# WEB PROJECT

## Deliverable 2:
## Web 2.0 Application Development

**Ariza Pérez, José Ramon**

48257747K

**Buturuga, Mihaela Alexandra**

X9278777W

**Chen, Zihan**

Y1070565K

**Rodríguez Gonzàlez, Arnau**

39508430L

**Yang, Hao**

X4700996A

Universitat de Lleida

Escola Politècnica Superior

Computer Engineering

2023, May 19th

# Contents

# 1 Introduction

The goal of this document is to explain the changes and that have been made in the design since the last deliverable of the project to include Web 2.0 features, that allow users to create, update and delete instances of some models.

We also have made some E2E tests to check the correct behavior of the application instead of doing it manually.

Since this deliverable is not about deploying the application online or how to deploy it using Docker, we have created a new GitHub repository with a copy of the last project that does not use a Docker-compose structure, in order to facilitate the development of this new deliverable and avoiding deleting files.

The GitHub repository of this deliverable can be accessed via the following link:

`https://github.com/GEI-WebProject/bookreviews_2.git`

Otherwise, the GitHub repository of the previous project can also be accessed via the following link:

`https://github.com/GEI-WebProject/bookreviews.git`

Currently, the website has the next users that can be used to access and test the features manually:

- Superuser
  - User: admin
  - Password: admin
- Test User 1
  - User: test
  - Password: pass12345
- Test User 2
  - User: test2
  - Password: pass12345

We are aware that using `admin` as a password for the user `admin` is not secure at all, but it has been decided to be this way for commodity when implementing new features and models for the application, but it wouldn't be like this in a real deployment.

Finally, the books that are already reviewed are:

- Twilight
- The Hunger Games

## 2 Create, update and delete instances

Since our application is called BookReviews, to turn our Web 1.0 website into a Web 2.0 one, in this project it has been implemented that registered users are able to write reviews about existing books on the website, and update or delete them as well.

We have created a new Django application called `reviews` that handle the models, the views and the URLs of this new feature of the website.

Now, users can create multiple reviews for each book of the website. Besides, they only have permission to update or delete their own reviews. This is restricted not only in the interface since the buttons designed for these tasks are only visible to the owners, but also if you try to access via URL you will get a `403 Forbidden` error.

This has been achieved using the Django built-in class-based views like `CreateView`, `UpdateView` and `DeleteView`. These classes handle automatically the creation, update and deletion of reviews using a form.

These classes have been complemented with the `LoginRequired` and `UserPassesTest` mixins, that handle that only logged in users can perform certain tasks and only the owners of the reviews can update or delete them.

Otherwise, they are redirected to the login page if they are not logged in, or an HTTP error is returned in case they don't own that review.

# 3 Data incorporation from an external API

Since the application needs books to be shown and to let users create reviews about them, instead of adding them manually, an external API has been used to facilitate this task. Specifically, it has been used the OpenLibrary API and the GoogleBooks API.

It has been decided to use two APIs to combine the results from both of them and be able to add books that have all the information needed, since these APIs are not always consistent with the information they handle.

To incorporate all the data, it has been created a new Django command called `api_insertion`. This command retrieves 100 books of different pre-selected genres and languages. This file is stored in `bookapp/management/commands`.

Combining information of the both APIs used, all the books have defined all the fields registered in the model filled (Title, ISBN, Author, Synopsis, Genres, Language, Publisher, Cover).

The command to be executed is:

```
$ python manage.py api_insertion
```

In case that you want to change the pre-existent specifications of the books that are retrieved, you must modify the `_api_commands.py` script that contains all the methods that make HTTP petitions to the APIs. It is also stored with the previous file.

# 4 E2E Tests

Finally, in order to test all the new features implemented in the application, it has been included some E2E tests using `Splinter` and the `Behave` schema.

Splinter is a Python library that enables automated web browser interactions, allowing developers to simulate user actions such as clicking buttons, filling forms, and navigating through web pages.

Behave, on the other hand, is a behavior-driven development (BDD) framework for Python that uses a common language to describe the application's behavior in a human-readable format. Behave combines natural language-like test scenarios with executable code, making it easier to understand and maintain test cases.

So, in order to test all the features, the next situations have been tested:

- A user can create an account (Sign up).
- A user can access its account (Log in).
- A user can search books.
- A registered user can create a review.
- A registered user can only update its own review.
- A registered user can only delete its own review.
- A non-registered user can't create, update or delete reviews.

Finally, since the reviews are rated, each book has an average rating that is displayed in its detail page. To check that the average rating is calculated correctly, it has been included some Django tests that check the cases when a book has multiple reviews and a book has not been reviewed yet.