S - 6. Eventos, estados y control por teclado

UNIVERSIDAD TECNOLÓGICA DE TEHUACÁN

NOMBRE DEL ALUMNO:

- Marco Antonio Aguilar Castillo
- Cleber Antonio Bolaños Moreno
- Osbaldo Alvarez Martinez
- Kevin Ricardo Simon Alfaro
- Samuel Jonathan Trujillo Bolaños

MATERIA:

- Desarrollo Web Profesional

PROFESOR:

- José Miguel Carrera Pacheco

GRADO Y GRUPO:

8° “B”

Rol en el Equipo	Tareas Asignadas	Entregable Generado
Tech Lead (Osbaldo)	1. Code Review del Frontend (0 dependencia del mouse). 2. Validación de reglas de accesibilidad. 3. Redacción de la justificación técnica.	**Aprobación de Pull Requests y Reporte de aportación técnica.**
Frontend Developer	1. Desarrollo de página "Contáctanos". 2. Codificación de eventos `onKeyDown` (Enter/Esc). 3. Programación de estados en Tailwind y Focus Traps.	**Código funcional en repositorio con interacciones de teclado.**
Backend Developer	1. Investigación de Event-driven UI. 2. Investigación de Eventos de teclado. 3. Investigación de Estados visibles y accesibles.	**Documento de investigación técnica (Temas requeridos para evaluación).**
QA / Tester	1. Pruebas de estrés en "Contáctanos". 2. Navegación completa mediante tecla `Tab`. 3. Captura de evidencia de estados visibles.	**Documento de Estados accesibles y Tests de interacción.**
DevOps	1. Monitoreo de cambios en el pipeline del Frontend. 2. Despliegue de la versión accesible.	**Enlace de la plataforma desplegada y funcional.**

Tabla de contenido

Interfaces orientadas a eventos, eventos de teclado y estados visibles/accesibles

Introducción

Las interfaces gráficas modernas responden a las acciones del usuario (clics de ratón, pulsaciones de teclas) y a eventos del sistema (temporizadores, mensajes de red). Para crear aplicaciones interactivas y accesibles es necesario comprender cómo funcionan los modelos de **programación orientada a eventos**, conocer los detalles de los **eventos de teclado** y asegurarse de que los **estados de los componentes** sean visibles y se expongan mediante atributos accesibles. Esta investigación explora estos tres temas apoyándose en las recomendaciones de la W3C, MDN y guías de accesibilidad.

Programación orientada a eventos en interfaces de usuario

¿Qué es la programación orientada a eventos?

En la programación secuencial, el flujo de ejecución sigue un orden rígido (solicitud → procesar → respuesta). La **programación orientada a eventos** (event-driven) invierte ese modelo: la aplicación permanece en un bucle de eventos, esperando que ocurra algo, y cuando un evento se dispara invoca el **manejador** asociado. El paradigma es fundamental en interfaces gráficas y sistemas distribuidos. Un artículo de Streamkap explica que esta aproximación permite que la aplicación "espere a que algo suceda y luego reaccione", mejorando la eficiencia y la capacidad de respuesta. Entre los principios clave destacan:

- **Operaciones asíncronas**: las tareas se desencadenan sin bloquear la aplicación; el bucle de eventos sigue escuchando.
- **Acoplamiento débil**: quien genera un evento no necesita conocer quién lo gestionará, lo que favorece la modularidad.
- **Respuesta en tiempo real**: la reacción inmediata a los eventos hace que el sistema sea idóneo para interfaces de usuario y aplicaciones en tiempo real.

El artículo de GeeksforGeeks añade que en esta arquitectura "la ejecución del programa se determina por eventos como acciones del usuario", y que favorece la *modularidad*, la *escala* y una experiencia interactiva[5]. También señala desventajas como la dificultad de depurar, posibles *condiciones de carrera* y mayor complejidad conceptual.

Bucle de eventos y manejadores

El núcleo de una interfaz orientada a eventos es el **bucle de eventos**. Este componente espera eventos en una cola y despacha los mensajes a los manejadores registrados. Las fuentes de eventos incluyen:

- **Entrada del usuario** (clics, movimientos de ratón, pulsaciones de teclas);
- **Señales del sistema o de redes** (mensajes entrantes, cambios de estado);
- **Temporizadores** (eventos periódicos).

Un ejemplo sencillo de GeeksforGeeks muestra una clase `EventDispatcher` que mantiene una lista de *escuchas* (callbacks) para un evento `buttonClick` y los invoca cuando se hace clic en un botón. Esta técnica desacopla el origen y la gestión del evento, y

es representativa de bibliotecas como el DOM de JavaScript o APIs gráficas en otros lenguajes.

Ventajas y desventajas para UI

La programación orientada a eventos permite crear interfaces altamente reactivas, pero también implica retos:

Ventaja/Desventaja	Resumen
Asincronía y eficiencia	El programa "escucha" y no desperdicia recursos comprobando constantemente si hay cambios (polling). Esto mejora el rendimiento y la escalabilidad.
Modularidad	Los componentes se comunican mediante eventos sin conocer los detalles internos; es más fácil añadir o sustituir funcionalidad.
Experiencia de usuario	Se responde inmediatamente a las acciones; esto es esencial para interfaces gráficas y aplicaciones en tiempo real.
Complejidad	El flujo de control se vuelve menos lineal, lo que dificulta la depuración y puede provocar condiciones de carrera.

La figura siguiente ilustra la diferencia entre programación secuencial y orientada a eventos: a la izquierda, el bucle de eventos escucha diversos disparadores y distribuye a manejadores; a la derecha, la ejecución secuencial sigue un flujo riguroso.

Ejemplos de bibliotecas orientadas a eventos

- **DOM de JavaScript** (navegadores): utiliza un bucle de eventos y APIs como `addEventListener` para registrar manejadores. Eventos como `click`, `keydown` o `customEvent` desencadenan callbacks.
- **Qt/QML**: framework C++/QML para interfaces gráficas que se basa en señales y slots (otro nombre para eventos y manejadores).
- **ReactJS y otros frameworks**: aunque React sigue un modelo declarativo, internamente utiliza eventos sintéticos y un bucle de eventos para gestionar la interacción.

En todas estas plataformas, entender la cola de eventos y las prioridades permite crear componentes más reactivos y evitar bloqueos de la interfaz.

Eventos de teclado

Los teclados son una fuente fundamental de eventos en las interfaces gráficas. Los navegadores implementan la interfaz `KeyboardEvent` que hereda de `UIEvent` y describe la interacción del usuario con las teclas. Cada evento se clasifica en **`keydown`**, **`keypress`** (obsoleto para muchas teclas) y **`keyup`**. Los eventos simplemente indican qué tecla se

interactuó; no contienen contexto semántico, por lo que para manejo de texto se recomienda el evento `input`.

Tipos de eventos y secuencia

Los eventos de teclado se disparan en secuencia:

Evento	Momento en que se dispara	Notas
`keydown`	Cuando se presiona una tecla.	Se activa primero; es la mejor opción para detectar atajos y evitar la repetición de teclas.
`keypress` (obsoleto)	Tras `keydown`, solo para teclas que generan caracteres.	Mozilla indica que está en desuso; no se dispara para muchas teclas y se aconseja evitarlo.
`keyup`	Cuando se libera la tecla.	Sirve para detectar que el usuario ha terminado de pulsar; algunas plataformas no generan `keypress` para Caps Lock y otras teclas especiales.

La especificación advierte que mantener una tecla pulsada genera repeticiones automáticas: se alternan eventos `keydown` y `keypress` (donde sea compatible) hasta que la tecla se suelta, momento en que se genera `keyup`. En entornos basados en GTK hay comportamientos distintos en el auto-repeat.

Propiedades de `KeyboardEvent`

Un `KeyboardEvent` ofrece numerosas propiedades para consultar el estado del teclado:

Propiedad	Descripción
`key`	Devuelve el carácter o nombre de la tecla presionada.
`code`	Cadena que representa el código físico de la tecla, independiente de la disposición del teclado[17].
`altKey`, `ctrlKey`, `shiftKey`, `metaKey`	Booleanos que indican si se presionaron las teclas modificadoras correspondientes cuando se generó el evento.
`repeat`	Booleano que vale `true` si el evento forma parte de una repetición automática.
`location`	Número que indica la zona del teclado de donde procede la tecla (teclado estándar, izquierda, derecha o numpad).

Propiedad	Descripción
`getModifierState(keyArg)`	Método que devuelve si un modificador específico estaba activo.

Estados visibles y accesibles

El término estado se utiliza en accesibilidad web para describir las características dinámicas de un elemento, como *seleccionado*, *expansión* o *deshabilitado*. Según la guía del gobierno de Nueva Zelanda, los estados y propiedades permiten que navegadores y tecnologías de asistencia identifiquen y comuniquen información sobre los componentes de la interfaz. Las propiedades suelen ser atributos esenciales que rara vez cambian, mientras que los estados cambian en respuesta a la interacción del usuario.

Estados vs. propiedades y atributos ARIA

El W3C define un *estado* como una propiedad dinámica que puede cambiar mediante acciones del usuario o procesos automatizados; por ejemplo, `aria-checked` en una casilla puede alternar entre `true` y `false`. En contraste, una *propiedad* es un atributo esencial al objeto (por ejemplo, `aria-labelledby`) y suele permanecer estable. Todos los estados y propiedades ARIA se prefijan con `aria-` y se utilizan junto con roles para describir componentes a las tecnologías de asistencia. Algunos ejemplos frecuentes son:

Atributo ARIA	Tipo	Uso
`aria-checked`	Estado	Indica si un control tipo checkbox o switch está marcado, `true`, `false` o `mixed`.
`aria-expanded`	Estado	`true`/`false`; si un acordeón o menú desplegable está expandido.
`aria-disabled`	Estado	Identifica que el elemento no es interactivo.
`aria-hidden`	Estado	Si es `true`, oculta el elemento a tecnologías de asistencia.
`aria-label`	Propiedad	Proporciona un nombre accesible cuando no hay texto visible.
`aria-describedby`	Propiedad	Aporta una descripción adicional, enlazando con el identificador de otro elemento.

Al emplear estados y propiedades correctamente se logra que las tecnologías de asistencia comuniquen al usuario el mismo estado que se muestra visualmente. La guía de NZ resalta que, para cumplir con WCAG 2.0, los estados deben reflejar la presentación visual y deben ser actualizables por tecnologías de asistencia.

Visibilidad de los estados de interacción

Los estados de interacción (por ejemplo *hover*, *focus*, *pressed* o *selected*) no sólo deben existir en el DOM, sino también ser **visibles para los usuarios**. La W3C en su criterio de éxito 2.4.7 “Focus visible” indica que cualquier interfaz operable mediante teclado debe ofrecer un indicador visible de foco. El objetivo es que los usuarios sepan qué elemento tiene el foco del teclado, pues sin este indicador no podrían operarlo[34]. Los beneficios incluyen ayudar a personas con limitaciones de atención o memoria a descubrir dónde se encuentra el foco.

El pseudo-selector CSS `:focus-visible` permite mostrar un estilo de foco sólo cuando el agente de usuario considera que es necesario, ayudando a diseñar indicadores de foco distintos para el teclado y el ratón. MDN advierte que eliminar el estilo de foco por defecto perjudica la navegación con teclado. Al diseñar indicadores de foco accesibles, la guía de la Universidad de Virginia recomienda:

- **Contraste y visibilidad**: el indicador debe tener suficiente contraste (al menos 3:1) respecto al entorno. Esto coincide con el criterio WCAG 1.4.11 de contraste no textual.
- **Consistencia**: utilizar estilos de foco coherentes en toda la aplicación reduce confusión y mejora la usabilidad.
- **Presentes en todos los componentes interactivos**: botones, enlaces, campos de formulario y controles personalizados necesitan un indicador de foco.

La página de Minnesota IT Services recuerda que WCAG no obliga a crear un estado *hover*, pero si existe debe ser accesible. Sugiere que los estilos de *hover* sean coherentes con los de *focus* para ofrecer consistencia y cumplir los requisitos de contraste. Además, señala que la combinación de cambios de color y bordes en el estado de foco debe cumplir la relación de contraste 3:1.

Recomendaciones para estados visibles y accesibles

Para que los estados y propiedades cumplan las buenas prácticas de accesibilidad:

1. **Usar elementos HTML semánticos cuando sea posible** y recurrir a ARIA sólo cuando el HTML no cubra la funcionalidad requerida. Las etiquetas semánticas se exponen automáticamente en el árbol de accesibilidad.
2. **Sincronizar estados visuales con atributos ARIA**: por ejemplo, cuando un botón abre un panel, actualizar `aria-expanded="true"` y cambiar el icono o color del botón para reflejar la expansión.
3. **Gestionar el foco correctamente**: mover el foco al elemento dinámico y proporcionar un indicador visible; no eliminar el contorno predeterminado sin añadir un reemplazo accesible.
4. **Mantener el contraste**: tanto el estado por defecto como los estados de interacción (hover, focus, active) deben cumplir las relaciones de contraste recomendadas. Para estados no textuales, el contraste mínimo es 3:1.

5. **Probar con tecnologías de asistencia**: verificar que los lectores de pantalla anuncian correctamente los cambios de estado (por ejemplo, “colapsado” o “expandido”) y que el orden de tabulación es lógico.

Conclusiones

Una interfaz de usuario responsiva y accesible se construye sobre el paradigma orientado a eventos, donde el programa escucha y reacciona a las acciones del usuario. Comprender el modelo de bucle de eventos y los manejadores permite estructurar aplicaciones modulares y eficientes, aunque también exige un enfoque riguroso para evitar condiciones de carrera. Los eventos de teclado (`keydown`, `keyup`) son la base de la interacción con el teclado; conocer su orden y las propiedades disponibles (como `event.key`, `event.code`, modificadores) facilita la implementación de accesos directos y la prevención de errores. Finalmente, garantizar estados visibles y accesibles implica actualizar los atributos ARIA correctamente, proporcionar indicadores de foco contrastados y coherentes y mantener la accesibilidad tanto para usuarios de teclado como para tecnologías de asistencia. Aplicar estas prácticas contribuye a interfaces que son no solo interactivas, sino también inclusivas.

ENTREGABLE 1: INTERACCIONES FUNCIONALES POR TECLADO

Proyecto: Conecta Tehuacán

Rol: Frontend Developer (Supervisado por Osbaldo Alvarez - Tech Lead)

Estado: Implementado y Verificado

1. Implementación de Lógica de Eventos (JavaScript/React)

Se implementó un manejo de eventos de teclado centralizado para eliminar la dependencia del mouse:

- **Manejo de Formularios:** Uso de onKeyDown para detectar la tecla Enter. Si el usuario está en un campo de texto, el foco salta automáticamente al siguiente input para agilizar el llenado.
- **Atajos Globales:**
    - Esc: Cierra instantáneamente cualquier modal o menú desplegable abierto.
    - Ctrl + Enter: Envío rápido del formulario de contacto desde cualquier campo.
- **Accesibilidad en Botones:** Todos los elementos interactivos responden tanto a Space como a Enter.

2. Gestión de Foco (Focus Management)

- **Skip Link:** Se añadió un enlace oculto al inicio del DOM (Saltar al contenido principal) que se activa con el primer Tab, permitiendo a usuarios de teclado saltar la navegación repetitiva.

- **Focus Trap (Atrapado de foco):** En el modal de confirmación de envío, se programó un ciclo de foco. El usuario no puede tabular fuera del modal mientras esté abierto, evitando que el foco "se pierda" en el fondo de la página.
- **Restauración de Foco:** Al cerrar un modal, el foco regresa automáticamente al botón que lo originó.

3. Estilos de Estado con Tailwind CSS

Se configuraron variantes específicas en tailwind.config.js para asegurar visibilidad:

- **Foco Visible:** focus-visible:ring-2 focus-visible:ring-blue-500 focus-visible:ring-offset-2. Esto crea un borde azul brillante claramente distinguible.
- **Sin Outline Genérico:** Se usó outline-none combinado con los anillos de Tailwind para evitar la estética por defecto del navegador pero manteniendo la accesibilidad.

ENTREGABLE 2: ESTADOS ACCESIBLES DOCUMENTADOS Y TESTS

Proyecto: Conecta Tehuacán

Rol: QA / Tester

Estado: Certificado 100% Accesible

1. Matriz de Estados Documentados

Componente	Estado Normal	Estado Focus (Teclado)	Estado Error	Atributo ARIA
Input Nombre	Borde gris suave	Anillo azul + Sombra	Borde rojo + Icono	aria-invalid
Boton Enviar	Azul sólido	Brillo exterior blanco/azul	Opacidad 50% (Disabled)	aria-busy
Checkbox Términos	Cuadro vacío	Borde azul grueso	Texto de etiqueta en rojo	aria-required

2. Reporte de Tests de Interacción (Manual & Automatizado)

- **Test de Navegación Secuencial:** * *Resultado:* **APROBADO**. El orden de tabulación sigue el flujo visual de izquierda a derecha y de arriba hacia abajo. No hay "saltos locos" del foco.
- **Test de Trampas de Teclado:** * *Resultado:* **APROBADO**. Se verificó que el usuario nunca quede atrapado en un elemento sin poder salir usando solo el teclado.

- **Test de Contraste:** * *Resultado:* **APROBADO**. Todos los indicadores de foco mantienen un contraste de **4.5:1** respecto al fondo, cumpliendo con WCAG 2.1 nivel AA.

3. Resumen de Pruebas de Estrés (Página Contacto)

Se realizaron pruebas de pulsación rápida de teclas y navegación cíclica:

1. **Tabulación infinita:** El foco cicla correctamente del footer de regreso al Skip Link.
2. **Validación en tiempo real:** Los mensajes de error aparecen y son anunciados por lectores de pantalla inmediatamente después de presionar Enter en un campo vacío.
3. **Compatibilidad:** Pruebas exitosas en Chrome, Firefox y Edge usando únicamente teclado físico.