Error Handling

🎯 Goal

We'll implement these equivalents:

Express Component	FastAPI Equivalent
`ApiError.js`	✅ `api_exceptions.py` (custom exception class)
`ApiResponse.js`	✅ `api_response.py` (standard response wrapper)
`asyncHandler.js`	✅ FastAPI handles async natively, but we can use `try-except` decorators for reuse

---

📁 Folder Structure (Suggested)

```
app/
├── main.py
├── routes/
│   └── user.py
├── core/
│   ├── api_exceptions.py
│   ├── api_response.py
│   └── error_handler.py
```

📦 1. `api_exceptions.py` (🌋 Equivalent of ApiError.js)

```
# core/api_exceptions.py

class ApiException(Exception):
    def __init__(self, status_code=500, message="Something went wrong",
errors=None):
        self.status_code = status_code
        self.message = message
        self.success = False
        self.errors = errors or []
        super().__init__(self.message)
```

📦 2. `api_response.py` (📦 Equivalent of ApiResponse.js)

```python
# core/api_response.py

from fastapi.responses import JSONResponse

class ApiResponse:
    def __init__(self, data=None, message="Success", status_code=200):
        self.status_code = status_code
        self.message = message
        self.data = data
        self.success = status_code < 400

    def send(self):
        return JSONResponse(
            status_code=self.status_code,
            content={
                "success": self.success,
                "message": self.message,
                "data": self.data
            }
        )
```

🧤 3. Global Error Handler (🧯 Equivalent of Express `.use(errorHandler)`)

```python
# core/error_handler.py

from fastapi import Request
from fastapi.responses import JSONResponse
from core.api_exceptions import ApiException
from fastapi.exceptions import RequestValidationError
from starlette.status import HTTP_500_INTERNAL_SERVER_ERROR

def register_exception_handlers(app):

    @app.exception_handler(ApiException)
    async def api_exception_handler(request: Request, exc: ApiException):
        return JSONResponse(
            status_code=exc.status_code,
            content={
                "success": False,
                "message": exc.message,
                "errors": exc.errors,
                "data": None
            }
        )

    @app.exception_handler(RequestValidationError)
    async def validation_exception_handler(request: Request, exc: RequestValidationError):
        return JSONResponse(
            status_code=422,
            content={
                "success": False,
                "message": "Validation Error",
                "errors": exc.errors(),
                "data": None
            }
        )

    @app.exception_handler(Exception)
    async def general_exception_handler(request: Request, exc: Exception):
        return JSONResponse(
            status_code=HTTP_500_INTERNAL_SERVER_ERROR,
            content={
                "success": False,
                "message": str(exc),
                "errors": [],
                "data": None
            }
        )
```

⚙️ 4. `main.py` – Tie It All Together

```
# main.py

from fastapi import FastAPI
from core.error_handler import register_exception_handlers
from routes import user

app = FastAPI()

# Register global exception handlers
register_exception_handlers(app)

# Include routes
app.include_router(user.router)
```

🧪 5. Sample Route with Custom Response & Error

```
# routes/user.py

from fastapi import APIRouter
from core.api_response import ApiResponse
from core.api_exceptions import ApiException

router = APIRouter(
    prefix="/user",
    tags=["User"]
)

@router.get("/profile")
async def get_user():
    # Simulate a condition
    raise ApiException(status_code=404, message="User not found 🕵️")

@router.get("/welcome")
async def welcome_user():
    response = ApiResponse(data={"name": "Darshan"}, message="Welcome 🎉")
    return response.send()
```

✅ Final API Response Examples

✅ Success Response

```
{
  "success": true,
  "message": "Welcome 🎉",
  "data": {
    "name": "Darshan"
  }
}
```

❌ Error Response (Custom)

```
{
  "success": false,
  "message": "User not found 🕵️",
  "errors": [],
  "data": null
}
```

✅ Advantages of This Setup

🔁 **Reusable** 📦 **Consistent Response Shape** 🧯 **Centralized Error Handling** 🧪 **Test Friendly & Scalable** 💻 **Developer-Friendly Debugging**

---

async/await natively. Unlike Express.js where you need `asyncHandler()` to catch errors in async functions (since unhandled promise rejections can crash the app), FastAPI's internal engine (Starlette + ASGI) **already handles async errors properly**.

So, technically, you **don't need an `asyncHandler` like in Express**.

☑️ But... What if you want **middleware-like async wrappers**?

You **can** implement a reusable `async_handler` decorator in FastAPI for:

- Logging errors 🌐
- Converting raw exceptions into your custom `ApiException` 🚨
- Centralizing error wrapping across multiple endpoints 📦

⚙️ Create `async_handler` Decorator (Optional)

```python
# core/async_handler.py

from functools import wraps
from core.api_exceptions import ApiException

def async_handler(func):
    @wraps(func)
    async def wrapper(*args, **kwargs):
        try:
            return await func(*args, **kwargs)
        except ApiException as ae:
            raise ae  # Let FastAPI handle this via your global handler
        except Exception as e:
            # Convert unhandled errors into your custom ApiException
            raise ApiException(status_code=500, message=str(e))
    return wrapper
```

🧪 Use It in Routes (Optional)

```python
# routes/user.py

from fastapi import APIRouter
from core.api_response import ApiResponse
from core.api_exceptions import ApiException
from core.async_handler import async_handler

router = APIRouter(prefix="/user", tags=["User"])

@router.get("/profile")
@async_handler
async def get_user():
    # Simulating error
    raise ApiException(status_code=404, message="User not found 💥")

@router.get("/safe")
```

```python
@async_handler
async def safe_route():
    # Simulating unknown error
    1 / 0  # This will raise ZeroDivisionError
```

🔄 Flow with `async_handler`

```
Request ⟶ Route Handler ⟶ async_handler Decorator
                                  ⬇️
                       Try → Await function
                       Catch → Wrap unknown error in ApiException
                                  ⬇️
                      Global error_handler catches ApiException
```

✅ Summary

Use Case	Needed in FastAPI?	How to Implement
Catch async route errors	❌ Handled natively	
Uniform error wrapping	✅ Optional decorator	
Centralized error format	✅ Via global handler	