Proposta nº 2024.008

Plataforma de Automação de Testes para Web

Proposta Técnica v.0.1

Junho de 2024

Responsável Técnico
Fabiano Salles
fabiano@aeonsoft.com.br

Cliente
Rainbow Tecnologia

Índice

Objeto da Proposta

Desenvolvimento de uma plataforma para criação e execução de testes de interface de usuário para sistemas web.

Detalhamento

A idéia é ter uma plataforma que permita escrever e executar testes que simulam casos de usos/interações reais de um usuário comum, ou seja, o sistema deverá ser capaz de iniciar uma instância controlada do navegador e executar uma sequência de passos tal qual um humano o faria, preenchendo campos, modificando dados, clicando em elementos, navegando pela aplicação, movendo o mouse, etc…

O problema pode ser dividido em duas partes:

1. Como especificar um teste para o sistema
2. Como validar uma especificação
3. Como executar as ações definidas numa aplicação real rodando no navegador

Escrevendo os casos de teste

Para escrever os casos de testes vamos utilizar o YAML, uma linguagem simples e amplamente utilizada para escrever arquivos de configuração, arquivos de dados complexos em um formato compacto e de simples leitura. Além dessas características, existem excelentes parses de uso livre disponíveis para uso imediato em C#.

Um arquivo YAML conterá um caso de testes (podendo suportar múltiplos casos numa futura versão). Este será o aspecto de um caso de teste:

```yaml
test:
  name: 'Try login without password'
  version: 1.0
  arrange:
      set_window: { w: 800,h: 600 }
      navigate: '/login'
  act:
      send_keys: { element: tbx_login, value: admin }
      click: btnLogin
      wait: 3000 # wait for 3 seconds
  assert:
    # 1. check if the error-msg element exists
    assert_exists:
      element: 'error-msg'
      error: 'pop up de erro não encontrado'
    # 2. check if the error-msg element contains a specific value
    assert_text_equals:
        element: 'error-msg'
        value: 'Password must be informed'
```

O código acima contém todos os passos para executar um caso de teste em que o usuário tenta fazer o login para o usuário “admin” sem especificar a senha. O teste falhará se, após clicar no botão “btnLogin” a mensagem de erro “Password must be informed” não estiver visível ao usuário.

O código é dividido em 3 secções lógicas:

1. **Arrange**
   Código de inicialização do teste. É aqui que devemos iniciar a instância do navegador, redimensioná-lo e navegar para página ou seção que queremos testar.
   Aqui estarão disponíveis todos os métodos que o webdriver suportar
2. **Act**
   O código nesta seção deve simular o comportamento do usuário que é de interesse para o teste em questão.
   Aqui estarão disponíveis todos os métodos que o webdriver suportar
3. **Assert**
   Depois de simular os passos do usuário, é hora de verificar se o sistema se comportou como esperado. É a partir dos métodos desta seção que os resultados dos testes serão produzidos.
   Aqui estarão disponíveis uma lista de métodos de assertion a definir

Validando os casos de teste

Para validar os casos de teste escritos no formato definido acima, vamos utilizar o YamlDotNet para realizar o parser e nos entregar objetos fortemente tipados para trabalhar. Uma vez parseados e validados os casos de testes, podemos entregar a estrutura de dados resultantes para um interpretador que saberá como lidar com o navegador e como produzir dados de resposta para feedback do usuário/programador da plataforma.

Executando os casos de teste

Esta é a parte mais complexa do sistema pois envolve muitos componentes externos.

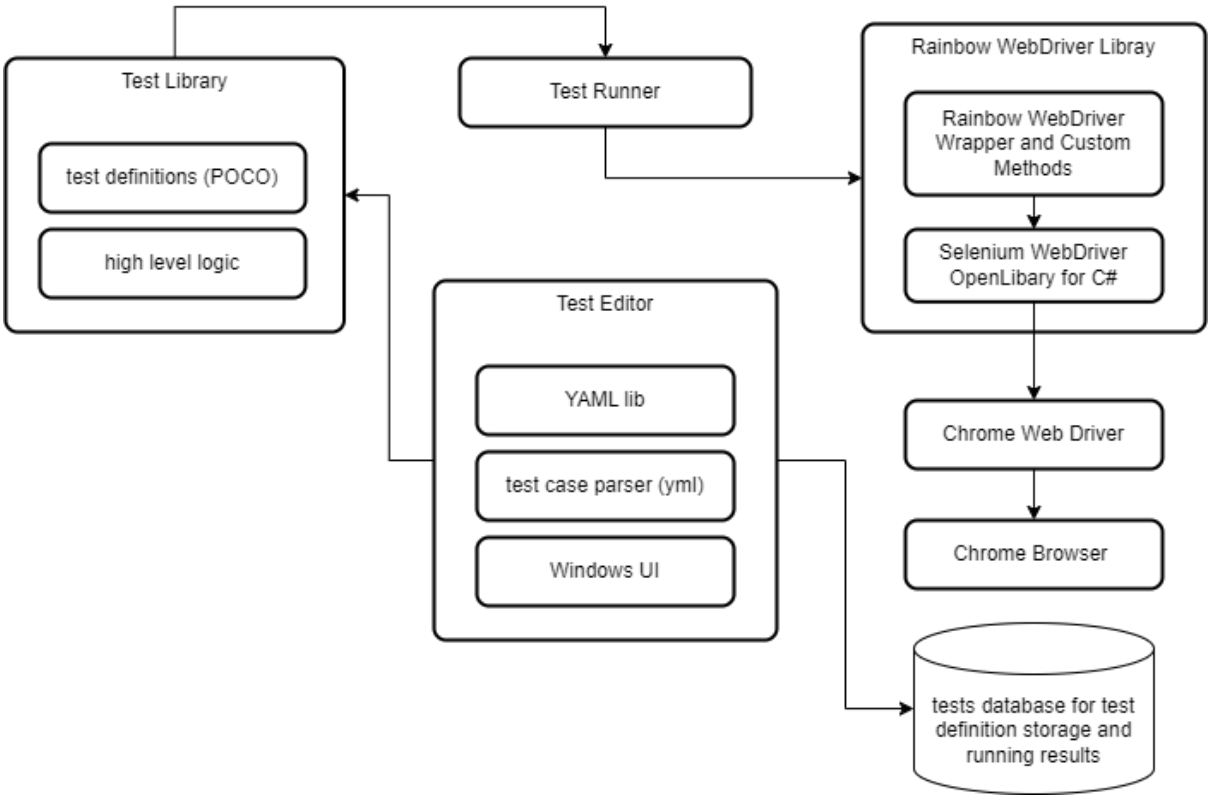
Primeiro, os navegadores oferecem suporte ao que chamamos de “webdriver”, um software que deve estar instalado na máquina do usuário que expõe uma api para intermediar a automação de tarefas no navegador. A especificação do protocolo do webdriver pode ser encontrada na seção de links deste documento.

Uma opção rápida para implementação deste protocolo, é criar uma camada própria em cima da biblioteca .net do Selenium, mas isto vem com o ônus de adicionar uma camada a mais de dependência. Caso isto se torne um problema no futuro, podemos revisar o projeto e criar nossa própria implementação do webdriver sem o uso de bibliotecas intermediárias.

Bom, de posse das estruturas de dados do teste e do webdriver, chamaremos um “test runner”, que será um conjunto de objetos que saberão executar os testes em sequência ou, quando possível em paralelo, se valendo da possibilidade de ter múltiplas instâncias de um único webdriver controlando, cada um sua própria instância do navegador.

Overview da Arquitetura

O diagrama a seguir mostra um overview da arquitetura da plataforma onde podemos ver a interação de todos os componentes envolvidos.



Deliverables

Além do Códig-fonte, este projeto será considerado concluído quando os seguintes artefatos forem entregues e aceitos

Parser

Utilitário de linha de comando para parsear e validar arquivos yml de casos de testes

Runner

Utilitário de linha de comando para executar um ou mais arquivos yml de casos de teste.

VSCode Templates

Temples do VSCode para integrar os utilitários acima, permitindo o parsing, a execução e o acompanhamento dos logs/resultados via terminal integrado

Links e Referências

- Especificação Procolo WebDriver (W3C)
- Biblioteca YamlDotNet
- Biblioteca Selenium DoNet
- YAML Web Editor Component