

Tarea 02

Diseñar y construir una plataforma tecnológica para la transmisión, almacenamiento y verificación de datos de un censo poblacional

Alumnos:

Josselyn Fajardo
Michael Avilés
Edwin Bajaan
Pedro Palma

Docente:

Ing. Edwin Boza G, Phd.

Fecha:

Guayaquil, 17 de junio del 2023

Contenido

Descripción general.....	3
Requerimientos mínimos.....	3
Arquitectura de la solución propuesta	4
Alcance de la tarea 2.....	4
Herramientas requeridas	5
Desarrollo	6

Descripción general

La plataforma tecnológica diseñada y construida para el censo poblacional tiene como objetivo facilitar la recopilación, transmisión, almacenamiento y verificación de datos de manera eficiente y segura. A continuación, se detallan las principales características y componentes de la plataforma:

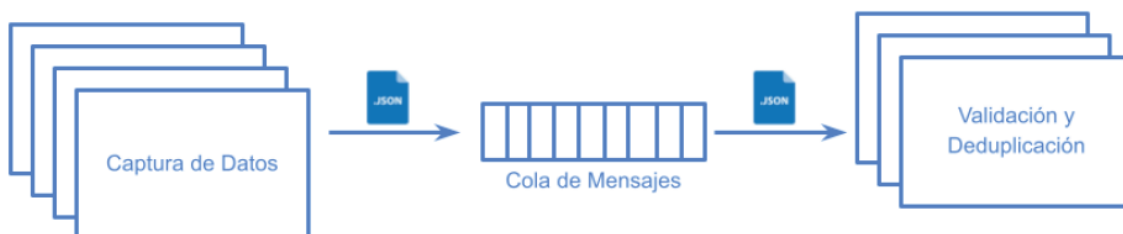
La aplicación proporciona formularios electrónicos y funcionalidades que guían a los encuestadores durante el proceso de recopilación de datos.

Se pueden incluir controles de validación en tiempo real para reducir errores y asegurar la calidad de los datos ingresados.

La plataforma cuenta con una infraestructura segura que garantiza la privacidad y confidencialidad de los datos durante la transmisión.

Se desarrolla una interfaz de usuario intuitiva que permite a los usuarios autorizados acceder y consultar los datos del censo.

Se pueden incluir herramientas de análisis y visualización de datos para facilitar la comprensión y extracción de conocimientos.



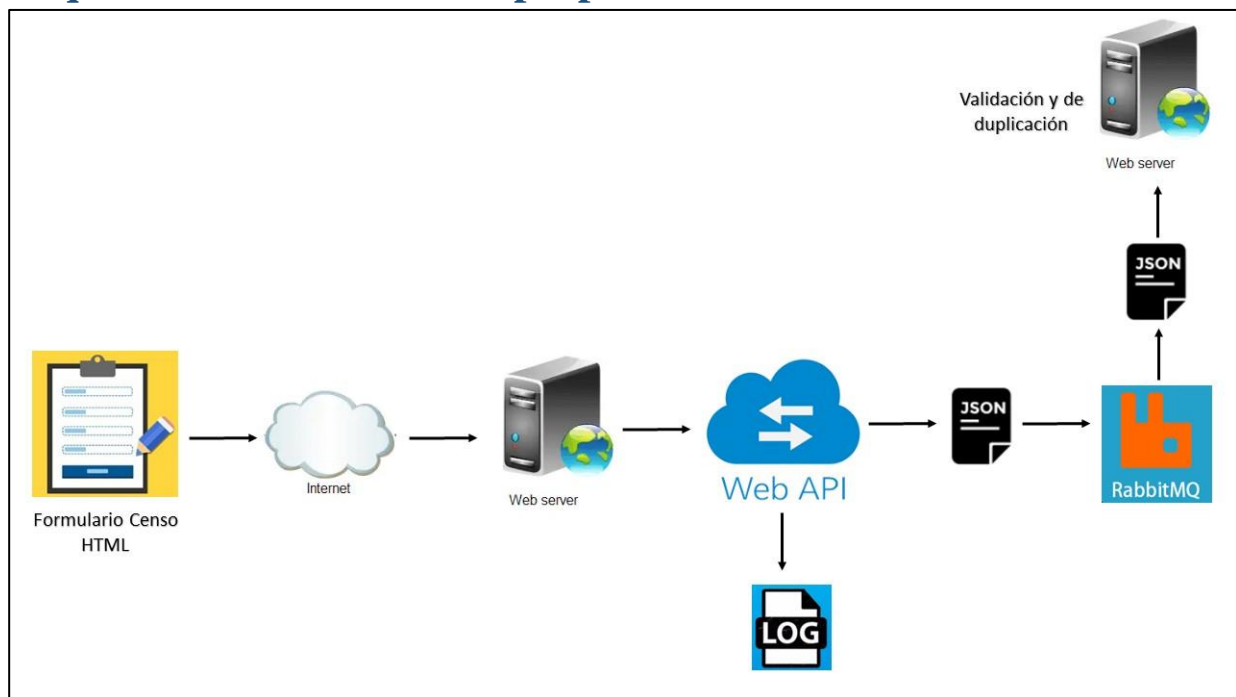
Requerimientos mínimos

1. **Módulo de Captura de Datos:** Será una aplicación independiente que generará de forma automática los formularios del censo llenos, para lo cual puede llenarlos de forma aleatoria. Cada vez que genere un nuevo formulario, deberá conectarse por medio de la red IP a la cola de mensajes y depositar el formulario codificado en formato JSON.
2. **Módulo de Cola de Mensajes:** Debe ser funcional, es decir que debe permitir la comunicación efectiva entre los módulos de Captura de Datos y de Validación/De duplicación.
3. **Módulo de Validación/De duplicación:** Se encargará de obtener los formularios en formato JSON desde la cola de mensajes. Para esta entrega bastará con que obtenga los formularios y realice la validación.

Definición del proyecto

La creación de un sistema de censo poblacional es el proceso de diseñar, desarrollar e implementar una infraestructura tecnológica y metodológica que permite recopilar y gestionar datos demográficos y socioeconómicos de una determinada población. Este sistema tiene como objetivo principal obtener información precisa y actualizada sobre la composición y características de una población en particular.

Arquitectura de la solución propuesta



Alcance de la tarea 2

Diseño y desarrollo del servidor Linux:

Configuración del servidor Linux con los requisitos necesarios, como el sistema operativo, servicios web, base de datos, etc.

Implementación de medidas de seguridad para proteger el servidor y los datos almacenados.

Desarrollo de la aplicación web en Windows:

Utilización de herramientas de desarrollo web, como HTML, CSS y JavaScript, para crear la interfaz de usuario de la aplicación.

Uso de Java para desarrollar la lógica del lado del servidor y la interacción con el servidor Linux y RabbitMQ.

Implementación de la integración con RabbitMQ para el intercambio de mensajes entre la aplicación web y el servidor Linux.

Implementación de RabbitMQ:

Configuración de RabbitMQ como un servicio de mensajería para facilitar la comunicación entre el servidor Linux y la aplicación web en Windows.

Definición de las colas de mensajes y los intercambios necesarios para el procesamiento de datos del censo.

Intercambio de archivos JSON:

Definición de un formato de archivo JSON para el intercambio de datos del censo entre el servidor Linux y la aplicación web.

Implementación de mecanismos para enviar y recibir archivos JSON a través de RabbitMQ, asegurando la integridad de los datos durante el proceso.

Registro de logs:

Implementación de un sistema de registro de logs en el servidor Linux y la aplicación web para realizar un seguimiento de las operaciones realizadas y facilitar el proceso de depuración y monitoreo.

Definición de los eventos y mensajes relevantes a registrar, así como el almacenamiento y acceso a los registros generados.

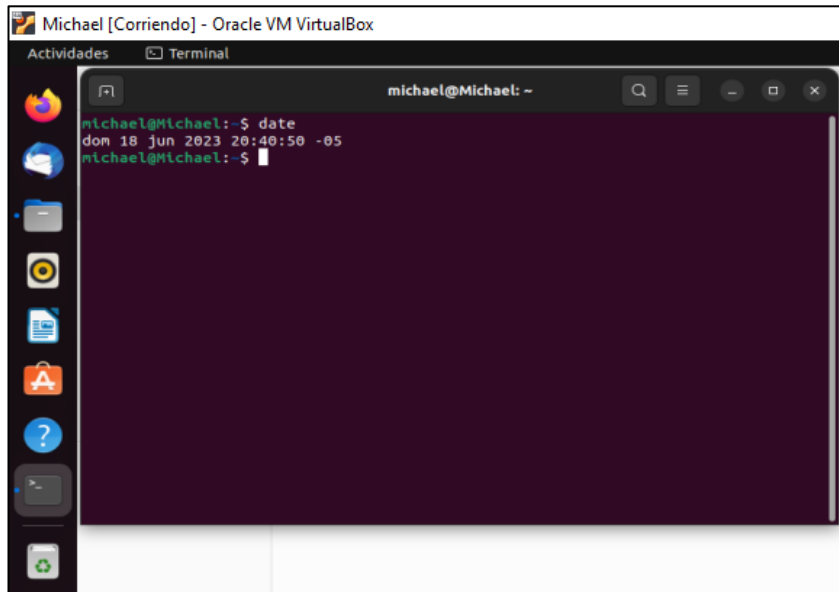
Herramientas requeridas

Las herramientas que vamos a usar para el desarrollo del proyecto son:

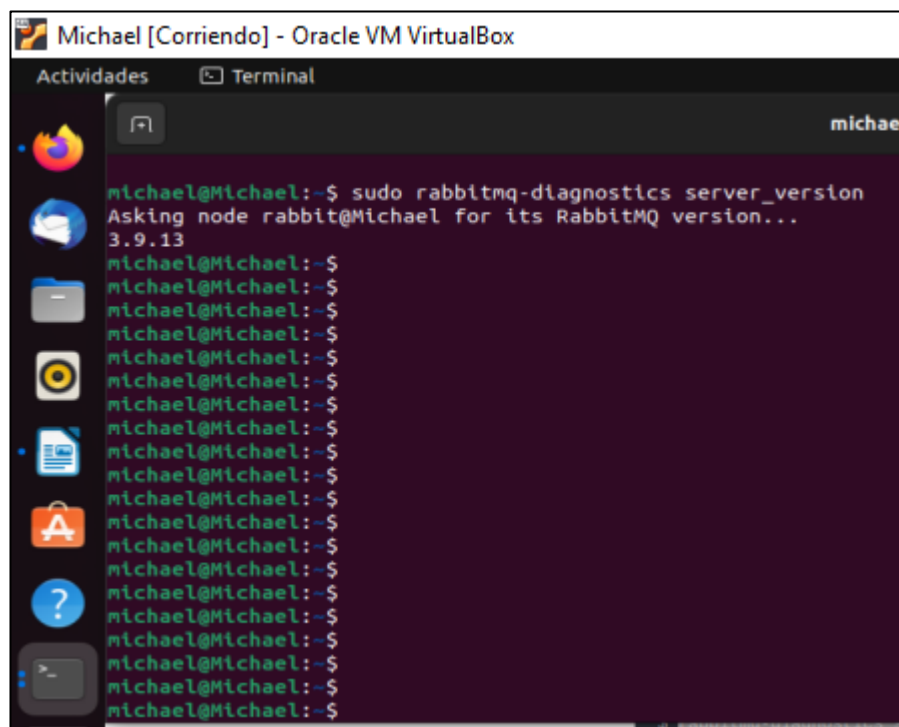
- Oracle VM Virtual Box, se instaló una máquina virtual Ubuntu.
- Visual Studio Code, con las Librerías necesarias para operar en Windows y Linux UBUNTU
- Entorno de Ejecución: Node.js Sobre el motor de JavaScript V8 de Google Chrome
- Docker Desktop, para Operación y Administración de Contenedores:
- Contenedor de RABBIT MQ, operando en Linux UBUNTU:
- Administrador de RABBITMQ Versión 3.12.0; para gestión del contenedor.

Desarrollo

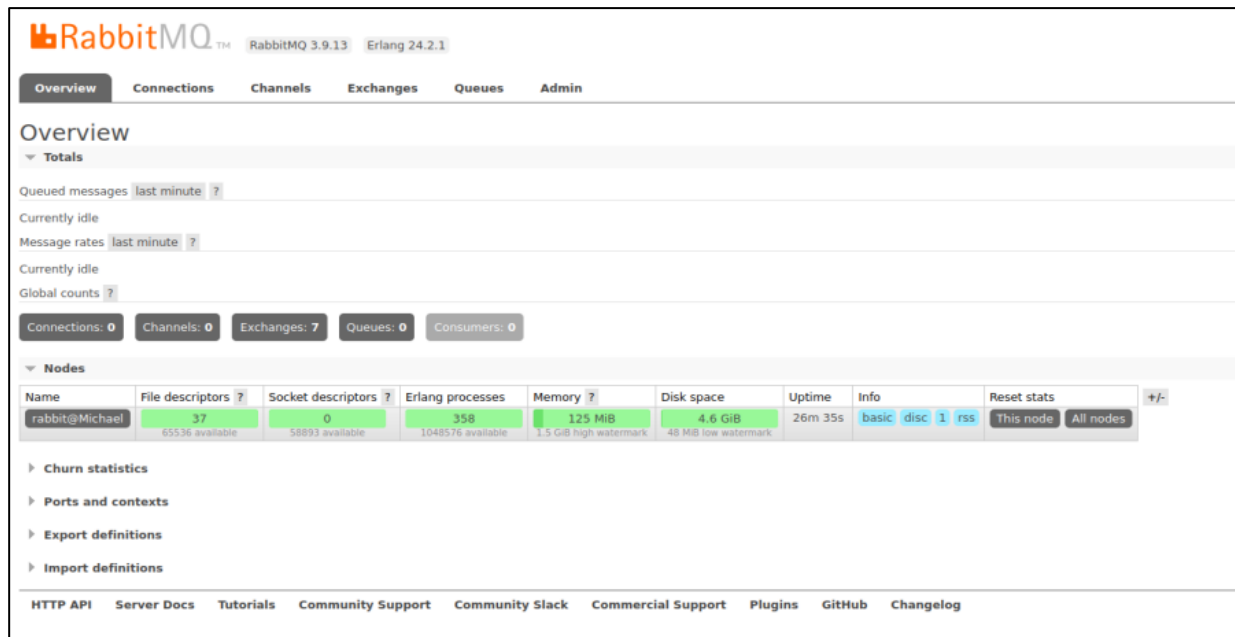
Máquina virtual Ubuntu instalada en Oracle VM VirtualBox:



Rabbit instalado a nivel de servidor:



Administrador de RabbitMQm Versión 3.9.13



Formulario en Html para el llenado de datos “index.html”

```

1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <meta charset="utf-8">
6      <title>Principal</title>
7      <link rel="stylesheet" href="estilo.css">
8  </head>
9
10 <body>
11     <div class="cuerpo_principal">
12
13         <div class="cont_paginas">
14             <form action="http://localhost:3000/submit" method="POST">
15                 <div class="paginas">
16                     <h3 class="txt_integrantes_titulo">Integrantes del Grupo:</h3>
17                     <li class="txt_integrantes">Josselyn Fajardo</li>
18                     <li class="txt_integrantes">Edwin Bajaña</li>
19                     <li class="txt_integrantes">Pedro Palma</li>
20                     <li class="txt_integrantes">Michael Avilés</li>
21
22                     <h3 class="texto_cabecera">Formulario de censo poblacional</h3>
23
24                     <div id="contenido" class="contenedor-inputs">
25
26                         <div class="input1">

```

Servidor en js para obtención de los datos y creación del json “server1.js”

```
JS server1.js X
Programacion > JS server1.js > [?] server > [?] http.createServer() callback > [?] req.on('end') callback
1  const http = require('http');
2  const fs = require('fs');
3  const querystring = require('querystring');
4  const { timeStamp } = require('console');
5
6  const server = http.createServer((req, res) => {
7    if (req.method === 'GET' && req.url === '/') {
8      // Servir el archivo index.html cuando se solicita la página de inicio
9      fs.readFile('index.html', (err, data) => {
10        if (err) {
11          res.writeHead(500);
12          res.end('Error interno del servidor');
13        } else {
14          res.writeHead(200, { 'Content-Type': 'text/html' });
15          res.end(data);
16        }
17      });
18    } else if (req.method === 'POST' && req.url === '/submit') {
19      // Manejar la solicitud POST del formulario
20      let body = '';
21      req.on('data', chunk => {
22        body += chunk;
23      });
24      req.on('end', () => {
25        // Realiza cualquier acción necesaria con los datos recibidos
26        const formData = querystring.parse(body);
27        const name = formData.txt_nombre;
28        const nacionalidad = formData.txt_nacionalidad;
```

Archivo json creado para enviarlo a la cola de mensajes “pasar.json”

```
C:\xampp\htdocs\Programacion\pasar.json - Notepad++
Archivo  Editar  Buscar  Vista  Codificación  Lenguaje  Configuración  Herramientas  Macro  Ejecutar  Plugins  Ventana  ?
pasar.json
1  [{"id":"2023-06-19T14:45:26.267Z_0930991310","Nombre":"Michael Aviles","Nacionalidad":"Ecuatoriana","Cedula":"0930991310","Direc
2  {"id":"2023-06-21T02:47:18.263Z_0925032722","Nombre":"Joselynn Fajardo","Nacionalidad":"Ecuatoriana","Cedula":"0925032722","Dir
3  {"id":"2023-06-21T05:52:10.892Z_0969999999","Nombre":"Juan","Nacionalidad":"Perez","Cedula":"0969999999","Direccion":"Suburvio
4
```

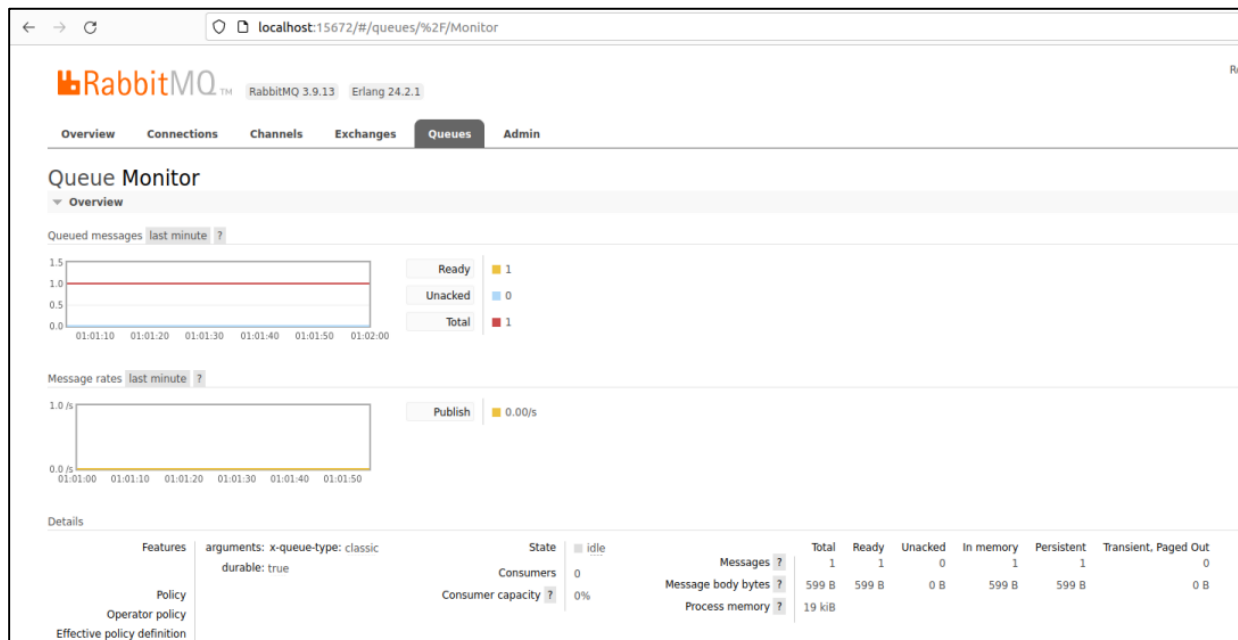

Servidor de js el cual leer el archivo .json y envía los datos a la cola de mensajes de Rabbit por el canal Monitor “envia_rabbit.js”

```

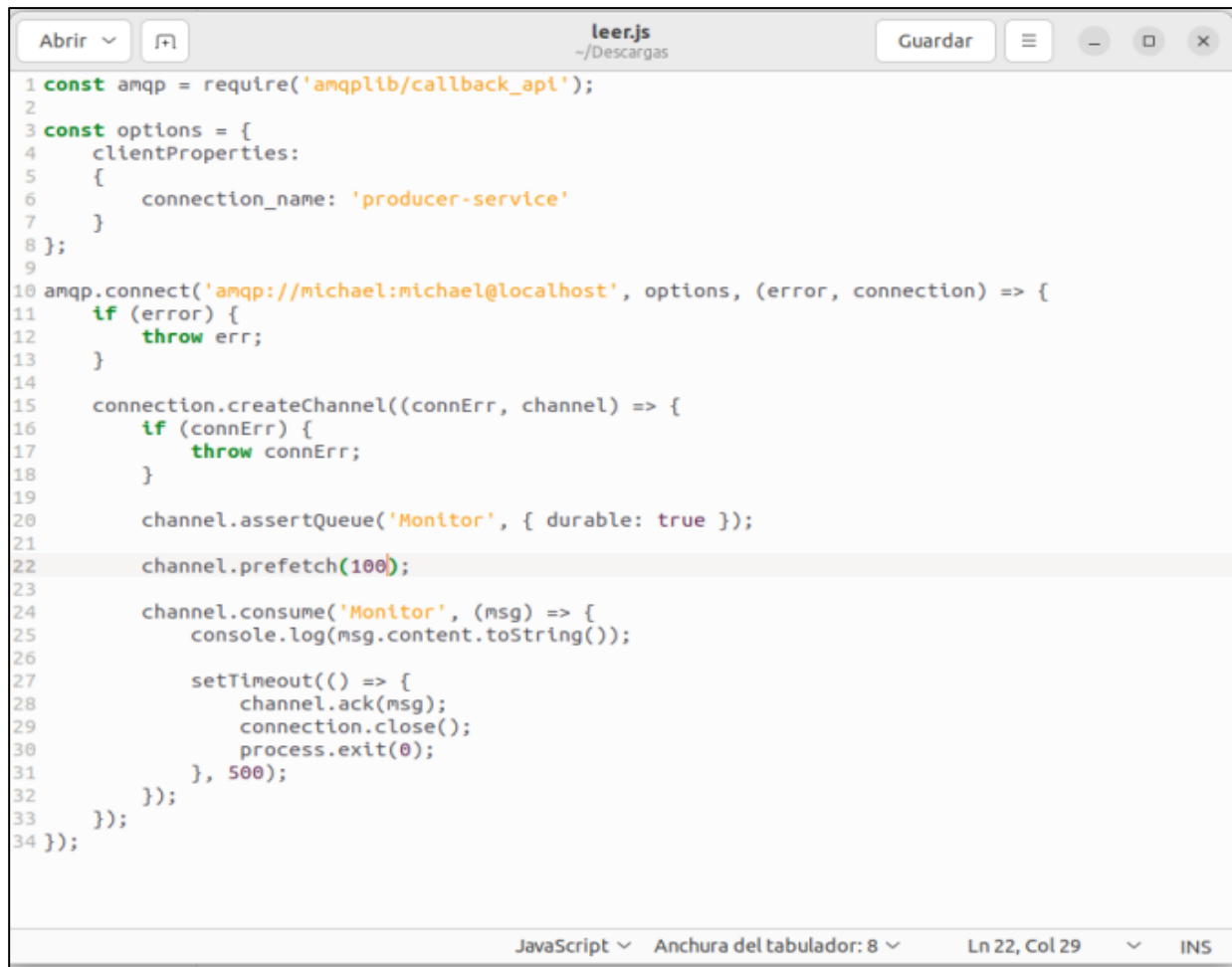
JS envia_rabbit.js X
Programacion > JS envia_rabbit.js > amqp.connect("amqp://michael:michael@192.168.100.134") callback > connection.cre
1  const amqp = require('amqplib/callback_api');
2
3  const options = {
4    clientProperties:
5    {
6      connection_name: 'producer-service'
7    }
8  };
9
10 amqp.connect('amqp://michael:michael@192.168.100.134', options, (error, connection) => {
11   if (error) {
12     throw error;
13   }
14
15   connection.createChannel((connErr, channel) => {
16     if (connErr) {
17       throw connErr;
18     }
19
20     channel.assertQueue('Monitor', {
21       durable: true
22     });
23
24     const fs = require('fs');
25
26     // Ruta al archivo que deseas leer
27     const rutaArchivo = 'C:/xampp/htdocs/Programacion/pasar.json';
28
29     // Función para leer el archivo
30     fs.readFile(rutaArchivo, 'utf8', (error, contenido) => {

```

Monitor de canal de Rabbit donde se verifica que llegó el mensaje



Servidor de js que verifica los mensajes que existen en el canal de Rabbit “leer.js”



```

1 const amqp = require('amqplib/callback_api');
2
3 const options = {
4   clientProperties:
5     {
6       connection_name: 'producer-service'
7     }
8 };
9
10 amqp.connect('amqp://michael:michael@localhost', options, (error, connection) => {
11   if (error) {
12     throw err;
13   }
14
15   connection.createChannel((connErr, channel) => {
16     if (connErr) {
17       throw connErr;
18     }
19
20     channel.assertQueue('Monitor', { durable: true });
21
22     channel.prefetch(100);
23
24     channel.consume('Monitor', (msg) => {
25       console.log(msg.content.toString());
26
27       setTimeout(() => {
28         channel.ack(msg);
29         connection.close();
30         process.exit(0);
31       }, 500);
32     });
33   });
34 });
  
```

Captura del mensaje de Rabbit por medio de un js “leer.js”



```

Michael@Michael: ~/Descargas
Michael@Michael:~/Descargas$ nodejs leer.js
{"id":"2023-06-19T14:45:26.267Z_0930991310","Nombre":"Michael Aviles","Nacionalidad":"Ecuatoriana","Cedula":"0930991310","Direccion":"Mucho Lote 2","Telefono":"0960676644","Correo":"naviles@uees.edu.ec"}
{"id":"2023-06-21T02:47:18.263Z_0925032722","Nombre":"Josselyn Fajardo","Nacionalidad":"Ecuatoriana","Cedula":"0925032722","Direccion":"Mucho Lote 2","Telefono":"0999999999","Correo":"josselyn.fajardo@uees.edu.ec"}
{"id":"2023-06-21T05:52:10.892Z_0969999999","Nombre":"Juan","Nacionalidad":"Perez","Cedula":"0969999999","Direccion":"Suburvio","Telefono":"0969999999","Correo":"jperez@ug.edu.ec"}
  
```

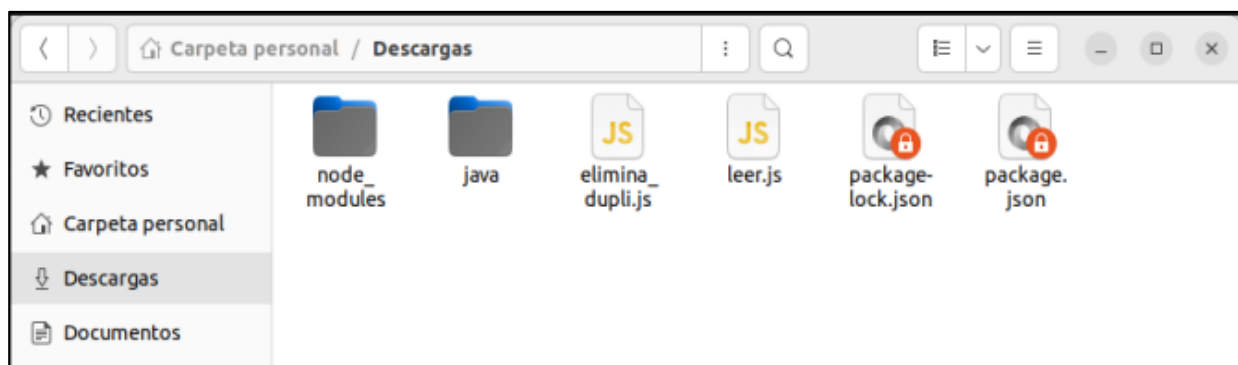

Archivo js que verifica que no se duplique los mensajes recibidos en la cola de mensajes de rabbit
“elimina_dupli.js”

```

JS elimina_dupli.js X
Programacion > JS elimina_dupli.js > conectarRabbit
1  const amqp = require('amqplib');
2
3  async function conectarRabbit() {
4      try {
5          // Conexión al servidor RabbitMQ
6          const conexion = await amqp.connect('amqp://localhost');
7          const canal = await conexion.createChannel();
8
9          // Declaración de la cola
10         const nombreCola = 'Monitor';
11         await canal.assertQueue(nombreCola, { durable: false });
12
13         console.log('Conexión a RabbitMQ establecida. Esperando mensajes...');
14
15         // Almacenar IDs de mensajes procesados
16         const mensajesProcesados = new Set();
17
18         // Consumir mensajes de la cola
19         canal.consume(nombreCola, (mensaje) => {
20             const contenido = mensaje.content.toString();
21             const idMensaje = mensaje.properties.id;
22
23             // Verificar si el mensaje ya fue procesado
24             if (mensajesProcesados.has(idMensaje)) {
25                 console.log(`Mensaje duplicado recibido, descartando: ${contenido}`);
26                 canal.ack(mensaje);
27                 return;
28             }
29

```

Archivos creados en máquina virtual



Repositorio de GitHub “https://github.com/themic08/sistemas_distribuidos”

