**Universidad Mayor Real y Pontificia de**

**San Francisco Xavier de Chuquisaca**

**Facultad de Tecnología**



**Proyecto Final SIS313 - INFRAESTRUCTURA, PLATAFORMAS TECNOLÓGICAS Y REDES**

**Universitarios:   
Quispe Sucullani José David (Ing. Ciencias de la computación)**

**Fecha de Presentación: 26/06/2025**

**Sucre – Bolivia**

**1. Introducción**

El presente documento detalla el diseño, la implementación y las configuraciones clave de un sistema de gestión de usuarios desarrollado con una arquitectura cliente-servidor. Este sistema permite realizar operaciones fundamentales de bases de datos (Crear, Leer, Actualizar y Eliminar, conocidas como CRUD) sobre los registros de usuarios. Se ha utilizado MariaDB como el sistema de gestión de bases de datos relacionales, complementado con Node.js y el framework Express.js para la lógica del servidor (API). La interfaz de usuario ha sido construida con tecnologías web estándar como HTML, CSS y JavaScript nativo. Todo el entorno del proyecto ha sido desplegado y configurado dentro de una Máquina Virtual Ubuntu, simulando un entorno de servidor de producción para fines de desarrollo y prueba.

**2. Arquitectura General del Sistema y Topología de Red**

La arquitectura propuesta para este sistema se adhiere a un modelo de tres capas: presentación, lógica de negocio y datos. Esta arquitectura está completamente contenida y se ejecuta dentro de una única Máquina Virtual (VM) basada en Ubuntu, lo que permite replicar un entorno de servidor de manera controlada y eficiente.

* **Topología de Red Simplificada (Dentro de la Máquina Virtual Ubuntu):**
  + **Capa de Presentación (Cliente - Navegador):** Un navegador web (como Opera GX) es el punto de acceso para el usuario. Este navegador, residiendo en la misma Máquina Virtual, interactúa con la aplicación a través de la dirección local de http://localhost:3001. Es responsable de renderizar la interfaz de usuario y de enviar las peticiones de los usuarios a la capa de lógica de negocio.
  + **Capa de Lógica de Negocio (Servidor de Aplicaciones - Node.js/Express):** Esta capa es manejada por una aplicación desarrollada en Node.js utilizando el framework Express.js. Su función principal es escuchar las peticiones HTTP/HTTPS entrantes en un puerto específico (ej., 3001). Actúa como el intermediario entre la interfaz de usuario y la base de datos, procesando la lógica de negocio, validando los datos, sirviendo los archivos estáticos de la interfaz de usuario (HTML, CSS, JavaScript) y exponiendo una API RESTful para las operaciones CRUD.
  + **Capa de Datos (Servidor de Base de Datos - MariaDB):** MariaDB opera como el sistema de gestión de bases de datos relacionales, escuchando en su puerto estándar (3306). Es el repositorio central donde se almacena toda la información estructurada de los usuarios. La aplicación Node.js se comunica directamente con esta capa para realizar todas las operaciones de persistencia de datos.
* **Ubicación de Servicios:**
  + Todos los componentes clave del sistema (el servidor de base de datos MariaDB y el servidor de aplicaciones Node.js/Express) están ubicados y se ejecutan dentro de la misma Máquina Virtual Ubuntu.
  + La Máquina Virtual, a su vez, se ejecuta sobre un sistema operativo anfitrión (como Windows) utilizando el software de virtualización VirtualBox. Esta disposición permite un desarrollo y pruebas eficientes en un entorno aislado y replicable.

**3. Justificación de las Tecnologías y Herramientas Elegidas**

La selección de las tecnologías y herramientas para este proyecto se fundamenta en su capacidad para ofrecer un sistema robusto, eficiente, seguro y de fácil mantenimiento, aprovechando soluciones de código abierto y ampliamente reconocidas en la industria.

* **MariaDB (Sistema de Gestión de Base de Datos Relacional - SGBDR):**
  + **Justificación:** MariaDB fue elegido por su alta fiabilidad, rendimiento y su naturaleza de código abierto. Al ser un *fork* de MySQL, ofrece una compatibilidad casi total con el ecosistema de MySQL, incluyendo herramientas y conectores como mysql2 de Node.js. Proporciona una plataforma estable para el almacenamiento estructurado de datos, garantizando la integridad y consistencia de la información de los usuarios.
* **Node.js (Entorno de Ejecución de JavaScript en el Servidor):**
  + **Justificación:** La elección de Node.js facilita el desarrollo *full-stack* con JavaScript, lo que promueve la reutilización de conocimientos y acelera el proceso de desarrollo. Su modelo de E/S no bloqueante y asíncrono lo hace excepcionalmente eficiente para construir APIs rápidas y escalables, capaz de manejar múltiples conexiones concurrentes sin sacrificar el rendimiento.
* **Express.js (Framework Web para Node.js):**
  + **Justificación:** Express.js es un framework web minimalista y altamente flexible. Su diseño simplifica la creación de APIs RESTful, la gestión de rutas HTTP y la integración de middlewares, ofreciendo un control granular sobre la arquitectura de la aplicación y permitiendo un desarrollo ágil y estructurado del backend.
* **mysql2 (Paquete NPM - Conector de MariaDB/MySQL para Node.js):**
  + **Justificación:** Se seleccionó mysql2 por encima de otros conectores por sus características avanzadas, como soporte para promesas (facilitando el uso de async/await para un código asíncrono más limpio y legible), y su robustez en la gestión de conexiones a bases de datos MariaDB y MySQL. Su uso es clave para una interacción segura y eficiente con la capa de datos.
* **body-parser (Middleware de Express):**
  + **Justificación:** Este middleware es fundamental para que el servidor Express pueda interpretar y procesar los cuerpos de las peticiones HTTP, especialmente aquellos enviados en formato JSON o URL-encoded desde el frontend. Sin body-parser, los datos enviados en el cuerpo de una petición (req.body) no serían accesibles.
* **cors (Middleware de Express):**
  + **Justificación:** El middleware cors es esencial para habilitar el Intercambio de Recursos de Origen Cruzado. Permite que el frontend, que puede estar sirviéndose desde un origen diferente (aunque en este caso comparten localhost:3001), se comunique sin restricciones con la API del backend. Esta es una práctica de seguridad estándar para el desarrollo de aplicaciones web distribuidas.
* **HTML (HyperText Markup Language):**
  + **Justificación:** Como lenguaje fundamental de la web, HTML se utiliza para definir la estructura y el contenido de la interfaz de usuario. Permite crear los formularios para la entrada de datos de usuarios y las tablas para la visualización de los registros.
* **CSS (Cascading Style Sheets):**
  + **Justificación:** CSS es el lenguaje encargado de la presentación visual y el diseño de la interfaz de usuario. Se ha optado por un conjunto de estilos CSS nativos y concisos para lograr una apariencia limpia y funcional, priorizando la legibilidad y la usabilidad sin la sobrecarga de un framework CSS completo.
* **JavaScript (Frontend - Navegador):**
  + **Justificación:** JavaScript es el lenguaje que proporciona la interactividad dinámica al lado del cliente. Gestiona la comunicación asíncrona con la API de Node.js a través de fetch, manipula el Modelo de Objeto de Documento (DOM) para actualizar la interfaz en tiempo real, valida los datos del formulario y controla el comportamiento de los modales personalizados.
* **npm (Node Package Manager):**
  + **Justificación:** npm es el gestor de paquetes estándar para Node.js. Su función es crucial para la instalación, gestión y resolución de dependencias del proyecto, asegurando que todas las bibliotecas de Node.js requeridas estén disponibles y sean compatibles entre sí.
* **Ubuntu (Sistema Operativo Linux):**
  + **Justificación:** Se eligió Ubuntu como el sistema operativo base por su estabilidad, seguridad y su amplia adopción en entornos de servidor. Ofrece un entorno de desarrollo robusto con excelentes herramientas de línea de comandos y cuenta con una vasta comunidad de soporte.
* **VirtualBox (Software de Virtualización):**
  + **Justificación:** VirtualBox se utiliza para crear un entorno de servidor virtualizado y aislado. Esto permite la configuración y experimentación con servicios (como MariaDB y Node.js) sin afectar el sistema operativo principal de la máquina anfitriona, replicando un entorno de servidor dedicado de manera eficiente.
* **nano (Editor de Texto en Consola):**
  + **Justificación:** nano es un editor de texto simple y de fácil uso, preinstalado en muchas distribuciones de Linux. Es ideal para realizar ediciones rápidas y directas de archivos de configuración y código desde la terminal de Ubuntu. Su simplicidad lo hace accesible para tareas de edición en entornos de servidor sin interfaz gráfica.

**4. Plan de Implementación y Configuración de Cada Servicio**

Este plan describe el proceso de configuración y despliegue del sistema, absteniéndose de listar comandos específicos, y centrándose en las acciones a realizar.

**4.1. Configuración del Entorno Base (Máquina Virtual Ubuntu)**

El primer paso consiste en preparar el sistema operativo subyacente que albergará la aplicación.

1. **Instalación del Sistema Operativo:** Se realiza la instalación de una distribución Ubuntu (ya sea la versión Server o Desktop) dentro de VirtualBox. Posteriormente, se configura la red de la Máquina Virtual para asegurar la conectividad a Internet y, si es necesario, se le asigna una dirección IP estática para un acceso predecible desde el sistema anfitrión u otras máquinas virtuales.
2. **Actualización del Sistema:** Una vez instalado el sistema operativo, se procede a actualizar todos los paquetes y dependencias del sistema para garantizar la estabilidad y la inclusión de los últimos parches de seguridad.
3. **Instalación de Node.js y npm:** Se instalan el entorno de ejecución Node.js y su gestor de paquetes npm. Adicionalmente, se recomienda actualizar Node.js y npm a sus versiones estables más recientes para aprovechar las últimas características y mejoras de rendimiento. También se instalan herramientas de compilación esenciales que pueden ser requeridas por algunos módulos de Node.js.

**4.2. Configuración del Servidor MariaDB**

La base de datos es el componente central para la persistencia de datos del sistema de usuarios.

1. **Preparación del Entorno de Base de Datos:** Si ya existe una instalación previa de MySQL o MariaDB y se desea una configuración completamente limpia, se procede a detener el servicio, purgar todos los paquetes relacionados y eliminar los directorios de datos y configuración persistentes. Este es un paso destructivo que borra todas las bases de datos existentes.
2. **Instalación de MariaDB Server:** Se instala el paquete del servidor MariaDB. Durante este proceso, el sistema solicita establecer una contraseña para el usuario root de MariaDB, siendo fundamental definir una contraseña robusta y documentarla adecuadamente.
3. **Configuración del Servidor MariaDB como Maestro:** Se accede al archivo de configuración principal de MariaDB (mysqld.cnf) para ajustar parámetros cruciales para la replicación y la seguridad. Se asigna un server-id único para identificarlo como el maestro, se habilita el log binario (log\_bin) y se especifica que solo los cambios en la base de datos sis313 deben ser registrados para replicación. Además, se configuran límites para el tamaño y la expiración de los logs binarios. Por razones de seguridad, se configura MariaDB para que escuche conexiones solo desde localhost.
4. **Creación de la Base de Datos y la Tabla de Usuarios:** Se accede a la consola de MariaDB como usuario root. Se procede a eliminar cualquier base de datos sis313 (o tienda) preexistente para asegurar un inicio limpio. Luego, se crea la nueva base de datos llamada sis313. Dentro de esta base de datos, se define la estructura de la tabla usuarios con columnas para id, nombre, email, password y fecha\_registro, especificando las restricciones de unicidad y obligatoriedad. Finalmente, se insertan algunos registros de usuarios de ejemplo para poblar la tabla inicialmente.
5. **Reinicio del Servicio MariaDB:** Una vez completadas las configuraciones, el servicio MariaDB se reinicia para que los cambios en el archivo mysqld.cnf surtan efecto.
6. **(Opcional, para Replicación):** Se crea un usuario de replicación (repl\_user) con privilegios específicos para la replicación, sin contraseña como se solicitó. Se recargan los privilegios y se obtiene el estado actual del maestro (posición y nombre del log binario), información crucial si en el futuro se desea configurar un servidor esclavo real.

**4.3. Configuración de la Aplicación Node.js (CRUD Usuarios)**

Esta aplicación es el puente entre la interfaz de usuario y la base de datos.

1. **Preparación del Directorio del Proyecto:** Se crea un directorio específico para la aplicación (ej., crud\_usuarios) y se navega a él.
2. **Inicialización de NPM e Instalación de Dependencias:** Se inicializa un nuevo proyecto Node.js dentro del directorio, lo que genera el archivo package.json. Posteriormente, se instalan las bibliotecas express, mysql2, body-parser y cors, que son esenciales para el funcionamiento del backend. También se crea una subcarpeta llamada public que albergará los archivos del frontend.
3. **Creación de Archivos de Código:** Se procede a crear los archivos principales del proyecto:
   * app.js (Backend): Contiene la lógica del servidor Express.js, las rutas API para las operaciones CRUD en la tabla usuarios y la configuración de la conexión a MariaDB (sis313). Es vital asegurar que la contraseña del usuario root de MariaDB esté correctamente configurada en este archivo.
   * public/index.html (Frontend HTML): Define la estructura de la interfaz de usuario, incluyendo el formulario para añadir/editar usuarios y la tabla para listarlos.
   * public/style.css (Frontend CSS): Contiene las reglas de estilo CSS para dar una apariencia limpia y funcional a la interfaz.
   * public/app.js (Frontend JavaScript): Implementa la lógica del lado del cliente, incluyendo la comunicación con la API, la manipulación del DOM para la tabla y el formulario, y la funcionalidad de los modales personalizados.
4. **Inicio de la Aplicación:** La aplicación Node.js se inicia desde la terminal. Se monitorea la salida para confirmar la conexión exitosa a la base de datos y que el servidor Express.js está escuchando en el puerto configurado (ej., 3001).

**4.4. Verificación y Pruebas del Sistema**

Una vez que todos los servicios están en ejecución, se realizan pruebas para validar la funcionalidad.

1. **Acceso a la Interfaz Web:** Se abre un navegador web (como Opera GX) en la Máquina Virtual Ubuntu y se navega a la URL local de la aplicación (ej., http://localhost:3001).
2. **Pruebas CRUD Completas:**
   * **Lectura:** Se verifica que la tabla de usuarios muestre correctamente los usuarios de ejemplo insertados en la base de datos.
   * **Creación:** Se utiliza el formulario para introducir los datos de un nuevo usuario. Tras enviarlo, se confirma que el usuario aparece en la tabla y que se muestra un mensaje de éxito.
   * **Edición:** Se selecciona un usuario de la tabla para editarlo. Sus datos se cargan en el formulario, se modifican y se guardan. Se verifica que la tabla refleje los cambios.
   * **Eliminación:** Se selecciona un usuario y se utiliza la opción de eliminar. Tras una confirmación, se verifica que el usuario desaparece de la tabla.
   * **Modales:** Se confirma que todas las interacciones de usuario (mensajes de éxito/error, confirmaciones de eliminación) se presenten a través de los modales personalizados implementados, en lugar de las ventanas de alerta/confirmación nativas del navegador.

**5. Estrategias de Hardening y Optimización Aplicadas**

Para asegurar un sistema robusto, se han integrado diversas estrategias que abordan tanto la seguridad como el rendimiento.

**5.1. Hardening (Seguridad)**

1. **Contraseña Robusta para el Usuario root de MariaDB:** Se enfatiza la creación de una contraseña fuerte y única para el usuario administrativo de la base de datos durante la instalación, siendo la primera línea de defensa contra accesos no autorizados a los datos.
2. **Principio de Menor Privilegio para MariaDB:** En un entorno de producción, se implementaría la creación de un usuario MariaDB dedicado para la aplicación, con permisos estrictamente limitados a las operaciones CRUD (SELECT, INSERT, UPDATE, DELETE) y solo sobre la base de datos específica (sis313), evitando el uso del usuario root para las conexiones de la aplicación.
3. **Restricción de Acceso a MariaDB (bind-address):** La configuración del servidor MariaDB se ajusta para que solo escuche conexiones en la interfaz de localhost (127.0.0.1). Esto limita la superficie de ataque, ya que la base de datos no es directamente accesible desde la red externa, solo por aplicaciones que residen en la misma máquina.
4. **Validación de Entradas (Backend):** La lógica del servidor Node.js incluye validaciones para asegurar que los datos esenciales (nombre, email, contraseña) sean proporcionados por el frontend. Esto previene la inserción de datos incompletos o mal formados en la base de datos y es una primera barrera contra posibles ataques de inyección.
5. **Manejo de Errores Controlado (Backend):** Las respuestas de error de la API hacia el cliente son genéricas y no exponen detalles sensibles del servidor, como trazas de pila o mensajes internos de la base de datos. Los detalles completos de los errores se registran solo en la consola del servidor para fines de depuración interna.
6. **Uso de Sentencias Preparadas (Backend):** El conector mysql2 se utiliza de forma que las variables de usuario se pasan como parámetros separados en las consultas SQL. Esta técnica, inherentemente, previene la mayoría de los ataques de inyección SQL al asegurar que los datos del usuario se traten como valores y no como parte de la lógica SQL.
7. **Sustitución de alert() y confirm() (Frontend):** Se han reemplazado las ventanas de diálogo nativas del navegador por modales personalizados. Esta mejora no solo ofrece una experiencia de usuario más consistente y estética, sino que también elimina un vector potencial de *phishing* o bloqueo de interfaz que pueden explotar las alertas nativas.

**5.2. Optimización (Rendimiento y Almacenamiento)**

1. **Servicio de Archivos Estáticos (Backend):** El servidor Express.js está configurado para servir los archivos del frontend (HTML, CSS, JavaScript) directamente desde una carpeta public. Este método es eficiente para el entorno de desarrollo y aplicaciones de tamaño pequeño a mediano, ya que permite al servidor Node.js gestionar tanto la API como la entrega de activos estáticos.
2. **Selección Específica de Columnas (Backend):** En las consultas SELECT a la base de datos, se especifican explícitamente solo las columnas necesarias (ej., id, nombre, email, fecha\_registro) en lugar de usar SELECT \*. Esta práctica reduce la cantidad de datos recuperados y transferidos desde la base de datos, mejorando la eficiencia de las consultas y reduciendo la carga de red.
3. **Manejo Asíncrono de Operaciones (Backend):** Node.js, por su naturaleza asíncrona y su modelo de E/S no bloqueante, es altamente eficiente. El uso de async/await en el frontend y el manejo de callbacks en el backend para operaciones de base de datos aseguran que las peticiones a la API no bloqueen el hilo principal del servidor, permitiendo que la aplicación maneje múltiples solicitudes concurrentes y mantenga una alta capacidad de respuesta.
4. **Caché del Navegador:** Los recursos del frontend (HTML, CSS, JavaScript) son elegibles para ser almacenados en caché por el navegador del cliente. Esto significa que en visitas subsiguientes a la aplicación, el navegador puede cargar estos recursos desde su caché local en lugar de solicitarlos nuevamente al servidor, lo que resulta en tiempos de carga de página significativamente más rápidos.
5. **mysql2 sobre mysql:** La elección de mysql2 no solo se justifica por sus mejoras de seguridad, sino también por su rendimiento optimizado y su capacidad para integrarse mejor con patrones de programación asíncrona modernos (Promesas), lo que contribuye a un código más eficiente y fácil de mantener.
6. **Indexación de Clave Primaria:** La definición de id como PRIMARY KEY en la tabla usuarios automáticamente crea un índice. Esto asegura que las operaciones de búsqueda, actualización y eliminación basadas en el ID del usuario sean extremadamente rápidas, ya que la base de datos puede localizar los registros directamente.
7. **Optimización de Tipos de Datos y Restricciones:** El uso de VARCHAR(100) para nombre y email optimiza el uso del espacio de almacenamiento. La restricción UNIQUE en la columna email no solo garantiza que cada dirección de correo electrónico sea única, sino que también crea un índice implícito que acelera las búsquedas y validaciones basadas en el email.
8. **Redundant Array of Independent Disks (RAID) - Nivel 1 (Mirroring):**
   * **Justificación:** RAID 1, también conocido como "mirroring" (espejo), es una estrategia de configuración de almacenamiento que implica la duplicación de datos en dos o más discos duros físicos. Cada dato escrito en un disco se escribe simultáneamente en otro disco idéntico.
   * **Beneficios:**
     + **Redundancia de Datos:** Proporciona alta tolerancia a fallos. Si un disco falla, el sistema puede seguir operando sin interrupción utilizando la copia idéntica en el disco espejo.
     + **Recuperación Sencilla:** La recuperación de datos es directa; simplemente se reemplaza el disco fallido, y los datos se reconstruyen automáticamente desde el disco funcional.
     + **Mejora en la Lectura:** Las operaciones de lectura pueden beneficiarse al poder leerse desde cualquiera de los discos, aumentando potencialmente el rendimiento de lectura.
     + **Aplicación en Base de Datos:** Para un servidor de base de datos como MariaDB, RAID 1 es crucial. Asegura que la información de los usuarios (que es crítica y debe estar siempre disponible) no se pierda ante la falla de un solo disco, manteniendo la continuidad del servicio y la integridad de los datos.

La combinación de estas estrategias de seguridad y rendimiento crea un sistema robusto, eficiente y bien protegido, listo para escalar y operar de manera confiable en un entorno de producción.