Tema 8

Punto 4 Animación

Las animaciones en CSS permiten que elementos como texto e imágenes aparezcan con movimiento en la pantalla. Un ejemplo común es hacer que un párrafo de texto se desplace de un extremo a otro.

Pasos para crear la animación

1. Se define un selector para `<p>` en CSS, estableciendo la duración de la animación (`5s`) y su repetición infinita.
2. Se crea una animación con `@keyframes`, usando el mismo nombre que en `animation-name`.
3. Se definen los valores de la animación en distintos intervalos: inicio (0%), dos puntos intermedios (30% y 70%) y el final (100%).

Ejemplo de código

```
p {
  animation-duration: 5s;
  animation-iteration-count: infinite;
  animation-name: animaTexto;
}

@keyframes animaTexto {
  from {
    margin-right: 90%;
    width: 100%;
  }
  30% {
    font-size: 300%;
    margin-right: 25%;
    width: 150%;
  }
  70% {
    font-size: 600%;
    margin-right: 50%;
    width: 120%;
  }
  to {
    margin-right: 0%;
    width: 100%;
  }
}
```

Este código hace que el texto cambie de posición y tamaño en diferentes momentos de la animación, logrando un efecto dinámico y repetitivo.

Punto 6 Transiciones

Las **transiciones** permiten cambios graduales en los valores de propiedades de los elementos, activados por una acción, como pasar el cursor sobre una imagen.

Elementos clave de una transición

- **Propiedad**: Indica qué característica cambiará (ej. `background-color`).
- **Duración**: Especifica el tiempo de la transición (en segundos).
- **Retardo**: Define el tiempo antes de que inicie la transición (en segundos).
- **Tipo de acción**: Controla la forma en que se realiza la transición (ej. `ease`, `linear`, `ease-in-out`).

Ejemplo de transición en CSS

```
button {
  background-color: blue;
  color: white;
  transition: background-color 0.5s ease-in-out;
}

button:hover {
  background-color: red;
}
```

En este caso, al pasar el cursor sobre el botón (`hover`), el color de fondo cambia de azul a rojo de manera progresiva en 0.5 segundos.

Las propiedades que pueden animarse con transiciones incluyen `background-color`, `color`, `width`, `height`, `opacity`, `margin`, `border`, `top`, `left`, `right`, entre otras.

7. Transformaciones en CSS

- Permiten aplicar efectos visuales como rotaciones, desplazamientos y escalados.
- **Propiedad `transform`**: Define la transformación tomando como origen el eje central del elemento.
- **Propiedad `transform-origin`**: Permite cambiar el punto de origen de la transformación.
- Tipos de transformaciones:
    - `scale`: Modifica el tamaño del elemento.
    - `skew`: Distorsiona el elemento.
    - `translate`: Desplaza el elemento en los ejes X e Y.

    - `rotate`: Rota el elemento en grados.
- Puede combinarse con `transition` para animaciones más fluidas.

8. Steps en animaciones CSS

- Permite dividir la animación en una cantidad específica de frames o pasos.
- Se usa con la función `steps(número de frames)`.

Ejemplo:

```
 section .caja {
  background: #95A5A6;
  transition: background 2s steps(3);
}
section .caja:hover {
  background: #2980B9;
}
```

-

9. Interactividad con jQuery (`animate`)

- jQuery permite agregar comportamiento interactivo a los elementos HTML.
- Se carga mediante `<script src="jquery.js"></script>`.
- Métodos de selección de elementos en jQuery:
    - `$("#ID")` → Selecciona por identificador.
    - `$("p")` → Selecciona elementos por etiqueta.
    - `$(".clase")` → Selecciona por clase.
    - `$("[atributo]")` → Selecciona por atributo.

Función `animate()`:

```
 $(elemento).animate({
  propiedades
}, [duración, easing, callback]);
```

-
    - **Duración**: Milisegundos o valores como `'slow'`, `'fast'`.
    - **Easing**: Define la velocidad de la animación (`'linear'` para constante).
    - **Callback**: Función ejecutada al finalizar la animación.

10. Efectos en jQuery

- Se pueden aplicar efectos visuales a elementos HTML con jQuery.
- Funciones principales:
    - `.show()`: Muestra el elemento.
    - `.hide()`: Oculta el elemento.
    - `.slideDown()`: Muestra el elemento con desplazamiento vertical.
    - `.slideUp()`: Oculta el elemento con desplazamiento vertical.

    - `.fadeIn()`: Modifica la opacidad al 100%.
    - `.fadeOut()`: Modifica la opacidad al 0%.

Ejemplo de uso:
```
 $('.nombreElemento').fadeOut(500, function(){
  // acciones después del efecto
});
```

-

Tema 9

2. Comportamiento interactivo con jQuery

- jQuery es una librería de JavaScript que facilita la manipulación del DOM y CSS.
- Su eslogan es **“Write less, do more”**, ya que simplifica tareas comunes.
- **Formas de incluir jQuery en un proyecto:**
    1. **Acceso local:** Descargar la librería y enlazarla en el HTML.
    2. **Acceso desde CDN:** Usar un enlace a un repositorio en línea (requiere conexión a Internet).

Ejemplo de inclusión en HTML:
```
 <script src="jquery.js"></script> <!-- Acceso local -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script> <!-- CDN -->
```

-

2.1. Revisión del DOM y ejecución del código jQuery

- **El DOM (Document Object Model)** permite acceder y modificar elementos HTML como nodos.
- Tipos de nodos en el DOM:
    - **Document:** Nodo raíz.
    - **Element:** Representa etiquetas HTML.
    - **Attr:** Representa atributos de elementos.
    - **Text:** Contenido dentro de etiquetas.
    - **Comment:** Comentarios en HTML.
- Métodos de selección en jQuery:
    - `$("#ID")` → Selecciona un elemento por su ID.
    - `$("p")` → Selecciona elementos por su etiqueta.
    - `$(".clase")` → Selecciona elementos por su clase.
    - `$("[atributo]")` → Selecciona elementos por atributo.

Para asegurar que el código jQuery se ejecute después de cargar el DOM:

```
 $(document).ready(function(){

  $("#b1").click(function() {

    alert("Hola jQuery!");

  });

});
```

-

2.2. Funciones DOM en jQuery

a) Extracción de valores

- `getElementsByTagName("etiqueta")`: Obtiene todos los elementos de una etiqueta específica.
- `getElementsByName("nombre")`: Obtiene elementos con un atributo `name` específico.
- `getElementById("id")`: Obtiene un elemento por su ID.

b) Inserción de elementos

- `document.createElement("etiqueta")`: Crea un nuevo elemento HTML.
- `document.createTextNode("texto")`: Crea un nodo de texto.
- `element.appendChild(nuevoElemento)`: Agrega un elemento dentro de otro existente.

4. Eventos con jQuery: Teclado

- Los eventos de teclado detectan acciones producidas al presionar teclas.
- Se aplican generalmente al documento (`document`).
- Métodos principales:
  - `keypress()`: Mientras se mantiene pulsada una tecla.
  - `keyup()`: Al soltar una tecla.
  - `keydown()`: Al presionar una tecla sin necesidad de mantenerla.
- Ejemplo con `keydown()`: Muestra la tecla pulsada en un elemento HTML.

5. Eventos con jQuery: Ventana

- Detectan cambios en la ventana del navegador.
- Métodos principales:
  - `load()`: Cuando todos los elementos de la página han cargado.
  - `unload()`: Al salir de la página.
  - `scroll()`: Al desplazarse por la ventana.

    - `resize()`: Cuando se redimensiona la ventana.
- Ejemplo: Contador de desplazamiento con `scroll()` y detección de ancho de pantalla con `resize()`.

6. Eventos con jQuery: Ratón

- Detectan acciones realizadas con el puntero del ratón.
- Métodos principales:
    - `click()`: Al hacer clic en un elemento.
    - `dblclick()`: Al hacer doble clic.
    - `hover()`: Al pasar el ratón sobre un elemento.
    - `mousedown()`: Al presionar un botón del ratón.
    - `mouseup()`: Al soltar el botón del ratón.
    - `mouseenter()`: Al entrar el puntero en un elemento.
    - `mouseleave()`: Al salir el puntero de un elemento.
- Ejemplo con `mousedown()` y `mouseup()`: Detecta cuándo se presiona y suelta el botón del ratón.

7. Funciones interactivas con jQuery: `replaceWith()`

- Permite reemplazar un elemento HTML por otro.
- Se usa cuando se desea cambiar dinámicamente el contenido de la página.
- Ejemplo: Al hacer clic en un botón, cambia una imagen por otra.

8. Funciones interactivas con jQuery: `css()`

- Permite obtener y modificar estilos CSS de un elemento.
- Métodos principales:
    - `css('propiedad')`: Obtiene el valor de una propiedad CSS.
    - `css('propiedad', 'nuevo valor')`: Modifica una propiedad CSS.
    - `css({'propiedad1': 'valor1', 'propiedad2': 'valor2'})`: Modifica múltiples propiedades a la vez.
- Ejemplo: Cambiar el ancho, el color de fondo y la altura de un elemento con `css()`.

Tema 10

2. Conceptos fundamentales sobre accesibilidad

2.1 ¿Qué es la accesibilidad?

- La accesibilidad se refiere a la facilidad de acceso y manejo de un sitio web, asegurando que cualquier usuario pueda acceder a él sin importar sus características.
- Se enfoca en implementar mecanismos que faciliten el acceso, a diferencia de la usabilidad, que se centra en la facilidad de uso.

2.2 El Consorcio World Wide Web (W3C)

- El W3C es un consorcio internacional que desarrolla estándares y directrices para mejorar el acceso a la web.
- Iniciativas clave del W3C:
    - **Web Accessibility Initiative (WAI)**: Directrices de accesibilidad para la web 2.0.
    - **Pautas WCAG**: Pautas que facilitan la creación de sitios web accesibles.
    - **ISO/IEC 40500:2012**: Estándar internacional que recoge las pautas WCAG 2.0.
- **Niveles de conformidad**:
    - **Nivel A**: Requisitos mínimos.
    - **Nivel AA**: Nivel A + 13 criterios adicionales.
    - **Nivel AAA**: Nivel AA + 23 criterios adicionales.

3. Principios generales de accesibilidad

3.1 Principio 1: Diseño perceptible

- La información y los componentes deben ser mostrados de manera comprensible para todos los usuarios.

3.2 Principio 2: Diseño operable

- Los elementos de la interfaz deben ser fácilmente accesibles y operables.

Directrices:

- Usar texto alternativo para contenido no textual.
- Ofrecer alternativas para contenido multimedia.
- Asegurar que el contenido sea accesible por teclado.
- Garantizar tiempo suficiente para leer y usar el contenido.

3.3 Principio 3: Diseño comprensible

- El contenido y los elementos deben ser fácilmente comprensibles por los usuarios.

3.4 Principio 4: Diseño robusto

- El contenido debe ser robusto y compatible con tecnologías actuales y futuras.

Niveles de adecuación:

1. Cumplir al menos con uno de los niveles de conformidad (A, AA, AAA).
2. Las pautas deben aplicarse en toda la página web, no solo en algunas secciones.
3. Las pautas deben ser cumplidas en todo el sitio web, no solo en la página de inicio.
4. Las tecnologías deben ser compatibles con la accesibilidad.
5. La tecnología no debe interferir con la navegación del usuario.

5. Herramientas para la evaluación de la accesibilidad

a. Color Contrast Checker

- **Descripción**: Herramienta de WebAIM que evalúa la selección de colores en un sitio web.
- **Función**: Permite verificar el contraste entre el texto y el fondo, asegurando el cumplimiento de las pautas de accesibilidad.

b. Wave

- **Descripción**: Herramienta online de evaluación de accesibilidad (https://wave.webaim.org/).
- **Función**: Proporciona un análisis completo de la accesibilidad de un sitio web mostrando los problemas en la interfaz, destacando errores, advertencias y elementos no verificados.

6. TAW (Test de Accesibilidad Web)

- **Descripción**: Herramienta utilizada para evaluar el cumplimiento de un sitio web con los niveles de conformidad A, AA y AAA.
- **Función**: Permite analizar HTML, CSS y JavaScript de un sitio y genera un informe detallado sobre problemas, advertencias y no verificados. También permite recibir el informe por correo.
- **Certificación**: Ofrece certificación de accesibilidad a través de tres niveles:
    - **WAI-A**: Sin problemas automáticos de nivel 1 y problemas manuales de prioridad 1 resueltos.
    - **WAI-AA**: Sin problemas automáticos de nivel 1 y 2, ni problemas manuales de prioridad 1 y 2.
    - **WAI-AAA**: Sin problemas automáticos ni manuales de ningún nivel.

7. Hojas de estilo auditivas

- **Descripción**: Herramientas útiles para usuarios con dificultades visuales, que permiten convertir el contenido visual de una página en información auditiva.
- **Función**: Facilita la accesibilidad para personas con problemas de visión al proporcionar una forma de "leer" el contenido mediante audio, mejorando la interacción con la web.
- **Propiedades principales**:
    - **Volumen**: Controla el volumen de la voz que lee el contenido.
    - **Tipo de habla**: Define cómo se procesa el elemento (normal, eliminar sonido).
    - **Pausas**: Establece pausas durante la lectura del contenido.
    - **Azimuth**: Ajusta la dirección de la voz, creando una experiencia auditiva más humana.

Tema 11

2. Concepto de Usabilidad

La **usabilidad** se refiere a la facilidad con la que los usuarios pueden interactuar con un sitio web. En el contexto del diseño de interfaces web, implica la facilidad con la que los usuarios pueden navegar y realizar tareas en la interfaz. Como Jakob Nielsen indicó, "Si no lo haces fácil, los usuarios se marcharán de tu web".

Características de una buena usabilidad:

- **Eficiencia de uso**: Tiempo necesario para completar una tarea. Un diseño eficiente no debe hacer que el usuario pierda tiempo buscando botones o menús.
- **Facilidad de aprendizaje**: Los usuarios deben aprender rápidamente cómo interactuar con la interfaz.
- **Retención del tiempo**: La capacidad de recordar cómo utilizar la interfaz en visitas posteriores, especialmente para sitios web no utilizados de forma constante.
- **Satisfacción**: Grado de satisfacción del usuario con el sistema.

2.1. Principios Básicos y Recomendaciones (Jakob Nielsen, 1995)

1. **Visibilidad del estado del sistema**: El usuario debe saber siempre qué está sucediendo con la interfaz.
2. **Coincidencia entre el sistema y el mundo real**: Utilizar un lenguaje comprensible y familiar para los usuarios, teniendo en cuenta la diversidad cultural.
3. **Control y libertad del usuario**: Los usuarios deben poder navegar sin limitaciones innecesarias.
4. **Consistencia y estándares**: El diseño debe ser coherente y predecible para que el usuario no dude sobre qué acción realizar.
5. **Prevención de errores**: El diseño debe minimizar la posibilidad de errores por parte del usuario.
6. **Reconocer en lugar de recordar**: Minimizar la necesidad de que los usuarios recuerden cómo usar la interfaz; debe ser intuitiva.
7. **Flexibilidad y eficiencia de uso**: Adaptar la interfaz a usuarios novatos y experimentados, ofreciendo accesos rápidos para los más expertos.
8. **Diseño estético y minimalista**: Evitar la sobrecarga de información en los diálogos.
9. **Ayuda para reconocer y corregir errores**: Los mensajes de error deben ser claros y constructivos.
10. **Ayuda y documentación**: La documentación debe estar bien organizada y ser clara, aunque el diseño no debería depender excesivamente de ella.

3. Tipos de Usuario

Para lograr una interfaz usable, es fundamental adaptar el diseño a los diferentes tipos de usuarios:

- **Usuario registrado**: Accede a secciones privadas y utiliza control de sesiones.
- **Usuario beta tester o experto**: Testea el sitio antes de su lanzamiento, identificando posibles fallos.
- **Usuario anónimo**: Accede solo a la parte pública del sitio.

Además, la interfaz puede adaptarse según el tipo de acceso:

- **Interfaz de usuario de texto (TUI)**: Los usuarios interactúan mediante texto.
- **Interfaz gráfica de usuario (GUI)**: La más común, en la que los usuarios interactúan con el ratón y el teclado.
- **Interfaz de usuario de voz (VUI)**: Interacciones basadas en voz, útil para personas con dificultades motoras.
- **Interfaz de usuario natural (NUI)**: Utiliza gestos y movimientos como interacción.

5. Barreras Identificadas por los Usuarios

Existen diversas barreras que afectan la interacción de los usuarios con un sitio web, especialmente cuando se trata de personas con discapacidades o dificultades. Identificar y abordar estas barreras es fundamental para garantizar la accesibilidad y la usabilidad.

a. Dificultad visual
Los usuarios con dificultades visuales pueden enfrentar barreras como:

- Imposibilidad de acceder al contenido o manejar la interfaz sin teclado o ratón.
- Falta de textos alternativos para imágenes y elementos no textuales.
- Formularios y tablas complejas que son difíciles de interpretar.
- Diseño sin una correcta organización o uso inapropiado de elementos estructurales.
- Pobre contraste de colores en el diseño.
- Uso de CAPTCHAs sin alternativas accesibles, como una versión en audio.

b. Dificultad motora
Usuarios con afecciones motoras pueden enfrentar dificultades al interactuar con la interfaz. Algunas barreras son:

- Imposibilidad de usar el teclado o ratón. Se recomienda el uso de dispositivos alternativos, como trackballs.
- Enlaces gráficos o elementos que no son accesibles mediante herramientas de reconocimiento de voz.

c. Dificultad auditiva
Las barreras auditivas se relacionan principalmente con el contenido audiovisual, por ejemplo:

- Falta de transcripción de audio o subtítulos para contenido en vídeo o sonido.

6. Normativa

La **Organización Internacional de la Normalización (ISO)** establece normas para garantizar la calidad, seguridad y eficiencia de productos y servicios. En el contexto de la usabilidad web, algunas normas clave incluyen:

- **ISO/IEC 9126**: Estándar internacional para la calidad del software, abarcando modelos de calidad y métricas.

- **ISO/IEC 9241**: Guía de usabilidad que detalla los beneficios de las medidas de usabilidad.
- **ISO 13407**: Principios relativos al diseño de aplicaciones e interfaces centradas en el usuario.
- **ISO/TR 9241/151**: Directrices para el diseño de interfaces web centradas en el usuario.
- **UNE 139803:2004**: Requisitos de accesibilidad para usuarios con discapacidades al acceder a la web.

7. Navegación en la Web

La navegación web consiste en desplazarse por un conjunto de enlaces que llevan de una sección a otra de un sitio web. Para una navegación efectiva, deben cumplirse cuatro principios clave:

1. **Información accesible**: El acceso a cualquier parte del sitio debe ser sencillo, idealmente con la "regla de los tres clics", que establece que el usuario no debería necesitar más de tres clics para acceder a cualquier funcionalidad.
2. **Estructura consistente**: El sitio debe tener una estructura coherente y fácil de seguir sin necesidad de instrucciones continuas.
3. **Información persistente**: Elementos clave del sitio, como el encabezado o el menú de navegación, deben mantenerse consistentes en todas las páginas.
4. **Sencillez en la navegación**: La navegación debe ser fácil de recordar, asegurando que los usuarios siempre sepan dónde se encuentran y cómo volver o avanzar.

8. Análisis y Verificación de la Usabilidad: Técnicas

Existen dos tipos principales de pruebas para evaluar la usabilidad:

a. Pruebas con expertos
Los expertos realizan una **evaluación heurística** en la que analizan la aplicación en base a principios de usabilidad. Esta evaluación se divide en dos partes:

- **Evaluación detallada**: Análisis exhaustivo de cada pantalla, acción, menú, etc.
- **Evaluación de alto nivel**: Análisis general del funcionamiento del sitio.

b. Pruebas con usuarios
Las pruebas de usuarios se basan en la evaluación de un grupo real de usuarios que interactúan con el producto. Se utilizan cuestionarios y observación directa de su interacción para identificar posibles fallos. Este tipo de pruebas son fundamentales para descubrir problemas que los expertos pueden haber pasado por alto.

8.1. Herramientas para Evaluar la Usabilidad de un Sitio Web

Varias herramientas online permiten evaluar la usabilidad de un sitio web:

- **UsabilityHub**: Permite crear pruebas de usabilidad sobre el prototipo visual de la interfaz.

- **Crazy EGG**: Evalúa el comportamiento de los usuarios en la web, mostrando mapas de calor que indican las áreas más visitadas por los usuarios.
- **PageSpeed Insights**: Herramienta de Google que evalúa la usabilidad en términos de tiempos de carga y proporciona sugerencias para mejorar el rendimiento del sitio.

Estas herramientas ayudan a mejorar la usabilidad y la experiencia del usuario en los sitios web, lo que facilita una navegación más eficiente y agradable.

Tema 12

Resumen por puntos:

---

2. Patrones de Diseño. MVC:

- **React.js**: Librería creada por Facebook que facilita la creación de interfaces web dinámicas.
- **Modelo-Vista-Controlador (MVC)**: Un patrón de diseño de software que divide el código en tres partes:
    - **Modelo**: Gestiona los datos y la lógica de negocio, facilitando la recuperación y modificación de los datos.
    - **Vista**: Interfaz que permite la interacción del usuario con la aplicación. Se pueden utilizar librerías como Angular.js y React.js.
    - **Controlador**: Intermediario que recibe las solicitudes del usuario y las redirige adecuadamente para generar una respuesta.

---

3. Introducción a React.js:

- React.js está diseñado para manejar el **front-end** y mejorar la experiencia de usuario, limitando la interacción con el servidor.
- Utiliza **componentes reutilizables** para facilitar el desarrollo. Cada parte de la aplicación se maneja como un componente que puede ser reutilizado en otras partes.

---

4. Herramientas recomendadas para React.js:

- **Create React App**: Herramienta útil para crear proyectos de una sola página en React.js.

Comandos para crear un proyecto:
```
 npx create-react-app my-app
cd my-app
```

npm start

-
- **Node.js y npm**: Para utilizar React.js, necesitas tener Node.js y npm instalados. Los comandos:
    - `node --version` para verificar Node.js.
    - `npm --version` para verificar npm.

---

5. Componentes en React.js:

- **Componentes**: Son bloques reutilizables de código que forman la interfaz de usuario. Cada componente es un objeto JavaScript.
- **JSX**: Permite escribir HTML dentro de JavaScript, facilitando la creación de componentes.

Ejemplo de componente estático:

```
 import React from 'react';
const Button = () => (
    <a href="https://xxxx" className="button1">Inicio</a>
);
export default Button;
```

    -

---

6. Props en React.js:

- **Props**: Son elementos clave para hacer que los componentes sean dinámicos. Permiten pasar datos a los componentes, modificando su comportamiento y apariencia.

Ejemplo de uso:

```
 <p>{this.props.nombreAtributo}</p>
```

    -

Sintaxis:

```
 this.props.nombreAtributo
```

-

---

7. Estados en los componentes:

- Cada componente tiene un **estado interno** que refleja cómo está en un momento dado (por ejemplo, si un usuario ha interactuado con él).
- Se utiliza el **estado global** para comunicar los cambios de estado entre componentes.

Sintaxis para modificar el estado:

```
this.setState({
  nombreVariable: nuevoValor
});
```

9. Renderizando Componentes en React:

- La función **render()** permite mostrar un componente en el navegador.
- Cada vez que se crea o actualiza un componente, se llama a **render()** para mostrar el JSX en la interfaz.
- **Propiedades en componentes**: Los componentes pueden enviar parámetros a JSX para modelar el HTML mostrado en el navegador.

Ejemplo:

```
<Button url="urlA" class="buttonA" content="BOTÓN A" />
<Button url="urlB" class="buttonB" content="BOTÓN B" />
```

Esqueleto de componente con render:

```
class Componente extends React.Component {
  constructor(...args) {
    // Constructor
  }
  render() {
    // JSX
  }
}
```

---

10. Composición de Componentes:

- **Composición de componentes**: Un componente puede invocar a otros componentes para construir la interfaz de forma modular.

    - Ejemplo de componente **Saludar**:

```
 function Saludar(props) {
  return <h2>Hola {props.name}</h2>;
}
function ComponentePrincipal() {
  return (
    <div>
      <Saludar name="Diana" />
      <Saludar name="Pepe" />
    </div>
  );
}
```

- La composición facilita fragmentar el código, permitiendo manejar partes específicas de la interfaz por separado.

---

11. Caso práctico 2: Creación de Componentes con Estado:

- **Objetivo**: Crear un componente que almacene y modifique el estado de la aplicación, como un contador que incrementa su valor.
- **Componentes con estado**: Los componentes pueden tener un estado interno (`state`) que solo es accesible por el propio componente y sus hijos.

Parámetros de diseño:

- **Componente Incrementador**: Un componente que incrementa una variable al presionar un botón.
- **Estado inicial**: `variableIncrementar = 0`

Código de ejemplo:

```
class Incrementador extends React.Component {
  constructor(props) {
    super(props);
    this.state = { variableIncrementar: 0 };
  }
  render() {
    return (
      <div>
        <p>Valor: {this.state.variableIncrementar}</p>
```

```
        <button onClick={() => this.setState({ variableIncrementar: this.state.variableIncrementar + 1 })}>
          Incrementar
        </button>
      </div>
    );
  }
}
```

Este resumen cubre la renderización de componentes, cómo componer múltiples componentes y la gestión del estado dentro de un componente en React. Si tienes dudas o necesitas más detalles, ¡avísame!

Tema 13

Resumen por puntos:

---

3. Routing en React:

- **Routing**: El proceso de determinar cómo llegar de un punto A a un punto B, creando rutas entre las diferentes páginas o secciones de un sitio web.
- Se definen rutas en el código para que los usuarios puedan navegar entre ellas.

Eventos de atención:

- **onFocus**: Se dispara cuando un elemento recibe el foco.
- **onBlur**: Se dispara cuando un elemento pierde el foco.
- **onChange**: Se dispara cuando el contenido de un elemento cambia (por ejemplo, un `select`).

Eventos de formulario:

- **onSubmit**: Se dispara cuando se envía un formulario.
- **onReset**: Se dispara cuando se restablece un formulario.

3.1. Optimización de rutas y React Router:

- **Problema**: A medida que aumentan las rutas, el código dentro de las etiquetas `<Switch>` también aumenta. Para optimizar esto, se utiliza un objeto de rutas con dos propiedades clave:
    - **component**: El componente asociado a la ruta.
    - **path**: La cadena de texto que representa la ruta.

Código 5 (Implementación básica):

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
import SeccionPpal from "./components/SeccionPpal";
import View1 from "./components/views/View1";
import View2 from "./components/views/View2";

export default function App() {
  return (
    <Router>
      <SeccionPpal />
      <Switch>
        <Route path="view-1" component={View1} />
        <Route path="view-2" component={View2} />
      </Switch>
    </Router>
  );
}
```

Optimización con objetos:

```
const View1 = {
  component: views.View1,
  path: "/view-1"
};
```

Instalación y Configuración de React Router:

- **React Router**: Una librería para gestionar la navegación en aplicaciones React.
- **Comandos**:

Para instalar React Router:
```
npm install --save react-router
```

1.

Para importarlo en tu código:
```
import { Router, Route, browserHistory } from 'react-router';
```

2.

---

4. Patrones de Software: Flux:

- **Flux**: Es una arquitectura desarrollada por Facebook para gestionar el flujo de datos en aplicaciones front-end. Se utiliza junto con React.
- **Capas de Flux**:
    - **Views**: Componentes React que representan la interfaz.
    - **Stores**: Almacenan los datos y gestionan la conexión con las vistas y acciones.
    - **Actions**: Modelan las funciones y operaciones en la app, disparadas por eventos en las vistas.
    - **Dispatcher**: Se encarga de direccionar las acciones hacia el store correspondiente.

Este resumen cubre el routing con React Router y la arquitectura Flux, lo cual facilita la navegación y gestión de datos en una aplicación React. Si tienes más preguntas sobre algún tema, ¡no dudes en preguntar!

Tema 15

2. Introducción a Redux

Redux es una librería JavaScript basada en el patrón de arquitectura Flux. Su objetivo principal es separar el desarrollo visual de los datos de estado de una aplicación. En otras palabras, gestiona el estado global de la aplicación en un único lugar (el store), lo que facilita el control de los datos en aplicaciones grandes y complejas.

Objetivo de Redux:

- Mantener el estado de los componentes de la interfaz centralizado y consistente.
- Permitir un flujo de datos predecible, gestionando los cambios del estado de manera controlada.

Instalación de Redux

Redux se instala mediante npm (Node Package Manager). Para comenzar a usarlo, primero es necesario tener instalado Node.js en el equipo.

Instalar Redux:

npm i -S redux

1.
2. **Instalar paquetes adicionales para React y herramientas de desarrollo**:

React-Redux: Biblioteca para integrar Redux con React.
npm i -S react-redux

-

Redux DevTools: Herramienta de desarrollo para facilitar la depuración.
npm i -D redux-devtools

-

2.1. Creación del Store con Redux

En Redux, se utiliza **un único store** (a diferencia de Flux, que puede tener múltiples stores). Este store contiene todo el estado de la aplicación y sigue una estructura en forma de árbol, donde cada pieza de datos se inserta como un nodo en ese árbol.

Modelo de Store en Flux:

- En Flux, múltiples stores están conectados al dispatcher y gestionan diferentes partes del estado.

Modelo de Store en Redux:

- En Redux, el estado de toda la aplicación se almacena en un único store que tiene una estructura de árbol.

Código para crear un Store único en Redux:

El store en Redux es comúnmente creado en un archivo como `./src/store/index.js`. En este archivo, se configura el estado global de la aplicación y se define cómo interactuar con el mismo.

Ejemplo básico de cómo se crea el store en Redux:

```
import { createStore } from 'redux';

// Estado inicial
const initialState = {
```

```
  count: 0
};

// Reducer: describe cómo cambia el estado en respuesta a las acciones
function rootReducer(state = initialState, action) {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
}

// Crear el store con el reducer
const store = createStore(rootReducer);

export default store;
```

Este código muestra un ejemplo de cómo configurar un **store** en Redux, donde el estado inicial tiene una propiedad `count`, y el **reducer** define cómo se modifica ese estado en función de las acciones (`INCREMENT` o `DECREMENT`).

Flux vs Redux:

- **Flux**: Varios stores, cada uno con un estado específico.
- **Redux**: Un único store para manejar todo el estado global de la aplicación.