Obiettivi generali dell'esercizio

Obiettivo principale dello sviluppo backend è la definizione di una piccola API REST che restituisca il contenuto di un file CSV in formato JSON.

Risultato

L'esercizio deve essere consegnato come codice sorgente, con istruzioni di installazione ed esecuzione.

Requisiti

- Linguaggio di programmazione: **PHP (>7.0)**, con framework **CakePHP**
- Web Server: **Apache**
- Utilizzare **Docker**
- Scrivere il file di **README** in lingua inglese, con le istruzioni di installazione ed esecuzione

Api Rest

- L'API Rest è composta da **due rotte**.
- Entrambe le rotte utilizzano come **datasource** il CSV di esempio, che **deve essere letto come stream**, evitando l'intero caricamento del file in memoria.
- le risposte fornite, in **formato JSON,** devono rispettare quanto riportato nella sezione del documento "Formati di risposta"

1. Rotta GET /flyers.json

Restituisce la **lista dei flyer attivi - non scaduti (***risposta 1*).
Un flyer è attivo se start_date <= CURRENT_DATE <= end_date. Tutti **i volantini non attivi non devono essere mai restituiti** da questa rotta.

Paginazione dei risultati (page, limit)

- prevedere in querystring i parametri **page** e **limit** per gestire la paginazione.
- i risultati sono paginati di default a gruppi di 100 (limit=100 page=1)
- ignorare eventuali valori errati per page e limit (interi negativi, lettere, stringhe) utilizzando, in questi casi, i defaults
- se la paginazione non dovesse restituire risultati, rispondere **404 Not Found** (*risposta 4*)

Selezione dei campi (fields)

- prevedere in querystring un parametro **fields** per scegliere i campi da restituire, con la lista dei campi separati da virgole. Esempio: ***/flyers.json?page=1&limit=100&fields=id,title***
- se vengono richiesti dei fields non esistenti, rispondere **400 Bad Request** (risposta 3, debug: "Not allowed fields: {{wrong field list separed by comma}}")

Filtri (filter)

- prevedere in querystring un parametro **filter** che vada a filtrare i risultati, nel formato filter[field]=value (esempio: filter[is_published]=1)
- gli unici filtri consentiti sono: **category**, **is_published**
- si deve poter filtrare contemporaneamente su entrambi i campi: Esempio: ***/flyers.json?filter[category]=Discount&filter[is_published]=1***
- se vengono applicati filtri su campi non esistenti o non consentiti, rispondere **400 Bad Request** (risposta 3, debug: “Not allowed filters: {{wrong filter list}}”)
- se le combinazioni non portano ad avere risultati, rispondere con status code **404 Not Found** (risposta 4, debug: “Not found”)

Esempio completo riportante tutti i parametri in querystring:
/flyers.json?page=2&limit=50&fields=title,category&filter[category]=Discount&filter[is_published]=1

2. Rotta [GET] flyers/{{id}}.json

Restituisce la risorsa flyer il cui ID è {{id}} (*risposta 2*).
La rotta restituisce anche i flyers non attivi.

Selezione dei campi (fields)

- prevedere in querystring un parametro **fields** per scegliere i campi da restituire, con la lista dei campi separati da virgole. Es: ***/flyers.json?page=1&limit=100&fields=id,title***
- se vengono richiesti dei fields non esistenti, rispondere con status code **400 Bad Request**, con un payload json riportante l’errore

{{id}} non esistente

- se l’id richiesto non esiste, rispondere con status code **404 Not Found** (risposta 4, debug: “Resource {{id}} not found”)

Formati di risposta

1. **Success - 200**

   Status Code: 200
   Body:

```
{
    “success”: true,
    “code”: 200,
    “results”: [
        {
            “id”: 1,
            “title: “Volantino 1”,
            ….
        },
        …..
```

```
                    {
                        “id”: N,
                        “title: “Volantino N”,
                        ….
                    },
                ]
        }
```

2. **Success - 200**

   Status Code: 200
   Body:

```
        {
            “success”: true,
            “code”: 200,
            “results”: {
                    “id”: 1,
                    “title: “Volantino”,
                    ….
                }
        }
```

3. **Error - Bad Request - 400**

   Status Code: 400
   Body:

```
        {
            “success”: false,
            “code”: 400,
            “error”: {
                    “message”: “Bad Request”,
                    “debug”: “{{custom depending by error}}”
                }
        }
```

4. **Error - Not Found - 404**

   Status Code: 404
   Body:

```
        {
            “success”: false,
            “code”: 404,
            “error”: {
                    “message”: “Not found”,
                    “debug”: “{{custom depending by error}}”
                }
        }
```

Eventuali altri errori ed eccezioni devono essere gestiti e restituiti nel formato di risposta “Error”, riportando il corretto status code, i campi success:false, code: “status code”, message: “messaggio dell’eccezione”, debug: vuoto.

Flyers Data (allegato flyers_data.csv)

I dati nel CSV corrispondono a una piccola parte dei dati dei volantini di ShopFully.
In particolare:

- id: id del volantino
- title: titolo del volantino
- start_date: data di inizio dell'offerta del volantino
- end_date: data di scadenza dell'offerta del volantino
- is_published: flag booleano
- retailer: nome del retailer di riferimento
- category: nome della categoria del volantino

Suggerimenti

- Scrivi codice leggibile e documentato, rispettando tutte le Best Practices. Eventuali commenti devono essere in lingua inglese
- Rispetta il pattern MVC
- Scrivi funzioni riutilizzabili, prevedendo una possibile crescita dell'applicazione (es.: come gestiresti la presenza di N sorgenti CSV?)
- Cerca di utilizzare le librerie messe a disposizione da CakePHP (es. Hash)
- Abbi un occhio di riguardo alle performance

Bonus

- Conosci i servizi AWS? Pubblica il tuo test su ELB e forniscici l'endpoint! (https://aws.amazon.com/free)
- Aggiungi gli Unit Test
- Aggiungi le specifiche OpenAPI 3.0 per gli endpoints