Detección automática de problemas de accesibilidad a partir de eventos de interacción de usuario

Maximiliano Jonathan Toledo, Julián Grigera[1], Alejandra Garrido[1]

LIFIA, Fac. de Informática, Universidad Nacional de La Plata, Argentina
[1]También CONICET, Argentina

{maximiliano.toledo, juliang, garrido}@lifia.info.unlp.edu.ar

Resumen. *Muchas actividades de nuestra vida cotidiana se encuentran integradas a aplicaciones web, las cuales deben ser accesibles a todos los usuarios. Por esto resulta fundamental incorporar la accesibilidad al proceso de desarrollo. Para facilitar esta integración y la detección de problemas de accesibilidad, en este trabajo se desarrolló una herramienta automática para la detección y reporte de este tipo de problemas denominada ABF (Accessibility BadSmells Finder). La misma busca ayudar a los desarrolladores a proveer sitios web accesibles, gracias a que realiza búsquedas dinámicas de diversos problemas de accesibilidad y que no son detectados por otras herramientas debido a la complejidad de su detección.*

Keywords: Bad Smell, Aplicaciones web, Accesibilidad, Análisis dinámico de eventos.

1. Introducción

La accesibilidad web está enfocada en brindar independencia para permitir autonomía a los usuarios de poder navegar por el contenido presente en la web, más allá de sus discapacidades. A pesar de la gran cantidad de estudios, herramientas y soluciones propuestas, la mayor parte del contenido web que se produce no tiene en cuenta las normas de accesibilidad ni las necesidades de personas con discapacidad [Carvalho2016; Carvaja2020]. Generalmente, las empresas de software tratan de disminuir el tiempo y los recursos afectados a los procesos de desarrollo para competir rápidamente en el mercado, y por ende no invierten en el proceso de accesibilidad de sus productos. Esto igualmente produce que el costo de la **no-accesibilidad** sea alto, lo cual incluye no sólo la pérdida de usuarios y clientes que no encuentran o no pueden acceder a lo que buscan, sino también sanciones que se pueden llegar a recibir por no proveer un producto accesible y eventuales costos de contratación de servicios externos para subsanar los problemas de accesibilidad. Por ende, es necesario que los desarrolladores de pequeñas y medianas empresas puedan contar con mecanismos y herramientas que les permitan generar aplicaciones web accesibles de una manera rápida y a bajo costo.

Con el fin de proveer un mecanismo que la industria pueda adoptar fácilmente para evaluar la accesibilidad de sus sistemas, la W3C ha creado las "Web Content Accessibility Guidelines" [WCAG]. Las mismas constituyen pautas o recomendaciones para lograr que el contenido web sea accesible, que se agrupan con sus criterios de éxito y posibles soluciones cuando los criterios no se cumplen. Las WCAG son consideradas estándares internacionales de accesibilidad Web y constituyen el instrumento más popular utilizado para guiar a los desarrolladores en la creación y evaluación de sitios web accesibles [W3C2005]. Sin embargo, y a pesar de cumplir con las recomendaciones de la W3C, una aplicación puede no ser usable o accesible, debido a que personas con

discapacidades podrían aún tener dificultades para navegar e interactuar con el contenido de manera cómoda, fácil y eficaz. Esto se da sobre todo en las aplicaciones con contenido dinámico, por ejemplo, en las llamadas Single Page Applications (SPA) o Rich Internet Applications (RIA), que cambian la página web y el estado de los widgets en la página a medida que el usuario interactúa con ella.

Asociadas a las WCAG se han construido herramientas de detección automática que ayudan a los programadores a identificar violaciones a las pautas [TAW]. Sin embargo, estas herramientas realizan un **análisis estático** a nivel de código y, por lo tanto, la cantidad de problemas que pueden encontrar es limitada. Por ejemplo, la Figura 1 muestra dos sitios web con mensajes de formularios o modales inaccesibles ya que un usuario con discapacidad visual nunca se entera de su presencia. Este problema no puede ser detectado por estas herramientas, dado que se trata de aplicaciones RIA y las herramientas solo chequean y analizan el código HTML estático. La principal característica de estos elementos es que no aparecen desde un principio cuando el usuario ingresa al sitio web, sino que son desplegados luego de alguna acción generada por el mismo usuario. Por esto, una búsqueda estática no es suficiente para detectar este tipo de problemas.

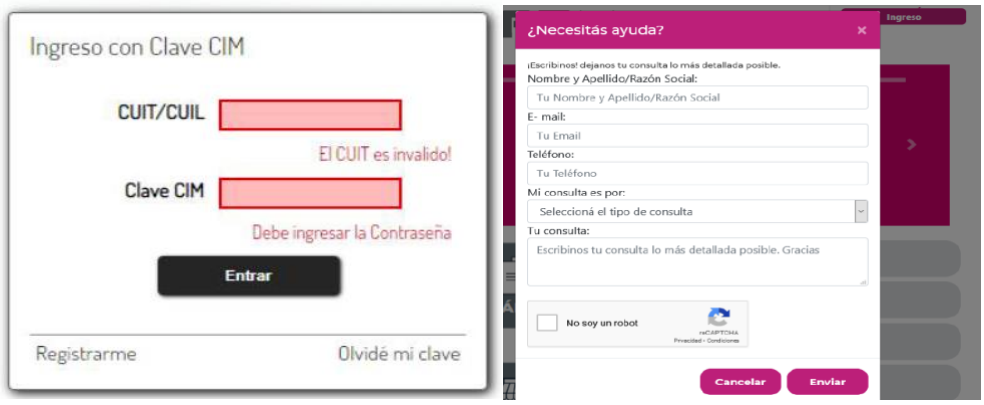

Fig. 1: Formulario de Ingreso con varios mensajes de error desplegados y modal desplegado para completar formulario web.

Para poder detectar estos problemas que aparecen a medida que un usuario interactúa con el sitio, es necesario realizar **búsquedas dinámicas** analizando los patrones de interacción generados por los usuarios. Existen diferentes propuestas en trabajos relacionados para encontrar problemas de accesibilidad que aparecen dinámicamente [Watanabe2016; Antonelli2019]. Sin embargo, solo pueden detectar un número acotado de problemas de accesibilidad, necesitan de la ayuda humana para poder realizar una detección eficaz, generan reportes que solo personas técnicamente preparadas pueden entender, simulan eventos de usuario de forma poco eficiente y generan falsos positivos, entre otras desventajas.

Este trabajo presenta un nuevo enfoque, basado en una herramienta denominada Accessibility BadSmells Finder (ABF)[1], que mediante el uso de varias funcionalidades del lenguaje JavaScript y una extensión web, permite realizar búsquedas dinámicas en aplicaciones web, y así lograr detectar elementos web que generen *accessibility smells* [Toledo2021]. El fin de esta herramienta no es solo detectar estos elementos problemáticos, sino además almacenar su información para luego generar reportes de

[1] Accesible desde http://sedici.unlp.edu.ar/handle/10915/125055

accesibilidad que sean de fácil lectura. Además, a través del desarrollo de algoritmos de detección (*finders*) individuales se logró una herramienta que puede ser extendida sin limitaciones para atacar diferentes tipos de problemas de accesibilidad.

2. Marco teórico y trabajo relacionado

2.1 Evaluación estática de la accesibilidad web

Existen varias herramientas que permiten la evaluación online de accesibilidad de sitios web como Level Access, TAW, AChecker y OAW. Estas herramientas online aplican métodos de **búsqueda estáticos en el código** de la página web, en busca de incumplimientos de las guías de accesibilidad WCAG. No obstante, hay varios problemas de accesibilidad que no pueden ser encontrados de esta manera. Por ejemplo, los botones hechos con elementos HTML `<a>` (para definir links) o `<div>` (para agrupar contenido), impiden a un usuario que utiliza un lector de pantalla poder encontrarlos, lo cual lleva a que realice varios recorridos dentro de la misma sección o página para lograr ubicarlos.

La W3C también ha creado una especificación técnica para el desarrollo de componentes RIA accesibles llamada WAI-ARIA [WAI-ARIA]. La misma permite agregar semántica a elementos dinámicos, especificando su tipo y funcionalidad a través de atributos, pero no se define la manera de asegurar su cumplimiento.

En definitiva, ninguna herramienta en sí misma puede determinar si un sitio cumple o no completamente con las pautas de accesibilidad [Antonelli2019]. Para determinar si un sitio Web es accesible, es necesaria la evaluación humana [W3C2005].

2.2 Refactoring para la mejora de la usabilidad y accesibilidad

Una propuesta para solucionar problemas de usabilidad o accesibilidad que aparecen en las aplicaciones web es el concepto de “Refactoring Web” [Garrido2011; Garrido2014]. La propuesta asociada a este concepto consiste en cambiar aspectos de la navegación o presentación de una aplicación web en producción sin cambiar su comportamiento, con el fin de mejorar su usabilidad o accesibilidad. El mayor inconveniente de esta técnica es que resulta muy difícil para una persona discapacitada reconocer qué refactoring instalar para solucionar los problemas que sufre.

Por otro lado, estos refactorings están catalogados dependiendo del problema o “smell” que solucionan, que en el caso de usabilidad se denominan **usability smells** [Garrido2011;Buono2020]. Grigera et al. han desarrollado una técnica de detección automática de usability smells en aplicaciones web que ya están en producción [Grigera2017]. Esta estrategia está basada en el análisis dinámico de los eventos de interacción del usuario. Sin embargo, se enfoca en las interacciones de usuarios videntes, ya que se basa principalmente en el análisis de eventos de mouse. En base a este trabajo, se ha desarrollado una extensión que permite procesar interacciones que incluyan eventos de accesibilidad [Durgam2020]. Para esta extensión se definió un catálogo de **accessibility smells** para usuarios con dificultades visuales que pueden detectarse automáticamente al observar la interacción a través de teclado y lector de pantalla. En este caso, la principal desventaja es que los reportes de smells deben ser leídos por personas con experiencia técnica y aun así en muchos casos es muy difícil identificar la causa concreta del problema; además existen algunos reportes que no pueden ser solucionados debido a que aún no existen refactorings desarrollados para este fin.

2.3 Detección de problemas de accesibilidad a través de la simulación de eventos

En el trabajo de Antonelli et al. se describe el desafío de las herramientas de evaluación automática de accesibilidad, que no son capaces de identificar componentes RIA en el código [Antonelli2019]. Los enfoques existentes aproximan esta identificación a través de la simulación de la interacción de usuarios [Fernandes2013; Watanabe2016].

El trabajo de Watanabe et al. propone una herramienta que identifica automáticamente a los elementos de tipo menú drop-down en una aplicación web [Watanabe2016]. Una vez que los elementos están identificados son analizados para determinar si están implementados utilizando las recomendaciones de ARIA. Esta herramienta está basada en la simulación de escenarios de interacciones de usuario, Mutation Observers y cambios de visibilidad en la interfaz del browser. Específicamente se simula a los usuarios generando eventos de desplazamiento en los elementos de una página web. Los cambios en el DOM son observados con Mutation Observers para clasificar elementos como menú desplegable o no, a través del análisis del resultado de la ejecución de código HTML, CSS y JavaScript dentro del navegador. Una vez detectados los menús desplegables, se analizan las reglas ARIA aplicadas a estos elementos y se generan los respectivos reportes. Las principales desventajas de este trabajo son la generación de falsos positivos por los simuladores y el gran tiempo requerido para cada simulación. Para evitar esta desventaja nuestro enfoque observa la ejecución de eventos de interacción reales, como describe la próxima sección.

3. Metodología

Nuestro objetivo es la detección automática de los problemas de accesibilidad que ocurren dinámicamente en una aplicación web, y que llamamos “accessibility smells”. En especial nos enfocamos en accessibility smells para usuarios con discapacidad visual que utilizan un lector de pantalla. Con este fin hemos desarrollado la herramienta ABF, que genera reportes con el listado de los diferentes accessibility smells detectados por la herramienta junto a la información detallada de los elementos de la página que generaron estos problemas. ABF está constituida por tres componentes principales (ver Figura 2):

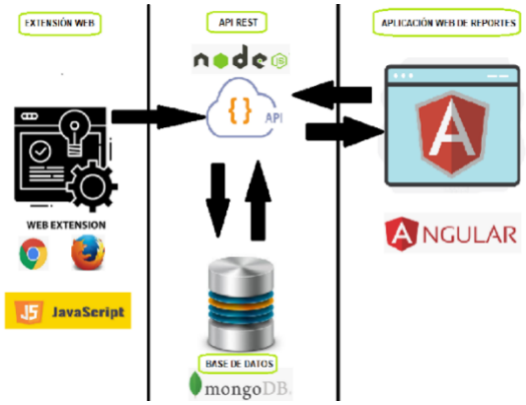

Fig. 2: Arquitectura de ABF

1. **Extensión web**: encargada de detectar elementos web que generen algún tipo de accessibility smell.

2. **API REST**: encargada de proveer la comunicación entre la extensión web, la base de datos y la aplicación de reportes.

3. **Aplicación de reportes**: aplicación web en la cual tanto el propietario del sitio web como los desarrolladores pueden visualizar los accessibility smells de su aplicación.

Dichos componentes integran las siguientes tres etapas dentro del funcionamiento de ABF, respectivamente: *Detección, Almacenamiento y manejo de la información*, y etapa de *Reporte*. La instancia de detección es la más importante de la herramienta, ya que es donde se analiza y se detectan los problemas de accesibilidad en páginas web. Un desarrollador que desee utilizar ABF, debe instalar la extensión web que implementa la detección dentro del mismo browser mientras está navegando en la aplicación web donde desea realizar la evaluación de accesibilidad. De esta forma ABF comenzará a observar los eventos *reales* que disparan los usuarios al acceder al contenido de su aplicación, sin necesidad de utilizar simulación. Cada vez que la extensión web detecte un elemento que genera un accessibility smell, se comunica a través del API REST para almacenar esta información en la base de datos, para luego poder ser consumida por la aplicación de reportes. Accediendo a este reporte cada cierto intervalo de tiempo, el desarrollador podrá obtener el listado de smells ocurridos en ese intervalo.

El componente de detección funciona a través de *finders*. Los finders son algoritmos que analizan el sitio web en busca de elementos que generen un tipo específico de accessibility smell. Varios de estos finders realizan búsquedas dinámicas, es decir que dependen de los eventos de interacción del usuario para poder detectar elementos inaccesibles. En este trabajo nos concentramos en tres tipos específicos de accessibility smells que fueron detectados en búsquedas manuales utilizando el lector de pantalla NVDA. Particularmente se analizaron manualmente más de 20 sitios web conocidos.

La búsqueda de elementos inaccesibles se enfocó en aquellos involucrados en la interacción del usuario, es decir, que mediante NVDA y la interacción utilizando teclado, no es posible acceder a dichos elementos. A continuación, se describirán los tres finders implementados para este trabajo.

3.1 Análisis de accesibilidad en elementos web interactivos con handlers JS asignados

Los **elementos interactivos** son aquellos que pueden reaccionar a eventos que disparan los usuarios y así agregan algún tipo de funcionalidad al sitio. De entre estos, podemos encontrar varios casos de elementos que simulan ser otros mediante el uso de *handlers JavaScript*. Por ejemplo: botones desarrollados con DIV's, elementos link `<a>` sin el atributo HREF con forma de botón y hasta elementos `<span>` que simulan ser links.

Visualmente parecen links y botones, con lo cual un usuario sin discapacidad visual los reconoce. Sin embargo, al utilizar un lector de pantalla como NVDA, estos elementos son ignorados, impidiendo que el usuario con discapacidad acceda contenido o funcionalidad, por ejemplo, impidiéndole el envío de un formulario. Durante el análisis manual preliminar encontramos este problema en elementos HTML de tipo: **div**, **span**, **li** y **a**. Por lo tanto, la búsqueda se centró en este tipo de elementos. No obstante, cabe destacar que no todos los elementos de esos tipos son inaccesibles, ya que algunos desarrolladores consideran la accesibilidad durante el proceso de desarrollo, y mediante la utilización de atributos ARIA y el uso consciente de JavaScript en el código, logran desarrollar elementos interactivos accesibles.

3.1.1 Desarrollo del finder

La sección principal del código del *finder* realiza dos acciones principales:

1. Búsqueda de handlers declarados en el código del elemento HTML. Ejemplo:

```
<div onclick="cancel()"></div>
```

2. Búsqueda de handlers utilizando la función *getEventListeners()* provista por el browser. Estos handlers pueden estar declarados como código JS dentro de tags `<script>` en el mismo documento .html de la página o en archivos .js externos.

Una vez que se detectó si un elemento de la página posee un handler asignado, se almacena junto a diversa información del elemento. Esta información incluye el tipo de elemento (div, span, a …), su ID, clases, los valores de varios atributos utilizados para brindar accesibilidad como *tabindex*, *aria_label* y *role*(ARIA), e información de los elementos hijos. Es importante notar que si los hijos de un elemento son accesibles gracias a sus atributos, el elemento padre también será accesible. Como validación extra se decidió agregar a cada uno de estos elementos con handlers JS, un *listener* JS, es decir, un objeto oyente de JS para escuchar eventos de foco. Así también podremos corroborar si el usuario logró interactuar con dicho elemento.

La información de los elementos que recibieron el foco es enviada a la API REST, quien es la encargada de almacenarlos en la base de datos, además se almacenan los eventos de foco que cada uno de estos elementos recibió. Luego esta información va a ser consumida por la aplicación de reportes. Para este Finder, se decidió que la lógica de analizar y reportar si un elemento es accesible o no, lo va a realizar la misma aplicación de reportes.

3.2 Análisis de accesibilidad en mensajes dinámicos de formulario web

Los formularios se utilizan para todo tipo de efectos interactivos en la web. Éstos permiten a los usuarios seleccionar y comprar productos, rellenar encuestas y cuestionarios, inscribirse en cursos, buscar información, etc.

Los formularios y otros elementos de interacción pueden ser confusos o difíciles de utilizar para muchas personas, lo cual las induce a cometer errores. Algunos ejemplos de ayuda para prevenir y corregir errores serían:

- Instrucciones descriptivas, mensajes de error y sugerencias de corrección.
- Ayuda contextual para funcionalidades e interacciones más complejas.
- Opción de revisar, corregir o revertir envíos si es necesario.

Esto incluye comentarios en línea o cerca de los campos del formulario y comentarios generales que generalmente se brindan después del envío del formulario. Además de los consejos anteriores, cada mensaje de error debe tener el atributo *role* con valor *alert*, para que puedan percibirlo los usuarios de lectores de pantalla.

3.2.1 Desarrollo del finder

Los mensajes de errores en los formularios son generalmente elementos HTML de texto como `<p>` y `<span>`. La detección de estos mensajes no se puede hacer de forma totalmente estática revisando el código de la aplicación web, ya que estos mensajes se pueden ir generando dinámicamente dentro o fuera del formulario. Por esto, la forma más efectiva de saber si los mensajes son accesibles es generarlos y analizarlos.

El desarrollo e implementación de este finder se basó en el trabajo de Watanabe et al. para identificar drop-down menú widgets utilizando “mutation observers” y “visibility changes” [Watanabe2016]. Es decir, un objeto incorporado a JS que observa

un elemento DOM y dispara una función callback cuando hay cambios en él (visibility changes). De esta forma podremos observar dinámicamente cuando aparecen estos mensajes de error mientras el usuario interactúa con el formulario.

En nuestro caso, el primer paso es obtener todos los formularios de la página web actual (elementos <form>). El segundo paso es asignarle a cada elemento `<form>` un listener JavaScript (MutationObserver) que captura los eventos que agregan un elemento al formulario. Luego, cuando un cambio ocurre, filtramos por tipo de elemento: solamente nos quedamos con los elementos **p**, **span** y **label**. En los casos en que estos elementos no poseen atributos rol o aria*, reportamos el elemento y se envía a la API REST para su almacenamiento.

3.3 Análisis de accesibilidad en elementos web flotantes

Los modales son interfaces bastante comunes en la Web. Los desarrolladores y diseñadores pueden darles diferentes nombres: lightbox, modal window, dialog, overlay, etc., pero son básicamente lo mismo. La W3C denomina a estos elementos flotantes como **dialogs** o **modal dialogs**. Por lo general, el resto de la página está atenuado y los usuarios no pueden interactuar con ella hasta que cierren el cuadro de diálogo. En términos de ARIA, este es un diálogo "modal", **modal dialog**.

Los modales al igual que las alertas, interrumpen el flujo de lo que hace un usuario si no puede controlar la apertura de un modal. Para mitigar esta problemática, se pueden agregar controles para la apertura y cierre del modal con botones. Sin embargo, también hay cuadros de diálogo "no modales", non-modal dialogs, que permiten la interacción con el resto de la página y en este caso se ven como "paneles" que flotan en la ventana principal. Estos paneles muestran algún contenido sobre la página web, mientras que el resto de la página continúa siendo 100% funcional y accesible.

Durante nuestra búsqueda utilizando NVDA para detectar problemas de accesibilidad relacionados a elementos flotantes, descubrimos que hay una gran presencia de non-modal dialogs en las diferentes páginas web testeadas. Encontramos sitios webs donde hay hasta más de 5 elementos non-modal dialogs presentes al ingresar a la página.

Los diferentes tipos de non-modal dialogs que podemos encontrar son: *Cookie notifications, Toast messages, Toggletips o Tooltips, Popups activados por el usuario con elementos interactivos*.

WAI-ARIA menciona dos pasos principales para desarrollar estos elementos:

1. **Asignación de roles**: El elemento dialogo, en general de tipo `<div>`, debe contener el rol *dialog*, el cual ayuda a las tecnologías de asistencia a identificar el contenido del diálogo como *agrupado y separado del resto del contenido de la página*.

2. **Asignación de propiedades y relaciones importantes**: Etiquetar a los elementos dentro del diálogo proporcionará información contextual sobre los elementos interactivos dentro del diálogo.

Si un diálogo ya tiene un elemento de título visible, el texto dentro de ese elemento se puede usar para etiquetar el diálogo. Para esto conviene usar el atributo *aria-labelledby* en el elemento *role="dialog"*. Si el diálogo contiene texto descriptivo además de su título, este texto se puede asociar con el diálogo mediante el atributo *aria-describedby*.

3.3.1 Implementación del finder

Durante la investigación para este caso, encontramos que todos los elementos analizados manualmente, están definidos con etiquetas `<div>`. Esto nos provee un punto de partida para detectar este tipo de elemento. Sabiendo que los elementos flotantes que buscamos son `<div>`, debemos encontrar un patrón de diseño que los caracterice y diferencie de los demás *DIVs* de la página. Los modales generalmente poseen estilos CSS particulares que hacen que aparezcan por encima del contenido, diferenciándose del contenido principal. Comúnmente se utilizan las propiedades CSS como **z-index** (mayor a 0 para posicionarse por encima del contenido principal de la página), **Right, Bottom, Left, Top** (para posicionar al elemento en alguna esquina de la pantalla). Siempre están fijos en una misma posición de la pantalla mientras el contenido principal de la página se mueve con la navegación en el contenido), **Position**: fixed y **Display**: flex.

Por lo tanto, nos quedamos con los `<div>` que tengan definidos los estilos CSS mencionados anteriormente, que pueden obtenerse a través del método **getComputedStyle(elem).getPropertyValue()** y así comprobar si cumplen con el patrón de estilos que definimos. En principio, en la búsqueda nos quedamos con los elementos que posean los atributos con los valores *z-index > 1* y *position = 'fixed';* el chequeo de estos valores fue suficiente para detectar con suficiente certeza los elementos que nos interesan. Continuando con la búsqueda de elementos flotantes que se muestran al iniciar la página web, el último paso en la detección, consiste en revisar si los elementos poseen los atributos correspondientes de Rol y ARIA para poder ser accesibles.

Durante esta búsqueda, se lograron capturar y analizar la mayor parte de los elementos flotantes de nuestro interés. Durante la investigación en distintos sitios webs, hemos encontrado que generalmente hay entre 1 y 2 elementos flotantes (*non-modal dialogs*). Incluso, se lograron capturar los casos donde hay *non-modal dialogs* que se muestran dinámicamente al pasar cierto periodo de tiempo dentro de la página web. Cabe destacar que estos elementos no son renderizados por causa de algún evento disparado por parte del usuario, sino que es el mismo sitio web el que automáticamente los muestra.

Para lograr ampliar nuestra detección a elementos flotantes que se despliegan en la página dinámicamente, se utilizó como guía de desarrollo el diseño de la herramienta de detección desarrollada por Watanabe et al. [Watanabe2016], en la cual se realiza una búsqueda previa de los elementos que están ocultos, para luego observar si éstos son mostrados al usuario. En concreto, la solución consiste en observar los elementos que contienen los atributos *display: none* y *visibility: hidden*, como podemos observar en el código de la Figura 3.

```
function obtenerElementosOcultos() {
    let divs = getBodyDiv();
    return [...divs].filter(elem => {
        // Attributos de ocultamiento
        let display = window.getComputedStyle(elem).getPropertyValue('display');
        let visibility = window.getComputedStyle(elem).getPropertyValue('visibility');
        return visibility === 'hidden' || display === 'none';
    });
}
```

Fig. 3: Código de la función JS con la cual se obtienen los elementos ocultos.

Al tener detectados los elementos a los cuales vamos a observar, al igual que en la herramienta de Watanabe et al. [Watanabe2016], se fijan los Mutation Observer a estos

elementos ocultos para lograr capturar si en algún momento pasan a ser visibles. Esta detección funciona dinámicamente; por ende, va a detectar aquellos elementos flotantes que cambien de estado oculto a desplegado en cualquier momento.

Cuando se detecta que un elemento estaba oculto y luego pasa a ser visible, se analiza si es un elemento de tipo flotante. Para eso se analizan sus atributos CSS, y si cumple, se busca entre sus atributos la ausencia de roles y atributos *ARIA*. Si no existe ninguno de estos atributos, se reporta el elemento a la *API REST* indicando que contiene el accessibility smell denominado *elemento web flotante inaccesible*.

Esta nueva solución dinámica permitió además detectar otros tipos de modales y elementos flotantes diferentes a los non-modal dialogs que pasan de estar ocultos a mostrarse debido a alguna acción disparadora que realiza el usuario.

3.4 Implementación de API REST y aplicación de reportes

Después de que cada finder finalice con su búsqueda y detección de elementos web inaccesibles, se envía la información de estas detecciones a la API Rest de la herramienta, y ésta se encarga de darle el formato correcto para almacenar la información en la base de datos. Dentro de la API Rest, hay un método que se encarga de realizar toda la lógica necesaria para lograr almacenar toda la información necesaria sobre los eventos de bad smells capturados por la extensión web, el método saveBadSmellEvent(). Dentro de este método se almacena el evento de bad smell en la colección BadSmellEvents. Sobre el evento se guarda las propiedades **BaseURI,** que indica la URL de la página web donde se originó el evento y **Data,** que contiene información sobre atributos del elemento web: contenido del elemento, tipo de elemento, código HTML del elemento, tipo de smell generado por el elemento, fecha y hora en la que se generó el evento.

Además, en este mismo método se registra el evento en otra colección llamada ReportedPages. El propósito de esta es tener almacenados los elementos web que provocaron eventos de accesibilidad en cada página web en donde se ejecutó la herramienta de una manera más organizada. Cada elemento de esta colección lo denominamos ReportedPage, y contiene la siguiente información: BaseURI (URL de la página web donde se ejecutó la web extensión), Reported_elements (contiene una colección de elementos que generaron smells de accesibilidad dentro de la página. De cada elemento almacenamos cierta información útil para lograr identificarlos y llevar un conteo de ocurrencias: tipo de elemento, código HTML del elemento, tipo de bad smell generado, cantidad de veces que se reportó este evento, fecha y hora en la que se generó el evento, fecha y hora en la que se actualizó por última vez).

El paso final es armar el reporte de los bad smells para cada sitio web en los cuales se ejecutó la extensión web, un ejemplo puede verse en la Figura 5 y 9. De esto la encargada es la aplicación web de reportes, la cual se comunica con la API Rest para que esta le brinde la información que se encuentra en la base de datos.

4. Experimentación y evaluación de la propuesta

4.1 Análisis de accesibilidad en elementos web interactivos con handlers JS asignados

Para mostrar el resultado final de la herramienta para la detección de elementos web inaccesibles con handlers JS, se ejecutó la misma en varios sitios web, un ejemplo de los resultados es el siguiente.

Página de principal del portal de APR

URL

www.apronline.gob.ar

Comportamiento de NVDA

En este caso *NVDA* no mencionó ninguno de los dos elementos que están señalados en la Figura 4. Por lo tanto, el usuario no puede utilizar el carrusel de trámites.

Fig. 4: Screenshot de los elementos para controlar el scroll del carrusel.

Como se puede observar en la Figura 5, se reportaron ambos elementos del carrusel. Los cuales son links <a> con handlers JS asignados y sin atributo HREF ni atributos para lograr accesibilidad.

Fig. 5: Reporte generado por ABF para la página principal de APR.

Ampliación de búsqueda para elementos no detectados

Luego de probar la herramienta en varios sitios web, se han logrado detectar la gran mayoría de elementos con handlers JS. Aun así, había algunos elementos que no podían ser detectados, y la causa fue que utilizaban handlers diferentes a los que eran buscados por el finder de búsqueda. Por ejemplo, en un principio el finder buscaba elementos con handlers “onClick”, pero esto no detectaba la mayoría ya que existen elementos que manejan el evento “click”. Por lo tanto, luego de añadir este evento a la búsqueda del finder, se logró ampliar aún más la cantidad de elementos detectados. Por ejemplo, en el sitio de autogestión de APR, hay elementos de tipo link <a> los cuales antes de incluir eventos “click” en el finder, no podían ser detectados y ahora lo son. Como se puede observar en la Figura 6, se pueden ver los elementos con handlers JS accesibles y los que

no lo son.

Fig. 6: Diagrama que muestra los elementos con handlers JS de la página de APR reportados y clasificados.

Eventos no detectados

Para seguir ampliando la capacidad de detección de este finder, se necesitó seguir añadiendo más tipos de eventos a la etapa de búsqueda del algoritmo. Para este trabajo nos enfocamos en elementos que reaccionan a eventos **onClick** y **click**, debemos destacar que para elementos de tipo link <a> no solo realizan su acción al recibir el click de mouse sino también al presionar la tecla *enter*. Sin embargo, al igual que los botones, esta es una característica de estos elementos. Para el caso de los y esto no es así, el usuario no los va a poder utilizar si no poseen una handler para recibir eventos de tecla *ENTER* (con handler *onclick* no es suficiente).

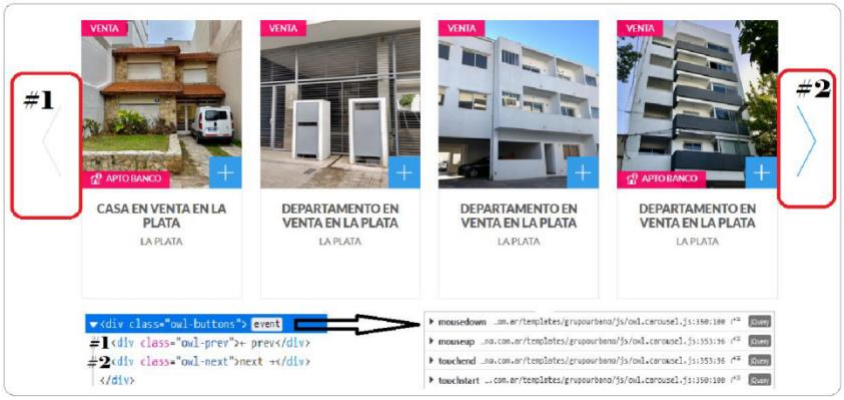

Fig. 7: Captura del carrusel de imágenes del sitio web de grupo-urbano.com.ar, junto al código HTML y los handlers asignados.

No obstante, hay casos donde se crean elementos interactivos que reaccionan a otro tipo de eventos, por ejemplo, como se puede ver en la Figura 7, los dos elementos para manipular el carrusel de imágenes están declarados con 's que tienen los siguientes handlers asignados: **mousedown**, **mouseup**, **touchend** y **touchstart**. Sería bueno para trabajos futuros incluir estos eventos a la búsqueda de elementos.

4.2 Análisis de accesibilidad en mensajes dinámicos de formulario web

Página de registro del portal de APR

URL

https://autogestion.apronline.gov.ar/registro

Comportamiento de NVDA

En este caso el lector de pantalla *NVDA* no mencionó ninguno de los errores que se muestran en pantalla. Simplemente hizo un salto al primer input con error.

La herramienta reporta los elementos que representan a un error en el formulario y que no son accesibles. En este ejemplo, los elementos de la Figura 8 que contienen los mensajes de error contienen el atributo for, pero están declarados como <*span*> y no como <*label*>. El atributo *for* solo funciona en elementos *label* y *output*, es decir en este caso no permite que el elemento sea accesible para el lector de pantalla. Estos elementos fueron detectados y reportados por la herramienta como se observa en la Figura 9, por lo tanto, podemos concluir que en este caso la detección funcionó correctamente.

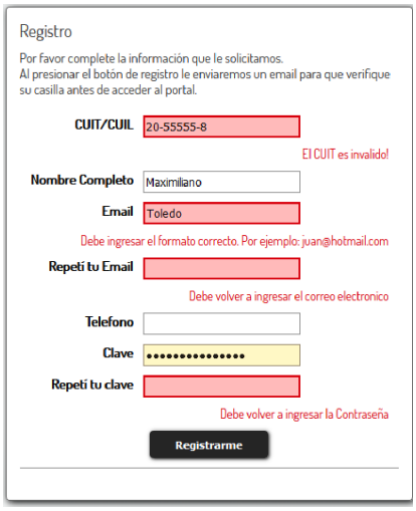

Fig. 8: Formulario web del sitio de APR con mensajes de error desplegados.

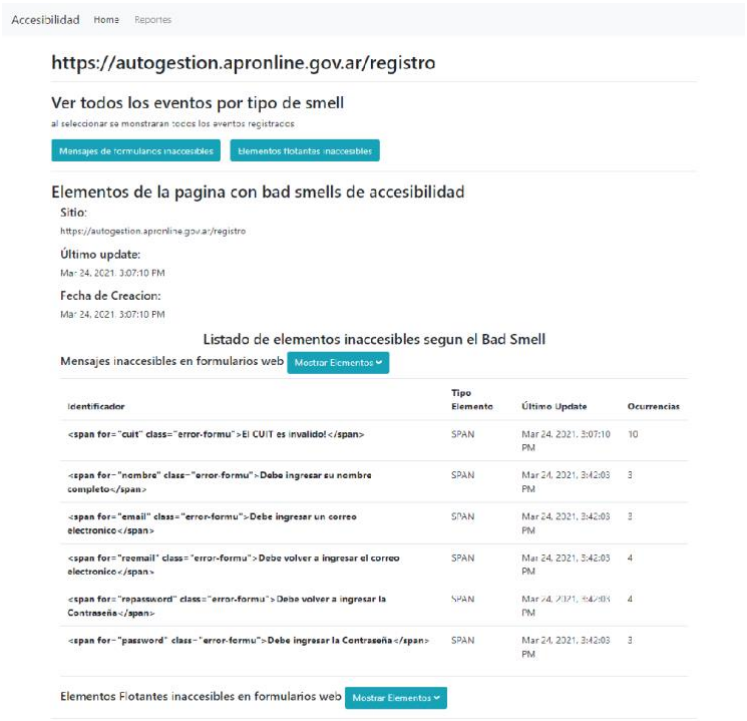

Fig. 9: Reporte generado por la aplicación de reportes para la página de registro de APR.

4.3 Análisis de accesibilidad en elementos web flotantes

Página de inicio de APR Online

URL

http://www.apronline.gov.ar/

Comportamiento de NVDA

Continuando con el caso de la página de APR Online encontramos que al ingresar a la página web aparece un modal desplegado. Este modal cubre el contenido de la página con una capa oscura (ver Figura 10). De una manera contraintuitiva, mientras el modal esté desplegado, el usuario puede seguir interactuando con los demás elementos de la página. Más allá de ello, el usuario no es notificado sobre el modal desplegado como debiera suceder.

Fig. 10: Modal al inicio de la página web de APR Online.

Elementos Flotantes web inaccesibles

Mostrar Elementos

Identificador	Tipo Elemento	Último Update	Ocurrencias	Acciones
<a href="?modulo=pf_res..	DIV	Apr 28, 2021, 7:43:50 PM	4	Ver Más info
<a data-toggle="modal" da..	DIV	Apr 28, 2021, 7:43:50 PM	4	Ver Más info
<div class="hide modal fade show" id="events-modal-popup" style="paddi..	DIV	Apr 28, 2021, 7:43:50 PM	3	Ver Más info
	DIV	Apr 28, 2021, 7:43:50 PM	3	Ver Más info

Fig. 11: Resultado final de la detección de elementos flotantes con problemas de accesibilidad.

Analizando el resultado de la herramienta mostrado en la Figura 11, nos detendremos en los últimos dos elementos reportados:

a) class="hide modal fade show" id="events-modal-popup"

b) class="modal-backdrop fade show"

Ambos están relacionados con el modal de la Figura 10; el modal en sí que contiene el contenido es el elemento reportado a), y está correctamente reportado ya que no posee ningún atributo o rol ARIA para ser accesible. El elemento b) también carece de estos atributos, pero no es correcto reportarlo ya que es la sombra del modal. Fue reportado por la herramienta ya que posee las mismas características de un elemento flotante. Este fue el único caso en estos tipos de elemento donde ABF reporta un falso positivo. En futuras versiones se planea ignorar este tipo de elementos durante la detección de elementos flotantes.

5. Conclusiones y trabajo futuro

La accesibilidad web no es algo sencillo de lograr si la misma no está incluida en el proceso de desarrollo. Por otro lado los desarrolladores no suelen tener en cuenta a la accesibilidad durante el desarrollo de una aplicación web, y tampoco poseen capacitación

en el tema. En general, recién se comienza a dar importancia a la accesibilidad web luego de que el costo de no incluirla en el desarrollo comienza a aumentar. Para ayudar a los desarrolladores a lograr proveer sitios webs accesibles, actualmente existen varias herramientas online y gratuitas para evaluar un sitio en busca de problemas de accesibilidad. Sin embargo, están limitadas a solo poder encontrar aquellos problemas de accesibilidad que pueden ser detectados a través del análisis estático de código.

En este trabajo se buscó brindar a los desarrolladores y dueños de sitios web una herramienta que facilite la forma en que pueden encontrar diferentes tipos de elementos webs que generen *accessibility smells*. El fin de esta herramienta no es solo la de detectar los elementos problemáticos, sino además almacenar su información para luego generar reportes de accesibilidad que sean de fácil lectura. Una de las principales ventajas de la herramienta ABF es que su código es público, así que puede ser extendido sin limitaciones para lograr detectar más casos de elementos web inaccesibles. Otra ventaja importante es que los *finders* definidos en la extensión web están desarrollados en JavaScript, lo cual debería facilitar la incorporación de nuevos finders de otros *accessibility smell* a detectar. Además, la experimentación ha demostrado que ABF tiene un buen rendimiento [Toledo2021].

Como trabajos futuros se plantean ampliar la búsqueda utilizando nuevos tipos de eventos para el finder *análisis de accesibilidad en elementos web interactivos con handlers JS asignados.* En el caso del finder *análisis de accesibilidad en mensajes dinámicos de formulario web*, hubo algunos casos de mensajes inaccesibles que no fueron detectados debido a que estos elementos son generados fuera del formulario, y la herramienta está desarrollada para detectar solo cambios del DOM dentro del formulario. Como trabajo futuro buscaremos integrar a la herramienta existente, la lógica para poder capturar estos mensajes por fuera del formulario. En el finder *análisis de accesibilidad en elementos web flotantes*, el siguiente desafío consta en continuar mejorando el filtrado de aquellos elementos flotantes que no son deseados en las detecciones, por ende, no es de nuestro interés analizarlos en búsqueda de problemas de accesibilidad.

Además, se desea continuar con la mejora de los finders de la herramienta definidos en este trabajo y continuar con el desarrollo de nuevos finders que ataquen a otras problemáticas de accesibilidad web y diferentes elementos web. También se propone agregar dentro de los reportes, la zona específica en el código de la aplicación web donde se generan los bad smells detectados. Por otro lado también buscaremos incluir la aplicación de *refactorings* de código a los elementos detectados y reportados por la herramienta cuando sea posible, y así lograr aplicar una solución a estos problemas. Asimismo buscaremos avanzar en escalar la herramienta a un ambiente de producción. Esto permitirá que varios usuarios reporten a una misma base de datos, y así poder generar reportes de accesibilidad con datos generados por múltiples usuarios. Además, se va a continuar con la realización de experimentos con usuarios no videntes para testear la funcionalidad de la herramienta y así analizar sus resultados para continuar con la mejora continua de la herramienta.

Referencias

[Antonelli2019] Antonelli, H. L., Sensiate, L., Watanabe, W. M., & de Mattos Fortes, R. P. (2019). Challenges of automatically evaluating Rich Internet Applications accessibility. Proc. of the 37th ACM Int. Conf. Design of Communication, pp. 1-6.

[Buono2020] Buono, P., Caivano, D., Costabile, M., Desolda, G., Lanzilotti, R. (2020). Towards the Detection of UX Smells: The Support of Visualizations, in IEEE Access, vol. 8, pp. 6901-6914.

[Carvaja2020] Carvajal, Carlos Máñez. (2020). Evaluación de accesibilidad web de las universidades chilenas. Formación universitaria, 13(5), pp. 69-76.

[Carvalho2016] Pedroso Carvalho, L. , Pereira Ferreira, L., Pimenta Freire, A. (2016). Accessibility evaluation of rich internet applications interface components for mobile screen readers. In Proc. 31st ACM Symp. on Applied Computing. pp. 181–186.

[Durgam2020] Durgam, F. (2020). Detección de problemas de accesibilidad en la utilización de lectores de pantalla en aplicaciones web. Tesis de Maestría, Fac. de Informática, Univ. Nacional de La Plata.

[Fernandes2013] Fernandes, N., Batista, A. S., Costa, D., Duarte, C., Carriço, L. (2013). Three web accessibility evaluation perspectives for RIA. In Proc. 10th Int. cross-disciplinary conference on web accessibility, pp. 1-9.

[Garrido2011] Garrido, A., Rossi, G., Distante, D. (2011) Refactoring for usability in web applications. IEEE Software 28(3), pp. 60–67.

[Garrido2014] Garrido, A., Rossi, G., Medina-Medina, N., Grigera, J., Firmenich, S. (2014) Improving Accessibility of Web interfaces: refactoring to the rescue. Univ. Access in the Information Society 13, pp. 387-399.

[Grigera2017] Grigera, J.; Garrido, A.; Rivero, J. M.; Rossi, G. (2017) Automatic detection of usability smells in web applications. International Journal of Human-Computer Studies 97, pp. 129-148.

[TAW] Taw. Test de accesibilidad web. https://www.tawdis.net/.

[Toledo2021] Toledo, M. (2021). Detección automática de problemas de accesibilidad a partir de eventos de interacción de usuario. Tesis de Licenciatura, Fac. de Informática, Univ. Nacional de La Plata.

[W3C2005] W3C. 2005 Web Accessibility Initiative (WAI). Accessibility Introduction. https://www.w3.org/WAI/fundamentals/accessibility-intro/

[WAI-ARIA] WAI ARIA, the Accessible Rich Inter net Applications Suite. https://www.w3.org/WAI/standards guidelines/aria/.

[WCAG] WCAG 2.1 2018. Web Content Accessibility Guidelines. https://www.w3.org/TR/WCAG21/.

[Watanabe2016] Watanabe, W. M., & de Mattos Fortes, R. P. (2016, April). Automatic identification of drop-down menu widgets using mutation observers and visibility changes. In Proc. 31st Annual ACM Symp. on Applied Computing (pp. 766-771).