Laboratorio 04

Tema: NodeJS + Express

Docente	Escuela	Asignatura
M.Sc. Ing. Richart Smith Escobedo Quispe rescobedoq@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web Semestre: III Código: 1702122

Laboratorio	Tema	Duración
04	NodeJS + Express	06 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	20 Mayo 2024	24 Mayo 2024

Índice

1. Especificaciones del Laboratorio **2**
1.1. Objetivos del curso . . . 2
1.2. Objetivos del laboratorio . . . 2
1.3. Equipos, materiales y temas . . . 2

2. Marco teórico **2**
2.1. NodeJS . . . 2
2.2. Solicitud de archivo en Node.js . . . 3
2.3. Uso de Fetch . . . 3
2.4. Express . . . 3

3. Guía de laboratorio **4**
3.1. NodeJS . . . 4
3.2. Express . . . 4
3.3. Ejercicios . . . 6
3.3.1. Directorio público . . . 6
3.3.2. Petición AJAX . . . 6
3.3.3. CORS . . . 7
3.3.4. AJAX-POST . . . 7

4. Tarea **10**
4.1. Descripción . . . 10
4.2. Pregunta . . . 10
4.3. Entregables . . . 11

5. Rúbricas **12**
5.1. Sobre el Informe . . . 12
5.2. Contenido del Informe . . . 12

6. Referencias **14**

1. Especificaciones del Laboratorio

1.1. Objetivos del curso

- Proporcionar los conocimientos y habilidades para el uso de las principales metodologías de análisis y diseño de sistemas.
- Brindar los conocimientos para la utilización de técnicas para el análisis y diseño de sistemas web.
- Proporcionar conocimientos y habilidades en el manejo de herramientas para el desarrollo de sistemas Web.
- Desarrollar sistemas de información dentro de una arquitectura cliente servidor.

1.2. Objetivos del laboratorio

- Utilizar el sistema de control de versiones Git.
- Utilizar la plataforma GitHub para replicar y administrar su proyectos en NodeJS.
- Aprender peticiones asíncronas en JavaScript usando JSON para la comunicación.
- Programar en BackEnd usando JavaScript.
- Entender el concepto de promises y los objetos no bloqueantes.

1.3. Equipos, materiales y temas

- Sistema Operativo (`GNU/Linux` de preferencia).
- Editor de texto plano (`GNU Vim` de preferencia).
- Navegador Web: Chrome, Firefox, Edge, Brave, Opera, etc.
- `Git`.
- Cuenta en `GitHub` asociada al correo institucional.
- NodeJS

2. Marco teórico

2.1. NodeJS

- Node.js es un entorno de servidor de código abierto
- Node.js es gratis
- Node.js se ejecuta en varias plataformas (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js usa JavaScript en el servidor
- Sitio web : `https://nodejs.org/es`

2.2. Solicitud de archivo en Node.js

- Una tarea común para un servidor web puede ser abrir un archivo en el servidor y devolver el contenido al cliente.
- Así es como PHP o ASP maneja una solicitud de archivo:
  1. Envía la tarea al sistema de archivos de la computadora.
  2. Espera mientras el sistema de archivos se abre y lee el archivo.
  3. Devuelve el contenido al cliente.
  4. Listo para manejar la próxima solicitud.
- Así es como Node.js maneja una solicitud de archivo:
  1. Envía la tarea al sistema de archivos de la computadora.
  2. Listo para manejar la próxima solicitud.
  3. Cuando el sistema de archivos ha abierto y leído el archivo, el servidor devuelve el contenido al cliente.
- Node.js elimina la espera y simplemente continúa con la siguiente solicitud.
- Node.js ejecuta programación asincrónica de un solo subproceso, sin bloqueo, que es muy eficiente en cuanto a memoria.

2.3. Uso de Fetch

- La API Fetch proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, tales como peticiones y respuestas.
- También provee un método global fetch() (en-US) que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red.
- Este tipo de funcionalidad se conseguía previamente haciendo uso de `XMLHttpRequest`. Fetch proporciona una alternativa mejor que puede ser empleada fácilmente por otras tecnologías como Service Workers (en-US).
- Fetch también aporta un único lugar lógico en el que definir otros conceptos relacionados con HTTP como CORS y extensiones para HTTP.

2.4. Express

- Infraestructura web rápida, minimalista y flexible para Node.js
- Aplicaciones web: Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.
- API: Con miles de métodos de programa de utilidad HTTP y middleware a su disposición, la creación de una API sólida es rápida y sencilla.
- Rendimiento: Express proporciona una delgada capa de características de aplicación web básicas, que no ocultan las características de Node.js que tanto ama y conoce.

3. Guía de laboratorio

3.1. NodeJS

- Instale NodeJS.
- Cree un nuevo proyecto NodeJS. (Cree un archivo `index.js`)

Listing 1: Instalando NodeJS en GNU/Linux

```
$ sudo apt update
$ sudo apt-get install nodejs
$ sudo apt-get install npm
$ node -v
```

Listing 2: Creando index.js

```
$ mkdir -p $HOME/rescobedoq/pw2-24a/lab04/exercises
$ cd $HOME/rescobedoq/pw2-24a/lab04/exercises
$ vim index.js
```

```
const http = require('http');
const server = http.createServer((request, response) => {
  console.log(request.url);
    response.end("Hola mundo");
});
server.listen(3000);
console.log("Escuchando en: http://localhost:3000")
```

Listing 3: Ejecutando un archivo NodeJS

```
$ node index.js
```

3.2. Express

- Instale Express.
- Cree un nuevo proyecto Express. (Cree un archivo `index.js`)
- Tenga cuidado donde instala express, ya que se creará directorios con librerías.

Listing 4: Directorio para Express en GNU/Linux

```
$ mkdir -p $HOME/rescobedoq/pw2-24a/lab04/exercises/express
$ cd $HOME/rescobedoq/pw2-24a/lab04/exercises/express
```

Listing 5: Instalando Express en GNU/Linux

```
$ npm install express
```

Listing 6: Creando index.js

```
$ vim index.js
```

```
1  const path = require('path');
2  const express = require('express');
3  const app = express();
4
5  app.listen(3000, () => {
6      console.log("Escuchando en: http://localhost:3000")
7
8  });
9
10 app.get('/', (request, response) => {
11     response.sendFile(path.resolve(__dirname, 'index.html'));
12 });
```

- Deberá crear un archivo index.html en el directorio de su proyecto.
- Note que el servidor espera recibir una petición de tipo GET y el URL no contiene el nombre del archivo.

3.3. Ejercicios

3.3.1. Directorio público

- Cree una aplicación web que ejecute javascript en el cliente (dentro de la carpeta pub) y nodejs en el servidor.

```
const path = require('path');
const express = require('express');
const app = express();
app.use(express.static('pub'));
app.listen(3000, () => {
    console.log("Escuchando en: http://localhost:3000")

});

app.get('/', (request, response) => {
    response.sendFile(path.resolve(__dirname, 'index.html'));
});
```

3.3.2. Petición AJAX

- Cree una aplicación web que realice una petición AJAX en el lado del cliente y responda usando nodejs en el lado del servidor.
- En el lado del servidor :

```
const fs = require('fs')
const path = require('path')
const express = require('express')
const app = express()
app.use(express.static('pub'))

app.listen(3000, () => {
    console.log("Escuchando en: http://localhost:3000")
});

app.get('/', (request, response) => {
    response.sendFile(path.resolve(__dirname, 'index.html'))
})

app.get('/recitar', (request, response) => {
    fs.readFile(path.resolve(__dirname, 'priv/poema.txt'), 'utf8',
        (err, data) => {
            if (err) {
                console.error(err)
                response.status(500).json({
                    error: 'message'
                })
                return
            }
            response.json({
                text: data.replace(/\n/g, '<br>')
            })
```

```
        })
    //
})
```

- En el lado del cliente :

```
function recitar() {
    const url = 'http://localhost:3000/recitar'
    fetch(url).then(
        response => response.json()
    ).then(
        data => {
            document.querySelector("#poema").innerHTML = data.text
        }
    )
}
```

3.3.3. CORS

- El intercambio de recursos entre orígenes (CORS) permite que las solicitudes AJAX omitan la política del mismo origen y accedan a recursos desde hosts remotos.
- En esta publicación, le mostraré cómo habilitar la compatibilidad con CORS en Express . También brindaré algunos consejos para manejar casos de uso comunes que surgen al trabajar con aplicaciones de una sola página, como exponer sesiones HTTP y encabezados personalizados.

Listing 7: Instalando Instalar el módulo cors

```
    $ npm install --save cors
```

```
var express = require('express');
var cors = require('cors');
var app = express();
app.use(cors());
/* your regular routes go here */
```

3.3.4. AJAX-POST

- Cree una aplicación que haga peticiones AJAX usando POST a un servidor NodeJS.
- En el lado del servidor :

```
const fs = require('fs')
const path = require('path')
const express = require('express')
const bp = require('body-parser')
const MarkdownIt = require('markdown-it'),
    md = new MarkdownIt();
```

```
const app = express()

app.use(express.static('pub'))
app.use(bp.json())
app.use(bp.urlencoded({
    extended: true
}))

app.listen(3000, () => {
    console.log("Escuchando en: http://localhost:3000")
})

app.get('/', (request, response) => {
    response.sendFile(path.resolve(__dirname, 'index.html'))
})

app.post('/', (request, response) => {
    console.log(request.body)
    let markDownText = request.body.text
    console.log(markDownText)
    let htmlText = md.render(markDownText)
    response.setHeader('Content-Type', 'application/json')
    response.end(JSON.stringify({
        text: htmlText
    }))
})
```

- Note que tanto el tanto GET como POST comparten el mismo URL.
- En el lado del cliente :

```
function recitar(markupText) {
const url = 'http://localhost:3000/'
const data = {
  text: markupText
}
console.log(data)
const request = {
  method: 'POST', // íPodra ser GET
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
}
http = fetch(url, request)
http.then(
  response => response.json()
).then(
  data => {
    document.querySelector("#htmlCode").innerHTML = data.text
  }
    )
}
document.addEventListener('DOMContentLoaded', function () {
```

```
    const text = document.querySelector('#markupText')
    document.querySelector('#markupForm').onsubmit = () => {
        recitar(text.value)
        return false;
    }
})
```

Listing 8: Instalando Instalar el módulo cors

```
$ npm install --save cors
```

```
var express = require('express');
var cors = require('cors');
var app = express();
app.use(cors());
/* your regular routes go here */
```

4. Tarea

4.1. Descripción

- Cree una aplicación NodeJS con express, para administrar una agenda personal.
- Home ("/") : Página Principal
- Trabaje todo en una misma interfaz.
- Ejemplo de estructura de la agenda cuando se explora "Eventos"

```
agenda[DIR]
  |
|------------- 2023-05-15 [DIR]
|   |-------------- 10-00.txt [FILE]
|   |-------------- 13-30.txt [FILE]
|------------- 2023-05-20 [DIR]
  |-------------- 09-10.txt [FILE]
  |-------------- 20-40.txt [FILE]
```

- 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 traves de un puerto y copiar el proyecto web para acceder desde la máquina anfitrion.)
- Produción acceder a la aplicación NodeJS+Express a traves de un servidor web robusto (Nginx).
- Ejemplo: http://127.0.0.1:8084/lab04/

4.2. 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.

4.3. Entregables

- URL al directorio específico del laboratorio en su repositorio GitHub privado donde esté todo el código fuente y otros que sean necesarios. Evitar la presencia de archivos: binarios, objetos, archivos temporales, cache, librerias, entornos virtuales. Si hay configuraciones particulares puede incluir archivos de especificación como: packages.json, requirements.txt o README.md.
- No olvide que el profesor debe ser siempre colaborador a su repositorio que debe ser privado (Usuario del profesor **@rescobedoq**).
- El informe debe estar elabórado en LaTeX
- Usted debe describir sólo los **commits** más importantes que marcaron hitos en su trabajo, adjutando capturas de pantalla, del commit, porciones de código fuente, evidencia de sus ejecuciones y pruebas.
- En el informe siempre se debe explicar las imágenes (código fuente, capturas de pantalla, commits, ejecuciones, pruebas, etc.) con descripciones puntuales pero precisas.
- Agregar la estructura de directorios y archivos de su laboratorio hasta el nivel 4.

```
rescobedoq/
|--- pw2-24a
|   |--- README.md
|   |--- .gitignore
|   |--- lab04
|       |--- README.md
|       |--- index.html
|       |--- index.js
|       |--- css
|           |--- style.css
|       |--- exercises
|           |--- index.js
|       |--- latex
|           |--- rescobedoq_pw2_23b_lab04.tex
|           |--- rescobedoq_pw2_23b_lab04.pdf
```

5. Rúbricas

5.1. Sobre el Informe

Tabla 1: Rúbrica para el Informe

Informe		**Cumple**	**No cumple**
LaTeX	El informe está en formato PDF desde LaTeX, con un formato limpio (buena presentación) y fácil de leer.	20	0
Observaciones	Respetar la estructura de organización para ubicación de los entregables. Por cada observación dentro del informe se le descontará puntos. Se debe incluir el código fuente latex del informe (`*.tex`)	-	-

5.2. Contenido del Informe

- El alumno deberá autocalificarse, marcando o dejando en blanco las celdas de la columna **Checklist**, deacuerdo a si cumplió o no con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación siempre será sobre la nota mínima aprobatoria, siempre y cuando cumpla con todos lo items. (Máximo 24 horas)
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la tabla de calificación de niveles de desempeño:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: 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 onservación)	4			
2. Commits	Hay porciones de código fuente asociado a los commits planificados con explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	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. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2			
7. Ortografía	El documento no muestra errores ortográficos. (Se descontará puntos por error encontrado)	2			
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			
Total		20			

6. Referencias

- https://github.com/rescobedoq/backend-js
- https://medium.com/zero-equals-false/using-cors-in-express-cac7e29b005b
- https://www.w3schools.com/nodejs/nodejs_intro.asp
- https://nodejs.org/en/docs/guides/getting-started-guide
- https://www.w3schools.com/js/js_api_fetch.asp
- https://expressjs.com/es/
- https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Using_Fetch
- https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction
- https://nodejs.org/docs/latest-v18.x/api/