

Actividad de Construcción Aplicada

Sergio Antonio Trujillo del Valle

Septiembre de 2021

Corporación Unificada Nacional.

Cundinamarca.

Materia: Servidores y Servicios Web

Docente: Dilsa Enith Triana Martinez

Índice

Actividad de Construcción Aplicada

Código del servidor:	3
Código del cliente:	5
Capturas de pantalla:	6
Conclusiones:	6
Referencias	8

Servidores y servicios web Actividad de Construcción Aplicada

Para el desarrollo de la actividad se eligen: nodejs para la capa servidor y javascript como lenguaje de programación.

Código del servidor:

```
const net = require('net')

//Creamos el servidor
const server = net.createServer()

// Alimentamos la informaicón de los arrays
const phrases = [
  "La tecnología se alimenta a si misma. La tecnología hace posible más tecnología.-Alvin Toffler.",
  "La tecnología es sólo una herramienta. En términos de llevar a los niños a trabajar juntos y motivarlos, el profesor es el más importante.-Bill Gates.",
  "La máquina tecnológicamente más eficiente que el hombre ha inventado es el libro.-Northrop Frye.",
  "Ya no hacen más los bugs como bunny - Olav Mjelde",
  "Un lenguaje de programación es de bajo nivel cuando requiere que prestes atencion a lo irrelevante.-Alan J. Perlis.",
  "Hablar es barato. Enséñame el código.-Linus Torvalds ",
  "No me importa si funciona en su máquina! No me envían su máquina!. -Vidiu Platon",
  "Siempre codifica como si la persona que finalmente mantendrá tu código fuera un psicópata violento que sabe dónde ives. -Martin Golding"
]

const books = [
  "Divina Comedia - Dante Alighieri",
  "Don Quijote de la Mancha - Miguel de Cervantes",
  "Cien años de soledad - Gabriel García Márquez",
  "Moby Dick - Herman Melville",
  "Ana Karenina - Lev Tolstói",
  "Eneida - Virgilio",
  "Otelo - William Shakespeare",
  "El viejo y el mar - Ernest Hemingway",
```

```

    "Orgullo y prejuicio - Jane Austen"
  ]

  //Creamos la función para elegir un valor aleatorio
  function shuffle(array) {
    return array[Math.floor(Math.random()*array.length)];
  }

  //Escribimos la lógica del servidor
  server.on('connection', (socket)=> {
    //Cuando el servidor recibe información ejecuta
    socket.on('data', (data)=> {
      switch(data.toString('utf-8')){
        case 'book':
          socket.write(shuffle(books))
          break
        case 'libro':
          socket.write(shuffle(books))
          break
        case 'phrase':
          socket.write(shuffle(phrases))
          break
        case 'frase':
          socket.write(shuffle(phrases))
          break
        default:
          socket.write('No existen resultados, intenta
nuevamente')
          break
      }
      console.info('Consulta Recibida para
'+data.toString('utf-8')+'!')
    })
    //Si hay algun tipo de error
    socket.on('error', (error)=> {
      console.log(error.message )
    })
    //Cuando se cierra la conexión ejecuta
    socket.on('close', ()=> {
      console.info("\nLa comunicación ha sido finalizada")
    })
  })
})

```

```
//Lanzamos el servidor
server.listen(5000, () => {
  console.info("Servidor activo y en escucha desde el puerto ",
server.address().port)
})
```

Código del cliente:

```
const net = require('net')
const read = require('readline-sync')

//Funcion para capturar texto por consola
function sendPetition() {
  let consoleRead = read.question('\nIngrese el tipo de
consulta:\n')

  if (consoleRead == 'exit') {
    console.info('\nConexión Finalizada!')
    client.end()
  } else {
    client.write(consoleRead)
  }
}

//Creamos la conexión del cliente
const client = net.createConnection({
  port: 5000,
  host: '127.0.0.1'
})

//Aplicamos la logica cuando se conecta
client.on('connect', ()=>{
  console.info('Conexión exitosa!\n')
  sendPetition()
})

//Cuando el servidor envíe información se ejecutara
client.on('data', (data)=>{
  console.clear()
})
```

```

    console.info('La respuesta obtenida del servidor es:\n
->' + data.toString('utf-8'))
    sendPetition()
  })

//En caso de tener un error la conexión mostrara el error
client.on('error', (error)=> {
  console.log(error.message)
})

```

Capturas de pantalla:

The screenshot shows two VS Code editor windows. The left window, titled 'server.js', displays the server code. The right window, titled 'client.js', displays the client code. Both windows have a terminal at the bottom showing the command prompt.

```

server > JS server.js > ...
1 const net = require('net')
2
3 //Creamos el servidor
4 const server = net.createServer()
5
6 // Alimentamos la información de los arrays
7 const phrases = [
8   "La tecnología se alimenta a si misma. La tecnolog
9   "La tecnología es sólo una herramienta. En término
10  "La máquina tecnológicamente más eficiente que el
11  "Ya no hacen más los bugs como bunny - Olav Mjelde
12  "Un lenguaje de programación es de bajo nivel cuan
13  "Hablar es barato. Enséñame el código.-Linus Torva
14  "No me importa si funciona en su máquina! No me en
15  "Siempre codifica como si la persona que finalment
16 ]
17
18 const books = [
19   "Divina Comedia - Dante Alighieri",
20 ]

```

```

client > JS client.js > ...
1 const net = require('net')
2 const read = require('readline-sync')
3
4 //Funcion para capturar texto por consola
5 function sendPetition() {
6   let consoleRead = read.question('\nIngrese el tipo de consulta
7 }
8 if (consoleRead == 'exit') {
9   console.info('\nConexión Finalizada!')
10  client.end()
11 } else {
12   client.write(consoleRead)
13 }
14 }
15
16 //Creamos la conexión del cliente
17 const client = net.createConnection({
18   port: 5000,
19   host: '127.0.0.1'
20 })

```

Terminal output for server.js:

```

[almatysta@localhost server]$

```

Terminal output for client.js:

```

[almatysta@localhost client]$

```

This screenshot is identical to the one above, showing the same VS Code editor windows for 'server.js' and 'client.js' with their code and terminal output.

Conclusiones:

- El modelo cliente servidor es uno de los mayores desarrollos que tuvo la web con la que se pueden lograr la creación de múltiples soluciones para el manejo de información ya sea para crear una web estática, una aplicación web, una api rest full o un microservicio.
- Al ser agnóstico de la tecnología, el modelo cliente servidor permite que se pueda desarrollar en diferentes lenguajes de programación de forma muy similar al momento de implementarlo.
- Gracias al modelo cliente servidor es posible en un solo desarrollo, la implementación de microservicios y apis para ejecutar diversas acciones en la aplicación o recibir de distintas fuentes la información requerida.

Referencias

MonkeyWit. 29 de Diciembre de 2020. Cliente-Servidor TCP con Node.js - Comunicación via Socket. Recuperado el 11 de septiembre de 2021 de <https://www.youtube.com/watch?v=LFU7gJAOegA>