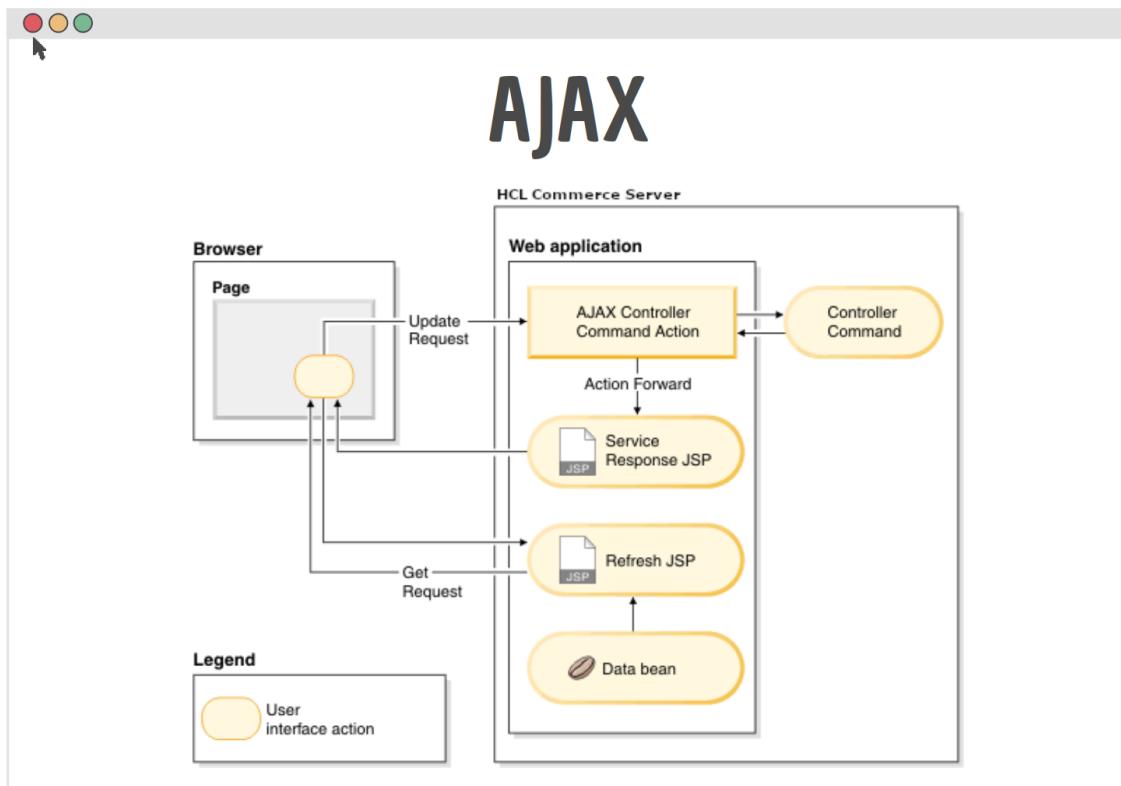




PROGRAMACIÓN WEB 2



Profesor(a):
Carlo Jose Luis Corrales Delgado

Estudiante:
Mamani Anahua, Victor Narciso
Mamani Huarsaya, Jorge Luis
Quispe Marca Edysson Darwin
Velarde Saldaña Jhossep Fabritzio
Zuñiga Villacorta Peter Sebastian

Repositorios GitHub:
<https://github.com/jorghee/tareaAjax>
<https://github.com/jorghee/ajax-markdown>

Peticiones AJAX a un servidor node.js

Creando el directorio que contiene los archivos Markdwon

Empezamos copiando archivos de ejemplo .md al directorio files_markdown. Este directorio es el que responde a las peticiones AJAX.

```

1 $ ls --tree
2 files_markdown
3     calculator_with_java.md
4     example_readme_github.md
5     learning_github.md
6     project_ajax_regions.md
7     server_with_nodejs.md

```

Creando el servidor con Express

En este ocasión vamos a utilizar todos los siguientes paquetes para para poder resolver direcciones a archivos y directorios, escribir y leer archivos en directorios y lo más importante es el uso del paquete **markdown-it** que se encarga de hacer la conversion de markdown a html.

```

1 const path = require("path");
2 const fs = require("fs");
3 const MarkdownIt = require("markdown-it");
4 const bp = require("body-parser");
5 const express = require("express");
6 const app = express();
7 const md = new MarkdownIt();
8 const MARKDOWN_DIR = path.resolve(__dirname, "files_markdown");
9
10 app.use(bp.json());
11 app.use(express.static("public"));

```

Como se observa, hemos declarado una variable estatica **MARDOWN_DIR** que almacena la dirección completa al directorio con el cual vamos a trabajar. Además, se ha configurado para que el servidor sirva archivos estaticos desde el directorio **public** con el objetivo de organizar mejor nuestro proyecto, ya que aqui se encontraran los archivos html, css y los scripts.

Ahora podemos servir el archivo **index.html** para que nuestro proyecto pueda visualizarse.

```

1 /**
 * Creamos el servidor, request maneja las solicitudes que hacemos y
 * response es el objeto que maneja las respuestas que envia el servidor.
 */
5 app.get('/', (request, response) => {
6     response.sendFile(path.resolve(__dirname, "public/index.html"));
7 });

```

Implemetando la función que lista los archivos Markdown

Lo primero que hacemos es configurar el servidor para que pueda manejar la petición y enviar una respuesta al navegador. Como ahora solo necesitamos leer, vamos a usar el paquete **fs** de node.js que nos otorga funciones para poder interactuar con el sistema de archivos del sistema operativo donde se encuentra levantado el servidor.

Podemos hacer que el servidor responda en formato **JSON** como es común hoy en día.

```

1 // Listar archivos Markdown
2 app.get("/files", (request, response) => {
3   fs.readdir(MARKDOWN\_DIR, (error, files) => {
4     if (error) {
5       console.error("No se pudo leer los archivos:", error);
6       return;
7     }
8     response.json(files);
9   });
10 });

```

Solicitud AJAX

Ahora que ya hemos configurado el servidor, podemos empezar a construir nuestra petición AJAX, en este caso usaremos la función `fetch()` para poder manejar las promesas.

La solicitud la hacemos a la dirección en la cual ha sido configurada el servidor. Una vez que hecha la petición, el servidor envía los nombres de los archivos .md y con la función `then()` manejamos esta respuesta.

```

1 function listMarkdownFiles() {
2   fetch("/files")
3     .then(response => response.json())
4     .then(files => {
5       let fileList = "";
6       files.forEach(file => fileList += `<li>${file}</li>`);
7       document.getElementById("files").innerHTML = fileList;
8     })
9     .catch(error => console.error("Error al listar los archivos:", error));
10 }
11
12 export { listMarkdownFiles };

```

Entonces comenzamos iterando por cada valor, que en este caso son los nombres de los archivos, y generamos código html que se encargue de mostrar estos nombres como una lista. La lista que hemos generado con la etiqueta `li` la ponemos en el elemento de ID `files`, concluyendo así la petición ajax.

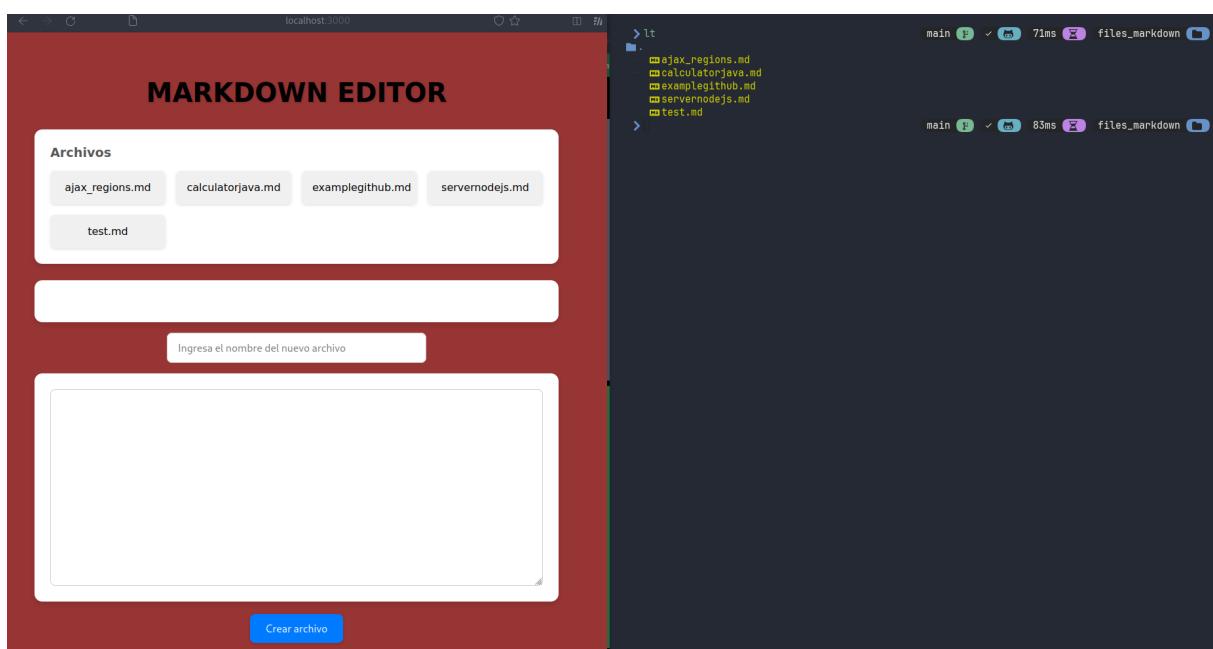


Figure 1: Listado de los archivos markdown

Implementando la función que visualiza un archivo Markdown

Del mismo modo, nosotros tenemos que implementar la lógica para decirle al servidor cómo debe manejar una solicitud. En este excepcional caso, nos daremos cuenta que necesitamos enviar un identificador que le permita al servidor identificar qué archivo se desea visualizar. Por lo tanto, la configuración debe recibir un parametro.

```

1 app.get('/files/:fileName', (req, res) => {
2   const filename = req.params.fileName;
3   const filePath = path.resolve(MARKDOWN\_DIR, filename);
4
5   fs.readFile(filePath, 'utf8', (err, data) => {
6     if (err) {
7       return res.status(500).json({ error: "Unable to read file" });
8     }
9     const htmlContent = md.render(data);
10    res.json({html: htmlContent});
11  });
12});
```

Como se observa, el servidor recibe un parametro, dicho parametro es el nombre de archivo que se desea ver. Aquí lo complicado fue pensar cómo recuperar el nombre y enviarlo al servidor. Despues veremos que agregando un evento a cada etiqueta `li` podemos enviar este identificador.

Con el identificador ya recibido, podemos buscar el archivo en el correspondiente directorio y empezar a leerlo. Sin embargo, lo que hemos leido solo es texto con la extension .md, entonces, aquí usamos el paquete `markdown-it` que se encarga de hacer la conversion y que pueda ser visualizado por el navegador.

Realizando la petición AJAX

Con el servidor configurado, podemos empezar creando la función que se encarga de hacer la petición. La función entonces envia el identificador al servidor y este responde con un texto formateado a html. Basicamente ese el funcionamiento de esta función la cual modifica la etiqueta `div` de ID `content` en la cual muestra el texto html.

```

1 export { showMarkdownFile };
2
3 function showMarkdownFile(fileName) {
4   fetch(`/files/${fileName}`)
5     .then(response => {
6       if (!response.ok) {
7         throw new Error(`HTTP error! status: ${response.status}`);
8       }
9       return response.json();
10    })
11    .then(htmlContent => {
12      document.getElementById("content").innerHTML = htmlContent.html;
13    })
14    .catch(error => console.error("Error al mostrar el archivo:", error));
15  }
16
17 export { showMarkdownFile };
```

localhost:3000

MARKDOWN EDITOR

Archivos

- ajax_regions.md
- calculatorjava.md
- examplegithub.md
- servernodejs.md
- test.md**

Fundamentos de Git

Git es un sistema de control de versiones que te permite como desarrollador rastrear y gestionar cambios en tu proyecto de software.
 No uses Git con interfaces graficas de usuario (GUI), usalo solo por consola (CLI) - Paz Valderrama

Los tres estados de Git

Committed (confirmado)
 Significa que los estados estan almacenados de manera segura en nuestra base de datos local.

Modified (modificado)
 Significa que hemos modificado el archivo pero todavia no lo hemos confirmado en nuestra base de datos.

Staged (preparado)
 Significa que hemos marcado un archivo modificado en su version actual para que vaya en nuestra proxima confirmacion (commit).

Las tres secciones principales

Working Directory (Directorio de trabajo)
 Es una copia de una version del proyecto. **Estos archivos, se sacan de la base de datos comprimida en el Git Directory, y se colocan en disco para que lo podamos usar y modificar.**

Staging Area (Area de preparacion)
 Es simplemente un archivo, generalmente contenido en el Git Directory, que **almacena informacion acerca de lo que va a ir en nuestra proxima confirmacion.**

Git directory (Directorio de Git)
 Almacena los metadatos y la base de datos de objetos para nuestro proyecto. Es la parte mas importante de Git, y es lo que se copia cuando clonamos un repositorio desde otro ordenador.

Relacion entre los tres estados y secciones de Git
 Si una version concreta de un archivo está en el Git Directory, se considera confirmada

Figure 2: Vista de un archivo especifico

Implementando la función que crea y almacena un archivo Markdown en el servidor

Finalmente necesitamos guardar en el servidor los archivos creados en la pagina web. Por lo tanto la funcion especifica del paquete **fs** que usaremos es **writeFile()**.

El servidor necesita recibir el nombre del archivo y el contenido en si del archivo, por ello recuperamos los valores enviamos en el cuerpo de la solicitud y los usamos en el metodo mencionado del paquete **fs**

```

1 // Crear un archivo
2 app.post("/files", (request, response) => {
3     // Recuperamos los valores
4     const { filename, content } = request.body;
5     console.log(filename, content);
6

```

```

7  const filePath = path.resolve(MARKDOWN_DIR, filename);
8  fs.writeFile(filePath, content, error => {
9      if (error) {
10          console.log(error);
11          response.json({ error: "No se pudo crear el archivo" });
12      }
13
14      response.json({ message: "Archivo creado existosamente" });
15  });
16});
```

En este caso el servidor no devuelve nada específico que se deba usar en el cliente, solo necesitamos saber si el almacenamiento del archivo fue existoso.

Realizando la petición AJAX

Para poder crear y elegir un nombre para guardar el archivo, hemos creado 2 elementos: **input** y **textarea**, entonces necesitamos obtener los valores de estos elementos. Una vez recogidos los valores, lo enviamos en el cuerpo de la solicitud ajax. Basicamente en eso consiste la solicitud, finalizando así las tres funcionalidades de esta pagina web.

```

1 import { listMarkdownFiles } from "./listMarkdown.js";
2
3 function createMarkdownFile() {
4     const filename = document.getElementById("newFileName").value;
5     const content = document.getElementById("newFileContent").value;
6     console.log(filename, content);
7
8     // Generamos los parametros de la solicitud
9     const request = {
10         method: "POST",
11         headers: {
12             "Content-Type": "application/json",
13         },
14
15         // Pasamos el objeto con los valores de filename y content
16         body: JSON.stringify({ filename, content }),
17     };
18
19     fetch("/files", request)
20         .then(response => response.json())
21         .then(data => {
22             console.log(data.message);
23             listMarkdownFiles();
24         })
25         .catch(error => console.error("Error al crear el archivo", error));
26     }
27
28     export { createMarkdownFile }
```

localhost:3000

MARKDOWN EDITOR

Archivos

[ajax_regions.md](#) [calculatorjava.md](#) [examplegithub.md](#) [hello.md](#)
[servernodejs.md](#) [test.md](#)

Hello world

hello.md

```
# Hello world
```

[Crear archivo](#)

Figure 3: Creando archivo y almacenando en el servidor

Como se ha observado en todas las funciones de petición AJAX, al final siempre exportábamos dicha función. Esta idea fue con el objetivo de organizar nuestro proyecto y hacer más manejable en cuanto a los errores.

```

1 import { listMarkdownFiles } from "./listMarkdown.js";
2 import { showMarkdownFile } from "./showMarkdown.js";
3 import { createMarkdownFile } from "./createMarkdown.js";
4
5 listMarkdownFiles();
6
7 document.getElementById('files').addEventListener('click', (event) => {
8     if (event.target.tagName === 'LI')
9         showMarkdownFile(event.target.textContent);
10 });
11
12 document.getElementById("createFile").addEventListener("click", createMarkdownFile);

```

Ejercicios de Ajax en w3schools

Capturas de pantalla:

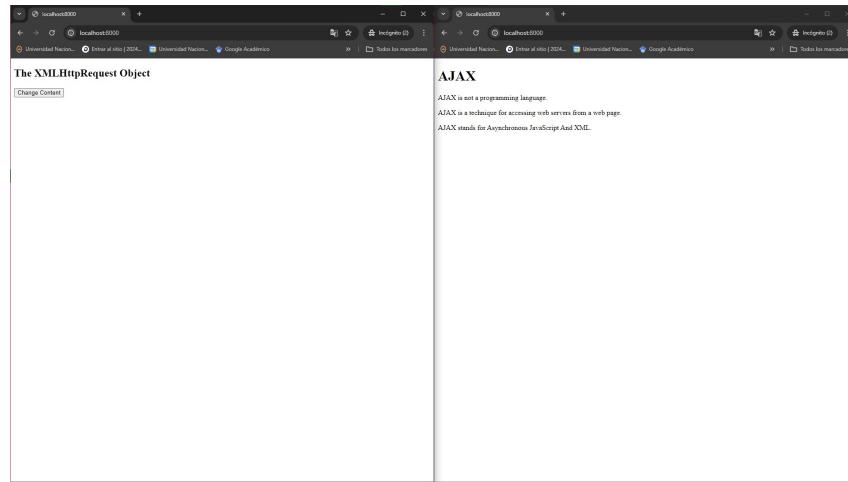


Figure 4: Creando un objeto XMLHttpRequest

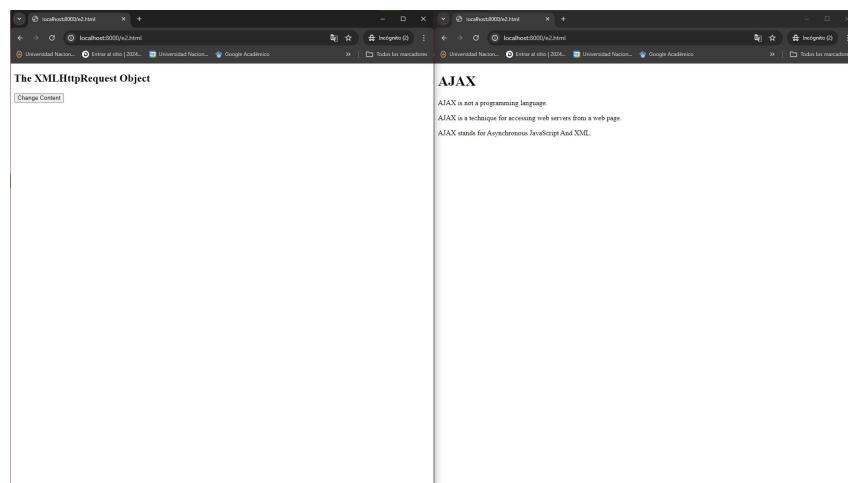


Figure 5: Creando otro objeto XMLHttpRequest

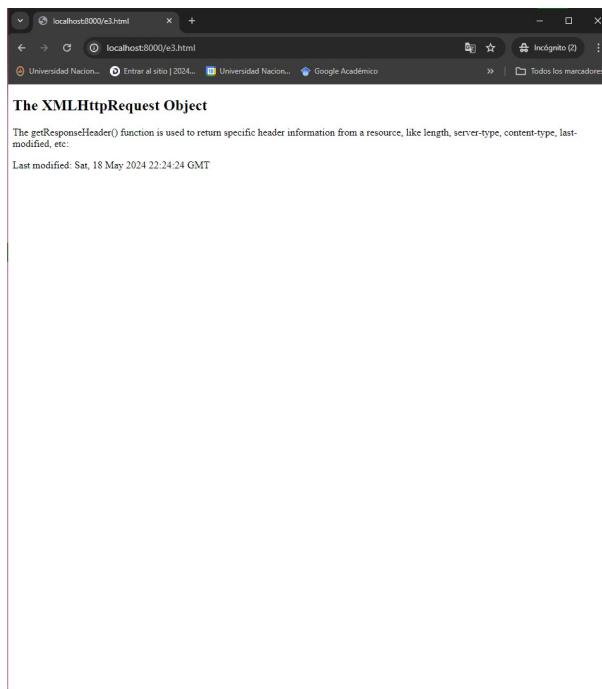


Figure 6: Modificando el objeto XMLHttpRequest

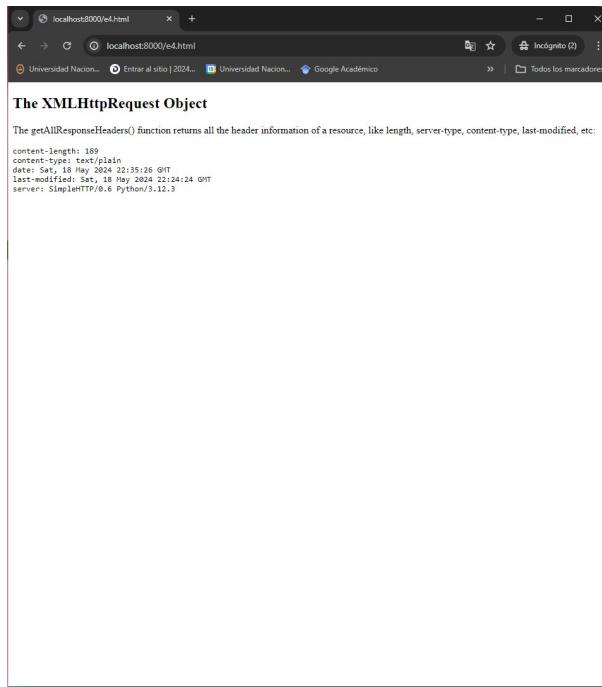


Figure 7: Añadiendo contenido al objeto XMLHttpRequest

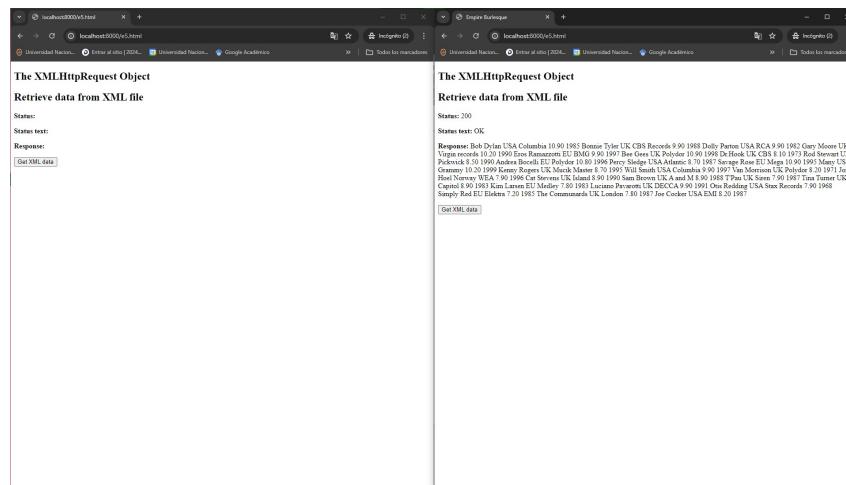


Figure 8: Pidiendo un reporte del status del objeto XMLHttpRequest

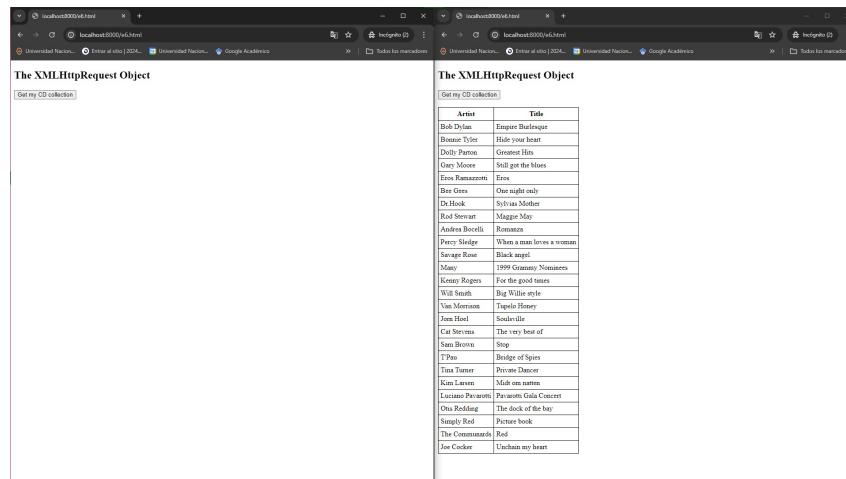


Figure 9: Obtener informacion a travez de Ajax de un archivo XML

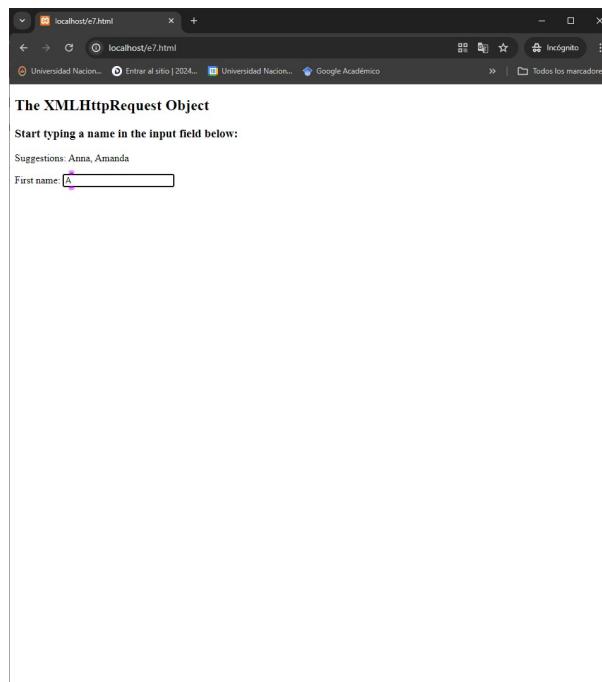


Figure 10: Comunicacion entre un servidor web y una pagina web mientras se ingresan datos

```
<!DOCTYPE html>
<html>
<body>
<script>
function addName(str) {
    if (str.length > 0) {
        document.getElementById("txtHint").innerHTML = "";
        return;
    }
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("txtHint").innerHTML =
                this.responseText;
        }
    };
    xhttp.open("GET", "gethint.asp?q=" + str);
    xhttp.send();
}
</script>
</body>
</html>
```

The XMLHttpRequest Object

Start typing a name in the input field below:

Suggestions: Brittany

First name:

Figure 11: Obtener informacion de una base de datos usando Ajax

```
<!DOCTYPE html>
<html>
<head>
<title>Customer Info</title>
<script>
function showCustomer(str) {
    if (str == "") {
        document.getElementById("txtHint").innerHTML = "";
        return;
    }
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("txtHint").innerHTML =
                this.responseText;
        }
    };
    xhttp.open("GET", "getcustomer.php?q=" + str);
    xhttp.send();
}
</script>
</head>
<body>
<div id="txtHint">Customer Info will be listed here...</div>
</body>
</html>
```

The XMLHttpRequest Object

NorthSouth

CustomerID	NORTS
CompanyName	North South
CustomerName	Simon Crother
Address	South House 300 Queensland
City	London
PostalCode	SW1 1EE
Country	UK

Figure 12: Mostrando datos en una tabla HTML

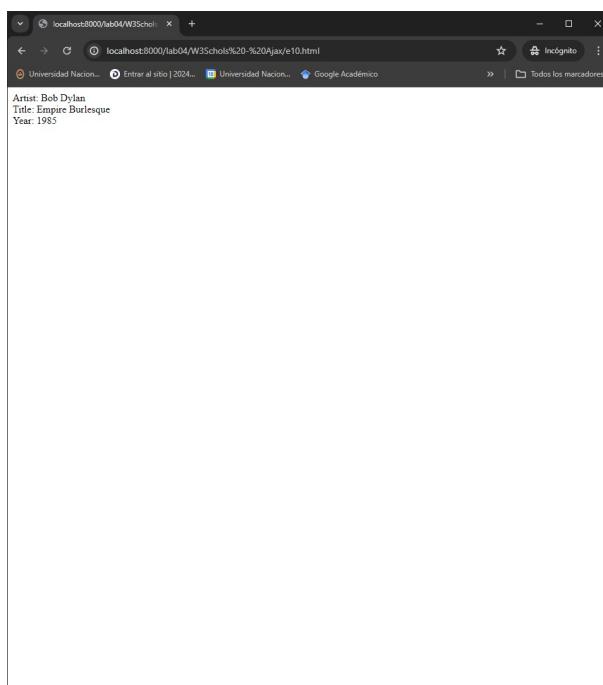


Figure 13: Pidiendo informacion de la base de datos

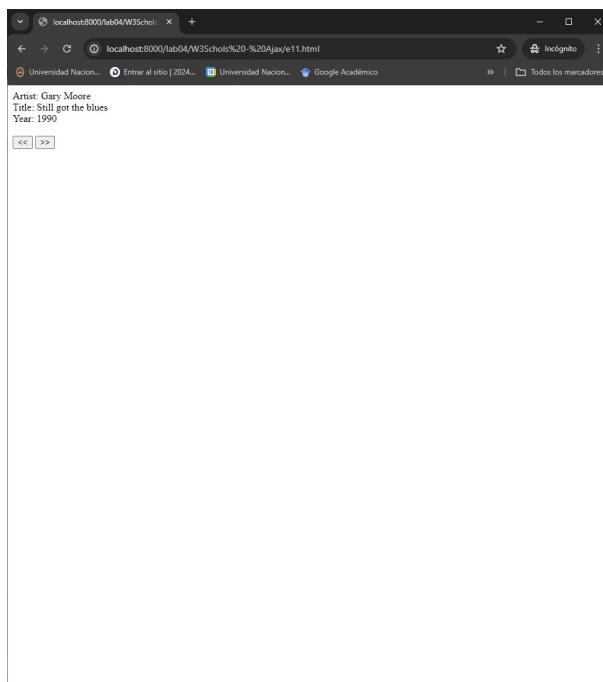


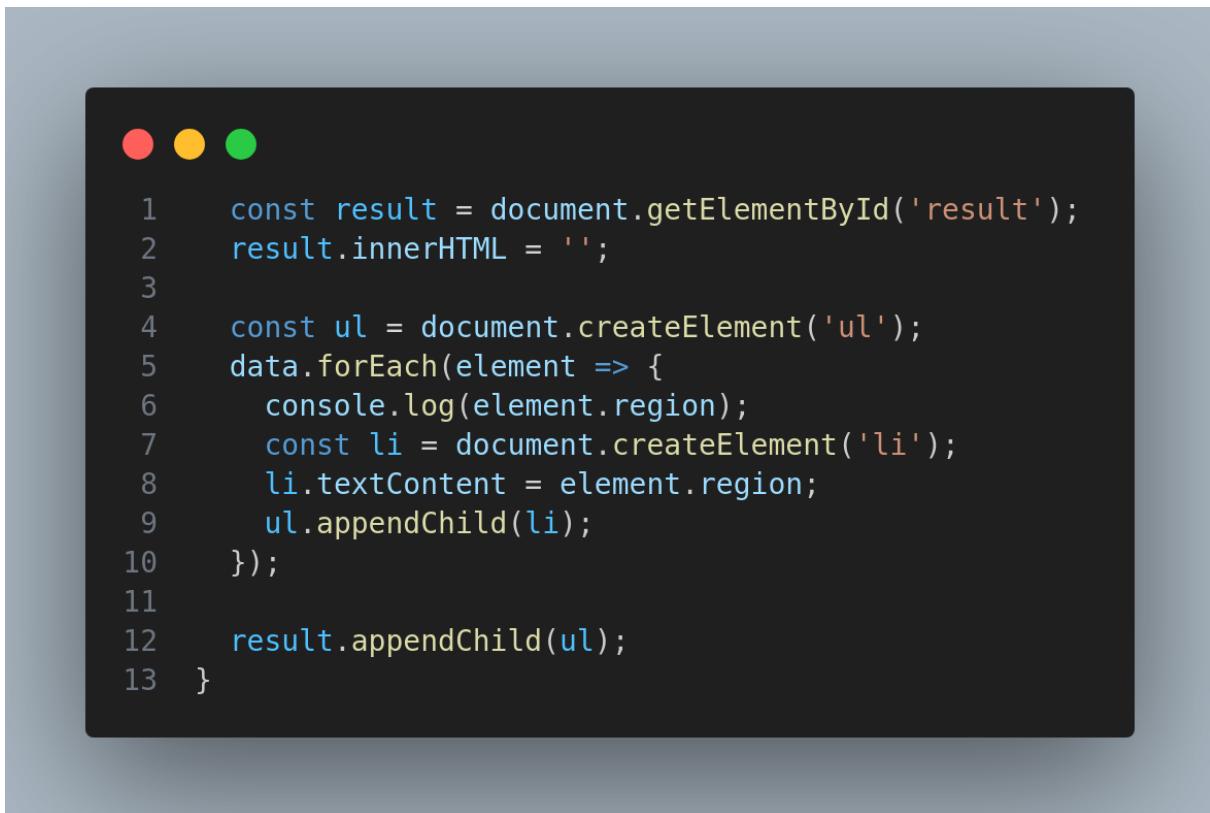
Figure 14: Recorriendo la informacion de la base de datos

Artist	Title
Bob Dylan	Empire Burlesque
Bonnie Tyler	Hide your heart
Dolly Parton	Greatest Hits
Gary Moore	Still got the blues
Eros Ramazzotti	Eros
Bee Gees	One night only
Dr Hook	Sylvia's Mother
Rod Stewart	Maggie May
Andrea Bocelli	Romanza
Percy Sledge	When a man loves a woman
Savage Rose	Black angel
Many	1999 Grammy Nominees
Kenny Rogers	For the good times
Will Smith	Big Willie style
Van Morrison	Tupelo Honey
Joni Mitchell	Soulsville
Cat Stevens	The very best of
Sam Brown	Stop
T'Pau	Bridge of Spies
Tina Turner	Private Dancer
Kim Larsen	Midi om natten
Luciano Pavarotti	Pavarotti Gala Concert
Otis Redding	The dock of the bay
Simply Red	Picture book
The Communards	Red
Joe Cocker	Unchain my heart

Figure 15: Mostrar los elementos en una tabla HTML

Los 8 ejercicios JavaScript con Ajax y Google Chart

En el primer ejercicio tenemos una función de JavaScript llamada `listRegions` esta función recibe un arreglo de datos como parámetro y actualiza el contenido de un elemento HTML identificado por el id 'result'. Primero, borra cualquier contenido previo dentro de este elemento. Luego, itera sobre cada elemento del arreglo de datos. Para cada elemento, imprime el valor de la propiedad 'region' en la consola del navegador y crea un nuevo elemento de lista (li) con este valor como contenido de texto. Finalmente, añade este elemento de lista como hijo del elemento 'result' en el documento HTML.



```
1 const result = document.getElementById('result');
2 result.innerHTML = '';
3
4 const ul = document.createElement('ul');
5 data.forEach(element => {
6   console.log(element.region);
7   const li = document.createElement('li');
8   li.textContent = element.region;
9   ul.appendChild(li);
10 });
11
12 result.appendChild(ul);
13 }
```

Figure 16: ejercicio1.js

En el segundo ejercicio tenemos la función `totalConfirmed` toma un arreglo de datos como entrada y calcula el total de casos confirmados por región a partir de esos datos. Luego, genera una lista en el documento HTML que muestra cada región junto con su total de casos confirmados.

```

1 export function totalConfirmed(data){
2     // Obtener el elemento donde se mostrarán los resultados
3     const resultDiv = document.getElementById('result');
4     resultDiv.innerHTML = ''; // Limpiar cualquier contenido existente
5
6     // Calcular el total de casos confirmados por región
7     const regionTotals = data.reduce((acc, regionData) => {
8         const totalConfirmed = regionData.confirmed.reduce((sum, dayData) => {
9             return sum + parseInt(dayData.value, 10); // Sumar los valores confirmados por día
10        }, 0);
11        acc[regionData.region] = totalConfirmed; // Almacenar el total en el objeto acumulador
12        return acc;
13    }, {});
14
15    // Crear una lista no ordenada para mostrar los resultados
16    const ul = document.createElement('ul');
17    for (const [region, confirmed] of Object.entries(regionTotals)) {
18        const li = document.createElement('li'); // Crear un elemento de lista para cada región
19        li.textContent = `${region}: ${confirmed}`; // Establecer el texto del elemento de lista
20        ul.appendChild(li); // Agregar el elemento de lista a la lista no ordenada
21    }
22    resultDiv.appendChild(ul);
23 }

```

Figure 17: ejercicio2.js

En el tercer ejercicio la función top10Regions también recibe un arreglo de datos y muestra los 10 principales regiones junto con sus totales de casos confirmados en orden descendente en el documento HTML. Comienza limpiando cualquier contenido previo dentro del elemento HTML identificado por el id 'result'. Luego, calcula el total de casos confirmados por región, similar a la función totalConfirmed. Después, ordena las regiones por el número total de casos confirmados en orden descendente y toma las primeras 10. Finalmente, crea una lista no ordenada en el documento HTML para mostrar los resultados y agrega cada región junto con su total de casos confirmados como elementos de lista.

```

1 export function top10Regions(data) {
2     // Obtener el elemento donde se mostrarán los resultados
3     const resultDiv = document.getElementById('result');
4     resultDiv.innerHTML = ''; // Limpiar cualquier contenido existente
5
6     // Calcular el total de casos confirmados por región
7     const regionTotals = data.reduce((acc, regionData) => {
8         const totalConfirmed = regionData.confirmed.reduce((sum, dayData) => {
9             return sum + parseInt(dayData.value, 10); // Sumar los valores confirmados por día
10            }, 0);
11            acc[regionData.region] = totalConfirmed; // Almacenar el total en el objeto acumulador
12            return acc;
13        }, {});
14
15     // Ordenar las regiones por número total de casos confirmados en orden descendente y obtener las 10 primeras
16     const sortedRegions = Object.entries(regionTotals).sort((a, b) => b[1] - a[1]).slice(0, 10);
17
18     // Crear una lista no ordenada para mostrar los resultados
19     const ul = document.createElement('ul');
20     for (const [region, confirmed] of sortedRegions) {
21         const li = document.createElement('li'); // Crear un elemento de lista para cada región
22         li.textContent = `${region}: ${confirmed}`; // Establecer el texto del elemento de lista
23         ul.appendChild(li); // Agregar el elemento de lista a la lista no ordenada
24     }
25
26     // Agregar la lista no ordenada al elemento de resultados
27     resultDiv.appendChild(ul);
28 }
```

Figure 18: ejercicio3.js

En el cuarto ejercicio la función arequipaInfected se encarga de visualizar los datos de infectados en la región de Arequipa a lo largo del tiempo. Comienza filtrando los datos para obtener solo la información relacionada con la región de Arequipa. Luego, crea una nueva tabla de datos de Google Visualization y agrega una columna para las fechas y columnas para cada región en Arequipa. Posteriormente, itera sobre las fechas y para cada fecha, crea una nueva fila y agrega los valores de infectados correspondientes a cada región en esa fecha. Establece opciones de configuración para el gráfico, como título, dimensiones y estilos de ejes, y finalmente, crea una instancia del gráfico de líneas de Google Visualization y lo dibuja en el elemento HTML con el ID "result".

```

1 // Esta función visualiza los datos de infectados en la región de Arequipa en el tiempo de los valores
2 export function arequipaInfected(data){
3     // Filtrar los datos para obtener solo la información relacionada con la región de Arequipa
4     const newData = data.filter(region => region.region == 'Arequipa');
5     // Crear una nueva tabla de datos de Google Visualization
6     const chartData = new google.visualization.DataTable();
7     // Agregar una columna para las fechas
8     chartData.addColumn('string', 'Fecha');
9     // Agregar columnas para cada región en Arequipa
10    newData.forEach(region => chartData.addColumn('number', region.region));
11
12    // Extraer las fechas del primer objeto de región en newData
13    const dates = newData[0].confirmed.map(day => day.date);
14    // Iterar sobre las fechas
15    dates.forEach((date, index) => {
16        // Crear una nueva fila para cada fecha
17        const row = [date];
18        // Iterar sobre cada región en newData
19        newData.forEach(region => {
20            // Obtener el valor de infectados para la región actual en la fecha actual
21            const value = region.confirmed[index] ? parseInt(region.confirmed[index].value) : 0;
22            // Agregar el valor a la fila
23            row.push(value);
24        });
25        // Agregar la fila completa a la tabla de datos
26        chartData.addRow(row);
27    });
28
29    const options = {
30        title: 'Infectados en Arequipa',
31        width: 1200, // Ancho del gráfico
32        height: 800, // Alto del gráfico
33        hAxis: {
34            title: 'Fecha', // Título del eje x
35            slantedText: true,
36            slantedTextAngle: 45
37        },
38        vAxis: {
39            title: 'Número de Infectados', // Título del eje y
40            viewWindow: { //Min y Max para los numeros de infectados
41                min: 0,
42                max: 1200
43            }
44        },
45        legend: {
46            position: 'right'
47        }
48    };
49
50    // Crear una instancia del gráfico de líneas de Google Visualization y dibujarlo en el elemento HTML con el ID "result"
51    const chart = new google.visualization.LineChart(document.getElementById("result"));
52    chart.draw(chartData, options);
53 }

```

Figure 19: ejercicio4.js

En el quinto ejercicio la función comparativeLineChart analiza datos sobre casos confirmados en todas las regiones y crea un gráfico de líneas que compara el crecimiento de estos casos a lo largo del tiempo. Para ello, primero filtra los datos para obtener información de todas las regiones disponibles. Luego, estructura estos datos en una tabla de Google Visualization, donde cada fila representa una fecha y cada columna representa una región, con los valores de casos confirmados en cada fecha. Se definen opciones para personalizar el aspecto del gráfico, como el título y los ejes x e y. Finalmente, utiliza la biblioteca de Google Visualization para dibujar el gráfico en un elemento HTML específico identificado por el ID "result".

```

1  export function comparativeLineChart(data) {
2      // Filtrar datos para obtener todas las regiones
3      const newData = data.filter(region => region.region);
4
5      // Crear nueva tabla de datos de Google Visualization
6      const chartData = new google.visualization.DataTable();
7      chartData.addColumn('string', 'Fecha'); // Agregar columna para las fechas
8      newData.forEach(region => chartData.addColumn('number', region.region)); // Agregar columnas para cada región
9
10     // Obtener fechas del primer objeto de región en newData
11     const dates = newData[0].confirmed.map(day => day.date);
12
13     // Iterar sobre las fechas
14     dates.forEach((date, index) => {
15         const row = [date]; // Crear una nueva fila para cada fecha
16         newData.forEach(region => {
17             // Obtener el valor de confirmados para la región actual en la fecha actual
18             const value = region.confirmed[index] ? parseInt(region.confirmed[index].value) : 0;
19             row.push(value); // Agregar el valor a la fila
20         });
21         chartData.addRow(row); // Agregar la fila completa a la tabla de datos
22     });
23
24     // Opciones de configuración del gráfico
25     const options = {
26         title: 'Crecimiento en todas las regiones', // Título del gráfico
27         width: 1200, // Ancho del gráfico
28         height: 800, // Alto del gráfico
29         hAxis: {
30             title: 'Fecha', // Título del eje x
31             slantedText: true, // Texto inclinado en el eje x
32             slantedTextAngle: 45 // Ángulo de inclinación del texto en el eje x
33         },
34         vAxis: {
35             title: 'Número de Infectados', // Título del eje y
36             viewWindow: {
37                 min: 0, // Valor mínimo del eje y
38                 max: 2500 // Valor máximo del eje y (ajustar según el rango de tus datos)
39             }
40         },
41         legend: {
42             position: 'right' // Posición de la leyenda
43         }
44     };
45
46     // Crear instancia del gráfico de líneas de Google Visualization y dibujarlo en el elemento HTML con el ID "result"
47     const chart = new google.visualization.LineChart(document.getElementById("result"));
48     chart.draw(chartData, options);
49 }

```

Figure 20: ejercicio5.js

En el sexto ejercicio tenemos la función growthWithoutLimaCallao realiza un análisis similar a la función comparativeLineChart, pero excluye las regiones de Lima y Callao. Comienza filtrando los datos para excluir estas dos regiones y luego estructura los datos restantes en una tabla de Google Visualization. Posteriormente, crea un gráfico de líneas que muestra el crecimiento de casos confirmados para las regiones restantes a lo largo del tiempo. Se definen opciones de configuración para el gráfico, como el título y los ejes x e y. Finalmente, utiliza la biblioteca de Google Visualization para dibujar el gráfico en un elemento HTML específico identificado por el ID "result".

```

1  export function growthWithoutLimaCallao(data) {
2    const newData = data.filter(region => region.region !== 'Lima' && region.region !== 'Callao');
3
4    const chartData = new google.visualization.DataTable();
5    chartData.addColumn('string', 'Fecha');
6    newData.forEach(region => chartData.addColumn('number', region.region));
7
8    const dates = newData[0].confirmed.map(day => day.date);
9    dates.forEach((date, index) => {
10      const row = [date];
11      newData.forEach(region => {
12        const value = region.confirmed[index] ? parseInt(region.confirmed[index].value) : 0;
13        row.push(value);
14      });
15      chartData.addRow(row);
16    });
17
18    const options = {
19      title: 'Crecimiento sin Lima y Callao',
20      width: 1200,
21      height: 800,
22      hAxis: {
23        title: 'Fecha',
24        slantedText: true,
25        slantedTextAngle: 45
26      },
27      vAxis: {
28        title: 'Número de Infectados',
29        viewWindow: {
30          min: 0,
31          max: 1000 // Ajustar según el rango de tus datos
32        }
33      },
34      legend: {
35        position: 'right'
36      }
37    };
38
39    const chart = new google.visualization.LineChart(document.getElementById("result"));
40    chart.draw(chartData, options);
41  }

```

Figure 21: ejercicio6.js

En el séptimo ejercicio la función compareRegions compara el crecimiento de casos confirmados entre múltiples regiones utilizando una estructura similar a las funciones anteriores, pero sin excluir regiones específicas. Comienza creando una tabla de datos de Google Visualization con columnas para las fechas y cada región. Luego, añade filas de datos asumiendo que todas las regiones tienen las mismas fechas. Finalmente, genera un gráfico de líneas con opciones de configuración como título y ejes, mostrando la comparación en un elemento HTML con ID "result". Esta función simplifica el proceso de comparación al evitar la necesidad de excluir regiones específicas en el código.

```

1 // Por el momento es la misma lógica del mostrar todas las regiones sin Lima y Callao, es
2 // decir, estamos repitiendo código con pequeños cambios.
3
4 export function compareRegions(data) {
5   const chartData = new google.visualization.DataTable();
6   chartData.addColumn("string", "Fecha");
7
8   // Añadir columnas para cada región seleccionada
9   data.forEach(region => chartData.addColumn('number', region.region));
10
11 // Suponemos que todas las regiones tienen las mismas fechas
12 const dates = data[0].confirmed.map(day => day.date);
13
14 // Añadir filas de datos
15 dates.forEach((date, index) => {
16   const row = [date];
17   data.forEach(region => {
18     const value = region.confirmed[index] ? parseInt(region.confirmed[index].value) : 0;
19     row.push(value);
20   });
21   chartData.addRow(row);
22 });
23
24 const options = {
25   title: "Comparación entre Regiones",
26   width: 1200,
27   height: 800,
28   hAxis: {
29     title: "Fecha",
30     slantedText: true,
31     slantedTextAngle: 45
32   },
33   vAxis: {
34     title: "Número de Infectados",
35     viewWindow: {
36       min: 0,
37       max: 1000
38     }
39   },
40   legend: {
41     position: "right"
42   }
43 };
44
45 const chart = new google.visualization.LineChart(document.getElementById("result"));
46 chart.draw(chartData, options);
47 }
48

```

Figure 22: ejercicio7.js

En el octavo ejercicio la función drawComparativeChart, genera un gráfico comparativo del crecimiento de casos de infectados en Perú, excluyendo las regiones de Lima y Callao. Filtra la información para eliminar estas dos regiones, crea una tabla de datos y prepara filas con datos de casos confirmados por fecha y región. Luego, establece opciones para el gráfico, como título y ejes, y finalmente dibuja el gráfico de líneas en un elemento HTML específico. Esta función ofrece una forma clara de comparar el avance de la enfermedad en las regiones peruanas excluyendo Lima y Callao.

```

 1 // Autor: Jhossep Velarde
 2 export function drawComparativeChart(info) {
 3   // Filter out Lima and Callao
 4   let filteredInfo = info.filter(region => region.region !== 'Lima' && region.region !== 'Callao');
 5
 6   // Create the data table.
 7   let data = new google.visualization.DataTable();
 8   data.addColumn('string', 'Fecha');
 9
10   // Extract the unique dates
11   let dates = filteredInfo[0].confirmed.map(entry => entry.date);
12
13   // Add a column for each region
14   filteredInfo.forEach(region => {
15     data.addColumn('number', region.region);
16   });
17
18   // Prepare rows for the data table
19   let rows = dates.map((date, index) => {
20     let row = [date];
21     filteredInfo.forEach(region => {
22       row.push(parseInt(region.confirmed[index].value));
23     });
24     return row;
25   });
26
27   data.addRows(rows);
28
29   // Set chart options
30   let options = {
31     title: "Crecimiento de Infectados en Perú (excepto Lima y Callao)",
32     width: 1200,
33     height: 800,
34     hAxis: {
35       title: "Fecha",
36       slantedText: true,
37       slantedTextAngle: 45,
38     },
39     vAxis: {
40       title: "Número de Infectados"
41     },
42     chartArea: {
43       width: '70%',
44       height: '70%'
45     }
46   };
47
48   // Instantiate and draw our chart, passing in some options.
49   let chart = new google.visualization.LineChart(document.getElementById("result"));
50   chart.draw(data, options);
51 }

```

Figure 23: ejercicio8.js

Tenemos un main donde se ejecutara el script importa ocho funciones desde archivos JavaScript externos, cada una diseñada para realizar una tarea específica en la visualización de datos sobre casos de infectados en regiones de Perú. Estas funciones abarcan desde la simple visualización de listas de regiones hasta la comparación detallada del crecimiento de casos confirmados en diferentes áreas geográficas. Al asociar cada función a un botón en la interfaz de usuario, el script permite una fácil interacción del usuario, lo que facilita la exploración y comprensión de los datos sobre la situación de la pandemia en Perú.

```

1 import { listRegions } from "./ejercicio1.js";
2 import { totalConfirmed } from "./ejercicio2.js";
3 import { top10Regions } from "./ejercicio3.js";
4 import { arequipaInfected } from "./ejercicio4.js";
5 import { comparativeLineChart } from "./ejercicio5.js";
6 import { growthWithoutLimaCallao } from "./ejercicio6.js";
7 import { compareRegions } from "./ejercicio7.js";
8 import { drawComparativeChart } from "./ejercicio8.js";
9
10 google.charts.load('current', {'packages':['corechart']});
11 google.charts.setOnLoadCallback(initialize);
12
13 let data;
14
15 function loadFile() {
16     let url = "http://localhost:8000/data.json";
17
18     return fetch(url)
19         .then(response => response.json())
20         .then(jsonData => data = jsonData)
21         .catch(error => console.log(error));
22 }
23
24 function initialize() {
25     loadFile().then(() => {
26         document.getElementById('listRegions').addEventListener('click', showlistRegions);
27         document.getElementById('totalConfirmed').addEventListener('click', showtotalConfirmed);
28         document.getElementById('top10Regions').addEventListener('click', showtop10Regions);
29         document.getElementById('arequipaInfected').addEventListener('click', showarequipaInfected);
30         document.getElementById('comparativeLineChart').addEventListener('click', showcomparativeLineChart);
31         document.getElementById('growthWithoutLimaCallao').addEventListener('click', showGrowthWithoutLimaCallao);
32         document.getElementById('choiceToCompareRegions').addEventListener('click', choiceToCompareRegions);
33         document.getElementById('compareRegions').addEventListener('click', showCompareRegions);
34         document.getElementById('dailyGrowthWithoutLimaCallao').addEventListener('click', () => drawComparativeChart(data));
35     });
36 }
37 function showlistRegions(){
38     if(data){
39         listRegions(data);
40     }
41     return;
42 }
43
44 function showtotalConfirmed(){
45     if(data){
46         totalConfirmed(data);
47     }
48     return;
49 }
50
51 function showtop10Regions(){
52     if(data){
53         top10Regions(data);
54     }
55     return;
56 }
57
58 function showarequipaInfected(){
59     if(data){
60         arequipaInfected(data);
61     }
62     return;
63 }

```

Figure 24: main.js

```

1  function showcomparativeLineChart(){
2    if(data){
3      comparativeLineChart(data);
4    }
5    return;
6  }
7
8  function showGrowthWithoutLimaCallao() {
9    if (!data) return;
10   growthWithoutLimaCallao(data);
11 }
12
13 function showCompareRegions() {
14   const regionSelect = document.getElementById('regionSelect');
15   const regions = Array.from(regionSelect.selectedOptions).map(option => option.value);
16
17   if (regions.length === 0) {
18     alert("Selecciona al menos una región.");
19     return;
20   }
21
22   // Filtrar los datos para incluir solo las regiones seleccionadas
23   const newData = data.filter(region => regions.includes(region.region));
24
25   compareRegions(newData);
26 }
27
28 function choiceToCompareRegions() {
29   // Mostramos el div oculto
30   document.querySelector('.choiceRegions').style.display = 'block';
31
32   // Rellenamos el select con las regiones
33   let options = "";
34   data.forEach(region => options += `<option value="${region.region}">${region.region}</option>`);
35   document.getElementById('regionSelect').innerHTML = options;
36 }
37
38 initialize();
39

```

Figure 25: main.js

La ejecucion de los 8 ejercicios en el index.HTML

El primero :

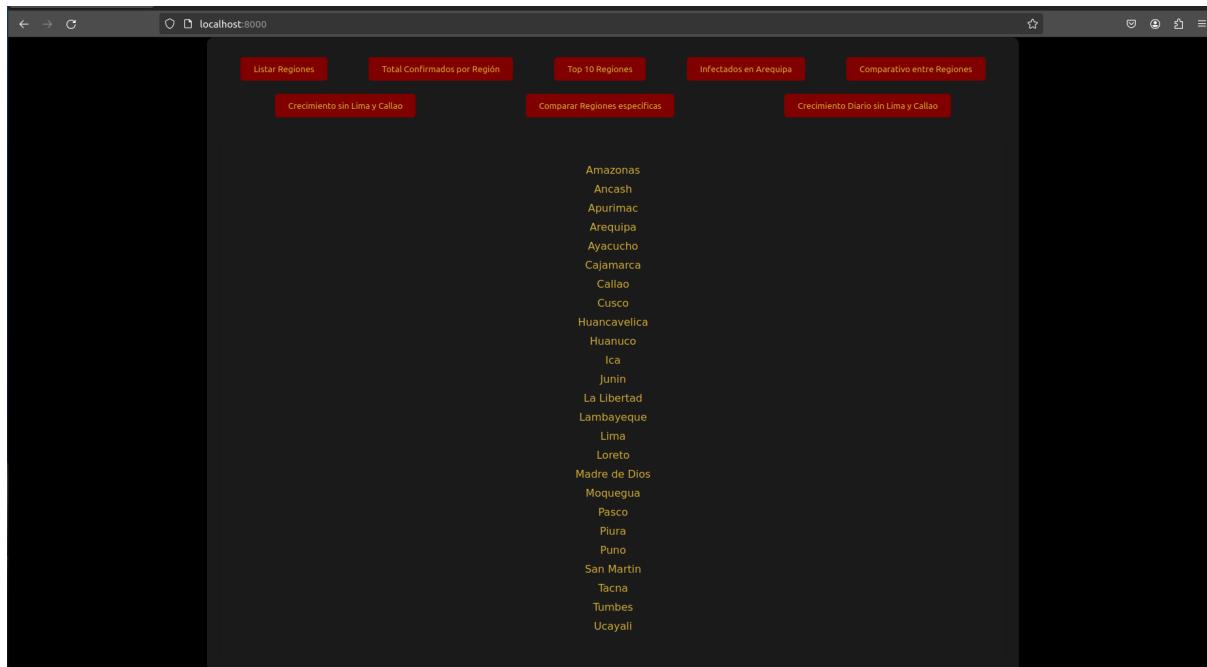


Figure 26: Ejecucion

El segundo :

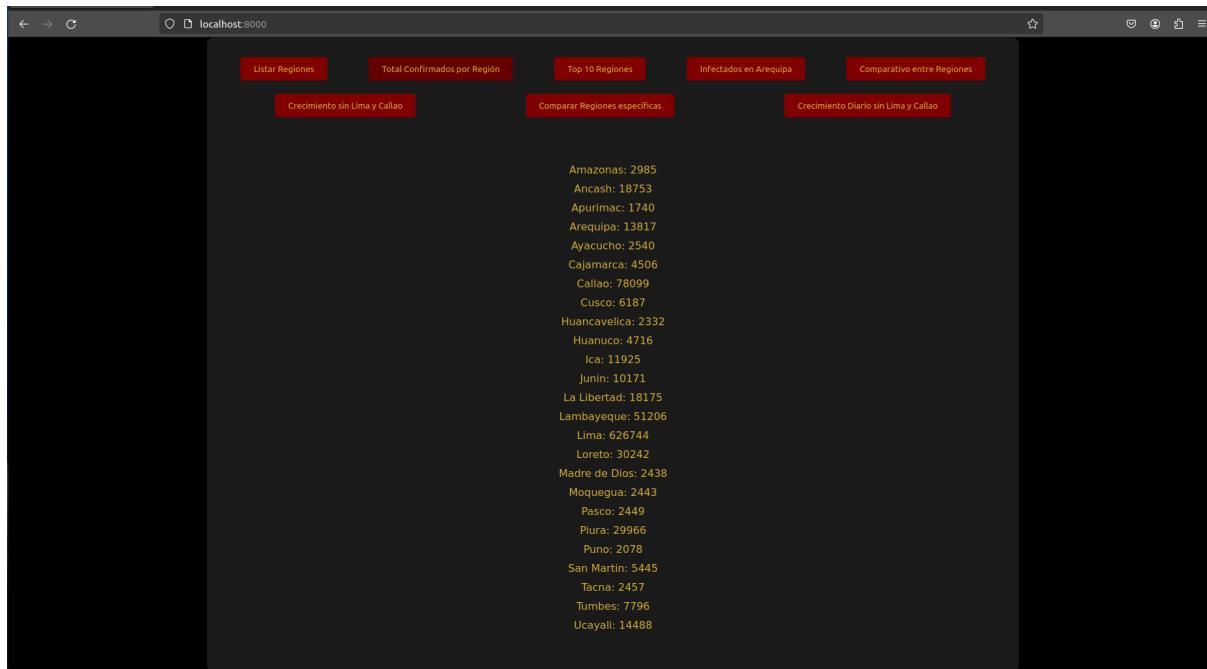


Figure 27: Ejecucion

El tercero :

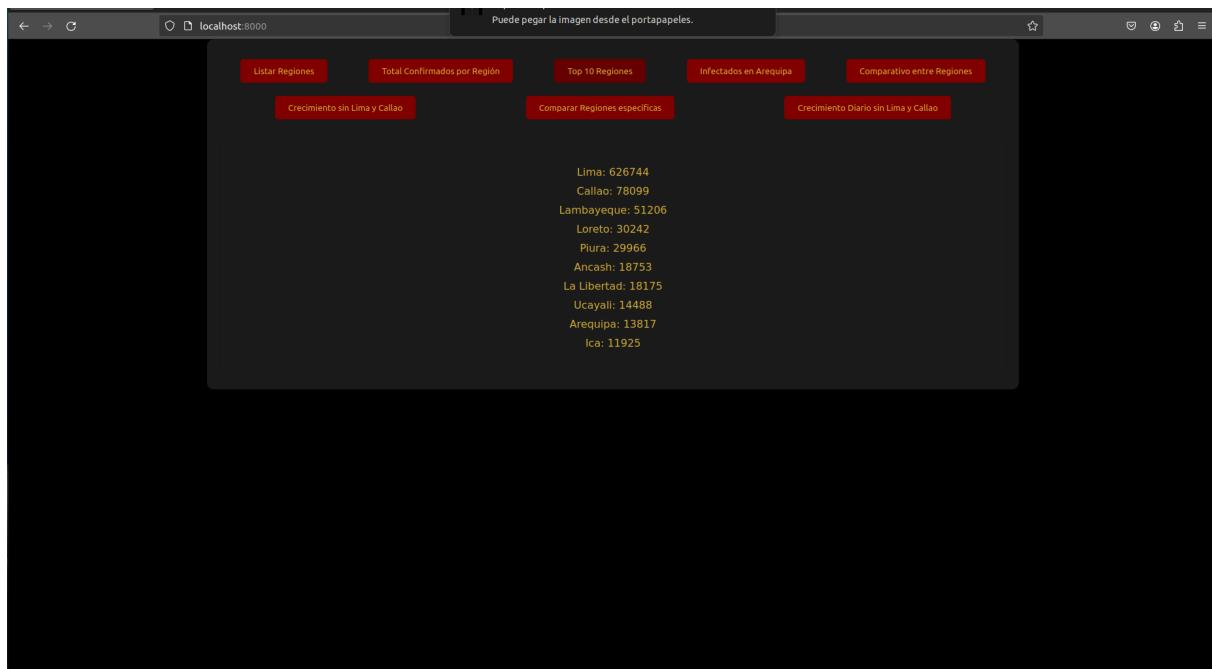


Figure 28: Ejecucion

El cuarto :

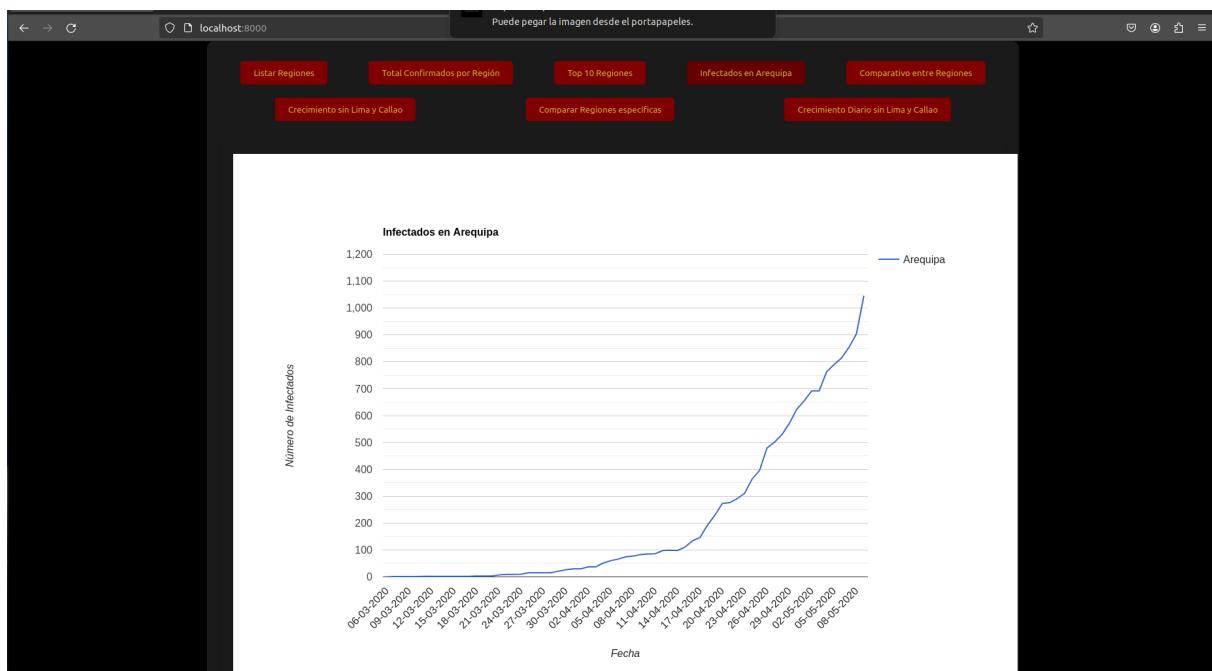


Figure 29: Ejecucion

El quinto :

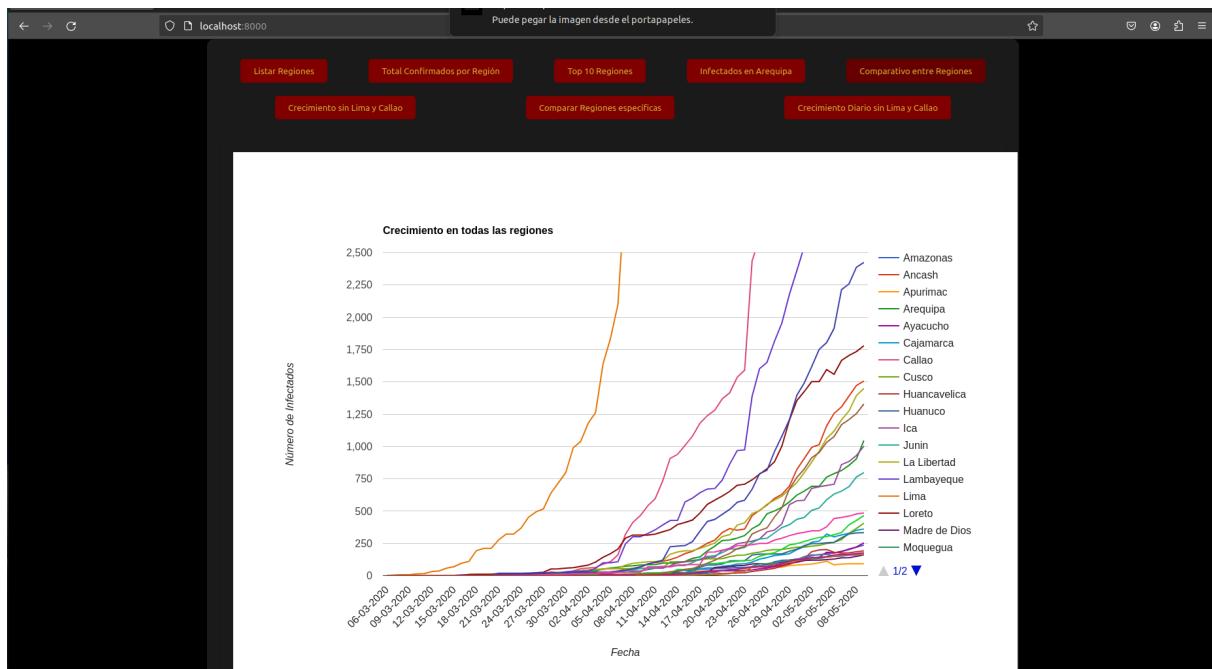


Figure 30: Ejecucion

El sexto :

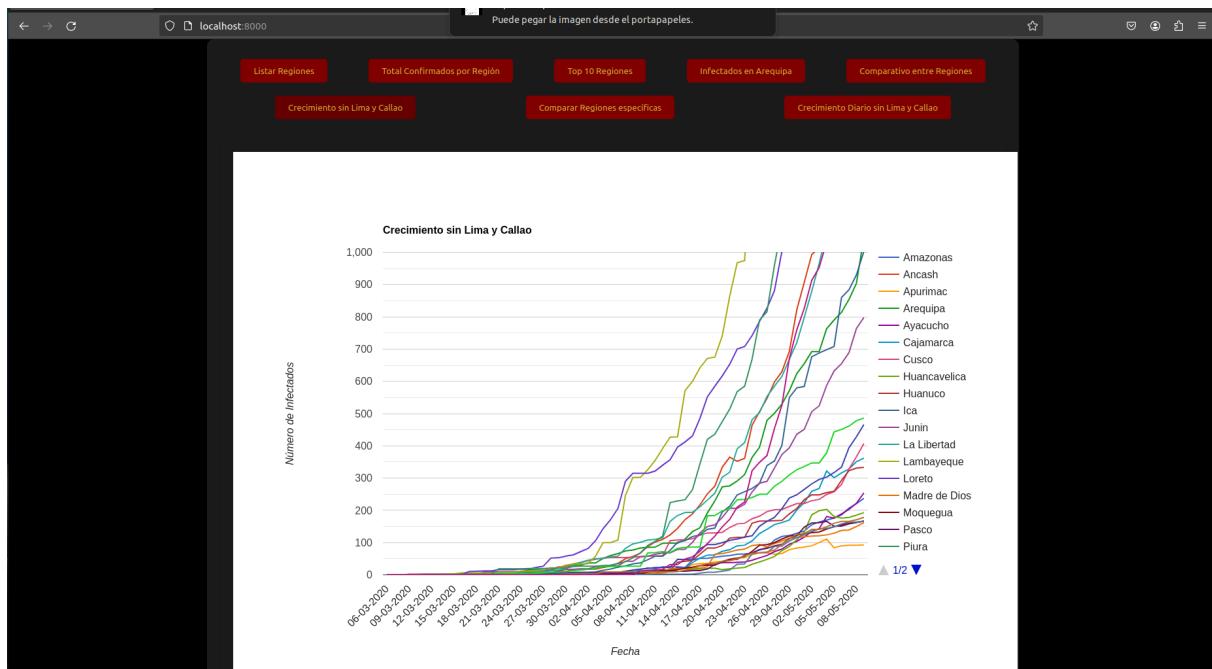


Figure 31: Ejecucion

El septimo :

