

Informe de Laboratorio 04

Tema: NodeJS + Express

Nota

Estudiante	Escuela	Asignatura
Mariel Alisson Jara Mamani mjarama@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: I Código: 1702122

Laboratorio	Tema	Duración
04	NodeJS + Express	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 22 Mayo 2024	Al 25 Mayo 2024

Índice





































1. Tarea	3
1.1. Descripción	3
1.2. Commits	3
1.3. Código	4
1.3.1. Lado Cliente	4
1.3.2. Lado Servidor	6
1.4. Diseño Responsivo	9
1.5. Pruebas	10
1.6. DockerFile	13
2. Pregunta	14
2.1. XMLHttpRequest (XHR)	14
2.1.1. Ejemplo	14
2.2. jQuery.ajax	14
2.2.1. Ejemplo	14
2.3. Fetch API	14
2.3.1. Ejemplo	15
3. Entregables	15
4. URL de Repositorio de Git Hub	15
5. Estructura de laboratorio 04	16
6. Rúbrica	17
7. Referencias	17































1. Tarea

1.1. Descripción

- Cree una aplicación NodeJS con express, para administrar una agenda personal.
- Home (“/”) : Página Principal
- Trabaje todo en una misma interfaz.
- La aplicación debe permitir:
 - Crear evento: fecha y hora. (Si ya existe el archivo no debería ingresar el evento)(La primera línea es el título del evento, las demás líneas son la descripción del evento.
 - Editar evento. (Se muestran el archivo donde esta el detalle del evento)
 - Eliminar evento
 - Ver eventos. Utilizar el formato árbol especificado anteriormente, donde debería incluirse sólo el título del evento.
- Utilice DockerFile para realizar operaciones automatizadas en Docker (incluido arrancar el servidor web nginx a través de un puerto y copiar el proyecto web para acceder desde la máquina anfitrión.)

1.2. Commits

Agenda Personal: Se crea la funcionalidad de listado del lado del cliente Alsnj20 committed 2 days ago	c641727			
Agenda Personal: Se crea el index.html y se sirve con el servidor Alsnj20 committed 2 days ago	6a79901			
Agenda Personal: Se crea el index.js y un archivo de prueba Alsnj20 committed 2 days ago	8930fde			
Tarea: Se crea el proyecto de Node Alsnj20 committed 2 days ago	6802857			
ejercicio 2, lado servidor Alsnj20 committed 3 days ago	5415921			
modified Alsnj20 committed 3 days ago	7dde150			
Esctructurando los archivos Alsnj20 committed 3 days ago	739e407			
Ejercicio 1 completado Alsnj20 committed 3 days ago	1e91a8e			
solo reconoce aquellas carpeta dentro de express ya que de ahi estamos partiendo a partir del get Alsnj20 committed 3 days ago	5429115			
carpeta staticas Alsnj20 committed 3 days ago	873273b			
ejercicio 2 usando express Alsnj20 committed 3 days ago	335b634			
ejercicio 1 server node Alsnj20 committed 3 days ago	593b44c			

Agenda Personal: Modificando create	36efa49			
Agenda Personal: Reescribiendo el HTML	605a5ce			
Agenda Personal: Btn Create funciona en el lado cliente y servidor	e0c618b			
Agenda Personal: Modificando algunos elementos para un mejor legibilidad en el diseño	0a1ec67			
Agenda Personal: El servidor envia los archivos de forma correcta	4f44b29			
Commits on May 24, 2024				
add id, corrigiendo errores de sintaxis	a2c6f05			
Commits on May 23, 2024				
Agenda Personal: Se crea la funcionalidad de listado del lado del servidor	1fb7f3b			
Agenda Personal: Se recrea la estructura de arbol para los eventos	95ba8bd			
Agenda Personal: Se crea la funcionalidad de listado del lado del cliente	c641727			
Agenda Personal: Se crea el index.html y se sirve con el servidor	6a79901			

1.3. Código

1.3.1. Lado Cliente

- A continuación se muestran las partes del código
- **Método 'create'** : En el lado del cliente, el código se encarga de capturar los valores del título, descripción y fecha del evento desde un formulario HTML. Luego, se separa la fecha de la hora y se verifica que todos los campos estén completos. Si los datos están completos, se envían al servidor mediante una solicitud POST utilizando el objeto XMLHttpRequest. Una vez que se envía la solicitud, se espera la respuesta del servidor y se maneja adecuadamente. Si la solicitud se completa con éxito, se llama a la función 'list()' para actualizar la lista de eventos en el cliente.

```
const create = () => {
  const title = document.getElementById('title').value;
  const description = document.getElementById('description').value;
  const date = document.getElementById('date').value;
  //Separar la fecha de la hora
  const parts = date.split('T');
  const datePart = parts[0]; //Fecha
  const timePart = parts[1].replace(':', '-'); //Hora
  if (title && description && datePart && timePart) {
    console.log("Titulo: " + title, "Descripcion: " + description, "Fecha: " +
      datePart, "Hora: " + timePart);
    //Enviar la informacion al servidor
    const xhr = new XMLHttpRequest();
    xhr.open('POST', '/create', true);
    xhr.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
    xhr.onreadystatechange = () => {
      if (xhr.readyState === 4 && xhr.status === 0) {
        console.log('Evento mandado al servidor');
        list();
      }
    };
  }
}
```

```
    }  
  }  
  xhr.send(JSON.stringify({ title: title, description: description, date: datePart,  
    time: timePart }));  
}  
}
```

- **Método 'edit':** Este código crea una interfaz para editar eventos en el navegador. Al hacer clic en el botón de editar de un evento, muestra un formulario con campos para el nuevo título y descripción. Cuando se envían los cambios, realiza una solicitud POST al servidor con los nuevos datos del evento y actualiza la lista de eventos en el cliente si la operación se realiza con éxito.

```
btnAccept.addEventListener('click', () => {  
  if (title.value !== '' || description.value !== '') {  
    editItem.style.display = 'none';  
    const newTitle = title.value;  
    const newDescription = description.value;  
  
    //Enviar la informacion al servidor  
    const xhr = new XMLHttpRequest();  
    xhr.open('POST', '/edit', true);  
    xhr.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');  
    xhr.onreadystatechange = () => {  
      if (xhr.readyState === 4 && xhr.status === 0) {  
        console.log('Evento editado');  
        list();  
      }  
    }  
    xhr.send(JSON.stringify({ title: newTitle, description: newDescription, time:  
      dateF}));  
    console.log('Evento mandado al servidor');  
    editItem.style.display = 'none';  
  } else {  
    console.log('Error en editar');  
  }  
})
```

- **Método 'remove':** Este código se ejecuta cuando se hace clic en el botón de eliminar un evento en la interfaz del usuario. Extrae la hora del evento a eliminar del elemento HTML correspondiente, realiza una solicitud POST al servidor con esta información y actualiza la lista de eventos en el cliente si la operación se realiza con éxito.

```
const remove = (btn) => {  
  ...  
  //Enviar la informacion al servidor  
  const xhr = new XMLHttpRequest();  
  xhr.open('POST', '/remove', true);  
  xhr.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');  
  xhr.onreadystatechange = () => {  
    if (xhr.readyState === 4 && xhr.status === 0) {  
      console.log('Evento eliminado');  
      list();  
    }  
  }  
  xhr.send(JSON.stringify({ time: dateF}));  
}
```

```
console.log('Evento mandado al servidor');  
}
```

- **Método 'list':** Este método envía una solicitud GET al servidor para obtener la lista de eventos. Cuando se recibe una respuesta exitosa del servidor, se procesa la respuesta y se crea una estructura de bloques de eventos con la función `createBlock()`. Cada bloque de evento contiene un título de fecha (h2) y una lista de tareas (divTasks) que se crea con la función `createTask()`.

```
const list = () => {  
  //Solicitud al servidor HTTP  
  const xhr = new XMLHttpRequest();  
  //Cuando la solicitud se complete  
  xhr.onreadystatechange = () => {  
    if (xhr.readyState === 4 && xhr.status === 200) {  
      //Obtiene la respuesta del servidor en formato JSON  
      const data = JSON.parse(xhr.responseText);  
      data.data.forEach((day) => {  
        createBlock(day.date, day.data);  
      });  
    }  
  }  
  xhr.open('GET', '/list', true);  
  xhr.send();  
};
```

- **Método 'listTask':** Similar a `list()`, este método envía una solicitud GET al servidor para obtener la lista de eventos, pero formatea los eventos en forma de árbol. Cuando se recibe una respuesta exitosa del servidor, se procesa la respuesta y se crea una estructura de lista de eventos en forma de árbol utilizando elementos `li` y `ul`.

```
const createBlock2 = (date, data) => {  
  const divDateTasks = document.createElement('li');  
  const divTitles = document.createElement('p');  
  const divTasks = document.createElement('ul');  
  divTitles.innerHTML = date + " [DIR]";  
  data.forEach((file) => {  
    const task = document.createElement('li');  
    task.innerHTML = file.time + ".txt [FILE]";  
    divTasks.appendChild(task);  
  });  
  divDateTasks.appendChild(divTitles);  
  document.getElementById('list').appendChild(divDateTasks);  
}
```

1.3.2. Lado Servidor

- **Método 'create' :** El código en el lado del servidor se encarga de recibir una solicitud POST en la ruta `/create`, extraer los datos del evento del cuerpo de la solicitud, verificar si los datos están completos, crear una carpeta para la fecha del evento si no existe y finalmente crear un archivo con la información del evento si el archivo correspondiente no existe. Si el evento se crea correctamente, se envía una respuesta con el mensaje 'Evento creado'.

```
//Crear un evento
```

```
app.post('/create', (req, res) => {
  console.log('POST /create');
  //Obtener la informacion del evento
  console.log(req.body);
  const { title, description, date, time } = req.body;
  if (!title || !description || !date || !time) {
    return res.status(400).send('Datos incompletos');
  }
  //Existe o no la fecha
  const dateFolder = path.resolve(__dirname, 'private', 'agenda', date);
  if (!fs.existsSync(dateFolder)) {
    fs.mkdirSync(dateFolder, { recursive: true });
  }
  //Crear el archivo con la informacion del evento
  const filePath = path.resolve(dateFolder, `${time}.txt`);
  if (!fs.existsSync(filePath)) {
    fs.writeFileSync(filePath, `${title}\n${description}\n`);
    res.status(201).send('Evento creado');
    console.log('Evento creado');
  } else {
    console.log('Evento ya existe');
  }
});
```

- **Método 'edit':** Este código maneja la edición de eventos. Recibe los nuevos datos del evento, busca el archivo correspondiente en el sistema de archivos, actualiza su contenido y envía una respuesta con el mensaje 'Evento editado'.

```
app.post('/edit', (req, res) => {
  console.log('POST /edit');
  //Obtener la informacion del evento
  console.log(req.body);
  const { title, description, time } = req.body;
  console.log(title, description, time);
  folders.forEach((folder) => {
    const files = fs.readdirSync(path.resolve(__dirname, 'private', 'agenda', folder))
    files.forEach((file) => {
      if (file === `${time}.txt`) {
        const filePath = path.resolve(__dirname, 'private', 'agenda', folder, file);
        fs.writeFileSync(filePath, `${title}\n${description}\n`);
        res.status(201).send('Evento editado');
        console.log('Evento editado');
      }
    })
  })
});
```

- **Método 'remove':** Este fragmento de código maneja la eliminación de eventos. Recibe la hora del evento a eliminar, busca el archivo correspondiente en el sistema de archivos y lo elimina. Luego envía una respuesta con el mensaje 'Evento eliminado'.

```
app.post('/remove', (req, res) => {
  console.log('POST /edit');
  //Obtener la informacion del evento
  console.log(req.body);
```

```
const {time } = req.body;
console.log(time);
folders.forEach((folder) => {
  const files = fs.readdirSync(path.resolve(__dirname, 'private', 'agenda', folder))
  files.forEach((file) => {
    if (file === `${time}.txt`) {
      const filePath = path.resolve(__dirname, 'private', 'agenda', folder, file);
      fs.unlinkSync(filePath);
      res.status(201).send('Evento eliminado');
      console.log('Evento eliminado');
    }
  })
})
});
```

- **Método 'list':** 'app.get('/list', ...)', define una ruta para manejar solicitudes GET a '/list'. Cuando se realiza una solicitud GET a esta ruta, el servidor responde con la lista de eventos almacenados en el sistema de archivos. El servidor lee los directorios y archivos en la ubicación de almacenamiento de eventos y los organiza en una estructura de datos JSON. Para cada evento encontrado, se agrega a la lista de eventos en la fecha correspondiente.

```
app.get('/list', (req, res) => {
  console.log('GET /list');
  //Fecha y titulo del evento
  const data = {
    dates: [],
  };
  //Lectura de directorios
  folders.forEach((folder) => {
    const files = fs.readdirSync(path.resolve(__dirname, 'private', 'agenda', folder))
    files.forEach((file) => {
      const content = fs.readFileSync(path.resolve(__dirname, 'private', 'agenda',
        folder, file), 'utf8')
      //Datos del evento
      let dateObj = data.dates.find(carpetas => carpetas.date === folder);
      if (!dateObj) {
        dateObj = {
          date: folder,
          data: []
        };
        data.dates.push(dateObj);
      }
      dateObj.data.push({
        title: content.substring(0, content.indexOf('\n')),
        description: content.substring(content.indexOf('\n') + 1, content.length),
        time: file.substring(0, file.indexOf('.'))
      })
    })
  });
  console.log(data);
  res.json(data);
});
```


1.4. Diseño Responsivo

- **Media Query:** Esta media query nos permite manejar el diseño sin que haya colapsos de ancho o alto en relación al tamaño de la pantalla.

```
@media (max-width: 799px) {  
  main {  
    padding: 4%;  
    grid-template-columns: repeat(1, 1fr);  
    row-gap: 20px;  
    height: auto;  
  }  
}
```

1.5. Pruebas

afsdhafsdjhfadslfadks fasdjkhfasdlhk

Editar

Eliminar

TITULO

Hacer Laboratorio

DESCRIPCIÓN

Laboratorio con node js y express

FECHA Y HORA

25/05/2024 07:30 p. m.

ADD EVENT

CANCEL

Plazo maximo hasta las 10pm

2024-05-25	
Hacer lab	
Plazo maximo hasta las 10pm	
<div>Editar</div> <div>Eliminar</div>	10-23
Hacer Laboratorio	
Laboratorio con node js y express	
<div>Editar</div> <div>Eliminar</div>	19-30

Hacer Laboratorio

Laboratorio con node js y express

[Editar](#)[Eliminar](#)

19-30

Hacer laboratorio 04

Laboratorio con node js y express en el
lado servidor

[Aceptar](#)[Cancelar](#)

Hacer laboratorio 04

Laboratorio con node js y express en el lado servidor

[Editar](#)[Eliminar](#)

19-30

Hacer laboratorio 04

Laboratorio con node js y express en el lado servidor

[Editar](#)[Eliminar](#)

19-30

List Task

[Ver todo](#)

2023-05-23 [DIR]

12-00.txt [FILE]

2023-05-24 [DIR]

08-00.txt [FILE]

12-00.txt [FILE]

12-30.txt [FILE]

2024-05-25 [DIR]

10-23.txt [FILE]

2024-06-02 [DIR]

12-30.txt [FILE]

13-30.txt [FILE]

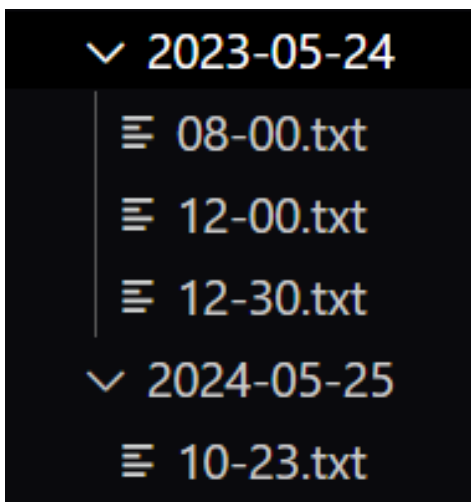
14-30.txt [FILE]

16-30.txt [FILE]

17-30.txt [FILE]

18-30.txt [FILE]

20-30.txt [FILE]



```
description: 'Laboratorio con node js y express',
date: '2024-05-25',
time: '19-30'
}
Evento creado
GET /list
{
  dates: [
    { date: '2023-05-23', data: [Array] },
    { date: '2023-05-24', data: [Array] },
    { date: '2024-05-25', data: [Array] },
    { date: '2024-06-02', data: [Array] }
  ]
}
POST /edit
{
  title: 'Hacer laboratorio 04',
  description: 'Laboratorio con node js y express en el lado servidor',
  time: '19-30'
}
Hacer laboratorio 04 Laboratorio con node js y express en el lado servidor 19-30
Evento editado
```

1.6. DockerFile

- Configuración para correr nginx

```
FROM node:18 as node_app
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
FROM nginx:alpine
COPY --from=node_app /app /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

2. Pregunta

- Mencione la diferencia entre conexiones asíncronas usando el objeto XMLHttpRequest, JQuery.ajax y Fetch. Justifique su respuesta con un ejemplo muy básico. Eje: Hola Mundo, IMC, etc.

Las diferencias fundamentales entre XMLHttpRequest (XHR), JQuery.ajax y Fetch son principalmente en su sintaxis, manejo de promesas, soporte para APIs modernas y facilidad de uso.

2.1. XMLHttpRequest (XHR)

Es una API antigua y fundamentalmente basada en eventos. Su uso implica una sintaxis más compleja y un manejo manual de eventos para realizar solicitudes y manejar respuestas. No es compatible con Promesas directamente, por lo que a menudo se envuelven en Promesas para un manejo más fácil y legible.

2.1.1. Ejemplo

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'https://api.example.com/data', true);
xhr.onload = function() {
  if (xhr.status >= 200 && xhr.status < 300) {
    console.log(xhr.responseText);
  } else {
    console.error('Request failed');
  }
};
xhr.send();
```

2.2. jQuery.ajax

Es una abstracción de XMLHttpRequest y proporciona una interfaz más simple y consistente. Es más legible y fácil de usar que XHR nativo. jQuery.ajax devuelve un objeto Promise, lo que facilita el manejo de solicitudes asíncronas y el encadenamiento de acciones.

2.2.1. Ejemplo

```
$.ajax({
  url: 'https://api.example.com/data',
  method: 'GET',
  success: function(response) {
    console.log(response);
  },
  error: function(xhr, status, error) {
    console.error('Request failed');
  }
});
```

2.3. Fetch API

Es la API más moderna y está integrada en los navegadores. Proporciona una sintaxis más simple basada en Promesas. Fetch devuelve una Promise que resuelve la respuesta, lo que facilita el manejo

de solicitudes y respuestas. Tiene soporte nativo para JSON y permite un manejo más fácil de los encabezados HTTP.

2.3.1. Ejemplo

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.error('Request failed', error));
```

3. Entregables

- Informe de laboratorio.
- Archivos correspondiente en el Repositorio.
- URL del Repositorio.

4. URL de Repositorio de Git Hub

- <https://github.com/Alsnj20/pw2-24a>

5. Estructura de laboratorio 04

- El contenido que se entrega en este laboratorio es el siguiente:

```
lab04/
|---/app
|---/node_modules
|---/private
|---/Agenda
|---/2024-05-25
|---10-23.txt
|---public
|---script (lado cliente)
|---create.js
|---edit.js
|---list.js
|---listTask.js
|---remove.js
|---script.js
|---style
|---ejercicio01.css
|---index.html
|---test.json (como se almacenan los datos de forma interna)
|---.gitignore
|---index.js (lado servidor)
|---LICENSE
|---package-lock.json
|---package.json
|---/execises
|---exercise1(express)
...
|---/latex
|--- linopinto_pw2_24a_lab04.tex
|--- linopinto_pw2_24a_lab04.pdf
|--- README.md
|---.gitignore
```


6. Rúbrica

Tabla: Rúbrica para contenido del Informe y evidencias

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Repositorio se pudo clonar y se evidencia la estructura adecuada para revisar los entregables. (Se descontará puntos por error o observación)	4	×	4	
2. Commits	Hay porciones de código fuente asociado a los commits planificados con explicaciones detalladas. (El profesor puede preguntar para refrendar calificación)	4	×	4	
3. Ejecución	Se incluyen comandos para ejecuciones y pruebas del código fuente explicadas gradualmente que permitirían replicar el proyecto. (Se descontará puntos por cada omisión)	4	×	4	
4. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación)	2	×	2	
7. Ortografía	El documento no muestra errores ortográficos. (Se descontará puntos por error encontrado)	2	×	1	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente con explicaciones puntuales pero precisas, agregando diagramas generados a partir del código fuente y refleja un acabado impecable. (El profesor puede preguntar para refrendar calificación)	4	×	3	
Total		20	Completo	18	

7. Referencias

- <https://github.com/>
- <https://git-scm.com/>
- <https://www.w3schools.com/nodejs/>
- https://www.w3schools.com/xml/ajax_xmlhttprequest_send.asp