1 Practice 8.1. API REST

> This practice is the next in the series of practices consisting of a Contacts application. This practice is based on the previous one. Before start it, finish the previous one, make the corrections you consider, if needed, and start coding.
>
> The inclusion of comments in the scripts will be valued.
>
> Before start, create a new git branch named `contacts-u8` and change to it:
>
> ```
> git branch contacts-u8
> git checkout contacts-u8
> ```
>
> Do all the practice in the `contacts-u8` branch. Once finished, do the submission via GitHub Classroom as detailed in the end of the practice.

1.1 Exercise 1. REST Web services

Build the next web services:

Action	Method	Route
List of all contacts	GET	/api.php
Single contact	GET	/api.php?id=XX
Insert a new contact	POST	/api.php
Update a contact	PUT	/api.php?id=XX
Delete a book	DELETE	/api.php?id=XX

Make sure that, for the POST and PUT methods, the data are **validated** with the next rules:

- **title**: one of 'Mr.', 'Mrs.' or 'Miss'. If the provided value is none of these, insert the default value 'Mr.', no error returned.
- **name** and **surname**: required , not empty.
- **birthDate**: required and must be a valid date (use the `checkContactDate` method).
- **phone**. required and must be a valid phone (use the validation rules of unit 4).
- **email**. required and must be a valid email (use the validation rules of unit 4).
- **favourite**, **important** and **archived**: must be a boolean (`true`|`false`). If no data provided, insert the default value `false`.

Use a `validate()` method as shown in the Unit 8.

Structure of the responses:

- The first element must be `"result"` with 2 possible values: `"OK"` or `"Error"`.
- The second element must be `"data"`. The query results must contain an object with the element data (GET by id) or an array containing all the objects of the DB (GET all). For the PUT, POST and DELETE methods, must contain an object with the modified/added element data. If no `id` provided (PUT, DELETE and GET by id) or if the `id` doesn't exist, must shown an error message.
- A third element, `"errors"`, only must be shown if there are validation errors. Must contain an array with the property and the corresponding error message.

Some examples of queries:

Get requests:

```
//Get a contact by id
GET localhost/contacts-app/api.php?id=1
```

Result, correct id:


```
{
    "result": "OK",
    "data": {
        "id": 1,
        "title": "Mr.",
        "name": "Mike",
        "surname": "Molina",
        "birthDate": "1975-10-21",
        "phone": "555445466",
        "email": "molina@mail.com",
        "favourite": true,
        "important": true,
        "archived": true
    }
}
```


Figure 1: Get by id

Result, incorrect id:

Figure 2: Get by id not found

Post request:

```
// Correct POST request:
POST localhost/contacts-app/api.php
{
  "title": "Miss",
  "name": "Ruth",
  "surname": "Parsons",
  "birthDate": "1967-11-23",
  "phone": "611223344",
  "email": "parsons@mail.com",
  "favourite": true,
  "important": false,
  "archived": true
}
```

Result:

```
{
    "result": "OK",
    "data": {
        "id": 0,
        "title": "Miss",
        "name": "Ruth",
        "surname": "Parsons",
        "birthDate": "1967-11-23",
        "phone": "611223344",
        "email": "parsons@mail.com",
        "favourite": true,
        "important": false,
        "archived": true
    }
}
```


Figure 3: POST

Put request:

```
// Incorrect PUT request, invalid name and date:
PUT localhost/contacts-app/api.php?id=21
{
  "title": "Miss",
  "name": "    ",
  "surname": "Parsons",
  "birthDate": "2123-11-23",
  "phone": "611223344",
  "email": "parsons@mail.com",
  "favourite": true,
  "important": false,
  "archived": true
}
```

Result:

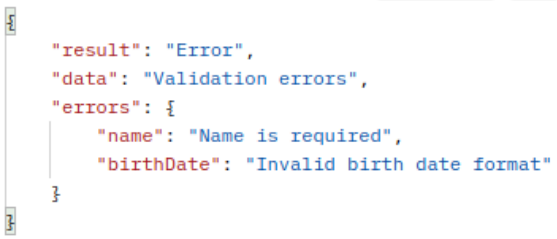

Figure 4: PUT

1.2 How to submit to GitHub Classroom

1. This task must be submitted to the same repository of the previous one. Remember to make a new branch before starting the practice.
2. Once you finish the task, make a commit with the comment "PRACTICE 8.1 SUBMISSION COMMIT", merge the branch into the main branch, and push it to GitHub. For example:

```
git commit -m "PRACTICE 8.1 SUBMISSION COMMIT"
git checkout main
git merge contacts-u8
git push
```

3. Before that, you can do the commits and pushes you want. If you change your code after your submission commit, make another commit and push with the same text in the message adding the corrections you've done. Make a pull-request if you want so i could know your code is submitted.
4. Once pushed to GitHub, create a tag with the name "Practice 8". Create a new tag if you make corrections after the first submission (ex. "Practice 8 corrected").

If you have any doubt in your task, you can push your code and ask me by email what's your problem.