
Escuela de Ingeniería en Computadores
CE-3101
Bases de Datos
Grupo 1

# Documentación Técnica - OperaCE

Profesor:
Marco Rivera Meneses

Estudiantes:
Dylan Garbanzo Fallas (2021057775)
Alejandra Rodríguez Castro (2021131070)
Carlos Eduardo Rodríguez Segura (2022437835)
Ricardo Borbón Mena (2021132065)
José María Vindas Ortiz (2022209471)

I Semestre 2024

# Contenidos

# Introducción

En vista a las necesitades de los operadores encargados de los laboratorios de Ingeniería en Computadores se identificó la necesidad de emplear una herramienta de reservación de activos y laboratorios que surge como respuesta a la creciente demanda de uso tanto de las aulas como de los activos disponibles.

Para agilizar y simplificar este proceso, se ha decidido implementar un sistema que permita a diferentes usuarios, tales como administradores, operadores y profesores, gestionar eficientemente las reservaciones y préstamos. La herramienta contendrá tres vistas principales: Administrador, Operador y Profesor, cada una diseñada para satisfacer las necesidades específicas de los usuarios correspondientes.

La vista del Administrador estará centrada en la configuración y gestión del sistema, permitiendo el control completo sobre los laboratorios, activos y servicios proporcionados. Esto incluye la capacidad de agregar, modificar o eliminar información de laboratorios y activos, así como la gestión de los usuarios y la aprobación de operadores.

Por otro lado, la vista del Profesor se enfocará en la solicitud de reservaciones de laboratorio y préstamos de activos. Los profesores podrán buscar activos y laboratorios disponibles, solicitar reservas y también tienen la opción de aprobar préstamos pendientes que estén asignados a ellos.

Finalmente, la vista del Operador estará destinada a la gestión directa de las reservaciones y préstamos. Los operadores podrán registrar solicitudes de reservaciones de laboratorios por parte de estudiantes, así como gestionar préstamos y devoluciones de activos, garantizando el correcto funcionamiento del sistema en el día a día.

En resumen, esta herramienta ofrecerá una solución integral para la gestión de reservaciones y préstamos de activos y laboratorios, mejorando la eficiencia y organización en el entorno de trabajo de la empresa.

# Objetivos del Proyecto

## Objetivo General

Desarrollar una aplicación que permita la administración efectiva de los laboratorios de la escuela de Computadores brindando una gestión efectiva de activos y aulas para los usuarios que utilicen este servicio.

## Objetivos Específicos

➔ Aplicar los conceptos del modelo conceptual y relacional.

➔ Crear una Base de Datos relacional en Microsoft SQL Server para que permita el almacenamiento de los datos.

➔ Crear un servicio API para que centralice la funcionalidad.

➔ Crear una aplicación móvil utilizando SQL Lite como Base de datos empotrada.

➔ Crear una página Web para que exponga la funcionalidad al usuario.

# Descripción de los Componentes

## Arquitectura Utilizada

La arquitectura del proyecto se estructura en torno a la interacción entre tres componentes principales. En primer lugar, se encuentran la página web y la aplicación móvil, que sirven como interfaces de usuario para acceder a las funcionalidades del sistema. Estas interfaces están diseñadas para ofrecer una experiencia de usuario intuitiva y fluida.

El segundo componente es la API REST o Web Service, que actúa como un intermediario entre los clientes (página web y aplicación móvil) y la base de datos. Esta API proporciona procesos que permiten a los clientes realizar diversas operaciones, como crear, leer, actualizar y eliminar datos. Estos servicios están diseñados siguiendo los principios RESTful para garantizar una comunicación eficiente y escalable.

Finalmente, la base de datos es el repositorio central de datos del sistema. Aquí se almacena toda la información relevante, como detalles de laboratorios, activos, usuarios y registros de actividad. La base de datos está diseñada para ser robusta y escalable, permitiendo un fácil acceso y manipulación de los datos mediante consultas SQL.

Esta arquitectura ofrece varios beneficios. En primer lugar, proporciona escalabilidad, ya que cada componente puede escalar de forma independiente según las necesidades del sistema. Además, la modularidad de la arquitectura facilita el mantenimiento, permitiendo la identificación y corrección de errores de manera eficiente.

Por último, la arquitectura garantiza la seguridad de los datos, ya que la API REST actúa como un punto de acceso seguro a la base de datos, permitiendo implementar medidas de seguridad como autenticación y autorización a nivel de API.

## Base de Datos

Para realizar de forma satisfactoria la construcción de una base de datos que cumpla los requisitos del problema planteado y sea capaz de realizar solicitudes acordes con la

información requerida para cada una de las funciones de los usuarios. Seguiremos una metodología bien definida.

Lo primero que realizaremos será crear un diagrama conceptual utilizando notación de Chen que nos dará una idea general de la estructura de la base. A continuación, tomaremos este diagrama y lo transformaremos a una representación relacional que muestre la estructura de las tablas que tendrá la base de datos final.

## Modelo Conceptual

Primero debemos de identificar las entidades fuertes, es decir las que poseen una llave distintiva para cada registro. Entre esas encontramos los *usuarios* que engloban tanto a los administradores, profesores y operadores. Otra entidad fuerte relacionada es el *rol* de los usuarios, las últimas corresponden a los *laboratorios* y los *activos*. Para cada uno de ellos representamos sus atributos haciendo la salvedad de que las facilidades de los laboratorios es un atributo multivaluado y los nombres son atributos compuestos.

Ahora representamos las entidades débiles, comenzando por el *registro de horas* de los operadores que depende de usuario. Además, encontramos las *solicitudes* de laboratorios y activos que dependen de las clases que poseen el mismo nombre. Para estas colocamos las relaciones de dependencia con sus padres.

Por último, terminamos de realizar las relaciones restantes entre usuario-rol, usuario-soliLab, usuario-soliAct y laboratorio-activo; establecemos su cardinalidad y participación para así terminar de representar la conducta de la situación de la vida real que estamos abstrayendo. El resultado lo podemos observar a continuación:

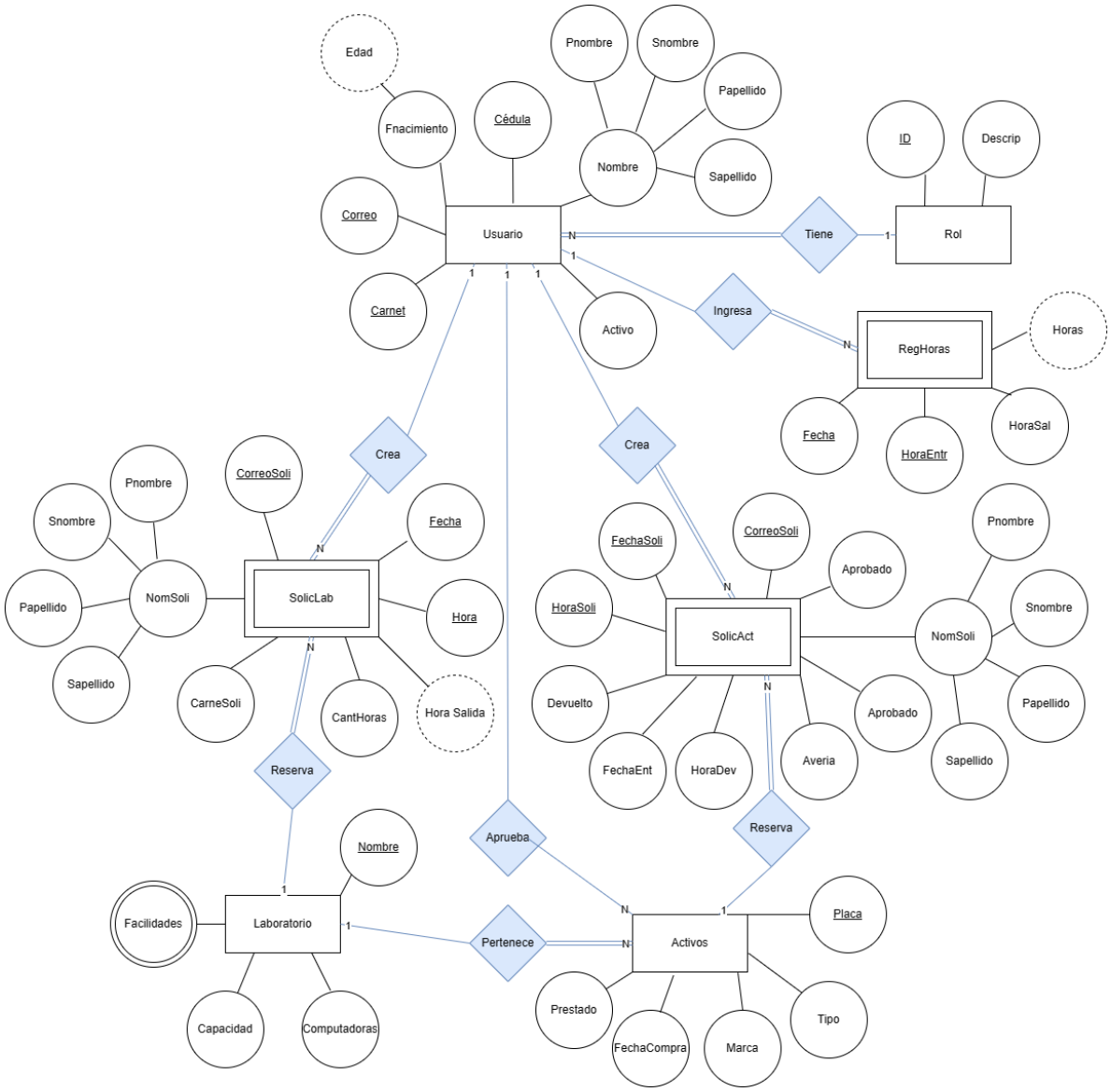


Figura 1. Diagrama conceptual del problema utilizando la notación de Chen.

A continuación, tomaremos este diagrama y lo trasladaremos a un modelo más técnico que muestra la distribución en tablas de la información de la base

## Modelo Relacional

De manera análoga primero representamos en tablas las entidades fuertes recalcando su llave, colocando su información y dejando de lado los atributos multivariados. En forma similar representamos las entidades débiles e incluimos como llave foránea su entidad padre.

Luego de esto, procedemos a tratar las relaciones entre las entidades. Para nuestro contexto solo existen relaciones *1 a N*, por lo tanto, debemos de colocar la llave primaria del lado del *1* como llave foránea en la entidad del lado *K*. Todas estas relaciones y llaves las podemos observar representadas por las flechas en el diagrama a continuación

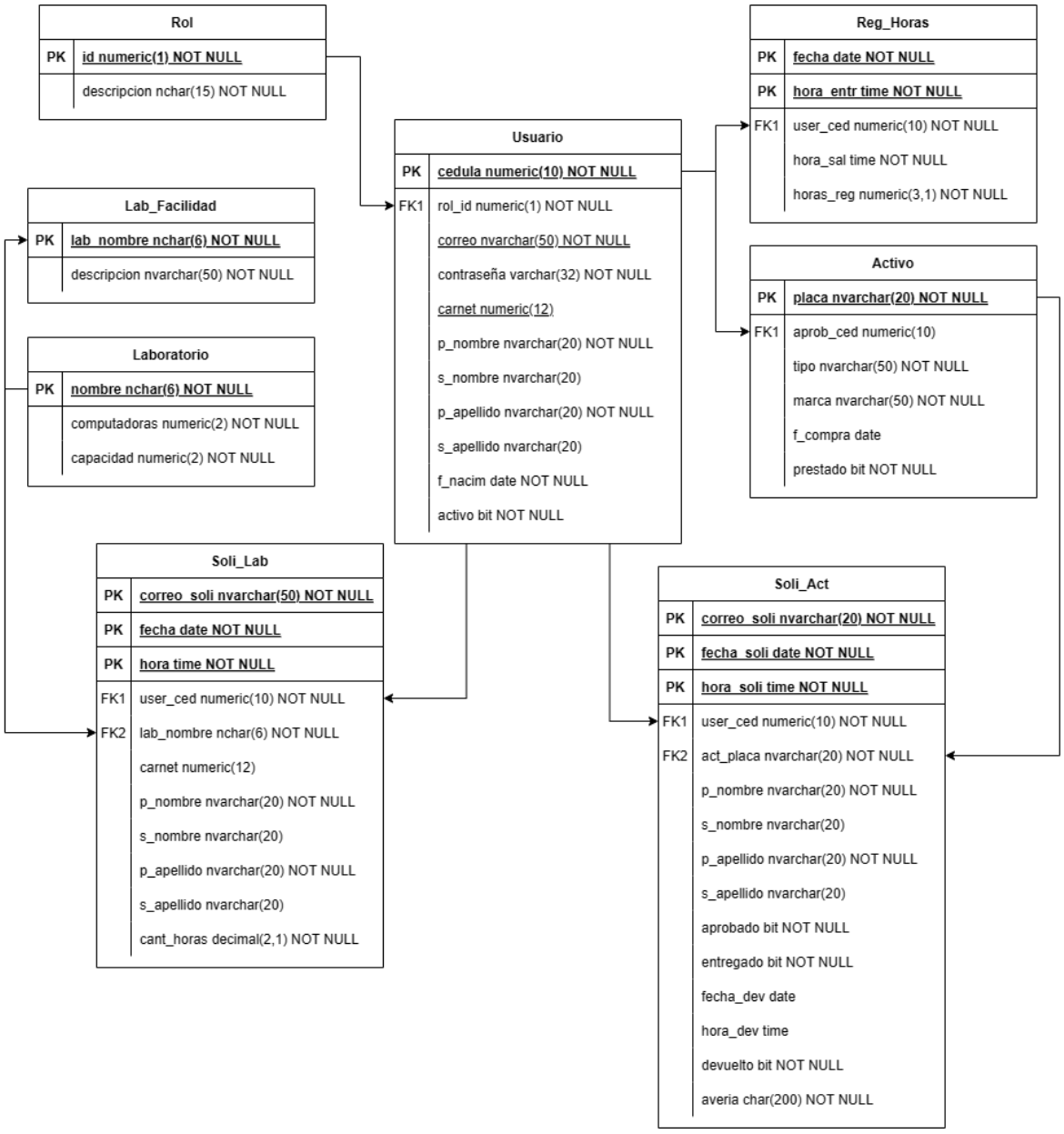


**Figura 2.** Diagrama Relacional de la base de datos implementada para OperaCE.

## Estructuras de Datos

A continuación, explicaremos de forma detallada las estructuras y constitución de las tablas utilizadas. Antes de iniciar tenemos que mencionar que la base usada para realizar la base de datos relacional de SQL y usamos como motor el entorno de desarrollo SQL Server.

Primero iniciaremos con el usuario, como llave principal usamos la cédula, además almacenamos su información personal y su contraseña cifrada usando el algoritmo “md5”. Cada usuario posee un id del rol al que pertenece (1 = Administrador, 2 = Profesor y 3 = Operador) para identificar sus funciones dentro del sistema.

Los usuarios operadores pueden marcar su turno lo que almacena en su registro de horas la fecha del turno, la hora de entrada, la hora de salida y la cédula del operador que esta almacenando el registro del turno.

Los usuarios pueden realizar solicitudes de laboratorios, estas solicitudes tienen como llave el solicitante, la fecha y la hora. Y almacenan la cantidad de horas de la solicitud, la información del solicitante, el nombre del laboratorio que se está reservando y el operador que está procesando el préstamo.

De forma análoga, los usuarios también pueden realizar solicitudes de activos, que posee más información a almacenar, correspondiente a los datos del solicitante, la fecha y hora de la solicitud, banderas que almacenan el estado del activo (aprobado, entregado, devuelto), la placa del activo que se está solicitando y el operador que está facilitando el préstamo. Luego cuando se devuelva el activo almacenamos la fecha hora de devolución y un reporte de averías.

Ahora hablando de los recursos del edificio de computadores, encontramos las tablas de laboratorios y activos. Los laboratorios guardan su nombre como llave principal, su capacidad, cantidad de computadores y las facilidades se almacenan en una tabla de búsqueda aparte. Por último, los activos usan una placa como llave primaria, almacenamos además su tipo, su marca, su fecha de compra y la cédula del profe que aprueba su préstamo.

Con esto finalizamos la explicación de la estructura, composición y relaciones entre las tablas utilizadas en el proyecto.

## API / Web Service / Rest Service

Para la API, se utiliza C# / .NET Core para su progamación, y para su testeo, se utiliza la herramienta Swagger enviando jsons o información en el mismo URL, dependiendo de la cantidad de información que el procedimiento de la base da datos necesita.

Primero, en la carpeta API se encuentran diversas carpetas, en la carpeta Controlador, se encuentran todos los controladores que utiliza la API, en la carpeta Recursos, se encuentran la mayoría de clases utilizadas por los controladores (algunas de estas clases se encuentran en los mismos controladores) y finalmente la carpeta APIPublicada, la cual cuenta con la API publicada para así ser utilizada en servicios como IIS.

La estructura base de cada controlador es la misma para todos, primero, todos los controladores son clases los cuales están conectados a una ruta http, y para diferenciarlos entre si, se usa el nombre de la clase como parte de la ruta http. Además se usa un string privado el cual cuenta con la información necesaria para poder conectarse a la db y poder ejecutar los procedimientos con la información recibida por la página web o la aplicación móvil.

Seguidamente cuenta con un try and catch, el cual, realiza la conexión a la base de datos, si dicha conexión es exitosa, se instancia el comando del procedimiento de la base de datos, con un json el cual contiene la información necesaria para el procedimiento. Seguidamente se ejecuta dicho comando, y se devuelve un true (200) si el procedimiento se ejecutó de forma correcta, o se devuelve el error dado por la base de datos.

## Aplicación Web

Lo aplciación web es donde tanto administradores, profesores y operadores, hacen su trabajao principal. En la **vista de administradores**, estos pueden:

- Gestionar información de los laboratorios y los activos.
- Gestionar a los profesores, ya sea creando sus cuentas, eliminándolos o editando su información.

- Aprobar o rechazar solicitudes para ser operador.
- Restableces la contraseña de tanto operadorees, administradores o profesores.
- Generar reportes, para ver el avance de las horas trabajadas de los operadores.

**Los profesores pueden**:

- Aprobar prestamo de activos de estudiantes.
- Hacer solicitudes para reservar laboratorios.
- Cambiar su contraseña.

Finalmente, **la vista de operadores puede**:

- Reservar laboratorios, para un estudiante que haga la solicitud.
- Prestamo de activos a profesores, como controles de pantallas o controles de los proyectores.
- Prestamo de activos a estudiantes.
- Devolución de activos, sea con averia o sin averia.
- Ver su reporte de horas, en la vista, y además generar el pdf.

Y para su elaboración se utilizó react native. Siendo su estructura principal. App.js, donde se le dan las rutas y la protección de rutas necesarias. Ahí, están el llamado a todas las vistas necesarias para la aplicación. Dentro de una carpeta llamada components, está todo lo llamado por app.js. Ahí en cada carpeta llamada como su vista, están los archivos jsx y css, con su respectiva lógica.

## Componentes Principales:

**AdminPage:** encargada de toda la lógica de la vista de administrador antes mencionada. **LoginPage:** encargada del logueo del usuario.

**OperatorPage:** encargada de todas las funcionalidades de la vista de operador. **RegisterPage:** encargada de registrar a operadores.

**LabsEst:** encargada del formulario para reservar el laboratorio por parte de un estudiante.
**LabsPro:** encargada del formulario para reservar el laboratorio por parte de un profesor.
**LabsPage:** encargada de observar todas las reservas de x laboratorio.

Y finalmente

**ProfesorPage**: encargada de toda la funcionalidad de la vista de profesor. Se tiene un css principal, para tener toda la lógica necesaria ahí, pero cada vista tiene su css, para cambios especificos de la página.

Cabe mencional la importancia de **assets**, que contiene todas las imagenes y anexos necesarios para que corra la aplicación.

## Aplicación Móvil

La aplicación móvil del proyecto es una plataforma diseñada para profesores. Les permite realizar tres tareas principales:

1. Gestionar solicitudes de activos.
2. Reservar laboratorios.
3. Cambiar la contraseña de su cuenta.

El proyecto está desarrollado utilizando React Native, una biblioteca de JavaScript para construir interfaces de usuario móviles. La estructura del proyecto se organiza de la siguiente manera:

- **Screens** Esta carpeta contiene los componentes individuales de las pantallas que conforman la interfaz de usuario de la aplicación. Estos componentes están implementados en JavaScript.
- **App.tsx** Este archivo es el punto de entrada de la aplicación. Configura la navegación y proporciona la estructura principal del flujo de la aplicación.
- **updateDB.js** Este código es una aplicación de React Native que maneja una base de datos SQLite local y la sincroniza con un servidor remoto. Sus principales funcionalidades son:

    1. Configuración de la base de datos**:**
        - Se crea una base de datos SQLite local llamada "MainDB".
        - Se definen las estructuras de las tablas que se utilizarán.
    2. Actualización de la base de datos**:**

    - Cuando el dispositivo tiene conexión a Internet, se actualizan los datos de la base de datos local obteniendo información del servidor remoto.
3. Obtener datos del servidor**:**
    - Se realizan solicitudes HTTP al servidor para obtener los datos necesarios.
4. Actualizar tablas de la base de datos**:**
    - Los datos obtenidos del servidor se almacenan en las tablas correspondientes de la base de datos local.
5. Sincronizar con el servidor**:**
    - Después de actualizar la base de datos local, se envían los datos de ciertas tablas al servidor utilizando solicitudes HTTP.

## Funcionalidades Principales:

- **Inicio de sesión (LoginScreen):** Permite a los usuarios autenticarse en la aplicación ingresando su correo electrónico y contraseña. Utiliza una base de datos SQLite local para verificar las credenciales del usuario. Si las credenciales son válidas, muestra un mensaje de éxito, almacena un identificador de cliente y redirige al usuario a la pantalla principal (HomeScreen). Si las credenciales no son válidas, muestra un mensaje de error. La interfaz de usuario consta de un fondo de imagen, un título, campos de entrada para el correo electrónico y la contraseña, y un botón de "Iniciar Sesión".
- **Página principal (HomeScreen):** Es la pantalla principal de la aplicación. Muestra un fondo de imagen y un título "Inicio". Contiene cuatro botones: "Ver solicitudes de préstamos" que redirige a la pantalla LoansScreen, "Solicitar laboratorio" que redirige a la pantalla LaboratoriesScreen, "Cambiar contraseña" que redirige a la pantalla PasswordChangeScreen y "Cerrar sesión" que limpia el identificador de cliente (clientID) y redirige al usuario a la pantalla de inicio de sesión (LoginScreen).
- **Pantalla de solicitudes de préstamo (LoansScreen):** Muestra una lista de solicitudes de préstamo de activos. Al cargar la pantalla, se realiza una consulta SQL para obtener las solicitudes de préstamo de la base de datos SQLite. Cada solicitud de préstamo muestra detalles como: fecha, placa del activo, tipo de activo, nombre del solicitante y nombre del operador. Para cada solicitud de préstamo, se muestran dos botones: "Aprobar" y "Rechazar". Al presionar el botón "Aprobar", se ejecuta la función `approveLoan` que actualiza el estado de la solicitud a "aprobada" en la base de datos. Al presionar el botón "Rechazar", se ejecuta la función `rejectLoan` que actualiza el estado de la solicitud a "rechazada" en la base de datos. La interfaz de usuario consta de un fondo de imagen, un título "Solicitudes de préstamos", y un ScrollView que contiene la lista de solicitudes de préstamo.
- **Pantalla de laboratorios disponibles (LaboratoriesScreen):** Muestra una lista de los laboratorios disponibles. Al cargar la pantalla, se ejecuta la función `fetchLaboratories` que realiza una consulta SQL para obtener la lista de laboratorios de la base de datos SQLite. Cada laboratorio muestra detalles como: nombre, capacidad y cantidad de computadoras. Cada elemento de la lista de laboratorios es un componente `TouchableOpacity`, lo que permite al usuario presionar sobre un laboratorio específico. Al presionar un laboratorio, se ejecuta la función `navigateToLabsSchedule` que navega a la pantalla `LabAvailabilityScreen` pasando el nombre del laboratorio seleccionado como parámetro. La interfaz de usuario consta de un fondo de imagen, un

título "Laboratorios Disponibles", y un `ScrollView` que contiene la lista de laboratorios.

- **Pantalla de disponibilidad de laboratorio (LabAvailabilityScreen):** Muestra la disponibilidad semanal de un laboratorio específico. Recibe el nombre del laboratorio como parámetro de la pantalla anterior (LaboratoriesScreen). Utiliza la librería moment para el manejo de fechas. Obtiene las reservaciones del laboratorio para la semana actual desde la base de datos SQLite. Muestra un título con el nombre del laboratorio. Muestra el rango de fechas de la semana actual. Cada día se muestra en una columna con los espacios de tiempo disponibles en verde y los reservados en rojo. Permite reservar un espacio de tiempo disponible presionándolo. Al reservar un espacio, navega a la pantalla ReservationScreen pasando los detalles del laboratorio, fecha y hora seleccionados. Incluye botones para cambiar a la semana siguiente o anterior. Muestra un mensaje de error si se intenta navegar a una semana sin datos disponibles.
- **Pantalla de reservación (ReservationScreen):** Permite confirmar una reservación de laboratorio. Recibe los detalles del laboratorio, fecha y hora seleccionados desde la pantalla anterior (LabAvailabilityScreen). Muestra la información del laboratorio. Utiliza un Picker para permitir al usuario seleccionar la cantidad de horas para la reservación. Al cargar la pantalla, obtiene los datos del usuario actual desde la base de datos SQLite. Cuando se presiona el botón "Confirmar Reservación": formatea la fecha y hora para el formato adecuado en la base de datos, inserta una fila en la tabla Soli_Lab de la base de datos con los detalles de la reservación y los datos del usuario, si la operación es exitosa, muestra un mensaje de confirmación y navega a la pantalla principal (HomeScreen). La interfaz de usuario consta de un fondo de imagen, un contenedor con la información de la reservación, un Picker para seleccionar la duración, y un botón para confirmar la reservación.
- **Pantalla de cambio de contraseña (PasswordChangeScreen):** Permite al usuario cambiar su contraseña. Muestra campos de entrada para la contraseña actual, la nueva contraseña y la confirmación de la nueva contraseña. Al intentar cambiar la contraseña: obtiene el ID del cliente actual utilizando la función `getClientId`. Realiza una transacción en la base de datos SQLite, verifica si la contraseña actual ingresada coincide con la contraseña almacenada para el usuario, si todo está correcto, actualiza la contraseña del usuario en la base de datos. La interfaz de usuario consta de un fondo de imagen, un título "Cambiar contraseña", campos de entrada para las contraseñas y un botón para confirmar el cambio de contraseña.

# Problemas encontrados:

Para la base de datos:

- No se puede realizar la eliminación de profesores, laboratorios y activos que estén relacionados con otras tablas de la base, es decir que existan como llaves foráneas en otras tablas. Esto es una característica asociada a las bases de datos relacionales. Para solucionarlo se debe evitar de eliminar registros.

Para la página web:

- No se han encontrado problemas

Para la aplicación móvil:

- En caso de que un profesor niegue la solicitud de un activo, este registro de solicitud le seguirá apareciendo al profesor para que lo apruebe (aunque ya lo rechazó previamente) debido a un limitación de base de datos.

Para la API:

- Las casillas que pueden recibir nulo como un valor válido en la base de datos no funcionan correctamente a la hora de validar una casilla que efectivamente no contenga nada.

# Problemas conocidos:

Para la base de datos:

- En caso de querer modificar que un activo pueda ser aprobado por diferentes profesores o usuarios, se debe de reestructurar la tabla activos de la base de datos para poder realizar esta funcionalidad.
- En caso de que una persona ingrese a dos dispositivos y realice una misma solicitud exactamente al mismo tiempo, se generará un código de error pues la llave de solicitudes se basa en que no existan dos solicitudes iguales al mismo tiempo.

Para la página web:

- La vista de facilidades no permite hacer una modificación acorde a la forma en la que se almacenan. Lo que se realiza, es eliminar las facilidades existentes para que el usuario ingrese una descripción de las nuevas características de los laboratorios. Este aspecto puede implementarse de mejor forma.

Para la aplicación móvil:

- La sincronización con la base de datos principal no valida el caso en que una persona ya haya realizado una solicitud de laboratorio mientras el usuario de la aplicación móvil este offline. Por lo tanto, si la persona que se encuentra sin conexión realiza una reserva a esa misma hora, a la hora de sincronizar ambas solicitudes se almacenarán, aunque colisiones las horas solicitadas para el laboratorio.

Para la API:

- No se han encontrado problemas

# Evidencias de trabajo:

Al realizar previamente un proyecto similar como tarea del curso, los roles del equipo se mantuvieron y la división de trabajos fue sencilla de realizar, pues, la mayoría trabajo en lo que había desarrollado previamente a excepción de la incorporación de la base de datos.

La comunicación de los miembros de trabajo se realizó a través de la herramienta de WhatsApp. En un grupo destinado a las tarea del proyecto. Aquí se informó acerca de los avances de cada estudiante, el estado de cada uno de los componentes y la coordinación para trabajar en unísono. Ejemplo de ello podemos observar a continuación:

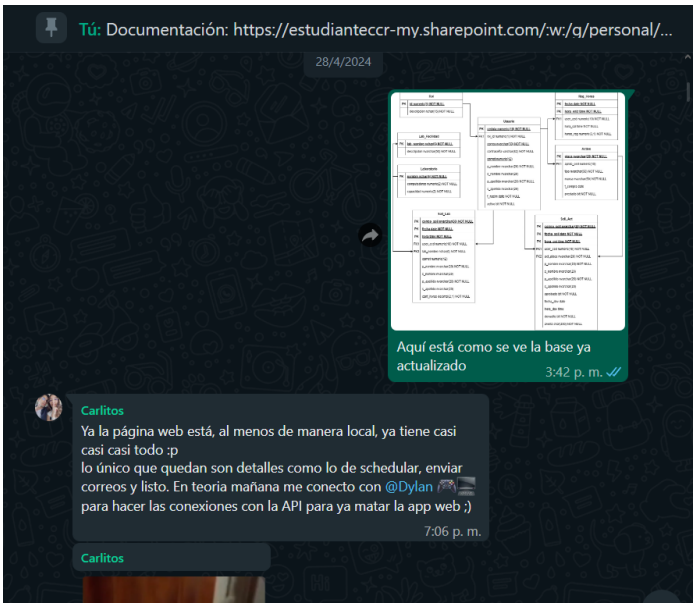


**Figura 3.** Evidencia del canal de comunicación principal del equipo de trabajo (WhatsApp)

Además, se utiliza la herramienta de versiones de código GitHub, de esta forma podíamos trabajar de forma cómoda en cada una de las tareas, consultar el avance de las demás personas y gestionar la unión de los componentes evitando cualquier conflicto de compatibilidad que pudiera surgir en caso de no utilizar controladores de versiones.

A continuación, podemos observar la división de ramas del repositorio, con la finalidad de organizar las secciones del proyecto y modularizar lo mayor posible los componentes garantizando así una alta escalabilidad y un menor acoplamiento:

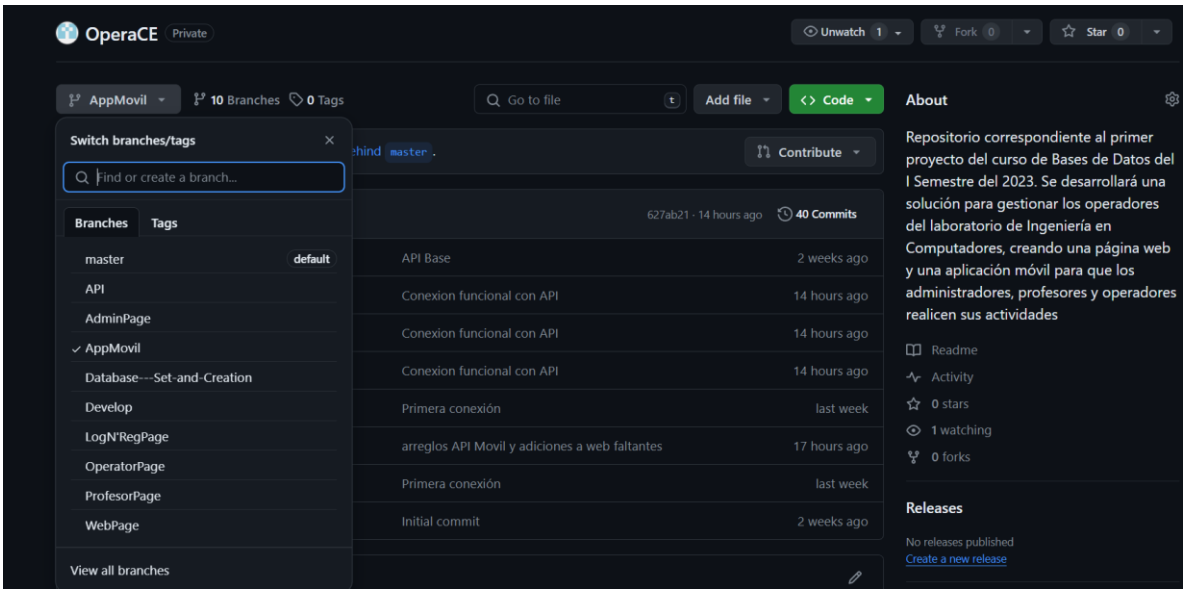

**Figura 4.** Repositorio de GitHub del proyecto y sus ramas correspondientes a las funcionalidades desarrolladas.

Por último, es importante añadir que la mayoría de los integrantes del grupo pasaban su tiempo en las instalaciones del TEC, por lo tanto, mucha de la comunicación y coordinación se realizó de forma presencial. Ya sea, en las clases de Bases de Datos o fuera de horario en los laboratorios de Ingeniería en Computadores.

# Conclusiones

Se logró desplegar satisfactoriamente «usando IIS» un API/WebService/REST Service como podemos observar en la sección llamada de esta forma. Así logramos facilitar la transmisión de información y comunicación entre la aplicación web, la aplicación móvil y la base de datos.

Se desarrolló de forma exitosa una página web para administradores, profesores y operadores utilizando la herramienta de React y sus componentes característicos, evidencia de ello pudimos observar en las descripciones de todas las partes que conforman el sitio web en la sección “Página Web”

Se implementó una aplicación móvil enfocada a los profesores usando React Native con todas las prestaciones requeridas apoyándose en una base de datos local propulsada en SQL lite para almacenar todas las solicitudes sin conexión de usuario para su próxima vinculación a la base de datos principal.

Por último, se logró cumplir a cabalidad la visión original del producto pues gracias a todos los productos mencionados anteriormente se desarrolló un prototipo de aplicación que permite usuarios de los laboratorios de ingeniería de computadores realizar sus actividades de manera efectiva, rápida, confiable y centralizada TEC.

# Recomendaciones

- Antes de comenzar a codificar, es recomendable tener un plan claro. Esto debería incluir una comprensión de las necesidades del proyecto, así como una idea de las estructuras de datos y las funciones que se utilizarán.
- Es crucial entender cómo se estructuran los objetos JSON y cómo se pueden manipular. Esto permitirá un manejo más eficiente de estos objetos y evitará errores comunes.
- Antes de utilizar una REST API, es recomendable definir primero los modelos que se van a utilizar. Esto puede ayudar a evitar la creación de modelos innecesarios y a mantener el código más limpio y eficiente.
- Antes de diseñar la base de datos, es recomendable realizar diagramas que funcionen como guía y modelo del proyecto. En nuestro caso utilizamos diagramas conceptuales y relacionales pero no tiene porque limitarse solo a estos dos.

# Bibliografía


*Tutorial: Intro to React – React*. (s. f.). React. https://legacy.reactjs.org/tutorial/tutorial.html

*Learn the Basics · React Native*. (2023, 8 diciembre). https://reactnative.dev/docs/tutorial

*C# Tutorial (C Sharp)*. (s. f.). https://www.w3schools.com/cs/index.php

Jc Miron (Director). (2022, julio 4). *SETUP C#, .Net Core, and VS Code in 2 MINS! 2022*. https://www.youtube.com/watch?v=SQim2adwVJI

Rodrigo Méndez (Director). (2023, septiembre 20). *CURSO Básico de REACT NATIVE*. https://www.youtube.com/watch?v=XlraT4cAS9E

Programming with Mosh. (2023, March 12). *React tutorial for beginners* [Video]. YouTube. https://www.youtube.com/watch?v=SqcY0GlETPk

Fireship. (2020, September 8). *React in 100 seconds* [Video]. YouTube. https://www.youtube.com/watch?v=Tn6-PIqc4UM

*Datatable in React JS*. (n.d.). YouTube. https://www.youtube.com/playlist?list=PLZboEx3gZXge7zhg45b5TzmjhNPudSaZn

Code With Yousaf. (2023, February 6). *React Axios Crud App with JSON server | ReactJS Axios Rest API | React Crud App with JSON server* [Video]. YouTube. https://www.youtube.com/watch?v=Rm4__WgPncI

Elmasri y Navathe. Fundamentals of Database Systems, 6ta Edición. Addison Wesley. 2010.

Raheem, Nasir. Big Data: A Tutorial-Based Approach, 2019

Albert Y Zomaya editor.; Sherif Sakr. Handbook of Big Data Technologies. SpringerLink. 2017

M. Tamer Özsu Patrick Valduriez. Principles of distributed database systems. 2011

Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. 2015