🚀 Step-by-Step: Setup + GET Method in FastAPI

🧪 1. Set up Virtual Environment (Windows/Linux/Mac)

```
# Install virtualenv (if not already installed)
pip install virtualenv

# Create virtual environment
virtualenv venv

# Activate virtual environment
# Windows
venv\Scripts\activate

# macOS/Linux
source venv/bin/activate
```

📦 2. Install FastAPI and Uvicorn

```
pip install fastapi uvicorn
```

🗂 3. Create Project Structure

```
fastapi_project/
│
├── main.py
├── requirements.txt
└── venv/
```

Save dependencies for later:

```
pip freeze > requirements.txt
```

🧠 4. Create `main.py` with GET Methods

✅ Using **Path Parameters**, **Predefined Values**, and **Query Parameters**

```
from fastapi import FastAPI, Query
from typing import Optional
from enum import Enum

app = FastAPI()

# 🧩 Predefined Path Parameter using Enum
class ModelName(str, Enum):
    alexnet = "alexnet"
    resnet = "resnet"
    lenet = "lenet"

# ✅ Basic GET Method with Path Parameter
@app.get("/items/{item_id}")
def read_item(item_id: int):
    return {"item_id": item_id}

# ✅ Path Parameter with Predefined Values
@app.get("/models/{model_name}")
def get_model(model_name: ModelName):
    if model_name == ModelName.alexnet:
        return {"model_name": model_name, "message": "Deep Learning FTW!"}
    elif model_name == ModelName.lenet:
        return {"model_name": model_name, "message": "LeNet is classic!"}
    return {"model_name": model_name, "message": "ResNet Rocks!"}

# ✅ With Query Parameters (Optional + Defaults)
@app.get("/products/")
def get_products(skip: int = 0, limit: int = 10, category: Optional[str] =
Query(None, min_length=3)):
    return {
        "message": "Fetching products",
        "skip": skip,
        "limit": limit,
        "category": category or "all"
    }
```

▶️ 5. Run the Server

```
uvicorn main:app --reload
```

- `--reload` allows auto-reload during development
- Visit: http://127.0.0.1:8000

▧ 6. Explore Interactive Docs (Swagger)

Open in browser:

```
http://127.0.0.1:8000/docs  # Swagger UI
http://127.0.0.1:8000/redoc # ReDoc
```

🔍 Usage Examples

- `GET /items/42` → `{"item_id": 42}`
- `GET /models/alexnet` → Returns custom message
- `GET /products/?skip=5&limit=2&category=shoes`

🧾 Summary Notes

Feature	Example	Description
Path Param	`/items/{id}`	URL-based dynamic value
Enum Param	`/models/{model_name}`	Restrict value to predefined set
Query Param	`/products/?skip=0&limit=10`	Additional filters, optional/default values
Swagger UI	`/docs`	Interactive API documentation
Virtualenv	`venv\Scripts\activate` / `source venv`	Isolate project dependencies

Absolutely! Here's a well-structured and beginner-to-advanced **note on FastAPI Routers** including why **routers are important**, how **validation works behind the scenes using Pydantic**, and how all of it comes together.

✅ 1. **Why Use Routers in FastAPI?**

Routers are used to **modularize your FastAPI app** — just like how controllers work in other frameworks (like Express.js, Django, Laravel).

✅ **Benefits:**

- 🔗 Separation of concerns: Separate logic by features (e.g., `/products`, `/users`)
- 📦 Scalability: Easier to manage larger codebases
- ♻️ Reusability: Routers can be reused and nested
- 🧪 Testability: Each router can be tested independently
- 🧼 Clean Code: Keeps your `main.py` small and readable

🧩 2. **What Is a Router in FastAPI?**

A `router` is an instance of `APIRouter`, where you can define endpoints just like `@app.get()` but independently.

💡 **Example:**

```python
# routers/products.py

from fastapi import APIRouter

router = APIRouter()

@router.get("/products")
def get_products():
    return {"msg": "List of products"}
```

⚙️ 3. **How to Register a Router**

You register routers in your main app like this:

```python
# app/main.py

from fastapi import FastAPI
from app.routers import products

app = FastAPI()
app.include_router(products.router, prefix="/api/v1", tags=["Products"])
```

🧠 This will expose the route: `/api/v1/products`

🔍 4. **Behind the Scenes: Validation Using Pydantic**

FastAPI uses **Pydantic** to validate:

☑ **Path Parameters**

☑ Query Parameters ☑ Request Body (JSON/Post Data)

Example:

```python
from pydantic import BaseModel, Field

class Product(BaseModel):
    name: str = Field(..., min_length=3)
    price: float
```

Behind the scenes:

- When a request hits the endpoint, FastAPI automatically parses and validates the incoming JSON into a `Product` object.
- If data is **invalid**, FastAPI **auto-generates a 422 error response** with all validation issues.
- You never have to manually `try/except` invalid input. Pydantic does it for you.

📦 5. **Using Pydantic with Routers**

```python
# routers/products.py

from fastapi import APIRouter
from app.models.product import Product

router = APIRouter()

@router.post("/products")
def create_product(product: Product):
    return {"product": product}
```

- 🧠 When someone sends a POST request with JSON body, it's **automatically parsed and validated**.
- FastAPI converts it into a Python object (`Product`).
- If fields are missing or invalid, it returns a structured error.

🧾 6. Validation Error Response Format (Auto-generated):

```json
{
  "detail": [
    {
      "loc": ["body", "product", "name"],
      "msg": "field required",
      "type": "value_error.missing"
    }
  ]
}
```

🔍 7. FastAPI Dependency Injection with Routers (Advanced)

You can pass `dependencies` into routers for:

- Auth middleware
- DB connection injection
- Role-based access

```python
router = APIRouter(
    prefix="/products",
    tags=["Products"],
    dependencies=[Depends(auth_dependency)],
)
```

🧠 Summary: Why Routers + Pydantic Matter

Concept	Description
Routers	Help organize routes modularly, cleanly
Pydantic Models	Define data schemas + validation rules for request/response
Auto Validation	FastAPI + Pydantic auto-validate data before your logic runs
Error Handling	Validation errors are returned with proper HTTP status + reason
Docs Integration	Models are reflected in Swagger Docs for each route automatically

---

🔍 FastAPI Path Parameters

🔐 Predefined Values (Enum)

🔎 Query Parameters

Each section includes:

- What it is
- How it works
- Code examples
- Behavior behind the scenes (Pydantic + FastAPI)

📌 1. **Path Parameters in FastAPI**

☑ What are Path Parameters?

Path parameters are dynamic parts of the URL path that act as **variables**.

🧠 **FastAPI treats them as required parameters** and maps them using Python function parameters.

🧾 Syntax Example:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/users/{user_id}")
```

```python
def read_user(user_id: int):
    return {"user_id": user_id}
```

🛠️ How it Works:

- URL: `/users/42`
- FastAPI converts `"42"` to an `int` (based on type hint)
- If type mismatch (e.g., `/users/abc`), FastAPI returns a `422` error automatically.

⚙️ Behind the Scenes:

FastAPI uses:

- **Python type hints** to enforce type
- **Pydantic** to validate the value before function execution
- Generates **OpenAPI docs** automatically with correct parameter types

✅ Optional Parameters?

No. **Path parameters must be required.** If optional, you must redesign using **query parameters** instead.

🎯 2. **Predefined Values with Enums (Validation)**

✅ What are Predefined Values?

Sometimes, you want to restrict a path parameter to specific **allowed values** — not anything.

Use **Python Enums** for this.

🧱 Example:

```python
from enum import Enum
from fastapi import FastAPI

app = FastAPI()

class ModelName(str, Enum):
    alexnet = "alexnet"
    resnet = "resnet"
    lenet = "lenet"

@app.get("/models/{model_name}")
def get_model(model_name: ModelName):
    return {"model_name": model_name}
```

🌐 Usage:

- `/models/alexnet` ✅
- `/models/invalid_model` ❌ → 422 Validation Error

🔍 Why Use Enum?

- Ensures clients use only valid values
- Auto-validates input (via Pydantic)
- Automatically shows options in Swagger Docs dropdown
- Prevents typos and unexpected inputs

⚙️ How it Works:

- FastAPI internally maps the path to the Enum.
- If not part of the Enum, it returns `422` with a message like:

```
{
  "detail": [
    {
      "loc": ["path", "model_name"],
      "msg": "value is not a valid enumeration member",
      "type": "type_error.enum"
    }
  ]
}
```

📥 3. **Query Parameters in FastAPI**

✅ What are Query Parameters?

Parameters that appear **after the** `?` in a URL. Used for filtering, sorting, pagination, searching, etc.

🧾 Example:

```
@app.get("/products")
def get_products(skip: int = 0, limit: int = 10, q: str = None):
    return {"skip": skip, "limit": limit, "q": q}
```

📌 URL: `/products?skip=5&limit=20&q=shoes`

💡 Key Points:

- Not positional like path parameters
- Can be optional or have default values

- Auto-validated using type hints

🧰 Advanced: Use `Query` for extra validation

```
from fastapi import Query

@app.get("/search")
def search_items(q: str = Query(..., min_length=3, max_length=50)):
    return {"query": q}
```

- `...` = required
- `min_length`, `max_length`, `regex` supported
- Shows up in API docs with proper validation

🦉 Pydantic + Query Magic:

Pydantic validates:

- Required/Optional values
- Data types (e.g., `int`, `bool`, `float`, etc.)
- Value constraints (min/max length, regex)

And **FastAPI auto-generates**:

- Swagger documentation
- Interactive query parameter fields
- Error responses for invalid queries

📥 Optional Query Parameters

```
@app.get("/filter")
def filter_items(category: str = Query(None), in_stock: bool = Query(False)):
    return {"category": category, "in_stock": in_stock}
```

🧠 Summary Table

Feature	Path Parameters	Predefined (Enum)	Query Parameters
URL Format	`/items/{id}`	`/models/{model_name}`	`/products?skip=0&limit=10`
Required	✅ Always	✅ Always	❌ Can be optional

Feature	**Path Parameters**	**Predefined (Enum)**	**Query Parameters**
Type Validation	✅ (int, str, etc.)	✅ Only allowed values via Enum	✅ With type hints & Query()
Auto Swagger Docs	✅	✅ Dropdown with values	✅ Shows default and constraints
Common Use Case	Dynamic routes (ID, slug)	Model names, categories, user roles	Filtering, searching, pagination
Validation Method	Pydantic via type hints	Pydantic Enum	Pydantic + fastapi.Query