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

Documentación Técnica - ResTEC

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 respuesta a la necesidad de mejorar la gestión del comedor institucional del TEC, se ha concebido el proyecto RestTEC. Este proyecto tiene como objetivo principal desarrollar un sistema integral que optimice la experiencia de los usuarios al realizar pedidos, así como la eficiencia en la gestión de estos. En esta documentación técnica, se detallará el proceso de desarrollo de un prototipo de aplicación que permitirá gestionar la descripción del caso asociado a cada pedido.

El propósito fundamental del proyecto RestTEC es desarrollar un prototipo de aplicación capaz de gestionar de manera efectiva la descripción del caso vinculado a los pedidos realizados en el comedor institucional. Para alcanzar este objetivo general, se han establecido una serie de metas específicas.

En primer lugar, se busca crear un API/WebService/REST Service para facilitar la comunicación entre distintos componentes del sistema. Esta capa de comunicación será fundamental para garantizar la interoperabilidad entre la aplicación web, la aplicación móvil y la base de datos subyacente.

Además, se tiene como objetivo desarrollar una página web utilizando como herramienta React. Esta página web servirá como la interfaz principal para la gestión de pedidos, proporcionando una experiencia intuitiva y funcional tanto para el personal administrativo como para los clientes.

Asimismo, se pretende implementar una aplicación móvil que permita a los clientes interactuar con el sistema desde sus dispositivos móviles. Esta aplicación móvil ofrecerá funcionalidades específicas, como la capacidad de realizar pedidos desde cualquier lugar y en cualquier momento, eliminando así la necesidad de hacer filas en el comedor.

Finalmente, se planea instalar localmente una aplicación web que incluya tanto el frontend como el backend. Esta instalación local garantizará un rendimiento óptimo y la seguridad de los datos de los usuarios, al tiempo que se cumplen con las restricciones de no utilizar servicios en la nube como AWS o Azure.

En resumen, el proyecto RestTEC tiene como objetivo modernizar y optimizar el proceso de pedidos en el comedor institucional, brindando una experiencia más ágil y personalizada tanto para los usuarios como para el personal encargado de la gestión de pedidos.

Objetivos del Proyecto

Objetivo General

Desarrollar un prototipo de aplicación que permita gestionar de manera eficiente la descripción de casos asociados al proceso de pedidos en el comedor institucional del TEC.

Objetivos Específicos

- Crear un API/WebService/REST Service para facilitar la comunicación entre la aplicación web, la aplicación móvil y la base de datos.
- Desarrollar una página web utilizando tecnologías como Angular/React, Bootstrap, HTML5 y CSS3, para proporcionar una interfaz amigable y altamente funcional para la administración y gestión de pedidos.
- Implementar una aplicación móvil que permita a los clientes interactuar con el sistema desde sus dispositivos inteligentes, facilitando así el proceso de pedido y eliminando la necesidad de hacer filas en el comedor.

Descripción de los Componentes

Página Web

Lo primero que debemos recalcar es que la herramienta empleada para crear la página web fue React que es una biblioteca de JavaScript que se utiliza para construir interfaces de usuario interactivas y dinámicas para aplicaciones web. React funciona mediante el principio de “divide y vencerás” pues se desarrollan una gran cantidad de componentes sencillos que luego se unen para crear un resultado más elaborado y complejo.

Para nuestra implementación podemos identificar una gran cantidad de componentes .jsx, no obstante, podemos realizar un separación en función de su objetivo que nos resultará muy útil para explicar cómo funciona, como se emplearon y cuál es su importancia dentro del prototipo de solución elaborado.

A todo esto, hay que agregar que también existen una gran cantidad de archivos de estilos .css donde se encuentra todas las configuraciones gráficas de los módulos para hacer la visualización del sitio más atractivo, sin embargo, no vamos a entrar en profundidad en lo que contiene estos archivos y como se desarrollaron

Página para Administradores

La primera agrupación de componentes la podemos atribuir a la construcción de una de las dos variantes principales de la visualización de la página, la vista de administrador. Para construir esta interfaz utilizamos, de los más importantes, los siguientes componentes

Continuamos con ModalCreateMenu y ModalCreatePlato que son básicamente ventanas emergentes que aparecen en la aplicación web, ambos sirven para crear un nuevo elemento, menú o plato. Cuando se abre, muestra un formulario una cierta información. Cuando completas el formulario y presionas "Crear", se guarda la información del nuevo menú o plato. Si decides cancelar, puedes cerrar el modal haciendo clic en "Cancelar".

Además, encontramos también ModalEditMenu y ModalEditPlato que funcionan específicamente para editar menú o platos ya existentes. Al abrirlo, muestra un formulario con campos para la información respectiva. Puedes cambiar los valores en los campos y luego

guardar los cambios haciendo clic en "Guardar Cambios". Si decides cancelar, puedes cerrar el modal haciendo clic en "Cancelar".

Por último, el módulo más importante es AdminPage, esta contiene toda la visualización de información sobre pedidos activos, gestión de platos y menús, reportes de platos y los mejores 10 clientes. Utiliza una serie de estados para administrar el resto de los componentes de los que hablamos anteriormente para que de esta forma el usuario pueda utilizar sus funcionalidades correctamente. Además, posee funciones para obtener la información respectiva y mostrarla de la mejor manera posible.

Página para Chefs

Para la segunda visualización de la página, que corresponde al lado del chef. Resulto más simple pues este rol realiza menos acciones en comparador a los administradores. Por lo tanto, solo necesitamos emplear un componente para desarrollar el sitio correspondiente. El nombre del componente se llama ChefPage y representa la página de un chef

Este componente muestra diferentes pestañas para visualizar y gestionar los pedidos actuales del chef, los pedidos asignados a otros chefs y los pedidos pendientes que aún no han sido tomados por ningún chef. Utiliza varias bibliotecas, entre ellas react-bootstrap para el diseño de la interfaz de usuario y react-data-table-component para mostrar los datos tabulares de manera eficiente.

Además, implementa funciones para eliminar, reasignar y preparar pedidos, así como también emplea funciones auxiliares para calcular el tiempo restante en la preparación de los pedidos en función de los datos que ya han sido agregado previamente por los administradores. Es decir, obtiene constantemente las actualizaciones de la información para mostrar la información correctamente.

Página de Inicio de Sesión

La última sección importante del sitio web es la página que recibe a los usuarios, que radica en el inicio de sesión. El componente utilizado para realizar esta parte es llamado LoginForm,. Permite a los usuarios ingresar su correo electrónico y contraseña para iniciar sesión. Utiliza el hook useState para manejar el estado de los campos de correo electrónico,

contraseña y cualquier mensaje de error. Además, utiliza el hook useNavigate de React Router para redirigir a diferentes páginas después del inicio de sesión exitoso.

El formulario incluye campos para ingresar el correo electrónico y la contraseña, así como íconos correspondientes a cada campo para una mejor experiencia de usuario. También muestra un mensaje de error si las credenciales ingresadas son incorrectas.

Al enviar el formulario, el componente busca en los datos de usuarios almacenados localmente para encontrar una coincidencia de correo electrónico y contraseña. Si se encuentra un usuario, se redirige a la página correspondiente según el rol del usuario (ya sea chef o administrador). Si no se encuentra ningún usuario con las credenciales proporcionadas, se muestra un mensaje de error.

Funciones Auxiliares

A manera de ayudar a los otros módulos a realizar ciertas acciones que son ajenas a la visión principal del componente decidimos realizar una sección de métodos auxiliares para agregarla a los módulos principales. La función que detectamos necesaria tiene que ver con el formato del tiempo de la página.

El módulo proporciona una función llamada calcularTiempoRestante, que se utiliza para determinar y representar el tiempo restante en un formato legible. Esta función toma como entrada un tiempo inicial y una cantidad de minutos adicionales a sumar. En primer lugar, se crea un objeto de fecha para representar el tiempo actual. Luego, el tiempo de entrada dado se convierte en otro objeto de fecha.

A continuación, se suman los minutos adicionales al tiempo de entrada. La función calcula la diferencia en milisegundos entre el tiempo actual y el tiempo de entrada modificado. Posteriormente, esta diferencia se convierte en días, horas, minutos y segundos. Finalmente, se construye una cadena de texto que representa el tiempo restante en un formato legible, incluyendo días, horas, minutos y segundos según corresponda.

Este módulo es útil para aplicaciones que necesitan mostrar información sobre plazos, tiempos límite o intervalos de tiempo de una manera fácilmente comprensible para los usuarios.

Almacenamiento de Información

Lo último que compone nuestra página web es una carpeta donde tenemos la información correspondiente al menú, platos, usuarios, etc. Como en la especificación no podemos implementar una base de datos web, entonces trabajamos con archivos .json que es un formato de intercambio de datos ligero y legible para JavaScript.

En esta carpeta tenemos archivos para almacenar los datos ingresados por los administradores relacionado con los menús y los platos. Tenemos un archivo JSON que almacena todo lo relacionado a los pedidos, si están en espera o realizándose, quien los está realizando y otro datos relacionados. Por último, para poder entrar en la página, se almacenan las credenciales de inicio de sesión de los usuarios.

Aplicación Móvil

La aplicación móvil del proyecto es una plataforma que permite a los usuarios encargar comida de manera conveniente y eficiente. Ofrece características clave que incluyen inicio de sesión, registro de usuarios, home, selección y comprar, visualización del pedido, factura, visualización de pedidos activos, actualización de la información de la cuenta y retroalimentación de los pedidos.

El proyecto está desarrollado en React Native, una biblioteca de JavaScript para construir interfaces de usuario móviles. La estructura del proyecto se organiza principalmente en:

- Screens: Contiene los componentes de pantalla individuales que componen la interfaz de usuario de la aplicación, implementados en JavaScript.
- App.tsx: Punto de entrada de la aplicación que configura la navegación y proporciona la estructura principal del flujo de la aplicación.

Funcionalidades Principales:

- Inicio de sesión (LoginScreen): La pantalla de inicio de sesión permite a los usuarios autenticarse en la aplicación proporcionando su correo electrónico y contraseña. Esta funcionalidad utiliza una solicitud POST para enviar los datos de inicio de sesión al servidor y proporciona retroalimentación al usuario según la respuesta del servidor.

- Registrarse (RegisterScreen): La pantalla de registro de usuario permite a los nuevos usuarios crear una cuenta en la aplicación proporcionando su información personal, incluido nombre, correo electrónico, contraseña, etc. Utiliza una solicitud POST para enviar los datos de registro al servidor y muestra mensajes de retroalimentación según la respuesta del servidor.
- Actualizar información de la cuenta (UpdateScreen): La pantalla de actualización de información del usuario permite al usuario modificar sus datos personales registrados en la aplicación, como nombre, apellidos, correo electrónico, contraseña, número de cédula, fecha de nacimiento, números de teléfono y dirección. Además, proporciona la opción de eliminar la cuenta de usuario.
- Página principal (HomeScreen): La pantalla de actualización de información del usuario permite al usuario modificar sus datos personales, como nombre, contraseña, dirección, etc. Utiliza una solicitud PUT para enviar los cambios al servidor y una solicitud DELETE para eliminar la cuenta del usuario, si es necesario.
- Comprar (ShoppingScreen): La pantalla de compra permite a los usuarios explorar y seleccionar platos disponibles para ordenar en el comedor institucional. Al cargar la pantalla, se realiza una solicitud GET al servidor para obtener la información del menú disponible. Esta solicitud obtiene los datos de los platos, incluyendo su nombre, descripción, precio, tipo, calorías y cantidad en el carrito de compras. Los botones presentes en la pantalla ofrecen funcionalidades clave para interactuar con el menú y el carrito de compras. Al hacer clic en el botón "Añadir al carrito", el usuario puede agregar un plato al carrito de compras, mientras que al hacer clic en "Remover del carrito", puede ajustar la cantidad de platos en el carrito, ya sea decrementando su cantidad o eliminándolos por completo.
- Visualizar pedido (OrderScreen): La pantalla de visualizar pedido muestra un resumen de los artículos seleccionados por el usuario para comprar. Permite al usuario confirmar su pedido, lo que desencadena una solicitud POST al servidor para realizar la orden. Además, calcula y muestra el precio total del pedido.
- Facturación (ReceiptScreen): La pantalla de factura muestra los detalles de un pedido realizado, incluidos los platos seleccionados, el total y la información de fecha y hora.

Utiliza una solicitud GET para obtener los detalles del pedido del servidor y formatea la información para su visualización al usuario.

- Ver pedidos activos (ActiveOrdersScreen): La pantalla de visualización de órdenes activas muestra todas las órdenes que están actualmente en proceso. Utiliza una solicitud GET para obtener la lista de órdenes activas del servidor y calcula el progreso de cada orden para mostrarlo visualmente al usuario. Además, si una orden alcanza el 100% de progreso, se ofrece al usuario la opción de proporcionar retroalimentación sobre la orden.
- Retroalimentación (FeedbackScreen): La pantalla de retroalimentación permite al usuario calificar los platos recibidos en un pedido completado. Utiliza una solicitud POST para enviar las calificaciones al servidor y proporciona un componente de calificación de estrellas para facilitar la entrada del usuario.

API/WebService/REST Service

En el contexto de este proyecto, la API REST se utiliza para conectar la aplicación con una base de datos que consiste en varios archivos JSON. Estos archivos JSON actúan como almacenes de datos, cada uno representando una entidad o conjunto de datos específicos.

La API REST proporciona una serie de endpoints, que son URLs específicas que la aplicación puede solicitar para realizar operaciones en los datos almacenados en los archivos JSON. Estas operaciones pueden incluir:

- Adiciones: La API permite agregar nuevos datos a la base de datos. Por ejemplo, si la aplicación necesita agregar un nuevo usuario, puede enviar una solicitud POST al endpoint correspondiente con los detalles del usuario en formato JSON.
- Cambios: Se pueden realizar cambios en los datos existentes en la base de datos. Esto se logra mediante solicitudes PUT a los endpoints relevantes, proporcionando los datos actualizados que se desean modificar.
- Llamados: La API permite consultar datos específicos de la base de datos. Esto se hace a través de solicitudes GET a los endpoints adecuados. Por ejemplo, si la aplicación necesita recuperar la información de un usuario en particular, enviaría una

solicitud GET al endpoint que maneja los usuarios, posiblemente proporcionando un identificador único como parámetro.

- Eliminación: La API también permite eliminar datos de la base de datos. Esto se realiza enviando solicitudes DELETE a los endpoints correspondientes, especificando qué datos deben eliminarse.

La API está desarrollada en C#, utilizando .NET 8.0. La estructura del proyecto se organiza principalmente en un único programa llamado Program.cs, el cual contiene el mapeo de las diferentes llamadas, así como el manejo respectivo de cada archivo JSON utilizado.

- Funcionalidades Principales:
  app.MapPost("/login", async context => {...}): Este bloque de código maneja las solicitudes POST a la ruta "/login". Se utiliza para autenticar a los usuarios. Lee el cuerpo de la solicitud, que se espera que sea un objeto JSON con un nombre de usuario y una contraseña. Luego, verifica estos datos con los usuarios almacenados en un archivo JSON. Si el usuario existe y la contraseña es correcta, devuelve un código de estado 200 y algunos datos del usuario. Si el usuario no existe o la contraseña es incorrecta, devuelve un código de estado 401. Si el formato del archivo JSON es incorrecto, devuelve un código de estado 500.
- app.MapPost("/register", async context => {...}): Este bloque de código maneja las solicitudes POST a la ruta "/register". Se utiliza para registrar nuevos usuarios. Lee el cuerpo de la solicitud, que se espera que sea un objeto JSON con los datos de registro del usuario. Luego, verifica si el correo electrónico ya existe en el archivo JSON de usuarios. Si el correo electrónico ya está registrado, devuelve un código de estado 400. Si no, crea un nuevo usuario, lo agrega al archivo JSON y devuelve un código de estado 201.
- app.MapGet("/dishes", async context => {...}): Este bloque de código maneja las solicitudes GET a la ruta "/dishes". Se utiliza para obtener una lista de platos del menú. Lee los archivos JSON de platos y menú, y devuelve un objeto JSON con la lista de platos y menú. Si hay un error al leer los archivos JSON, devuelve un código de estado 500.

- app.MapPost("/order", async context => {...}): Este método maneja las solicitudes POST a la ruta "/order". Se utiliza para crear una nueva orden. Lee el cuerpo de la solicitud, deserializa el JSON a un objeto Order, genera un ID de orden aleatorio, lee y actualiza la información de los platos y usuarios en archivos JSON, y finalmente escribe la nueva orden en el archivo de pedidos.
- app.MapGet("/order/{orderId}", async (HttpContext context, int orderId) => {...}): Este método maneja las solicitudes GET a la ruta "/order/{orderId}". Se utiliza para obtener una orden específica por su ID. Lee el archivo de pedidos, busca la orden con el ID especificado y la devuelve. Si la orden no se encuentra, devuelve un error 404.
- app.MapGet("/orders/{clientId}", async (HttpContext context, int clientId) => {...}): Este método maneja las solicitudes GET a la ruta "/orders/{clientId}". Se utiliza para obtener todas las órdenes de un cliente específico por su ID. Lee el archivo de pedidos, busca todas las órdenes con el ID de cliente especificado y las devuelve. Si no se encuentran órdenes, devuelve un error 404.
- app.MapGet("/menu", async context => {...}): Este método maneja las solicitudes GET a la ruta "/menu". Se utiliza para obtener el menú de platos disponibles. Lee el archivo de menú y devuelve la lista de platos. Si ocurre un error al leer el archivo, devuelve un error 500.
- app.MapPut("/update/{userId}", ...): Esta ruta maneja las solicitudes PUT a "/update/{userId}". Se utiliza para actualizar la información de un usuario en particular. El ID del usuario se extrae de la ruta de la solicitud. Luego, se lee el cuerpo de la solicitud, que se espera que sea un objeto JSON que representa los datos actualizados del usuario. Estos datos se utilizan para actualizar la entrada correspondiente en un archivo JSON que actúa como una base de datos de usuarios. Si la actualización es exitosa, se devuelve un código de estado HTTP 200 y un mensaje de éxito. Si ocurre un error, se devuelve un código de estado HTTP apropiado y un mensaje de error.
- app.MapDelete("/delete/{userId}", ...): Esta ruta maneja las solicitudes DELETE a "/delete/{userId}". Se utiliza para eliminar un usuario en particular. Al igual que en la ruta anterior, el ID del usuario se extrae de la ruta de la solicitud. Luego, se busca al usuario en el archivo JSON y, si se encuentra, se elimina. Si la eliminación es

exitosa, se devuelve un código de estado HTTP 200 y un mensaje de éxito. Si ocurre un error, se devuelve un código de estado HTTP apropiado y un mensaje de error.

- app.MapPost("/feedback", ...): Esta ruta maneja las solicitudes POST a "/feedback". Se utiliza para recibir comentarios de los usuarios sobre los platos. El cuerpo de la solicitud se espera que sea un objeto JSON que representa el feedback. Este feedback se utiliza para actualizar la calificación de un plato en particular en un archivo JSON que actúa como una base de datos de platos. Si la actualización es exitosa, se devuelve un código de estado HTTP 200 y un mensaje de éxito. Si ocurre un error, se devuelve un código de estado HTTP 500 y un mensaje de error.
- app.MapGet("/pedidos", ...): Esta ruta maneja las solicitudes GET a "/pedidos". Se utiliza para obtener todos los pedidos. Los pedidos se leen de un archivo JSON y se devuelven en la respuesta. Si el archivo JSON es válido y contiene una clave "pedidos", se devuelve un código de estado HTTP 200 y los pedidos. Si el archivo JSON no es válido o no contiene una clave "pedidos", se devuelve un código de estado HTTP 500 y un mensaje de error.
- app.MapGet("/pedidos/admin", ...): Esta ruta maneja las solicitudes GET a "/pedidos/admin". Funciona de manera similar a la ruta anterior, pero la respuesta se envuelve en un objeto con una clave "data". Esto puede ser útil para proporcionar metadatos adicionales en la respuesta.
- app.MapGet("/usuarios", ...): Esta ruta maneja las solicitudes GET a "/usuarios". Se utiliza para obtener todos los usuarios. Los usuarios se leen de un archivo JSON y se devuelven en la respuesta. Si el archivo JSON es válido y contiene una clave "usuarios", se devuelve un código de estado HTTP 200 y los usuarios. Si el archivo JSON no es válido o no contiene una clave "usuarios", se devuelve un código de estado HTTP 500 y un mensaje de error.
- app.MapGet("/platos", ...): Esta ruta maneja las solicitudes GET a "/platos". Se utiliza para obtener todos los platos. Los platos se leen de un archivo JSON y se devuelven en la respuesta. Si el archivo JSON es válido y contiene una clave "platos", se devuelve un código de estado HTTP 200 y los platos. Si el archivo JSON no es válido o no contiene una clave "platos", se devuelve un código de estado HTTP 500 y un mensaje de error.

- app.MapPost("/actualizar/jsons", ...): Este controlador maneja las solicitudes POST a la ruta "/actualizar/jsons". Lee el cuerpo de la solicitud, que se espera que sea un JSON que representa una actualización a los menús y platos. Deserializa el JSON, realiza algunas transformaciones en los datos (como asignar nuevos ID a los elementos que no tienen uno), y luego escribe los datos actualizados de vuelta a dos archivos JSON: "db\menu.json" y "db\platos.json". Si todo va bien, devuelve una respuesta 200 OK. Si algo sale mal, captura la excepción y devuelve una respuesta 500 Internal Server Error.
- app.MapPut("/update/{id_orden}/{id_chef}", ...): Este controlador maneja las solicitudes PUT a la ruta "/update/{id_orden}/{id_chef}". Lee el archivo "db\pedidos.json", busca una orden con el id_orden especificado en la ruta, y si la encuentra, actualiza el id_chef de esa orden. Luego escribe las órdenes actualizadas de vuelta al archivo "db\pedidos.json". Si todo va bien, devuelve una respuesta 200 OK. Si la orden no se encuentra, devuelve una respuesta 404 Not Found. Si algo sale mal, captura la excepción y devuelve una respuesta 500 Internal Server Error.
- app.MapDelete("/eliminar/{id_orden}", ...): Este controlador maneja las solicitudes DELETE a la ruta "/eliminar/{id_orden}". Lee el archivo "db\pedidos.json", busca una orden con el id_orden especificado en la ruta, y si la encuentra, la elimina de la lista de órdenes. Luego escribe las órdenes actualizadas de vuelta al archivo "db\pedidos.json". Si todo va bien, devuelve una respuesta 200 OK. Si la orden no se encuentra, devuelve una respuesta 404 Not Found. Si algo sale mal, captura la excepción y devuelve una respuesta 500 Internal Server Error.

Problemas encontrados:

Para la página web:

- El principal problema encontrado, es que, al editar platos y menús, no se logra editar el nombre, las demás características si se logran editar. Se piensa que es por la sintaxis utilizada. No se sabe el motivo del error.

Para la aplicación móvil:

- A veces, el tiempo que dura el pedido en terminar, puede que muestre tiempos intrínsecos al hacer los cálculos, o si son pedidos largos, puede que no se muestre el día en que se terminaría el pedido, ya que no se considera que el pedido dure más de un día.

Para la API:

- No se han encontrado problemas sin resolver.

Problemas conocidos:

Para la página web:

- No se puede crear un menu, si no hay un plato con su mismo nombre. Esto se debe a que cuando se realizan los cálculos de ganancias totales de un plato, se necesitan características de tanto el plato como el menu. Esto se solucionó colocando una ventana emergente, diciéndole al usuario que tenga cuidado.

Para la aplicación móvil:

- En la ventana de Feedback, al presionar el botón de enviar, se envía la información, pero no se devuelve automáticamente a la ventana de pedidos en progreso.
- En la ventana de Feedback, las estrellas que deberían de demostrar para darle una calificación de 1 a 5, no cargan y en su lugar aparecen cuadros con líneas sin textura.

Para la API: No se encuentran problemas conocidos.

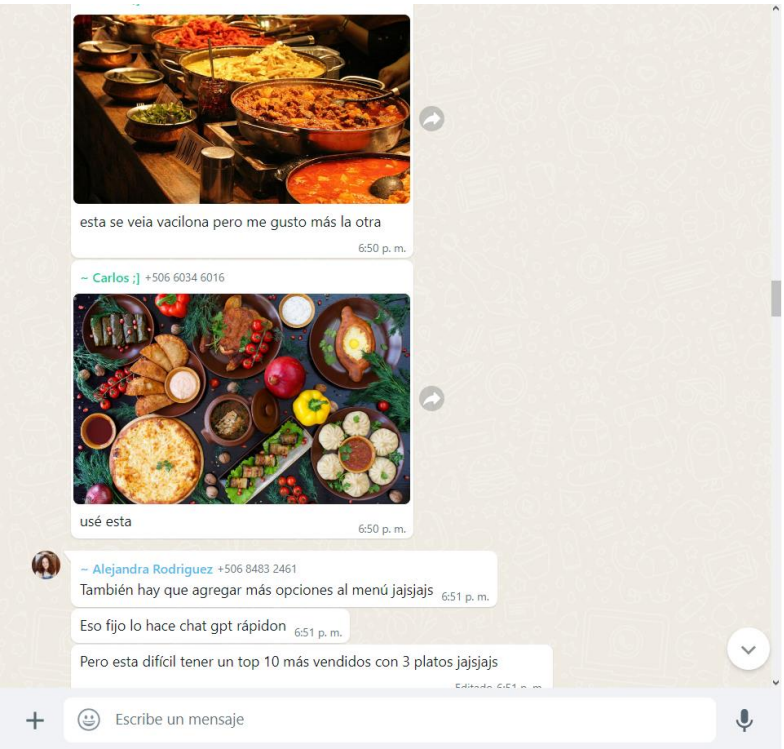
Plan de actividades

Actividad	Descripción	Estimación de Tiempo	Responsables	Fecha de Entrega
Diseño Inicial de la Solución	Para esta sección se hablará que tareas deben de realizarse y quien es el responsable de hacerlas. Además, se realiza un proceso de estudio de las herramientas a utilizar	7 días	Todos	28/2/24
Desarrollo del Login de la Pagina Web	Se desarrolla usando react la vista que recibe a los usuarios, comprueba sus credenciales y lo acepta o rechaza comparándolo con la base	3 días	José María, Carlos y Ricardo	2/3/24
Desarrollo del Login de la Aplicación Móvil	Se desarrolla usando react-native el login en la móvil, comprueba sus credenciales y lo acepta o rechaza comparándolo con la base	3 días	Dylan y Alejandra	2/3/24
Desarrollo de la vista de Administrador de la página web	Se desarrolla todas las tablas relacionadas que cumplan con los requerimientos para el administrador	7 días	José María, Carlos y Ricardo	9/3/2024
Desarrollo de la lógica para solicitar los pedidos en la aplicación móvil	Se desarrolla todas funcionalidades para que los clientes puedan solicitar	7 días	Dylan y Alejandra	9/3/2024

	pedidos y dar feedback			
Desarrollo de la vista de Chef en la página web	Se desarrolla todas las tablas relacionadas que cumplan con los requerimientos para el chef	7 días	José María, Carlos y Ricardo	16/3/2024
Desarrollo de la API/WebService/REST Service	Usando C# se desarrollará una API capaz de vincular la información generada tanto por la página web como la aplicación móvil.	4 días	Dylan y Carlos	19/3/2024
Comunicación entre los distintos componentes del proyecto	Se verifica que las acciones asociadas a los componentes modifiquen correctamente la base de datos y esta información se actualice correctamente	2 días	Dylan y Carlos	21/3/2024
Documentación	Este proceso se realizará a la par de todas las etapas anteriores documentando las principales características que se estén desarrollando para los principales modelos del proyecto	Indefinido	Todos	21/3/2024
Entrega de la tarea	El equipo trabajará para pulir los últimos detalles y de esta forma realizar la entrega del proyecto de forma satisfactoria.	1 día	Ricardo	22/3/2024

Evidencias de trabajo:

Para el control de trabajo se estableció un grupo de WhatsApp donde se estuvo discutiendo el planteamiento, avances y finalización del proyecto.



Además, muchas de las decisiones tomadas fueron en reuniones presenciales.

También se adjunta captura de algunos commits realizados en la página de GitHub.

Conclusiones

Se logró crear satisfactoriamente 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 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 usando React Native con todas las prestaciones requeridas que permita a los clientes interactuar con el sistema desde sus dispositivos como se puede ver en las descripciones de funcionalidad de esta.

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 a los dueños del restaurante gestionar de manera eficiente el proceso de pedidos en el comedor institucional del 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.

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