**part 0: FUNDAMENTOS DE LAS APLICACIONES WEB**

HTTP GET

El **servidor** y el **navegador web** se comunican entre sí mediante el protocolo HTTP. La pestaña Network (Red) muestra cómo se comunican el navegador y el servidor.

APLICACIONES WEB TRADICIONALES

Al ingresar a la página, el navegador obtiene el documento HTML que detalla la estructura y el contenido textual de la página desde el servidor. El servidor ha formado este documento de alguna manera:

* El documento puede ser un archivo de texto estático guardado en el directorio del servidor.
* El servidor también puede formar los documentos HTML dinámicamente de acuerdo con el código de la aplicación, utilizando, por ejemplo, datos de una base de datos. El código HTML de la aplicación de ejemplo se ha formado de forma dinámica, porque contiene información sobre el número de notas creadas.

CONTROLADORES DE EVENTOS Y FUNCIONES DE DEVOLUCIÓN DE LLAMADA

La solicitud al servidor se envía en la última línea del JS, pero el código para manejar la respuesta se puede encontrar más arriba.

Un **controlador de eventos** es un método que está enlazado a un evento. Cuando ese evento se genera, el código del controlador de eventos se ejecuta. Cada controlador de eventos proporciona dos parámetros que permiten controlar el evento correctamente.

Cuando cambia el estado del objeto, el navegador llama a la función del controlador de eventos. El código de la función verifica que readyState sea igual a 4 (que describe la situación La operación está completa) y que el código de estado HTTP de la respuesta es 200.

El mecanismo de invocación de controladores de eventos es muy común en JavaScript. Las funciones del controlador de eventos se denominan **funciones de devolución de llamada (callback)**. El código de la aplicación no invoca las funciones en sí, sino el entorno de ejecución –el navegador–, invoca la función en el momento adecuado, cuando se ha producido el evento.

MODELO DE OBJETOS DEL DOCUMENTO O DOM

Podemos pensar en las páginas HTML como estructuras de árbol implícitas.

El funcionamiento del navegador se basa en la idea de representar los elementos HTML como un árbol. Document Object Model, o DOM es una interfaz de programación de aplicaciones, (una API), que permite la modificación programática de árboles de elementos correspondientes a páginas web.

El código JavaScript introducido en el capítulo anterior utilizó DOM-API para agregar una lista de notas a la página.

CSS

El elemento head del código HTML de la página de Notes contiene un enlace, que determina que el navegador debe obtener una hoja de estilos CSS de la dirección main.css.

Cascading Style Sheets (Hojas de Estilos en Cascada), o CSS, es un lenguaje de hojas de estilo utilizado para determinar la apariencia de las páginas web.

El archivo define dos **selectores de clase**. Se utilizan para seleccionar ciertas partes de la página y definir reglas de estilo para aplicarles estilo. Una definición de selector de clase siempre comienza con un punto y contiene el nombre de la clase.

Las **clases** son atributos, que se pueden agregar a elementos HTML.

* La regla CSS define que los elementos con la misma clase tendrán el mismo estilo.
* Los elementos HTML también pueden tener otros atributos además de clases como, por ejemplo, un atributo id. El código JavaScript usa el id para encontrar el elemento.

La pestaña Elements (Elementos) de la consola se puede utilizar para cambiar los estilos de los elementos.

Cualquier cambio que quiera realizarse de forma permanente debe realizarse en los archivos del servidor.

**Cargando una página que contiene JavaScript – revisión**

Imagen que contiene Tabla

Descripción generada automáticamente

* El navegador ejecuta el código JavaScript. El código realiza una solicitud HTTP GET a la dirección https://studies.cs.helsinki.fi/exampleapp/data.json, que devuelve las notas como datos JSON.
* Cuando se han obtenido los datos, el navegador ejecuta un controlador de eventos, que muestra las notas en la página utilizando DOM-API.

AJAX

AJAX (JavaScript Asíncrono y XML) es un término introducido en febrero de 2005 sobre la base de los avances en la tecnología de los navegadores para describir un nuevo enfoque revolucionario que **permitió la obtención de contenido en páginas web utilizando JavaScript incluido dentro del HTML, sin la necesidad de volver a renderizar la página**.

Antes de la era AJAX, todas las páginas web funcionaban como la aplicación web tradicional que vimos anteriormente en este capítulo. Todos los datos que se muestran en la página se obtuvieron con el código HTML generado por el servidor.

La página Notas utiliza AJAX para obtener los datos de las notas. El envío del formulario todavía utiliza el mecanismo tradicional de envío de formularios web. Las URLs de la aplicación reflejan los viejos tiempos sin preocupaciones. Los datos JSON se obtienen de la URL https://studies.cs.helsinki.fi/exampleapp/data.json y se envían nuevas notas a la URL https://studies.cs.helsinki.fi/exampleapp/new\_note. Hoy en día, URLs como estas no se consideran aceptables, ya que no siguen las convenciones generalmente reconocidas de las APIs RESTful, que analizaremos más en la parte 3. La cosa denominada AJAX es ahora tan común que se da por sentado. El término se ha desvanecido en el olvido, y la nueva generación ni siquiera ha oído hablar de él.

APLICACIÓN DE UNA SOLA PÁGINA

En nuestra aplicación de ejemplo, la página de inicio funciona como una página web tradicional: toda la lógica está en el servidor y el navegador solo muestra el HTML como se indica.

La página Notas da parte de la responsabilidad al navegador, la generación del código HTML para las notas existentes. El navegador aborda esta tarea ejecutando el código JavaScript que obtuvo del servidor. El código obtiene las notas del servidor como datos JSON y agrega elementos HTML para mostrar las notas en la página usando la DOM-API.

En los últimos años, ha surgido el estilo de **Aplicación de una sola página (SPA)** para crear aplicaciones web. Los sitios web de estilo SPA no obtienen todas sus páginas por separado del servidor como lo hace nuestra aplicación de muestra, sino que comprenden solo una página HTML obtenida del servidor, cuyo contenido se manipula con JavaScript que se ejecuta en el navegador.

La página Notas de nuestra aplicación tiene cierto parecido con las aplicaciones de estilo SPA, pero aún no está del todo lista. Aunque la lógica para representar las notas se ejecuta en el navegador, la página sigue utilizando la forma tradicional de agregar nuevas notas. Los datos se envían al servidor con el envío del formulario, y el servidor indica al navegador que vuelva a cargar la página Notas con un redireccionamiento.

Puedes encontrar una versión "aplicación de una sola página" de nuestra aplicación de ejemplo en https://studies.cs.helsinki.fi/exampleapp/spa. A primera vista, la aplicación se ve exactamente igual que la anterior. El código HTML es casi idéntico, pero el archivo JavaScript es diferente (spa.js) y hay un pequeño cambio en **cómo se define la etiqueta form**. El formulario no tiene atributos de *action* o *method* para definir cómo y dónde enviar los datos de entrada.

Abre la pestaña Network (Red) y vacíala. Cuando ahora crees una nueva nota, notaras que el navegador envía solo una solicitud al servidor. La versión SPA de la aplicación no envía los datos del formulario de la forma tradicional, sino que utiliza el código JavaScript que obtuvo del servidor.

LIBRERÍAS JAVASCRIPT

La aplicación de muestra se realiza con el llamado vanilla JavaScript, utilizando solo la DOM-API y JavaScript para manipular la estructura de las páginas.

En lugar de utilizar JavaScript y DOM-API únicamente, a menudo se utilizan diferentes librerías que contienen herramientas con las que es más fácil trabajar en comparación con DOM-API para manipular páginas. Una de estas librerías es la popular jQuery.

**jQuery** se desarrolló cuando las aplicaciones web seguían principalmente el estilo tradicional del servidor que genera páginas HTML, cuya funcionalidad se mejoró en el lado del navegador usando JavaScript escrito con jQuery. Una de las razones del éxito de jQuery fue la llamada compatibilidad entre navegadores. La librería funcionó independientemente del navegador o de la empresa que la hizo, por lo que no hubo necesidad de soluciones específicas para cada navegador. Hoy en día, el uso de jQuery no está tan justificado dado el avance de **VanillaJS**, y los navegadores más populares generalmente soportan bien las funcionalidades básicas.

El auge de las aplicaciones de una sola página trajo varias formas más "modernas" de desarrollo web que jQuery. El favorito de la primera ola de desarrolladores fue **BackboneJS**. Después de su lanzamiento en 2012, **AngularJS** de Google rápidamente se convirtió casi en el estándar de facto del desarrollo web moderno.

Sin embargo, la popularidad de Angular se desplomó después de que el equipo de Angular anunció que el soporte para la versión 1 terminaría, y que Angular 2 no sería retrocompatible con la primera versión. Angular 2 y las versiones más nuevas no recibieron una bienvenida muy cálida.

Actualmente, la herramienta más popular para implementar la lógica del lado del navegador en las aplicaciones web es la biblioteca **React** de Facebook. Durante este curso, nos familiarizaremos con React y la biblioteca **Redux**, que se usan juntos con frecuencia. El estado de React parece sólido, pero el mundo de JavaScript cambia constantemente. Por ejemplo, recientemente un recién llegado -VueJS- ha estado captando cierto interés.

DESARROLLO WEB FULL STACK

¿Qué significa el nombre del curso, Desarrollo web full stack? Full stack es una palabra de moda de la que todo el mundo habla, aunque nadie sabe realmente lo que significa. O al menos, no existe una definición acordada para el término.

Prácticamente todas las aplicaciones web tienen (al menos) dos "capas": el navegador, al estar más cerca del usuario final es la capa superior, y el servidor la inferior. A menudo también hay una capa de base de datos debajo del servidor. Por lo tanto, podemos pensar en la arquitectura de una aplicación web como una especie de stack (pila) de capas.

A menudo, también hablamos sobre el frontend y el backend. El navegador es el frontend y el JavaScript que se ejecuta en el navegador es el código del frontend. El servidor, por otro lado, es el backend.

En el contexto de este curso, el desarrollo web full stack significa que nos enfocamos en todas las partes de la aplicación: **el frontend**, **el backend** y **la base de datos**. A veces, el software del servidor y su sistema operativo se ven como parte del stack, pero no vamos a entrar en ellos.

Programaremos el backend con JavaScript, utilizando el entorno de ejecución Node.js. El uso del mismo lenguaje de programación en múltiples capas de la pila le da al desarrollo web full stack una dimensión completamente nueva. Sin embargo, no es un requisito del desarrollo web full stack utilizar el mismo lenguaje de programación (JavaScript) para todas las capas del stack.

Solía ser más común que los desarrolladores se especializaran en una capa del stack, por ejemplo, el backend. Las tecnologías en el backend y el frontend eran bastante diferentes. Con la tendencia full stack, se ha vuelto común que los desarrolladores dominen todas las capas de la aplicación y la base de datos. A menudo, los desarrolladores full stack también deben tener suficientes habilidades de configuración y administración para operar su aplicación, por ejemplo, en la nube.

**Fatiga de JavaScript**

El desarrollo web full stack es un desafío en muchos aspectos. Suceden cosas en muchos lugares a la vez y la depuración es un poco más difícil que con las aplicaciones de escritorio normales. JavaScript no siempre funciona como cabría esperar (en comparación con muchos otros lenguajes), y la forma asíncrona en que funcionan sus entornos de ejecución genera todo tipo de desafíos. Comunicarse en la web requiere conocimientos del protocolo HTTP. También se deben manejar las bases de datos y la administración y configuración del servidor. También sería bueno saber suficiente CSS para hacer las aplicaciones al menos algo presentables.

El mundo de JavaScript se desarrolla rápidamente, lo que conlleva sus propios desafíos. Las herramientas, las librerías y el propio lenguaje están en constante desarrollo. Algunos están empezando a cansarse del cambio constante y han acuñado un término para ello: fatiga de JavaScript. Ve como manejar la fatiga de JavaScript con auth0 o fatiga de JavaScript en Medium.

**part 1: INTRODUCCIÓN A REACT**

La finalidad principal de React es facilitar la creación de interfaces de usuario interactivas y dinámicas en aplicaciones web. React es una biblioteca de JavaScript de código abierto desarrollada por Facebook que se utiliza comúnmente para construir interfaces de usuario de una sola página (Single Page Applications - SPAs) y aplicaciones web de alto rendimiento. Algunos de los objetivos principales de React son:

1. **Componentización**: React fomenta la creación de aplicaciones mediante componentes reutilizables. Los componentes en React permiten dividir la interfaz de usuario en piezas independientes y modulares, lo que facilita el mantenimiento y la reutilización del código.

2. **Virtual DOM**: React utiliza un modelo de DOM virtual para optimizar el rendimiento de las aplicaciones web. En lugar de actualizar directamente el DOM del navegador cada vez que cambia el estado de la aplicación, React compara el estado actual del DOM virtual con su estado anterior y solo actualiza los elementos que han cambiado, lo que resulta en una mayor eficiencia y rendimiento.

3. **Declarativo**: React utiliza un enfoque declarativo para definir la interfaz de usuario, lo que significa que los desarrolladores describen cómo debería ser la interfaz de usuario en un momento dado, en lugar de tener que preocuparse por manipular manualmente el DOM para reflejar los cambios en el estado de la aplicación.

4. **Reactivity**: React facilita la creación de interfaces de usuario reactivas, lo que significa que los componentes pueden actualizarse automáticamente cuando cambia el estado de la aplicación, lo que reduce la necesidad de manejar manualmente la actualización del DOM.

En resumen, la finalidad de React es proporcionar a los desarrolladores una herramienta eficiente y poderosa para construir interfaces de usuario interactivas y dinámicas en aplicaciones web de una manera modular, eficiente y fácil de mantener.

Comenzamos con la creación de una aplicación React simple y con el conocimiento de los conceptos básicos de React. La forma más fácil de empezar es utilizando una herramienta llamada **Vite**. Vite inicia la aplicación por defecto en el puerto 5173. Si este no está libre, Vite utiliza el siguiente numero de puerto libre.

**Componente**

El archivo App.js define un componente de React con el nombre App. El comando en la línea final del archivo main.jsx renderiza su contenido dentro del elemento div, definido en el archivo index.html, que tiene el valor 'root' en el atributo id.

Cuando se usa React, todo el contenido que necesita ser renderizado es generalmente definido como componentes de React.

**Renderizar**

En el contexto del desarrollo frontend, "renderizar" se refiere al proceso de convertir datos en una representación visual en la interfaz de usuario (UI) de una aplicación web. Este proceso implica tomar datos, generalmente en forma de estructuras de datos como objetos JavaScript o datos provenientes de una API, y convertirlos en elementos visuales en el navegador web, como HTML, CSS y elementos de la interfaz gráfica.

En resumen, renderizar en el frontend implica tomar datos y transformarlos en la interfaz gráfica que los usuarios ven y con la que interactúan en un navegador web. Esto se logra utilizando tecnologías y frameworks frontend como HTML, CSS, JavaScript y bibliotecas/frameworks populares como React, Vue.js, Angular, etc.

Cualquier código de JavaScript entre llaves es evaluado y el resultado de esta evaluación se incrusta en el lugar definido en el HTML producido por el componente.

El **export** no se muestra en la mayoría de los ejemplos del material de este curso. Sin este export el componente y la aplicación completa se romperían.

**JSX**

Un componente no es más que la declaración de una función.

Parece que los componentes de React están devolviendo marcado HTML. Sin embargo, éste no es el caso. El diseño de los componentes de React se escribe principalmente usando JSX. Aunque JSX se parece a HTML, en realidad estamos tratando con una forma de escribir JavaScript. Bajo el capó, el JSX devuelto por los componentes de React se compila en JavaScript.

También es posible escribir React como "JavaScript puro" sin usar JSX. Aunque, nadie con una mente sana lo haría realmente.

En la práctica, JSX se parece mucho a HTML con la distinción de que con JSX puede incrustar fácilmente contenido dinámico escribiendo JavaScript apropiado entre llaves. La idea de JSX es bastante similar a muchos lenguajes de plantillas, como Thymeleaf, que se utiliza junto con Java Spring, que se utiliza en servidores.

JSX es similar a XML, lo que significa que todas las etiquetas deben cerrarse.

Escribir componentes con React es fácil, y al combinar componentes, incluso una aplicación más compleja puede ser bastante fácil de mantener. De hecho, una filosofía central de React es componer aplicaciones a partir de muchos componentes reutilizables especializados.

Otra fuerte convención es la idea de un componente raíz llamado App en la parte superior del árbol de componentes de la aplicación. Sin embargo, como aprenderemos en la parte 6, hay situaciones en las que el componente App no es exactamente la raíz, sino que está incluido en un componente de utilidad apropiado.

**props: pasar datos a componentes**

Puede haber un número arbitrario de props y sus valores pueden ser strings "incrustados en el código" ("hard coded") o resultados de expresiones JavaScript. Si el valor del prop se obtiene usando JavaScript, debe estar envuelto con llaves.

* Los nombres de los componentes de React deben iniciar con mayúscula.
* El contenido de un componente de React (normalmente) debe contener un elemento raíz. Si, por ejemplo, intentamos definir el componente App sin el elemento div más externo.
* En React, las cosas individuales a ser renderizadas dentro de llaves deben ser valores primitivos, como números o cadenas.
* React también permite renderizar arreglos si el arreglo contiene valores que son elegibles para renderizar (como números y cadenas).
* NO hay que crear componentes dentro de otros componentes.

**part 11: INTRODUCTION TO CI/CD**

* Los Sistemas de Integración Continua (CI) son MV, servidores que nos permiten operar en la nube para asegurar que trabajamos siempre con la misma configuración y tenemos una extracción limpia.
* Los principales son GitHub Actions, Jenkins y Travis CI.