Obligatorio Bases de Datos

**Documentación**

**Constanza Bonino, Sebastián Freire**

Contenido

[Instructivo 2](#_Toc202267798)

[Clonación del repositorio 2](#_Toc202267799)

[Base de datos 2](#_Toc202267800)

[Correr la aplicación localmente 2](#_Toc202267801)

[Layout 3](#_Toc202267802)

[Informe 4](#_Toc202267803)

[Tecnologías utilizadas 4](#_Toc202267804)

[Bitácora de trabajo 4](#_Toc202267805)

[Mejoras y extensiones propuestas 5](#_Toc202267806)

[Referencias 6](#_Toc202267807)

# Instructivo

## Clonación del repositorio

En Símbolo del Sistema, navegue hasta la carpeta en la que quiera clonar el repositorio y pegue la siguiente línea:

“*git clone* [*https://github.com/sebastian-freire/Obligatorio-BD.git*](https://github.com/sebastian-freire/Obligatorio-BD.git)”

Una vez clonado el repositorio ingrese al proyecto con su IDE de confianza.

## Base de datos

Para poder tener los datos cargados al empezar a correr el programa, se deberá correr el archivo llamado *CafesMarloy.sql* en DataGrip. Para esto, una vez abierto el archivo y colocadas las credenciales correspondientes, se hará click en el botón  en la esquina superior izquierda del programa. Una vez corrido el archivo, se podrá avanzar a la aplicación

## Correr la aplicación localmente

Para poder completar este proceso se deben cumplir con tres pasos:

1. Instalar dependencias: este comando leerá el archivo package.json del proyecto y descargará e instalará todas las dependencias listadas por el mismo. Para lograr esto de deberá abrir una terminal y correr el comando:

*npm i*

1. Correr el backend: este será quien conecte la aplicación a la base de datos.

Para ello se deberá escribir los siguientes comandos en la terminal:

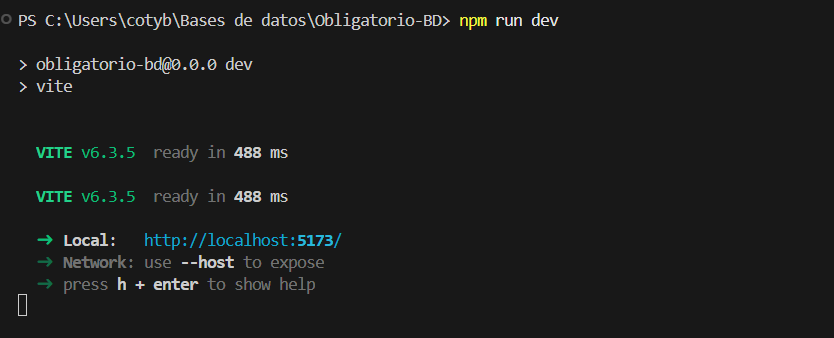
*cd backend*

*python api.py*

1. Iniciar el frontend: para poder utilizar el programa, en otra terminal (nunca cortando la ejecución de el backend), se escribirá el siguiente comando:

*npm run dev*

Se obtendrá algo así:

****

A este punto, podremos acceder al programa utilizando el link en azul, copiándolo en un navegador web o haciendo ctrl+click sobre el mismo.

# Layout

* Log In
* MenuPage: en esta página se listarán todas las tablas a las que se podrá acceder. Dependiendo del rol que se tenga como usuario serán las opciones accesibles: los usuarios normales (no administradores) tendrán acceso a las tablas de Clientes, Insumos y Mantenimientos, mientras que los usuarios administradores podrán acceder a esas además de a Proveedores, Máquinas y Técnicos.
  + Clientes: se podrán encontrar 3 tablas repartidas en 3 pestañas. La primera listara todos los clientes, donde no solo se podrán editar y eliminar los clientes listados sino que también agregar clientes nuevos.

En la segunda pestaña podremos ver una tabla listando a los clientes que tengan maquinas, con sus montos a pagar por consumos, alquiler de máquinas y el total a pagar (suma de las dos anteriores).

En la última pestaña podremos ver listados aquellos clientes que tienen la mayor cantidad de máquinas instaladas.

* + Insumos: se podrán encontrar 2 pestañas con diferente información. La primera listara todos los insumos, donde no solo se podrán editar y eliminar los insumos listados sino que también agregar insumos nuevos.

En la segunda pestaña habrá una tabla que contendrá a los insumos con más consumidos.

* + Mantenimientos: se podrá observar una tabla con el listado de los mantenimientos, junto con todos sus detalles. Se podrán editar y eliminar cada uno de estos mantenimientos, al igual que agregar nuevos mantenimientos.
  + Máquinas: se listan todas las maquinas registradas junto con su información pertinente. Desde aquí se podrán añadir nuevas máquinas y editar o eliminar aquellas listadas.
  + Proveedores: se podrá observar una tabla con el listado de los proveedores, junto con todos sus detalles. Se podrán editar y eliminar cada uno de estos proveedores, al igual que agregar nuevos proveedores.
  + Técnicos: se podrán encontrar 2 pestañas con diferente información. La primera listara todos los técnicos, donde no solo se podrán editar y eliminar los técnicos listados sino que también agregar técnicos nuevos.

En la segunda pestaña habrá una tabla que contendrá a los técnicos con más mantenimientos realizados.

# Informe

## Tecnologías utilizadas

Para la creación de la base de datos se utilizó el sistema de gestión de bases de datos relacionales MySQL, donde se almacenaron todas las tablas con sus datos pertinentes. Esta permite un fácil acceso a los datos por medio de queries desde el backend.

La tecnología que se dicidió usar para levantar el servidor fue Flask, un framework que hace posible levantar un servidor web local de forma muy sencilla. Permite la creación de rutas o endpoints que llaman a funciones específicas conocidas como funciones de vista, las cuales se encargan de procesar la solicitud y devolver una respuesta. También se utilizó mysql.connector, una librería oficial de MySQL para Python que permite la conexión a bases de datos MySQL y la ejecución de consultas.

La autenticación de usuarios se utilizó la restricción GRANT en la base de datos, una declaración que otorga permisos a usuarios o roles sobre ciertos objetos de la base de datos. Esto nos permitió diferenciar los permisos de administradores de los de un usuario común.

Para el frontend se decidió sobre React web + JavaScript. Ambos integrantes del equipo, en simultaneo a Base de Datos, se encuentran cursando la materia Desarrollo web y Mobile, donde pudieron aprender a utilizar estas herramientas. Con esto en cuenta, se vio la oportunidad de aplicar los conocimientos y reforzarlos en este proyecto. Por la misma razón, fue fácil la utilización de la API fetch para las peticiones a el backend. Esta API proporciona una interfaz que permite hacer peticiones HTTP a el servidor, con solicitudes GET, POSY, PATH, etc.

Para los avisos de errores, éxito y confirmación implementamos la librería de React llamada React Hot Toast. Esta permite mostrar toasts (notificaciones emergentes, temporales) de forma simple y personalizable.

En relación con los estilos de la página, se implementaron archivos CSS que permitieron compartir los estilos entre páginas con displays similares, pudiendo tener una estética coherente para toda la aplicación. En esta área, inteligencias artificiales como ChatGPT o Gemini fueron de gran ayuda al equipo para alcanzar el nivel visual que se esperaba.

## Bitácora de trabajo

Para la realización del trabajo, el equipo tomó la decisión de trabajar por separado con el fin de ser más eficientes. Primero que nada se creó la base de datos con sus correspondientes tablas en equipo, para poder tener ambos en cuenta los atributos, restricciones y reglas de integridad que iban a tener que seguirse en la creación del programa. Luego, se separaron las tareas: Sebastián se encargaría del backend con python, mientras que Constanza del frontend con React + JavaScript.

### Backend

En las primeras semanas, Sebastián fue quien tuvo un mayor avance, ya que facilitaría el trabajo de hacer los fetchs en front teniendo el backend terminado. Lo primero que se decidió investigar fue como levantar un servidor simple de Python. Se comenzó a investigar acerca de cursores, los cuales son objetos que permiten interactuar con la base de datos desde un lenguaje de programación, en nuestro caso Python. Los cursores permiten ejecutar sentencias SQL y leer los resultados de una consulta.

Después de aprender a levantar el servidor con Flask y conectar la base de datos con mysql.connector, se crearon los primeros endpoints con sus respectivas funciones, para comenzar con las primeras pruebas de conexión. Dentro de esas funciones se usaron cursores para ejecutar consultas SQL y manejar los datos de la bitácora.

Se integró Flask-CORS para permitir que el frontend pueda comunicarse con el backend sin problemas de seguridad por solicitudes entre dominios distintos.

Aunque al principio parecía manejable tener todo el código en un solo archivo, rápidamente se hizo evidente que esa forma no era cómoda ni escalable. Por eso, se decidió modularizar el proyecto usando Blueprints para organizar mejor las rutas y facilitar el mantenimiento. Los Blueprints son una forma de organizar y dividir el código de una aplicación en módulos o componentes separados. En lugar de tener todas las rutas en un solo archivo los blueprints permiten agrupar rutas, funciones y recursos relacionados en archivos independientes.

Luego, se integró un archivo dedicado a auth con funciones decoradoras que aseguraban que los usuarios estuvieran autenticados correctamente antes de acceder a ciertos endpoints, evitando repetir la lógica de autenticación en cada función.

### Frontend

Una vez finalizadas las conexiones, comenzó el trabajo visual. Para la navegación entre paginas se utilizó react-router-dom, un npm package que permite el ruteo dinámico en la aplicación, en App.jsx. Esto nos permite poder cambiar la url en el navegador web y poder navegar a la pagina indicada.

Se comenzó por los listados de todas las distintas áreas, con sus respectivas pestañas de edición y creación, poniendo también la opción de agregar nuevos ítems a las tablas. Para ello se hizo una carpeta con cada área (Clientes, Mantenimientos, Insumos, etc.), donde se encontraba una page para el listado, la edición y el agregado de un ítem de la respectiva área. El ensamble de todas estas páginas se implementó en MenuPage donde, dependiendo de si el usuario es admin o no, este podría visualizar y acceder a sus áreas permitidas.

Se decidió tener una carpeta aparte para los hooks, para poder separar la lógica de lo visual. En ellas tenemos archivos como useClientes, useMaquinas, etc. donde tenemos la funcionalidad de alcance al backend. Se pueden encontrar funciones de solicitudes GET, POST, PATCH, DELETE a partir de la API fetch, usando como endpoint aquellos planteados en el backend.

Luego de terminado el layout base y principal del programa, se decidió a enfocarse en las consultas más específicas. Para esto el equipo tuvo que plantearse como navegar a las mismas y de que manera se mostrarían. Se esperaba una forma de acceder eficiente, práctica y coherente para el usuario. Este nunca esperaría ver los clientes con más máquinas accediendo desde los Insumos, ni queríamos que tuviera que navegar múltiples pages buscando el resultado. Por esto es por lo que se optó por un diseño de pestañas, facilitando así la búsqueda de los resultados. A partir de este punto, la aplicación tomó un gran salto en relación con los estilos, llegando a la estética que tenemos ahora.

Por último, Sebastián tomo el trabajo del Log In de usuario y autenticación de este. Fue en este punto donde estuvo nuestro mayor reto.

### El conflicto con la autenticación

…

## Mejoras y extensiones propuestas

1. Que los administradores tengan el poder de añadir nuevos usuarios.
2. Poder agregar nuevas consumiciones

## Conclusión

Como equipo nos sentimos muy orgullosos del trabajo entregado. Es un programa completo, que cumple con todos los puntos planteados en el Obligatorio. Logramos aprender muchas cosas con él, siendo este el primero programa con el que unimos un a base de datos, un backend y un frontend.

# Referencias

API Fetch. (s.f.). Obtenido de https://developer.mozilla.org/es/docs/Web/API/Fetch\_API/Using\_Fetch

CSS. (s.f.). Obtenido de https://developer.mozilla.org/en-US/docs/Web/CSS

Flask. (s.f.). Obtenido de https://flask.palletsprojects.com/en/stable/

GitHub. (s.f.). GitHub Copilot. Obtenido de https://github.com/features/copilot

Google. (s.f.). Gemini. Obtenido de https://gemini.google.com/app

JavaScript. (s.f.). Obtenido de https://developer.mozilla.org/es/docs/Web/JavaScript

MySQL. (s.f.). Obtenido de https://dev.mysql.com/doc/

OpenAI. (s.f.). ChatGPT. Obtenido de https://openai.com/index/chatgpt/

Python. (s.f.). Obtenido de https://docs.python.org/3.13/

React. (s.f.). Obtenido de https://es.react.dev/

React Hot Toast. (s.f.). Obtenido de https://react-hot-toast.com/docs/toast