FastAPI Parameters: Detailed Explanation

Parameters wo data hote hain jo client (jaise browser, mobile app, ya koi aur tool) server ko bhejta hai. FastAPI mein yeh data alag alag tareekon se aata hai, aur har tareeke ka apna specific use case hota hai. Main tumhe teen main types (Path Parameters, Query Parameters, Request Body)

FastAPI ek modern Python framework hai jo APIs banane ke liye use hota hai. Yeh bohot tezi se kaam karta hai aur developer-friendly hai. FastAPI ka sabse bada faida yeh hai ke yeh Python ke type hints ka use karta hai, jo automatically data validate karta hai aur interactive documentation banata hai (jaise Swagger UI). Yani, tum jo API banao, uski testing aur documentation apne aap ban jati hai.

Example: Agar tum ek API banana chahte ho jo users ki details fetch kare, toh FastAPI ke sath yeh kaam bohot asaan hai. Tum bas code likho, aur FastAPI baaki validation aur presentation ka khayal rakhta hai.

---

1. Path Parameters

Path Parameters wo variables hote hain jo URL ke path mein directly shamil hote hain. Yeh usually specific resources (jaise ek user, product, ya item) ko identify karne ke liye use hote hain.

Real-World Example:

Maan lo tum ek e-commerce website ke liye API bana rahi ho. Tumhe ek specific product ki details fetch karni hain, jiska ID hai `456`. Toh URL aisa hoga:

```
/products/456
```

Yahan `456` path parameter hai.

Code Example:

```python
from fastapi import FastAPI, Path

app = FastAPI()

@app.get("/products/{product_id}")
```

```
async def read_product(
    product_id: int = Path(
        ...,  # Required parameter
        title="Product ID",
        description="A unique identifier for the product",
        ge=1,  # Must be >= 1
        le=10000  # Must be <= 10000
    )
):
    return {"product_id": product_id, "message": f"Fetching details for product {product_id}"}
```

Detailed Explanation:

- **`@app.get("/products/{product_id}")`**: Yeh ek GET endpoint hai jo `/products/456` jaisa URL handle karega. `{product_id}` ek placeholder hai jo actual value (jaise `456`) se replace ho jata hai.
- **`product_id: int`**: Yeh batata hai ke `product_id` ek integer hona chahiye. Agar client non-integer value (jaise `/products/abc`) bhejta hai, toh FastAPI automatically 422 error dega.
- **`Path(...)`**: Yeh `Path` function validation aur metadata ke liye use hota hai:
  - `...` (ellipsis): Yeh kehta hai ke `product_id` required hai. Agar client isko nahi bhejta, toh error aayega.
  - `title` aur `description`: Yeh documentation ke liye metadata hain, jo Swagger UI mein dikhte hain.
  - `ge=1`: Yeh validation hai ke `product_id` 1 ya usse bada hona chahiye.
  - `le=10000`: Yeh kehta hai ke `product_id` 10000 se zyada nahi ho sakta.
- **Response**: Agar tum `/products/456` hit karo, toh response hoga:
  ```json
  {
    "product_id": 456,
    "message": "Fetching details for product 456"
  }
  ```
- **Error Handling**: Agar tum galat value dete ho, jaise `/products/0` ya `/products/15000`, toh FastAPI 422 Unprocessable Entity error dega aur batayega ke value `ge=1` ya `le=10000` ke rule ko violate karta hai.

Practical Use:

- Path parameters ka use tab hota hai jab tumhe ek specific resource fetch ya update karna ho, jaise:

  - `/users/123` (user ID 123 ki details)
  - `/orders/789` (order ID 789 ka status)
- **Swagger UI**: Jab tum `http://localhost:8000/docs` kholo, toh yeh endpoint dikhega. Tum `product_id` ke liye input daal sakte ho aur "Execute" button se test kar sakte ho. Agar validation fail hoti hai, toh error message clearly dikhega.

Try This*:
- Server chala kar browser mein `/products/10` aur `/products/0` try karo. Dekho kaise FastAPI validation errors deta hai.
- Swagger UI mein endpoint test karo aur validation rules check karo.

---

2. Query Parameters**

Query Parameters wo key-value pairs hote hain jo URL ke baad `?` ke sath aate hain. Yeh optional hote hain aur filtering, sorting, ya pagination ke liye use hote hain.

****Real-World Example**:**

Tum ek search API bana rahi ho jo products ko search karta hai. Client search query, page number, aur limit bhej sakta hai:
```
/products?search=phone&skip=10&limit=20
```
Yahan:
- `search=phone`: Search query
- `skip=10`: 10 products skip karo (pagination ke liye)
- `limit=20`: 20 products return karo

****Code Example**:**

```python
from fastapi import FastAPI, Query

app = FastAPI()

@app.get("/products/")
async def search_products(
    search: str | None = Query(
        None,  # Optional parameter
        title="Search Query",
        description="Search term to filter products",
```

```
        min_length=3,
        max_length=100,
        pattern="^[a-zA-Z0-9 ]+$"  # Only alphanumeric and spaces
    ),
    skip: int = Query(0, ge=0, description="Number of products to skip"),
    limit: int = Query(10, ge=1, le=100, description="Max number of products to
return")
):
    return {
        "search": search,
        "skip": skip,
        "limit": limit,
        "message": f"Searching for '{search}' with skip={skip} and limit={limit}"
    }
```

****Detailed Explanation**:**

- **`@app.get("/products/")`**: Yeh endpoint query parameters ke sath kaam karta hai.
- **`search: str | None`**: Yeh kehta hai ke `search` ek string ho sakta hai ya `None` (optional). Agar client `search` nahi bhejta, toh `None` use hoga.
- **`Query(...)`**:
  - `None`: Default value `None` hai, yani optional.
  - `min_length=3`: Agar `search` diya jata hai, toh kam se kam 3 characters hone chahiye.
  - `max_length=100`: Maximum 100 characters tak allowed hain.
  - `pattern="^[a-zA-Z0-9 ]+$"`: Yeh regex kehta hai ke `search` mein sirf letters, numbers, aur spaces allowed hain. Special characters (jaise `@`, `#`) nahi chalenge.
- **`skip: int = Query(0, ge=0)`**: `skip` ek integer hai, default value 0, aur negative nahi ho sakta.
- **`limit: int = Query(10, ge=1, le=100)`**: `limit` default 10 hai, 1 se kam nahi ho sakta, aur 100 se zyada nahi ho sakta.
- **Response**: Agar tum `/products?search=phone&skip=10&limit=20` hit karo, toh response hoga:
  ```json
  {
    "search": "phone",
    "skip": 10,
    "limit": 20,
    "message": "Searching for 'phone' with skip=10 and limit=20"
  }
  ```

- **Error Handling**: Agar tum invalid values dete ho, jaise `search=ab` (too short) ya `limit=150` (too high), toh FastAPI 422 error dega aur batayega ke kaunsa rule violate hua.

Practical Use:
- Query parameters ka use tab hota hai jab tumhe data filter ya paginate karna ho, jaise:
  - Search APIs (`/products?category=electronics`)
  - Pagination (`/users?skip=20&limit=10`)
  - Sorting (`/items?sort=price&order=desc`)
- **Swagger UI**: Docs mein tumhe `search`, `skip`, aur `limit` ke liye input fields milenge. Tum invalid values daal kar dekho ke validation errors kaise aate hain.

Try This:
- Server chala kar `/products?search=phone&skip=5&limit=15` try karo.
- Invalid inputs try karo, jaise `search=a` (too short) ya `limit=200` (too high), aur error messages dekho.
- Swagger UI mein regex validation test karo (jaise `search=phone@123` daal kar dekho).

---

3. Request Body

Request Body tab use hota hai jab client ko complex data server ko bhejna ho, usually JSON format mein. Yeh POST, PUT, ya PATCH requests ke sath common hai, jaise new resource create karna ya existing resource update karna.

Real-World Example:
Tum ek API bana rahi ho jo new product create karta hai. Client JSON data bhejta hai:
```json
{
  "name": "Smartphone",
  "description": "Latest model with 5G support",
  "price": 699.99
}
```

Code Example:
```python
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, description="Optional search
term"),
    product: Product = Body(..., description="Product data (JSON 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
```

Detailed Explanation:**

- **`class Product(BaseModel)`**: Yeh Pydantic model define karta hai jo request body ka structure batata hai:
  - `name: str`: Product ka naam, required field.
  - `description: str | None = None`: Description optional hai.
  - `price: float`: Price required hai aur float hona chahiye.
- **`@app.post("/products/validated/{product_id}")`**: Yeh POST endpoint path parameter (`product_id`), query parameter (`search`), aur request body (`product`) handle karta hai.
- **`product: Product = Body(...)`**: Yeh kehta hai ke request body mein `Product` type ka JSON data required hai.
- **Request Example**: Tum Swagger UI se yeh JSON bhej sakte ho:
  ```json
  {
    "name": "Smartphone",
    "description": "Latest model with 5G support",
    "price": 699.99
  }
  ```

```
- **Response**: Agar tum `/products/validated/123?search=phone` hit karo, toh response hoga:
```json
{
  "product_id": 123,
  "search": "phone",
  "product": {
    "name": "Smartphone",
    "description": "Latest model with 5G support",
    "price": 699.99
  },
  "message": "Created product with ID 123"
}
```
- ****Error Handling****: Agar tum invalid JSON bhejte ho, jaise `price="abc"` (string instead of float) ya `name` miss karte ho, toh FastAPI 422 error dega aur batayega ke kaunsa field galat hai.

****Practical Use****:
- Request body ka use tab hota hai jab tumhe complex data bhejna ho, jaise:
  - New user create karna (`POST /users`)
  - Product update karna (`PUT /products/123`)
  - Form submission (`POST /forms`)
- ****Swagger UI**:** Docs mein tumhe JSON editor milega jahan tum `Product` ka data daal sakte ho. Invalid data try karo (jaise `price="invalid"`) aur error messages dekho.

****Try This****:
- Server chala kar Swagger UI se POST request test karo.
- Valid JSON aur invalid JSON (jaise missing `name` ya wrong `price`) bhej kar dekho.
- Response aur error messages compare karo.

****Complete Code****

Yeh complete `main.py` code hai jo upar ke teenon examples ko combine karta hai:

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

app = FastAPI()
```
```

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

# Path Parameter Example
@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}"}

# Query Parameter Example
@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, description="Number of products to skip"),
    limit: int = Query(10, ge=1, le=100, description="Max number of products to
return")
):
    return {
        "search": search,
        "skip": skip,
        "limit": limit,
        "message": f"Searching for '{search}' with skip={skip} and limit={limit}"
    }

# Request Body Example
@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, description="Optional search
term"),
    product: Product = Body(..., description="Product data (JSON 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

1. **Setup**:
   ```bash
   uv init fastdca_p1
   cd fastdca_p1
   uv venv
   source .venv/bin/activate
   uv add "fastapi[standard]"
   ```

2. **Save Code**:
   Upar diya hua code `main.py` mein save karo.

3. **Run Server**:
   ```bash
   fastapi dev main.py
   ```

4. **Test in Docs**:
   Browser mein `http://localhost:8000/docs` kholo aur endpoints test karo.

Key Points (Yaad Rakho)

1. **Path Parameters**:
   - URL ke hisse hote hain.
   - `Path()` se validate karo (jaise `ge`, `le`, `pattern`).
   - Specific resources ke liye use hote hain.
2. **Query Parameters**:
   - URL ke baad `?` ke sath aate hain.

  - `Query()` se validate karo (jaise `min_length`, `max_length`, `regex`).
  - Filtering, sorting, pagination ke liye use hote hain.
3**. **Request Body****:
  - JSON data ke liye Pydantic models use karo.
  - Complex data (jaise forms, objects) ke liye ideal.
4. ****Validation**:**
  - FastAPI automatically 422 error deta hai agar validation fail ho.
  - Error messages detailed hote hain, jo client ko samajhne mein madad karte hain.
5**. **Swagger UI**:**
  - `http://localhost:8000/docs` pe interactive testing karo.
  - Har endpoint ke liye input fields aur error messages clearly dikhte hain.

Practical Tips for Projects

- **E-commerce API**: Path parameters use karo product IDs ke liye (`/products/123`), query parameters use karo filtering ke liye (`/products?category=electronics`), aur request body use karo new products create karne ke liye (`POST /products`).
- **User Management**: Path parameters use karo user profiles ke liye (`/users/456`), query parameters use karo search ya pagination ke liye (`/users?role=admin&limit=10`), aur request body use karo user creation ke liye (`POST /users`).
- **Error Handling**: Hamesha validations define karo taake client galat data na bhej sake. Yeh security aur reliability ke liye important hai.
