**Hometrack API Challenge**

**Agenda :**

* Build a base API test automation framework.
* Test the “Profanity Filter” API endpoints by [PurgoMalum — Free Profanity Filter Web Service](http://www.purgomalum.com/)
* Discusses the approach and future updates

**Use case :**

* Profanity Filter contains several combinations of GET requests, accepting query parameters.
* Based on the input it will indicate if an input had a curse word.

**API endpoint :**

* **Base URI** :[**https://www.purgomalum.com/service/**](https://www.purgomalum.com/service/)
* **Methods :**

| **Methods** | **Return Type** | **Tests** |
| --- | --- | --- |
| **xml** | **xml** | 1. **Test XML method type with/without slang** 2. **Assert asterix while slang or otherwise** |
| **JSON** | **jason** | 1. **Test JSON method type with/without slang** 2. **Assert asterix when slang or otherwise** |
| **plane** | **txt** | 1. **Test txt method type with/without slang** 2. **Assert asterix when slang or otherwise** |
| **containsprofanity** | **txt** | 1. **Test response** 2. **Assert asterix when slang or otherwise** |

* **Parameters :**

| **Params** | **Required** | **Tests** |
| --- | --- | --- |
| **text** | **Yes** | 1. **This is mandatory params & got tested in all cases** |
| **add** | **optional** | 1. **Tested this with word replacement method** |
| **fill\_text** | **optional** | 1. **Tested this with word replacement method** |
| **fill\_char** | **txt** | 1. **NOT TESTED** |

* **Stack used :** 
  + **C#**
  + **RestSharp**
  + **SpecFlow**
  + **NUnit**
  + **Extent Report**
* **Important Points :**
  + A BDD approach is taken with dynamic input value
  + Hooks are used for setup, teardown or any manipulation of test patterns.
  + 2 feature file & corresponding Step Def is created to segregate between method test and parameter tests.
  + Reporting is added using Extent report
  + Context injection used to decouple/share common objects
* **Improvement scope given time :**
  + Isolate in a Page object manner if more endpoints to test eg: separate class for method types etc
  + Adding custom Logger, screenshot method upon failure, excel /sql driven
  + Implementing CI/CD if dev code sits together & provision executors on cloud.
  + Cover more negative test criteria, extreme cases & proper documentation
* **Note :** 
  + Kept one failed test case to test reporting
  + For running,simply build & run