Backend with Python

Why Python, the major web frameworks, and a Django starter you can demo quickly.

1) Why Python for Backend?

- Fast to write/read; batteries included.
- Rich ecosystem: Django, Flask, FastAPI, SQLAlchemy, Requests/HTTPX.
- Good enough performance; scale horizontally.
- Strong docs & community → easy onboarding/maintenance.

2) Popular Frameworks

- **Django**: Full■stack, batteries included (ORM, admin, auth, templates, forms).
- **Flask**: Microframework, pick■your■own components.
- **FastAPI**: Modern, async■first, type hints, automatic docs (OpenAPI).

3) Django MVT Recap

- **Model**: Class → DB table via ORM; migrations keep schema in sync.
- **View**: Request handler returning a response.
- **Template**: HTML with template language; renders context.
- **URLconf**: Maps path pattern to a view.
- **Admin**: Auto CRUD for registered models.

4) Quickstart Commands

```
python -m venv .venv && source .venv/bin/activate
pip install "django==5.0.*"
django-admin startproject bookshelf
cd bookshelf
python manage.py startapp library
# add 'library' to INSTALLED_APPS in settings.py
python manage.py makemigrations && python manage.py migrate
python manage.py createsuperuser
python manage.py runserver
```

5) Project Structure — What Each File Does

- **manage.py**: Django CLI (runserver, migrate, shell…).
- **settings.py**: Global config (apps, DB, templates, static, middleware).
- **urls.py**: Project URL router that includes app routers.
- **wsgi.py/asgi.py**: Server entrypoints for production.
- **app/**: Your features (models, views, urls, templates, static, admin).

6) Minimal Book Feature (End■to■End)

Model (library/models.py)

```
from django.db import models
from django.contrib.auth import get_user_model
```

```
class Book(models.Model):
    STATUS_CHOICES = [
        ("to_read", "To Read"),
        ("reading", "Reading"),
        ("read", "Read"),
    ]
    owner = models.ForeignKey(get_user_model(), on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=120)
    status = models.CharField(max_length=10, choices=STATUS_CHOICES, default="to_read")
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.title} by {self.author}"
```

URLs (library/urls.py)

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

urlpatterns = [
    path("", views.BookListView.as_view(), name="book_list"),
    path("book/<int:pk>/", views.BookDetailView.as_view(), name="book_detail"),
    path("book/create/", views.BookCreateView.as_view(), name="book_create"),
    path("book/<int:pk>/update/", views.BookUpdateView.as_view(), name="book_update"),
    path("book/<int:pk>/delete/", views.BookDeleteView.as_view(), name="book_delete"),
]
```

Views (library/views.py — generic CBVs)

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from django.views.generic import ListView, DetailView, CreateView, UpdateView, DeleteView
from .models import Book

class BookListView(LoginRequiredMixin, ListView):
    model = Book
    paginate_by = 10
    def get_queryset(self):
        qs = Book.objects.filter(owner=self.request.user)
        q = self.request.GET.get("q")
        if q: qs = qs.filter(title__icontains=q)
        status = self.request.GET.get("status")
        if status: qs = qs.filter(status=status)
        return qs.order_by("-created_at")

class BookDetailView(LoginRequiredMixin, DetailView):
    model = Book
    def get_queryset(self): return Book.objects.filter(owner=self.request.user)

class BookCreateView(LoginRequiredMixin, CreateView):
    model = Book
    fields = ["title", "author", "status"]
    success_url = reverse_lazy("book_list")
    def form_valid(self, form):
        form.instance.owner = self.request.user
        return super().form_valid(form)

class BookUpdateView(LoginRequiredMixin, UpdateView):
    model = Book
    fields = ["title", "author", "status"]
```

```
    success_url = reverse_lazy("book_list")
    def get_queryset(self): return Book.objects.filter(owner=self.request.user)

class BookDeleteView(LoginRequiredMixin, DeleteView):
    model = Book
    success_url = reverse_lazy("book_list")
    def get_queryset(self): return Book.objects.filter(owner=self.request.user)
```

Templates (snippets)

```
<!-- base.html -->
<link rel="stylesheet" href="{% static 'library/styles.css' %}">
{% if user.is_authenticated %}<a href="{% url 'logout' %}">Logout</a>{% else %}<a href="{% ur
{% block content %}{% endblock %}

<!-- book_list.html -->
<form method="get">
  <input name="q" placeholder="Search title…" value="{{ request.GET.q }}">
  <select name="status"><option value="">All</option>...</select>
  <button>Filter</button>
</form>
<ul>{% for book in object_list %}<li>{{ book.title }}</li>{% endfor %}</ul>
```

7) Where Things Go

- Models → app/models.py
- Views → app/views.py
- URLs → app/urls.py (included by project urls.py)
- Templates → app/templates/<app_name>/*.html (+ base.html)
- Static → app/static/<app_name>/*
- Admin → app/admin.py
- Forms → app/forms.py
- Settings → project/settings.py

8) Common Pitfalls

- Forgot to add app to INSTALLED_APPS.
- Missing {% csrf_token %} in POST forms → 403.
- Wrong template/static paths.
- URL name mismatches in {% url %}.
- Not filtering by owner → data leaks.