

Tarea 02

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

Alumnos:

Edwin Bajaanã
Josselyn Fajardo
Pedro Palma
Michael Avilés

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:

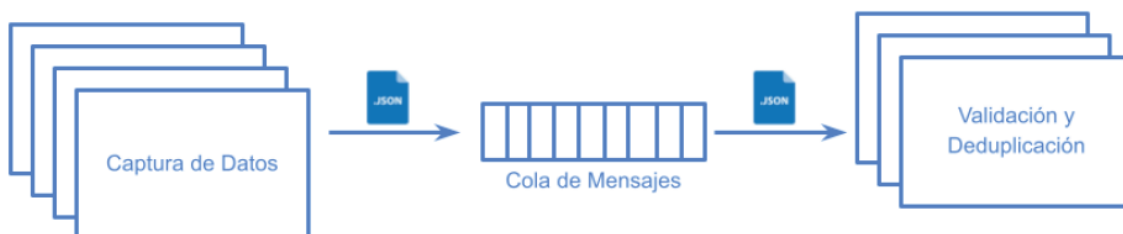
La aplicación proporciona formularios electrónicos y funcionalidades que guían a los encuestadores durante el proceso de recopilación de datos.

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.

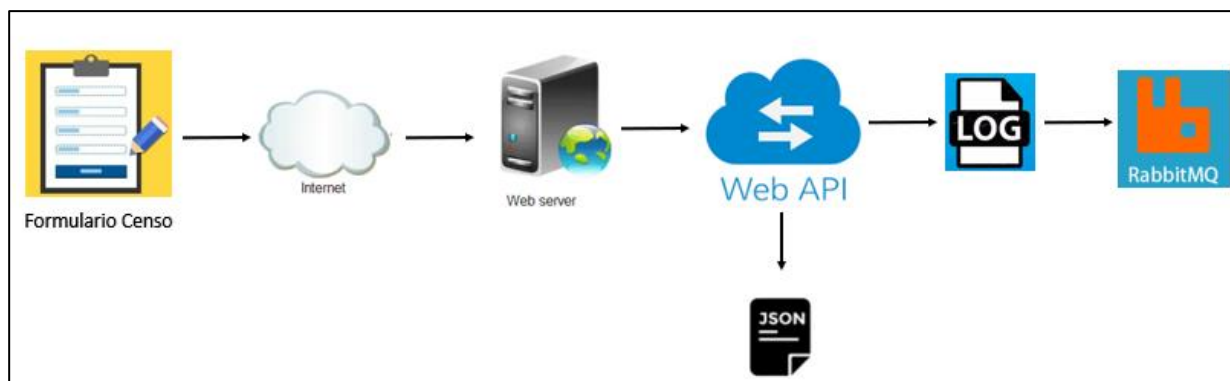


Requerimientos mínimos

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/Deduplicación.
3. **Módulo de Validación/De deduplicación:** 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.
4. **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.

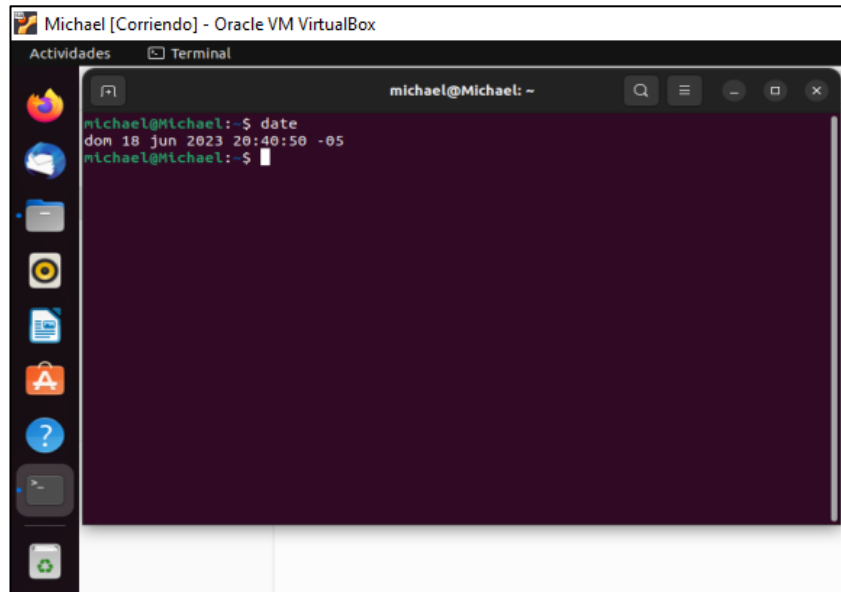
Herramientas requeridas

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

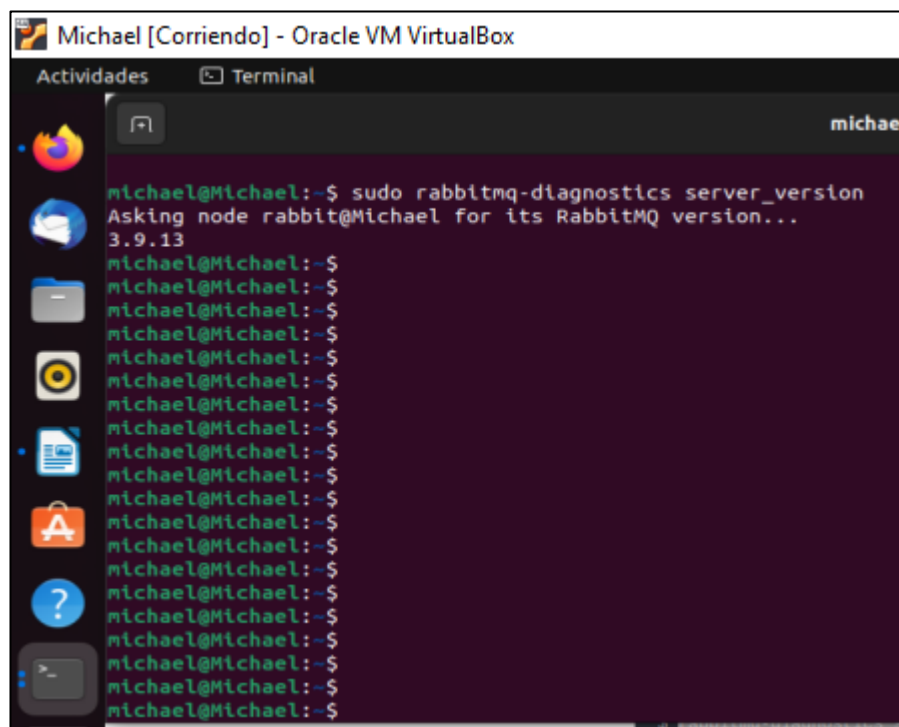
- Oracle VM Virtual Box, se instaló una máquina virtual Ubuntu.
- Visual Studio Code, con las Librerías necesarias para operar en Windows y Linux UBUNTU
- Entorno de Ejecución: Node.js Sobre el motor de JavaScript V8 de Google Chrome
- Docker Desktop, para Operación y Administración de Contenedores: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

Desarrollo

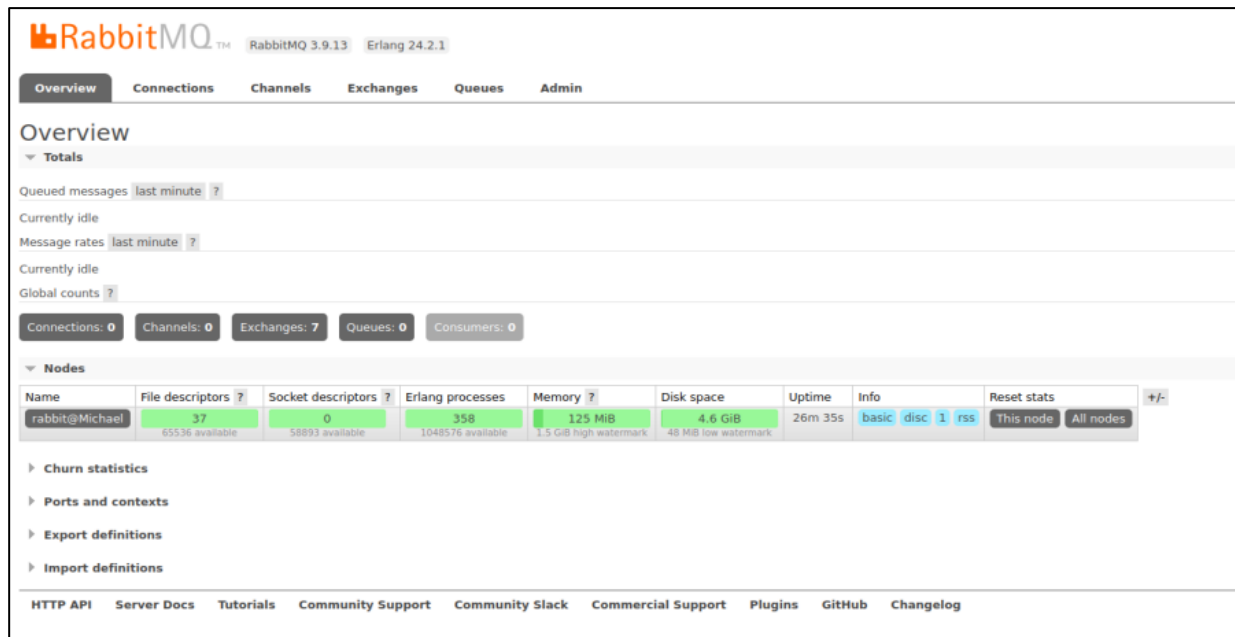
Máquina virtual Ubuntu instalada en Oracle VM VirtualBox:



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 log “server1.js”

```
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
25     req.on('end', () => {
```

Servidor de js el cual envía el archivo .log a Rabbit “envia_rabbit.js”

```
1  const http = require('http');
2  const fs = require('fs');
3  const amqp = require('amqplib');
4  const util = require('util');
5
6  // Configuración de RabbitMQ
7  const rabbitmqUrl = 'amqp://michael:michael@192.168.100.134:15672';
8  const exchangeName = 'myExchange';
9  const routingKey = 'file.upload';
10
11 // Configuración del servidor HTTP
12 const port = 3000;
13 const filePath = 'C:/xampp/htdocs/Programacion/datos.log';
14
15 const startServer = async () => {
16   const server = http.createServer(async (req, res) => {
17     if (req.method === 'POST') {
18       if (req.url === '/upload') {
19         try {
20           const fileData = fs.readFileSync(filePath);
21
22           // Conectarse a RabbitMQ y enviar el archivo
23           const connection = await amqp.connect(rabbitmqUrl);
24           const channel = await connection.createChannel();
25
```

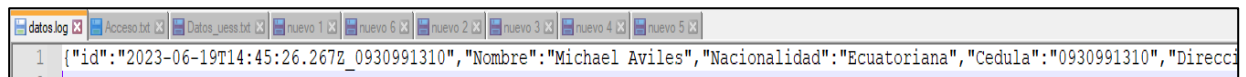

Archivo .js para verificar cola de Rabbit “cola_rabbit.js”

```

1  const axios = require('axios');
2
3  // Configuración de RabbitMQ
4  const rabbitmqUrl = 'http://192.168.100.134:15672';
5  const username = 'michael';
6  const password = 'michael';
7  const virtualHost = '/';
8  const queueName = 'myQueue';
9
10 const checkQueue = async () => {
11   const auth = {
12     username,
13     password
14   };
15
16   try {
17     const response = await axios.get(`${rabbitmqUrl}/api/queues/${encodeURIComponent(virtualHost)}/${queueName}`, { auth });
18     const queueData = response.data;
19
20     if (queueData.messages > 0) {
21       console.log(`La cola ${queueName} tiene mensajes pendientes`);
22     } else {
23       console.log(`La cola ${queueName} está vacía`);
24     }
25   } catch (error) {

```

Archivo log creado “datos.log”

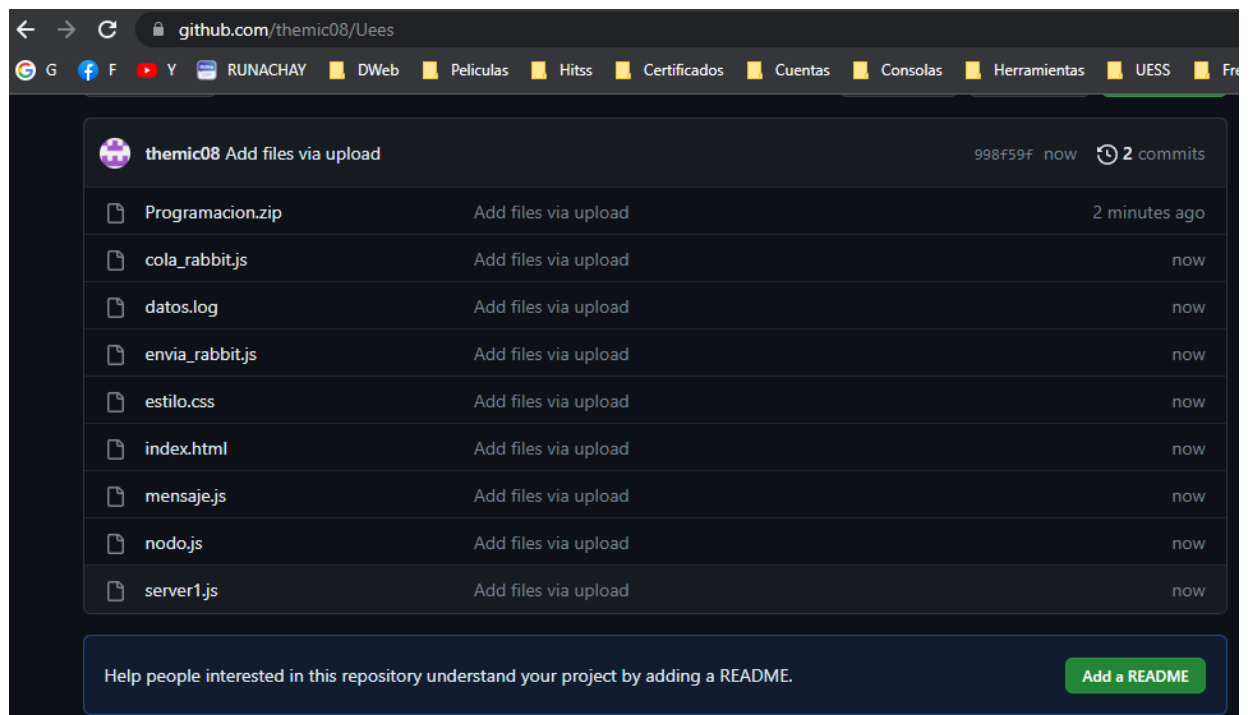


```

1  {"id":"2023-06-19T14:45:26.267Z_0930991310","Nombre":"Michael Aviles","Nacionalidad":"Ecuatoriana","Cedula":"0930991310","Direcc

```

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



github.com/themic08/Uees

themic08 Add files via upload 998f59f now 2 commits

File	Action	Time
Programacion.zip	Add files via upload	2 minutes ago
cola_rabbit.js	Add files via upload	now
datos.log	Add files via upload	now
envia_rabbit.js	Add files via upload	now
estilo.css	Add files via upload	now
index.html	Add files via upload	now
mensaje.js	Add files via upload	now
nodo.js	Add files via upload	now
server1.js	Add files via upload	now

Help people interested in this repository understand your project by adding a README. [Add a README](#)