1 Practice 13.1. REST web service with Symfony

> This practice is the next of the series of practices, consisting of a Library application. Each practice will be based on the previous one. Before starting it, finish the previous one, make the corrections you consider, if needed, and start coding.
>
> Before start, create a new git branch named `library-u13` and switch to it:
>
> ```
> git branch library-u13
> git checkout library-u13
> ```
>
> Do all the practice in the `library-u13` branch. Once finished, do the submission via GitHub Classroom as detailed at the end of the practice.

1.1 Overview

In this practice we're going to do an API REST for the app. The app will have 3 access levels:

- **Public access**: can access the login and register url's.
- **Registered users**: can access all the url's to list, view and search for books and publishers, but can not edit, delete or add any data.
- **Admin users**: can access all the url's and can edit, delete and add all the data.

The api will be done with API Platform, not manually.

1.2 Exercise 1. REST Web services

Build the next web services:

Action	Method	Route	Notes
List of all books	GET	/api/books/	
Single book	GET	/api/books/{isbn}	
List of all publishers	GET	/api/publishers/	
Single publisher	GET	/api/publishers/{id}	
Insert a new book	POST	/api/books/	The book must have an existing publisher identified by its `id`

Action	Method	Route	Notes
Insert a new publisher	POST	/api/publishers/	
Update a book	PUT	/api/books/{isbn}	The book must have an existing publisher identified by its `id`
Update a publisher	PUT	/api/publishers/{id}	
Delete a book	DELETE	/api/books/{isbn}	
Delete a publisher	DELETE	/api/publishers/{id}	
List of all books of a single publisher	GET	/api/publishers/{id}/books	

Hide the operations not listed (PATCH).

The GET books operations must show the publisher's data as part of the response. For instance:


```
{
  "@id": "/api/books/F66676440",
  "@type": "Book",
  "isbn": "F66676440",
  "publisher": {
    "@id": "/api/publishers/2",
    "@type": "Publisher",
    "id": 2,
    "name": "Ecco",
    "email": "ecco_info@ecco.coma"
  },
  "title": "The Lathe Of Heaven",
  "author": "Ursula",
  "pages": 192,
  "pub_date": "2000-04-15T00:00:00+00:00"
},
```


Figure 1: Get books with publisher info

1.3 Exercise 2. Validation

Make sure that the data are **validated** following the next rules, before adding them to the database:

- Book:
  - ISBN and title, required
  - Pages, a numeric value equal or greater than zero.
- Publisher:
  - Name, required.
  - Email, not required and email format.

Some examples of queries:

```
// Correct POST request:
POST localhost:8000/api/books/
{
    "isbn": "A111B7",
    "title": "The Hobbit",
    "author": "J.R.R. Tolkien",
    "pages": 677,
    "pub_date": "2020-11-03",
    "publisher": "/api/publishers/2"
}
```

```
// Incorrect PUT request, pages less than zero:
PUT localhost:8000/api/books/A111B7
{
    "isbn": "A111B7",
    "title": "The Hobbit",
    "author": "J.R.R. Tolkien",
    "pages": -10,
    "pub_date": "2020-11-03",
    "publisher": "/api/publishers/2"
}
```

1.4 Exercise 3. Authentication and authorization

The API will have 3 access levels:

- **Public access**: can't access the API, only to the registration and login url's.
- **Registered users**: can access all the API routes to **list** and **view** books and publishers (GET), but can not edit (PUT), delete (DELETE) or add (POST) any data.
- **Admin users**: can access all the API routes and can edit, delete and add all the data.

The authentication must be based in a user entity. The users will be stored using an entity named `User` in a table named `user`.

The user fields to be stored are:

- Name (string, 100 characters, not null)
- Email
- Phone (string, 25 characters, can be null)
- Password

The user's identifier must be the email. And the password must be stored encrypted. It's not necessary to send a confirmation email to the user to complete the register.

Once done the table in the database, it should be:

```
mysql> describe user;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | int          | NO   | PRI | NULL    | auto_increment |
| email    | varchar(180) | NO   | UNI | NULL    |                |
| roles    | json         | NO   |     | NULL    |                |
| password | varchar(255) | NO   |     | NULL    |                |
| name     | varchar(100) | NO   |     | NULL    |                |
| phone    | varchar(25)  | YES  |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
```

> In MariaDB the `roles` field can be `longtext`.

Our app will have the next roles:

- Administrators, with the role `ROLE_ADMIN`.
- Registered users, with the role `ROLE_USER`.
- Unauthenticated users.

Make the authentication based on tokens, using the `lexik/jwt-authentication-bundle` library. The authentication route must be **`/auth`** (only POST) and the registration route **`/api/register`** (only POST).

1.5 How to submit to GitHub Classroom

Once you finish the task, make a commit with the comment "PRACTICE 13.1 SUBMISSION COMMIT" and push it to GitHub. Make a **pull-request** so that i could know your code is submitted.

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 13.1 SUBMISSION COMMIT”, merge the branch into the main branch, and push it to GitHub. For example:

```
git commit -m "PRACTICE 13.1 SUBMISSION COMMIT"
git checkout main
git merge library-u13
git push
```

3. It’s recommended to tag your commit with the tag “Practice_13.1”.
4. Before that, you can do the commits and push 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.

If you have any doubt in your task, you can push your code and ask me by email what’s your problem. It will make it easier for both the solutions of code issues.