Reporte experiencia - Pruebas de Reconocimiento

Monkey Testing

- **Descripción:** Monkey testing (también conocidas como pruebas aleatorias) es una técnica de pruebas de reconocimiento que consiste en la generación aleatoria de eventos y entradas sobre la interfaz gráfica o inyección de datos aleatorios a una función a una aplicación ya sea web o móvil.
- **Experiencia:** Teniendo en cuenta el uso de la herramienta **cypress** para el desarrollo y ejecución del **Monkey Test** en nuestra experiencia consideramos que la definición de script de automatización fue un poco compleja al principio dado que nos toco revisar en detalle la documentación de cypress para comprender la sintaxis y la semántica de las funciones que se estaba utilizando y como estas funciones se podían modificar para sacar mejor provecho en la generación de eventos y entradas sobre la interfaz gráfica de Ghost, por otro lado, consideramos que haber tenido experiencia en el lenguaje de JavaScript facilito la codificación del monkey para lograr los objetivos propuestos a la hora de la definición del monkey test. En cuanto a la ejecución del monkey tuvimos a lo largo de las pruebas errores de ejecución que se fueron solucionando a medida para lograr que el monkey se ejecutará correctamente, estos errores principalmente radicaron en el uso de versiones de cypress incompatibles con ciertas funcionalidades, errores de lógica con funciones de cypress, entre otras.
- **Ejecución de pruebas:** Con el fin de probar al máximo la interfaz gráfica de Ghost se realizaron diferentes cambios al código fuente de **monkey.js** y **smart-monkey.js**, entre estos cambios están: el completado automático de inicio de sesión permitiendo avanzar de esta etapa para explorar las otras funcionalidades, el cambio de versión de cypress a la 4.11.0 para habilitar la propiedad de ensureScrollable la extensión del scrollTo() y la especificación de los métodos de la librería faker para llenar campos de la interfaz con información aleatoria. Una vez realizado estos cambios se definió la siguiente estrategia de pruebas para la detección de errores y verificación de la calidad del producto:

Prueba	N. de eventos	Configuración monkey.js	Configuración smart-monkey.js
1	50	“seed”: 1234, “pctClicks”:19, “pctScroll”:17, “pctSelectors”:16, “pctKeys”:16, “pctSpKeys”:16, “pctPgNav”:16	“seed”: 1234, “pctClicks”:12, “pctScroll”:12, “pctSelectors”:12, “pctKeys”:12, “pctSpKeys”:12, “pctPgNav”:12, “pctBwChaos”:12, “pctActions”:16
2	50	“seed”: 1234, “pctClicks”:24, “pctScroll”:4, “pctSelectors”:24, “pctKeys”:14, “pctSpKeys”:14, “pctPgNav”:20	“seed”: 1234, “pctClicks”:16, “pctScroll”:4, “pctSelectors”:16, “pctKeys”:10, “pctSpKeys”:16, “pctPgNav”:10,

			“pctBwChaos”:12, “pctActions”:16
3	100	“seed”: 1234, “pctClicks”:19, “pctScroll”:17, “pctSelectors”:16, “pctKeys”:16, “pctSpKeys”:16, “pctPgNav”:16	“seed”: 1234, “pctClicks”:12, “pctScroll”:12, “pctSelectors”:12, “pctKeys”:12, “pctSpKeys”:12, “pctPgNav”:12, “pctBwChaos”:12, “pctActions”:16
4	100	“seed”: 1234, “pctClicks”:24, “pctScroll”:4, “pctSelectors”:24, “pctKeys”:14, “pctSpKeys”:14, “pctPgNav”:20	“seed”: 1234, “pctClicks”:16, “pctScroll”:4, “pctSelectors”:16, “pctKeys”:10, “pctSpKeys”:16, “pctPgNav”:10, “pctBwChaos”:12, “pctActions”:16
5	200	“seed”: 1234, “pctClicks”:19, “pctScroll”:17, “pctSelectors”:16, “pctKeys”:16, “pctSpKeys”:16, “pctPgNav”:16	“seed”: 1234, “pctClicks”:12, “pctScroll”:12, “pctSelectors”:12, “pctKeys”:12, “pctSpKeys”:12, “pctPgNav”:12, “pctBwChaos”:12, “pctActions”:16
6	200	“seed”: 1234, “pctClicks”:24, “pctScroll”:4, “pctSelectors”:24, “pctKeys”:14, “pctSpKeys”:14, “pctPgNav”:20	“seed”: 1234, “pctClicks”:16, “pctScroll”:4, “pctSelectors”:16, “pctKeys”:10, “pctSpKeys”:16, “pctPgNav”:10, “pctBwChaos”:12, “pctActions”:16
7	100	“seed”: **876**, “pctClicks”:19, “pctScroll”:17, “pctSelectors”:16, “pctKeys”:16, “pctSpKeys”:16, “pctPgNav”:16	“seed”: **876**, “pctClicks”:12, “pctScroll”:12, “pctSelectors”:12, “pctKeys”:12, “pctSpKeys”:12, “pctPgNav”:12, “pctBwChaos”:12, “pctActions”:16
8	100	“seed”: **468**, “pctClicks”:19, “pctScroll”:17,	“seed”: **468**, “pctClicks”:12, “pctScroll”:12,

		“pctSelectors”:16, “pctKeys”:16, “pctSpKeys”:16, “pctPgNav”:16	“pctSelectors”:12, “pctKeys”:12, “pctSpKeys”:12, “pctPgNav”:12, “pctBwChaos”:12, “pctActions”:16

El código fuente con los ajustes realizados al monkey test se encuentra alojado en el repositorio siguiendo el siguiente enlace:

- https://github.com/JulianP911/Pruebas-Automatizadas/tree/main/Semana%204/Monkeys%20-%20Rippers/Monkey/Codigo/monkey-cypress

- **Resultados:** Una vez ejecutado el conjunto de pruebas definidos tanto para **monkey.js** y **smart-monkey.js** se obtuvo **3** ejecuciones con errores de un total de **16** ejecuciones, estas 3 ejecuciones en donde se obtuvieron errores se presentaron **monke.js** revisando los reportes y los videos generados por cypress se encontró que el primer error radica al momento de adjuntar una imagen en la cabecera de un post donde la página web no responde y se bloquea **(**Prueba 5 y 8**)** y el segundo error radica en un elemento dentro de la edición de un post que al hacer doble clic el elemento no responde y no realiza de forma correcta la acción indicada **(**Prueba 6**)**. A continuación, se adjunta el enlace donde se encuentran los reportes HTML y video generados para cada prueba generada:
  - **Enlace a los reportes y videos:** https://github.com/JulianP911/Pruebas-Automatizadas/tree/main/Semana%204/Monkeys%20-%20Rippers/Monkey/Reportes%20-%20Videos
- **Reporte de incidencias:** A continuación, se adjunta los enlaces correspondientes a las dos incidencias encontradas en la aplicación de Ghost:
  - **Incidencia 1:** https://github.com/JulianP911/Pruebas-Automatizadas/issues/20
  - **Incidencia 2:** https://github.com/JulianP911/Pruebas-Automatizadas/issues/21
- **Beneficios vs. Limitaciones:** A continuación, se lista los beneficios y limitaciones al realizar pruebas de reconocimiento de tipo monkey test:
  - **Beneficios:**
    - Ejecución rápida de los eventos definidos.
    - Detectar casos extremos que no son fáciles de detectar por el humano.
  - **Limitaciones:**
    - Más probabilidad de producir eventos aleatorios de entrada inválidos.
    - Falta de comprensión del contexto de la aplicación.

Exploración sistemática con Rippers

- **Descripción:** Un Ripper es una herramienta de software que automatiza la exploración y prueba de interfaces gráficas de aplicaciones, permitiendo encontrar defectos y maximizar la cobertura de estados de manera más inteligente y eficiente que enfoques aleatorios.
- **Experiencia**: Para la prueba, se utilizó la librería **RIPuppet**. En nuestra experiencia, tuvimos bastantes problemas para poder ejecutar la herramienta. Esto se debió a que la

aplicación que estábamos probando requería una versión más avanzada de NodeJS que la que manejaba RIPuppet. Debido a este conflicto de versiones, se decidió utilizar un entorno virtual para allí emplear la herramienta, mientras que en local se desplegó Ghost.

En cuanto a la codificación de la prueba, fue necesario agregar condiciones de espera entre los eventos que el Ripper encontraba de manera recursiva. Esto se decidió porque la página no alcanzaba a renderizarse antes de que el Ripper la evaluara y la prueba terminaba prematuramente. Una vez añadidas estas condiciones, se procedió a configurar el inicio de sesión. En este caso, se optó por no omitirlo. Por lo tanto, nuestra exploración comenzó desde la pantalla de inicio de sesión, donde se introducían datos predefinidos y el Ripper continuaba su exploración una vez se hacía clic en el inicio de sesión.

- **Ejecución de pruebas:** Teniendo en cuenta que el Ripper funciona de manera recursiva y por heurísticas, con solo un nivel de profundidad el Ripper alcanza a escanear las pantallas de las funcionalidades elegidas en la estrategia de pruebas. Para la configuración, como se mencionó anteriormente, se utilizó un entorno virtual. En este caso, se dejó las credenciales del inicio de sesión en la configuración y se agregaron condiciones de espera para la renderización en el código. A continuación, se muestran la configuración utilizada en la prueba:

Prueba	config

1	<pre>{
	"url": "http://localhost:2368/ghost/",
	"headless": false,
	"depthLevels": 1,
	"inputValues": true,
	"values": {
	"identification": "pepe@grillo.com",
	"password": "pepegrillo"},
	"browsers": ["chromium"]
	}</pre>

El código fuente con los ajustes realizados se encuentra alojado en el repositorio siguiendo el siguiente enlace:

  - https://github.com/JulianP911/Pruebas-Automatizadas/tree/main/Semana%204/Monkeys%20-%20Rippers/Ripper

- **Resultados**

Al realizar la prueba con 1 nivel de profundidad, se detectaron 33 nodos, de los cuales 4 arrojaron errores encontrados en la interfaz de usuario (nodos: 6, 10, 11, 12). Principalmente, los errores encontrados son de renderización de elementos, variables de estilo no definidas y recursos no encontrados. El detalle de los nodos y la evidencia se encuentra en el siguiente enlace:

  - https://github.com/JulianP911/Pruebas-Automatizadas/tree/main/Semana%204/Monkeys%20-%20Rippers/Ripper/results/2023-11-03T02.37.07.944Z/chromium

- **Reporte incidencias** A continuación, se adjunta los enlaces correspondientes a las incidencias encontradas en la aplicación de Ghost:
  - **Incidencia 1:** https://github.com/JulianP911/Pruebas-Automatizadas/issues/22
  - **Incidencia 2:** https://github.com/JulianP911/Pruebas-Automatizadas/issues/23
  - **Incidencia 3:** https://github.com/JulianP911/Pruebas-Automatizadas/issues/24
  - **Incidencia 4:** https://github.com/JulianP911/Pruebas-Automatizadas/issues/25

- **Beneficios vs limitaciones:** A continuación, se lista los beneficios y limitaciones al realizar pruebas de reconocimiento de tipo Ripper:
  - **Beneficios**:
    - **Exploración automatizada de la interfaz gráfica:** Los rippers pueden generar eventos y explorar la interfaz gráfica de manera más estructurada y sistemática que enfoques aleatorios. Esto permite una exploración más efectiva y detallada.
    - **Memoria y Cobertura**: Los rippers recuerdan los eventos ejecutados y los estados visitados, lo que les permite maximizar la cobertura y explorar una amplia gama de estados de manera eficiente.
    - **Construcción de Modelos**: Al extraer información de la interfaz gráfica, los rippers pueden construir modelos de los componentes visibles y sus interacciones. Esto puede ser valioso para comprender la estructura y navegación de la aplicación.
  - **Limitaciones**:
    - **Requerimientos de Recursos**: Los rippers pueden requerir más recursos de procesador en comparación con enfoques aleatorios. Esto puede afectar el rendimiento, especialmente en sistemas con capacidades limitadas.
    - **Configuración y Adaptación**: Es posible que sea necesario configurar y adaptar el ripper para que funcione correctamente con la aplicación específica que se está probando. Esto puede requerir tiempo, esfuerzo y conocimiento del código fuente de la herramienta.
    - **Velocidad de Ejecución**: En comparación con enfoques más simples como los "Monkeys", los rippers pueden ser más lentos debido al proceso adicional de construcción de memoria de eventos y aplicación de heurísticas.

Comparación Monkeys vs. Rippers

	Monkeys	Rippers
Configuración, codificación y ejecución	La configuración y codificación del script para el monkey es sencilla en base al conocimiento de la herramienta de uso, así como, la ejecución propia de cada prueba definida.	La configuración y codificación del script del ripper puede llegar a ser complicada si no se tiene conocimiento previo del lenguaje y del funcionamiento de este. Del mismo modo, dependiendo de la app que se quiere probar puede

		tener mayor complejidad en la configuración
Tiempo de ejecución	El tiempo de ejecución es corto. Es importante recalcar, que al aumentar el número de eventos aleatorios el tiempo de ejecución aumenta proporcionalmente**.**	A pesar de que el código puede ir bastante rápido, el ripper tiene como limitante que debe ir al ritmo de la renderización de la app que se está probando, incluso en algunos casos deben incluirse *delays* dentro del código para una correcta evaluación.
Cobertura	Los monkeys no tienen alta cobertura esto debido principalmente a que no tienen memoria y no siguen una heurística de exploración por lo cual pueden quedar estancados sobre una misma vista y no explorar las demás.	Ripper tiene una alta cobertura debido a que cada vez que ingresa a un enlace dentro de la aplicación, realiza un escaneo de todos los componentes accionables y enlaces dentro de la misma e interactúa con ellos.
Acceso a puntos muertos de la App.	Los monkeys tienen como desventaja conocer a priori que elementos son accionables, estos interactúan con cualquier elemento que este definido sobre la interfaz generando eventos inválidos.	Ripper tiene como ventaja su capacidad de escaneo de elementos accionables, esto le permite saber dónde hay un flujo disponible y evita acciones invalidas.
Configuración de inputs fijos	Los monkeys permiten a partir de la sección de un elemento o clase dentro del código fuente llenar inputs con información establecida con el fin de completar campos necesarios para poder explorar más funcionalidades sin que tenga que quedarse en un flujo estancado.	En el caso del Ripper, este permite configurar desde el inicio campos que deban ser completados con alguna información específica. Esto puede ser útil en casos como el login para evitar que la prueba termine en esta funcionalidad y no explore más allá.
Reporte de resultados	Cypress provee un reporte y un video de ejecución por cada prueba que se ejecute permitiendo seguir la traza de los eventos y encontrar los errores que se generan.	RIPuppet provee un reporte automatizado con el grafo de los pasos seguidos durante la ejecución y los resultados de las pruebas.

Enlace al video con la justificación de las dos estrategias y análisis solicitado:

- Reporte Monkey-Ripper.mp4