Books wishlist

The coding challenge consists in the implementation of a web application that exposes a REST-ful API over HTTP for creating book's wishlists for different users.

Below you will find the requirements in the form of epic with its corresponding user stories. It is worth mentioning that they lack refined details so that you can fill the gap as you see fit.

This assignment will serve as a large portion of your technical interview evaluation, and will be the foundation for your future technical meeting with our team.

As deliverable, we expect:

- The code must be hosted on a **private** GitHub repository so that we can request access to clone it and run it locally using docker.
- To run it locally, you must provide a Dockerfile and instructions on how to build the image and run a container with the application.
- The repository must contain a README.md with the above instructions and the API documentation so that we can use cURL to interact with the application.

Epic

Any registered user can create one or more wishlists that will contain references to books from the Google book API.

User Stories

1) As a `new user`, I want to register to the application using **username** and **password**.
2) As a `registered user`, I want to log in with my **username** and **password**.
3) As an `authenticated user`, I want to search for books using the **Google API** https://developers.google.com/books.
4) As an `authenticated user`, I want to create a `wishlist` of books.
5) As an `authenticated user`, I want to add/delete a book from a `wishlist` of mine.
6) As an `authenticated user`, I want to list all the `wishlists` created by me.
7) As an `authenticated user`, I want to list the books of a `wishlist` of mine.
8) As an `authenticated user`, I want to delete a `wishlist` created by me.

Functional requirements

- An `authenticated user` is anyone accessing the application with an `access_token` associated to a user.
- When we search for books, we would like to do it at least by author, title and/or publisher.
- A `book` stored in the system must contain the Google book ID, the author, the title and the publisher.

Non-functional requirements

- We very much value code quality and technical design. Think about the structure of your data models and the readability of your code.
- For persistence, we recommend using SQLite but feel free to use any other relational database that we can spin-up using docker. In that case, we expect a docker-compose.yml that we can use to create and run all the containers.
- We expect you to design the API the way you consider appropriate. There are no hard rules. We do expect good proper use of HTTP status codes.
- Since requests to the Books API for public data must be accompanied by an **API key**, it must be provided by us as a query parameter in the endpoint that searches for books.

Example

Below you will find an example of what we expect you to include in the API documentation. Feel free to choose your own template.

While you can use the proposed API for US 1 and 2 as reference, you are not constrained to use it as is. We just want to give you something to get started.

POST /sign-up

The sign-up endpoint creates a new user with the credentials needed to sign in to the application.

Request

```
{
  "username": "robpike",
  "password": "passtheword"
}
```

Response

`empty`

- **201 Created** The user was created successfully.
- **400 Bad Request** The request is invalid.
- **409 Conflict** The username is already taken.

POST /sign-in

The sign-in endpoint authenticates an existent user using the provided credentials.

Request

```
{
  "username": "robpike",
  "password": "passtheword"
}
```

Response

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

- **200 OK** The user was authenticated successfully.
- **400 Bad Request** The request is invalid.
- **401 Unauthorized** The user provided invalid authentication credentials.