FastAPI Parameters:

Overview
Parameters are the data sent by the client (e.g., browser, mobile app, or another tool) to the server. In FastAPI, this data can come in different ways, and each method has its specific use case.
We’ll cover three main types of parameters:

- **Path Parameters**
- **Query Parameters**
- **Request Body**

FastAPI is a modern, high-performance Python framework for building APIs. It leverages Python’s type hints for automatic validation and generates interactive documentation (like Swagger UI).

---

1. Path Parameters

What Are Path Parameters?

Path parameters are variables placed directly in the URL path to access specific resources (like a product or user ID).

Example URL

/products/456

Code Example

```
from fastapi import FastAPI, Path

app = FastAPI()

@app.get("/products/{product_id}")
async def read_product(
    product_id: int = Path(
        ...,
        title="Product ID",
        description="A unique identifier for the product",
        ge=1,
        le=10000
    )
):
    return {"product_id": product_id, "message": f"Fetching details for product {product_id}"}
```

Explanation

- `@app.get("/products/{product_id}")`: Defines a GET endpoint that takes `product_id` as a path parameter.
- `product_id: int`: Ensures the parameter is an integer.
- `Path(...)` includes:
  - `...`: Required field.
  - `title` and `description`: Shown in Swagger UI.
  - `ge=1`, `le=10000`: Adds validation.

Response Example

```
{
  "product_id": 456,
  "message": "Fetching details for product 456"
}
```

Try This

- Test `/products/10` → Valid
- Test `/products/0` → Returns validation error (422)

---

2. Query Parameters

What Are Query Parameters?

Query parameters are key-value pairs passed in the URL after `?`. Used for filtering, pagination, etc.

Example URL

/products?search=phone&skip=10&limit=20

Code Example

```
from fastapi import FastAPI, Query

app = FastAPI()

@app.get("/products/")
async def search_products(
    search: str | None = Query(
        None,
        title="Search Query",
        description="Search term to filter products",
        min_length=3,
        max_length=100,
        pattern="^[a-zA-Z0-9 ]+$"
    ),
    skip: int = Query(0, ge=0),
    limit: int = Query(10, ge=1, le=100)
):
    return {
        "search": search,
        "skip": skip,
        "limit": limit,
        "message": f"Searching for '{search}' with skip={skip} and limit={limit}"
    }
```

Explanation

- `search: str | None`: Optional parameter, allows filtering.
- `Query(...)`: Adds validations.
- `skip` & `limit`: Help with pagination.

Response Example

```
{
  "search": "phone",
  "skip": 10,
  "limit": 20,
  "message": "Searching for 'phone' with skip=10 and limit=20"
}
```

Try This

- Test `/products?search=phone&skip=5&limit=15`
- Try invalid values like `search=ab` or `limit=150` to see error handling.

---

3. Request Body

What is Request Body?

Used to send complex data in JSON format—commonly with POST, PUT, or PATCH requests.

Example JSON

```
{
  "name": "Smartphone",
  "description": "Latest model with 5G support",
  "price": 699.99
}
```

Code Example

```
from fastapi import FastAPI, Path, Query, Body
from pydantic import BaseModel

class Product(BaseModel):
    name: str
    description: str | None = None
    price: float

app = FastAPI()

@app.post("/products/validated/{product_id}")
async def create_product(
    product_id: int = Path(..., title="Product ID", ge=1),
    search: str | None = Query(None, min_length=3),
    product: Product = Body(...)
):
    result = {
        "product_id": product_id,
        "message": f"Created product with ID {product_id}"
    }
    if search:
        result.update({"search": search})
    if product:
        result.update({"product": product.model_dump()})
    return result
```

Explanation

- `Product` model defines required JSON structure.
- `Body(...)`: Tells FastAPI to expect a JSON body matching `Product`.

Response Example

```
{
  "product_id": 123,
  "search": "phone",
  "product": {
    "name": "Smartphone",
    "description": "Latest model with 5G support",
    "price": 699.99
  },
  "message": "Created product with ID 123"
}
```

Try This

- Test with valid JSON data.
- Try invalid data (e.g., `price="abc"`) to see validation error.

Complete Code (main.py)

```
from fastapi import FastAPI, Path, Query, Body
from pydantic import BaseModel

app = FastAPI()
```

```
class Product(BaseModel):
    name: str
    description: str | None = None
    price: float

@app.get("/products/{product_id}")
async def read_product(
    product_id: int = Path(..., title="Product ID", ge=1, le=10000)
):
    return {"product_id": product_id, "message": f"Fetching details for product {product_id}"}

@app.get("/products/")
async def search_products(
    search: str | None = Query(None, min_length=3, max_length=100, pattern="^[a-zA-Z0-9
]+$"),
    skip: int = Query(0, ge=0),
    limit: int = Query(10, ge=1, le=100)
):
    return {
        "search": search,
        "skip": skip,
        "limit": limit,
        "message": f"Searching for '{search}' with skip={skip} and limit={limit}"
    }

@app.post("/products/validated/{product_id}")
async def create_product(
    product_id: int = Path(..., ge=1),
    search: str | None = Query(None, min_length=3),
    product: Product = Body(...)
):
    result = {
        "product_id": product_id,
        "message": f"Created product with ID {product_id}"
    }
    if search:
        result.update({"search": search})
    if product:
        result.update({"product": product.model_dump()})
    return result
```

Running the Application

Step 1: Setup Environment

```
uv init fastapi_p1
cd fastapi_p1
uv venv
source .venv/bin/activate
uv add "fastapi[standard]"
```

Step 2: Save Code

Save the code above into a file named `main.py`.

Step 3: Run the Server

```
fastapi dev main.py
```

Step 4: Test the API

Open your browser and go to:

http://localhost:8000/docs

Key Points Summary

- **Path Parameters**: Identify specific items (e.g., `/users/1`)
- **Query Parameters**: Used for optional data like search, sort, pagination
- **Request Body**: Used for sending structured data (usually with POST or PUT)
- **Validation**: FastAPI validates inputs automatically
- **Swagger UI**: Provides interactive API documentation

Project Tips

- **E-Commerce App**: Use path for product IDs, query for search/filter, body for product data.
- **User Management**: Use path for user ID, query for roles, body for user profile creation.

- **Always Add Validation**: Prevents errors, ensures data quality, improves API security.

Aap chahein to mai isay PDF format mein bhi export kar ke de sakta hoon. Chahein?