Easy service to service communication using Pydantic models

Michal Korbela

Michal Korbela

Staff engineer @ Ancillaries team, Kiwi.com

michal.korbela@kiwi.com

https://github.com/kabell

? >) 1 [[
@ ! 0

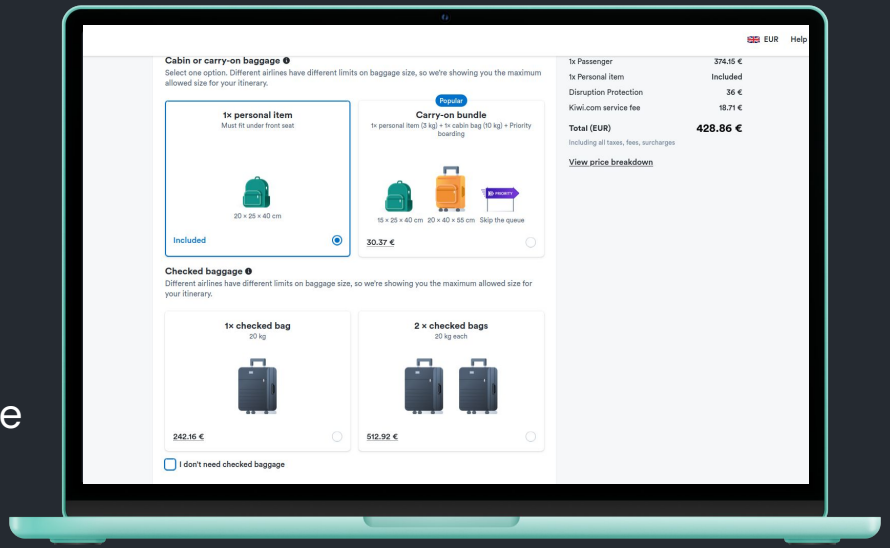
Follow our tech content
and events at **code.kiwi.com**

Open roles at **jobs.kiwi.com**

0 !! 1
<< % :: 1 @ 0 ? >

What we do

- provide baggage, seating and other ancillaries
- support 1000+ carriers
- used monolithic architecture
- provide seating for 10s itineraries/s
- provide baggage for 1M itineraries/s
- switched to microservices architecture



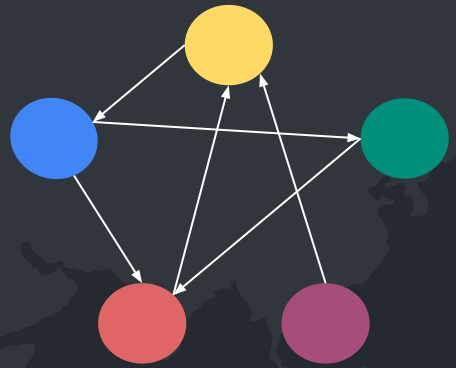
Monolithic architecture
vs
Microservices architecture

Monolithic vs Microservices architecture

Monolithic architecture

- Difficult to change dependency
- Difficult to scale
- One module can crash the whole app

Microservices architecture



- Communication complexity

Monolithic vs Microservices architecture

Monolithic Architecture

```python
from pricing import compute_bag_price

def get_bag_price(bag: Bag) -> Price:
    return compute_bag_price(bag)
```

Microservices Architecture

Client

```python
def get_bag_price(bag: dict) -> dict:
    return requests.post(url, json=bag).json()
```

Flask Server

```python
@app.post("/price_bag")
def price_bag() -> dict:
    return compute_bag_price(request.json)
```

Monolithic vs Microservices architecture

Microservices Architecture

Client

```
def get_bag_price(bag: dict) -> dict:
        return requests.post(url, json=bag).json()
```

Flask Server

```
@app.post("/price_bag")
def price_bag() -> dict:
        return compute_bag_price(request.json)
```

Monolithic vs Microservices architecture

Microservices Architecture

Client

```python
def get_bag_price(bag: Bag) -> Price:
        payload = {"weight": bag.weight}

        resp = requests.post(url, json=payload).json()

        if not isinstance(resp.get("amount"),float):
                raise ValueError("amount is not float")

        return Price(amount=response["amount"])
```

- Use models instead of dicts
- Validate requests/responses
- DRY principle !!!

Flask Server

```python
@app.post("/price_bag")
def price_bag() -> dict:
        """Compute price of a bag.

        Request:
        - weight: int - Weight of the bag
        Response
        - amount: float - Price of the bag in EUR
        """
        if not isinstance(request.json.get("weight"),int):
                return "weight is not int", 400

        bag = Bag(weight=request.json["weight"])

        price = compute_bag_price(bag)

        return {
                "amount": price.amount
        }
```

Tools & approaches

Tools & approaches evaluation

- Duplication level
  - Request
  - Response

Comparison

Approach	Request duplication level		Response duplication level	
	Client	Server	Client	Server
No tools	1	3	2	2

? >) 1 [[

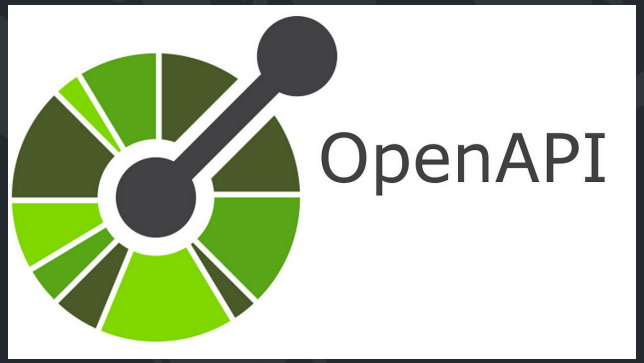
@ ! 0

OpenAPI

0 !! 1

<< % :: 1 @ 0 ? >

OpenAPI



- Known as Swagger
- A standard for describing and documenting RESTful APIs
- Language agnostic
- Uses YAML or JSON

```yaml
openapi: 3.1.0
info:
  version: 1.0.0
  title: Pricing API
  description: A simple Pricing API
paths:
  /price_bag:
    post:
      description: Returns a price for a given bag
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                weight:
                  type: integer
                  description: Weight of the bag
      responses:
        '200':
          description: Successful response
          content:
            application/json:
              schema:
                type: object
                properties:
                  amount:
                    type: number
                    description: Price of the bag in EUR
```

Swagger Viewer

http://localhost:8000/ui/

? >) 1 [[

Pricing API 1.0.0 OAS 3.0

A simple Pricing API

default

POST /price_bag

Returns a price for a given bag

Parameters Try it out

No parameters

Request body required application/json

Example Value | Schema

```
{
  "weight": 0
}
```

Responses

Code	Description	Links
200	Successful response	*No links*

Media type: application/json

Controls Accept header.

Example Value | Schema

```
{
  "amount": 0
}
```

0 !! 1

<< % :: 1 @ 0 ? >

? >) 1 [[

@ ! 0

Connexion

0 !! 1

<< % :: 1 @ 0 ? >

Flask + Connexion

CONNEXION
API-FIRST FOR ALL

app.py

```python
import connexion

app = connexion.FlaskApp(__name__)
app.add_api("openapi.yaml")
```

openapi.yaml

```yaml
openapi: 3.1.0
info:
  version: 1.0.0
  title: Pricing API
  description: A simple Pricing API
paths:
  /price_bag:
    post:

      description: Returns a price for a given bag
       requestBody:
         required: true
         content:
          application/json:
             schema:
               type: object
               properties:
                 weight:
                     type: integer
                     description: Weight of the bag
      responses:
        '200':
          description: Successful response
           content:
            application/json:
              schema:
                type: object
                properties:
                  amount:
                     type: number
                     description: Price of the bag in EUR
```

Comparison

Approach	Request duplication level		Response duplication level	
	Client	Server	Client	Server
No tools	1	3	2	2
Connexion	1	2	2	2

? >) 1 [[

@ ! 0

FastAPI

0 !! 1

<< % :: 1 @ 0 ? >

FastAPI

app.py

```python
from fastapi import FastAPI
from pydantic import BaseModel, Field

class Bag(BaseModel):
    weight: int = Field(..., description="Weight of the bag")

class Price(BaseModel):
    amount: float = Field(..., description="Price in EUR")

app = FastAPI(title="Pricing API")

@app.get("/price_bag")
def get_bag_price(bag: Bag) -> Price:
    return compute_bag_price(bag)
```

Models used:

- Parsing
- Validation
- Serialization
- OpenAPI schema generation

Comparison

Approach	Request duplication level		Response duplication level	
	Client	Server	Client	Server
No tools	1	3	2	2
Connexion	1	2	2	2
FastAPI	1	1	2	1

Client side optimizations

Use Pydantic for Bag and Price objects

Original

```python
def get_bag_price(bag: Bag) -> Price:
      payload = {"weight": bag.weight}

      resp = requests.post(url, json=payload).json()

      if not isinstance(resp.get("amount"),float):
            raise ValueError("amount is not float")

      return Price(amount=response["amount"])
```

With Pydantic

```python
class Bag(BaseModel):
   weight: int = Field(..., description="Weight of the bag")

class Price(BaseModel):
   amount: float = Field(..., description="Price in EUR")

def get_bag_price(bag: Bag) -> Price:
   payload = bag.model_dump(mode="json")
   response = requests.post(url, json=payload).json()
   return Price.model_validate(response)
```

Comparison

Approach	Request duplication level		Response duplication level	
	Client	Server	Client	Server
No tools	1	3	2	2
Connexion	1	2	2	2
FastAPI	1	1	2	1
Use Pydantic in clients (with FastAPI)	1	1	1	1

Sharing Pydantic models

Client

```python
class Bag(BaseModel):
   weight: int = Field(..., description="Weight of the bag")

class Price(BaseModel):
   amount: float = Field(..., description="Price in EUR")

def get_bag_price(bag: Bag) -> Price:
   payload = bag.model_dump(mode="json")
   response = requests.post(url, json=payload).json()
   return Price.model_validate(response)
```

FastAPI server

```python
from fastapi import FastAPI
from pydantic import BaseModel, Field

class Bag(BaseModel):
   weight: int = Field(..., description="Weight of the bag")

class Price(BaseModel):
   amount: float = Field(..., description="Price in EUR")

app = FastAPI(title="Pricing API")

@app.get("/price_bag")
def get_bag_price(bag: Bag) -> Price:
   return compute_bag_price(bag)
```

Shared Python package

- Extract common Pydantic models into a separated python package

Client

```python
from schemas_package.bag_price import Bag, Price

def get_bag_price(bag: Bag) -> Price:
    payload = bag.model_dump(mode="json")
    response = requests.post(url, json=payload).json()
    return Price.model_validate(response)
```

FastAPI server

```python
from fastapi import FastAPI
from schemas_package.bag_price import Bag, Price

app = FastAPI(title="Pricing API")

@app.get("/price_bag")
def get_bag_price(bag: Bag) -> Price:
    return compute_bag_price(bag)
```

Schemas package

bag_price.py

```python
from pydantic import BaseModel, Field

class Bag(BaseModel):
    weight: int = Field(..., description="Weight of the bag")

class Price(BaseModel):
    amount: float = Field(..., description="Price in EUR")
```

Comparison

Approach	Request duplication level		Response duplication level	
	Client	Server	Client	Server
No tools	1	3	2	2
Connexion	1	2	2	2
FastAPI	1	1	2	1
Use Pydantic in clients (with FastAPI)	1	1	1	1
Shared models package (with FastAPI)	1		1	

? >) 1 [[

@ ! 0

Generating Models

0 !! 1

<< % :: 1 @ 0 ? >

Generating Pydantic models from OpenAPI

Generating Pydantic models from OpenAPI

Datamodel-code-generator - https://github.com/koxudaxi/datamodel-code-generator/

FastAPI server

```python
from fastapi import FastAPI
from pydantic import BaseModel

class Bag(BaseModel):
   weight: int = Field(..., description="Weight of the bag")

class Price(BaseModel):
   amount: float = Field(..., description="Price in EUR")

app = FastAPI(title="Pricing API")

@app.get("/price_bag")
def get_bag_price(bag: Bag) -> Price:
   return compute_bag_price(bag)
```

Generated models

```python
# generated by datamodel-codegen:
#   filename:  openapi.json
#   timestamp: 2024-10-21T12:47:11+00:00

from __future__ import annotations

from typing import List, Optional, Union

from pydantic import BaseModel, Field

class Bag(BaseModel):
   weight: int = Field(..., description="Weight of the bag")

class Price(BaseModel):
   amount: float = Field(..., description="Price in EUR")
```

? >) 1 [[

@ ! 0

Django & Flask

0 !! 1

<< % :: 1 @ 0 ? >

Django ninja

- Create django endpoints in the FastAPI way
- Automatic validation and docs generation

urls.py

```python
from django.contrib import admin
from django.urls import path
from ninja import NinjaAPI

api = NinjaAPI()


@api.get("/price_bag")
def get_bag_price(bag: Bag) -> Price:
    return compute_bag_price(bag)


urlpatterns = [
    path("admin/", admin.site.urls),
    path("api/", api.urls),
]
```

? >) 1 [[

@ ! 0

Flask

0 !! 1

<< % :: 1 @ 0 ? >

Flask ninja

- Create flask endpoints in the FastAPI way
- Automatic validation and docs generation

app.py

```python
from flask import Flask
from flask_ninja import NinjaAPI

app = Flask(__name__)
api = NinjaAPI(app)

@api.get("/price_bag")
def get_bag_price(bag: Bag) -> Price:
   return compute_bag_price(bag)
```

? >) 1 [[
@ ! 0

Presentation and links to presented tools

Follow our tech content and events at **code.kiwi.com**

Open roles at **jobs.kiwi.com**

0 !! 1
<< % :: 1 @ 0 ? >

? >) 1 [[

@ ! 0

Presentation and links to presented tools

Follow our tech content and events at **code.kiwi.com**

Open roles at **jobs.kiwi.com**

0 !! 1

<< % :: 1 @ 0 ? >