Backend Foundations

1) What is the Backend?

The backend is the server■side part of an application that receives client requests (browsers/mobile), applies business logic, interacts with databases and services, and returns responses (often HTML or JSON).

2) Core HTTP Concepts

- **HTTP**: the protocol between client and server.
- **Request**: method, URL, headers, body.
- **Response**: status code, headers, body.
- **Methods**: GET (read), POST (create), PUT/PATCH (update), DELETE (remove).
- **Status codes**: 2xx success, 3xx redirect, 4xx client error, 5xx server error.

3) Keywords & Definitions

- **API**: Interface a program exposes (often HTTP endpoints).
- **Endpoint / Route**: URL path that maps to server logic.
- **Resource**: Named thing in your system (e.g., /api/books/).
- **JSON**: Lightweight data format for API responses.
- **Session**: Server■side or signed data to remember a user across requests.
- **Cookie**: Small browser■stored data; often holds a session ID.
- **Authentication (AuthN)**: Proving who you are (login).
- **Authorization (AuthZ)**: What you're allowed to do (permissions/roles).
- **Business Logic**: Your app's rules (e.g., only owners may edit a book).
- **Controller/View**: Request handler that returns a response.
- **Database (DB)**: Persistent storage (relational or NoSQL).
- **Transaction**: Unit of work that fully succeeds or fails.
- **Index**: Speeds lookups on columns/fields.
- **ORM**: Maps classes to DB tables.
- **Cache**: Fast storage (e.g., Redis) for hot data.
- **Environment/Config**: Settings per env (dev/staging/prod).
- **Observability**: Logs, metrics, traces to understand behavior.

4) Principles (Plain English)

- **Separation of Concerns**: Split responsibilities.
- **Single Responsibility**: Each component has one job.
- **12■Factor App**: Guidelines for cloud■friendly services.
- **Idempotency**: Same request twice → same effect.
- **Least Privilege**: Grant only the access needed.

- **Defense in Depth**: Multiple layers (validation + auth + permissions).

5) What counts as the backend?

- Web/API servers
- Background workers & queues (emails, reports)
- Database layer & migrations
- Integrations (payments, email, storage)
- AuthN/AuthZ services
- Admin dashboards & ops tools

6) Typical Architecture (High■Level)

Browser → Load Balancer/Reverse Proxy → App Server(s) → Database → Cache/Queue → External APIs. Stateless services scale horizontally.

Key Components

- **Load Balancer**: Distributes traffic across app servers.
- **Reverse Proxy (e.g., Nginx)**: Terminates TLS, compression, static files, forwards to app server (Gunicorn/Uvicorn).
- **App Server**: Runs your code (Django/Flask) and returns responses.
- **Database**: Persistent data (PostgreSQL/MySQL; NoSQL options exist).
- **Cache**: Redis to reduce DB load and speed responses.
- **Message Queue**: Redis/RabbitMQ + Celery for background tasks.

7) Sync vs Async Work

- **Synchronous**: Finished during the HTTP request (must be fast).
- **Asynchronous**: Long■running work moved to background jobs.

8) Security Essentials

- Passwords are always hashed (PBKDF2/Argon2).
- Enable **CSRF** protection on unsafe methods.
- Prevent **XSS** by escaping user input in templates.
- Avoid **SQL injection** with ORM/parameterized queries.
- Use HTTPS; never send secrets over plain HTTP.

9) Tooling You Will See

- Servers: Gunicorn (WSGI), Uvicorn (ASGI); Reverse proxy: Nginx.
- Databases: PostgreSQL (recommended), MySQL; SQLite for local dev.
- Caches/Queues: Redis; Celery for background jobs.
- Testing: pytest (+ pytest■django).
- Quality: black, isort, ruff; optional typing with mypy.
- Deploy: Docker/Compose; PaaS (Railway/Render/Heroku■like).

10) REST Quick Rules

- Resources: /api/books/ (list, create), /api/books/<id>/ (detail).
- 201 Created on POST; 204 No Content on DELETE.
- Use query params for filter/sort/pagination (e.g., ?q=term&page;=2&status;=read).
- Keep APIs predictable and documented.