Routers

Sistemas y Tecnologías Web (CC3062) - 2026

Routers

Dónde vivimos actualmente

- En la última clase se creó una aplicación empaquetada con Vite.
- La aplicación tiene dos vistas principales: un índice (lista) y un detalle (show).
- Se logró cambiar entre ellas inyectando contenido directamente al DOM.
- Se usa JavaScript puro para ocultar una sección y mostrar otra al seleccionarla.

El Problema del DOM Manual

- Aunque la pantalla cambia, la URL en el navegador sigue siendo la misma.
- Si el usuario encuentra un elemento interesante y recarga la página, vuelve al inicio.
- Se rompió la funcionalidad nativa del navegador.

Impacto Real

- UX (Experiencia de Usuario): El botón "Atrás" del navegador no funciona. Si el usuario lo presiona, sale de la aplicación.

- Compartir: No se puede enviar a un amigo el enlace de un elemento específico.

- SEO: Los motores de búsqueda (como Google) solo ven una única página vacía inicial. No indexan el contenido.

Router

- Es el puente entre lo que el usuario ve y la URL.
- Es una pieza de código que "escucha" la URL.
- Dependiendo de la URL actual, decide qué parte del DOM debe actualizarse.

Single Page Application (SPA)

- La aplicación de Vite es una SPA.
- Significa que el servidor web solo entrega un único archivo HTML (`index.html`).
- Toda la navegación posterior es una "ilusión" creada por JavaScript.
- El Router es el director de esa ilusión.

History API

- Herramienta propuesta por el navegador.
- History API permite a JavaScript interactuar con el historial del navegador.

- Cambiar la URL en la barra superior sin obligar al navegador a descargar la página de nuevo.

Ciclo de Navegación SPA

El proceso de navegación en una SPA se puede desglosar en un ciclo de vida de tres etapas críticas que permiten la transición de estados sin recargar el documento.

Paso 1: Captura y Delegación de Eventos

En lugar de asignar un listener a cada enlace (ineficiente), se utiliza la Delegación de Eventos. Se escucha el click en el nivel más alto y se filtra el origen.

- Intercepción: Se identifica si el elemento tiene el atributo `data-link` para evitar el comportamiento por defecto de los enlaces tradicionales.

- Prevención: `e.preventDefault()` detiene la petición HTTP que el navegador intentaría realizar al servidor.

```
document.addEventListener('click', e => {
  // Se verifica si el elemento clickeado
  // o su ancestro es un enlace de la aplicación
  if (e.target.matches('[data-link]')) {
    e.preventDefault();
    const url = e.target.href;
    navigateTo(url);
  }
});
```

Paso 2: Manipulación del Historial (pushState)

Una vez capturada la intención del usuario, se actualiza la barra de direcciones. Aquí es donde la History API entra en juego para mantener la "ilusión" de navegación.

- Sincronización: `history.pushState(state, title, url)` añade una nueva entrada al historial del navegador.

- Persistencia: Al cambiar la URL manualmente, se garantiza que si el usuario copia el enlace, este apunte al estado actual de la aplicación.

```
const navigateTo = url => {
  // Se actualiza la URL sin recargar la página
  window.history.pushState(null, null, url);

  // Se invoca manualmente al router
  // para que reaccione al nuevo estado
  router();
};
```

Paso 3: Resolución de Rutas y Renderizado

El Router actúa como un controlador que decide qué vista debe mostrarse basándose exclusivamente en el `pathname` actual de la URL.

- Diccionario de Rutas: Se define un objeto que mapea rutas de texto a funciones de renderizado.

- Manejo de Errores: Es fundamental contar con una ruta por defecto o "404" para manejar direcciones no registradas.

```
const routes = {
  '/': renderIndex,
```

```
  '/show': renderDetail,
  '/contact': renderContact
};

const router = () => {
  // Se obtiene la ruta actual
  const path = window.location.pathname;

  // Se busca la función correspondiente o error
  const viewFunc = routes[path] || renderNotFound;

  viewFunc();
};
```