Benemérita Universidad Autónoma de Puebla

FACULTAD DE CIENCIAS LA COMPUTACIÓN

Implementación de un sistema basado en aprendizaje reforzado para la realización de pruebas autónomas de aplicaciones web desde su interfaz gráfica.

TESIS

PARA OBTENER EL GRADO DE

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

Jesús Alberto Islas Fuentes

Director:
Dr. Ivo H. Pineda Torres

Co-Directora:
Dra. Darnes Vilariño Ayala

Puebla, Puebla. Enero 2021

Agradecimientos

A mis padres por todo su cariño y apoyo incondicional durante todos estos años, por siempre creer en mí y alentarme a seguir mis sueños cada día.

A mis asesores:

Dr. Ivo H. Pineda Torres, por el apoyo y la oportunidad de desarrollar este trabajo bajo su tutoría, también por cada uno de los consejos brindados y conocimiento transmitido que hicieron posible la conclusión de este trabajo de forma satisfactoria.

Dra. Darnes Vilariño Ayala, por aceptar ser mi tutora en este proyecto y brindarme todas las facilidades necesarias para llevar a buen puerto este trabajo.

A mis profesores y a la Facultad de Ciencias de Computación de la Benemérita Universidad Autónoma de Puebla, por todo el conocimiento y apoyo brindado que me permitió formarme como maestro en ciencias.

Dedicatorias

A mi madre Marina Fuentes Bailón por todo cariño y el apoyo, la educación brindada a lo largo de mi vida que me han llevado a ser la persona que soy hoy en día, por siempre creer en mí sin importar las circunstancias e impulsarme a seguir superándome.

A mi padre Gustavo Islas Morales, por todo el cariño y apoyo recibido a lo largo de mi vida, así como inculcarme desde pequeño el amor al conocimiento y ser un modelo a seguir tanto personal como profesionalmente.

A mi hermana Anabel Islas Fuentes, por estar siempre presente y apoyarme en cualquier circunstancia.

A mis amigos y familiares, por su apoyo y motivación para seguir superándome día con día.

ÍNDICE

LISTADO DE FIGURAS 4
RESUMEN 5
CAPÍTULO 1. INTRODUCCIÓN 6
1.1 OBJETIVO GENERAL 6
1.2 OBJETIVOS ESPECÍFICOS Y METAS 6
CAPÍTULO 2. ESTADO DEL ARTE 7
2.1 A DEEP LEARNING BASED APPROACH TO AUTOMATED ANDROID APP TESTING [1] 7
2.2 DRIFT: DEEP REINFORCEMENT LEARNING FOR FUNCTIONAL SOFTWARE TESTING [2] 7
2.3 CODELESS WEB TESTING USING SELENIUM AND MACHINE LEARNING [3]8
2.4 DEEP REINFORCEMENT LEARNING IN AUTOMATED USER INTERFACE TESTING [4] 8
2.5 MARCO TEÓRICO 9
2.5.1 Aplicaciones Web 9
2.5.2 Calidad y pruebas en aplicaciones web 10
2.6 MOTIVACIÓN 15
2.7 ALCANCES Y LIMITACIONES 15
CAPÍTULO 4. METODOLOGÍA DE DISEÑO DEL SISTEMA 16
4.1 DEFINICIÓN Y ACOTAMIENTO DE LA APLICACIÓN WEB ESTÁNDAR 17
4.1.1 Tipos de aplicaciones web [15] 17
4.1.3 Restricciones Técnicas 23
Automatic Web Browsing 23
4.1.4 Definición y acotamiento del tipo de aplicación web estándar y pruebas 25
TIPO DE APLICACIÓN WEB: MULTI-PÁGINA 25
4.2 ANÁLISIS Y DEFINICIÓN DEL PROBLEMA DE PRUEBAS AUTOMATIZADAS 26
4.3 DISEÑO E IMPLEMENTACIÓN DEL SISTEMA 30
4.3.1 Navegación automatizada de la aplicación 30
4.3.2 Aprendizaje por Refuerzo 34
4.4 PRUEBAS 45
4.4.1 Diseño de las pruebas 45
CAPÍTULO 5. RESULTADOS Y CONCLUSIONES 49
5.2 Conclusiones 54
5.3 TRABAJO FUTURO 55
ANEXOS 56
1 IMPLEMENTACIÓN DEL AGENTE Q LEARNING 56
2 IMPLEMENTACIÓN DE MÓDULO CRAWFORMS 57
3 IMPLEMENTACIÓN DE MÓDULO GENERADOR DE ESTADOS 58
BIBLIOGRAFÍA 60

Listado de Figuras

Figura 1 Diagrama de la metodología a seguir 16
Figura 2 Arquitectura de una aplicación web estática 18
Figura 3 Arquitectura de una aplicación web dinámica 19
Figura 4 Diagrama de interacción Cliente-Servidor de una aplicación Single Page 20
Figura 5 Diagrama de interacción Cliente-Servidor de una aplicación Multi Page 21
Figura 6 Ejemplo de formulario de la aplicación web estandar definida 28
Figura 7 Ejemplo de ciclo de vida de una aplicación web estándar 28
Figura 8 Representación de una aplicación web, sus estados posibles validos y erróneos 29
Figura 9 Árbol abreviado para representar el ciclo de vida y estados de una aplicación web 31
Figura 10 Representación HTML de un formulario web 32
Figura 11 Ejemplo del uso de Mechanize para la interacción directa con una aplicación web 33
Figura 12 Representación del funcionamiento de un algoritmo de aprendizaje por refuerzo 34
Figura 13 Representación de un formulario y su representación de estados y acciones 35
Figura 14 Ejemplo de la matriz de entorno 36
Figura 15 Descripción del algoritmo Q Learning 37
Figura 16 Ejemplo de representación de un formulario y sus estados posibles 38
Figura 17 Representación de las combinaciones posibles de estados basadas en un arbol.38
Figura 18 Ejemplo de Matriz de entorno en representación JSON 39
Figura 19 Ejemplo de formulario con sus opciones validas 40
Figura 20 Representación de un camino valido dentro del ciclo de vida de la aplicación web 41
Figura 21 Representación JSON del resultado de caminos que resultan en un error dentro de la aplicación web 42
Figura 22 Interfaz del sistema de pruebas autónomas empaquetado como un framework 43
Figura 23 Representación de la secuencia de estados de un error por medio de un grafo 44
Figura 24 Diagrama de la aplicación modelo de pruebas con sus diferentes recursos y errores establecidos 45
Figura 25 Vista de la interfaz de la aplicación web modelo de pruebas 48
Figura 26 Interfaz del sistema de pruebas autónomas, antes de iniciar la búsqueda de errores. 49
Figura 27 Interfaz del sistema de pruebas autónomas, durante la búsqueda de errores 50
Figura 28 Interfaz del sistema de pruebas autónomas, al finalizar la búsqueda de errores 51
Figura 29 Detalle de un error 500 a través de su representación de estados en un grafo 52
Figura 30 Listado de errores 400 encontrados 53
Figura 31Detalle de un error 400 a través de su representación de estados en un grafo 53

Resumen

Las aplicaciones web hoy en día juegan un papel muy importante en el mundo actual, a raíz de que las empresas y organizaciones comenzaron a adoptar la World Wide Web para ofrecer sus servicios y con la adopción masiva de la internet, las aplicaciones web han cambiado por completo nuestras vidas, la economía y la sociedad en general.

Por lo cual es de gran relevancia garantizar el correcto funcionamiento de estos sistemas, si bien es una práctica estándar la realización de pruebas automatizadas en la industria del software, su implementación y verificación continúan dependiendo de la intervención humana y por lo tanto susceptibles a posibles errores.

Este trabajo propone desarrollar un sistema que permita la realización de pruebas a una aplicación web desde su interfaz web de forma completamente autónoma, auxiliándose en herramientas como lo es el aprendizaje por refuerzo y árbol de búsqueda, sin bien el alcance de este trabajo se encuentra limitado por diferentes factores como el tipo de aplicaciones compatibles y los tipos de errores que son posibles detectar, se introduce una herramienta que puede auxiliar y complementar los proceso de pruebas actuales, aumentando el grado de cobertura de pruebas en una aplicación web y por lo tanto incrementando su calidad.

Capítulo 1. Introducción

Las aplicaciones web forman parte esencial de nuestra vida digital actual, van desde sencillos sitios web de entretenimiento hasta aplicaciones web críticas como servicios bancarios, comercio electrónico, entre otras que demandan una alta calidad que garantice su correcto funcionamiento ya que de lo contrario el fallo de estas puede reflejarse en grandes pérdidas económicas.

Para garantizar el correcto funcionamiento de una aplicación web y en general de todo tipo de software se lleva a cabo las pruebas de software que consiste en un método para verificar si el software cumple con los requerimientos esperados y se encuentra libre de defectos (bugs), las pruebas de software se pueden llevar a cabo a diferentes niveles, con diversas metodologías y de forma manual o automatizada, sin embargo las técnicas de pruebas actuales se encuentran limitadas por diversos motivos y muchas veces son implementadas de manera deficiente por los desarrolladores y equipos de verificación, debido a la complejidad y tiempo necesario para llevar a cabo una verificación completa debido a que al ser pruebas manuales o automatizadas su diseño y ejecución requieren de una inversión de tiempo considerable y deben ser diseñadas a medida para cada software desarrollado, además las aplicaciones web cuentan con algunas características peculiares que las hacen diferentes al software tradicional y que afectan al proceso de pruebas de manera considerable, por lo cual el enfoque de este trabajo es introducir una herramienta que permita disminuir la inversión en tiempo de pruebas y a la par contribuya a aumentar la calidad en el desarrollo de aplicaciones web.

1.1 Objetivo General

Diseño e implementación de un sistema que sea capaz de llevar a cabo pruebas funcionales de aplicaciones web estándar de forma autónoma a partir de la interacción con la interfaz gráfica de la aplicación web, el sistema será capaz de detectar y reportar errores HTTP nativos..

El sistema a desarrollar no pretende reemplazar las metodologías de prueba de software actuales, sino ser un complemento a las mismas, que permita acelerar los tiempos de pruebas y ayudar a los desarrolladores a encontrar casos de prueba que deriven en una falla de que otra forma tal vez no serían detectados.

1.2 Objetivos Específicos y Metas

1. Definición y acotamiento del tipo de aplicación web al que está enfocado el sistema, que es nuestra definición de aplicación web estándar.
2. Investigación detallada de métodos y técnicas de pruebas funcionales de aplicaciones web actuales.
3. Diseño del sistema para la realización de pruebas funcionales autónomas de aplicaciones web estándar, basado en aprendizaje reforzado y otras técnicas de búsqueda.
4. Implementación del sistema diseñado como un framework que puede ser reutilizado para realizar pruebas autónomas de diferentes aplicaciones web estándar.
5. Verificación del funcionamiento correcto del sistema y solución de posibles errores de implementación.
6. Validación de la efectividad del sistema para encontrar errores de forma autónoma.

Capítulo 2. Estado del Arte

En este capítulo se abordará el estado del arte relacionado a las pruebas de aplicaciones web en dos vertientes, los métodos y técnicas de pruebas estándares de la industria los cuales se describen dentro del subcapítulo 2.6 Marco teórico y el estado del arte relacionado de forma directa con un acercamiento similar al propuesto en este trabajo que se pueden encontrar a continuación:

2.1 A Deep Learning based Approach to Automated Android App Testing [1]

Propone Humanoid en enfoque basado en deep learning para la realización automática de pruebas en aplicaciones Android, se basa bajo la premisa de hacer posible aprender de interacciones humanas, el sistema es capaz de generar interacciones artificiales similares a las humanas en base a una interfaz de usuario presentada para realizar pruebas a la aplicación, en este trabajo implementaron una red neuronal profunda para aprender cómo los usuarios interactúan con la aplicación y en base a este aprendizaje construyeron un modelo que genera nuevos casos de prueba que pueden ser aplicados para la realización de pruebas en la aplicación, demostraron que con su modelo es posible realizar pruebas automatizadas para aplicaciones Android, con mayor cobertura y rapidez que otros generadores de casos de prueba tradicionales, sin embargo este trabajo considera algunas limitaciones y trabajo futuro por realizar, por ejemplo cómo mejorar la cobertura de las pruebas ya que a pesar de tener una mejor cobertura que otros generadores de pruebas automáticos aun sigue presentado en general una baja cobertura reportan que algunas aplicaciones es menor a 10% por lo cual proponen como posible solución incorporar asistencia humana para guiar el sistema en estados donde se vuelve muy complicado para el sistema avanzar por sí solo.

Otra limitación encontrada es que su modelo está basado en interacciones humanas reales por lo cual escalar el modelo para aumentar su eficacia requeriría un conjunto de datos muy grande que se vuelve complicado de obtener, por lo cual proponen que el sistema por sí mismo pueda aprender de interacciones no humanas, es decir generadas artificialmente siempre y cuando su calidad sea analizada por algún método, de esta manera se podría escalar y mejorar el modelo sin requerir grandes conjuntos de datos.

2.2 DRIFT: Deep Reinforcement Learning for Functional Software Testing [2]

En este trabajo se propone DRIFT un framework para realizar pruebas automáticas de software basado en aprendizaje reforzado Q-Learning a través de Batch-RL estos algoritmos operan bajo una representación simbólica de la interfaz gráfica y modela la función de valor de acción de estado a través de una GNN (Graph Neural Network), este framework se validó realizando pruebas de Windows 10 y se comprobó que es capaz de ejecutar las funciones deseadas en una manera completamente automatizada, además se lograron realizar tareas simples o combinadas demostrando que el framework puede funcionar de forma eficiente para realizar pruebas de software con un amplio rango de objetivos, una de las mejoras que se proponen en este sistema es agregar un agente de memoria que permita realizar pruebas más complejas donde se tome en cuenta los estados previos y no solo el actual.

2.3 Codeless Web Testing using Selenium and Machine Learning [3]

Este trabajo define e implementa un framework que permite la automatización de pruebas funcionales sin código en una aplicación web, para lograrlo hace uso de ML (Machine Learning) e Inteligencia artificial específicamente SVM (Support Vector Machines) para poder detectar y adaptarse al cambio y en base a ello generar casos de prueba eficientes para la realización de pruebas, es importante destacar que se asiste de Selenium el cual es una suite de herramientas de pruebas para aplicaciones web, sus resultados arrojaron que su sistema puede ser eficiente para llevar a cabo pruebas automáticas en la mayoría de sitios web estándar usando un caso de prueba genérico sin necesidad de adaptarlo a cada caso.

Una limitante en este trabajo y trabajo a futuro, es ampliar los casos de estudio ya que se centra en pruebas de aplicaciones web con interfaces de búsqueda.

2.4 Deep Reinforcement Learning in Automated User Interface Testing [4]

En este trabajo se propuso usar aprendizaje reforzado profundo para realizar la exploración y pruebas automáticas de software, específicamente aplicaciones web en donde se llevan a cabo pruebas funcionales y se analizan errores de comunicación con el backend a través de Javascript, se encontró que es posible representar los estados de la interfaz de usuario como un vector o bien como una imagen con el cual se puede operar los algoritmos de aprendizaje reforzado, a diferencia de otras técnicas basadas en aprendizaje supervisado no se tiene el problema de requerir grandes volumen de datos para lograr el entrenamiento de un modelo eficiente y aún así la generalización no está garantizada, una ventaja de este enfoque es que si el sistema cambia o se prueba uno nuevo y el modelo falla se puede re-entrenar desde cero para adaptarse al nuevo sistema, sin embargo uno de los principales problemas encontrados es que el proceso de entrenamiento para una aplicación real puede tomar días enteros y además no saber con exactitud cuándo parar el entrenamiento para alcanzar una eficiencia óptima por lo cual se usó la experiencia en otros campos como videojuegos para calcular de manera aproximada el tiempo óptimo de entrenamiento.

Como mejoras a futuro para este trabajo se propone lograr que el modelo entrenado pueda generalizarse para funcionar en diferentes aplicaciones dado que su tiempo de entrenamiento es muy extenso o bien encontrar un método que permita reducir el tiempo de entrenamiento de manera considerable para poder aplicarlo en múltiples aplicaciones web de forma eficiente, también proponen cambiar el modelo para mejorar el llenado de campos de texto tomando el cuenta los tipos de datos ya que actualmente se usan texto generados de forma aleatoria, además de mejorar la función de recompensa para centrarse en las partes más importantes de la aplicación, reduciendo el número de estados a probar y en consecuencia mejorar la eficiencia del sistema.

2.5 Marco Teórico

A continuación se exponen algunos conceptos importantes, para entender la naturaleza del problema planteado, así como las metodologías y estándares actuales de pruebas automatizadas en aplicaciones web, además técnicas y herramientas útiles para el desarrollo de este trabajo.

2.5.1 Aplicaciones Web

Cuando surgió la web consistía en un repositorio de contenido estático, lo que quiere decir que el contenido solo se podía visualizar o descargar sin poderlo modificar o interactuar con el sitio web, si un usuario queria publicar en la WWW solo lo podía realizar a través de publicar un documento HTML (Hypertext Markup Language) y despues subirlo a un hospedaje (servidor web) para poder ser accedido a través de la URL del sitio, en este caso el navegador web se encarga de interpretar el documento HTML y mostrar el contenido de forma visual y con sentido para las personas, por lo cual las personas que acceden a dicha URL pueden leer el contenido o visualizar las imágenes que el autor haya añadido al documento, pero si la comparamos con la web actual se puede decir que este enfoque era muy limitado, puesto que la web era un canal de comunicación se un solo sentido como otros medios de comunicación como la televisión o la radio, si el autor del documento HTML deseaba cambiar el contenido entonces tenía que realizar un proceso complejo, modificar el documento HTML y reemplazarlo en el servidor web.

La web estática continuo evolucionando gradualmente hasta llegar a convertirse en la web moderna que conocemos actualmente, uno de los avances fundamentales para lograr la web dinámica fue la integración de un método para lograr hacer las páginas interactivas es decir técnicamente que están fuera construidas en base a la petición que realizara el usuario, este método fue nombrado CGI y tenía la tarea de recibir y interpretar las peticiones HTTP hacia aplicaciones ajenas al servidor web pero que sin embargo se ejecutaban en la misma maquina, pero los CGIs tenían un problema y este era que cada vez qué se recibía una solicitud el servidor web comenzaba un nuevo hilo de proceso para correr el programa CGI y esto representaba una carga pesada para los servidores [5].

Actualmente la solución más usada para desarrollar aplicaciones web eficientes, es el uso de un lenguaje de programación encargado de procesar las peticiones HTTP que son recibidas desde el servidor web y a través de un puente entre el servidor u aplicación que les permite comunicarse, el servidor web se encarga de enviar y recibir las respuestas a los clientes y el lenguaje de programación en un entorno de ejecución de propio de entender y ejecutar el código que representa la lógica del negocio, al dejar aislado el servidor web se evitan problemas de desempeño, esta enfoque moderno ha hecho posible el desarrollo de robustas y escalables aplicaciones web que hoy en día forman parte de nuestra vida diaria y que han cambiado para bien muchas actividades que realizamos desde comunicarnos, comprar, hasta el entretenimiento, ya que todo esto y mucho más hoy lo podemos hacer a través de aplicaciones web desde la comodidad de nuestros hogares.

Una aplicación web puede ser considerada como un sistema distribuido con una arquitectura cliente-servidor o arquitectura multi-capa, que se pone a disposición de los usuarios a través de un navegador web, estas diferencias añaden algunos retos a comparación de las técnicas de pruebas tradicionales por ejemplo las pruebas de aplicaciones web deben considerar algunas variables extras como el desempeño y disponibilidad de la aplicación en diversas condiciones incluyendo cuando son accedidas de forma

concurrente por un gran número de usuarios, también se debe considerar otro tipo de variables como la correcta visualización y funcionamiento en diferentes navegadores web [6].

Además se debe tomar en cuenta la gran variedad de tecnologías disponibles para desarrollar aplicaciones web así como la gran variedad de infraestructura tecnológica existente disponible para alojar las aplicaciones web, esto implica gran diversidad de lenguajes de programación, frameworks de desarrollo, base de datos, sistemas operativos, servidores web, arquitecturas, cortafuegos, balanceadores de carga, CDNs entre otros componentes que pueden estar involucrados en el desarrollo y ejecución de una aplicación web moderna, lo que resulta en una infinidad de combinaciones y configuraciones posibles que se traduce en que cada aplicación web cuenta con condiciones únicas que vuelve prácticamente imposible reutilizar un entorno y casos de pruebas diseñados previamente para otra aplicación web, esto representa a la realización de pruebas una inversión de tiempo y recursos considerable que aunque se trata de un proceso crucial para cumplir con la calidad requerida muchas veces los desarrolladores, organizaciones y empresas no están dispuestos a cubrirlo de forma cabal.

2.5.2 Calidad y pruebas en aplicaciones web

Básicamente las pruebas de aplicaciones web consisten en hacer uso de la aplicación usando diferentes combinaciones de entradas de datos y estados de la aplicación, verificar su respuesta a cada una de estas combinaciones para descartar defectos en la aplicación [7].

Dependiendo del tipo de falla encontrado se puede atribuir a errores durante el desarrollo de la aplicación, al entorno de ejecución donde corre la aplicación e incluso a una combinación de ambos factores ya que una aplicación web se encuentra estrechamente relacionada con su entorno de ejecución por lo cual las pruebas siempre deben considerar esta integración para poder detectar con exactitud de dónde proceden los fallos.

Las pruebas de una aplicación web se puede dividir en 2 grandes grupos, pruebas de requerimientos funcionales y pruebas de requerimientos no funcionales, las primeros involucran la verificación de los servicios y funcionalidad específica de la aplicación [8], es decir el comportamiento directamente relacionado con la lógica del negocio para la que fue desarrollada la aplicación, las segundas involucran pruebas que no tienen que ver con los servicios provistos por la aplicación si no más bien con el nivel de calidad con la que responde la aplicación en diferentes circunstancias.

Para algunos autores la prueba del software ha sido definida como:

- El proceso de analizar un software para detectar diferencias entre condiciones existentes y requeridas y evaluar las características de los elementos del software.
- Es el proceso de analizar un programa con la intención de encontrar errores.

Poder descubrir defectos en el software puede llegar a volver muy complejo, la razón es que el software y cualquier sistema digital no son continuos, por lo cual para verificar determinados errores no es suficiente garantizar sólo el llamado “happy path” que es el ideal esperado que un usuario tendría al usar la aplicación, al contrario se deben verificar todos los posibles valores para garantizar la eficiencia de las pruebas.

El proceso estándar para la realización de pruebas a sistemas de software es el siguiente:

- Diseñar prueba
- Ejecutar prueba
- Identificar problemas
- Resolver problemas

Existen algunos principios que nos guían en la prueba de software, esto con la finalidad de generar una prueba efectiva en los diferentes casos, a continuación se presentan diferentes principios para las pruebas:

- Todas las pruebas deben de responder a los requerimientos del cliente.
- Las pruebas deben de planearse mucho antes de que comiencen la etapa de pruebas.
- El principio de Pareto aplica a la prueba de software, el 80% de todos los errores descubiertos durante la prueba deben provenir del 20% de los componentes del programa.
- La prueba debe comenzar desde lo más pequeño hasta llegar a lo más grande, interpretemos este principio de la siguiente forma. Primero debemos concentrarnos en encontrar errores en grupos integrados por componentes y de forma última el sistema entero.
- Prueba exhaustiva no es posible, nos referimos que el número de permutaciones incluso para programas pequeños es increíblemente grande. Por esta razón es casi imposible ejecutar cada combinación de caminos durante la prueba. Pero es posible cubrir la parte lógica del programa para asegurar que todas las condiciones a nivel de componentes diseñados han sido probadas.
- Para mayor eficacia, la prueba debe de ser dirigida por un tercer equipo independiente.

Es importante para un desarrollador tener en cuenta los objetivos de las pruebas siempre existe una gran posibilidad de encontrar errores, una buena prueba es aquella que tiene altas probabilidades de encontrar errores que no han sido detectados aún y una exitosa prueba es aquella que encuentra errores que ya han sido identificados y aquellos sin encontrar.

Tipos de pruebas en aplicaciones web

Algunos tipos de pruebas no funcionales son [9]:

Pruebas de carga: Evalúa si el desempeño del sistema es el esperado bajo condiciones y número de usuarios que se consideren dentro de parámetros normales.

Prueba de rendimiento: Evalúa el desempeño del sistema en parámetros como tiempo de respuesta, disponibilidad del servicio, bajo diferentes niveles de condiciones con el objetivo de evaluar el desempeño del sistema bajo las diferentes condiciones probadas.

Pruebas de volumen: Evalúa el desempeño y comportamiento del sistema cuando la aplicación maneja un gran volumen de datos.

Pruebas de estrés: Evalúa el comportamiento del sistema llevándolo a condiciones de uso que superan las especificaciones para las que fue diseñado, el objetivo es evaluar el comportamiento del sistema y evaluar si se puede recuperar por sí mismo bajo estas condiciones extremas o picos de uso.

Prueba de fiabilidad: Evalúa si el sistema es confiable, es decir que la aplicación se desempeña de forma continua como es esperado y sin fallas.

Pruebas de usabilidad: Evalúa si el sistema es de fácil aprendizaje y uso para los usuarios humanos.

Pruebas de compatibilidad: Evalúa si el sistema se comporta de forma correcta al ejecutarse en diferentes combinaciones de hardware y software, en el caso de las aplicaciones web se centra en verificar que funcione de la misma manera en diferentes navegadores web.

Pruebas de seguridad: Evalúa la capacidad de defensa del sistema ante intentos de acceso no autorizados y la capacidad de mantener la integridad de la aplicación y sus recursos ante un ataque externo.

Las pruebas funcionales como se describió en párrafos anteriores tienen el objetivo de verificar que los servicios y funciones provistos por la aplicación no presentan fallas y se comportan acorde a los requerimientos bajo los cuales se planeó y construyó la aplicación, cuando se habla de servicios nos referimos a los casos de uso que puede ofrecer una aplicación web por ejemplo alguno de los más comunes son: Registro y autentificación de usuarios, consulta, borrado y escritura en base de datos, operaciones con datos, subida y descarga de archivos estáticos, generación de vistas dinámicas.

Niveles de pruebas

Se refiere a los diferentes alcances que pueden tener las pruebas a realizar y que componentes de la aplicación se enfocarán las pruebas, algunos niveles de prueba comúnmente usados son [10]:

Pruebas Unitarias: Es el nivel más bajo, aquí se prueban componentes individuales es decir que tienen una tarea específica dentro de la aplicación, se valida que el componente funcione de la manera esperada, en el enfoque de las aplicaciones web cabe destacar que debe considerar pruebas unitarias para componentes tanto del lado del servidor como del cliente.

Pruebas de integración: En este nivel se prueba la integración de los diferentes componentes unitarios de la aplicación para validar su correcto funcionamiento cuando interactúan entre sí, de la misma manera que las pruebas unitarias se deben considerar pruebas de integración entre los componentes de lado del servidor, cliente y cliente-servidor.

Pruebas de Sistema: En este nivel se prueba la aplicación cuando todos sus componentes unitarios tanto del lado del servidor y cliente se encuentran integrados de forma completa, el objetivo de estas pruebas es evaluar si el sistema completo cumple con los requerimientos para los cuales fue diseñado.

Pruebas de aceptación: En este nivel las pruebas tienen el objetivo de determinar si el sistema es aceptado o no para su liberación a producción en base a los requerimientos y nivel de calidad.

Estrategias de Prueba
Consiste en el enfoque a usar para llevar a cabo las pruebas, los enfoques más comunes son [5]:

Pruebas de caja negra
Esta estrategia de pruebas se caracteriza por no requerir conocimiento de cómo está construido el sistema a probar, se enfoca solo en tratar al sistema como una caja negra al que se le proporcionan entradas y se analizan sus salidas para determinar si es la salida esperada para la entrada proporciona de acuerdo a los requerimientos del software, uno de los principales problemas de este enfoque es que depende totalmente de los estados de la aplicación y las entradas proporcionadas.

Pruebas de caja blanca
Esta estrategia a diferencia de las pruebas de caja negra, si se conoce la estructura interna del sistema y además se tiene acceso al código del sistema, por lo cual se concentra regularmente en pruebas internas del sistema, las pruebas se enfocan regularmente en:

- Detectar posibles agujeros de seguridad
- Detectar flujos estructurados deficientemente o rotos
- Analizar salidas para ciertas entradas
- Integridad de ciclos de condición
- Fugas de memoria
- Pruebas unitarias
-

Pruebas de caja gris
Esta estrategia combina las pruebas de caja negra y caja blanca, su objetivo es realizar la verificación de unidades del sistema o el sistema completo siguiendo un flujo de prueba de entrada-salida, pero teniendo como auxiliar el conocimiento del sistema y acceso al código de la aplicación.

Cobertura de pruebas
La cobertura de pruebas monitorea el número de pruebas que han sido ejecutadas, su reporte nos provee de información sobre las partes del software donde han sido implementadas pruebas [11].

La cobertura también puede evaluarse a través de diferentes tipos de pruebas. Sin embargo, el tipo de pruebas que se debe correr depende de las prioridades del cliente del equipo de prueba y de la organización detrás de ellos.

Algunos de los mecanismos de cobertura de pruebas son:

- Unit Testing, se ejecuta a nivel modular.
- Functional Testing, se prueban las funciones con características según los documentos FRS(Functional Requirement Specification)
- Acceptance Testing, determinar cuáles de los productos es apropiado para comercializarlo y uso público, en esta etapa los desarrolladores deben de recibir aprobación por quienes hacen las pruebas.
- Integration Testing, estas pruebas son llevadas a cabo cuando los módulos del software están completamente integrados.

Ventajas de la cobertura de pruebas:

- Informa sobre las partes del código que no han sido cubiertas por las pruebas de casos necesarias.
- Ayuda a detectar las áreas de prueba de casos que no son necesarias para el proyecto actual.
- Ayuda a los desarrolladores al crear casos de prueba como se requiera, para garantizar que la cobertura de prueba sea máxima.
- Ayuda a prevenir defectos de fuga.

Generación de casos de prueba

Un caso de pruebas es un conjunto de especificaciones, incluyendo condiciones específicas y variables bajo las cuales debe ser probado un sistema de software, puede aplicar en diferentes niveles (pruebas unitarias, de integración, de sistema o de aceptación) y también con diferentes estrategias (caja negra, caja blanca o caja gris), de forma general un caso de prueba consiste en los siguientes elementos, descripción del caso de prueba (cual es el objetivo de su prueba), pasos que se tienen que llevar a cabo durante la prueba, resultado esperado y resultado obtenido [12].

Algunas técnicas tradicionales para la generación de casos de prueba son las siguientes:

- Orientado a metas: este enfoque consiste en la generación de casos de prueba orientado a metas, por ejemplo cubrir un módulo o función en particular, en este caso el proceso de ejecución no es primordial pero sí verificar la meta que es el objetivo principal.
- Aleatorio: En este enfoque se generan casos de prueba en base suposiciones de errores que pudieran llevar a revelar fallas del sistema.
- Basada en especificaciones: este enfoque genera casos de prueba basados en las especificaciones formales para las cuales fue construido el sistema.
- Basada en código fuente: este enfoque de generación de casos toma como base el código fuente y sigue la ruta natural del flujo de ejecución del código y los casos de prueba se generan en consecuencia.
- Basado en modelos: consiste en la generación de casos de prueba tomando como base la representación del sistema basada en un modelo.

Como se ha expuesto hasta ahora los enfoques tradicionales de pruebas se centran en diferentes enfoques donde los casos de prueba son generados de manera manual por un experto en el sistema y sus requerimientos, su generación puede o no basarse en alguna metodología que optimice o reduzca el número de casos de prueba, para posteriormente ser ejecutados de manera manual o bien de forma automatizada con ayuda de las herramientas adecuadas que lo permitan.

El enfoque de este trabajo tiene objetivo brindar una herramienta para la realización de pruebas en aplicaciones web con la mínima intervención humana posible para reducir tiempos, costos y minimizar la posibilidad de errores, el principal proceso que tradicionalmente requiere intervención humana es la generación de los casos de prueba, a continuación se listan algunas que se usan para generar casos de prueba de manera automatizada:

- Generación aleatoria de datos de prueba
- Generación de casos de prueba simbólica
- Generación de pruebas estáticas
- Generación dinámica de datos de prueba
- Generación de casos de prueba basada en Ingeniería del software basada en búsqueda.

2.6 Motivación

Como se ha expuesto en los párrafos anteriores, las pruebas durante el proceso de desarrollo de software y en especial de aplicaciones web son cruciales para lograr obtener un producto de calidad y robusto, sin embargo también es conocido por ser un proceso tardado y costoso [13].

Actualmente el estado del arte de las pruebas automatizadas brindan cierta eficiencia durante el proceso de verificación de un software, sin embargo conllevan ciertos problemas y limitantes, por ejemplo los scripts de prueba son diseñados y codificados de forma manual sin embargo son susceptibles a cualquier cambio del software a probar por lo cual se deben actualizar en todo momento, no se pueden reutilizar en otras aplicaciones, sin dejar de mencionar que se encuentran limitados a los casos de prueba que el desarrollador considero por lo cual se pueden estar omitiendo estados importantes de la aplicación que no serán validados.

Con respecto a los trabajos similares se puede ver que es viable poder realizar un sistema de pruebas completamente automatizadas, sin embargo aún existen métodos por resolver como lo es lograr la interacción adecuada con los controles de una aplicación y aumentar la cobertura de pruebas.

Por otro lado las pruebas manuales son realizadas por personas, haciendo clic en la aplicación o interactuando con sus API's a través de las herramientas adecuadas, lo cual es muy costoso en tiempo y recursos, ya que requiere que alguien configure un entorno y ejecute las pruebas por sí mismo, además que puede ser propenso a errores humanos, ya que el evaluador puede cometer errores tipográficos u omitir pasos en el proceso de prueba.

Se calcula que aproximadamente el 50% del presupuesto de desarrollo de un nuevo software puede ser destinado a pruebas, sin embargo también es muy caro no realizarlas, se estima que el impacto a la economía debido a defectos de software en Estados Unidos ronda los $59.5 mil millones de dólares anualmente [14].

Si bien este trabajo no pretende reemplazar las metodologías de pruebas de software actuales, si se espera que sea un complemento a las mismas y que permita acelerar los tiempos de pruebas y ayudar a los desarrolladores a encontrar errores que de qué otra forma tal vez hubieran pasado por alto, por lo cual este sistema pretende aportar un impacto socioeconómico positivo al reducir los tiempo de pruebas y aumentar la calidad de las aplicaciones web desarrolladas.

2.7 Alcances y Limitaciones

Parte de los objetivos de este trabajo se encuentra el acotamiento del tipo de aplicación web al cual están enfocadas las pruebas automatizadas, puesto que debido a la naturaleza y tecnologías involucradas en el desarrollo web moderno es complicado lograr una herramienta que sea capaz de funcionar con cualquier tipo de aplicación web moderna, es por ello que ha definido de forma específica qué tipo de aplicaciones web son soportadas y bajo qué condiciones.

Además como se expone en páginas siguientes, el tipo de pruebas que se pueden aplicar a una aplicación web son múltiples tanto en alcance como en método, por lo cual también se ha limitado y acotado el tipo de prueba a realizar y errores que pueden ser encontrados con el sistema desarrollado.

Capítulo 4. Metodología de Diseño del Sistema

A continuación se describirá la metodología llevada a cabo para el cumplimiento de las metas y objetivos propuestos en este trabajo:

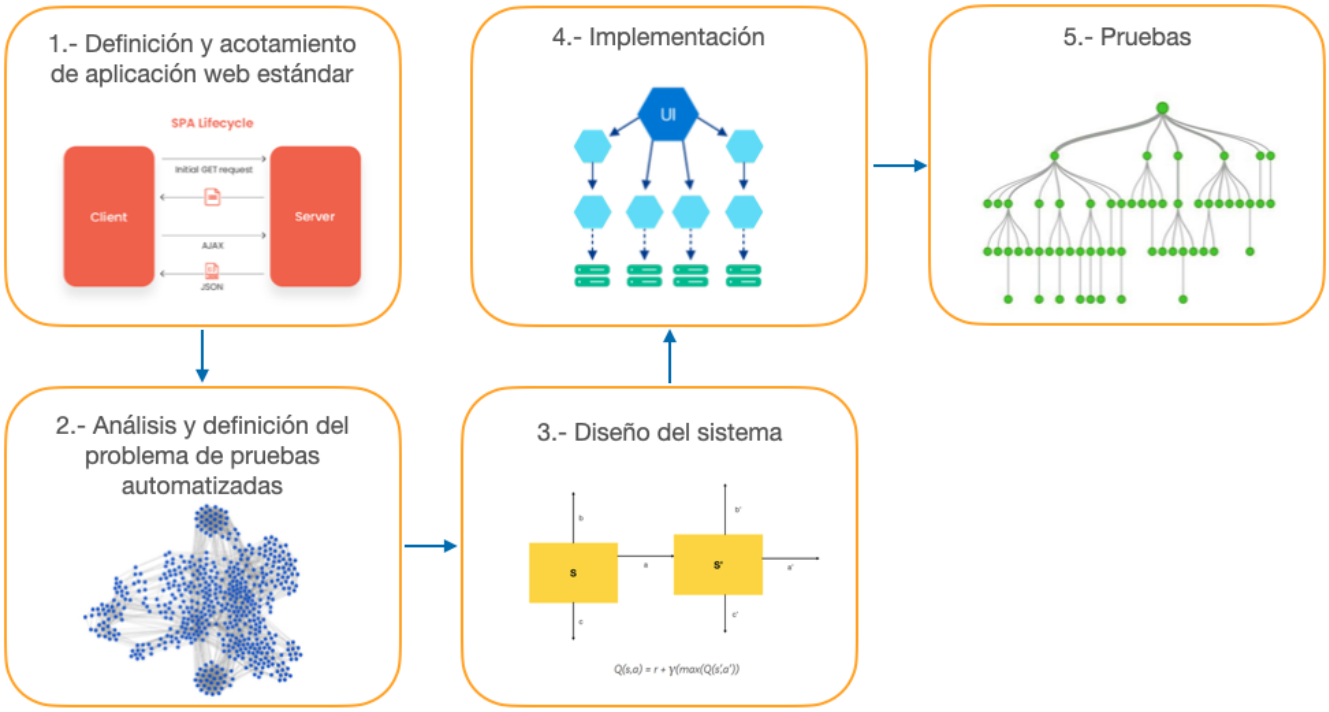


Figura 1 Diagrama de la metodología a seguir

Definición y acotamiento de la aplicación web estándar
En la web moderna existen diversos tipos de aplicaciones web, cada uno de ellos con características propias como arquitectura, tecnología de desarrollo, estándares, etc. que las hacen diferentes unas de otras, por lo cual para hacer viable el cumplimiento de los objetivos principales de este trabajo es necesario definir y acotar el tipo de aplicación web con la cual se validará el funcionamiento correcto del sistema desarrollado bajo las características y condiciones del tipo de aplicación web seleccionada.

Análisis y definición del problema de pruebas automatizadas
Una vez definido y acotado el tipo de aplicación con la cual se trabajará es necesario concentrarnos en los problemas y retos a resolver para cumplir con la objetivos propuestos respecto a la realización de pruebas automatizadas bajo las restricciones planteadas, para ello es necesario identificar los retos existentes y proponer los métodos requeridos para solucionarlos.

Diseño del sistema
Después de identificar los problemas específicos a resolver para cumplir con nuestros objetivos, además de haber propuesto los métodos y técnicas para su solución, el siguiente paso es el diseño del sistema donde se describe la forma en que se plantea la integración y funcionamiento de todos los módulos y

herramientas propuestas que en su conjunto nos llevarán a determinar el cumplimiento de los objetivos de este trabajo.

Implementación
En este apartado se describe el proceso de implementación de los módulos y herramientas propuestos, describiendo a detalle cómo se soluciono los problemas y retos a nivel técnico.

Pruebas
Finalmente es necesario verificar el funcionamiento correcto del sistema y solucionar posibles errores de implementación, además de diseñar un caso de prueba para poder validar la efectividad del sistema para encontrar errores de forma autónoma y evaluar su efectividad con respecto a los valores esperados de la prueba diseñada.

4.1 Definición y acotamiento de la aplicación web estándar

4.1.1 Tipos de aplicaciones web [15]

La web moderna (World Wide Web) está compuesta por una inmensa red de contenido digital y servicios los cuales consumimos diariamente como parte de nuestras vidas sin embargo todos estos servicios y contenido se encuentran basados como ya se describió en párrafos anteriores en el protocolo HTTP/HTTPS el cual como su nombre lo indica (Hypertext Transfer Protocol) nos permite realizar peticiones de datos y recursos a través de simples verbos como GET, POST, PUT, etc. y recibir una respuesta a la petición solicitada, todo esto realizado comúnmente desde un navegador web.

A estos conjuntos de peticiones y respuestas vía el protocolo HTTP/HTTPS se les conoce como aplicaciones web o comúnmente “sitios web” para referirnos a un tipo específico, a continuación se describen los 2 tipos de aplicaciones web existentes:

Aplicaciones web estáticas

Este es fue el primer tipo de aplicación web cuando surgió la web moderna, básicamente consiste en una combinación de documentos HTML, CSS, Javascript y archivos multimedia como imágenes, audio, video, los cuales son almacenados como archivos estáticos en un servidor desde donde son servidos a los usuarios que los solicitan a través de su URL la cual es solicitada comúnmente desde un navegador web (Cliente) a través del método GET de HTTP/HTTPS, por lo cual la única tarea que debe realizar el servidor web es identificar el recurso solicitado dependiendo la URL visitada y enviarlo de regreso al navegador web vía una respuesta HTTP.

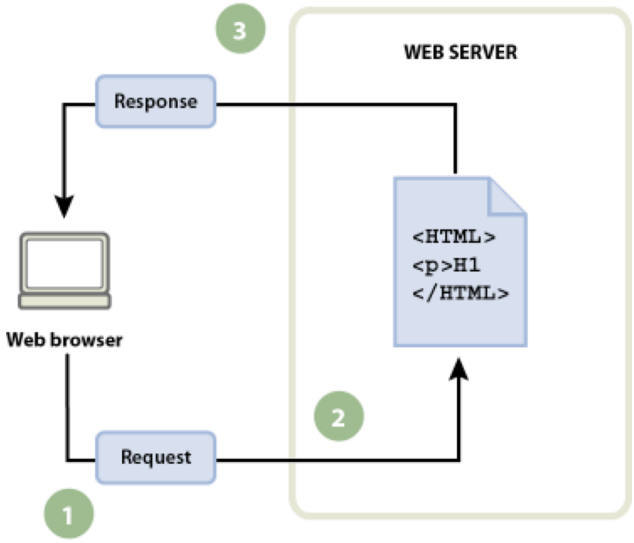


Figura 2 Arquitectura de una aplicación web estática

En resumen una aplicación web estática o comúnmente llamado sitio web estático tiene la particularidad de que el contenido solicitado será siempre el mismo para todos los usuarios ya que los archivos estáticos almacenados en el servidor se sirve sin ningún cambio hacia los usuarios, si bien actualmente este tipo de aplicaciones web ya son muy escasas, siguen existiendo sobre todo en aplicaciones donde solo se requiere presentar información a los usuarios sin tener ningún tipo de personalización o interactividad, ya que a pesar de sus limitaciones tiene la ventaja de ser una solución, ligera, rápida y económica.

Aplicaciones web dinámicas

Una aplicación web dinámica consiste en un enfoque más complejo, además de estar basadas en las mismas tecnologías del lado del navegador que permiten brindar a los usuarios una representación visual a través de documentos HTML, CSS, JS y elementos multimedia, la diferencia radica en que las respuesta a las peticiones de los usuarios no se encuentran almacenadas en archivos estáticos como sucede en la web estática, en este caso los respuestas dependen de múltiples factores del lado del cliente y el servidor que al combinarse hacen posible brindar una respuesta personalizada al usuario.

En las aplicaciones web estáticas el servidor recibe puede recibir una petición del usuario no solo solicitando un archivo en específico sino que además es capaz de enviar una serie de parámetros extras a través de otros métodos HTTP como POST y con ello generar una respuesta que pueda ser interpretada por el navegador web después de realizar múltiples acciones que generalmente representan la lógica del negocio para la cual la aplicación web fue diseñada, estas operaciones son ejecutadas del lado del servidor y son definidas a partir de un lenguaje de programación (por ejemplo PHP, Python, Ruby, Java, C#, etc.) y además frecuentemente también intervienen una serie de consultas a una base de datos tanto de escritura como lectura, todo esto hace posible brindar una respuesta dinámica y personalizada a los usuarios, por lo cual a este tipo de páginas se le conoce también como procesadas del lado del servidor (server-side processing).

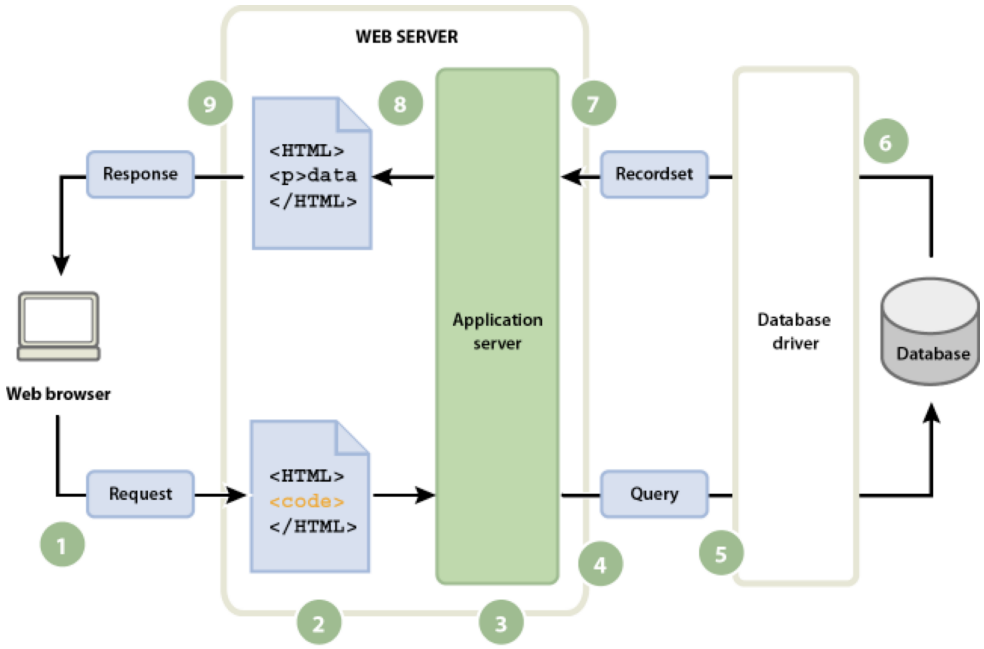


Figura 3 Arquitectura de una aplicación web dinámica

En conclusión, la interactividad y personalización de las aplicaciones web dinámicas hacen posible ofrecer servicios complejos, es por ello que actualmente todo servicio que se brinde a través de internet se encuentra basado en este paradigma.

4.1.2 Single Page Applications y Multi Page Applications [16]

Las aplicaciones web se han convertido en una gran alternativa a las aplicaciones de escritorio tradicionales ya que cuentan con diferentes ventajas por ejemplo, el mantenimiento y actualización no requiere intervención del usuario y una de las más importantes ventajas es que una aplicación web no se encuentra ligada a un dispositivo específico ya que su acceso se lleva a cabo a través de internet por medio de un navegador, se puede acceder prácticamente desde cualquier dispositivo que cuente con una conexión a internet y un navegador web.

Debido a estas ventajas incluso muchas aplicaciones que comenzaron como aplicaciones de escritorio tradicionales se están moviendo a una alternativa web, sin embargo muchas de estas aplicaciones web intentan simular la experiencia de usuario de una aplicación tradicional, este tipo de aplicaciones reciben el nombre de Single Page Applications, mientras que las aplicaciones web que conservan el enfoque tradicional de la web donde cada recurso es representado por una página se les conoce como Multi Page Applications, a continuación se describen más detalles sobre cada uno de estos tipos:

Single Page Applications

La característica principal de una Single Page Application (SPA) es que a pesar de funcionar dentro de un navegador web tradicional esta no requiere de recargar o cambiar de página durante su uso, todo el flujo de la aplicación sucede dentro de la misma pantalla de forma interactiva, es decir trata de simular la interacción tradicional que se podía ver en una aplicación de escritorio, sin tiempos de espera o recargas de página.

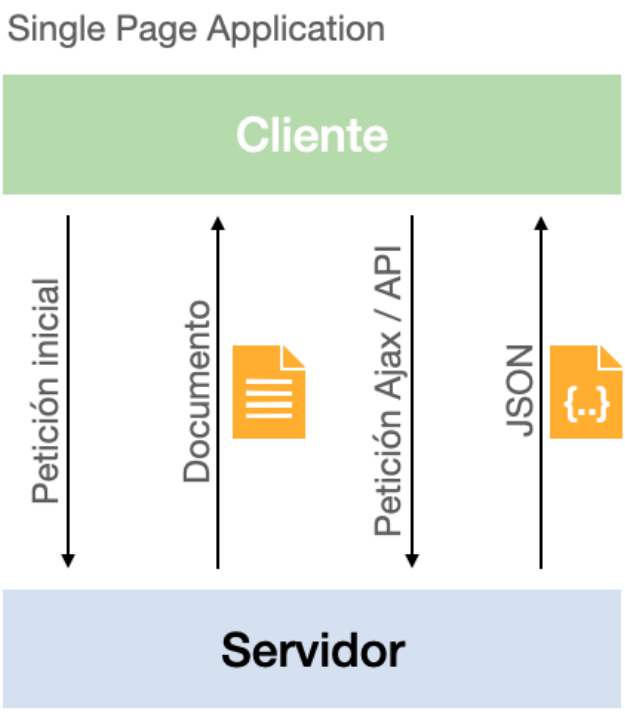


Figura 4 Diagrama de interacción Cliente-Servidor de una aplicación Single Page

Para poder lograr imitar este flujo interactivo las Single Page Application se valen del uso extensivo de Javascript para poder manipular la interfaz de usuario en tiempo real reaccionando a las peticiones del usuario de forma fluida, para lograr esto una SPA requiere de desligar la capa de datos del renderizado de la interfaz de tal modo que las peticiones de datos sean independientes la presentación de la interfaz, para ello es común representar cada función de la aplicación como un componente el cual cuenta con sus propios controles en la interfaz y se comunica con una función independiente del lado del servidor para cubrir el requerimiento de datos.

Del lado técnico, es común que este tipo de aplicaciones hagan uso de algún framework del lado de la interfaz como React.js, Ember.js, Angujar, etc y la capa de datos se encuentra separada y consumida a través de un API REST, aunque también es posible hablar de un SPA si se usa tecnología AJAX (Asynchronous JavaScript and XML) la cual es una tecnología que permite desacoplar la capa de datos del renderizado del documento HTML sin necesidad de usar un framework especializado, siempre y cuando se logre un desacoplamiento total de los datos y la presentación.

Multi Page Applications

Las aplicaciones web multi-página (MPA Multi Page Applications) funciona de la forma tradicional con la que la web fue diseñada, es decir siguiendo el modelo request-response en donde el cliente realiza una petición al servidor y este a su vez responde con un documento HTML identificado por una nueva URL, sin embargo a diferencia de una aplicación web estática el documento enviado por el servidor se encuentra personalizado de acuerdo a la petición solicitada por el cliente, una MPA que siga completamente este enfoque renderiza un documento nuevo por cada petición realizada por el cliente lo cual vuelve a las MPA muy extensas.

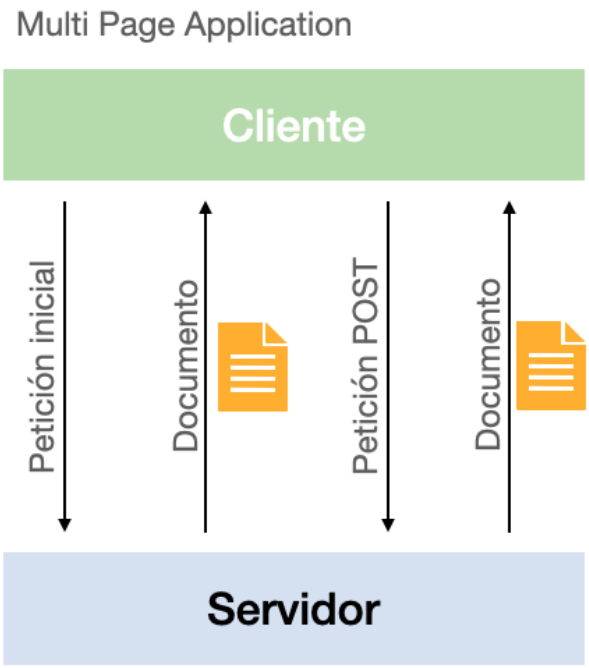


Figura 5 Diagrama de interacción Cliente-Servidor de una aplicación Multi Page

Sin embargo también existen enfoques de MPA en donde se hace uso de AJAX para desacoplar algunas funciones mejores de la aplicación y solo actualizar una sección específica de la interfaz sin tener que recargar toda la página completa y solo recargando la página cuando los cambios a realizar a la interfaz son mayores, este es un modelo híbrido que toma lo mejor de las SPA y MPA.

SPA VS MPA

Cada uno de estos enfoques de desarrollo de una aplicación web tiene sus ventajas y desventajas que a continuación se describen y que en nuestro caso tomamos en cuenta para la selección de nuestro tipo de aplicación web estándar en el que se encuentra centrado este trabajo.

SPA Ventajas:

En general tienen un mayor rendimiento, ya que solo en la primera petición se recibe todos los recursos estáticos necesarios para la construcción de la interfaz, las siguientes peticiones son mucho más ligeras ya que solo se transmiten datos.

Mejor experiencia de usuario, ya que al emular una aplicación de escritorio tradicional la interacción es más fluida y “natural”.

Desarrollo y mantenimiento más eficiente, debido a que es posible desacoplar la interfaz gráfica de la capa de la lógica y datos del negocio, su desarrollo se puede dividir más fácilmente a diferentes grupos de desarrolladores lo que al mismo tiempo hace su mantenimiento más sencillo.

SPA Desventajas:

Problemas de posicionamiento orgánico en buscadores, debido a su naturaleza en donde todo el ciclo de vida de la aplicación web SPA reside en una página única es complicado para los buscadores indexar el contenido de las mismas y por lo tanto tienden a tener un posicionamiento en buscadores limitado, sin embargo esto no es problema si la aplicación se encuentra enfocada a una aplicación privada es decir que se requiera autenticación para su uso

Problemas de rendimiento, si no se siguen buenas prácticas durante el desarrollo de este tipo de aplicaciones es posible caer en problemas de optimización que reduzcan el rendimiento de la aplicación web, sobre todo al tener un código base muy pesado lo que haga la carga inicial muy tardada o bien si se presentan inconsistencias en el desarrollo pueden existir fugas de memoria lo cual resulta en un rendimiento general de la aplicación muy pobre.

Dependiente totalmente de soporte Javascript, debido a que para el desarrollo de este tipo de aplicaciones web se requiere el uso extensivo de Javascript u algún framework basado en este lenguaje, es esencial que el navegador web donde se ejecute tenga soporte completo de JS de otra forma la aplicación web será inutilizable, esto generalmente ya no es un problema sin embargo pueden existir casos donde los clientes no tengan acceso a un navegador con soporte JS total debido al bajo rendimiento del dispositivo.

MPA Ventajas

Buen rendimiento SEO y rastreable por robots, debido a su naturaleza en donde el ciclo de vida de este tipo de aplicaciones se compone de múltiples páginas cada una con su propia URL y contenido, su rastreo por robots es muy accesible por lo cual los buscadores web y otras herramientas pueden acceder a su contenido de manera fácil y realizar una indexación del mismo.
Claridad en el mapa de interacción de la aplicación, debido a que cada página en aplicación MPA representa el resultado de una o un conjunto de unas pocas peticiones específicas el entendimiento del flujo de la aplicación suele ser más sencillo de desarrollar y entender por parte de los usuarios.

MPA Desventajas

La experiencia de usuario con este tipo de aplicaciones suele ser menor con respecto a una SPA ya que el usuario debe navegar a través de diferentes secciones para acceder a diferentes funciones y esperar además la carga de cada una de estas nuevas secciones.

El desarrollo y mantenimiento suele ser más complejo y tardado, debido a que la capa de la lógica del negocio y datos se encuentra muy integrada con la interfaz de usuario, los equipos de desarrollo deben estar muy bien coordinados para el desarrollo y mantenimiento ya que de otra forma el surgimiento de errores puede ser muy fácil.

4.1.3 Restricciones Técnicas

Automatic Web Browsing

Una parte esencial a considerar en este trabajo es que es necesario interactuar de forma automatizada con las aplicaciones web para poder operar sus controles y analizar la respuesta que brindan, por lo cual es fundamental considerar el tipo de aplicación que sea amigable con estas especificaciones.

Para ello es necesario entender a nivel técnico como es posible interactuar con los diferentes tipos de aplicaciones web que se han descrito en párrafos anteriores, por ejemplo para el caso de las SPA en donde la renderización de los controles y contenidos se llevan a cabo a nivel del cliente, si se requiere interactuar de forma automatizada es necesario contar con un navegador web intermediario que sea capaz de interpretar la respuesta de la SPA después de una petición.

Si bien existen herramientas que permiten llevar a cabo la interacción automatizada con las aplicaciones SPA es necesario una capa más de complejidad la cual tiene la tarea de reproducir el estado actual de la interfaz de la aplicación, lo cual se puede lograr a través de un controlador de un navegador web tradicional como es el caso de Selenium y Selenium Web Driver, sin embargo en el caso de este enfoque se presentan diversos problemas como un set de funciones limitados a la herramienta usada y un rendimiento bajo al tener que interactuar con un navegador real a través de un intermediario y además el acceso a los códigos de error de forma programática no es posible directamente se requiere la adición de un Proxy lo cual añade una capa extra de complejidad.

Otro enfoque posible para la interacción con una SPA es el uso de un navegador headless como Phantom JS el cual es un navegador web que es capaz de construir el documento de la interfaz gráfica pero solo en memoria sin representarlo visualmente, lo cual lo hace más eficiente, sin embargo aún se conserva la capa extra de interacción con el navegador reduciendo su eficiencia y nos encontramos restringidos en cuanto las operaciones brindadas por el API de esta herramienta.

Con respecto a la navegación automatizada de una MPA el panorama es diferente ya que en este tipo de aplicaciones web como se describió anteriormente el procesamiento y renderización del documento es realizado del lado del servidor por lo cual el cliente solo se debe encargar de interpretarlo visualmente, con este tipo de aplicaciones igual podemos usar los mismos métodos de navegación automatizada que con una SPA y adicionalmente se abre un nuevo enfoque el cual es el la interacción directa con la aplicación a través de interpretar el código HTML de la respuesta del servidor de forma programática sin necesidad de usar un navegador web intermediario, esto es posible ya que el código HTML contiene toda información requerida para poder interactuar con el.

Un ejemplo de herramienta que puede llevar a cabo esta tarea es la librería mechanize el cual se describe como “navegación web programática con estados en Python”, la cual permite navegar e interactuar una página a través de llenar formularios y dar clic a elementos de forma automatizada, esta herramienta además permite acceder al código de respuesta HTTP directamente sin necesidad de usar otra herramienta adicional, lo cual permitirá diferenciar las peticiones exitosas frente las erróneas, la forma en que funciona mechanize es interactuando con el HTML del recurso y enviando peticiones y respuestas HTTP

el cual hace uso de una serié de métodos para definir sus peticiones a continuación se describe cada método y su función:

OPTIONS
Este método tiene como función solicitar al servidor información acerca de las opciones de comunicación disponibles en las peticiones y respuestas, le permite determinar al cliente los requerimientos asociados con un recurso o las capacidades del servidor sin tener que iniciar la solicitud de un recurso.

GET
Solicita un recurso al servidor que es identificado por una URL de petición.

HEAD
Este método es similar al método a GET a excepción que solo devuelve meta-información contenida en la cabecera de la respuesta HTTP y no en el cuerpo del mensaje.

POST
Este método es usado para solicitar que el servidor acepte información que se encuentra en el cuerpo de la petición.

PUT
Este método solicita al servidor que se almacene un recurso enviado en el cuerpo del mensaje.

DELETE
Solicita al servidor que se elimine el recurso identificado en la URL de petición.

TRACE
Este método permite al cliente determinar que está siendo recibido al otro lado de la conexión y usar la información para un diagnóstico.

Códigos de estado de respuesta HTTP

Los códigos de estado de respuesta HTTP (HTTP Status Codes) son una parte fundamental del protocolo HTTP ya que tienen la tarea de identificar qué resultado tuvo una petición de forma precisa, existen diferentes tipos de códigos que según el dígito con el que inicia el código se puede saber la naturaleza del mismo, a continuación se describen los tipos de códigos definidos:

1xx Respuestas informativas: la petición fue recibida y continúa su procesamiento.
2xx Respuestas satisfactorias, la respuesta fue recibida y procesada correctamente, así como aceptada
3xx Respuestas de redirección, se indica que la solicitud tiene más de una posible respuesta que debe ser seleccionada por el cliente o bien un recurso ha sido movido.
4xx Errores por parte del cliente, se puede deber a una sintaxis errónea u otro motivo del lado del cliente.
5xx Errores del lado del servidor, se indica que un error del lado del servidor ha sucedido aunque la petición sea válida.

De estos tipos de errores en este trabajo nos enfocamos en los 3 más comunes que nos permitirán abarcar los casos más comunes de respuestas en una aplicación web:

200 OK: Nos indica que la petición fue correctamente aceptada y procesada.
404 Not Found: Nos indica que el recurso solicitado por el cliente no se pudo encontrar o no existe.
500 Internal Server Error: Nos indica que sucedió un error del lado del servidor la cual no se sabe cómo manejarla.

4.1.4 Definición y acotamiento del tipo de aplicación web estándar y pruebas

Para poder definir y acotar el tipo de aplicación web con la se trabajará para cumplir los objetivos de este proyecto fue necesario analizar la información expuesta en los párrafos anteriores de tal forma que se obtenga un tipo de aplicación web que sea el marco para validar las metas propuestas, puesto que de dejar abierto nuestro sistema para poder trabajar con cualquier tipo de aplicación web sería inviable.

A continuación se describen los puntos considerados que en su conjunto limitarán el tipo de aplicación con la que se trabajara.

Tipo de Aplicación Web: Multi-Página

Como se describió en párrafos anteriores una MPA a pesar de tener algunas desventajas sobre todo respecto a la experiencia del usuario durante su uso, aún este tipo de aplicaciones se encuentra presente en un gran número de aplicaciones web en el mercado sobre todo en aplicaciones web tradicionales donde el servicio brindado lleva una serie de pasos para completarlo como es el caso de tiendas en línea, portales de reserva, portales de consulta de información, entre otros, además este tipo de aplicación nos da la ventaja de poder interactuar directamente con su código HTML lo que hace más eficiente su navegación automatizada y su estructura basada en URLS para cada uno de sus recursos nos permitirá controlar mejor el flujo de acceso a cada uno de estos recursos.

Por lo cual estas limitaciones las podemos resumir en los siguientes puntos:

- Es necesario tener acceso al código HTML para poder automatizar eficientemente el proceso de pruebas, se debe lograr detectar e interactuar con los controles HTML de la aplicación, sin requerir un intérprete intermediario.
- Es necesario conocer la URL del recurso al que se está accediendo.
- Se busca evitar la interacción con métodos AJAX debido a su asincronía es necesario esperar a la respuesta AJAX para continuar las pruebas, pudiendo afectar significativamente el flujo de entrenamiento y pruebas, lo cual significa que quedan fuera las aplicaciones web MPA híbridas
- Controles HTML Estándar: Además es necesario que la aplicación web haga uso de los estándares definidos por la W3C para el desarrollo web, para que sea más sencillo detectar los controles de los formularios e interactuar con la aplicación web de forma automatizada.
- Uso básico de Javascript: Solo quedará reservada para usos básicos que contribuyan a la mejora de la experiencia del usuario sin interferir en el núcleo de la aplicación (no AJAX)

Alcance del tipo de pruebas a realizar por el sistema:

El tipo de pruebas que es capaz de llevar a cabo el sistema desarrollado a las cuales está restringido este trabajo son pruebas de integración a través de la interfaz de usuario de la aplicación web, es decir que cumplirán con los siguientes puntos:

- El tipo de pruebas contempladas se basa en la interacción con la interfaz de la aplicación web.
- La aplicación web se encuentra directamente conectada con el backend de la aplicación.
- Es posible a través de la interacción con la interfaz y las respuestas recibidas detectar errores provenientes del backend de la aplicación, para detectar estos errores nos centraremos en las respuestas HTTP básicas:

  - 200: Operación exitosa
  - 404: Recurso no encontrado
  - 500: Error en el servidor

- Se llevá a cabo un enfoque de pruebas de caja negra
- No se toma en cuenta cómo está construido el backend
- Solo se interactúa con la GUI (Interfaz de usuario) de la aplicación

4.2 Análisis y definición del problema de pruebas automatizadas

Una vez definido el tipo de aplicación web estándar y el tipo de pruebas con las que se trabaja, a continuación se plantea el problema específico que se tiene que resolver, así como algunos acercamientos técnicos que nos permitirán realizar un diseño del sistema centrado en el problema a resolver.

Una aplicación web estándar con las restricciones definidas para este trabajo consiste en una serie de páginas (recursos) los cuales como toda aplicación web están conformados por una estructura HTML en conjunto con otros elementos para dar estilos e interactividad a la interfaz como CSS y Javascript.
En nuestro caso el elemento principal en el cual nos concentramos es el HTML debido a que es especificación que dice cómo deben ser desplegadas las páginas web por cualquier navegador ya sea de una computadora de escritorio, laptop, tablet, móvil o cualquier otro dispositivo que posea un navegador web.

En los inicios de la web la única información que podía ser transmitida era texto por lo cual no se requería de un formato o estructura especial al texto, sin embargo cuando fue posible transmitir otro tipo de contenidos como imágenes, sonido y video, fue necesario agregar marcadores para poder dar formato las páginas web con este nuevo contenido multimedia.

Dentro de la especificación HTML existe un elemento de gran importancia para una aplicación web y para los objetivos de este trabajo, nos referimos a los formularios, estos elementos son usados de diferentes maneras, desde agregar productos a un carrito de compra hasta hacer posible dejar un comentario en un sitio web. Estos elementos en algunos casos son necesarios para continuar probando y descubriendo más páginas que son solo accesibles enviando un formulario específico.

Formularios
La etiqueta HTML <form> tiene diferentes atributos, como nombre para identificar el formulario y diferenciarlo de diferentes formularios existentes, la acción attribute especifica una URL donde el formulario puede obtener o enviar los datos. El navegador identifica si debe enviar o recibir datos a través del atributo “method”, donde existen dos valores GET o POST, el método POST envía los datos a una fuente específica mencionada en el atributo “action”. El método GET solicita datos a una fuente específica mencionada en el atributo “action”.

No podemos olvidar los elementos que acompañan a la etiqueta form , elementos tales como: <input>, <textarea>, <select>, <button>, cada uno de los cuales tienen sus propios atributos requeridos, como lo son los siguientes:

- “name”: identifica el elemento de todos los elementos en el objeto DOM.
- “value”: es requerida por el elemento input y select, aunque está permitido estar vacío, ya que cuando un usuario escribe un texto, este se convierte enteramente en el valor de “value”.
- “type”: es usado únicamente por el elemento <input> para determinar el propósito de la entrada es decir el tipo de datos que recibirá la etiqueta, dentro del HTML5 más elementos fueron introducidos para el atributo “type”, como “email”, “tel”, y “datepicker”.

Dentro de input podemos encontrar el atributo Hidden que es usado para agregar valores al formulario que el usuario no debe de ver ni usar. Es posible agregar múltiples campos hidden pero cada campo debe de tener un único nombre.

El input type text o number es un tipo de entrada de una sola línea que puede contener un valor de tipo texto por ejemplo es usado por los usuarios para llenar sus nombres y direcciones. Puede contener cualquier tipo de texto que el usuario introduzca pero cuando el formulario es enviado es turno del back-end para validar los datos del lado de la aplicación según sea necesario, si el valor no cumple con ciertas reglas el formulario con los datos es enviado de vuelta y la página muestra errores que provienen del campo que ha sido llenado de forma incorrecta.

- El password type puede contener cualquier texto que el usuario introduzca, casi como el elemento type pero el texto es escrito con asteriscos esto para proteger las contraseñas del usuario.

- El radio type se usa para que el usuario pueda escoger entre diversas respuestas, este representa una elección entre diferentes valores, se enviará únicamente el valor seleccionado por el usuario.

- El checkbox type es usado para permitir al usuario elegir representa una elección entre ninguno o más elementos es decir el usuario puede elegir diversos campos y los valores son enviados en forma de lista.

- un tipo especial es el file type. este tipo permite que puedas elegir un archivo de tu computadora y lo puedas cargar hacia el servidor requiere que el elemento del formulario tenga el argumento método en POST y adicionalmente el atributo “enctype” con el valor “multipart/form-data” en el back-end de la aplicación web, podemos manejar estas estas partes del formulario que recibimos cuando se elige un archivo.

- El último tipo de input es button type, el atributo “value” en este tipo de input no es usado para guardar información hacia el servidor pero contiene el atributo valor que se muestra en el botón.

Para los objetivos de este trabajo nos concentramos en el uso de formularios estándar HTML con los tipos de campos básicos: Texto, Número, Opciones y Checkbox (casilla de selección), en la siguiente imagen se puede ver un ejemplo de cómo luciría la interfaz de nuestra aplicación web estándar.

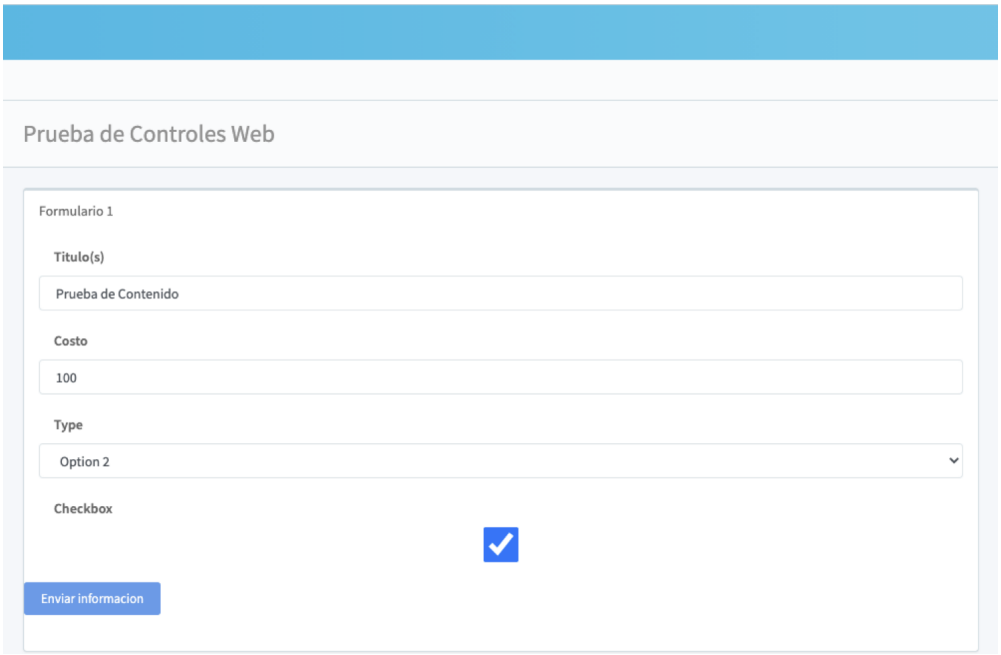


Figura 6 Ejemplo de formulario de la aplicación web estandar definida

Y nuestra aplicación web estándar podrá tener como se comentó una o varias páginas que componen el ciclo de vida de la aplicación, cada una de estas páginas es un recurso el cual puede contar con sus propios controles (formularios) y para acceder a cada uno de estos recursos se debe seguir un proceso definido.

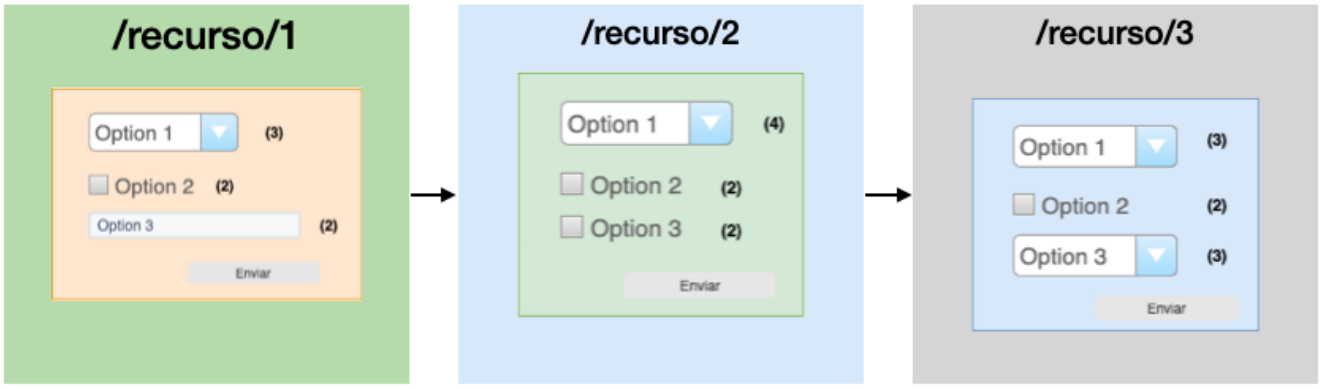


Figura 7 Ejemplo de ciclo de vida de una aplicación web estándar

Cada uno de los controles (formularios) con los que cuenta cada una de las páginas (recursos) pueden tener diferentes opciones y podrán ser operados dentro de cualquiera de sus combinaciones posibles,

cada una de estas combinaciones posibles con las que se envía un formulario como una petición hacia el servidor se le puede denominar como un estado ya que cada una de estas combinaciones es recibida e interpretada por el servidor de forma diferente y puede resultar en un comportamiento dependiente del estado previo recibido.

Por lo cual se debe considerar durante el proceso de pruebas automatizada que cada una de estas combinaciones que se envíe dentro de un formulario puede afectar las peticiones consecutivas, pudiendo llegara resultar en un gran número de combinaciones posibles considerando el ciclo de vida de la aplicación, por lo cual es esencial considerar que el sistema de pruebas automatizadas pueda ser capaz de reproducir todos los estados posibles y sus combinaciones con el resto de recursos, ya que un error puede encontrarse en cualquier lugar (combinación de estados específica) y el sistema debe ser capaz de detectarlo y mapearlo para su reporte.

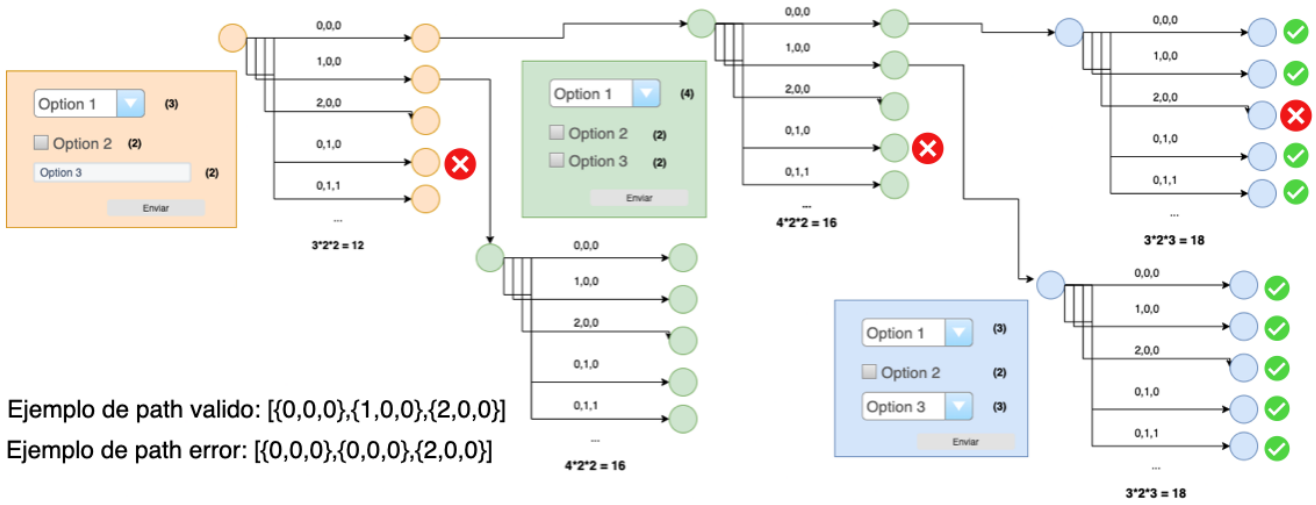


Figura 8 Representación de una aplicación web, sus estados posibles validos y erróneos

Por ejemplo si tomamos de ejemplo el diagrama anterior (Figura 8) donde se representa una aplicación web sencilla, compuesta de 3 páginas (recursos) y en donde cada una tiene un único formulario, cada uno de estos formularios tiene 3 campos con diferentes opciones posibles, para fines de pruebas se deben tomar en cuenta cada una de las combinaciones, tomando por ejemplo el formulario del recurso inicial de esta aplicación se puede ver que cuenta con 3 campos, 1 campo selector con 3 opciones, 1 campo checkbox con 2 opciones y un campo de texto que para fines prácticos se considera con 2 estados (vacío y lleno), al contabilizar las posibles combinaciones que puede representar este formulario en una petición, se tiene que serían un total de 12 (3*2*2), cada una de ellas representando un estado posible del formulario, ahora siguiendo el mapa de la aplicación se puede observar que al transitar al siguiente recurso el cual tiene diferentes campos y por lo tanto posibles estados, las combinaciones posibles se multiplican por el número de estados y niveles previos, en este ejemplo el número total de combinaciones teóricas de esta aplicación se puede obtener fácilmente multiplicando las combinaciones posibles de cada uno de los recursos: 12*16*18 = 3,456

Sin embargo esto solo sería cierto si todas y cada una de estas combinaciones previas al recurso final tuvieran como destino un recurso siguiente, sin embargo en la práctica esto puede no ser cierto ya que se dependen de múltiples factores como la lógica del negocio de la aplicación e incluso posibles errores

dentro del ciclo de vida de la misma, por lo cual en la práctica este número de combinaciones de estados puede llegar a ser mucho menor, sin embargo no se tiene la certeza de ello por lo cual en el diseño del sistema se deberá considerar el peor caso en donde si se cumple esta premisa.

Por lo cual el sistema desarrollar además deberá ser capaz de poder replicar y verificar cada una de estas posibles combinaciones y estados de la aplicación, evaluar la respuesta HTTP que produce y en caso de ser un estado válido continuar adelante o si se trata de un error ser capaz de reportar que combinación de estados específica llevaron a dicho error para que posteriormente pueda ser replicado por el equipo de desarrollo y poder tomar las acciones necesarias para solventar el problema.

Siguiendo con el ejemplo presentado en la Figura 8 se puede observar que una combinación de estados válida la cual nos lleva al final del ciclo de la aplicación sin presentar un error es el siguiente [{0,0,0},{1,0,0},{2,0,0}] es decir que el usuario seleccionó las opciones 0 en los 3 campos del primer recurso, posteriormente en el segundo recurso seleccionó la opción 1 del primer campo y la opción 0 de los campos siguientes, mientras que en el tercer recurso seleccionó la opción 2 del primer campo y en los campos siguientes la opción 0.

Mientras que una combinación de estados que desemboca en un error sería la siguiente: [{0,0,0},{0,0,0},{2,0,0}], de esta cadena de caracteres se puede inducir fácilmente cuáles fueron las acciones tomadas por el usuario que lo llevaron a una respuesta errónea, por lo cual será fácilmente reproducible posteriormente.

Por lo cual es fundamental que nuestro sistema sea capaz de reportar de forma similar los caminos y estados que desembocan en un error.

Otra consideración importante que se tiene dentro de los objetivos de este trabajo es que el sistema de pruebas automatizadas funcione con la mínima intervención humana posible con el objetivo de evitar introducir posibles errores humanos en la configuración del sistema de pruebas que puedan resultar en pruebas deficientes, es por ello que se propone el uso de machine learning en específico del tipo aprendizaje por refuerzo dentro de la implementación del sistema como un auxiliar para reducir la intervención humana para el correcto funcionamiento del sistema.

4.3 Diseño e implementación del sistema

En esta sección se explica de forma general el proceso de diseño del sistema que cumpliera con los objetivos y metas propuestos en este trabajo, así como la justificación a las decisiones tomadas para resolver los diversos problemas que surgieron en el proceso de desarrollo.

4.3.1 Navegación automatizada de la aplicación

Tomando como base el tipo de aplicación web estándar que se definió como base para el desarrollo de este trabajo, podemos traducir que el ciclo de vida de este tipo de aplicaciones se puede ver como un árbol en donde la raíz del árbol es la sección de inicio de la aplicación dentro de la cual puede haber uno o más formularios cada uno con sus propias opciones y por lo tanto posibles combinaciones, en el siguiente diagrama se puede apreciar de forma gráfica:

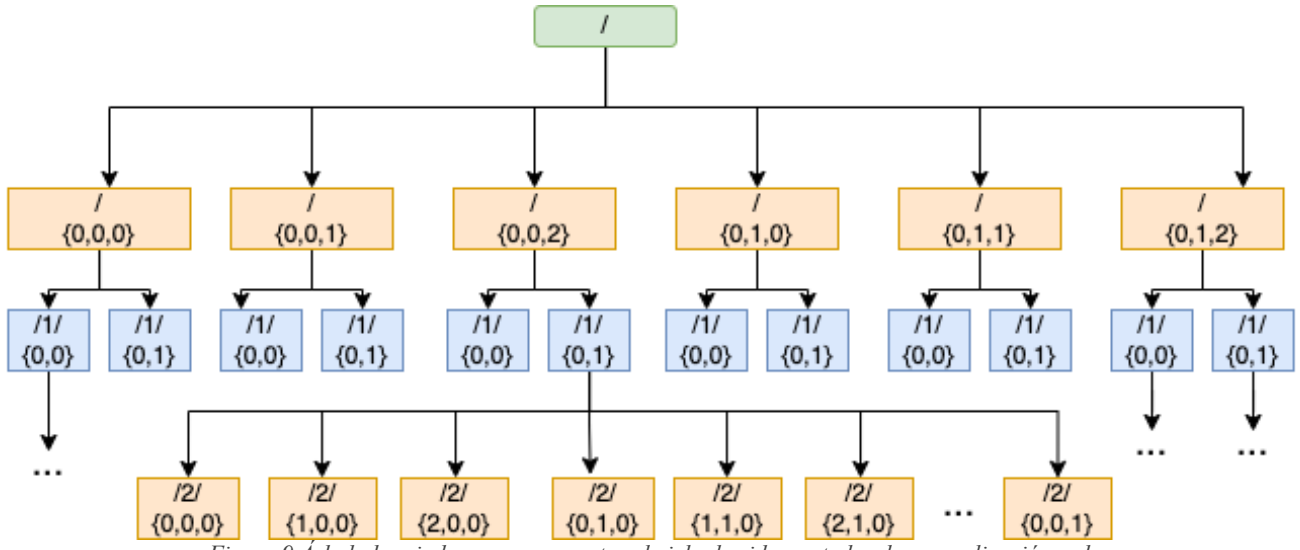


Figura 9 Árbol abreviado para representar el ciclo de vida y estados de una aplicación web

La raíz del árbol está representada por "/" la página inicial la cual contiene un formulario con 3 campos, uno con solo una opción, otro más con 2 opciones y el último campo con 3 opciones , por lo tanto se tienen 6 posibles combinaciones (1*2*3) o estados posibles, posteriormente el siguiente nivel del árbol está representado por una siguiente página o recurso "/1/" que también contiene un formulario en este ejemplo con solo 2 campos, las posibles combinaciones se replican en cada uno de los estados previos, finalmente en el tercer nivel se ilustra solo un ejemplo de este nivel en donde el formulario del recurso "/2/" cuenta con 3 campos pero con un mayor número de opciones, por lo cual estos estados se replicarán en cada uno de los estados del nivel previo y de esta forma se puede confirmar que es posible representar el mapa del ciclo de vida de nuestra aplicación web estándar como una estructura tipo Árbol.

Por lo cual el reto se encuentra en poder recorrer el árbol completo de la aplicación web estándar de forma autónoma de tal forma que se puedan verificar cada uno de los estados y transiciones en búsqueda de errores y si se encuentran poderlos reportar.

Ahora el primer reto a resolver es lograr controlar de forma autónoma los controles de una página dada, es decir a través del formulario web presente, para ello es importante recordar que en nuestro tipo de aplicación web estándar cada recurso (página) es un documento HTML que es construido desde el servidor y por lo cual su código fuente puede ser accedido directamente sin necesidad de un navegador intermediario que lo interprete.

Figura 10 Representación HTML de un formulario web

En el diagrama anterior (Figura 10) se puede observar la representación en código HTML del formulario de la izquierda, en este caso el formulario cuenta con 3 campos: Selector con 3 opciones, Checkbox con 2 opciones (verdadero y falso) y un campo de texto con 2 opciones (vacío y lleno), su representación HTML inicia con la etiqueta <form> y termina en </form> entre estas etiquetas se pueden encontrar las etiquetas HTML que representan a los campos y controles del formulario por ejemplo:

La etiqueta <select> en conjunto con <option> representa el campo Selector y sus 3 opciones.
La etiqueta <input type=”checkbox”> representa el campo tipo Checkbox
La etiqueta <input type=”text”> representa el campo de texto
Y finalmente la etiqueta <input type=”submit”> representa el botón para enviar el formulario al servidor vía el método POST de HTTP.

Al tener acceso al código HTML de forma directa es posible usar una librería que nos permita interactuar con el documento HTML y realizar peticiones de forma programática.

Mechanize
Es una librería open source basada en Python la cual permite interactuar con páginas web, interactuando directamente con su código fuente y el protocolo HTTP, pese a no tener soporte para interpretar Javascript, la gran ventaja de Mechanize frente a otras alternativas como lo puede ser un navegador web headless es su velocidad la cual es muy superior a la de otras alternativas al no tener que interactuar con un navegador web, es por ello que se eligió ya que la velocidad será crucial en las tareas que se desean realizar con esta herramienta ya que como se vio en párrafos anteriores el número de combinaciones y estados a comprobar en una aplicación web por muy pequeña que sea pueden crecer de forma exponencial, por lo cual se requiere de herramientas rápidas para hacer viable la exploración.

A continuación se ilustra un ejemplo del uso de Mechanize:

```
from mechanize import Browser

//recurso al que se desea acceder
url = "http://localhost:8080/page_1"

browser = Browser()
browser.open(url)
//seleccion de formulario
browser.select_form(nr=1)

//asignacion de valores en formulario
browser['title'] = ''
browser.set(True, tipos[0] , "type")
browser.find_control("checkbox").items[0].selected = False

//envio de peticion POST
try:
    response = browser.submit()
    //obtencion de codigo HTTP e impresion
    code = response.code
    print(code)
except mechanize.HTTPError as e:
    //si existe un error HTTP mostrarlo
    print('Error:', e)
```

Figura 11 Ejemplo del uso de Mechanize para la interacción directa con una aplicación web

Como se puede ver la interacción con una página web a partir del código HTML de la misma, además del envío de peticiones POST y captura de errores HTTP es sencillo.

En este ejemplo se está controlando el formulario de una página web conocido ya que se conoce el formulario a operar, los campos con los que cuenta y los valores posibles y por lo tanto es sencillo poder enviar una petición HTTP exitosa, sin embargo esto nos limita a requerir de intervención humana para poder seleccionar el formulario, conocer sus campos y llenarlos con valores válidos, por lo cual es necesario encontrar una forma de interactuar con el formulario y realizar peticiones válidas que nos permitan avanzar en el ciclo de vida de la aplicación.

Debido a que necesitamos poder controlar e interactuar de forma autónoma con cualquier formulario que se presente, siempre y cuando esté compuesto por alguno o todos los posibles campos que serán soportados en nuestro sistema, los cuales son los siguientes:

Campo Select con sus opciones: <select> <option> </option> </select>
Campo Checkbox: <input type=”checkbox”>
Campo de Texto: <input type=”text>
Campo de Numero: <input type=”number”>
Botón Submit: <input type=”submit”>

Además queremos ser capaces de interactuar y controlar formularios de forma autónoma sin importar el número de campos que tenga, el orden de los mismos o las opciones disponibles para cada campo, sin embargo esto sería complicado de lograr a través de un algoritmo tradicional, por lo cual aquí es donde toma sentido el uso de nuevos enfoques como es el aprendizaje por refuerzo que como se explicará en párrafos siguientes es una herramienta versátil que nos auxilia para resolver este problema.

4.3.2 Aprendizaje por Refuerzo

La idea de aprendizaje por refuerzo es maximizar recompensas escogiendo acciones correctas en diferentes situaciones [15]. Los principales componentes de RL son los agentes y él ambiente. Un agente observa, actúa y aprende en un ambiente. El agente interactúa con el ambiente escogiendo acciones. Las acciones pueden cambiar el estado del ambiente y dar al agente una recompensa positiva o negativa.

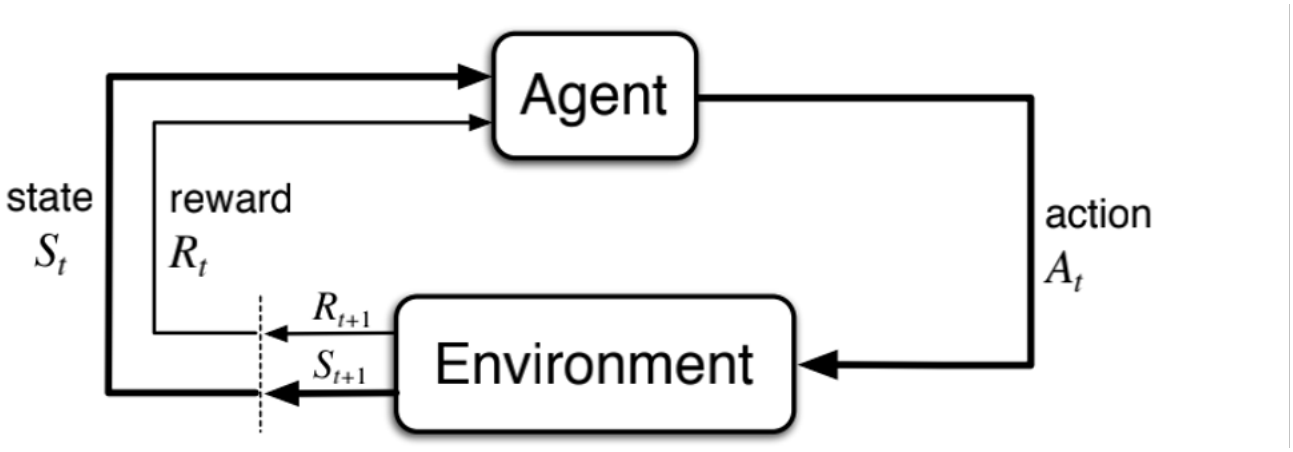


Figura 12 Representación del funcionamiento de un algoritmo de aprendizaje por refuerzo

Los componentes principales de un algoritmo de aprendizaje por refuerzo son los estados, acciones, dinámicas, y funciones de recompensa, los estados definen la situación actual, las acciones son las formas en las que el agente interactúa con el ambiente, las dinámica nos dicen cómo el ambiente cambia basado en el estado actual y la acción del agente y la función de recompensa es usada para incentivar o desincentivar al agente.

Por lo cual tomando como base el aprendizaje por refuerzo se aborda cómo resolver el problema del control autónomo de los formularios de nuestra aplicación web estándar.

Como se planteó en párrafos anteriores un formulario dentro de una página, con su conjunto de campos y acciones puede ser visto como un conjunto de estados posibles como se puede ver en el diagrama siguiente:

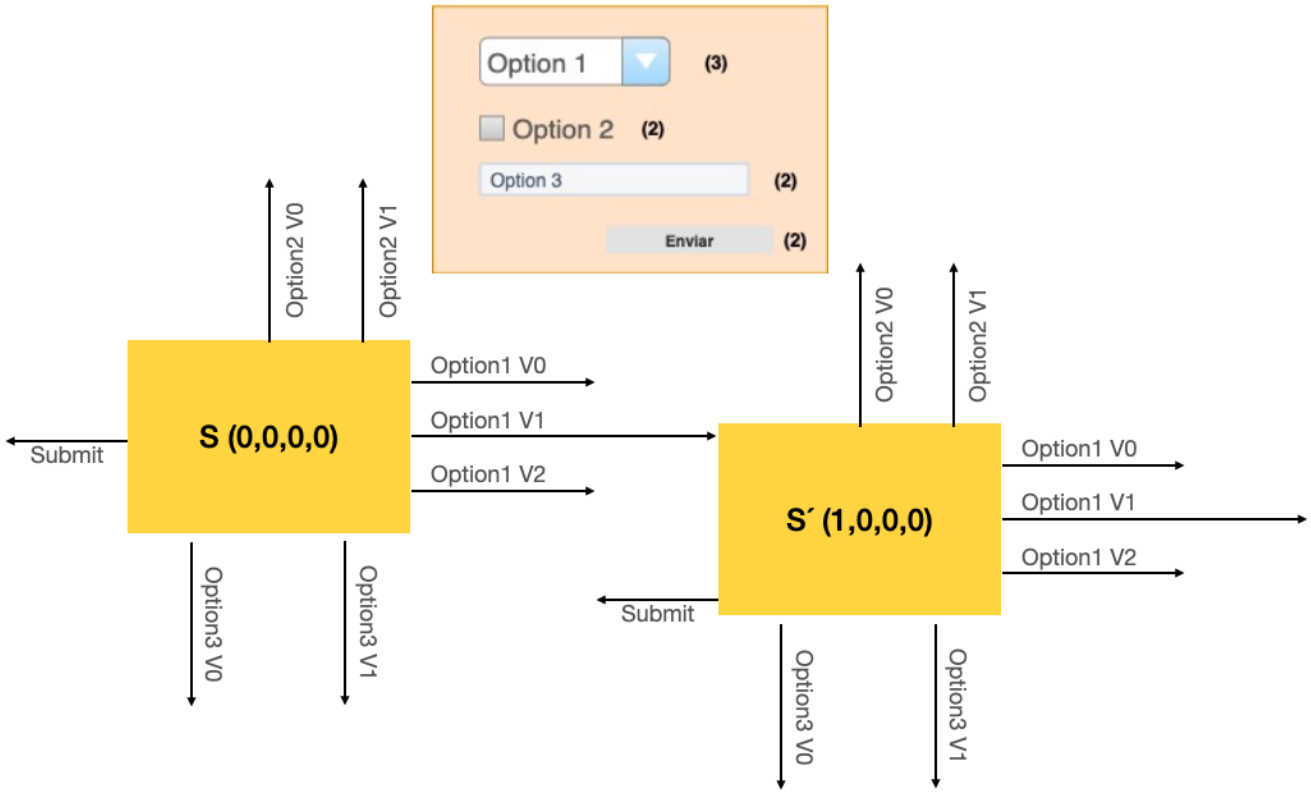


Figura 13 Representación de un formulario y su representación de estados y acciones

En este caso se tiene un formulario con 3 campos y botón Submit, Select (3 opciones), Checkbox (2 opciones), Texto (2 opciones) y botón Submit (2 opciones), cada una de estas combinaciones puede ser representado como un estado del formulario, por ejemplo el estado S (0,0,0,0) representa al formulario con todos sus campos vacíos y sin acción de envió (Submit).

Las acciones posibles del estado son las transiciones a otro estado, es decir el cambio en sus campos o botón de envió, por lo cual cada una de las posibles opciones que puede tomar un campo del formulario se considera como una acción, en el ejemplo se puede ver claramente que si el formulario se encuentra en el estado S (0,0,0,0) y se toma la acción "Option1 V1" es decir del campo Option se selecciona la opción V1 entonces transita al estado S'(1,0,0,0) o de manera más formal:

S (0,0,0,0) ---- Option1 V1 ---> S'(1,0,0,0)

Y este es el principio básico del aprendizaje por refuerzo, un entorno de acción que en este caso es el formulario, estados y acciones que representan el comportamiento del sistema, el siguiente paso es implementar un agente que pueda interactuar con el entorno de forma autónoma.

Para ello el siguiente punto a considerar es la matriz de entorno que consiste en una matriz donde se mapean los estados posibles del entorno y las posibles acciones que se pueden tomar, a continuación se muestra la matriz de entorno para el ejemplo actual:

					Acciones							
Estados	F1	F2	F3	S	F1OP0	F1OP1	F2OP0	F2OP1	F3OP0	F3OP1	F3OP2	SUBMIT
estado1	0	0	0	0	0	0	0	0	0	0	0	0
estado2	0	0	1	0	0	0	0	0	0	0	0	0
estado3	0	0	2	0	0	0	0	0	0	0	0	0
estado4	0	1	0	0	0	0	0	0	0	0	0	0
estado5	0	1	1	0	0	0	0	0	0	0	0	0
estado6	0	1	2	0	0	0	0	0	0	0	0	0
estado7	1	0	0	0	0	0	0	0	0	0	0	0
estado8	1	0	1	0	0	0	0	0	0	0	0	0
estado9	1	0	2	0	0	0	0	0	0	0	0	0
estado10	1	1	0	0	0	0	0	0	0	0	0	0
estado11	1	1	1	0	0	0	0	0	0	0	0	0
estado12	1	1	2	0	0	0	0	0	0	0	0	0
estado13	0	0	0	1	0	0	0	0	0	0	0	0
estado14	0	0	1	1	0	0	0	0	0	0	0	0
estado15	0	0	2	1	0	0	0	0	0	0	0	0
estado16	0	1	0	1	0	0	0	0	0	0	0	0
estado17	0	1	1	1	0	0	0	0	0	0	0	0
estado18	0	1	2	1	0	0	0	0	0	0	0	0
estado19	1	0	0	1	0	0	0	0	0	0	0	0
estado20	1	0	1	1	0	0	0	0	0	0	0	0
estado21	1	0	2	1	0	0	0	0	0	0	0	0
estado22	1	1	0	1	0	0	0	0	0	0	0	0
estado23	1	1	1	1	0	0	0	0	0	0	0	0
estado24	1	1	2	1	0	0	0	0	0	0	0	0

Figura 14 Ejemplo de la matriz de entorno

En este caso se tiene que el entorno tiene 24 posibles estados debido a las combinaciones posibles de sus campos y se cuentan con 8 posibles acciones que son las transiciones que llevarán de un estado a otro.

Q learning

El aprendizaje por refuerzo consiste en elegir la política óptima o la función de valor que maximice las recompensas para resolver un problema dado, ¿Cómo determinar dicha política?

Hay varios algoritmos para lograr esto, uno de ellos es Q Learning, el más común en el aprendizaje por refuerzo, la implementación más simple para Q Learning es una tabla que contiene acciones en un eje y estados en el otro, como la vista en la Figura 14.

Cada celda de la tabla determina la conveniencia de un movimiento del agente en un estado dado, inicialmente esta tabla como se pudo ver se encuentra en blanco y es tarea del Agente encontrar la política óptima que nos permita cumplir el objetivo propuesto de forma eficiente, para lograrlo en Q learning el agente hace uso del algoritmo Q el cual se describe a continuación:

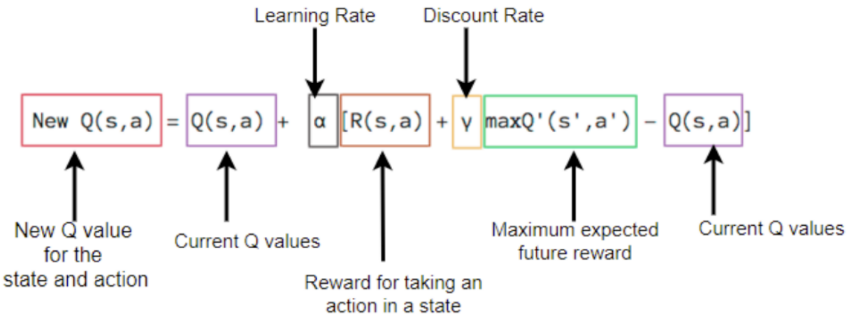


Figura 15 Descripción del algoritmo Q Learning

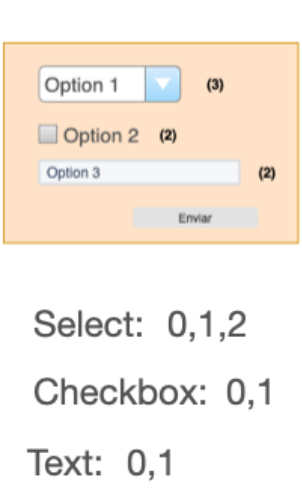
En los Anexos 1 se puede consultar la implementación del agente Q Learning.

El proceso de entrenamiento ocurre como sigue, se define un valor discount (0.9) el cual es un factor para restar valor a las recompensas futuras dentro de la ecuación Q, así como el learning rate el cual es un factor para determinar qué tanto va a aprender nuestro algoritmo de la acción tomada en el estado actual, en nuestro caso se estableció en 0.1 ya que requerimos que el algoritmo tenga un enfoque de exploración más que de explotación ya que para nosotros es una herramienta para descubrir los estados que nos llevan a una petición HTTP exitosa (200) o bien errónea (404, 500).

A continuación tenemos la función initialize() la cual se encarga de inicializar el entorno instanciando un objeto tipo browser de la librería mechanize la cual como se explicó en párrafos anteriores será la encargada de interactuar con los controles del formulario y realizar las peticiones HTTP correspondientes.

El siguiente paso es seleccionar un estado inicial de forma aleatoria para iniciar el entrenamiento, sin embargo obtener la matriz de estados y acciones de forma programática para cualquier formulario es otro reto a resolver, lo cual se llevó a cabo de la siguiente forma:

Consideremos un formulario con los siguientes campos y opciones, para obtener el número de combinaciones posibles simplemente debemos multiplicar las opciones posibles de cada campo por la de los demás en este ejemplo sería: 3*2*2 teniendo como resultado 12 posibles combinaciones.



	Select	Check	Text
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1
9	2	0	0
10	2	0	1
11	2	1	0
12	2	1	1

Figura 16 Ejemplo de representación de un formulario y sus estados posibles

Sin embargo no es suficiente con obtener el número de combinaciones posibles, para poder construir la matriz de entorno es necesario conocer cuales son todas esas combinaciones, que en este ejemplo serían las 12 combinaciones mostradas en la tabla del diagrama anterior.

El primer paso para poder obtener estas combinaciones de forma automatizada es explorar la página de interés con la ayuda de mechanize para encontrar los formularios que se deben explorar así como el número de campos, tipo de campo y opciones del campo en caso de ser un campo de opción múltiple, en los Anexos 2 se observar a detalle la programación del módulo que se encarga de esta tarea.

Este módulo nos devolverá una lista con los diferentes campos y sus parámetros como la siguiente:

results = [‘option1’:[‘select’, [‘0’, ‘1’, ‘’2], ‘option2’:[‘text’, ‘’], ‘option3’:[‘checkbox’, ‘’]]

Posteriormente para poder construir estas combinaciones de campos y opciones que serán los estados del entorno, se hizo uso del concepto de la estructura de árbol, ya que como se puede ver en el siguiente diagrama es posible representar las combinaciones posibles dentro de un árbol.

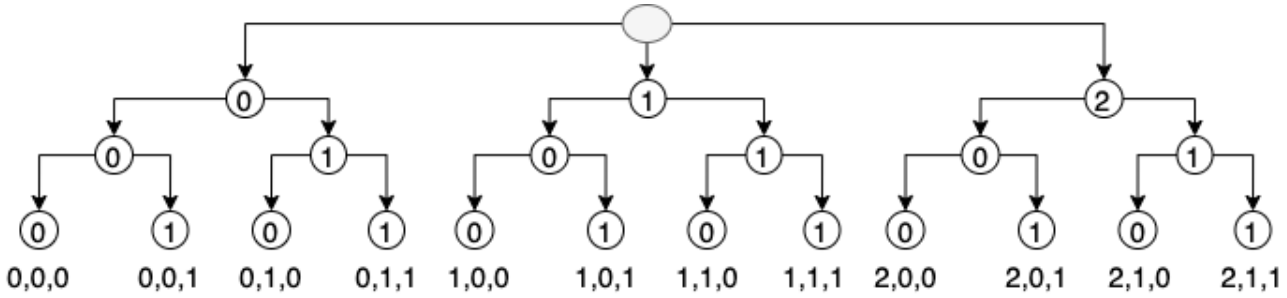


Figura 17 Representación de las combinaciones posibles de estados basadas en un arbol.

Por lo cual es necesario construir un árbol en donde cada uno de sus niveles represente las opciones de valores posibles que se tienen cada uno de los campos del formulario, en este ejemplo se puede observar en el primer nivel los 3 valores posibles del campo Select (0,1,2), a continuación como hojas de cada uno

de estos valores se asignan los valores posibles del siguiente campo Checkbox (0,1) y finalmente como hojas del nivel anterior se asignan los valores del campo Texto (0,1) y al final se puede reconstruir todas y cada una de las combinaciones posibles que este formulario puede tomar y que representan sus estados y acciones.

Para resolver la generación de este árbol y la recuperación de los estados se usó una estructura de datos conocida como listas ligadas, la cual es una estructura de datos de gran utilidad para poder representar árboles, este código se puede observar en los Anexos 3.

El código mencionado toma como entrada una matriz que representa las opciones posibles que pueden tomar los campos de un formulario por ejemplo [[0,1,2],[0,1],[0,1]] y retorna un diccionario con todas las combinaciones posibles que se pueden generar con estas opciones que representan los estados del formulario:

{'0,0,0': [0, 0, 0], '1,0,0': [1, 0, 0], '0,1,0': [0, 1, 0], '1,1,0': [1, 1, 0], '0,0,1': [0, 0, 1], '1,0,1': [1, 0, 1], '0,1,1': [0, 1, 1], '1,1,1': [1, 1, 1], '0,0,2': [0, 0, 2], '1,0,2': [1, 0, 2], '0,1,2': [0, 1, 2], '1,1,2': [1, 1, 2]}

Una vez obtenidas los estados posibles del formulario, es posible obtener la matriz de entorno la cual está basada en estos estados y las posibles acciones a tomar que en nuestro ejemplo son 7 acciones: [[0,1,2],[0,1],[0,1]]

```
environment_matrix =
{'000':[0,0,0,0,0,0,0],'001':[0,0,0,0,0,0,0],'002':[0,0,0,0,0,0,0],
 '010':[0,0,0,0,0,0,0],'011':[0,0,0,0,0,0,0],'012':[0,0,0,0,0,0,0],
 '100':[0,0,0,0,0,0,0],'101':[0,0,0,0,0,0,0],'102':[0,0,0,0,0,0,0],
 '110':[0,0,0,0,0,0,0],'111':[0,0,0,0,0,0,0],'112':[0,0,0,0,0,0,0]}

```

Figura 18 Ejemplo de Matriz de entorno en representación JSON

Si nos remitimos al código de Anexos 1 de la función de entrenamiento “training()”, podemos ver que el siguiente paso es realizar un proceso iterativo que se encargará de formar los diferentes estados posibles en base a tomar una u otra acción disponible dentro del entorno, esto se realizará de forma indefinida hasta encontrar un combinación que nos retorne un código de HTTP 200, 404 o 500, ya que esto significa que la combinación usada logró el envió del formulario y por lo tanto se agrega a un listado de los estados (combinaciones) que permitan un envió del formulario al servidor, para que estas combinaciones sean usadas posteriormente en la fase de explotación.

Un ejemplo de ello es el siguiente, tomemos como base el formulario en el que se viene trabajando, este cuenta con 3 campos y en total 7 posibles opciones:

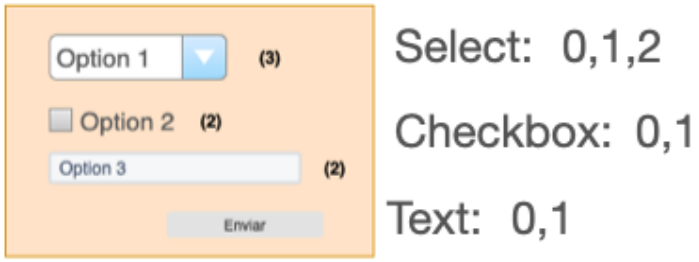


Figura 19 Ejemplo de formulario con sus opciones validas

Dependiendo la lógica con la que fue construida la aplicación web es posible que en este formulario los 3 campos sean obligatorios para poder enviar el formulario de forma exitosa o bien si no todos son obligatorios es posible que incluso se pueda enviar el formulario solo llenando 1 o 2 de ellos, y es precisamente estas combinaciones válidas las que son encontradas por nuestro algoritmo de aprendizaje por refuerzo, si el formulario requiere que los 3 campos estén llenos la salida sería la siguiente:

[‘option1’:[‘true’, [0,1,2]], ‘option2’:[‘true’, [0,1]], ‘option3’:[‘true’, [0,1]]]

Y un formulario que no requiere los 3 campos llenos se vería así:

[‘option2’:[‘true’, [0,1]], ‘option3’:[‘true’, [0,1]]]

Una vez obtenidos los campos válidos para poder enviar el formulario en una petición HTTP y recibir una respuesta, se inicia el proceso de explotación con el objetivo de explorar todo el árbol de la aplicación web en búsqueda de respuestas exitosas o bien errores, para ello el enfoque es el siguiente, como se mostró en la figura 7 el ciclo de vida de una aplicación web se puede representar como un árbol en donde cada una de sus páginas puede tener uno o más formularios y cada uno de ellos representa un set de estados a ser probados, por lo cual el método de exploración de este árbol conviene realizarla como una búsqueda a lo profundo, por lo cual el proceso usado es el siguiente:

1. Se accede a la página raíz de la aplicación y el algoritmo de aprendizaje por refuerzo se encarga de obtener los formularios y combinaciones válidas para enviar una respuesta HTTP exitosa, se guardan estos parámetros para su posterior uso.

2. Una vez obtenido el formulario y opciones válidas en el paso anterior se inicia la fase de explotación en donde se comienza a probar todas las combinaciones posibles con los campos válidos para enviar el formulario como una petición HTTP, la respuesta del servidor puede ser un código 200 de respuesta exitosa en conjunto con la URL de un nuevo recurso a explorar, un error 404 debido a que se llegó a un recurso no existente o bien un error 505 debido aun error del servidor, son precisamente estos errores los que se desean detectar de forma automatizada.

3. En el caso de que el envío del formulario haya sido exitoso entonces el servidor responderá con un código 200 y la URL de un nuevo recurso (página), la cual se consultará si ya ha sido previamente visitada o no, en caso de que sea nueva entonces se realiza el proceso de reconocimiento del formulario descrito en el paso 1 para obtener sus campos y opciones válidas.

4. Se repite el paso 2, 3 y 1 de forma consecutiva hasta llegar a un recurso que ya no devuelva una nueva página o bien que devuelva un error 404 o bien 500, en caso de recibir uno de estos códigos se retorna el camino de estados que llevó a dicho error, esto es posible gracias a que el árbol se construye bajo una estructura de listas ligadas, por lo cual es posible referenciar siempre hacia el estado anterior.

5. Una vez que se termina llega a un estado final del árbol (hoja), la exploración debe continuar retornando al nivel superior y continuar visitando los siguientes estados siguiendo la exploración a lo profundo de forma recursiva.

6. Una vez que se ha terminado de explorar el árbol de la aplicación completa entonces se retorna el listado de errores encontrados en conjunto con los estados que llevaron a dicho error.

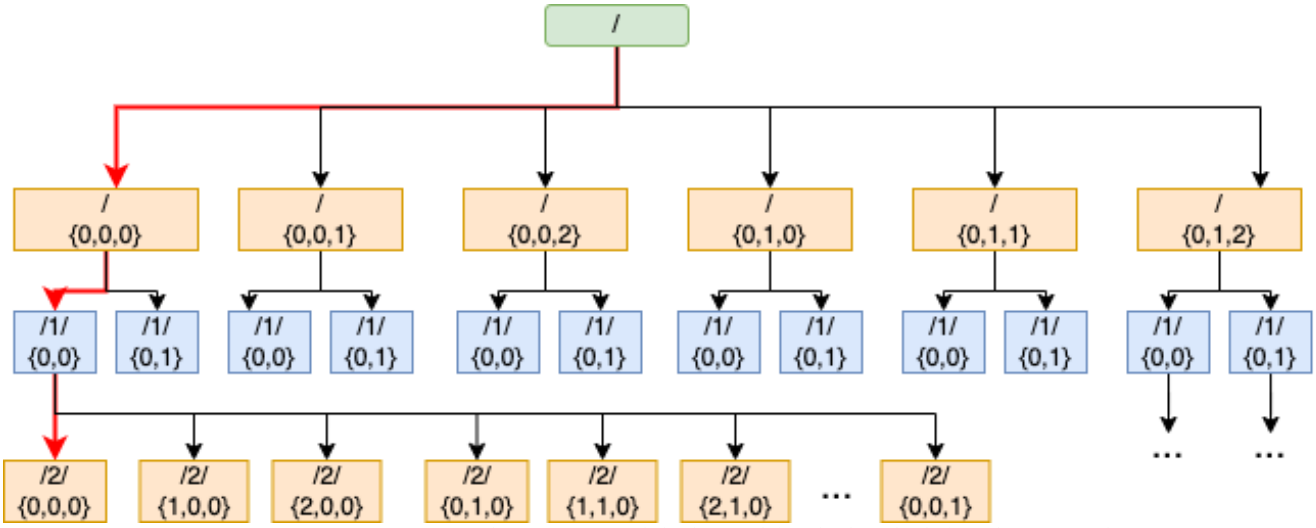


Figura 20 Representación de un camino valido dentro del ciclo de vida de la aplicación web

El algoritmo programa final de pruebas automatizadas se explica a continuación:

El entorno se representa como una clase llamada “Enviroment” la cual almacena datos requeridos durante el proceso de explotación, como lo es la URL inicial de la aplicación, los caminos creados que están basados en listas ligadas y representan el árbol, así como el stack de errores que como su nombre lo indica es una estructura de datos tipo Stack donde se almacenan los paths que desembocan en un error, para su reporte posterior.

El entorno se inicializa llamando a la función “get_fields_forms()” de la clase “CrawForms” descrita en Anexos 2 para obtener los campos y opciones posibles del formulario actual y posteriormente pasar la lista de resultado a la clase “Generator” descrita en Anexos 3 y a continuación se llama a la función “training” del módulo de aprendizaje por refuerzo que será la encargada de retornar las combinaciones de campos y opciones validos para enviar una petición POST aceptada por el servidor.

Con estas combinaciones de campos y opciones válidas, se inicia la etapa de explotación y donde de forma recursiva se realiza la exploración del los recursos de la aplicación web, construyendo el árbol durante el proceso, para ello nos apoyamos de una lista ligada que se usa para representar las ramas del

árbol junto con sus nodos que en este caso estará formado por Estados, en donde cada estado representará una combinación de campos y opciones válidas.

Un elemento importante de este algoritmo es la función recreate_state(state, entorno) la cual tiene la función durante la exploración a lo profundo del árbol de recrear de forma directa un estado determinado para continuar con la exploración una vez que se retorna a un estado de nivel superior, esta función recibe el estado que se desea recrear "state" y el "entorno" en que se guardan los parámetros de la exploración.

Cuando se detecta un error durante la exploración ya sea del tipo 404 o 500 entonces se guarda la rama que llevó a este error a través de la función build_error_chain(current_state, global_error_stack) la cual recibe el estado actual y será a partir del cual se inicie la recreación de la cadena de estados que llevaron al error a través de recorrer la rama de forma inversa y este camino se almacenará en el stack "global_error_stack" para finalmente reportarlos al final de la exploración en formato JSON como se puede ver a continuación.


```
1 [[{'id': 6, 'error': '400', 'url': 'http://localhost:8080/page_1', 'text': 'aaaa', 'number':
  '9', 'check': 'off', 'type': '0'}],
2  [{'id': 1, 'error': '200', 'url': 'http://localhost:8080/page_1', 'text': 'aaaa', 'number':
  '9', 'check': 'on', 'type': '0'}, {'id': 5, 'error': '500', 'url':
  'http://localhost:8080/page_1_1', 'text': 'aaaa', 'number': '9', 'check': 'off', 'type': '1'}],
  [{'id': 1, 'error': '200', 'url': 'http://localhost:8080/page_1', 'text': 'aaaa', 'number':
  '9', 'check': 'on', 'type': '0'}, {'id': 4, 'error': '500', 'url':
  'http://localhost:8080/page_1_1', 'text': 'aaaa', 'number': '9', 'check': 'on', 'type': '1'}],
  [{'id': 1, 'error': '200', 'url': 'http://localhost:8080/page_1', 'text': 'aaaa', 'number':
  '9', 'check': 'on', 'type': '0'}, {'id': 3, 'error': '400', 'url':
  'http://localhost:8080/page_1_1', 'text': 'aaaa', 'number': '9', 'check': 'off', 'type': '0'}],
  [{'id': 1, 'error': '200', 'url': 'http://localhost:8080/page_1', 'text': 'aaaa', 'number':
  '9', 'check': 'on', 'type': '0'}, {'id': 2, 'error': '400', 'url':
  'http://localhost:8080/page_1_1', 'text': 'aaaa', 'number': '9', 'check': 'on', 'type': '0'}]]
```


Figura 21 Representación JSON del resultado de caminos que resultan en un error dentro de la aplicación web

Empaquetamiento en un framework

Como parte de los objetivos de este trabajo se encuentra la implementación del algoritmo de pruebas automatizadas dentro de un framework que permita su uso de forma sencilla y amigable para el usuario, para ello se desarrolló una aplicación web basada en el framework Django el cual está basado en Python lo que lo vuelve ideal para uso ya que es posible integrar de forma directa el algoritmo que se desarrolló también en Python.

Se diseñó una interfaz de usuario sencilla basada en HTML, CSS y Javascript, la cual requiere solo de unos pocos parámetros para operar:

La URL inicial de la aplicación que se requiera probar de forma automatizada.

Y algunas opciones de campos por defecto, el campo número se puede seleccionar si se desea probar con un valor cero, si no se selecciona esta opción se probará con un número aleatorio.

La opción de cadena de prueba es si se desea introducir un texto de prueba por default.

Para iniciar el proceso de pruebas el usuario solo debe presionar el botón “Iniciar” y esperar a que concluya el proceso que dependiendo del tamaño y complejidad de la aplicación web a probar puede ser un tiempo corto o muy largo.

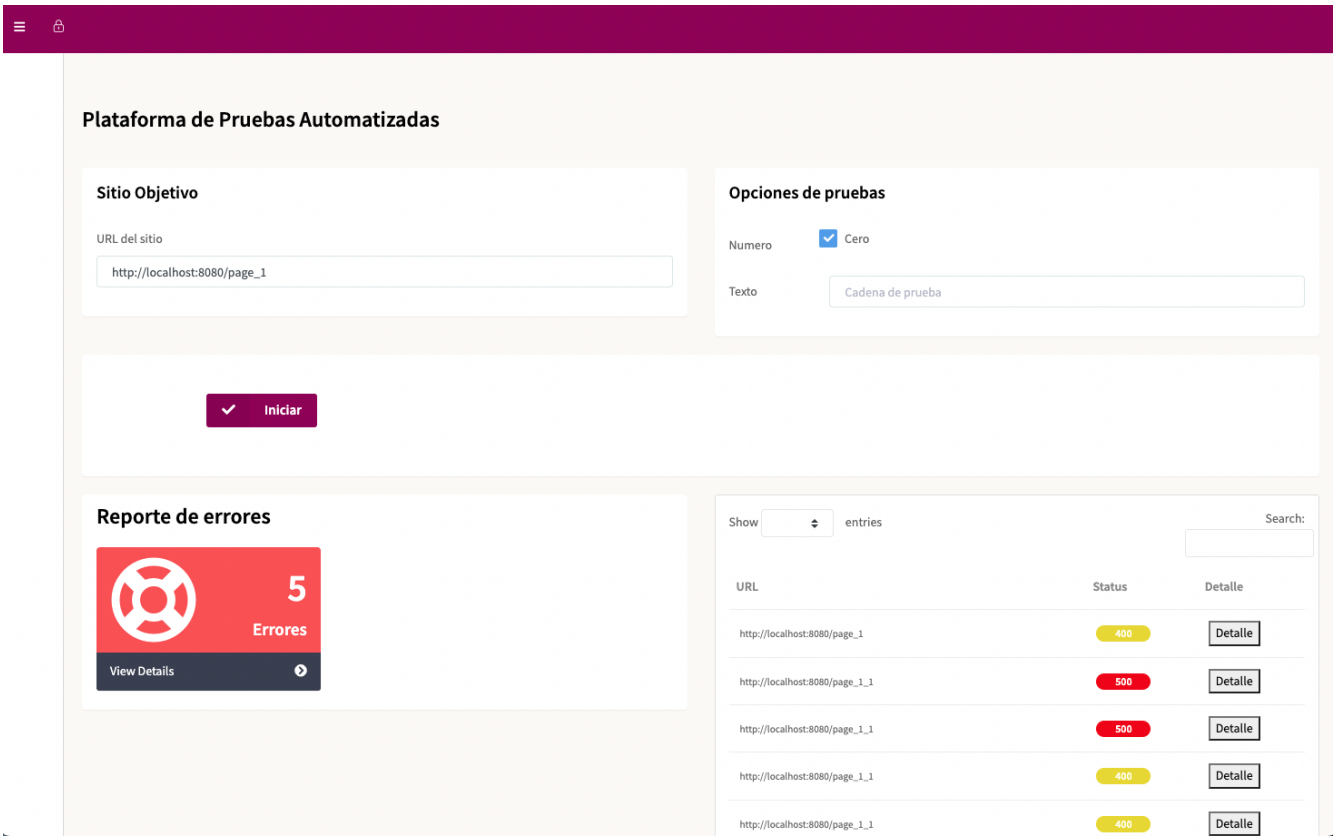


Figura 22 Interfaz del sistema de pruebas autónomas empaquetado como un framework

Al terminar el proceso, se puede observar el reporte de errores donde se indica el número de errores encontrados y del lado derecho el detalle de cada uno de estos errores.

Es posible observar el detalle de cada uno de los errores dando clic en el botón “Detalle”, se abrirá una ventana emergente donde se puede ver un Grafo que representa los estados desde la página inicial de la aplicación hasta el error, además de que se puede observar a detalle los valores de cada estado que llevaron a originarse el error final, lo cual es información de gran importancia para los desarrolladores ya que serán capaces de reproducir el error de forma fiel.

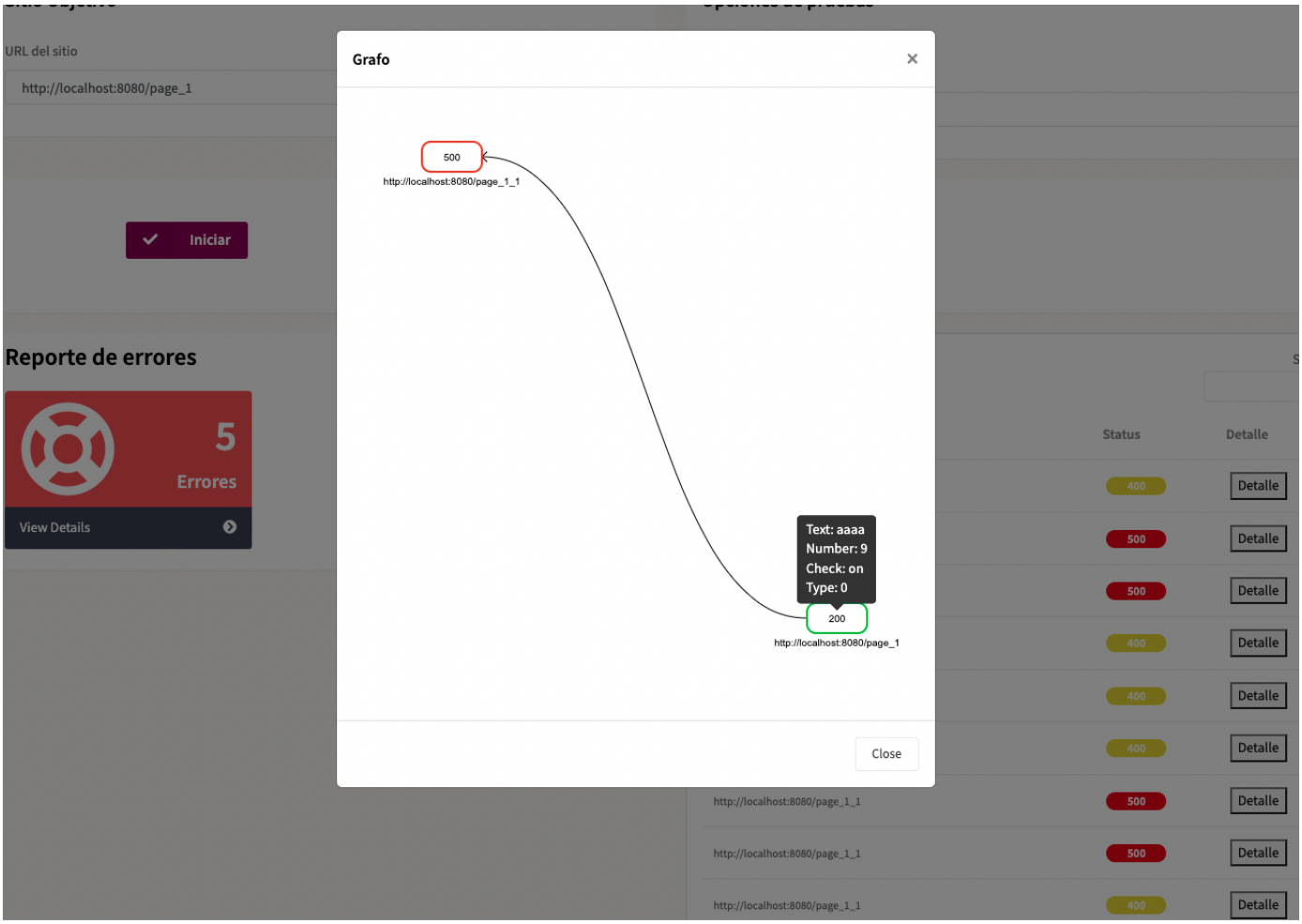


Figura 23 Representación de la secuencia de estados de un error por medio de un grafo

4.4 Pruebas

4.4.1 Diseño de las pruebas

Para realizar la validación y efectividad del sistema desarrollado se propone la simulación de una aplicación web formada por 7 páginas, cada una de estas páginas contiene un formulario con diferentes campos y opciones válidas, a los cuales se les incorporó de forma intencional una serie de errores que se presentarán después de llevar a cabo una serie de pasos específicos, en total se incorporaron 10 errores 500 y 35 errores 404.

Como se puede observar en el siguiente diagrama cada página cuenta con un formulario cada uno con un número diferente de opciones, el campo tipo "Text" solo puede tener una opción es decir que este es un campo obligatorio para el envío del formulario por lo cual siempre debe estar lleno, lo mismo sucede con el campo "Number" que es obligatorio, el campo "Checkbox" puede tener 2 opciones, True o False, mientras que el campo "Select" tiene diferente número de opciones que van de 5 a 8 opciones de acuerdo a la página .

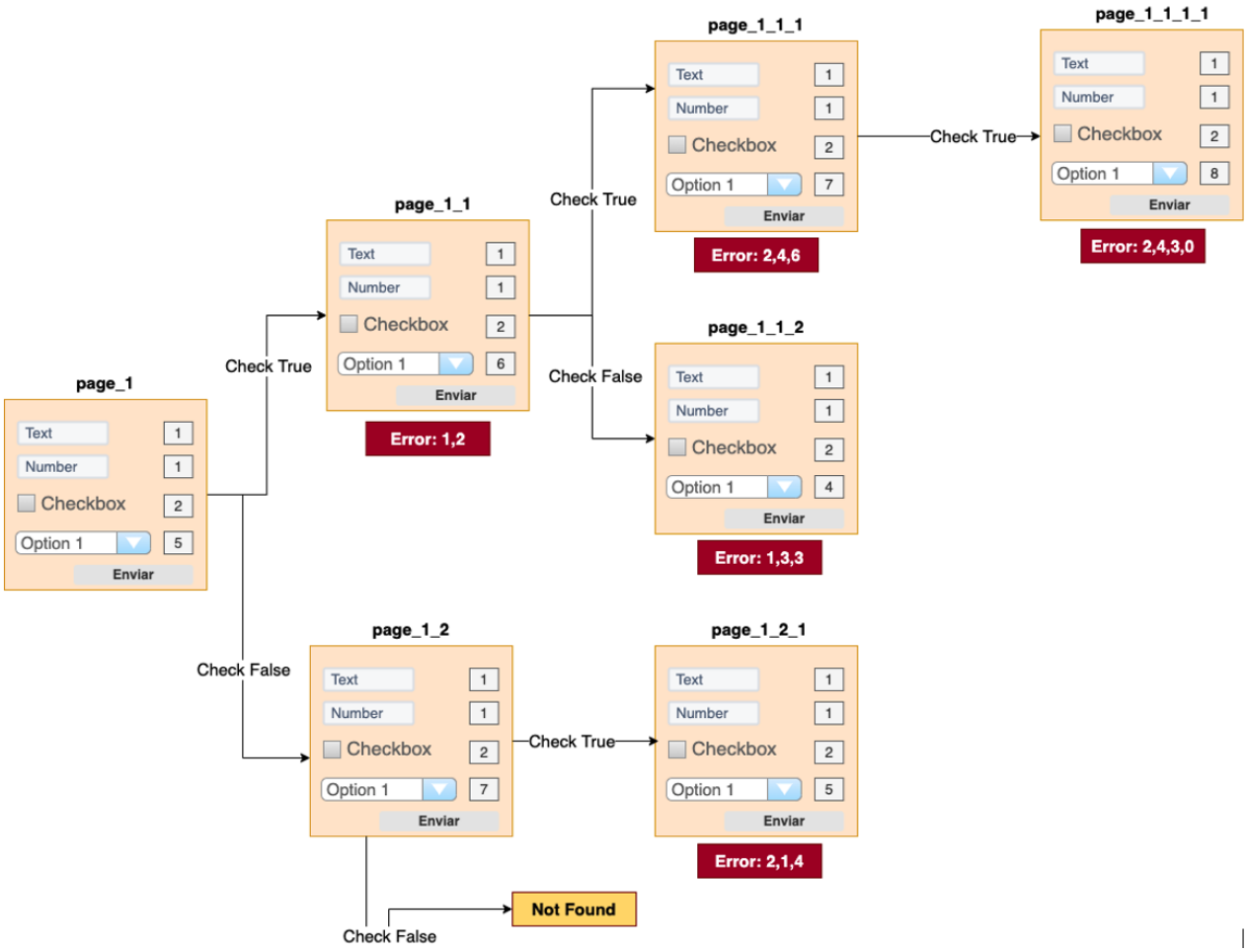


Figura 24 Diagrama de la aplicación modelo de pruebas con sus diferentes recursos y errores establecidos

A continuación se describen cada uno de los formularios de cada página con sus opciones válidas y cálculo de número de estados posibles:

Formulario	Estados
page_1 Text: 1 Number: 1 Checkbox: 2 Select: 5	1*1*2*5 = 10
page_1_1 Text: 1 Number: 1 Checkbox: 2 Select: 6	1*1*2*6 = 12
page_1_1_1 Text: 1 Number: 1 Checkbox: 2 Select: 7	1*1*2*7 = 14
	Subtotal Estados: (10/2)*(12/2)*(14/2)= 210
page_1_1_1_1 Text: 1 Number: 1 Checkbox: 2 Select: 8	1*1*2*8 =16
	Subtotal Estados: (10/2)*(12/2)*(14/2)*16= 3360
page_1_1_2 Text: 1 Number: 1 Checkbox: 2 Select: 4	1*1*2*4 = 8
	Subtotal Estados: (10/2)*(12/2)*8= 240
page_1_2 Text: 1 Number: 1 Checkbox: 2 Select: 7	1*1*2*7 = 14
	Subtotal Estados: (10/2)*(14/2) = 35
page_1_2_1 Text: 1 Number: 1 Checkbox: 2 Select: 5	1*1*2*5 = 10
	Subtotal Estados: (10/2)*(14/2)*10 = 350
	Total Estados = 3360 + 210 +240 + 35 + 350 = 4195

La combinación de estados posibles por formulario se calcula multiplicando las opciones disponibles por ejemplo del formulario de la sección page_1 sería: 1*1*2*5 = 10 estados posibles.

Y el número posible de estados dentro del ciclo de la aplicación aumenta rápidamente ya que se debe considerar los estados de los niveles previos, estos se pueden contabilizar al final de una "rama" por ejemplo consideremos la sección **page_1_1_1** para contabilizar el número de combinaciones de estados posibles hasta esta rama se debe considerar la multiplicación de los estados de los niveles previos y los propios por ejemplo:

page_1: 10
page_1_1 = 12
page_1_1_1 = 14

Sin embargo se debe considerar que en este caso existe una bifurcación entre cada nivel ya que dependiendo del estado del campo "Checkbox" el recurso destino será una página u otra, por lo cual se debe dividir entre 2 el número de estados (10/2)*(12/2)*(14/2) = 210, por lo cual en este caso se puede decir que existen 210 posibles combinaciones de estados que derivan desde la sección page_1 y que terminan en page_1_1_1.

La contabilización total de estados es la suma de las combinaciones posibles de cada rama terminal, en esta aplicación de prueba se tiene que son un total de 4195 posibles combinaciones de estados en todo el ciclo de vida de la aplicación, será este número de estados los que tendrá que explorar nuestro sistema de forma autónoma y dentro de ellos ser capaz de detectar los estados que desembocan en un error 500 o 404.

Los errores se asignaron en estados específicos y están basados en ciertos campos, para los errores 500 se basan en la secuencia de selección del campo Select a lo largo del ciclo de vida de la aplicación, estos errores se encuentran en las siguientes secuencias:

page_1_1: error 500
Secuencia de campo Select: 3,4

page_1_1_1: error 500
Secuencia de campo Select: 2,4,6

page_1_1_1_1: error 500
Secuencia de campo Select: 2,4,3,0

page_1_1_2: error 500
Secuencia de campo Select: 1,3,3

page_1_2_1: error 500
Secuencia de campo Select: 2,1,4

Por ejemplo tomando en cuenta el error 500 de la sección page_1_1_1_1 quiere decir que para obtener este error se debe seleccionar en el formulario de la sección page_1 la opción 2, en el formulario de page_1_1 la opción 4, en el formulario de page_1_1_1 la opción 3 y en el formulario de page_1_1_1_1 la opción 0 y en esta última fase el error puede surgir cuando el campo Checkbox es True o False, por esta razón cada error 500 tiene 2 versiones.

Los errores 404 se colocaron en el formulario de page_1_2 siempre que esté seleccionada la opción False del campo Checkbox de este formulario, por lo cual si se tiene en cuenta los 5 estados del formulario de la sección page_1 por las 7 posibilidades del formulario de page_1_2 se tienen un total de 35 errores 404.

Esta aplicación de prueba está basada también en el framework Django y la interfaz está basada en HTML, CSS y Javascript, lo cual permitió poder manipular fácilmente la asignación de errores para poder probar el sistema bajo diferentes escenarios.

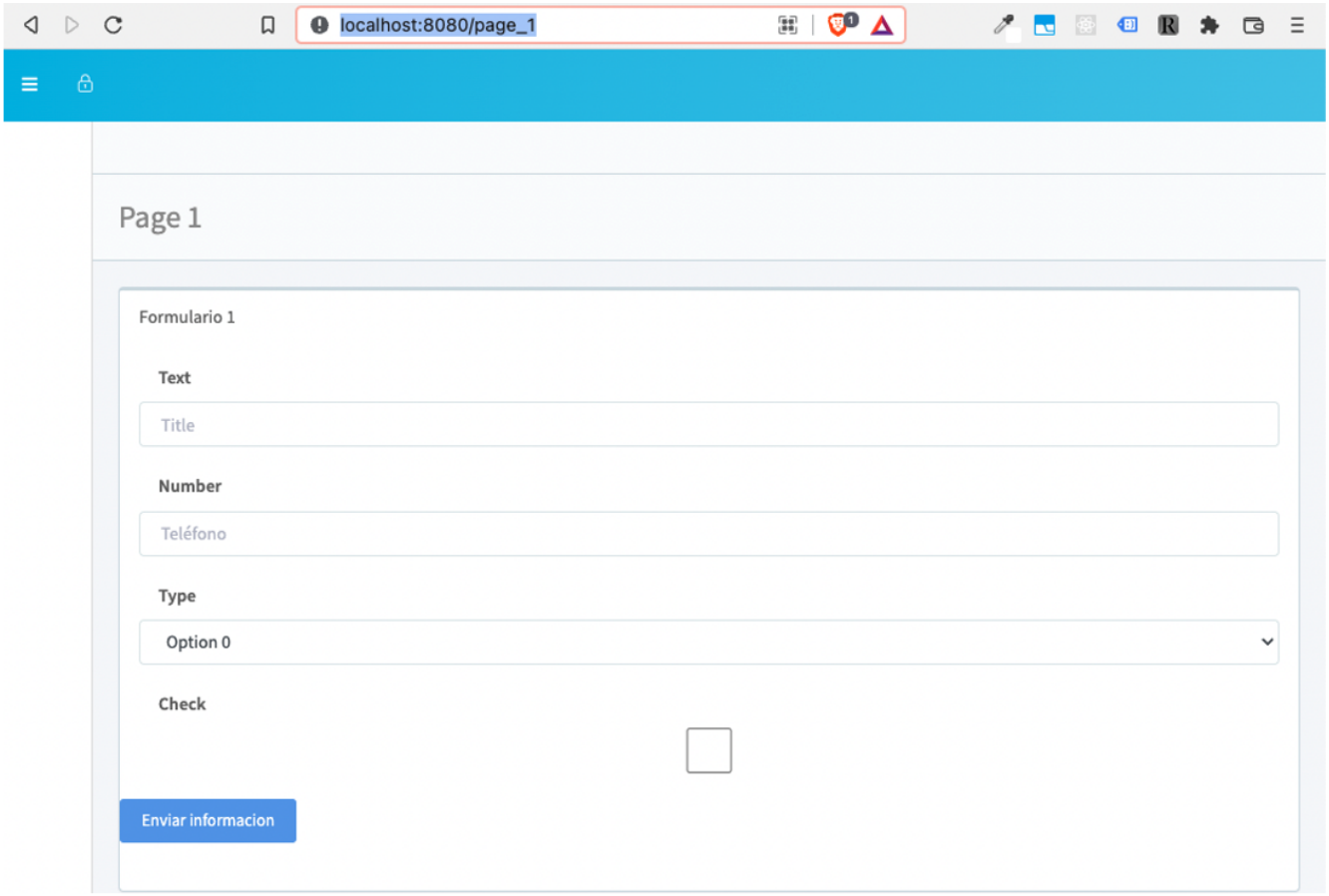


Figura 25 Vista de la interfaz de la aplicación web modelo de pruebas

Capítulo 5. Resultados y Conclusiones

Una vez diseñada e implementada la prueba para verificar el correcto funcionamiento de nuestro sistema modelo de pruebas automatizadas de aplicaciones web, conociendo de forma exacta qué errores esperamos y en qué estados de forma específica, se busca comparar con los resultados arrojados por el sistema, para ello se inicializa el servidor local donde se encuentra hospedado el sistema, se inicia sesión y una vez en la sección de pruebas se debe colocar la URL de la aplicación web a probar en este caso http://localhost:8080/page_1 que es la simulación de aplicación web diseñada y descrita en párrafos anteriores, cabe destacar que solo es necesario agregar la URL raíz de la aplicación web y el sistema se encargará de realizar la exploración autónoma de todos los recurso que componen la aplicación.

Se pueden modificar algunos parámetros adicionales como por ejemplo si se desea probar los campos de número con Cero o bien seleccionar un valor random y en el campo de Texto se puede seleccionar una cadena de prueba por default para probar.

Posteriormente solo se debe dar clic en el botón Iniciar.

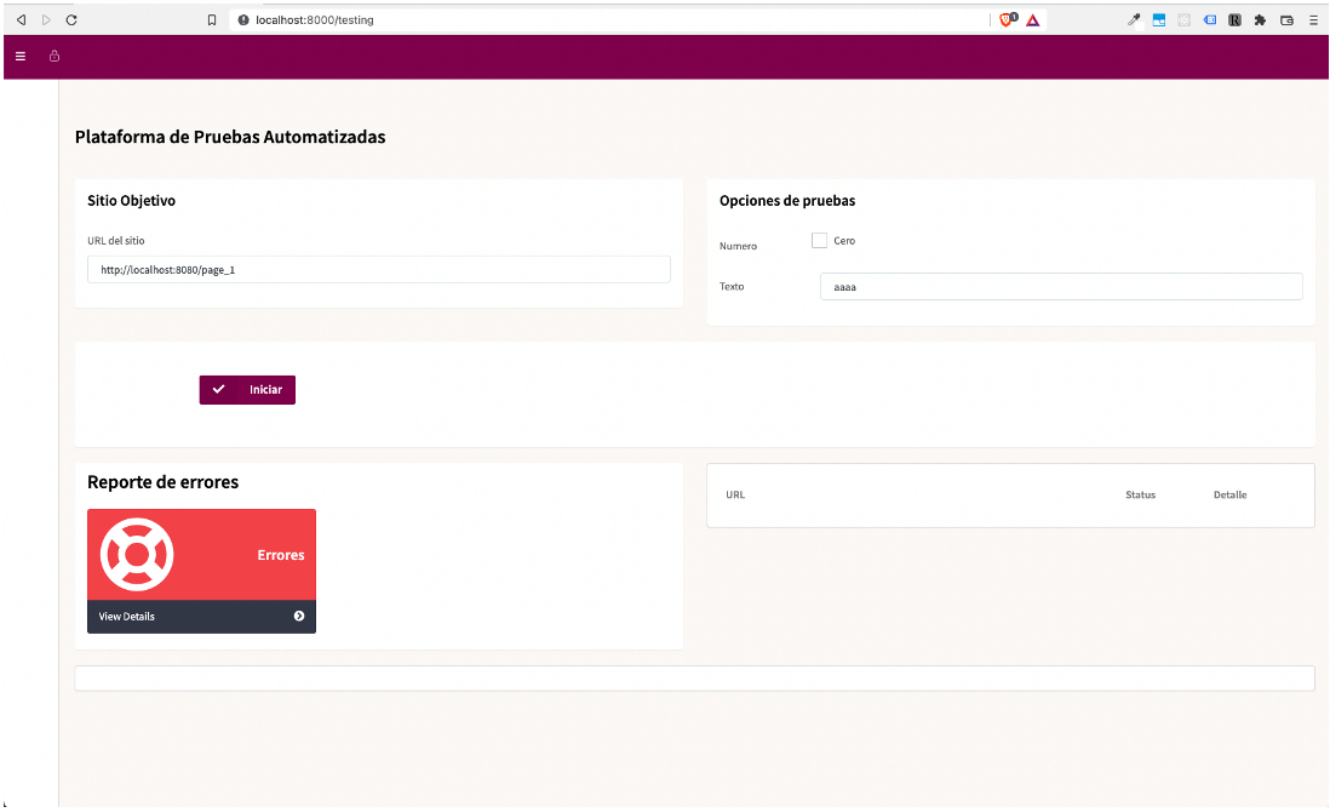


Figura 26 Interfaz del sistema de pruebas autónomas, antes de iniciar la búsqueda de errores.

El sistema comenzará a realizar las pruebas autónomas a toda la aplicación, este tiempo es variable y depende de la complejidad de la aplicación a probar, en la aplicación que se está probando en este ejemplo el tiempo de prueba fue de aproximadamente 10 minutos, durante este tiempo se muestra un icono indicando que se encuentra en proceso.

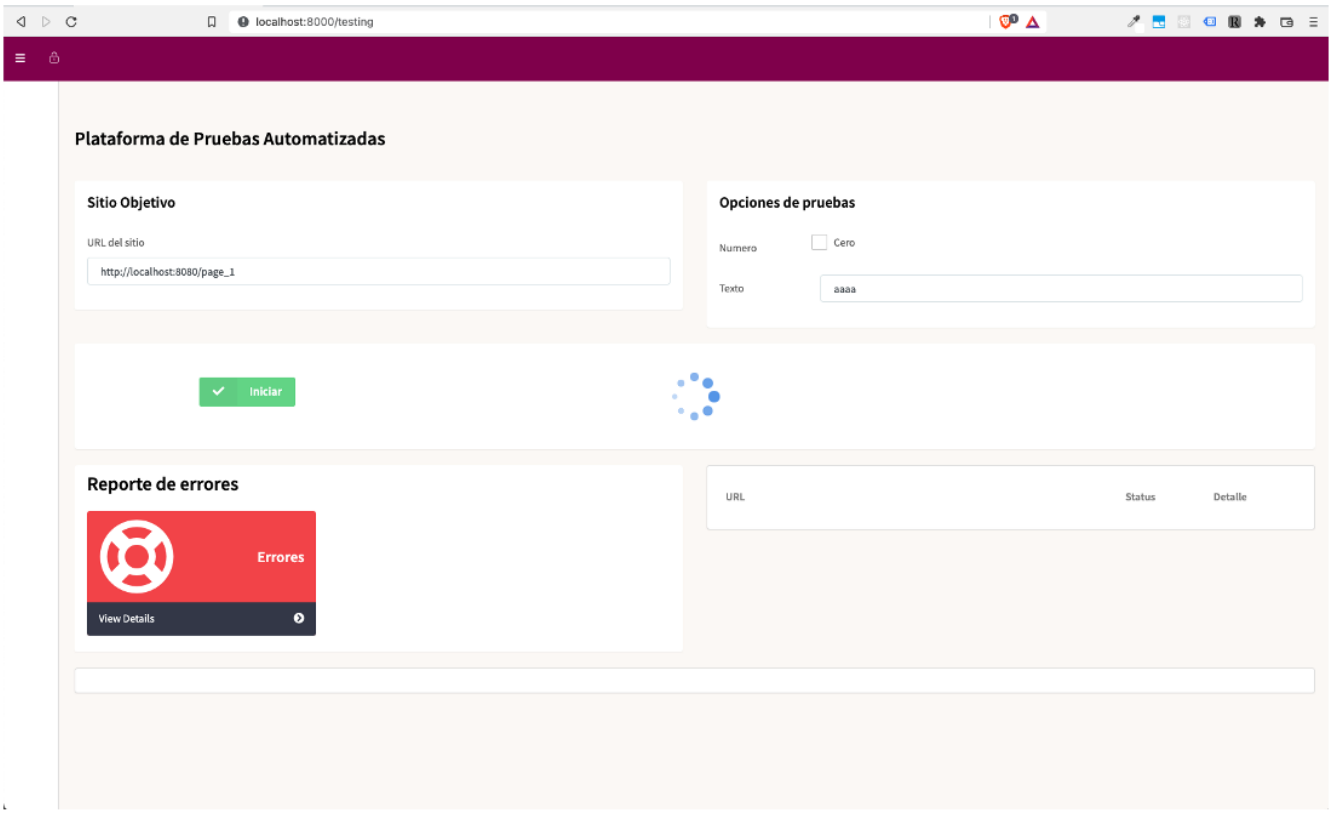


Figura 27 Interfaz del sistema de pruebas autónomas, durante la búsqueda de errores

Si bien el sistema solo debe comprobar 4195 estados lo cual puede parecer poco, en realidad este número crece exponencialmente ya que se debe considerar que durante el proceso de pruebas automatizadas el sistema debe explorar el árbol completo de la aplicación y para ello es necesario que por cada estado a probar se debe reproducir sus estados previos para continuar explorando los niveles y estados siguientes del árbol.

Una vez concluido el proceso en la parte inferior se muestra el número total de errores encontrados que en este caso fueron los 45 esperados, 10 errores 500 y 35 errores 404.

Del lado derecho se pueden filtrar los tipo de error por código, por ejemplo mostrar solo los errores 500.

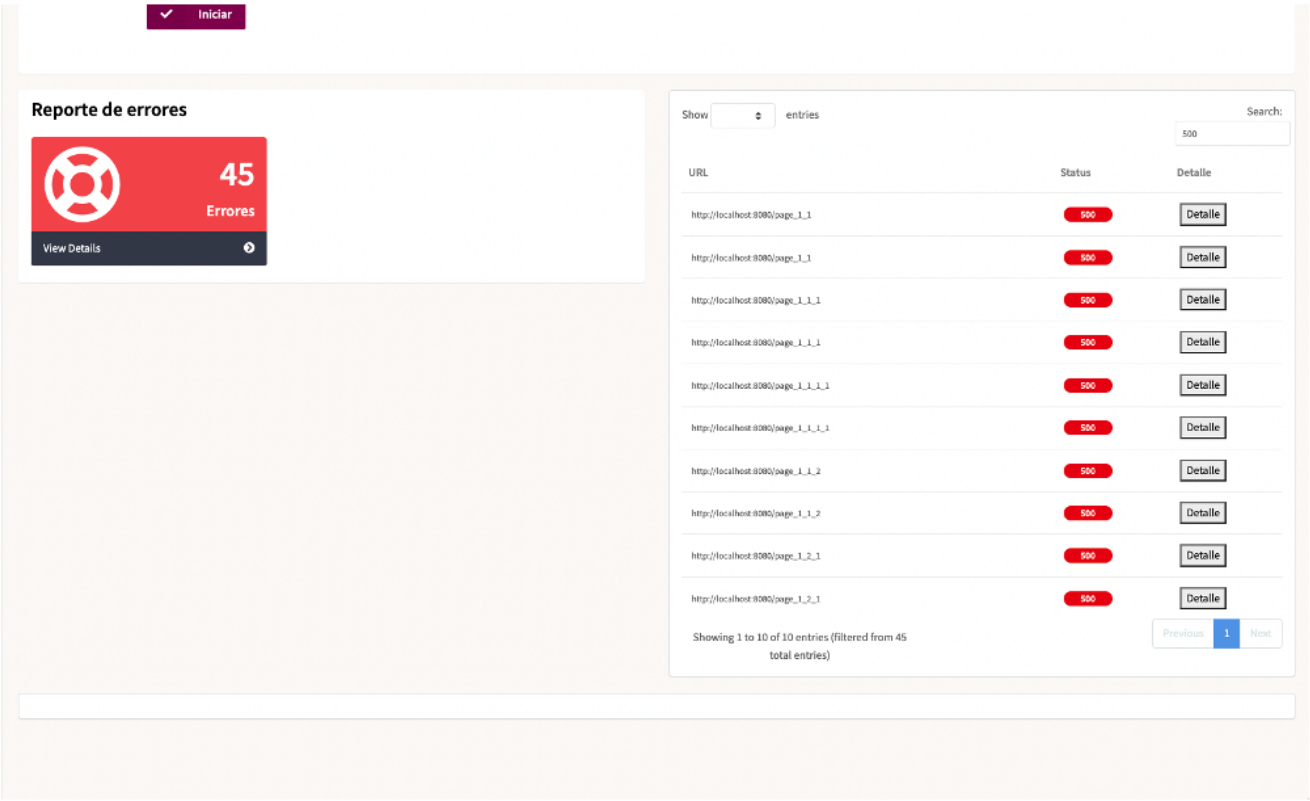


Figura 28 Interfaz del sistema de pruebas autónomas, al finalizar la búsqueda de errores

Para poder analizar un error en especifico, se debe dar clic en el botón detalle del error de interés y se desplegará una ventana emergente con un grafo que representa la rama del error con todos y cada uno de los estados que transcurrieron desde la raíz de la aplicación hasta el estado final donde se presentó el error y además es posible dar clic sobre cada estado para poder conocer los detalles de cada estado y de esta manera poder identificar qué combinaciones de estados originaron este error.

En el diagrama siguiente se puede ver el detalle del error de la sección page_1_1_1_1 en donde el error coincide con el introducido en la aplicación web de prueba:

page_1_1_1_1: error 500
Secuencia de campo Select: 2,4,3,0

En el ejemplo se puede observar el último estado tiene seleccionado el campo Select (Type) tiene valor 0 y en este caso el error es la versión con Check en estado false, se verificó y los estados previos coinciden con la secuencia dada: 2,4,3,0.

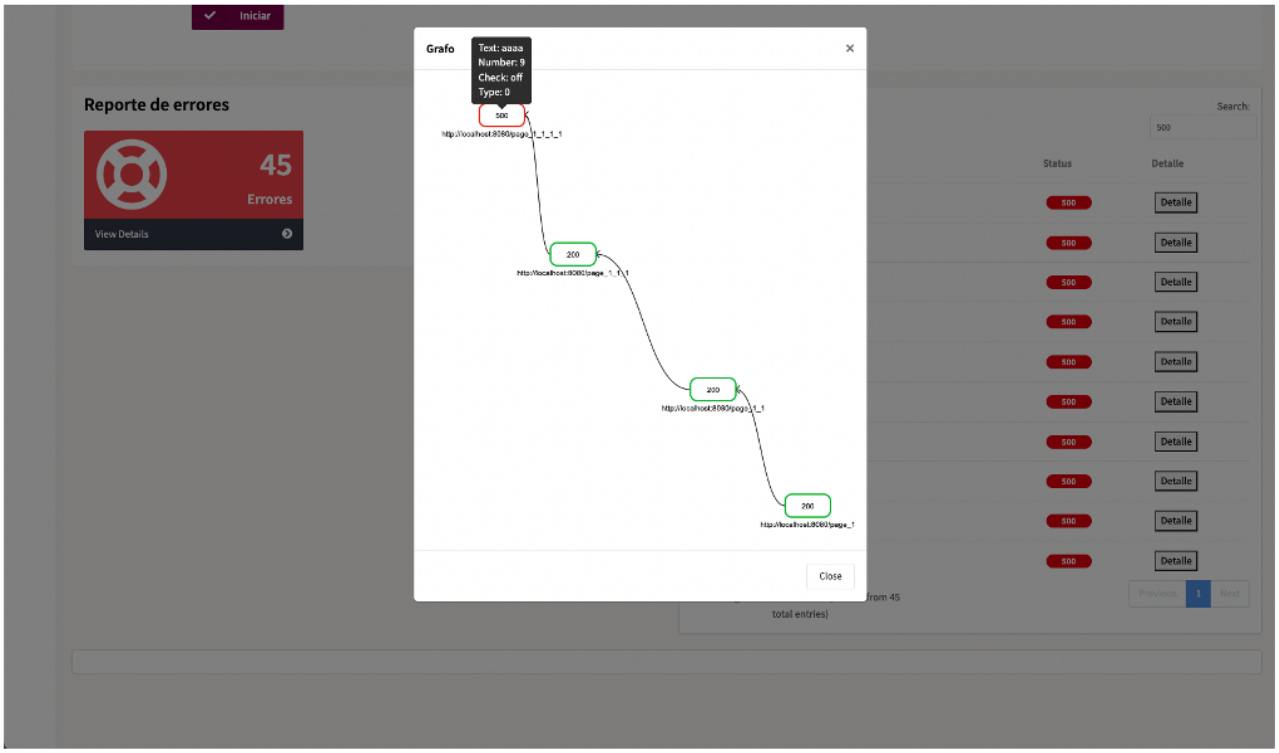


Figura 29 Detalle de un error 500 a través de su representación de estados en un grafo

De la misma forma se revisaron el resto de error 500 y coinciden fielmente con los errores introducidos en la aplicación web de prueba.

En el caso de los errores 404 también se pueden filtrar y se observa que se obtienen los 35 errores esperados.

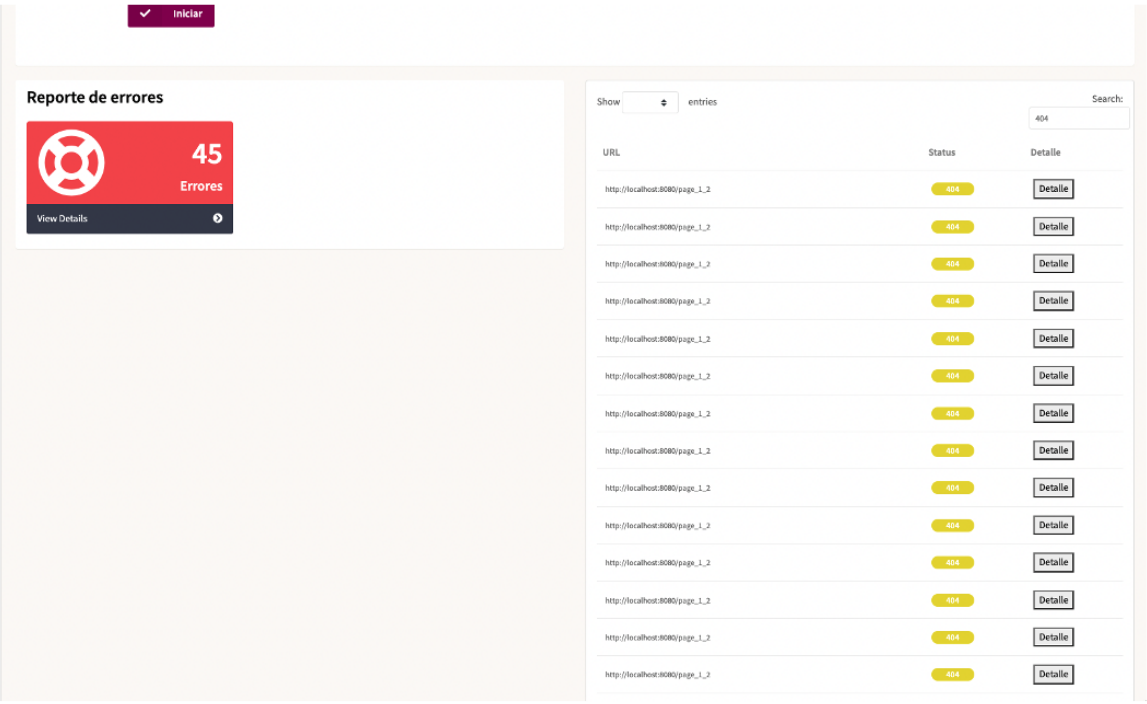


Figura 30 Listado de errores 400 encontrados

También se puede explorar el detalle del error, se confirma que el error coincide con el introducido en la aplicación de prueba.

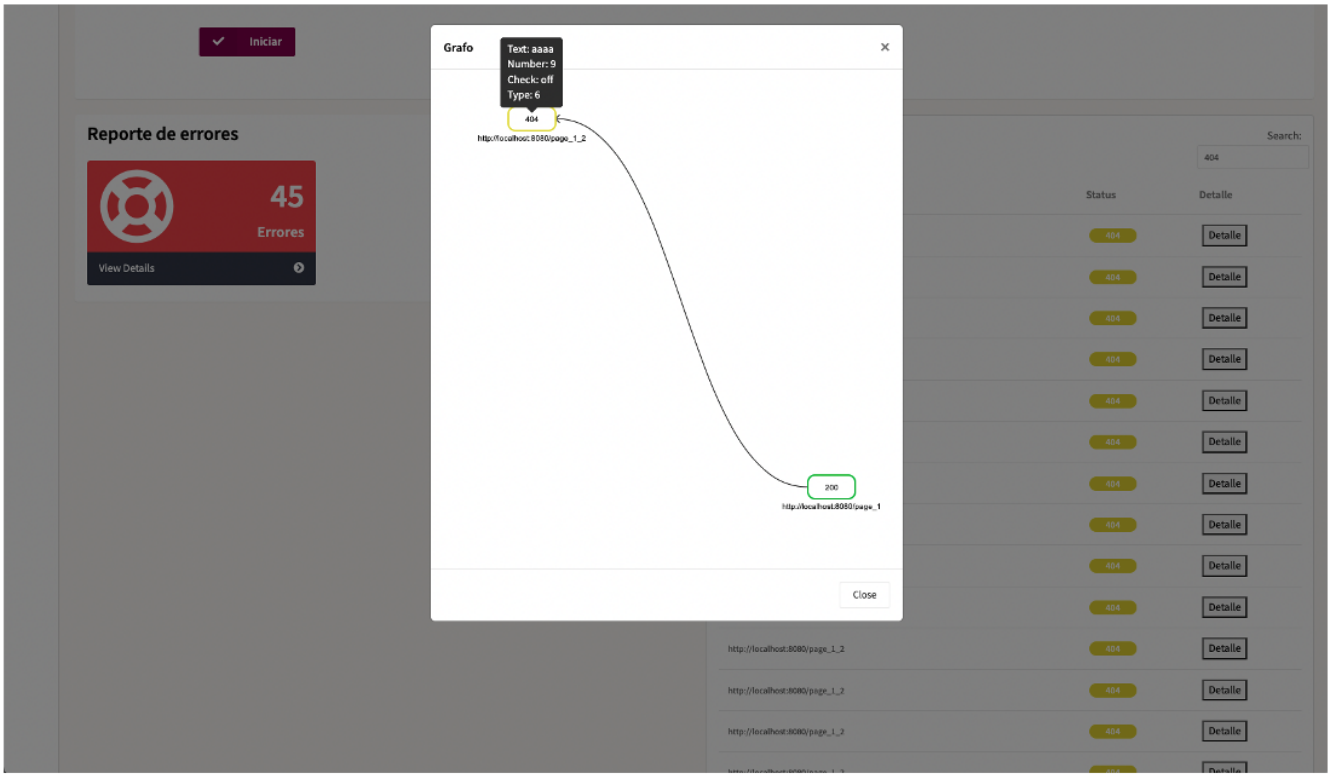


Figura 31Detalle de un error 400 a través de su representación de estados en un grafo

Adicionalmente se realizaron algunas pruebas adicionales modificando la secuencia de estados que producen los errores y se confirmó que también fueron encontrados de forma correcta.

5.2 Conclusiones

Podemos confirmar que se lograron cumplir con los objetivos propuestos al inicio de este trabajo, una parte fundamente fue poder realizar el acotamiento y definición del tipo de aplicación web estándar con la que se trabaría ya que como se pudo documentar en este trabajo hoy en día existen diferentes tipos de aplicaciones web que son muy diferentes en su arquitectura y en las tecnologías que están construidas por lo cual sería muy difícil ser capaces que este sistema funcionará con todas ellas, por lo cual para hacer viable este trabajo el acotamiento fue fundamental para poder dirigir los esfuerzos en el resolución de los problemas y retos del tipo de aplicación web estándar, que si bien es un tipo de aplicación web básica y ya casi en desuso debido a la irrupción de nuevas tecnologías en el desarrollo web, se logró constatar que este acercamiento en cuanto a poder realizar pruebas completamente automatizadas es posible y que si bien otros tipos de aplicaciones pueden tener retos mayores para implementar un sistema que funcione con ellas, se tiene un punto de partida para lograr hacerlo viable y de esta manera contar con una herramienta más que permita a los equipos de desarrollo incrementar la calidad del software que se desarrolla.

Otro punto que conviene destacar es el uso de las técnicas que se usaron para resolver el problema planteado, si bien inicialmente la hipótesis era el uso del aprendizaje por refuerzo como herramienta principal para poder resolver el problema planteado y aunque se mencionó la posibilidad de uso de otros tipos de algoritmos, al final si se utilizó aprendizaje por refuerzo como herramienta para resolver el problema del reconocimiento autónomo de los campos de formularios y opciones válidas para enviar una petición POST válida, sin embargo no se uso al 100% como se tenía planeado que era lograr el entrenamiento del modelo para poder resolver el problema presentado de forma eficiente, ya que durante el transcurso del desarrollo de este trabajo se encontró que un entrenamiento del algoritmo de aprendizaje por refuerzo en una aplicación web determinada no funciona al aplicarse en otra aplicación web diferente por lo cual el alcance del algoritmo de aprendizaje por refuerzo se limitó a la exploración y validación inicial de los formularios de cada recurso, sin llegar a utilizar su fase “entrenada”, en contraste se hizo uso de otras técnicas que no se tenían planeada inicialmente y que al final resultaron fundamentales para la resolución del problema, como lo fue la búsqueda a la profundo en una estructura de Árbol.

Como parte de los objetivos se estableció poder empaquetar el sistema de pruebas automatizadas en un framework que permitiría su uso de forma sencilla sin tener que incurrir en configuraciones complejas de un entorno o sin tener que hacer adaptaciones a la aplicación web a probar, este objetivo se logró con éxito al lograr encapsular el sistema en una sencilla aplicación web que puede ser desplegada en cualquier servidor en la nube e incluso en entornos locales ya que se encuentra contenerizada vía Docker lo que permite su replicación en cualquier tipo de infraestructura de forma rápida, sin embargo se debe considerar que dependiendo el tipo de aplicación a probar, los recursos requeridos en cuanto a memoria y poder de procesamiento pueden ser extensivos.

5.3 Trabajo Futuro

Durante el desarrollo de este trabajo se detectaron varias áreas de oportunidad que pueden ser solventadas con implementaciones futuras, a continuación se listan:

Ampliar la compatibilidad del sistema para poder abarcar otros tipos de aplicaciones web más comunes en el entorno actual, para ello será necesario centrarnos principalmente en ser capaces de soportar aplicaciones web desarrolladas con los frameworks de desarrollo web frontend más comunes hoy en día como React, Angular, Vue, Ember, etc. todos estas tecnologías tienen en común que son aplicaciones del tipo Single Page por lo cual no es posible interactuar directamente con el HTML de la aplicación y sería necesario usar un navegador web intermediario para poder interactuar con este tipo de aplicaciones por lo cual un reto será resolver el tema de velocidad y eficiencia de un sistema con estas características.

Otro punto considerado como trabajo a futuro es añadir soporte a las pruebas automatizadas de API REST ya que siguiendo las tendencias del desarrollo web moderno, una alternativa para realizar pruebas de integración de una aplicación web moderna es desde los end points de su interfaz API, lo cual es viable ya que se mantiene intacto los métodos HTTP y se tiene la ventaja que se interactúa directamente con las respuestas en texto plano sin tener que preocuparnos por interpretar la renderización de la interfaz como pasaría en una aplicación Single Page.

Finalmente un punto importante a considerar a futuro es añadir soporte para la paralelización del proceso de pruebas y con ello poder contribuir a disminuir los tiempos de prueba que pueden ser muy largos sobre todo al analizar aplicaciones web completas con millones de posibles estados, esto se considera viable ya que la búsqueda en un árbol por la naturaleza del problema es paralelizable, el reto radica en encontrar un método eficiente que pueda de forma automática dividir el problema completo en problemas más pequeños sin comprometer la integridad de los resultados.

Anexos

1 Implementación del agente Q Learning

```
def training(entorno, submit):
    discount = 0.9
    learning_rate = 0.1
    //cambiar numero de epocas de entrenamiento
    for _ in range(1):
        //inicializar entorno en cada epoca
        entorno.initialize()

        //obtener un estado de forma aleatoria.
        cur_pos, action = random.choice(list(environment_matrix.items()))
        counter = 0
        //mientras no se envie el formulario
        while(entorno.http_response == ''):
            counter = counter +1
            // obtener todas las acciones posibles a aplicar
            possible_actions = entorno.get_all_possible_actions()
            // seleccionar una acción de forma aleatoria
            action = random.choice(possible_actions)
            // obtener el siguiente estado en base a la acción seleccionada
            next_state = entorno.get_next_state(action)
            // actualizar la matriz Q
            q_matrix[cur_pos][action] = q_matrix[cur_pos][action] + \
            learning_rate *(environment_matrix[cur_pos][action] + \
            discount * max(q_matrix[next_state]) - q_matrix[cur_pos][action])

            // actualizar estado actual
            cur_pos = next_state
            print('Estado Actual:',cur_pos)
        //si la accion tomada resulto en un submit exitoso, guardar estado
        submit.append(cur_pos)
        print('Estatus HTTP: ', entorno.http_response)
        print('Iteraciones: ', counter)
        print("Episodio ", _ , " done")
```

2 Implementación de módulo CrawForms

```python
from mechanize import Browser

class CrawForms:
    def __init__(self, url):
        self.browser = Browser()
        self.browser.open(url)
        self.quantity = 0
        self.results = []
        self.get_field_forms()

    def get_field_forms(self):
        forms = self.browser.forms()
        nr_form = 0
        for form in forms:
            #print(form)
            print("Form name:", form.name)
            field_dict = {}
            self.browser.select_form(nr=nr_form)
            for control in form.controls:
                print ("type=%s, name=%s value=%s" % \
                (control.type, control.name, [control.name]))

                if control.type == 'select' or control.type == 'text'\
                    or control.type == 'number' or control.type == 'checkbox':
                    if control.type == 'select':
                        options = self.browser.form.possible_items(control.name)
                        field_dict[control.name] = [control.type, options]
                    else:
                        field_dict[control.name] = [control.type, '']

            self.results.append(field_dict)
            nr_form = nr_form + 1

        self.quantity = nr_form
```

3 Implementación de módulo Generador de estados

```
class Node:
    def __init__(self,data):
        self.data = data
        self.prev_node = None

class Generator:
    def __init__(self, range_list):
        self.root = None
        self.leaves = []
        self.counter = 0
        self.states = {}
        self.build_tree(range_list, None)
        #obtener caminos y reconstruirlos
        self.get_all_paths()

    #agregar nodo al arbol
    def append(self, prev_state, data):
        new_state = Node(data)
        if self.root == None:
            self.root = new_state
            return new_state
        else:
            new_state.prev_node = prev_state
            return new_state

    #reconstruir camino desde la hoja hacia la raiz
    def print_backward_path(self, leaf):
        combination_state = []
        current_node = leaf
        while current_node != None:
            combination_state.append(current_node.data)
            current_node = current_node.prev_node
        return combination_state
```

```
    #construir arbol en base a los campos brindados
    def build_tree(self, range_list, node_to_append):
        reduced_list = range_list[:]
        if len(reduced_list) != 0:
            reduced_list.pop(0)
        for list in range_list:
            for element in list:
                current_node = self.append(node_to_append, element)
                if len(range_list) == 1:
                    self.leaves.append(current_node)
                self.build_tree(reduced_list, current_node)
            break

    #obtener todas las combinaciones posibles
    def get_all_paths(self):
        counter = 0
        for leaf in self.leaves:
            combination = self.print_backward_path(leaf)
            hash_str_list = [str(int) for int in combination]
            hash_str = ",".join(hash_str_list)
            self.states[hash_str] = combination
            counter = counter + 1
        self.counter = counter

range_list = [[0,1,2], [0,1], [0,1]]
new_tree = Generator(range_list)
print('states', new_tree.states)
```

Bibliografía

[1] Z. Y. Y. G. X. C. Y. Li, «A Deep Learning based Approach to Automated Android App Testing,» *Key Laboratory of High Confidence Software Technologies,* 2019.

[2] R. S.-C. e. a. L. Harries, «DRIFT: Deep Reinforcement Learning for Functional Software Testing,» *Deep Reinforcement Learning Workshop,* 2019.

[3] S. M. D. P. Nguyen, «Codeless web testing using Selenium and machine learning,» *International Conference on Software Technologies,* 2020.

[4] J. Eskonen, «Deep Reinforcement Learning in Automated User Interface Testing,» *Aalto University,* 2019.

[5] D. Parsons, «Dynamic Web Application Development Using XML and Java,» *Cengage Learning EMEA,* p. 5, 2008.

[6] G. &. F. A. D. Lucca, «Web Application Testing,» de Web Engineering,» pp. 219-260, 2006.

[7] Mozilla, «JavaScript Guide,» [En línea]. Available: ttps://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction. [Último acceso: 2020].

[8] B. A. S. P. T. M. GUPTA, «SOFTWARE ENGINEERING & TESTING,» Jones and Bartlett Publishers, 2008, pp. 161-162.

[9] G. &. F. A. D. Lucca, «Web Application Testing,» pp. 223-227, 2006.

[10] M. A. Umar, «Comprehensive study of software testing: Categories, levels, techniques, and types,» *International Journal of Advance Research, Ideas and Innovations in Technology,* 2019.

[11] «Code Coverage vs Test Coverage : A Detailed Guide,» [En línea]. Available: https://www.browserstack.com/guide/code-coverage-vs-test-coverage. [Último acceso: 2020].

[12] D. &. H. T. Almog, «What Is a Test Case? Revisiting the Software Test Case Concept,» *Communications in Computer and Information Science,* 2009.

[13] B. K. R. Ferguson, «Software Test Data Generation using the Chaining Approach,» *INTERNATIONAL TEST CONFERENCE,* 1995.

[14] G. Tassey, «The economic impacts of inadequate infrastructure for software testing,» 2002.

[15] Adobe, «Understand web applications,» [En línea]. Available: https://helpx.adobe.com/gr_en/dreamweaver/using/web-applications.html. [Último acceso: 2021].

[16] «SPA vs MPA: Which One is Better For You?,» [En línea]. Available: https://yojji.io/blog/spa-vs-mpa. [Último acceso: 2021].

[17] R. S. S. a. A. G. Barto, «Reinforcement Learning: An Introduction,» *Stanford University,* 2014.