✨ FastAPI Metadata & Documentation Enhancements

> Operation Description | Status Code | Tags | Summary | Response Description

These features help you:

- 📘 Improve Swagger & ReDoc documentation
- 🧪 Clarify the API behavior for consumers
- ✅ Add rich descriptions and metadata

📌 1. **Operation Description Overview**

You can enhance each endpoint with:

- `summary`: 📝 A short title shown in docs
- `description`: 📖 A detailed explanation (supports Markdown)
- `response_description`: ✅ Message shown for successful responses

💡 Basic Example

```python
from fastapi import FastAPI

app = FastAPI()

@app.get(
    "/users/{user_id}",
    summary="Get a user by ID",
    description="Fetches a user based on their unique identifier. Useful for profile views.",
    response_description="User data retrieved successfully."
)
def get_user(user_id: int):
    return {"user_id": user_id}
```

🧠 This enhances your `/docs` interface automatically!

✅ 2. **Status Codes**

FastAPI automatically:

- Infers status codes (`200`, `201`, etc.)
- Lets you manually define them for clarity or correctness

🧪 Custom Status Code Example

```
from fastapi import status

@app.post("/items/", status_code=status.HTTP_201_CREATED)
def create_item():
    return {"message": "Item created"}
```

✅ You can also use plain integers:

```
@app.delete("/items/{item_id}", status_code=204)
def delete_item(item_id: int):
    return
```

💡 Common Status Code Shortcuts (from `fastapi.status`):

Constant	Code	Meaning
`HTTP_200_OK`	200	Success
`HTTP_201_CREATED`	201	Resource created
`HTTP_204_NO_CONTENT`	204	Deleted, no content
`HTTP_400_BAD_REQUEST`	400	Validation/client error
`HTTP_401_UNAUTHORIZED`	401	Auth required
`HTTP_404_NOT_FOUND`	404	Resource not found
`HTTP_500_INTERNAL_SERVER_ERROR`	500	Server error

🏷️ 3. Tags (Categorizing APIs)

Tags help **group related endpoints** in your docs UI (Swagger & ReDoc).

```
from fastapi import APIRouter

router = APIRouter()

@router.get("/products", tags=["Products"])
def get_products():
    return ["Phone", "Tablet"]

@router.post("/products", tags=["Products"])
def add_product():
    return {"msg": "Product added"}
```

📌 This creates a **"Products"** section in your Swagger UI.

You can also define tags globally:

```
app = FastAPI(
    title="My Store API",
    description="Cool API for eCommerce",
    version="1.0.0",
    openapi_tags=[
        {"name": "Products", "description": "Manage all product-related
endpoints"},
        {"name": "Users", "description": "User authentication & management"}
    ]
)
```

---

4. Summary vs. Description

Property	Purpose	Visible In
`summary`	One-liner for the route	Swagger UI
`description`	Detailed markdown-supported explanation	Swagger UI

Best practice: Use `summary` for a TL;DR, `description` for full context.

Example with Markdown Description:

```
@app.get(
    "/login",
    summary="User login",
    description="""
Login API for users.

- Accepts email and password
- Returns JWT token if successful
- Rate-limited to prevent abuse
    """,
    response_description="Login success response"
)
def login():
    return {"token": "abc.def.ghi"}
```

This renders well in Swagger with formatting.

---

5. Response Description

Defines the **meaning of the response on success** (200/201/etc.)

```python
@app.post(
    "/register",
    summary="Register new user",
    response_description="User created successfully"
)
def register():
    return {"msg": "Welcome!"}
```

🔚 Final Cheat Sheet

Feature	Parameter Name	Example Value	Purpose
Summary	`summary`	"Get a user by ID"	Short label for endpoint
Description	`description`	"Returns user with full data"	Markdown-rich explanation
Response Description	`response_description`	"User data fetched"	Clarifies what the response means
Status Code	`status_code`	`status.HTTP_201_CREATED`	Controls the HTTP status code returned
Tags	`tags`	`["Products"]`	Categorizes in docs UI

🧾 Example: All in One

```python
@app.post(
    "/users/",
    summary="Create new user",
    description="""
This endpoint creates a new user.

- Requires name and email
- Returns the created user's ID
- Can raise 400 if data is invalid
    """,
    response_description="User created successfully",
    status_code=status.HTTP_201_CREATED,
    tags=["Users"]
)
def create_user(name: str, email: str):
    return {"id": 1, "name": name, "email": email}
```
