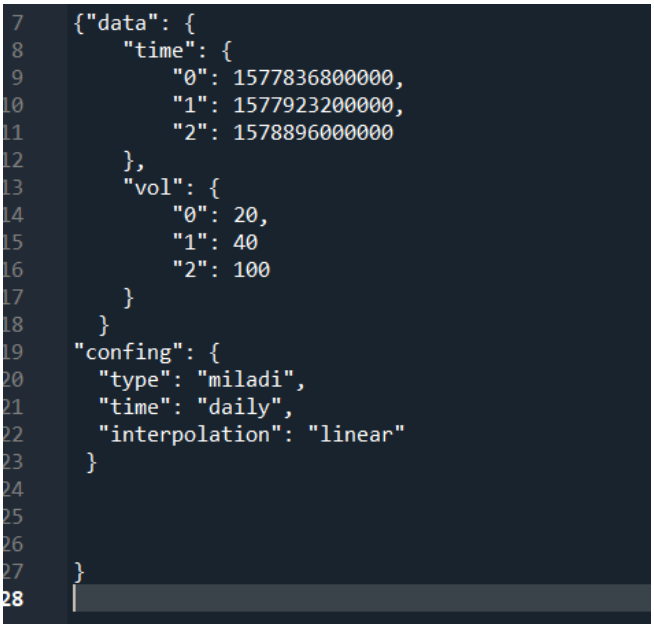
گزارش پیاده‌سازی سرویس 1

هدف این سرویس پر کردن داده‌های ناموجود در یک سری زمانی است. به این صورت که سرویس پیاده‌سازی‌شده یک json را که شامل بخش‌های زیر است، به‌عنوان **ورودی** دریافت می‌کند:



```
{"data": {
    "time": {
        "0": 1577836800000,
        "1": 1577923200000,
        "2": 1578896000000
    },
    "vol": {
        "0": 20,
        "1": 40
        "2": 100
    }
  }
"confing": {
  "type": "miladi",
  "time": "daily",
  "interpolation": "linear"
 }


}
```

افزون‌براین، برای پیاده‌سازی این سرویس، به یک فایل پایتون app، یک فایل پایتون interpolation_methods، یک فایل پایتون common، یک dockerfile، یک فایل متن requirements و یک فایل yml که در این‌جا آن را fandogh نام‌گذاری کردیم، نیاز است.

فایل json **خروجی** نیز به‌صورت زیر خواهد بود:

```
{
  "data": {
    "time": {
      "0": 1577836800000,
      "1": 1577923200000,
      "2": 1578009600000,
      "3": 1578096000000
    },
    "vol": {
      "0": 20,
      "1": 40,
      "2": 70,
      "3": 100
    }
  }
}
```

توضیحاتی درخصوص هر یک از بخش‌ها

app1:

```python
from flask import Flask, request
from utils.common import response_message, read_json_time_series
from utils.interpolation_methods import linear_interpolation
app = Flask(__name__)

@pp.route('/',methods=['GET','POST'])
def isup():
    return response_message('API is active')

@app.route('/service1',methods=['GET','POST'])
def interpolation():
    req = request.get_json()
    data = read_json_time_series(req['data'])
    config = req['config']

    if config['type'] == 'miladi':
        result = linear_interpolation(data, config)
        result = result.ro_json()
    return response_message(dict({"data" : result}))

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

در تابع interpolation نوشته‌شده، یک ریکوئست ورودی را به‌صورت json دریافت می‌شود.

Response_message را "API is active" تعریف کرده‌ایم.

در تابع interpolation، req، data و config تعریف شده‌اند و اگر config میلادی باشد، نتیجه دیتا را برمی‌گرداند.

input:

(عکس در صفحه 1)

شامل یک بخش دیتا و یک بخش از configها (که در آن type، میلادی، time، روزانه و interpolation به‌صورت خطی در نظر گرفته شده است.

:common

```python
import gzip
from flask import make_response, json
import pandas as pd
import numpy as np
import json

def response_message(data=None, status=200):
    if status in range(200, 400):
        content = gzip.compress(json.dumps(data, ensure_ascii=False, indent=3, default=convert,
                                           sort_keys=False).encode('utf8'),5)
    else:
        content = gzip.compress(
            json.dumps({'message':data, 'status': 'error'}, ensure_ascii=False, indent=3).encode('utf-8'), 5)
    response = make_response(content, status)
    response.headers['Content-Length'] = len(content)
    response.headers['Content-Encoding'] = 'gzip'
    response.headers['Content-Type'] = 'application/json; charset=utf-8'
def convert(o):
    if isinstance(o, np.bool_):
        if o:
            return True
        else:
            return False
    if pd.isna(o):
        return None

def read_json_time_series(dict_data):
    j_data = json.dumps(dict_data)
    data = pd.read_json(j_data)
    data.time = pd.to_datetime(data.time, unit= 'ms')
    return data
```

شامل توابع response_message، convert و read_json_time_series (که ورودی را به یک pandas datframe تبدیل می‌کند) است.

:interpolation_methods

```python
def linear_interpolation(data, config):
    if config['time'] == 'daily':
        data = data.set_index('time')
        data = data.resample('D')
        data = data.interpolate(method=config['interpolation'])
        data.reset_index(inplace=True)
    elif config['time'] == 'monthly':
        data = data.set_index('time')
        data = data.resample('M')
        data = data.index(inplace=True)
    else:
        data = None
    return data
```

همان‌طور که در عکس کد مشاهده می‌شود، این متود دیتا را می‌گیرد؛ time را ایندکس می‌کند. با resample تاریخ را به‌صورت روزانه درمی‌آورد، interpolation را انجام می‌دهد و در مرحله بعد دیتا را reset index می‌کند. و در خط آخر (15) دیتا را برمی‌گرداند.

:Dockerfile

```
FROM python:3.10-slim-buster

WORKDIR /app1

COPY . /app1

RUN pip install -r requirements.txt

ENTRYPOINT ["python"]

CMD ["app.py"]
```

که شامل نسخه استفاده‌شده، نام دایرکتوری، کپی کردن محتوا در دایرکتوری app، نصب پکیج‌های مورد نیاز (که در فایل requirement.txt قرار داده شده است) و در نهایت کد app را اجرا می‌کند.

Requirement.txt:

certifi==2022.5.18.1

click==8.1.3

colorama==0.4.4

Flask==2.1.2

itsdangerous==2.1.2

Jinja2==3.1.2

MarkupSafe==2.1.1

numpy==1.22.4

pandas==1.4.2

python-dateutil==2.8.2

pytz==2022.1

six==1.16.0

Werkzeug==2.1.2

wincertstore==0.2

:yml

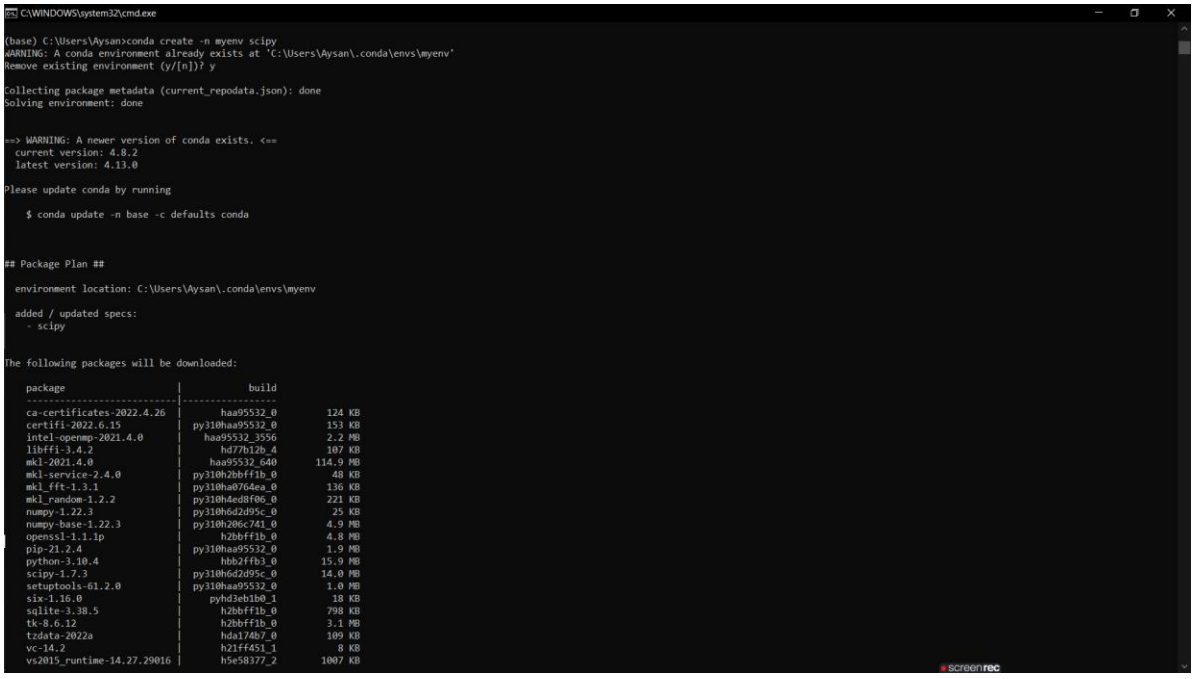
شامل بخش‌های kind، name و spec.

```
kind: ExternalService
name: fandogh-service
spec:
 image: fandoghi-service:0.1
 port: 80
 static_path: static
```

پیاده‌سازی بر روی بستر ابری:

- ساخت یک envorironment با کامند (درanaconda):

conda create --name myenv



- نصب CLI فندوق:

pip install fandogh_cli –upgrade

- در ادامه با دستور fandogh login در کامند وارد حساب فندق می‌شویم.

- سپس از دستور fandogh image publish –version 0.1 برای ساخت image استفاده می‌کنیم.

(تصویر بالا)

- همچنین پس از publish با دستور fandogh image publish –version 0.1 با استفاده از دستور fandogh service apply –f fandogh.yml، بعد از این‌که داکر image را ساختیم، مشخصات فایل yml را به آن می‌دهیم تا سرویس را اجرا کند.