**🌍 What is an API?**

* An **API (Application Programming Interface)** lets two programs **talk to each other**.
* Example: You check your order status online → your browser asks the server “what’s the tracking status?” → the server replies with the info.

**🔑 What is REST?**

* **REST (Representational State Transfer)** is a **style of building APIs**.
* It gives **rules and guidelines** so APIs are predictable and easy to use.

**🛠 How does a REST API work?**

1. **Client → Request**
   * The client (your program or browser) sends a request to a **server endpoint** (a special URL).
   * Example: /shipment?tracking\_id=12345
2. **HTTP Methods**
   * The request also specifies **what you want to do**:
     + GET → Read data (e.g., see shipment status)
     + POST → Create data (e.g., place a new order)
     + PUT → Update data (e.g., edit an order)
     + DELETE → Remove data (e.g., cancel an order)
3. **Headers**
   * Extra info like “I’m sending JSON data” or “Here’s my login token”.
4. **Body (optional)**
   * Data you send along, usually in **JSON format**.
   * Example: { "tracking\_id": "12345" }

**📦 Response from Server**

* The server processes the request and sends back:
  + **Status code** → quick answer about success or failure:
    - 200 OK → success
    - 404 Not Found → resource doesn’t exist
    - 500 Internal Server Error → server problem
  + **Headers** → metadata about the response.
  + **Body** → the actual data you asked for, often JSON.  
    Example: { "status": "Shipped", "expected\_delivery": "2025-08-20" }

**✅ Recap**

* REST API = An API that follows REST principles.
* **Client** sends a request (with method, endpoint, headers, body).
* **Server** replies with a response (status code, headers, body).
* Data is usually exchanged in **JSON**.
* REST makes APIs **structured, predictable, and easy to use**.

Great question 👍 Let’s pull the **differences** clearly out of that text so you can see when to use **FastAPI vs Flask vs Django**.

**🚀 FastAPI**

* **Purpose**: Modern framework built **specifically for APIs**.
* **Setup**: Easy to start (like Flask).
* **Performance**: Very **fast**, thanks to async support.
* **Data validation**: Built-in with Python type hints → automatic request/response validation.
* **Docs**: Auto-generates OpenAPI docs and schema.
* **Security**: Supports standards like OAuth2.
* **Best for**: High-performance APIs, modern Python apps, when you want speed + auto docs + type safety.

**🧩 Flask**

* **Purpose**: Minimal framework (good for simple APIs and small apps).
* **Setup**: Very easy, quick to start.
* **Performance**: Slower than FastAPI (no native async).
* **Tools**: Lightweight — most features (DB, auth, etc.) need third-party extensions.
* **Security**: Needs extensions for full coverage.
* **Best for**: Small/simple APIs or projects where you want full freedom without structure.

**🏗 Django**

* **Purpose**: A **full stack** framework (APIs + web apps).
* **Setup**: More initial setup (project structure, configs).
* **Performance**: Slower compared to FastAPI, heavier than Flask.
* **Tools**: Comes with many built-ins:
  + ORM (database)
  + Template engine (for HTML)
  + Migrations
  + Built-in security protections
* **Scaling**: More challenging because it’s opinionated and integrated.
* **Best for**: Large, full-featured web apps where you need database, admin panel, and strong security out of the box.

**📝 Quick Comparison Table**

| **Feature** | **FastAPI 🚀** | **Flask 🧩** | **Django 🏗** |
| --- | --- | --- | --- |
| Main focus | APIs only (modern) | Minimal apps + APIs | Full web apps + APIs |
| Setup | Easy, quick | Easy, quick | More setup, structured |
| Performance | ⭐ Very high (async) | Medium | Medium–low |
| Data validation | Built-in (type hints) | Manual / add-ons | Manual / serializers |
| Docs generation | Auto (OpenAPI, Swagger) | Not built-in | Not built-in |
| Built-in tools | Few (add yourself) | Few (add yourself) | Many (ORM, templates, migrations) |
| Security | OAuth2, integrations | Extensions needed | Many built-in protections |
| Best for | Modern APIs, speed | Simple APIs | Big apps with database + UI |

✅ **In simple words**:

* **Flask** → good for small/simple APIs, like a “DIY toolkit.”
* **Django** → good for big, full apps where you want most tools included.
* **FastAPI** → the modern middle ground: **fast + simple + feature-rich** for APIs.

Sure! Here's a **short summary**:

**Type Hints in Python (with FastAPI)**

**1. Dynamic Typing in Python:**

* Python **doesn’t require** types; it assigns them at runtime.
* x = "hello" # string
* x = 10 # now integer

**2. Type Hints:**

* **Optional** way to hint what type a variable should be.
* text: str = "hello"
* percentage: int = 90

**3. Functions with Type Hints:**

* Functions can specify input/output types:
* def root(number: int) -> float:
* return number \*\* 0.5

**4. Multiple Types:**

* Use | for multiple types (e.g., int | float):
* value: int | float = 10

**5. Optional Values:**

* Type can be None:
* name: str | None = None

**6. Lists, Tuples, Dicts:**

* Lists: list[int], Tuples: tuple[str, float], Dicts: dict[str, int]
* digits: list[int] = [1, 2, 3]

**7. Custom Classes:**

* You can use **custom classes** for type hints:
* class City:
* pass
* city: City

**8. FastAPI:**

* FastAPI uses **type hints** for automatic **data validation** and **documentation**.

This helps FastAPI auto-validate requests and provide excellent **docs** and **editor suggestions**. You can also use tools like mypy to check types.

Let me know if you'd like a **quick cheat sheet** with examples!