Saarthi .ai

Python API Development Assignment

Introduction

This is a short coding assignment, in which you will implement a REST API that calls an external API service to get information about books. Additionally, you will implement a simple CRUD (Create, Read, Update, Delete) API with a local database of your choice.

The external API that will be used here is the Ice And Fire API. This API requires no sign up / authentication on your part.

Ground Rules

- You must use Django framework.
- You may use any web server you like.
- Your solution should be submitted as a link to a GitHub repository containing a README file written in English.
- Please deploy the application on any linux based cloud hosting provider - AWS / DigitalOcean / Heroku[free] / PythonAnywhere [free] etc.
- Automated scripts to deploy the application would be a huge plus point
- We should be able to verify your submission by cloning your repository, configure the environment and running the application. Please document all details about project setup in the README file.
- We estimate that this test should take roughly 4-5 hours to complete.

Requirements to Implement

Your solution should provide an implementation for each of the following requirements. The tasks will be tested with the specified endpoints while expecting the specified JSON.

Requirement 1

When the endpoint:

```
GET http://localhost:8080/api/external-books?name=:nameOfABook
```

is requested, your application should query the Ice And Fire API and use the data received to respond with precisely the following JSON if there are results:

```
{
    "status_code": 200,
    "status": "success",
    "data": [
        {
            "name": "A Game of Thrones",
```

```
            "isbn": "978-0553103540",
            "authors": [
                "George R. R. Martin"
            ],
            "number_of_pages": 694,
            "publisher": "Bantam Books",
            "country": "United States",
            "release_date": "1996-08-01"
        },
        {
            "name": "A Clash of Kings",
            "isbn": "978-0553108033",
            "authors": [
                "George R. R. Martin"
            ],
            "number_of_pages": 768,
            "publisher": "Bantam Books",
            "country": "United States",
            "release_date": "1999-02-02"
        }
    ]
}
```

or precisely this JSON if the Ice and Fire API returns no results:

```
{
    "status_code": 200,
    "status": "success",
    "data": []
}
```

where :nameOfABook is a variable. Example value for :nameOfABook can be A Game Of Thrones.

Note that the JSON property names that Ice And Fire uses don't quite match the ones in the required output that your application needs to deliver, so pay attention to this. Also, not all of the output from Ice And Fire is required in your application's output - you may need to suppress some field(s).

Requirement 2

Create

When the endpoint:

```
POST http://localhost:8080/api/v1/books
```

is requested with the following data:

- name

- isbn
- authors
- country
- number_of_pages
- publisher
- release_date

a book should be created in the local database and the following response should be returned:

```
{
    "status_code": 201,
    "status": "success",
    "data": [
        { "book": {
            "name": "My First Book",
            "isbn": "123-3213243567",
            "authors": [
                "John Doe"
            ],
            "number_of_pages": 350,
            "publisher": "Acme Books",
            "country": "United States",
            "release_date": "2019-08-01"
        } }
    ]
}
```

Read

When the endpoint:

```
GET http://localhost:8080/api/v1/books
```

is requested, your solution will return a list of books from the local database using the following response:

```
{
    "status_code": 200,
    "status": "success",
    "data": [
        {
            "id": 1,
            "name": "A Game of Thrones",
            "isbn": "978-0553103540",
            "authors": [
                "George R. R. Martin"
            ],
            "number_of_pages": 694,
            "publisher": "Bantam Books",
            "country": "United States",
```

```
            "release_date": "1996-08-01"
        },
        {
            "id": 2,
            "name": "A Clash of Kings",
            "isbn": "978-0553108033",
            "author": [
                "George R. R. Martin"
            ],
            "number_of_pages": 768,
            "publisher": "Bantam Books",
            "country": "United States",
            "release_date": "1999-02-02"
        }
    ]
}
```

or precisely this JSON if the Books API returns no results:

```
{
    "status_code": 200,
    "status": "success",
    "data": []
}
```

The Books API should be searchable by name (string), country (string), publisher (string) and release date (year, integer).

Update

When the endpoint:

```
PATCH http://localhost:8080/api/v1/books/:id
```

is requested with any of the following form data:

- name
- isbn
- authors
- country
- number_of_pages
- publisher
- release_date

and a specific :id in the URL, where :id is a placeholder variable for an integer (for example 1), the specific book should be updated in the database and the following response should be returned:

```
{
    "status_code": 200,
    "status": "success",
```

```
    "message": "The book My First Book was updated successfully",
    "data": {
        "id": 1,
        "name": "My First Updated Book",
        "isbn": "123-3213243567",
        "authors": [
            "John Doe"
        ],
        "number_of_pages": 350,
        "publisher": "Acme Books Publishing",
        "country": "United States",
        "release_date": "2019-01-01"
    }
}
```


An alternative endpoint for updating books if you are not using a framework should be:

```
POST http://localhost:8080/api/v1/books/:id/update
```

Delete

When the endpoint:

```
DELETE http://localhost:8080/api/v1/books/:id
```

is requested with a specific :id in the URL, where :id is a placeholder variable for an integer (for example 1), the specific book should be deleted from the database and the following response will be returned:


```
{
    "status_code": 200,
    "status": "success",
    "message": "The book My First Book was deleted successfully",
    "data": []
}
```


An alternative endpoint for deleting a book could be:

```
POST http://localhost:8080/api/v1/books/:id/delete
```

Show

When the endpoint:

```
GET http://localhost:8080/api/v1/books/:id
```

is requested with a specific :id in the URL, where :id is a placeholder variable for an integer (for example 1), it should show the specific book and the following response will be returned:


```
{
    "status_code": 200,
```

```
    "status": "success",
    "data": {
        "id": 1,
        "name": "My First Book",
        "isbn": "123-3213243567",
        "authors": [
            "John Doe"
        ],
        "number_of_pages": 350,
        "publisher": "Acme Books Publishing",
        "country": "United States",
        "release_date": "2019-01-01"
    }
}
```

Evaluation Instructions

When evaluating the submitted solution, we will put attention on the following:

- **Fully functional solution**: we will test the API endpoints and see if they return the expected results based on the provided input.
- Make sure to have things well documented in the README, so we can easily set up the project and test it.
- **Good code design**: we put special attention to the design of the code. Make sure your code is clean and readable, it complies with the SOLID design principles and makes use of design patterns where it makes sense.
- **Tested code**: don't forget to write your tests before submitting. You may use Python's Unittest module or any third party testing framework like Nose or PyTest.
- **Code Coverage**: Make sure to submit the code coverage for the assignment, a code coverage would mean 95% or more.

- **Benchmark**: Benchmark the APIs for various load and stress scenarios.

Bonus Points!

Show the results in the browser using Angular or React frameworks in code highlighted JSON format.