INFORME DEL PLAN DE PROYECTO DEL SEMESTRE

Líder de grupo:

Jhonatan David Hernandez Martinez

Demás integrantes:

Darien Andres Castañeda Agudelo

Levir Heladio Hernandez Suarez

Juan Diego Atehortua Duque

Materia: Programación en la WEB

Grupo: B1

Escuela de ingeniería de sistemas e informática

Universidad Industrial de Santander

14 de febrero de 2023

TITULO DE LA APLICACIÓN WEB A DESARROLLAR

Webmarket

OBJETIVO GENERAL

Implementar un servicio de página web para supermercados.

OBJETIVOS ESPECIFICOS

- Implementar enrutamiento y paginas privadas en el Front-end
- Acoplar los componentes ya existentes con el Back-end
- Implementar servidor manejador de peticiones con Node.js
- Implementar envío de correos con nodemailer en el Back-end
- Implementar cliente de base de datos usando Sequelize

CREDENCIALES DE ACCESO AL PROYECTO

RUTAS DEL PROYECTO

Las páginas nombradas a continuación son detalladas en la sección de:

IMPLEMENTACIONES Y CAMBIOS REALIZADOS EN EL FRONT-END

Página principal | localhost:3000/principal

Página de registro | localhost:3000/registro

Página de usuario | localhost:3000/usuario

Página de envió | localhost:3000/envio

Página de inventario | localhost:3000/inventario

Demás paginas llevan a error 404 | localhost:3000/*

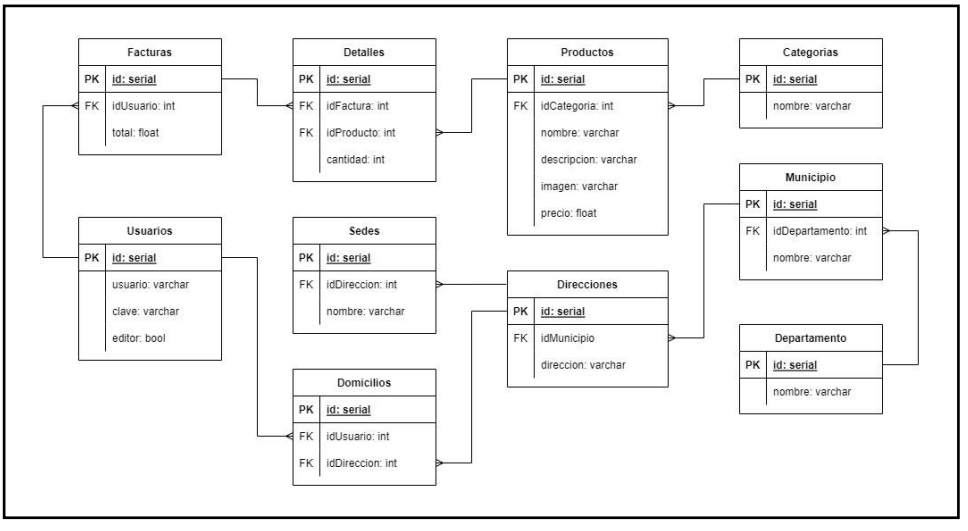
USUARIOS DISPONIBLES POR DEFECTO

Administrador2023@gmail.com | Administrador2023

ClienteAntiguo2023@gmail.com | ClienteAntiguo2023

ClienteNuevo2023@gmail.com | ClienteNuevo2023

MODELO RELACIONAL IMPLEMENTADO



JUSTIFICACION DE TABLAS

Los datos de las tablas Categorías y Productos se planean utilizar en la vista principal de la página y también en la pagina de manejo de inventario.

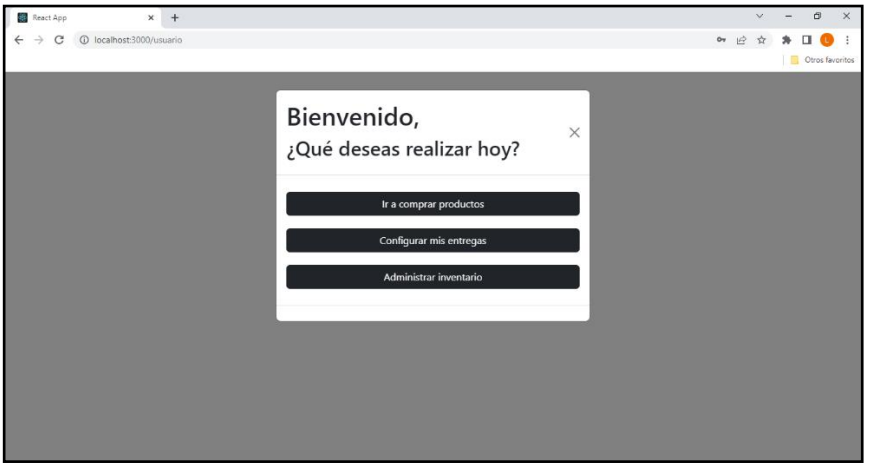
Los datos de la tabla Usuarios en la página serán utilizados en las páginas de login y registro de usuarios.

Los datos de la tabla Facturas y Detalles se planean usar para recomendar carritos de compras por defecto y ahorrarle tiempo al usuario.

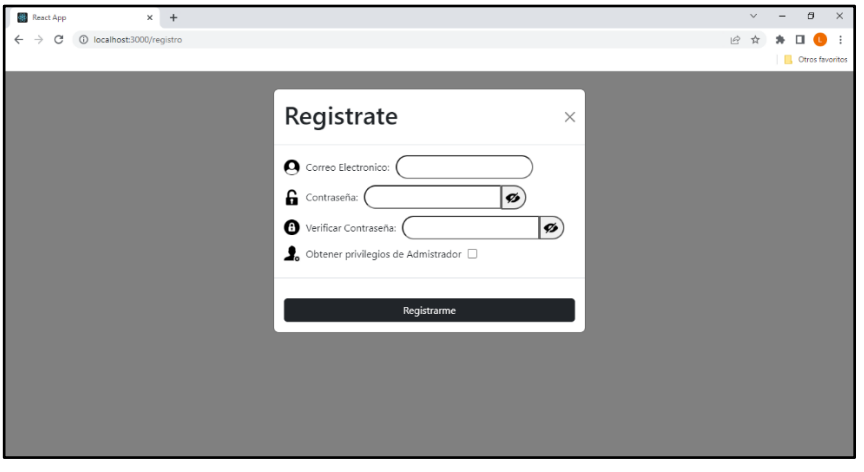
Los datos de las tablas Domicilios, Sedes y demás relacionados con direcciones serán utilizados en la página de configuración de envíos.

IMÁGENES DE LAS NUEVAS IMPLEMENTACIONES

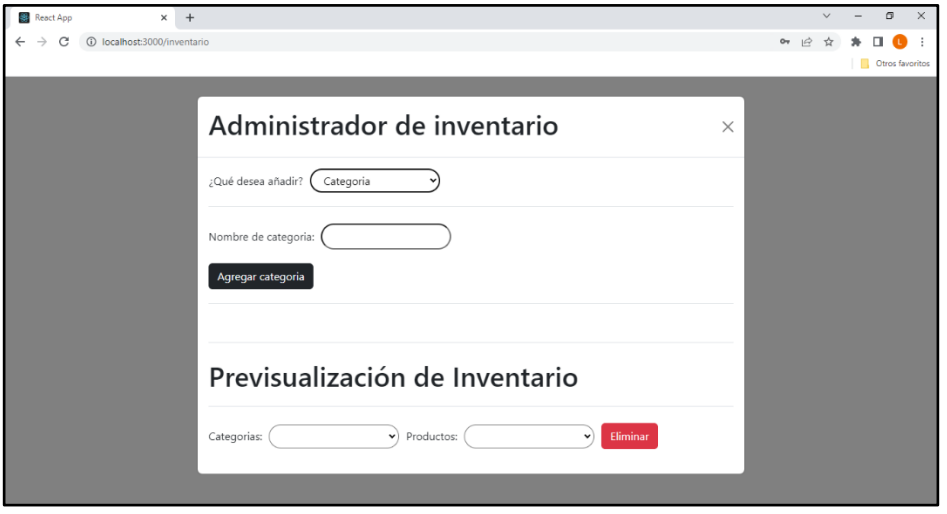
SECCION DE USUARIO



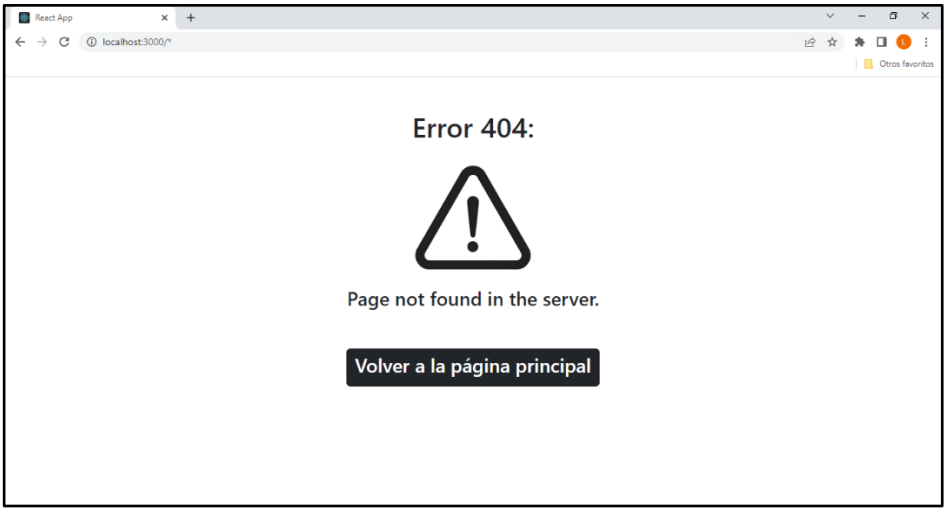
REGISTRO DE USUARIOS



MANEJO DE INVENTARIO



PAGINA DE RUTA INVALIDA



ESTRUCTURA JERARQUICA DEL PROYECTO

El proyecto consta de dos carpetas principales “client” en donde se encuentra el front-end realizado con React, mientras que la carpeta “server” contiene el back-end realizado con Node.js y la librería express.

La carpeta “server” consta de dos carpetas; “database” y “mailer”, además de dos archivos .js ; “cliente.js” y “server.js” los cuales se encargan de inicializar el servidor en el puerto 5000 y realizar peticiones a base de datos, mientras que las carpetas mencionadas anteriormente se utilizan para guardar archivos sql muy extensos como los de creación y llenado, además del gestor de correos de nodemailer.

La carpeta “client” contiene un proyecto con estructura similar al de la Fase I con una carpeta “src” consta de 4 carpetas y el archivo “index.js” necesario para el renderizado de la página.

1. La carpeta “aplicacion” contiene el componente principal “App.jsx” encargado de contener los demás componentes secundarios y manejar las rutas, por cuestiones de tiempo se colocaron todos los componentes de paginas en esta carpeta como; “Registro.jsx”, “Inventario.jsx”, entre otros.
2. La carpeta “componentes” contiene todos los componentes secundarios utilizados en la página, esta carpeta se divide en una carpeta de “index” y otra de “login”.
3. La carpeta “css” contiene todos los archivos css de la aplicación los cuales son importados en “App.jsx” para evitar importaciones en cada componente, a su vez esta carpeta también tiene dos subcarpetas “index” y “login”.
4. La carpeta “descartado” contiene componentes y diseños que finalmente no se llegaron a incorporar en el proyecto por diversas razones.

La carpeta “index” se detalla mas a fondo en la Fase I pero en general esta contiene una carpeta “login” contiene el componente “Login.jsx” el cual usa expresiones regulares para validar los correos ingresados y una contraseña con las siguientes especificaciones: Mínimo una mayúscula, una minúscula, un digito y con al menos una longitud de 8 caracteres.

NUEVAS FUNCIONALIDADES

IMPLEMENTACIONES REALIZADAS EN EL BACK-END

En la carpeta “database” se encuentran los archivos “.sql” con las instrucciones necesarias para crear la base de datos desde cero y llenarla.

El archivo “cliente.js” en términos generales asume el cargo de gestor de conexión a base de datos, en este archivo se crea una clase “clienteDB” que cuenta con diversos métodos con la capacidad de ejecutar consultas, actualizaciones, eliminaciones, etc.

En este archivo la librería “Sequelize” se utiliza para obtener un cliente de conexión a base de datos con PostgreSQL, el cual se envuelve dentro de “clienteDB” para separar la sección de manejo de base de datos de la de consultas al servidor.

También se utiliza la librería “Fs” de File System, la cual permite leer archivos con formato “.sql”, los cuales están presentes en la carpeta “database” y son extremadamente extensos.

Todos los métodos de “clienteDB” se declararon como “async” para luego utilizar “await” dentro de estos y así volver síncronas consultas de “Sequelize” que son naturalmente asíncronas.

El archivo “server.js” se encarga de prender el servidor en el “puerto 5000” y manejar tanto peticiones GET como POST y DELETE.

Este archivo utiliza tanto el gestor proveniente de “cliente.js” como el despachador de correos de “emisor.js”, evitando saturar este archivo principal del servidor con tanto código al realizar únicamente llamados a métodos y funciones de estos dos gestores, dejando “server.js” como un archivo encargado únicamente de recibir los parámetros de las peticiones y delegarlos a los métodos de los gestores.

En este archivo utilizamos la librería de “express”, además del modulo “body-parser” para interpretar las solicitudes POST con información en formato JSON provenientes del Front-end.

El archivo “mailer.js” se encarga de proveer un gestor de transferencia “smtp” para enviar correos en caso de olvido de contraseña, además de contener también una función para generar claves aleatorias para su posterior recuperación.

Su funcionamiento se puede observar iniciando sesión en “mailtrap.io” y manteniendo activa la bandeja de salida, en este caso “mailtrap” muestra cómo se vería el correo en un escenario real, sin embargo, estos correos no se envían si no se tiene una cuenta de pago de “mailtrap”.

En este archivo utilizamos la librería de “nodemailer” la cual es compatible con el despachador de correos “mailtrap”.

IMPLEMENTACIONES Y CAMBIOS REALIZADOS EN EL FRONT-END

Se tuvieron que realizar cambios significativos en varias secciones del Front-end debido a que la capa que simulaba la “persistencia” en la Fase I no permitía el uso de la API “fetch” la cual utilizamos para realizar consultas al servidor desde el Front-end activo en el puerto 3000. También se configuro un Proxy en el “package.json” para evitar la repetición de “localhost:5000” cada vez que se llamaba a “fetch”.

Para adecuar el front-end se necesitaron usar más variables de estado, además del uso de Hooks como “useEffect” el cual se encarga de realizar las consultas antes de renderizar la página. Sin embargo, las peticiones “fetch” al ser asíncronas en algunos casos presentan retrasos a pesar de manejarlas como “Promises” mediante clausulas “then”.

El archivo “App.jsx” en términos generales se encarga de gestionar las rutas de toda la aplicación utilizando componentes como; “BrowserRouter”, “Route”, entre otros provenientes de la librería “react-router-dom”. Además de proveer los contextos relacionados con el inicio de sesión y privilegios de administrador utilizando el Hook “useContext”.

Cada ruta del Router se relacionó con un componente especifico detallado a continuación:

“localhost:3000/principal” está ligado con el componente “Principal” el cual se encarga de la vista base de la página con los productos, categorías, carrito de compras y login mediante un “Modal”. Este componente mantiene gran parte del diseño de la Fase I pero fue modificado para realizar las consultas de productos en tiempo real a la base de datos. Además, esta es la ruta publica por defecto.

Además, el Modal de login contiene un link en caso de olvidar la contraseña, el cual manda un correo con una contraseña aleatoria temporal usando nodemailer.

“localhost:3000/registro” esta ligado al componente “Registro”, este componente gestiona los datos para tomar datos de registro y luego enviar una solicitud POST en la base de datos para actualizar la tabla de usuarios, además de poseer otras funcionalidades de verificación de entradas. Se coloco una opción para darle privilegios de administrador al usuario, pero en un proyecto real esto debería quitarse y realizarse manualmente desde base de datos. Esta es la segunda ruta publica de la aplicación.

“localhost:3000/usuario” esta ligado al componente “Usuario”, este componente le da opciones al usuario sobre que acciones puede realizar dependiendo de sus privilegios como; Continuar comprando, Gestionar e envió a domicilio o Administrar inventario. La opción de Administrar inventario depende de si el usuario tiene los privilegios requeridos, de lo contrario mantiene el inventario como ruta privada.

“localhost:3000/envio” esta ligado al componente “Envio”, este componente le permite al usuario seleccionar el domicilio al cual desea que envíen su pedido o también ir a recoger en una sede disponible, este componente realiza consulta inicialmente todos los departamentos con un “useEffect” y luego de seleccionar un departamento renderiza los municipios correspondientes. Estos datos fueron sacados de datos abiertos al público en Colombia disponibles en; Departamentos y municipios de Colombia, este formulario también presenta opciones para añadir más domicilios. Esta ruta es privada ya que depende del usuario seleccionado, por lo tanto, se debe logear antes de acceder a esta página.

“localhost:3000/inventario” esta ligado al componente “Inventario”, este componente permite añadir, eliminar y previsualizar los productos presentes en el inventario, se pueden añadir tanto categorías, como también productos. Y también eliminarlos en caso de desearlo. Los productos presentan un campo para colocar una dirección URL del producto en lugar de una opción para subir archivos debido a que al subir una imagen a la base de datos esta se transforma a un formato BLOB para objetos binarios grandes que pesa mucho más que una URL, por lo tanto decidimos dejarlo simple con la opción de URL.

Las demás direcciones redirigen a un componente “Error” el cual presenta un mensaje de error 404 y contiene un botón para volver a la página principal el cual usa “NavLink” de “react-router-dom”.

FUNCIONALIDADES NO IMPLEMENTADAS

Una de las funcionalidades esenciales relacionada con el pago decidimos descartarla, a pesar de haber encontrado un componente de pasarela de pago en la documentación de PayPal, preferimos descartarlo debido a que requería crear una cuenta especial de PayPal y configurar el componente “PaymentBlock.jsx” para realizar el pago a esta cuenta, más información sobre este componente se puede encontrar en la documentación para desarrolladores de PayPal.

Otra funcionalidad relacionada que no se implemento fue el guardado de métodos de pago para el usuario, por ello se descartaron las tablas de métodos de pago y no se presentaron en el modelo relacional, además existe un boceto “Pago.html” de cómo debía verse la página de configuración de métodos de pago.