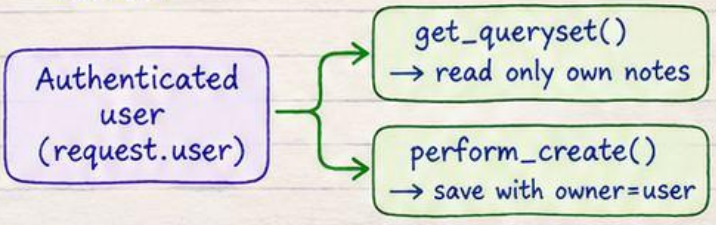
```
path("api/notes/", api.NoteListCreate.as_view(), name="api-notes"),
```

## WHAT THE GENERIC VIEW HANDLES

- GET → list the current user's notes.
- POST → validate data, save, return 201.
- Invalid data → 400 with field errors.

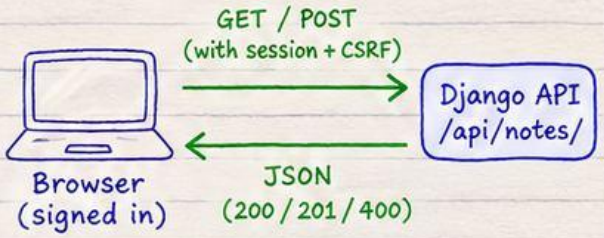
## OWNERSHIP IS ENFORCED TWICE

- get_queryset() limits what can be read.
- perform_create() sets the owner on the server.




## SESSION AUTH IN THIS EXAMPLE

- Sign in through the Django login page.
- Same-site browser requests use the session cookie.
- Unsafe methods need a valid CSRF token.
- IsAuthenticated does not automatically filter other users' records.
- Add pagination on page 17 before large datasets.



# 16 / 20 — API: RETRIEVE, UPDATE & DELETE

## notes/api.py — append this class

Reuse the imports from page 15.

```
class NoteDetail(generics.RetrieveUpdateDestroyAPIView):
    serializer_class = NoteSerializer
    authentication_classes = [SessionAuthentication]
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        return Note.objects.filter(
            owner=self.request.user
        )
```

## ROUTE — notes/urls.py

```
path(
    "api/notes/<int:pk>/",
    api.NoteDetail.as_view(),
    name="api-note-detail",
),
```

## ONE ENDPOINT, FOUR ACTIONS

| Method & URL | Action | Result |
|---|---|---|
| GET /api/notes/7/ | retrieve | → 200 |
| PUT /api/notes/7/ | complete update | → 200 |
| PATCH /api/notes/7/ | selected fields | → 200 |
| DELETE /api/notes/7/ | remove | → 204, empty body |

## PATCH EXAMPLE

```
{
    "title": "Practise Django"
}
```

→ The omitted body stays unchanged.




## PRIVATE BY DESIGN

- Lookup happens inside the user's filtered QuerySet.
- Unknown ID or another user's note → 404.
- Client cannot transfer ownership by adding an owner field.

## AUTH CHOICES

- This project uses sessions for a same-site browser.
- Mobile or separate clients may need a token-based scheme.
- Tokens do not replace permission checks.
- DRF core TokenAuthentication is not JWT.
- Use HTTPS and a deliberate credential-storage strategy.

# 17 / 20 — PAGINATION & QUERY PERFORMANCE

## DON'T RETURN EVERY ROW

Pagination limits response size and database work.




## CONFIGURATION

config/settings.py:

```
REST_FRAMEWORK = {
    "DEFAULT_PAGINATION_CLASS":
        "rest_framework.pagination.PageNumberPagination",
    "PAGE_SIZE": 10,
}
```

→ Keep stable ordering, such as "-created_at", "-id".

## REQUEST

```
GET /api/notes/?page=2
```

## PAGINATED RESPONSE SHAPE

```
{
    "count": 23,
    "next": "...?page=3",
    "previous": "...?page=1",
    "results": [...]
}
```

Illustrative shape; URLs and records depend on the request.

## FILTER FIRST



→ Start with owner=request.user, then apply allowed filters.

→ Use validated query parameters, not arbitrary field names.

## MEASURE BEFORE TUNING



- Inspect query counts and slow queries.
- Index fields used by important filters when justified.
- Avoid extra queries inside loops.
- Cache only with correct user separation and invalidation.
- Keep pagination and ownership checks together.

## AVOID N+1 QUERIES

If you display note.owner.username for many notes:

```
Note.objects.select_related("owner")
```

→ Use select_related for foreign-key / one-to-one links.

→ Use prefetch_related for many-to-many / reverse collections.

**N+1 (inefficient)**

1 query: get notes
+ N queries: get owner
~ N+1 queries ✗

→

**select_related (efficient)**

1 query (JOIN):
get notes + owner
✓ 1 query

# 18 / 20 — SECURITY THAT EVERY BACKEND NEEDS

## CSRF: PROTECT COOKIE-BASED ACTIONS

- Keep Django's CSRF middleware enabled.
- Use {% csrf_token %} in POST forms.
- For JavaScript requests, send the valid CSRF token.
- Do not disable CSRF to make an error disappear.

Template (form):

```
<form method="post">
  {% csrf_token %}
  <input type="text" name="title">
  <button type="submit">Save</button>
</form>
```

JavaScript requests:




Send CSRF token in X-CSRFToken header.

## XSS: UNTRUSTED CONTENT

- Templates escape variables by default.
- Do not apply safe to user-written HTML.
- Rich text needs deliberate sanitization.

```
{{ comment }}          → safe (escaped)
{{ comment|safe }}     → risky (avoid for user input)
```

## SQL INJECTION

- Use ORM queries or parameterized raw SQL.
- Never concatenate user input into SQL.

```
✓ User.objects.filter(username=username)

✗ User.objects.raw(
    "SELECT * FROM auth_user WHERE username='"
     + username + "'"
  )
```

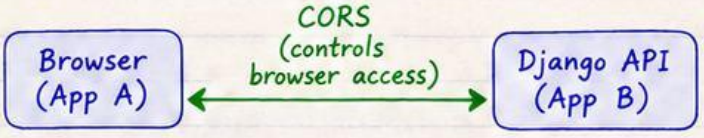
## SECRETS & PASSWORDS

- Keep SECRET_KEY, database passwords and tokens outside Git.
- Use Django's password tools.
- Never log passwords, session cookies or access tokens.

```
from django.contrib.auth.hashers import (
    make_password, check_password
)
hashed = make_password(password)
ok = check_password(password, hashed)
```

## CORS IS NOT AUTHENTICATION

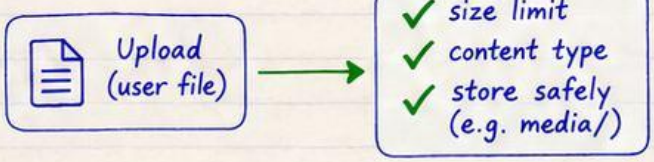
- CORS governs browser cross-origin access.
- It does not stop direct API clients.
- Allow only intended origins; still enforce permissions and CSRF.




CORS ≠ Authentication

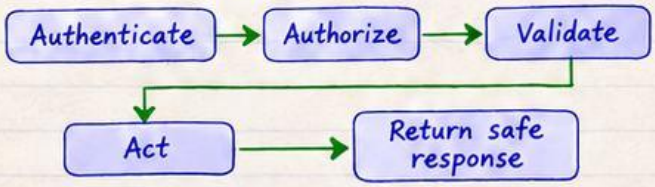
## UPLOADS & ABUSE

- Limit upload size and validate content.
- Treat uploads as untrusted; never execute them.
- Add rate limits for login and costly endpoints.
- Rate limiting is not a substitute for authorization.




## SECURITY CHECK FLOW

- Authenticate -> authorize -> validate -> act -> return safe response.
- Test unauthorized and cross-user requests, not only the happy path.



# 19 / 20 — TEST BEHAVIOR, NOT JUST SUCCESS

## RUN THE CHECKS

`python manage.py check`

→ Runs Django's system checks (config, settings, etc.).

`python manage.py test`

→ Discovers and runs all tests (e.g., in notes/tests.py).

## MEANINGFUL TEST CASES

- Anonymous visitor cannot access private notes.
- Zain can create and edit his own note.
- Zain cannot retrieve, edit or delete Ali's note.
- Blank and two-character titles fail validation.
- Server assigns the owner, ignoring forged owner input.
- GET never deletes a note.
- POST without CSRF is rejected when CSRF checks are enforced.
- Pagination does not reveal other users' notes.

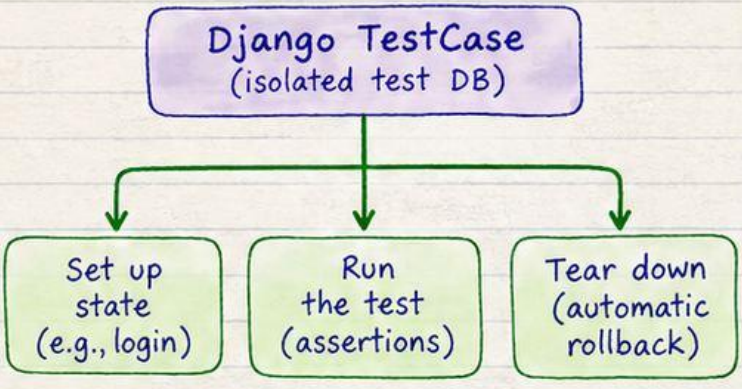
## SMALL TEST — notes/tests.py

```
from django.test import TestCase
from .forms import NoteForm

class NoteFormTests(TestCase):
    def test_short_title_is_rejected(self):
        form = NoteForm(data={"title": "Hi"})
        self.assertFalse(form.is_valid())
        self.assertIn("title", form.errors)
```

## TOOLS TO KNOW

- Django TestCase isolates test database changes.
- client.force_login(user) helps test protected pages.
- DRF APIClient helps test API requests and responses.
- Default test clients relax CSRF checks; use enforce_csrf_checks=True to test CSRF.




## DEBUG WITH PURPOSE

1. Read the traceback, identify the failing line.
2. Reproduce the issue.
3. Add a regression test.
4. Then fix it.

**Result**: the bug is fixed and less likely to return.

⚠ *Never expose DEBUG tracebacks on a public site.*

# 20 / 20 — DEPLOYMENT & FINAL REVISION

## BEFORE GOING LIVE

- DEBUG = False.
- Set ALLOWED_HOSTS to your real domains.
- Load SECRET_KEY and credentials from the environment.
- Use HTTPS and secure session/CSRF cookies.
- Use a production WSGI or ASGI server, not runserver.
- Apply migrations; collect static assets.
- Configure static files and user uploads separately.
- Back up the database and test restoration.
- Add useful logs and error monitoring without secrets.

## RELEASE CHECKS

```
$ python manage.py test
$ python manage.py check --deploy
$ python manage.py migrate
$ python manage.py collectstatic
```

→ Run deploy checks with production settings.

→ Configure STATIC_ROOT before collectstatic.

## STATIC VS MEDIA

- **Static** = CSS, JavaScript and app images.
- **Media** = uploaded user files; treat as untrusted.

## PROJECT COMPLETE WHEN

✓ Zain can log in → create note → see only his notes → edit → delete → log out.

✓ API enforces the same rules and rejects invalid input.

## QUICK MAP

URL → view → permission check → form / serializer → ORM → database → response

- Model: data.
- Migration: schema change.
- Form: HTML input.
- Serializer: API data.
- Authentication: identity.
- Authorization: allowed action.

## REFERENCES

- docs.djangoproject.com/en/5.2/
- django-rest-framework.org

**ZainSpire** • Build small. Understand each step.