⚡ FastAPI Other Concepts Overview

🚨 1. Error Handling in FastAPI

☑ Built-in Exception Handling

FastAPI provides built-in exceptions like `HTTPException`.

```
from fastapi import FastAPI, HTTPException

app = FastAPI()

@app.get("/item/{item_id}")
def read_item(item_id: int):
    if item_id == 0:
        raise HTTPException(status_code=404, detail="Item not found 🚫")
    return {"item_id": item_id}
```

☑ Custom Exception Handling

```
from fastapi import Request
from fastapi.responses import JSONResponse
from fastapi.exceptions import RequestValidationError

@app.exception_handler(RequestValidationError)
async def validation_exception_handler(request: Request, exc:
RequestValidationError):
    return JSONResponse(
        status_code=422,
        content={"message": "Validation failed 😵", "details": exc.errors()},
    )
```

🔐 Best Practices

- Use descriptive `detail` in errors.
- Handle common errors (e.g., 404, 400, 500) globally.
- Avoid leaking sensitive info in error messages.

📬 2. Custom Responses

☑ Using `JSONResponse`, `HTMLResponse`, `PlainTextResponse`

```python
from fastapi.responses import JSONResponse, HTMLResponse, PlainTextResponse

@app.get("/json")
def custom_json():
    return JSONResponse(content={"msg": "Hello JSON 👋"})

@app.get("/html")
def custom_html():
    return HTMLResponse(content="<h1>Hello HTML 👨‍💻</h1>")

@app.get("/text")
def custom_text():
    return PlainTextResponse(content="Hello Text 📝")
```

🔐 Best Practices

- Use appropriate content types.
- Set custom status codes when needed (`status_code=201`).
- Always document your custom responses using `@app.get(..., responses={})`.

📝 3. Headers in FastAPI

☑️ Accessing Request Headers

```python
from fastapi import Header

@app.get("/headers/")
def read_headers(user_agent: str = Header(...)):
    return {"User-Agent": user_agent}
```

☑️ Setting Response Headers

```python
from fastapi.responses import Response

@app.get("/set-header/")
def custom_header(response: Response):
    response.headers["X-Custom-Header"] = "FastAPI 💚"
    return {"message": "Custom header sent!"}
```

🔐 Best Practices

- Avoid sensitive data in headers.
- Use standard naming conventions (`X-Custom-Header`, etc).

🍪 4. Cookies in FastAPI

☑ Setting Cookies

```
@app.get("/set-cookie/")
def set_cookie(response: Response):
    response.set_cookie(key="session_id", value="abc123", httponly=True)
    return {"message": "🍪 Cookie set!"}
```

☑ Reading Cookies

```
from fastapi import Cookie

@app.get("/get-cookie/")
def get_cookie(session_id: str = Cookie(None)):
    return {"session_id": session_id}
```

🔐 Best Practices

- Use `httponly=True` to prevent JavaScript access.
- Set `secure=True` for HTTPS.
- Prefer JWTs or sessions over plain cookies.

📝 5. Form Data Handling

☑ Receiving Form Data

```
from fastapi import Form

@app.post("/login/")
def login(username: str = Form(...), password: str = Form(...)):
    return {"username": username, "message": "Login form submitted ☑"}
```

🔐 Best Practices

- Always use `Form` for `x-www-form-urlencoded` POST data.
- Use HTTPS to protect credentials.
- Consider CSRF protection for forms in production apps.

🔄 6. CORS (Cross-Origin Resource Sharing)

☑ Enable CORS Middleware

```python
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

origins = [
    "http://localhost:3000",  # React frontend
    "https://myapp.com",      # Production domain
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,          # or ["*"] for all
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Best Practices

- In production, restrict `allow_origins` to known domains.
- Don't use `["*"]` for `allow_credentials=True` (violates spec).
- Allow only necessary headers and methods.

Flow Overview Diagram (Text-Based)

```
Frontend ⟳ ⟶ CORS Middleware 🌍
          ⬇
     FastAPI Endpoint 🧠
          ⬇
   - Read Headers/Cookies
   - Parse Form Data
   - Handle Errors (Custom/Built-in)
   - Generate Response
          ⬇
    Response with Headers/Cookies ⬅ BACK
```

Summary Table

Feature	Module	Key Functionality
Error Handling	`HTTPException`, `exception_handler()`	Global & custom error responses
Custom Responses	`JSONResponse`, `HTMLResponse`	Control response format and headers
Headers	`Header`, `Response.headers`	Read and write HTTP headers

Feature	Module	Key Functionality
Cookies	`Cookie`, `response.set_cookie()`	Store sessions or metadata in cookies
Form Data	`Form`	Parse form submissions
CORS	`CORSMiddleware`	Secure frontend-backend communication

☑️ Final Best Practices Checklist

- ☑️ 🛡️ Handle all exceptions gracefully.
- ☑️ 💬 Provide meaningful error messages.
- ☑️ 📦 Use correct response types.
- ☑️ 🍪 Handle cookies with security flags (`HttpOnly`, `Secure`).
- ☑️ 📨 Use CORS to secure frontend-backend interaction.
- ☑️ 📝 Validate all form and query data.

🌐 HTTP Status Codes Cheat Sheet with FastAPI 💡

🟢 1xx — **INFORMATIONAL RESPONSES**

> ☑️ Request received, continuing process.

Code	Meaning	Description
100	Continue	Client should continue the request.
101	Switching Protocols	Server agrees to change protocols.
102	Processing (WebDAV)	Server has accepted request but not completed it.

🧠 **FastAPI Tip**: Rarely used directly.

🟢 2xx — **SUCCESSFUL RESPONSES**

> 🎉 The request was successfully received, understood, and accepted.

Code	Meaning	Description
200	OK	Standard success for GET/PUT/DELETE.
201	Created	Used after successful POST (e.g., resource created).
202	Accepted	Request accepted but not completed yet.
204	No Content	Success, but no response body (e.g., DELETE).

📦 **FastAPI Usage Example**:

```python
from fastapi import status

@app.post("/user", status_code=status.HTTP_201_CREATED)
def create_user():
    return {"message": "User created ☑"}
```

🟢 3xx — REDIRECTION MESSAGES

> 🔁 The client must take additional action.

Code	Meaning	Description
301	Moved Permanently	Resource has a new URL. Update your links!
302	Found (Temporary)	Temporarily moved to another URI.
304	Not Modified	Cached version can be used.
307	Temporary Redirect	Like 302 but preserves method (POST/GET).
308	Permanent Redirect	Like 301 but with method preservation.

📦 **FastAPI Example**:

```python
from fastapi.responses import RedirectResponse

@app.get("/old-link")
def old_link():
    return RedirectResponse(url="/new-link", status_code=307)
```

🟢 4xx — CLIENT ERROR RESPONSES

> 🚫 The request contains bad syntax or cannot be fulfilled.

Code	Meaning	Description
400	Bad Request	Client error (validation, format, etc).
401	Unauthorized	Authentication required.
403	Forbidden	Authenticated but not allowed.
404	Not Found	Resource doesn't exist.
405	Method Not Allowed	Method (GET/POST/etc) not supported.
409	Conflict	Resource conflict (duplicate user, etc).
422	Unprocessable Entity	Validation failed (e.g., FastAPI Pydantic errors).

Code	Meaning	Description
429	Too Many Requests	Rate limit hit.

FastAPI Example (422 Validation):

```
from fastapi import HTTPException

@app.post("/login")
def login(username: str):
    if username == "":
        raise HTTPException(status_code=400, detail="Username is required 🚫")
```

5xx — **SERVER ERROR RESPONSES**

> The server failed to fulfill a valid request.

Code	Meaning	Description
500	Internal Server Error	Generic server error.
501	Not Implemented	Server doesn't support the requested feature.
502	Bad Gateway	Invalid response from upstream server.
503	Service Unavailable	Server is down or overloaded.
504	Gateway Timeout	Upstream server timed out.

FastAPI Example (500 Error):

```
@app.get("/crash")
def cause_crash():
    raise Exception("💥 Boom! Something broke.")
```

Frequently Used HTTP Status Codes (Dev Cheatsheet)

Type	Codes	Description
☑ Success	200, 201, 204	Standard API success responses.
❌ Client	400, 401, 403, 404, 422	Validation and auth errors.
🔁 Redirect	301, 302, 307, 308	Redirection flows.
🧯 Server	500, 503	Server crash or overload.

FastAPI Usage Summary

☑ You can use **status codes** directly via:

```
from fastapi import status

@app.post("/resource", status_code=status.HTTP_201_CREATED)
def create():
    return {"msg": "Created"}
```

Or set manually in `JSONResponse`:

```
from fastapi.responses import JSONResponse

return JSONResponse(
    status_code=404,
    content={"success": False, "message": "Not Found"}
)
```

☑ Best Practices Summary

- 🎯 Always return proper status codes based on result.
- 🔒 Use 401 for auth and 403 for access control.
- 🧾 Use 422 for form/Pydantic validation errors.
- 💬 Combine with `ApiResponse` or `ApiException` classes for consistent shape.
- 🚨 Never expose stack traces or sensitive data in error responses in production.
