

Desenrotllament d'Aplicacions Web – 1r curs del Cicle Formatiu de Grau Superior

RA5.g) – LMSGI

RA5.g) Se han realizado conversiones sobre documentos para el intercambio de información.
10%

Enunciado:

En esta actividad, aprenderás a obtener datos desde una API en formato JSON, procesarlos con JavaScript para transformarlos a formato XML y mostrar el documento XML resultante en pantalla.

La conversión se realizará en el navegador mediante código JavaScript, sin utilizar servidores ni herramientas externas.

Utilizarás la función `fetch()` para hacer una solicitud HTTP y trabajarás con promesas, entendiendo cómo fluye la información en cada paso del proceso.

Conceptos clave:

a) ¿Qué es una API y cómo se obtienen datos desde una API?

Una API (Application Programming Interface) es una interfaz que permite la comunicación entre diferentes aplicaciones.

Para obtener datos desde una API, se usa `fetch(URL)`, que realiza una solicitud HTTP y devuelve una respuesta, normalmente preparada en formato XML o en formato JSON.

b) ¿Qué es JSON y cómo se convierte en un objeto de JavaScript?

JSON (JavaScript Object Notation) es un formato de intercambio de datos basado en texto.

La información está estructurada en formato de pares clave-valor.

Ejemplo 1: Objeto JSON simple

```
{"nombre": "Juan", "edad": 30};
```

Es una estructura compacta que se usa comúnmente para representar una entidad con propiedades específicas.

Ejemplo 2: Array de objetos JSON

```
[  
  { "id": 1, "name": "Leanne Graham", "email": "Sincere@april.biz" },  
  { "id": 2, "name": "Ervin Howell", "email": "Shanna@melissa.tv" }  
]
```

Contiene una lista de dos objetos, donde cada objeto representa una entidad con tres pares claves-valor.

Para convertir los objetos en formato JSON, en objetos para JavaScript, se usa `JSON.parse(textoJSON)`, o si los datos vienen de una API en formato JSON, entonces se encargará de convertirlo los objetos JSON a objetos de JavaScript el método `response.json()` que se encuentra dentro del `fetch()`.

Ejemplo:

```
let datosJSON = '{"nombre": "Juan", "edad": 30}';  
let objeto = JSON.parse(datosJSON); // Convierte JSON en formato texto a objeto JavaScript  
console.log(objeto.nombre); // "Juan"
```

c) ¿Cómo funcionan las promesas (.then(), .catch()) en fetch()?

Las promesas manejan operaciones asincrónicas, lo que significa que cuando el sistema local realiza una solicitud a un sistema remoto, no detiene la ejecución del script mientras espera la respuesta. En su lugar, continúa ejecutando otras instrucciones.

Cuando los datos son recibidos, la promesa se resuelve y ejecuta el código asociado, como si los hubiera recibido de inmediato.

Entonces:

- .then() se ejecuta cuando la promesa se resuelve con éxito y recibe el valor devuelto.
- .catch() captura errores en la petición.

Ejemplo:

```
fetch("https://fakestoreapi.in/api/users")  
  .then(response => response.json()) // Convertir respuesta a JSON  
  .then(data => mostrarUsuarios(data.users)) // Pasar solo la lista de usuarios  
  .catch(error => console.error("Error al obtener los datos:", error));
```

```
{  
  "status": "SUCCESS",  
  "message": "Here you go! You've received 20 users.",  
  "users": [  
    {  
      "id": 1,  
      "email": "michael@simpson.com",  
      "username": "michaelsimpson",  
      "password": "@K(5UejhL&",  
      "name": {  
        "firstname": "Michael",  
        "lastname": "Simpson"  
      },  
      "address": {  
        "city": "Joelton",  
        "street": "Angela Spring",  
        "number": "868",  
        "zipcode": "75070",  
        "geolocation": {  
          "lat": 19.7091875,  
          "long": -14.782061  
        }  
      },  
      "phone": "562.548.9768x73853"  
    },  
    // {...}  
    {  
      "id": 20,  
      "email": "timothy@burton.com",  
      "username": "timothyburton",  
      "password": "&$)QeGpZ25",  
      "name": {  
        "firstname": "Timothy",  
        "lastname": "Burton"  
      },  
      "address": {  
        "geolocation": {...}  
      },  
      "phone": "+1-293-912-5353x125"  
    }  
  ]  
}
```

¿Cuál es el flujo de ejecución en el anterior código?

- `fetch("https://fakestoreapi.in/api/users"):`
 - Se envía la solicitud a la URL indicada.
- `.then(response => response.json()):`
 - Convierte la respuesta a formato JSON.
 - Puede fallar si la respuesta no es válida.
- `.then(data => mostrarUsuarios(data.users)):`
 - Toma los datos y los pasa a la función `mostrarUsuarios(data.users)`.
 - Puede fallar si `data.users` no existe.
- `.catch(error => console.error("Error al obtener los datos:", error)):`
 - Si ocurre un error en cualquiera de los pasos anteriores, se ejecuta este bloque, capturando el error y mostrando un mensaje en la consola.

d) ¿Cómo manipular el DOM para mostrar los datos en una página web?

Se usa JavaScript para crear y modificar elementos HTML dinámicamente.

Ejemplo de cómo agregar usuarios a una lista `<ul id=listaUsuarios>`:

```
function mostrarUsuarios(usuarios) {  
  const lista = document.getElementById("listaUsuarios");  
  lista.innerHTML = ""; // Limpia la lista  
  
  usuarios.forEach(usuario => {  
    let item = document.createElement("li");  
    item.textContent = `${usuario.name} - ${usuario.email}`;  
    lista.appendChild(item);  
  });  
}
```

e) ¿Qué es XML y en qué se diferencia de JSON?

XML (Extensible Markup Language) es un formato de almacenamiento e intercambio de datos basado en etiquetas.

Se diferencia de JSON en que utiliza una estructura jerárquica con nodos anidados en lugar de pares clave-valor.

Ejemplo de datos en JSON:

```
{  
  "nombre": "Juan",  
  "edad": 30  
}
```

Ejemplo del mismo contenido en XML:

```
<persona>  
  <nombre>Juan</nombre>  
  <edad>30</edad>  
</persona>
```

f) ¿Cómo convertir JSON a XML en JavaScript?

Para convertir un objeto JSON a XML en JavaScript, se pueden seguir estos pasos:

- Crear un documento XML vacío.
- Recorrer los datos en formato JSON.
- Crear elementos XML y asignarles valores.
- Agregar los elementos al documento XML.
- Convertir el XML a texto para mostrarlo en pantalla.

Ejemplo simple en JavaScript:

Ejemplo:

```
let jsonData = { "nombre": "Juan", "edad": 30 };

let xmlDocument = document.implementation.createDocument("", "", null);

let root = xmlDocument.createElement("persona");

let nombre = xmlDocument.createElement("nombre");
nombre.textContent = jsonData.nombre;

root.appendChild(nombre);

let edad = xmlDocument.createElement("edad");
edad.textContent = jsonData.edad;

root.appendChild(edad);

xmlDocument.appendChild(root);

let serializer = new XMLSerializer();
console.log(serializer.serializeToString(xmlDocument));
```

g) ¿Cómo mostrar XML en pantalla?

Para mostrar el XML en una página web, podemos insertar el resultado dentro de un contenedor `<pre>` para mantener el formato:

```
<pre id="resultado"></pre>
```

Y entonces, desde JavaScript:

```
document.getElementById("resultado").textContent =
serializer.serializeToString(xmlDocument);
```

h) ¿Cómo guardar el documento XML en un archivo?

Necesitamos saber que:

- URL es un objeto global en JavaScript que pertenece al navegador y se usa para manipular y analizar direcciones web (URLs).
- `new Blob()` es un constructor en JavaScript que crea un **objeto** Blob (Binary Large Object).
 - Dicho objeto creado **se usa para almacenar datos en forma de archivos en memoria** dentro del navegador.

Ejemplo:

```
const blob = new Blob([xmlActual], { type: "text/xml" });

const enlace = document.createElement("a");
enlace.href = URL.createObjectURL(blob);
```

`enlace.download = "usuarios.xml"; // download define el nombre con el que el archivo
// se guardará en la computadora del usuario.`

`enlace.click();`

Para guardar el documento XML desde el navegador web a un archivo:

- Generamos el archivo a través de new Blob:
 - El Blob almacena los datos de `xmlActual`, pero en un formato que el navegador puede tratar como un archivo.
 - **blob** no es directamente un string, sino un objeto Blob que tiene:
 - El tamaño del contenido en bytes.
 - El tipo MIME ("text/xml").
 - Los datos de `xmlActual` en un formato binario interno.
- Creamos un enlace de descarga a dicho archivo y simulamos un click

Aunque Blob se asocia con datos binarios, también es útil para texto.

Lo usamos aquí porque nos permite crear un archivo descargable directamente desde el navegador de forma eficiente y compatible con todos los navegadores modernos.

i) ¿Cómo cargar el documento XML desde un archivo XML en el navegador web?

Necesitamos saber que:

- Tendremos que hacer uso de objetos event que son los que contienen información sobre la acción del usuario, como por ejemplo cuando el usuario selecciona un archivo XML en un `<input type="file">`.
- `event.target.files[0]` hace referencia al archivo seleccionado por el usuario, de manera que:
 - `event` es el objeto del evento que contiene información sobre la acción del usuario.
 - `event.target.files` hace referencia a la lista de archivos que el usuario ha seleccionado (target)
 - `[0]` nos quedamos con el primero de los archivos
- `FileReader` es un objeto de JavaScript que permite leer archivos desde el dispositivo del usuario.
- `reader.readAsText(file)` inicia la lectura del archivo en formato texto, en segundo plano.

Ejercicio 1

Crea un archivo `index.html`, un `script.js` y un archivo `styles.css`

El código que contendrá cada uno de ellos figura a continuación:

`index.html`

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="styles.css">
  <title>Lista de Usuarios</title>
</head>
<body>

  <h1>Lista de Usuarios</h1>

  <button id="cargarUsuarios">Cargar Usuarios</button>
  <button id="guardarXML" disabled>Guardar XML</button>
  <button id="cargarXML">Cargar XML desde Archivo</button>

  <input type="file" id="inputXML" style="display:none" accept=".xml">

  <pre id="resultado"></pre>

  <script src="script.js"></script>

</body>
</html>
```

`script.js`

```
document.addEventListener("DOMContentLoaded", function() {
  document.getElementById("cargarUsuarios").addEventListener("click", obtenerUsuarios);
  document.getElementById("guardarXML").addEventListener("click", guardarXML);
  document.getElementById("cargarXML").addEventListener("click", function() {
    document.getElementById("inputXML").value = ""; // Resetea el input para permitir
    cargar otro archivo
    document.getElementById("inputXML").click();
  });
  document.getElementById("inputXML").addEventListener("change", cargarXML);
});

let xmlActual = ""; // Variable para almacenar el XML generado

function obtenerUsuarios() {
  fetch("https://fakestoreapi.com/users")
    .then(response => response.json())
    .then(data => {
```



```
    let xmlData = convertirJSONaXML(data);
    xmlActual = xmlData;
    mostrarXML(xmlData);
    document.getElementById("guardarXML").disabled = false;
  })
  .catch(error => console.error("Error al obtener los datos:", error));
}

function convertirJSONaXML(json) {
  let xmlDocument = document.implementation.createDocument("", "", null);
  let root = xmlDocument.createElement("usuarios");

  json.forEach(usuario => {
    let userNode = xmlDocument.createElement("usuario");

    let id = xmlDocument.createElement("id");
    id.textContent = usuario.id;
    userNode.appendChild(id);

    let nombre = xmlDocument.createElement("nombre");
    nombre.textContent = `${usuario.name.firstname} ${usuario.name.lastname}`;
    userNode.appendChild(nombre);

    let email = xmlDocument.createElement("email");
    email.textContent = usuario.email;
    userNode.appendChild(email);

    let username = xmlDocument.createElement("username");
    username.textContent = usuario.username;
    userNode.appendChild(username);

    let telefono = xmlDocument.createElement("telefono");
    telefono.textContent = usuario.phone;
    userNode.appendChild(telefono);

    let direccion = xmlDocument.createElement("direccion");
    direccion.textContent = `${usuario.address.street} ${usuario.address.number},
    ${usuario.address.city} (${usuario.address.zipcode})`;
    userNode.appendChild(direccion);

    root.appendChild(userNode);
  });

  xmlDocument.appendChild(root);

  let serializer = new XMLSerializer();
  return serializer.toString(xmlDocument);
}

function mostrarXML(xmlString) {
  const contenedor = document.getElementById("resultado");
  contenedor.textContent = xmlString;
}
```

```
function guardarXML() {
  const blob = new Blob([xmlActual], { type: "text/xml" });
  const enlace = document.createElement("a");
  enlace.href = URL.createObjectURL(blob);
  enlace.download = "usuarios.xml";
  enlace.click();

  // Borrar los datos de la pantalla después de guardar
  document.getElementById("resultado").textContent = "";
  document.getElementById("guardarXML").disabled = true;
}

function cargarXML(event) {
  const file = event.target.files[0];
  if (!file) return;

  const reader = new FileReader();

  // La función anónima recibe un evento e,
  // que contiene el resultado de la lectura
  // en e.target.result.

  reader.onload = function(e) {
    // Asegurar que el contenido previo se borra antes de cargar el nuevo XML
    document.getElementById("resultado").textContent = "";

    // Mostrar el contenido XML en pantalla
    document.getElementById("resultado").textContent = e.target.result;

    // Guardar el XML cargado en la variable para permitir la descarga
    xmlActual = e.target.result;

    // Habilitar el botón de guardar
    document.getElementById("guardarXML").disabled = false;
  };
  reader.readAsText(file);
}
```

styles.css

```
body {
  font-family: Arial, sans-serif;
  text-align: center;
}

pre {
  background: #f4f4f4;
  padding: 10px;
  border-radius: 5px;
  text-align: left;
  white-space: pre-wrap;
}
```

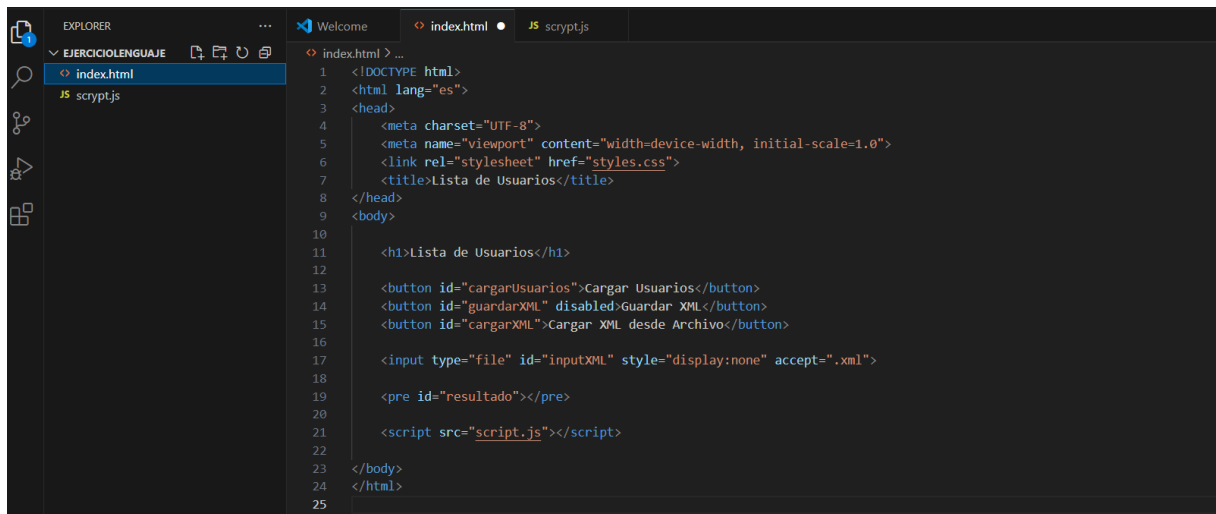
```
button {  
  margin: 10px;  
  padding: 10px;  
  font-size: 16px;  
  cursor: pointer;  
}  
  
button:disabled {  
  background: #ccc;  
  cursor: not-allowed;  
}
```

La API utilizada, mediante la instrucción `fetch("https://fakestoreapi.in/api/users")`, nos devolverá la siguiente cadena de texto siguiendo el formato JSON (han obviado 18 usuarios por abreviar):

```
{
  "status": "SUCCESS",
  "message": "Here you go! You've received 20 users.",
  "users": [
    {
      "id": 1,
      "email": "michael@simpson.com",
      "username": "michaelsimpson",
      "password": "@K(5UejhL&",
      "name": {
        "firstname": "Michael",
        "lastname": "Simpson"
      },
      "address": {
        "city": "Joelton",
        "street": "Angela Spring",
        "number": "868",
        "zipcode": "75070",
        "geolocation": {
          "lat": 19.7091875,
          "long": -14.782061
        }
      },
      "phone": "562.548.9768x73853"
    },
    // {...}
    {
      "id": 20,
      "email": "timothy@burton.com",
      "username": "timothyburton",
      "password": "&$)QeGpZ25",
      "name": {
        "firstname": "Timothy",
        "lastname": "Burton"
      },
      "address": {
        "geolocation": {...}
      },
      "phone": "+1-293-912-5353x125"
    }
  ]
}
```

Muestra mediante una captura de pantalla el contenido de cada uno de los documentos (html, JavaScript y CSS) y **otra captura de pantalla** que muestre el archivo descargado, generado desde la aplicación web.

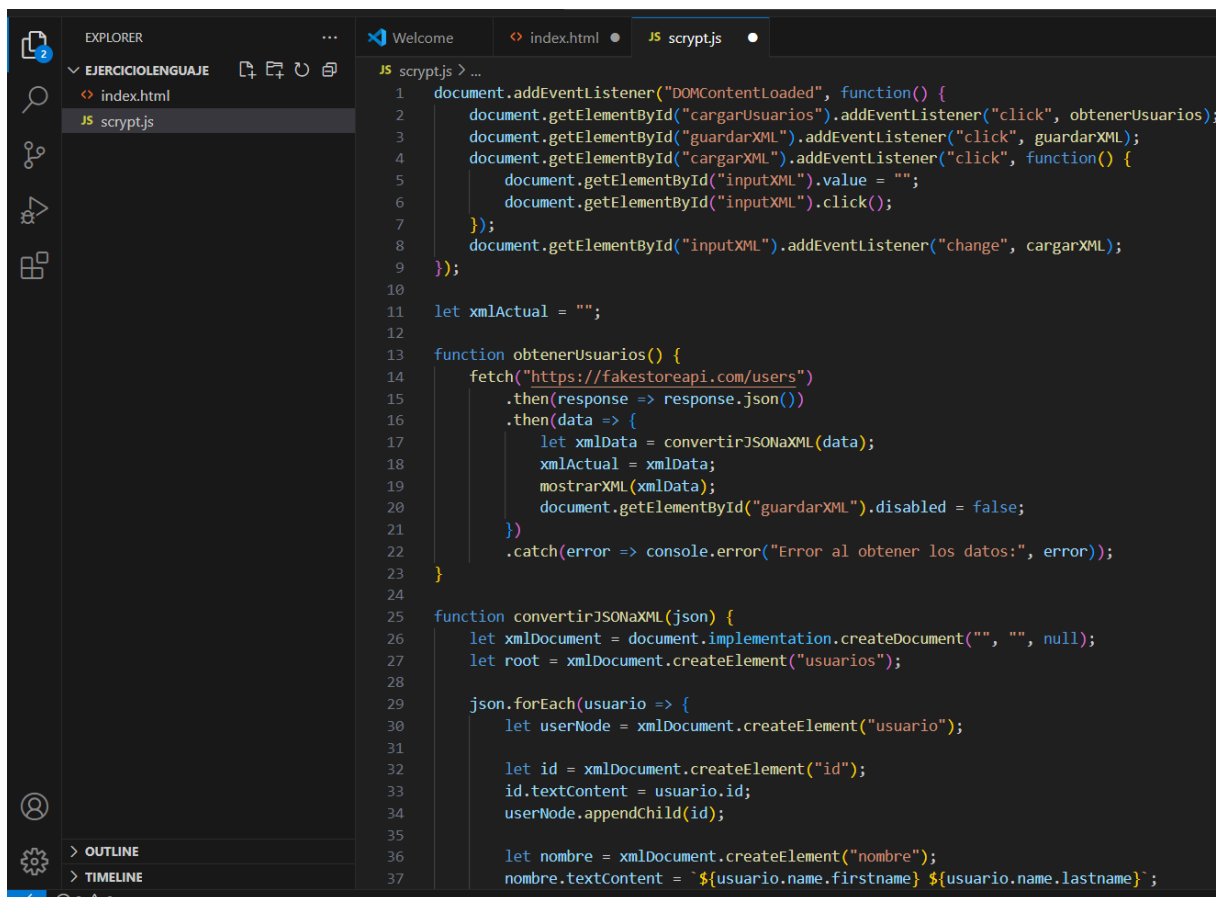
Captura contenido html



The screenshot shows the VS Code editor with the Explorer sidebar on the left. The Explorer sidebar shows a folder named 'EJERCICIO LENGUAJE' containing two files: 'index.html' and 'script.js'. The 'index.html' file is selected, and its content is displayed in the main editor area. The HTML content is as follows:

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="styles.css">
7   <title>Lista de Usuarios</title>
8 </head>
9 <body>
10
11   <h1>Lista de Usuarios</h1>
12
13   <button id="cargarUsuarios">Cargar Usuarios</button>
14   <button id="guardarXML" disabled="true">Guardar XML</button>
15   <button id="cargarXML">Cargar XML desde Archivo</button>
16
17   <input type="file" id="inputXML" style="display:none" accept=".xml">
18
19   <pre id="resultado"></pre>
20
21   <script src="script.js"></script>
22
23 </body>
24 </html>
25
```

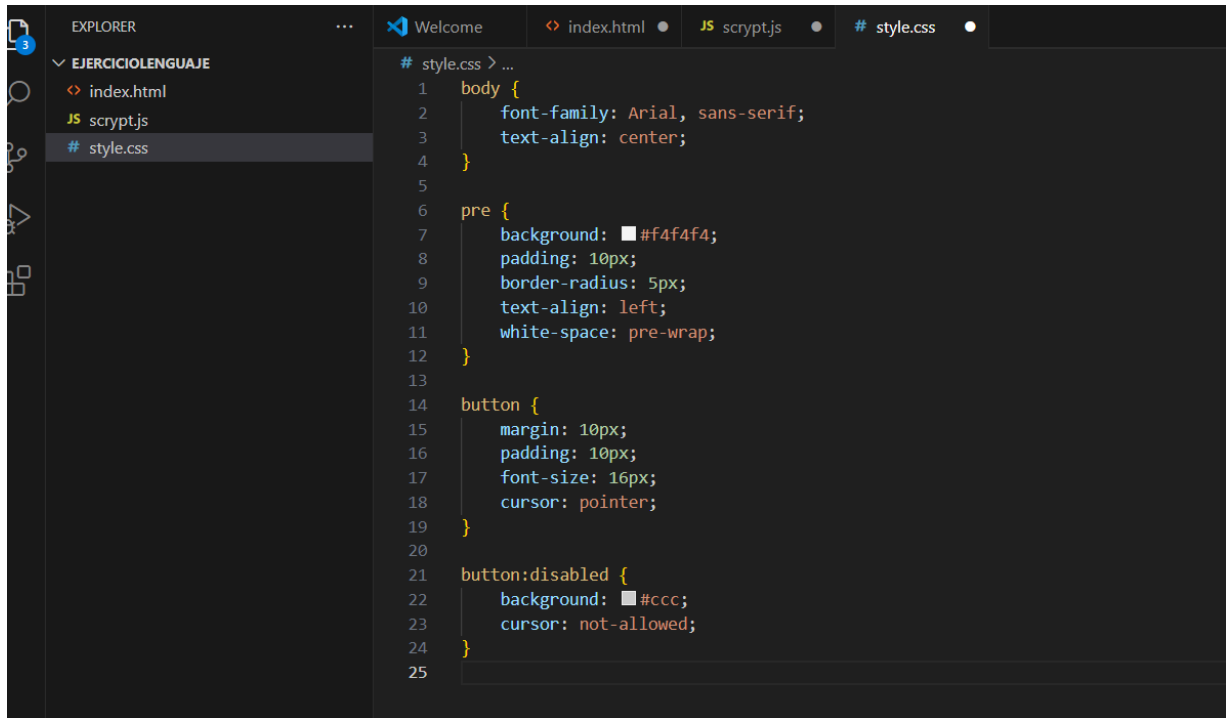
Captura contenido js



The screenshot shows the VS Code editor with the Explorer sidebar on the left. The Explorer sidebar shows a folder named 'EJERCICIO LENGUAJE' containing two files: 'index.html' and 'script.js'. The 'script.js' file is selected, and its content is displayed in the main editor area. The JavaScript content is as follows:

```
1 document.addEventListener("DOMContentLoaded", function() {
2   document.getElementById("cargarUsuarios").addEventListener("click", obtenerUsuarios);
3   document.getElementById("guardarXML").addEventListener("click", guardarXML);
4   document.getElementById("cargarXML").addEventListener("click", function() {
5     document.getElementById("inputXML").value = "";
6     document.getElementById("inputXML").click();
7   });
8   document.getElementById("inputXML").addEventListener("change", cargarXML);
9 });
10
11 let xmlActual = "";
12
13 function obtenerUsuarios() {
14   fetch("https://fakestoreapi.com/users")
15     .then(response => response.json())
16     .then(data => {
17       let xmlData = convertirJSONaXML(data);
18       xmlActual = xmlData;
19       mostrarXML(xmlData);
20       document.getElementById("guardarXML").disabled = false;
21     })
22     .catch(error => console.error("Error al obtener los datos:", error));
23 }
24
25 function convertirJSONaXML(json) {
26   let xmlDocument = document.implementation.createDocument("", "", null);
27   let root = xmlDocument.createElement("usuarios");
28
29   json.forEach(usuario => {
30     let userNode = xmlDocument.createElement("usuario");
31
32     let id = xmlDocument.createElement("id");
33     id.textContent = usuario.id;
34     userNode.appendChild(id);
35
36     let nombre = xmlDocument.createElement("nombre");
37     nombre.textContent = `${usuario.name.firstname} ${usuario.name.lastname}`;
```

Captura contenido css



```
# style.css > ...
1  body {
2      font-family: Arial, sans-serif;
3      text-align: center;
4  }
5
6  pre {
7      background: #f4f4f4;
8      padding: 10px;
9      border-radius: 5px;
10     text-align: left;
11     white-space: pre-wrap;
12 }
13
14 button {
15     margin: 10px;
16     padding: 10px;
17     font-size: 16px;
18     cursor: pointer;
19 }
20
21 button:disabled {
22     background: #ccc;
23     cursor: not-allowed;
24 }
25
```

Contenido de la web en XML
(no me descarga el archivo)

Ejercicio 2

Ahora incorpora a tu aplicación web las siguientes dos funcionalidades:

- Descarga del archivo conteniendo el documento XML en el navegador web
- Carga del archivo que contiene un documento XML, en el navegador web

tal y como acabas de hacer en el ejercicio 1, pero en este caso trabajando contra la API que en la actividad anterior ya elegiste de entre las siguientes:

1. Fake Store API

Descripción: Simula productos de una tienda online con imágenes, precios y categorías.

URL: <https://fakestoreapi.in/docs>

Ejemplo de endpoint:

- Productos: <https://fakestoreapi.in/api/products/2>
- Categorías: <https://fakestoreapi.in/api/products/category>

2. The Dog API

Descripción: Devuelve imágenes de perros aleatorios por raza.

URL: <https://dog.ceo/dog-api/>

Ejemplo de endpoint:

- Imagen de perro aleatoria: <https://dog.ceo/api/breeds/image/random>

3. TheCatAPI

Descripción: Devuelve imágenes aleatorias de gatos.

URL: <https://thecatapi.com/>

Ejemplo de endpoint:

- Imagen de gato aleatoria: <https://api.thecatapi.com/v1/images/search>

4. DummyJSON

Descripción: API con datos ficticios de productos, usuarios, posts y más.

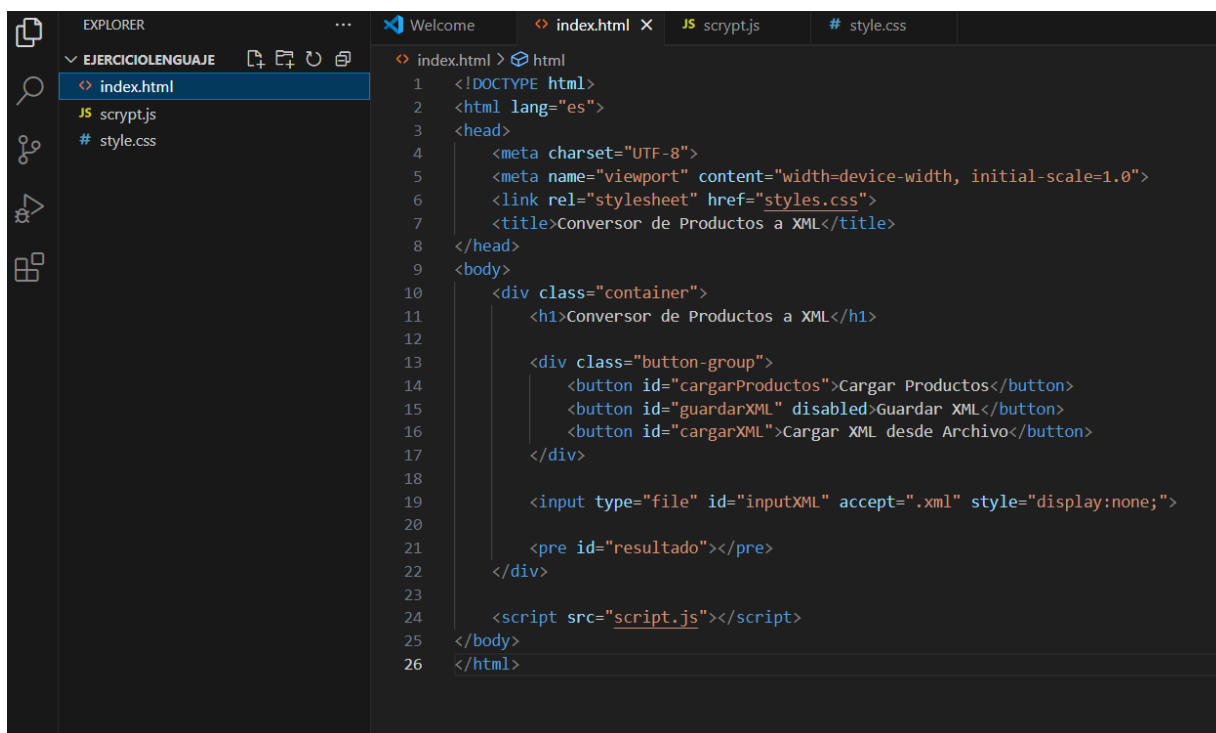
URL: <https://dummyjson.com/>

Ejemplo de endpoint:

- Productos: <https://dummyjson.com/products>
- Usuarios: <https://dummyjson.com/users>

Muestra mediante una captura de pantalla el contenido de cada uno de los documentos (html, JavaScript y CSS) y otra captura de pantalla que muestre el archivo descargado, generado desde la aplicación web.

Captura HTML



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'EJERCICIO LENGUAJE' with files 'index.html', 'script.js', and 'style.css'. The code editor shows the content of 'index.html'.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="styles.css">
7   <title>Conversor de Productos a XML</title>
8 </head>
9 <body>
10  <div class="container">
11    <h1>Conversor de Productos a XML</h1>
12
13    <div class="button-group">
14      <button id="cargarProductos">Cargar Productos</button>
15      <button id="guardarXML" disabled>Guardar XML</button>
16      <button id="cargarXML">Cargar XML desde Archivo</button>
17    </div>
18
19    <input type="file" id="inputXML" accept=".xml" style="display:none;">
20
21    <pre id="resultado"></pre>
22  </div>
23
24  <script src="script.js"></script>
25 </body>
26 </html>
```

CAPTURA JS

```

1 document.addEventListener("DOMContentLoaded", function() {
2   const cargarProductosBtn = document.getElementById("cargarProductos");
3   const guardarXMLBtn = document.getElementById("guardarXML");
4   const cargarXMLBtn = document.getElementById("cargarXML");
5   const inputXML = document.getElementById("inputXML");
6   const resultadoContainer = document.getElementById("resultado");
7
8   let xmlActual = ""; // Variable para almacenar el XML generado
9
10  cargarProductosBtn.addEventListener("click", obtenerProductos);
11  guardarXMLBtn.addEventListener("click", guardarXML);
12
13  cargarXMLBtn.addEventListener("click", function() {
14    inputXML.value = ""; // Resetea el input para permitir cargar otro archivo
15    inputXML.click();
16  });
17
18  inputXML.addEventListener("change", cargarXML);
19
20  function obtenerProductos() {
21    fetch("https://fakestoreapi.com/products")
22      .then(response => response.json())
23      .then(data => {
24        let xmlData = convertirJSONaXML(data);
25        xmlActual = xmlData;
26        mostrarXML(xmlData);
27        guardarXMLBtn.disabled = false;
28      })
29      .catch(error => {
30        console.error("Error al obtener los productos:", error);
31        resultadoContainer.textContent = "Error al cargar productos. Verifique su conexión.";
32      });
33  }
34
35  function convertirJSONaXML(json) {
36    let xmlDocument = document.implementation.createDocument("", "", null);
37    let root = xmlDocument.createElement("productos");
  
```

CAPTURA CSS

```

1 button:hover {
2   background-color: #45a049;
3 }
4
5 button:disabled {
6   background-color: #cccccc;
7   cursor: not-allowed;
8 }
9
10 pre {
11   background-color: #f1f1f1;
12   border: 1px solid #ddd;
13   border-radius: 5px;
14   padding: 15px;
15   text-align: left;
16   white-space: pre-wrap;
17   word-wrap: break-word;
18   max-height: 400px;
19   overflow-y: auto;
20 }
  
```

Al igual que anteriormente no me descarga el xml

Ejercicio 3

Ahora incorpora a tu aplicación web las siguientes dos funcionalidades:

- a) Descarga del archivo conteniendo el documento XML en el navegador web
- b) Carga del archivo que contiene un documento XML, en el navegador web

tal y como acabas de hacer en el ejercicio 1, pero en este caso trabajando contra la API que en la actividad anterior ya elegiste de entre las siguientes:

5. Reqres (Usuarios y autenticación)

Descripción: Simula usuarios y respuestas de autenticación.

URL: <https://reqres.in/>

Ejemplo de endpoint:

- Lista de usuarios: <https://reqres.in/api/users?page=2>
- Un recurso: <https://reqres.in/api/unknown/2>

6. Rick and Morty API

Descripción: Datos de personajes, episodios y ubicaciones de la serie.

URL: <https://rickandmortyapi.com/>

Ejemplo de endpoint:

- Todos los personajes: <https://rickandmortyapi.com/api/character>

7. PokéAPI (Datos de Pokémon)

Descripción: Devuelve información de Pokémon como estadísticas y habilidades.

URL: <https://pokeapi.co/>

Ejemplo de endpoint:

- <https://pokeapi.co/api/v2/pokemon/ditto> (Datos del Pokémon Ditto)

8. OpenWeatherMap (Clima)

Descripción: Datos del clima en tiempo real (requiere API Key gratuita).

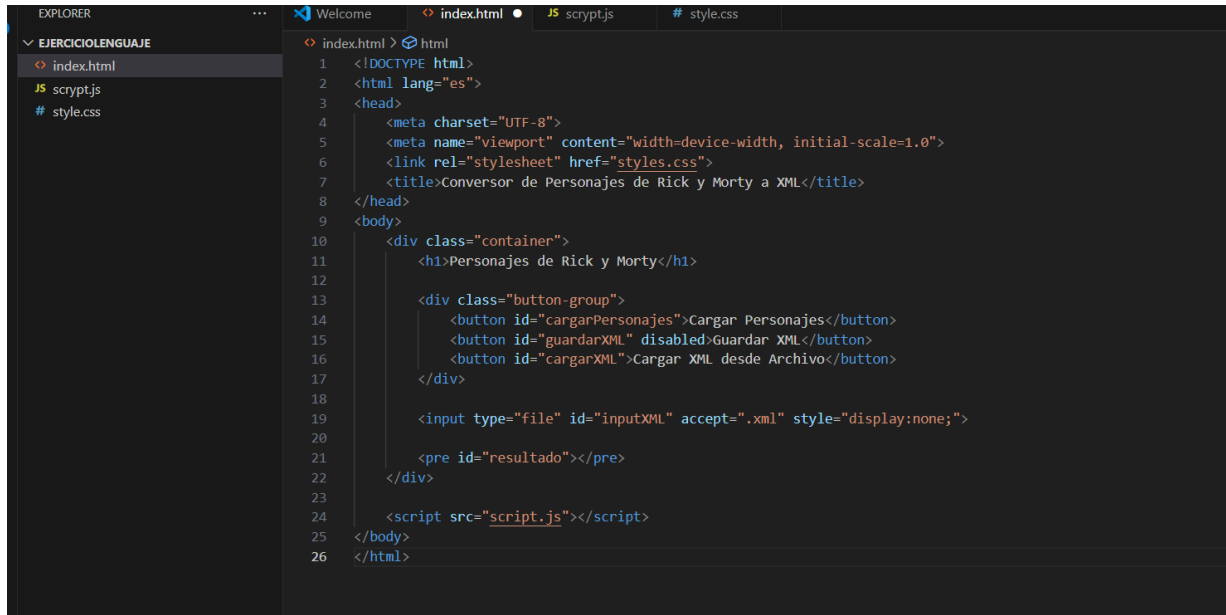
URL: <https://openweathermap.org/api>

Ejemplo de endpoint:

- https://api.openweathermap.org/data/2.5/weather?q=Madrid&appid=TU_API_KEY

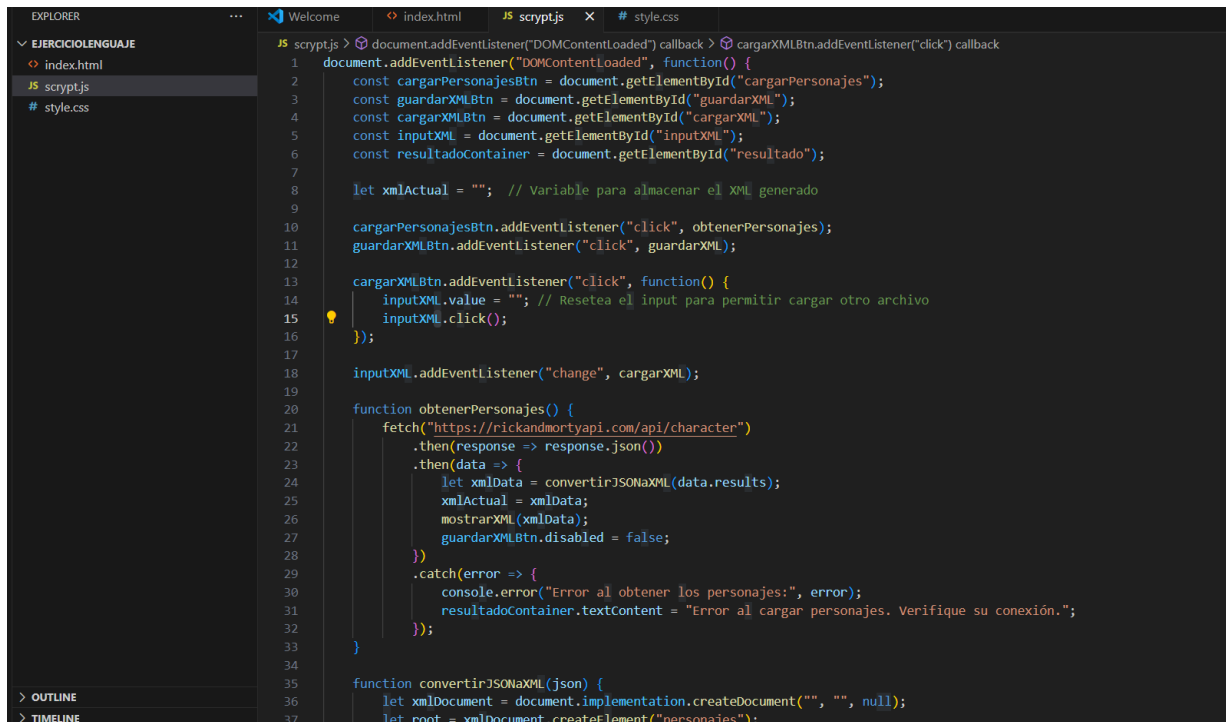
Muestra mediante una captura de pantalla el contenido de cada uno de los documentos (html, JavaScript y CSS) y otra captura de pantalla que muestre el archivo descargado, generado desde la aplicación web.

CAPTURA HTML



```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="styles.css">
7   <title>Conversor de Personajes de Rick y Morty a XML</title>
8 </head>
9 <body>
10   <div class="container">
11     <h1>Personajes de Rick y Morty</h1>
12
13     <div class="button-group">
14       <button id="cargarPersonajes">Cargar Personajes</button>
15       <button id="guardarXML" disabled="true">Guardar XML</button>
16       <button id="cargarXML">Cargar XML desde Archivo</button>
17     </div>
18
19     <input type="file" id="inputXML" accept=".xml" style="display:none;">
20
21     <pre id="resultado"></pre>
22   </div>
23
24   <script src="script.js"></script>
25 </body>
26 </html>
```

CAPTURA JS



```
1 document.addEventListener("DOMContentLoaded", function() {
2   const cargarPersonajesBtn = document.getElementById("cargarPersonajes");
3   const guardarXMLBtn = document.getElementById("guardarXML");
4   const cargarXMLBtn = document.getElementById("cargarXML");
5   const inputXML = document.getElementById("inputXML");
6   const resultadoContainer = document.getElementById("resultado");
7
8   let xmlActual = ""; // Variable para almacenar el XML generado
9
10  cargarPersonajesBtn.addEventListener("click", obtenerPersonajes);
11  guardarXMLBtn.addEventListener("click", guardarXML);
12
13  cargarXMLBtn.addEventListener("click", function() {
14    inputXML.value = ""; // Resetea el input para permitir cargar otro archivo
15    inputXML.click();
16  });
17
18  inputXML.addEventListener("change", cargarXML);
19
20  function obtenerPersonajes() {
21    fetch("https://rickandmortyapi.com/api/character")
22      .then(response => response.json())
23      .then(data => {
24        let xmlData = convertirJSONaXML(data.results);
25        xmlActual = xmlData;
26        mostrarXML(xmlData);
27        guardarXMLBtn.disabled = false;
28      })
29      .catch(error => {
30        console.error("Error al obtener los personajes:", error);
31        resultadoContainer.textContent = "Error al cargar personajes. Verifique su conexión.";
32      });
33  }
34
35  function convertirJSONaXML(json) {
36    let xmlDocument = document.implementation.createDocument("", "", null);
37    let root = xmlDocument.createElement("personajes");
```

CAPTURA CSS

```
EXPLORER
EJERCICIO LENGUAJE
  index.html
  script.js
  style.css

# style.css
1 body {
2   font-family: Arial, sans-serif;
3   background-color: #d1d1d1;
4   margin: 0;
5   padding: 20px;
6   display: flex;
7   justify-content: center;
8   align-items: center;
9   min-height: 100vh;
10  color: #e0e0e0;
11 }
12
13 .container {
14   background-color: #2c2c2c;
15   border-radius: 15px;
16   box-shadow: 0 8px 16px rgba(0, 0, 0, 0.3);
17   padding: 30px;
18   width: 90%;
19   max-width: 800px;
20   text-align: center;
21 }
22
23 h1 {
24   color: #4caf50;
25   margin-bottom: 20px;
26   font-size: 2em;
27   text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
28 }
29
30 .button-group {
31   display: flex;
32   justify-content: center;
33   gap: 15px;
34   margin-bottom: 20px;
35 }
```

CAPTURA DESCARGA

(lo mismo, no me carga se queda asi)

Personajes de Rick y Morty

Cargar Personajes

Guardar XML

Cargar XML desde Archivo