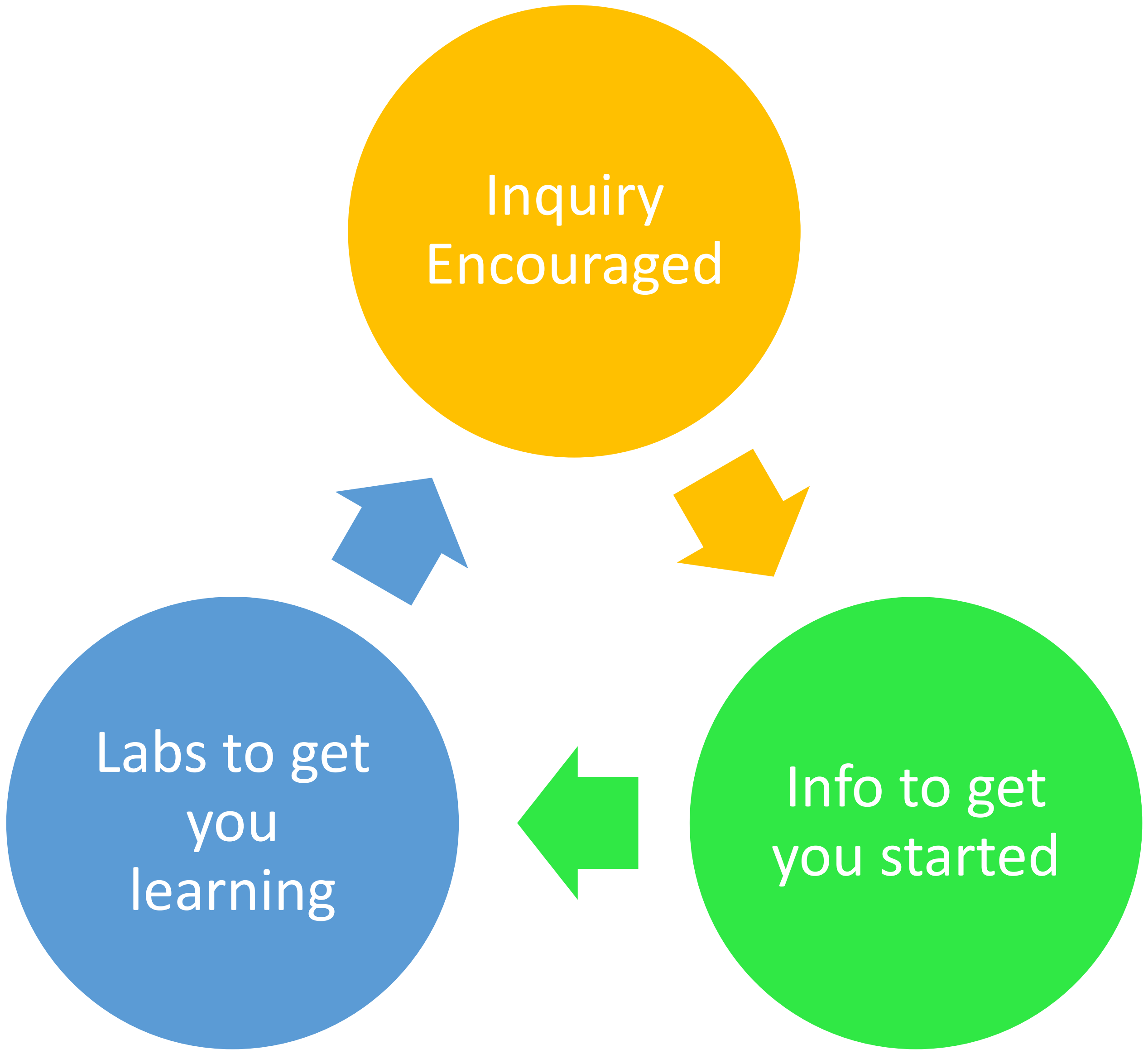
Fast API Fast

Nuri Halperin | nuri@plusnconsulting.com

CodeMash 2023

The Workshop



What Will You Need?

- Computer
- Internet Access
- For Lab:
    - Via browser on replit.com
        - Free account needed
    - OR: Local on your computer
        - Python ^3.10 + IDE

About Fast API

Created by **Sebastián Ramírez**
https://github.com/tiangolo

Debut circa 2019

Actively maintained

Well liked / starred

Fast API – Runtime Speed

Runtime Speed

- Benchmarks well
- Starlette (ASGI impl.)
- Pydantic

Fast API – Development Speed

Quick to develop

- Standing up an API is quick
- Easy to use
- “It just works” experience

Fast API – Covers your Use Case

Robust

- ASGI – Async Gateway Application Interface
- JSON Schema / OpenAPI
- Swagger OOB

Hello World

```python
from fastapi import FastAPI

api = FastAPI()


@api.get("/")
async def root():
    return "Hi."
```

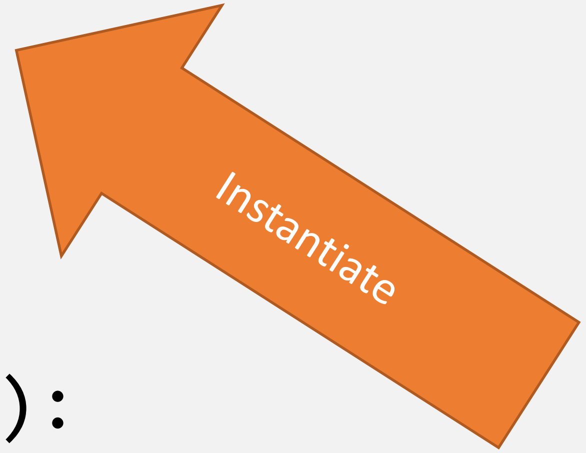
Hello World

```
from fastapi import FastAPI

api = FastAPI()


@api.get("/")
async def root():
    return "Hi."
```

The library

Hello World

```python
from fastapi import FastAPI

api = FastAPI()


@api.get("/")
async def root():
    return "Hi."
```



Hello World

Decorate

```
from fastapi import FastAPI

api = FastAPI()


@api.get("/")
async def root():
    return "Hi."
```

Hello World

```
from fastapi import FastAPI

api = FastAPI()


@api.get("/")
async def ro     :
    return "Hi.
```

Route

Hello World

```
from fastapi import FastAPI

api = FastAPI()


@api.get("/")
async def root():
    return "Hi."
```

Async! (ASGI…)

Function Name is arbitrary

Demo

Minimal hello world

What is Fast API

IS

- A web API framework
- Config + Decorators

Is Not

- A web server
- ORM / ODM

- Uvicorn
- Pydantic
- Starlette
- BYO-DB

- BYO-DB

Practice Time!

Lab activities ***M1 – Getting Started***

URL Path Parameters

- Path parameters
- Catch-all parameter

Path Parameters

```
from fastapi import FastAPI

api = FastAPI()


@api.get("/even/{number}")
async def get_even(number):
    return {
        "number": number,
        'ok': number % 2 == 0
```

Path Parameters - Typing

```
from fastapi import FastAPI

api = FastAPI()


@api.get("/even/{number}")
async def get_even(number: int):
    return {
        "number": number,
        'ok': number % 2 == 0
```

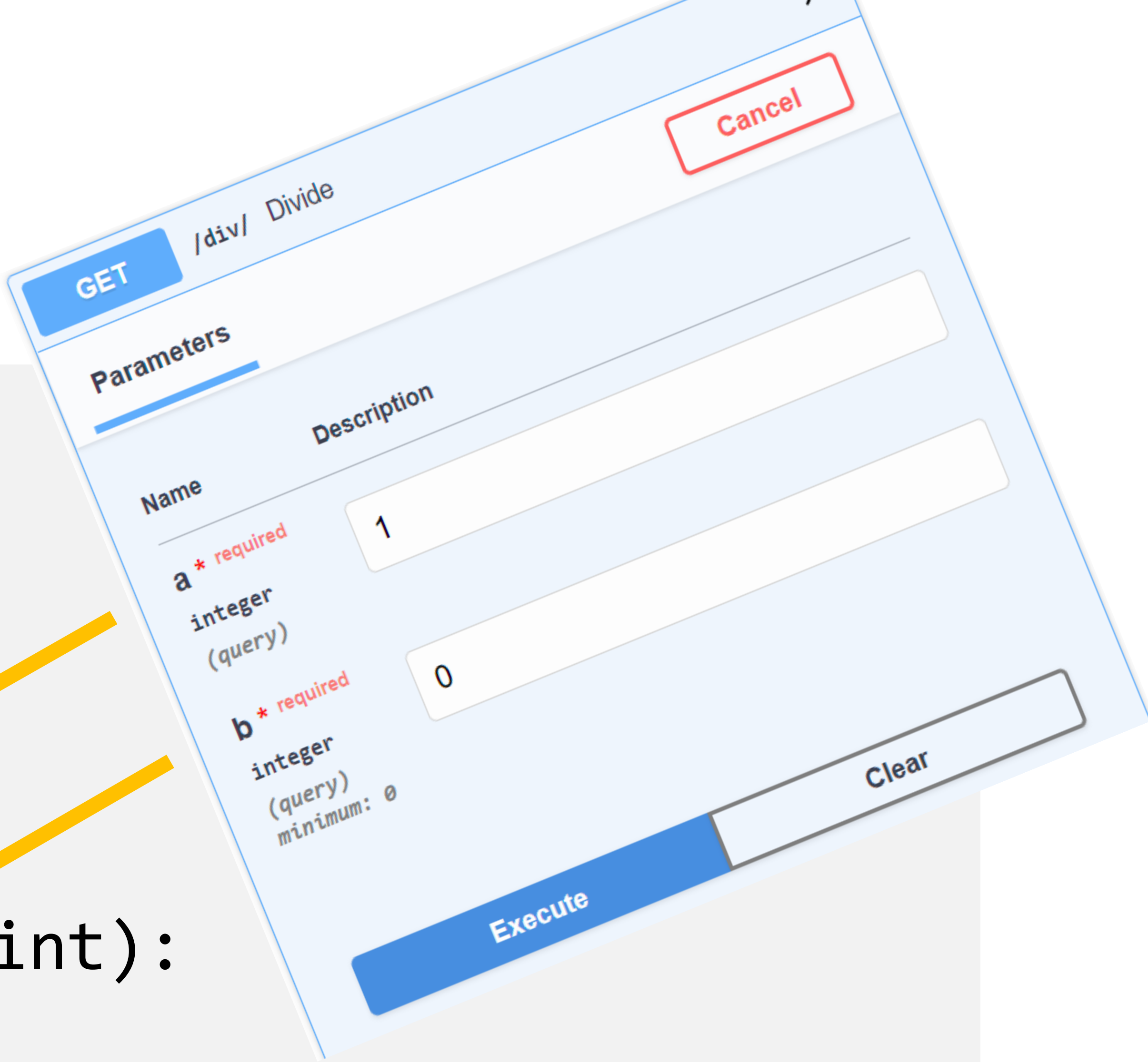
Typing of parameters:
- Type conversion
- Type validation

Path Parameters - Multiple

```python
from fastapi import FastAPI

api = FastAPI()


@api.get("/add/{a}/{b}")
async def add(a: int, b:int):
    return {
        "a": a,
        "b": b,
        "sum": a + b
    }
```

Query Parameters

```python
from fastapi import FastAPI

api = FastAPI()


@api.get("/div")
async def divide(a: int, b: int):
    return a*b
```

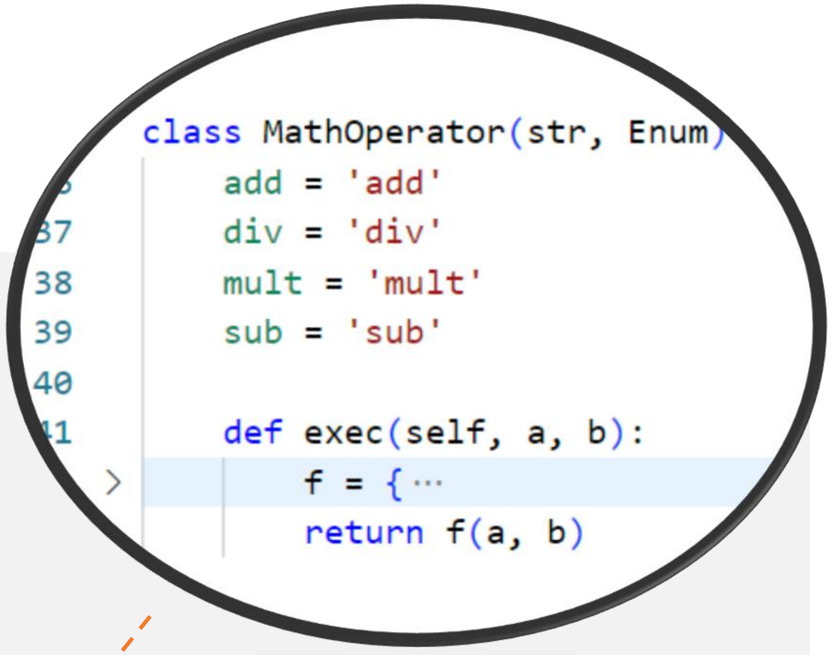


Query Parameters Validation

```
from fastapi import FastAPI, Query

api = FastAPI()


@api.get("/div/")
async def divide(a:int,
        b: int = Query(title="divisor", gt=0)) -> float:
    return {'result': a/b}
```

Other Parameter Types

```
from fastapi import FastAPI, Query

api = FastAPI()




@api.get("/calc/{op}")
async def calc(a: int, b: int, op: MathOperator):
    return {'op': 'op',  'result': op.exec(a, b)}
```



```
class MathOperator(str, Enum):
    add = 'add'
    div = 'div'
    mult = 'mult'
    sub = 'sub'

    def exec(self, a, b):
        f = {···
        return f(a, b)
```

Demo

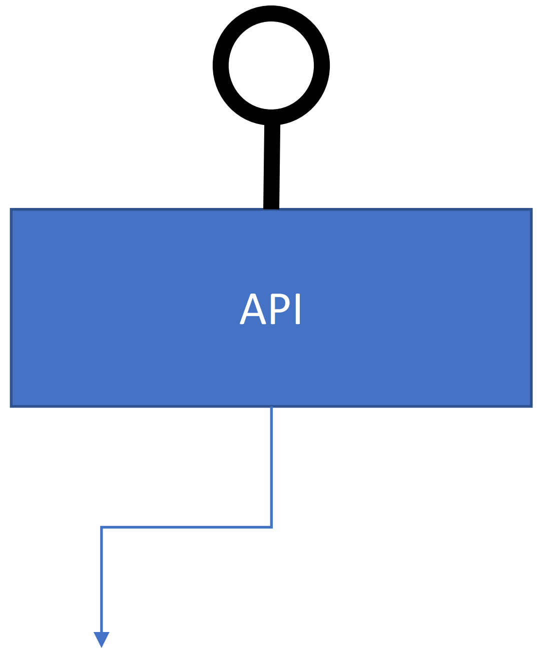
Path Parameters

Complex Schemas

- pydantic models as parameters
- Annotation / shaping
- Validation

The “why” of schemas in API

- Shape input parameters
- Constrain acceptable field values
- Validate output messages
- Communicate it all - via OpenAPI



The “how” of schemas in Fast API

```python
from pydantic import BaseModel

class Review(BaseModel):
    name: str
    stars: int

@api.post("/reviews")
async def create_review(review: Review):
     return heroes
```

Define

The “how” of schemas in Fast API

```python
from pydantic import BaseModel

class Review(BaseModel):
    name: str
    stars: int

@api.post("/reviews")
async def create_review(review: Review):
     return heroes
```

Use

The “how” of schemas in Fast API

```
from pydantic import BaseModel

class Review(BaseModel):
    name: str
    stars: int

@api.post("/reviews")
async def create_review(review: Review):
     return heroes
```

Import, inherit
BaseModel

More Control of Schema Fields

```python
from pydantic import BaseModel, Field

class ReviewSummary(BaseModel):
    subject: str = Field(title="Rated Item")
    author: str = Field(alias="Reviewer")
    stars: float = Field(gt=0, le=5)
```

Field() Reference Guide

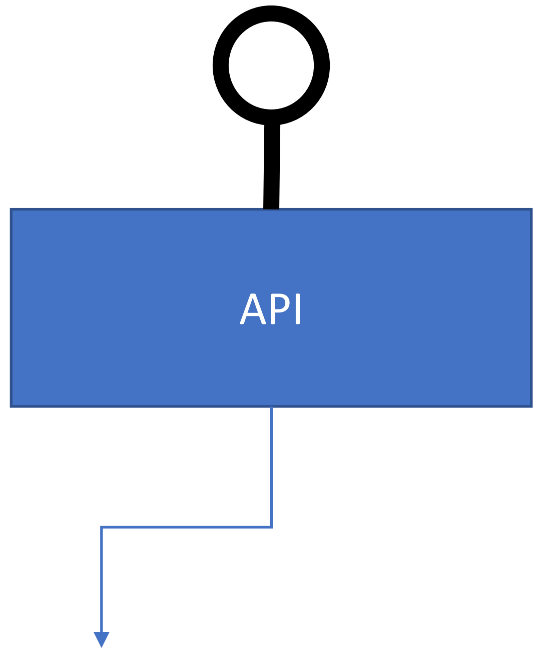
Parameters	More parameters	Even more parameters
default	gt	unique_items
default_factory	ge	min_length
alias	lt	max_length
title	le	allow_mutation
description	multiple_of	regex
exclude	max_digits	discriminator
include	decimal_places	repr
const	min_items	
	max_items	

Open API

- It just works ™
- Enriching
- Controlling Swagger / reDoc

Open API

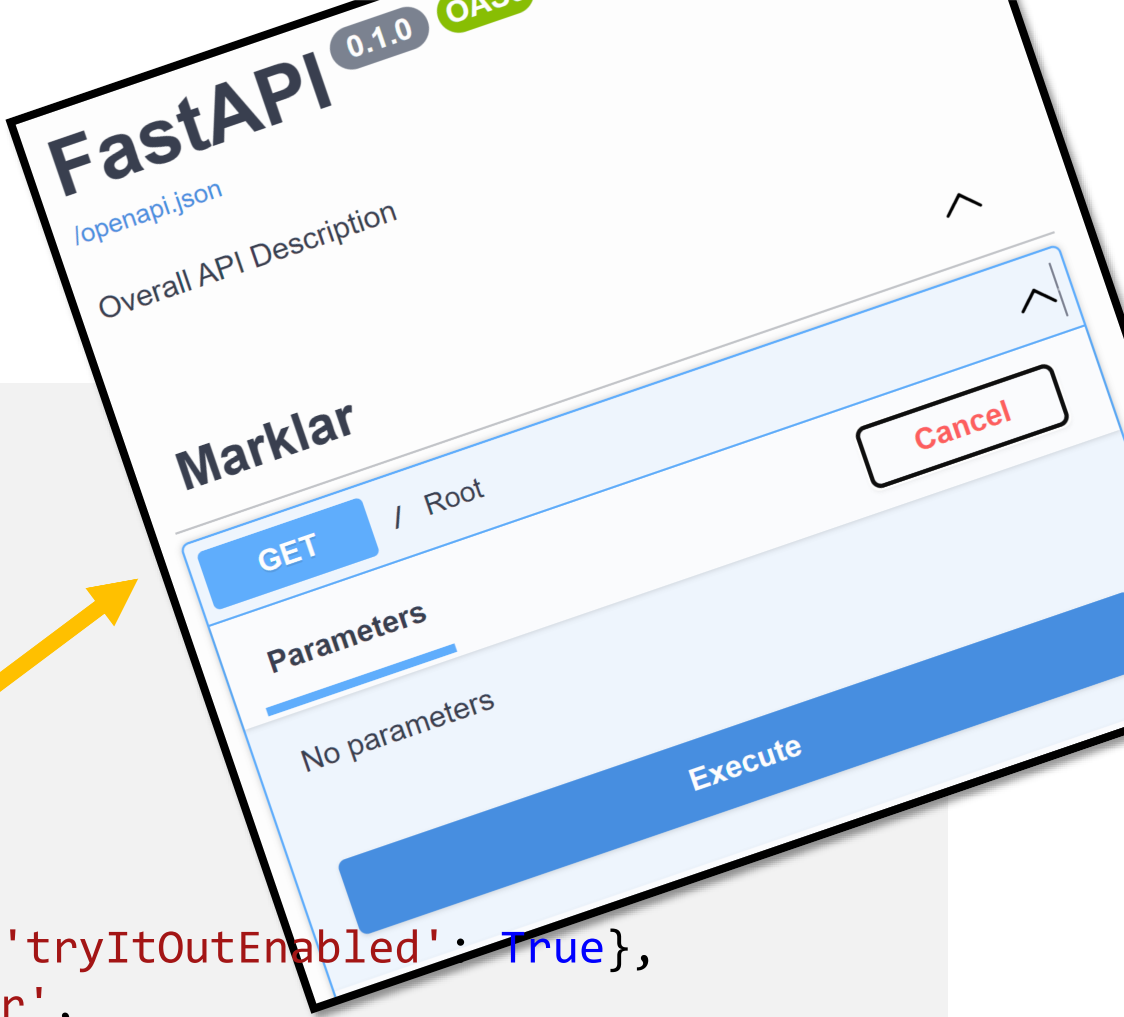
- A spec / standard
- Machine discovery
- Human discovery
  - With tools: Swagger, reDoc…



OpenAPI + Fast API

```
# Global

## Model

### Field

## Route
```

```
api = FastAPI(
    swagger_ui_parameters={'tryItOutEnabled': True},
    docs_url='/_/doc/swagger',
    description="Overall API Description"
)
```



OpenAPI + Fast API

```
# Global

## Model

### Field

## Route
```

```python
class Demo(BaseModel):
        """
        Docstrings describe model-level
        """
    name: str = Field(
        title='My field title',
        description='My field description')
```

OpenAPI + Fast API

```
# Global

## Model

### Field

## Route
```

```python
class Demo(BaseModel):
            """
            Docstrings describe model-level
            """
        name: str = Field(
            title='My field title',
            description='My field description')
```

OpenAPI + Fast API

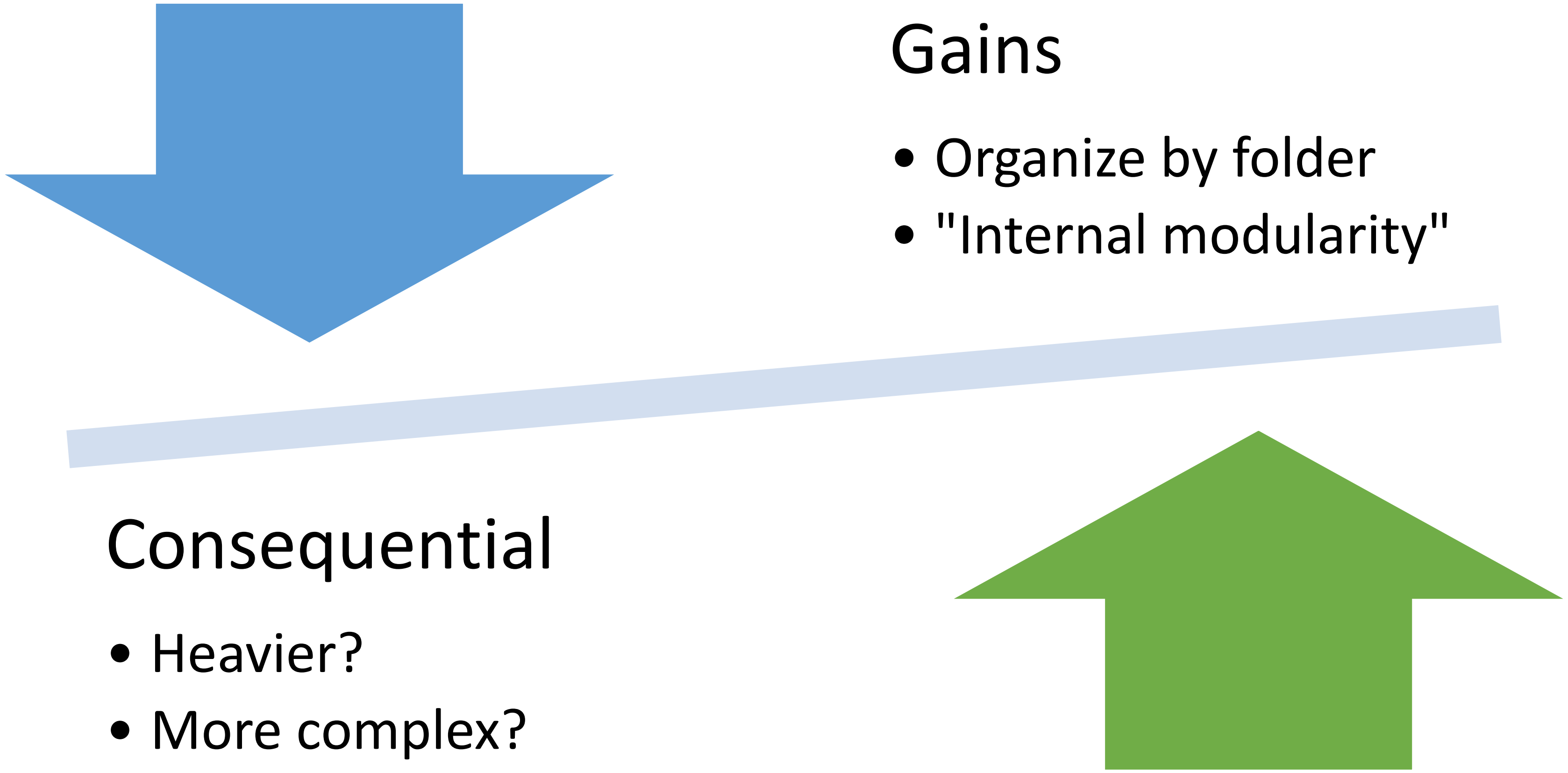
```
# Global

## Model

### Field

## Route
```

```
@api.get('/customers/{id}',
         summary='Get Customer',
         description='''
         Get a customer by the given id.

         More **things** to say here...
         ''')
```

Bigger Applications

- Modularize
- Router annotations
- include_router

Why Bigger



Gains

- Organize by folder
- "Internal modularity"

Consequential

- Heavier?
- More complex?

Bigger App File Layout

```
src/
├── __init__.py
├── subdomain_one
│   ├── __init__.py
│   └── main.py
├── subdomain_two
│   ├── __init__.py
│   └── main.py
├── common
│   ├── __init__.py
│   └── routing.py
└── main.py
```

BYO structure…

The APIRouter Exposes HTTP Verbs

```
src/
├── __init__.py
├── subdomain_one
│   ├── __init__.py
│   └── main.py
├── subdomain_two
│   ├── __init__.py
│   └── main.py
├── common
│   ├── __init__.py
│   └── routing.py
└── main.py
```

```python
from fastapi import APIRouter

router = APIRouter(
        prefix=f"/subdomain_one",
        tags=[tag]
    )

@router.get("/")
async def function_one():
    return {"message": "Hello One!"}
```

Consume Routes from Root Main

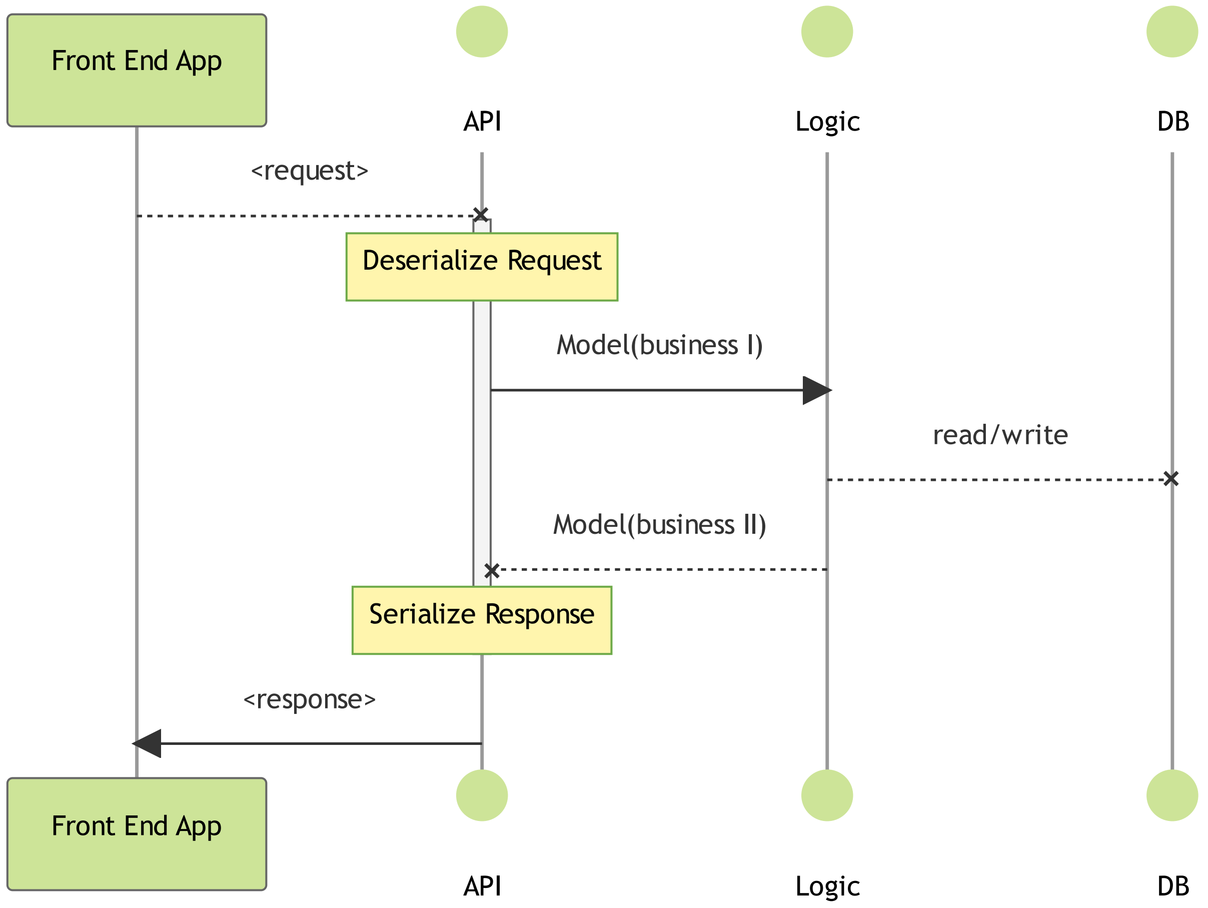
```
src/
├── __init__.py
├── subdomain_one
│   ├── __init__.py
│   └── main.py
├── subdomain_two
│   ├── __init__.py
│   └── main.py
├── common
│   ├── __init__.py
│   └── routing.py
└── main.py
```

```python
import subdomain_one.main as d1
import subdomain_two.main as d2


api = FastAPI()

api.include_router(d1.router)
api.include_router(d1.router)
```

Database

Adding a Data Layer with MongoDB

Pydantic for Everything?



MongoDB as Backend

- Document Oriented
- Solid
- Scalable
- Feature Rich

```python
import motor.motor_asyncio
# from pymongo import MongoClient
```

- Use the synchronous **MongoClient** from **pymongo**
- Use the async **AsyncIOMotorClient** from **motor**

MongoDB Setup

```
from pymongo import MongoClient
from fastapi import FastAPI

api = FastAPI()

client = MongoClient('mongodb://localhost/test')
db = client.get_default_database('test')
collection = db['fastapi']
```

Documents Need _id

```python
class Customer(BaseModel):
    email: str
    name: str
```

```python
@api.post('/customer')
async def create(customer:Customer):
    doc = customer.dict()
    doc['_id'] = doc.pop('email')
    collection.insert_one(doc)
```

Pydantic Has No _id

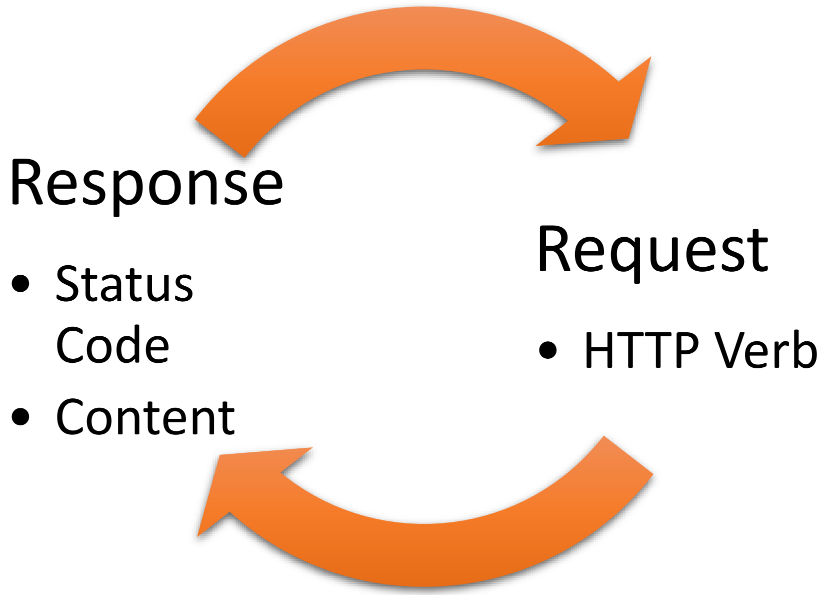
```
class Customer(BaseModel):
    email: str
    name: str
```

```

@api.get('/customer/{email}',response_model=Customer)async
def get_one(email: str):
    doc = collection.find_one({'_id': email})
    doc['email'] = doc.pop('_id')
    return Customer(**doc)
```

Responding & Error Handling

- HTTP Responses
- HTTP Error
- Middleware / Custom

HTTP Conversation



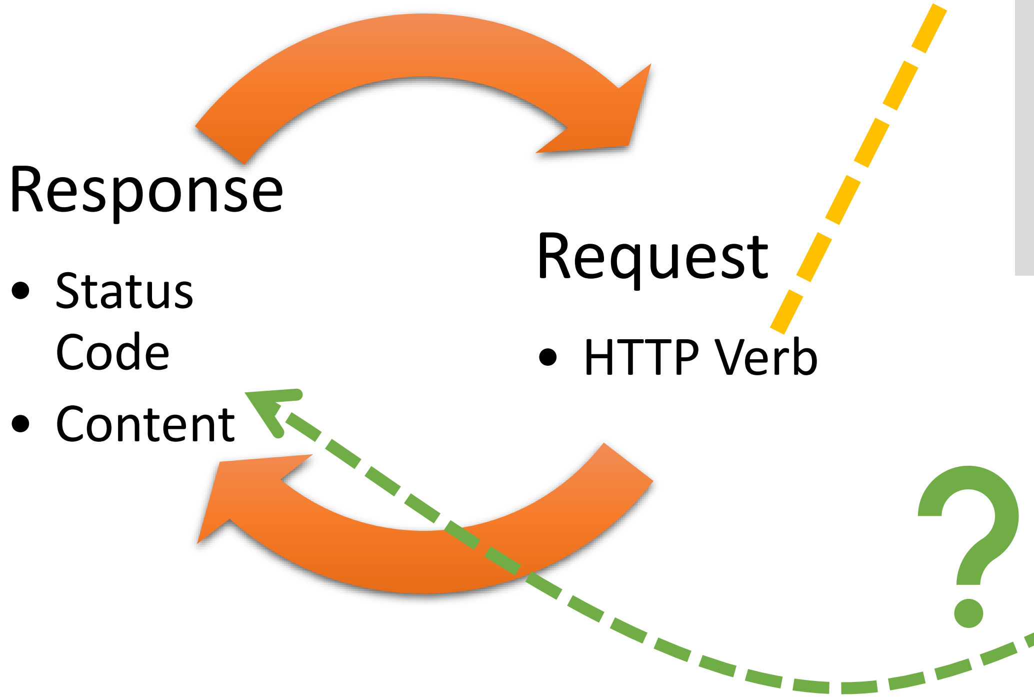
Response

- Status Code
- Content

Request

- HTTP Verb

HTTP Conversation



Response

- Status Code
- Content

Request

- HTTP Verb

```
@api.post('/blah')
def create(t:Thing):
        # …
        return ThingData(…)
```

HTTP Responses

- Response base class
- Exposes:
  - Status Code
  - Content : Any
  - Headers
  - Media Type
  - Charset

```
from fastapi import status
from fastapi.responses import(
    JSONResponse,
    FileResponse,
    HTMLResponse,
    RedirectResponse
    )

## Constants exist:
# status.HTTP_200_OK = 200
```

Errors

- Built ins
  - Request validation: 400's
  - Response / processing: 500's
  - Return `RedirectResponse()`: 307
- Your own:
  - Throw HTTPException()
  - Add your own via `FastAPI::add_exception_handler(…)`

Demo

Responding with different responses

Raising HTTPException

Exception Handling

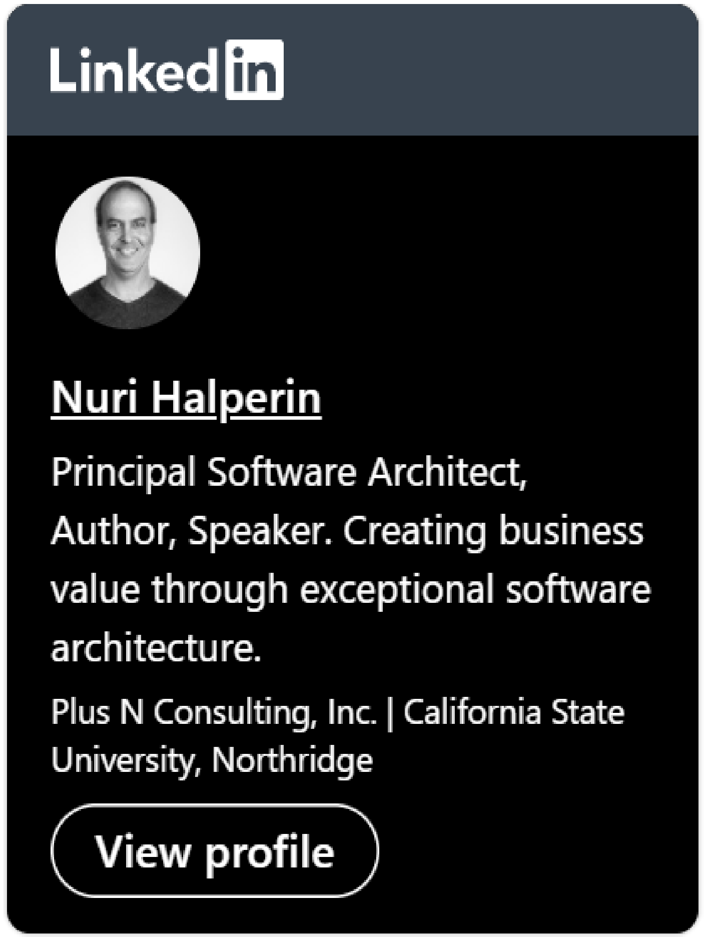
Thank You!

- Q&A

About the Instructor

- Startups … Enterprise
- Back-end
- Databases: RDBMS, MongoDB, Other.
- Monolith <==> Microservices
- On Prem, Cloud, Hybrid
- JavaScript, Python, C#, Java

LinkedIn: linkedin.com/in/**nurih**
http://**plusnconsulting.com**


LinkedIn

Nuri Halperin

Principal Software Architect, Author, Speaker. Creating business value through exceptional software architecture.

Plus N Consulting, Inc. | California State University, Northridge