Test Otomasyon Mühendisi Ödevi

Bölüm 1

1. Trendyol ana sayfasındaki tüm butik linkleri ve bu linklere request atıldığında dönen response code bir csv dosyasına kaydedilmeli.
2. Scroll edildiğinde gelen butik image linklerinin yüklenme süreleri ve response code'ları bir csv dosyasına kaydedilmeli.
3. Kendi tasarlayacağın login caselerini, data driven bir şekilde koşmanı bekliyoruz. Burada bir login case'inde yapılması gerektiğini düşündüğün tüm testleri yazmanı bekliyoruz.

- Sistem paralel ve crossbrowser olarak koşacak şekilde dizayn edilmeli.
- Login testinde case'lerden biri fail olacak şekilde koşup, fail anının screenshot'ı raporlanmalı.
- Test yazılırken okunabilirlik, isimlendirmeler, kod tekrarı, OOP gibi evrensel yazılım standartlarına dikkat edilmeli, mimari açıdan büyümeye hazır bir class yapısı kurulmalı ve tasarım desenleri kullanılmalıdır.
- Testler java diliyle, selenium kütüphanesi kullanılarak geliştirilmeli.
- Testlerin sonunda sonuçların gözüktüğü bir raporlama ekranı eklenmeli.
- Testlerin nasıl ve nerede koşacağı ile ilgili kısa bir read.me dosyası hazırlanmalı.
- Testler docker containerları içinde selenium-grid ile birlikte, compose dosyası üzerinden çalıştırılabilmelidir.

Bölüm 2

Your friend has written an app that gives you a convenient way of storing a list of books you own. The application uses a REST API with a backend server to exchange information. Your task is to test the REST API used by the app to make sure it works correctly.

The server contains only one object:

```
class Book: int id; // Read-only. String author; String title;
```

The communication between backend and app is serialized into JSON format for convenience and readability. An example book serialized to JSON appears as follows:

```
{ "id": 1, "author": "John Smith", "title": "Reliability of late night deployments" }
```

An example response containing a list of books is as follows:

```
[ {"id": 1, "author": "John Smith", "title": "SRE 101"}, {"id": 2, "author": "Jane Archer", "title": "DevOps is a lie"}, ]
```

For consistency, you *cannot* add a book with the same title and author twice. The API consists of the following endpoints:

```
/api/books/
```

When called with the GET method, returns the list of your books.

When called with the PUT method, inserts a new book into your list and returns the newly created book. Returns an error if any occurred.

```
/api/books/<book_id>/
```

When called with the GET method, returns a single book. Returns HTTP 404 Not Found if the book with the given id does not exist.

All error responses have a status code of HTTP 400 Bad Request and contain an object with a single key, error, that contains the error message. An example error message as follows:

```
{ "error": "Field 'author' is required" }
```

Requirements

1. Verify that the API starts with an empty store.
    - At the beginning of a test case, there should be no books stored on the server.
2. Verify that title and author are required fields.
    - PUT on /api/books/ should return an error Field '<field_name>' is required.
3. Verify that title and author cannot be empty.
    - PUT on /api/books/ should return an error Field '<field_name>' cannot be empty.
4. Verify that the id field is read–only.
    - You shouldn't be able to send it in the PUT request to /api/books/.
5. Verify that you can create a new book via PUT.
    - The book should be returned in the response.
    - GET on /api/books/<book_id>/ should return the same book.
6. Verify that you cannot create a duplicate book.
    - PUT on /api/books/ should return an error: Another book with similar title and author already exists.

Assessment/Tools

- The API might return an HTTP 500 Internal Server Error. This is never intended, and a test that encounters it should fail appropriately.
- Make sure your test class extends APITestCase.
- Make sure all your test cases are decorated with @Test.
- The testing API is reset before each of your test cases. You don't have to worry about it.
- The APITestCase.API_ROOT variable contains the API root you should use to access the API.
- Make sure at least one test detects regression in one of the requirements and breaks.
- Performance is not assessed; focus only on the correctness of the tests.