GOALS

The main goal of this tech test is to create a web crawler that automatically detects Marfeelizable sites. To simplify the requirements we'll consider a site is Marfeelizable if the <TITLE> tag contains the keywords "news" or "noticias"

The workflow is as follows:

1. Your crawler will expose a REST API that can receive a set of site URLs in JSON format.
2. For every URL, your crawler will visit the site and perform the test to determine if the site is Marfeelizable or not.
3. Then, it will persist the results of the site qualification.

REQUIREMENTS

- Create a web application with a REST interface, capable of receiving a JSON document with a list of URLs (see example at the end).
- You should parallelize your business logic, creating a multithreaded application (i.e: your crawler should visit different sites concurrently).
- Your application will have a persistence layer where to save the results.

TECHNICAL CONSIDERATIONS

- You can only use the following libraries / tools:
    - Spring MVC for the REST API.
    - Spring Data for the persistence layer, but you can use the persistence technology you wish (i.e: MongoDB, MySQL, H2…).
    - Java SDK 8.
    - jSoup Java library to parse HTML.
    - Maven to build the application structure.
    - JUnit, Spring Test, Mockito and/or Hamcrest for testing.
- Write your Spring application context manually. Avoid using tools like Spring Boot.

WE WILL EVALUATE THE DEVELOPMENT BASED ON:

- Clean, maintainable, easy-to-read and unit tested code.
- Good architectural practices, with clear separation of concerns.

- Good use of language, libraries and frameworks detailed in technical considerations.
- Crawling parallelization.
- Extensibility: Ability to add different qualifier implementations to check if a site is Marfeelizable or not.

NOT REQUIRED, BUT IMPRESSIVE

The received set of URL could be quite big, meaning long execution time. Therefore, your application should try avoid any blocking tasks. You can use techniques like:

- Use of async servlet.
- Apply reactive programming techniques that allow non blocking IO requests and avoid thread blocking.

JSON document example:

```
[
  {
    "url": "c-and-a.com"
  },
  {
    "url": "toshiba.es"
  }
]
```