

TRABAJO PRÁCTICO N°3

Alumno: Gianoglio, Malena.

Docentes: Pioli, Pablo
Ferreyra, Juan Pablo

ÍNDICE.

1. ENUNCIADO DEL PROBLEMA.....	2
2. PROPUESTA DE SOLUCIÓN.	3
2.1 OBJETIVO:.....	3
2.2 ALCANCE:	4
2.3 PROPUESTA TECNOLÓGICA:.....	5
3. ARQUITECTURA.	5
5. REQUERIMIENTOS.....	7
5.1 REQUERIMIENTOS FUNCIONALES.	7
6. DIAGRAMA DE CASOS DE USO.	8
7. DIAGRAMA ENTIDAD-RELACIÓN.....	9
8. INTERFACES DE USUARIO Y FLUJO DE DATOS.....	9
8.1. INTERFACES WEB	9
8.2. INTERFACES APP MOBILE	20
8.3. INTERFACES ADMINISTRADOR	36

1. ENUNCIADO DEL PROBLEMA.

Una empresa dedicada a la fabricación de materiales para la construcción se encuentra distribuida en diferentes 3 plantas productivas, una oficina comercial y vendedores que atienden a clientes mayoristas en diferentes zonas.


La sucursal A extrae materia prima que se utiliza como insumo en la planta C. La planta B elabora productos semi-terminados en base a alambres de acero que se utilizan para producir en la planta C.

Por su parte, la planta C utiliza elabora ladrillos, vigas de cemento y bloques pre armados de diferentes medidas. Desde la planta C se realiza el envío de los pedidos directamente al cliente.

Cada planta productiva realiza ingresos de stock de materias primas, consulta de stock, generación de órdenes de producción de los diferentes productos y envío de productos a las diferentes plantas.

Por decisión de la gerencia se necesita reducir los tiempos de atención a clientes minoristas, para ello se pretende ofrecer la posibilidad de cotizar y generar pedidos directamente en el sitio web de la empresa, para ello, una vez identificados los clientes podrán consultar los productos, ejemplo:

Podrán cotizar, ingresando cantidad de metros cuadrados a construir y tipos de materiales, en base a dicha información se debería poder determinar la cantidad de materiales necesarios, por ejemplo:

	Ladrillo Hueco 12x18x33cm 9 tubos Precio por unidad: \$390,00 Descripción: Ladrillo hueco cerámico 12x18x33 cm 9 tubos Ladrillo de cerramiento Uso: Especiales para tabiques divisorios y cerramientos (ambientes interiores y muros de cierre). Cantidad por pallet: 144 unidades
	Viga 4 mts Precio por unidad: \$ 10619 Descripción: Ladrillo hueco cerámico 12x18x33 cm 9 tubos Ladrillo de cerramiento Uso: Especiales para tabiques divisorios y cerramientos (ambientes interiores y muros de cierre). Uso: Son utilizadas para techar en la construcción. Se colocan sobre las paredes y van acompañadas entre viga y viga por ladrillos para techo y malla sima.

Para construir un galpón de 40m x 40m, de 6m de altura, con ladrillo de tipo bloques de 18cm x 33cm se necesitaría cubrir una superficie de 960 metros cuadrados, con lo cual la cantidad de ladrillos, considerando una separación de 40 cm entre vigas, se necesitaría:

- 16161 ladrillos, equivalentes a 112,23 pallets.
- Importe \$ 6.302.790.

Se debería poder gestionar los descuentos por cantidad, por ejemplo, a partir de los 10mil ladrillos ofrecer un 5% de descuento sobre el valor del producto.

A partir de dicha cotización el cliente podrá realizar un pedido, debiendo completar información de domicilio de envío. La empresa cuenta con servicio de envío.

Una vez aprobado el pedido, se acuerda una forma de pago. Una vez que el cliente realiza el pago se envía el pedido.

2. PROPUESTA DE SOLUCIÓN.

2.1 OBJETIVO:

La empresa busca mejorar la atención al cliente y aumentar la eficiencia en los procesos de cotización y venta de materiales de construcción. Para lograrlo, se implementará un sitio web que le permitirá a los clientes acceder de manera rápida y sencilla a la información de los productos, realizar cotizaciones y generar pedidos

desde la comodidad de sus hogares. Esto no solo mejorará la experiencia del usuario, sino que también optimizará el tiempo de respuesta y reducirá la carga de trabajo del personal.

Debido al éxito del sitio web, la gerencia ha solicitado el desarrollo de una aplicación móvil que amplíe las funcionalidades actuales del sitio web, añadiendo nuevas características para mejorar la experiencia del cliente y optimizar el proceso de venta y pago. La app permitirá realizar pagos, acumular puntos por compras y usar una API externa para facilitar el cálculo de metros cuadrados.

2.2 ALCANCE:

La primera iteración, está enfocada a los procesos relacionados a la gestión de ventas, que abarcará las funcionalidades principales:

- Catálogo de productos: visualización de productos disponibles, incluyendo imágenes, descripciones y precios.
- Cotizador: herramienta que permitirá a los usuarios ingresar dimensiones para calcular la cantidad de material necesario y el costo total, considerando descuentos por volumen.
- Realización de pedidos: proceso para que los usuarios puedan realizar pedidos en basados en las cotizaciones generadas.
- Proceso de compra: integración de opciones de pago y envío del pedido.
- Notificaciones por correo electrónico: confirmación automática de pedidos y actualizaciones de estado.

En la segunda iteración, el sistema se expandirá para incluir las siguientes funcionalidades adicionales:

- Plataformas de pago: Los usuarios podrán pagar los pedidos mediante diferentes plataformas de pago como Modo, MercadoPago, entre otras.
- Acumulación de puntos: Los clientes podrán sumar puntos por cada compra realizada, basados en la regla definida: $n \text{ (puntos)} = x \text{ (pesos)}$. Esta regla será modificable por un usuario administrador.
- API de Autocad: Implementación de una API externa que permitirá a los usuarios obtener automáticamente los metros cuadrados de pared y techo a

partir de un plano de Autocad. Esta información se utilizará para agilizar el proceso de cotización.

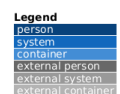
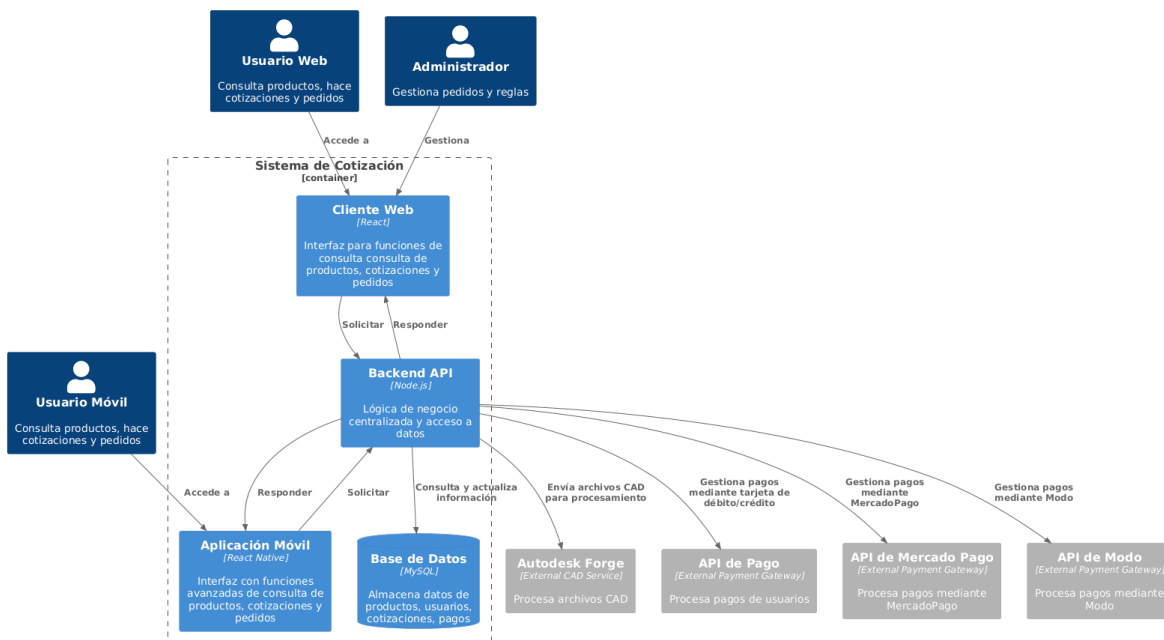
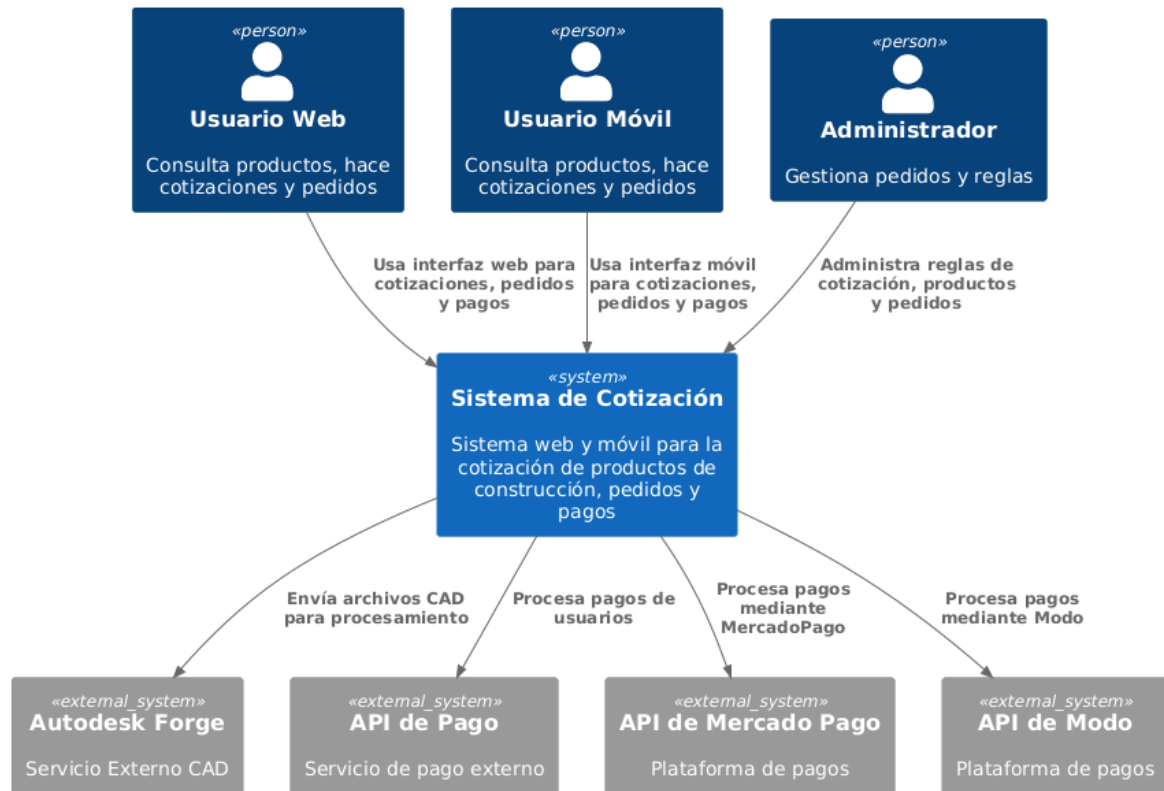
2.3 PROPUESTA TECNOLÓGICA:

Para la implementación de esta plataforma se sugieren las siguientes tecnologías:

- Frontend: React para la versión web y React Native para la aplicación móvil.
- Backend: Node.js
- Base de datos: MySQL
- Notificaciones por correo electrónico: Nodemailer para enviar notificaciones automáticas a los clientes.
- Plataformas de pago: Integración con Modo, MercadoPago, y otras pasarelas de pago.
- API de Autocad: Integración de API externa para el cálculo de metros cuadrados.

3. ARQUITECTURA.

El sistema mantendrá una arquitectura cliente-servidor, en la cual el cliente, ya sea a través de la aplicación móvil o el sitio web, se comunicará con el servidor que proporcionará los servicios necesarios. La información se intercambiará en formato JSON entre ambos componentes. Además, se mantendrá la separación de responsabilidades entre el frontend (React/React Native) y el backend (Node.js), lo que facilitará el mantenimiento y escalabilidad del sistema.



5. REQUERIMIENTOS.

5.1 REQUERIMIENTOS FUNCIONALES.

Requerimientos funcionales para la primera iteración:

RF1 – El sistema debe permitir a los usuarios iniciar sesión, y proporcionar la opción de registro para quienes no posean una cuenta.

RF2 – El sistema debe permitir a los usuarios, luego de que inicien sesión, consultar el catálogo de productos disponibles con su respectiva información.

RF3 – El sistema debe permitir a los usuarios realizar una cotización de un determinado producto, ingresando las medidas de su proyecto.

RF4 – El sistema debe calcular la cantidad de material necesario y el costo total en función de las medidas ingresadas por el usuario.

RF5 – El sistema debe aplicar descuentos automáticos cuando la cantidad de material supere el mínimo predefinido.

RF6 – El sistema debe generar la cotización, incluyendo cantidad de material necesario, costo total, descuento aplicado y costo total con descuento.

RF7 – El sistema debe permitir que los usuarios realicen un pedido a partir de una cotización.

RF9 – El sistema debe solicitar a los usuarios sus datos de envío para la distribución del pedido.

RF10 – El sistema debe ofrecer a los usuarios diversas formas de pago y permitir que elija la de su preferencia.

RF11 – El sistema debe confirmar el pedido y generar un comprobante una vez que el pago se procesó exitosamente.

RF12 – El sistema debe notificar a los usuarios sobre su pedido vía correo electrónico una vez abonado el mismo.

Requerimientos funcionales para la segunda iteración:

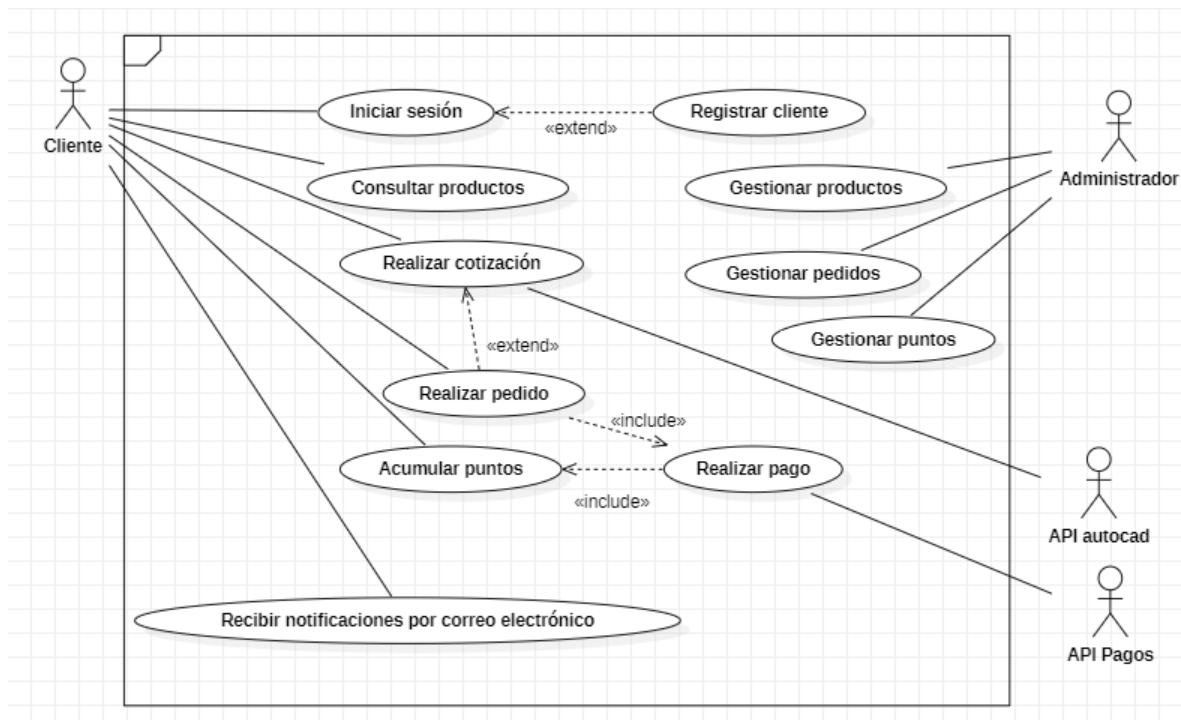
RF13 – El sistema debe permitir que los usuarios realicen el pago de sus pedidos mediante plataformas como Modo y MercadoPago.

RF14 – El sistema debe permitir que los usuarios acumulen puntos por cada compra, según la regla de conversión definida.

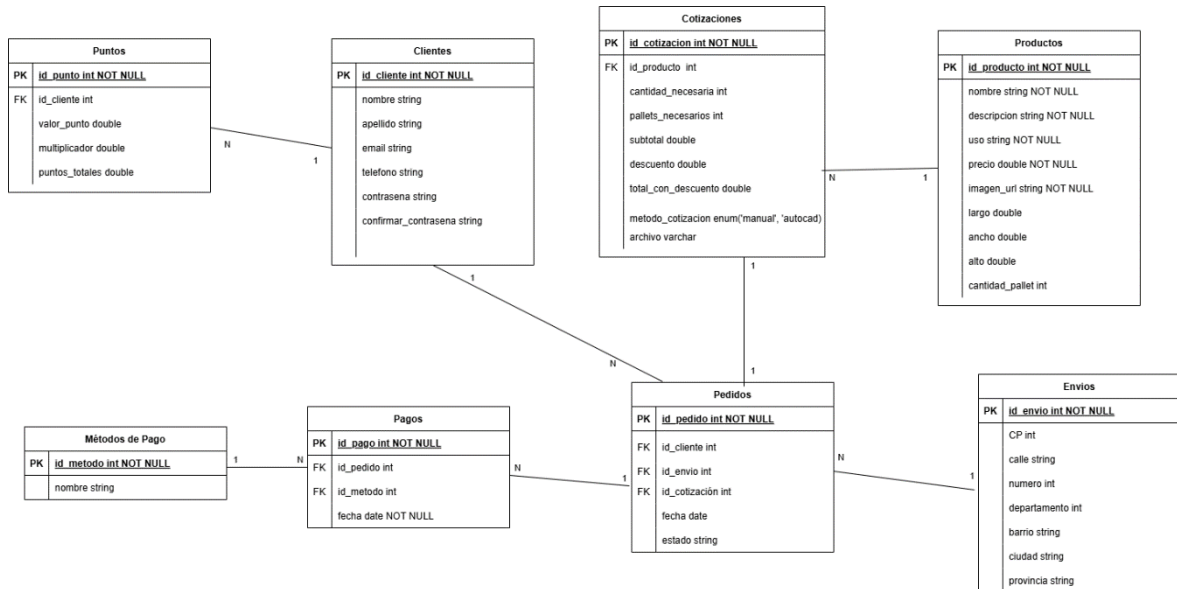
RF15 – El sistema debe permitir a los administradores modificar la cantidad de puntos otorgados por los montos gastados.

RF16 – El sistema debe integrar una API externa que, a partir de un plano Autocad, calcule los metros cuadrados de pared y techo, y utilice esta información para la cotización.

6. DIAGRAMA DE CASOS DE USO.



7. DIAGRAMA ENTIDAD-RELACIÓN.



8. INTERFACES DE USUARIO Y FLUJO DE DATOS.

8.1. INTERFACES WEB

Inicio de sesión:

Si el usuario ya se encuentra registrado, inicia sesión completando los campos correspondientes con la dirección de correo electrónico y contraseña asociada a su cuenta.

Flujo de datos:

- Del frontend al backend:

```
{
  "email": "string",
  "contraseña": "string"
}
```

- Proceso en el backend:

El sistema realiza la autenticación comparando los datos enviados con los almacenados en la base de datos. El backend genera un token de autenticación y lo envía de vuelta al frontend.

Si el resultado de la autenticación es exitoso, se redirigirá al usuario a la sección de productos del sitio web. Por el contrario, el backend responderá con un mensaje dependiendo del error que se haya producido.

- Del backend al frontend:

Fracaso:

```
{  
  "token": string,  
  "mensaje": "Correo electrónico no registrado."  
}  
  
{  
  "token": string,  
  "mensaje": "Contraseña incorrecta."  
}
```

El frontend guarda este token JWT en un lugar seguro (por ejemplo, en localStorage).



Diagrama de la interfaz de inicio de sesión. Se muestra un formulario centrado con los siguientes elementos:

- Un título "CORREO ELECTRÓNICO" en mayúsculas.
- Un campo de entrada de texto para el correo electrónico.
- Un título "CONTRASEÑA" en mayúsculas.
- Un campo de entrada de texto para la contraseña.
- Un botón azul con el texto "Iniciar sesión".
- Un enlace de texto "¿No tienes una cuenta aún? [Crear cuenta](#)".

Crear cuenta:

Si la persona no se encuentra registrada, deberá completar los campos correspondientes para el registro.

Flujo de datos:

- Del frontend al backend.

```
{  
  "nombre": string,  
  "apellido": string,  
  "email": string,  
  "telefono": string,  
  "contrasena": string,  
  "confirmar_contrasena": string  
}
```

- Proceso en el backend:

El backend recibe los datos y realiza las validaciones correspondientes (formato de datos válidos, correo electrónico no registrado anteriormente, contraseña coincidente).

Si los datos son correctos, se almacenan en la base de datos en la tabla de usuarios y devuelve una respuesta indicando el registro exitoso. Allí se encontrará un botón para “Ir a productos”.

Si hubo un error, se devolverá un mensaje indicando el mismo. Por ejemplo, “Correo electrónico ya registrado”. La persona podrá intentar nuevamente el registro.

- Del backend al frontend:

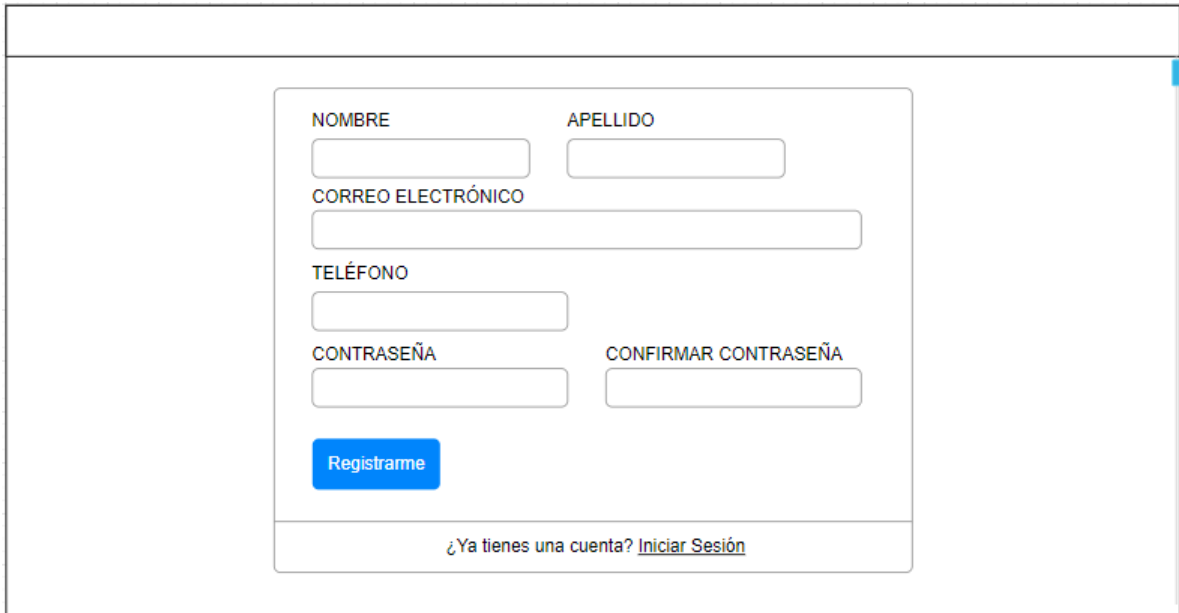
Éxito:

```
{  
  "mensaje": "Registro exitoso."  
}
```

Fracaso:

```
{  
  "mensaje": "Correo electrónico ya registrado."  
}
```

}

A registration form UI mockup. It features a central white box with rounded corners on a light gray background. Inside the box, there are input fields for 'NOMBRE' and 'APELLIDO' (two columns), 'CORREO ELECTRÓNICO' (one column), 'TELÉFONO' (one column), 'CONTRASEÑA' and 'CONFIRMAR CONTRASEÑA' (two columns). Below these fields is a blue 'Regístrate' button. At the bottom of the box, there is a link: '¿Ya tienes una cuenta? [Iniciar Sesión](#)'.

NOMBRE APELLIDO

CORREO ELECTRÓNICO

TELÉFONO

CONTRASEÑA CONFIRMAR CONTRASEÑA

Regístrate

¿Ya tienes una cuenta? [Iniciar Sesión](#)

Productos:

El frontend realiza una solicitud al GET al backend para obtener todos los productos.

Flujo de datos:

- Proceso en el backend:

Consulta la base de datos para recuperar los productos disponibles.

Si la solicitud se procesó correctamente, envía una respuesta con la lista de productos.

- Del backend al frontend:

```
{  
  "productos": [  
    {  
      "id_producto": integer,  
      "nombre": string,  
      "descripcion": string,  
      "uso": string,  
    }  
  ]  
}
```

```
"precio": double,  
"largo": double,  
"alto": double,  
"ancho": double,  
"imagen": url,  
"cantidad_palet": integer  
}  
]  
}
```



Cotizar:

Cuando el usuario presiona el botón “cotizar” asociado a un determinado producto, se abre un modal en el cual se deben ingresar las medidas correspondientes del proyecto a realizar la cotización y presionar “Calcular”.

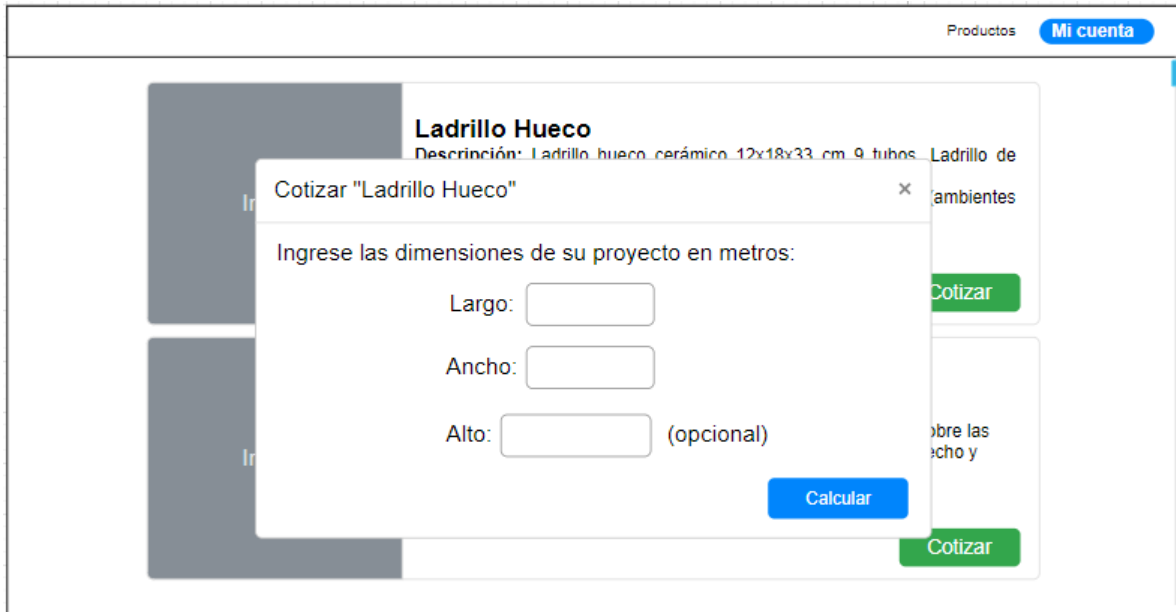
Flujo de datos:

- Del frontend al backend:

```
{  
  "id_producto": integer,  
  "largo": double,  
  "ancho": double,
```

"alto": double

}



- Proceso en el backend:

El backend recibe los datos, identifica el producto a cotizar en la base de datos y realiza los cálculos necesarios para determinar:

- Unidades requeridas del producto.
- Pallets requeridos.
- Precio total de los productos.
- Descuento del 5% si la cantidad supera las 10.000 unidades.
- Precio total aplicando el descuento.

Y crea un registro en la tabla de cotizaciones. Los datos de la cotización se guardan en el contexto (cotizacionData).

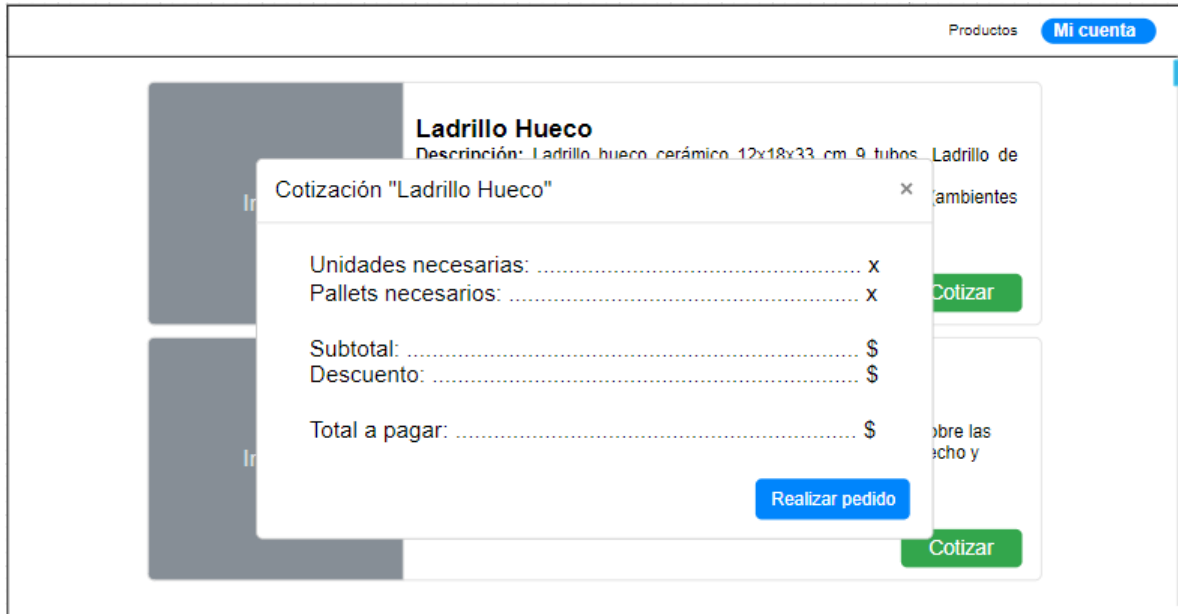
- Del backend al frontend:

{

```
"id_cotizacion": integer,  
"nombre_producto": string,  
"cantidad_necesaria": integer,  
"pallets_necesarios": integer,  
"precio_total": double,
```

```
"subtotal": double,  
"descuento": double,  
"total_con_descuento": double  
}
```

El modal se actualiza mostrando ahora dicha información.



Pedido:

Luego de presionar “realizar pedido” el cliente es redireccionado a otra pantalla, en la cual se mostrarán los datos guardados en el contexto (cotizacionData). A su vez el frontend hará una solicitud POST a el backend para crear un nuevo registro en pedidos y a esta nueva pantalla se le enviará el id del pedido.

```
{  
  "id_cotizacion": integer,  
  "id_usuario": integer,  
  "estado": “en proceso”  
}
```

En la pantalla de pedidos:

Se recuperan los datos guardados en el contexto (cotizacionData).

Flujo de datos:

En la misma pantalla el usuario deberá completar con sus datos de envío.

- Del frontend al backend:

```
{  
  "id_pedido": integer,  
  "CP": integer,  
  "calle": string,  
  "numero": integer,  
  "departamento": string,  
  "barrio": string,  
  "ciudad": string,  
  "provincia": string  
}
```

- Proceso en el backend:

El backend recibe los datos del envío, crea un nuevo registro en la tabla envíos y asocia el id_envío al pedido en la tabla de pedidos.

Productos **Mi cuenta**

Datos de Entrega

C.P.	<input type="text"/>	Ladrillo Hueco x (unidades)
Calle:	<input type="text"/>	Nro: <input type="text"/>
Departamento:	<input type="text"/> (opcional)	
Barrio:	<input type="text"/> (opcional)	
Ciudad:	<input type="text"/>	
Provincia:	<input type="text"/>	

Subtotal:	\$
Descuento:	\$
Total a pagar:	\$

[Continuar con el pago](#)

Pago:

Cuando el usuario decide “Continuar con el pago”, es redirigido a una pantalla donde se encuentran los medios de pago disponibles y la información del pedido.

Flujo de datos:

- Del backend al frontend:

```
{
  "id_pedido": integer,
  "id_cotizacion ": integer,
  "cotizacion": [
    {
      "nombre_producto": string,
      "cantidad_necesaria": integer,
      "pallets_necesarios"; integer,
      "precio_total": double,
      "subtotal": double,
      "descuento": double,
      "total_con_descuento": double
    }
  ]
}
```

Productos **Mi cuenta**

Pago

Medio de pago:

Tarjeta de débito o crédito

Efectivo

Pedido Nro: xxxx

Ladrillo Hueco x (unidades)

Subtotal:	\$
Descuento:	\$
Total a pagar:	\$

Confirmar pedido

Si el usuario decide pagar con tarjeta de crédito o débito, deberá completar los campos indicados.

Productos **Mi cuenta**

Pago Pedido Nro: xxxx

Medio de pago:

◀ Tarjeta de débito o crédito

Número de tarjeta

Titular de tarjeta Vencimiento (MM/AA) CVV

DNI

Ladrillo Hueco x (unidades)

Subtotal: \$

Descuento: \$

Total a pagar: \$

Confirmar pedido

- Del frontend al backend:

```
{  
  "nro_tarjeta": integer,  
  "nombre_titular": string,  
  "vencimiento_tarjeta": date,  
  "codigo_seguridad": integer,  
  "dni ": string,  
}
```

- Proceso en el backend:

Aquí se procesa el pago. Si el pago es exitoso, se actualizará el estado del pedido en la base de datos a “pagado”.

Si el usuario decide pagar en efectivo, el backend procesa la solicitud y cambia el estado del pedido a “Pendiente de pago”. Se le envía al usuario un email con los datos del pedido y la información del local.

Confirmación compra:

Una vez que el usuario ha procesado el pago, es redirigido a una pantalla de confirmación donde se muestra un resumen de la información del pedido y un mensaje de agradecimiento. Esta pantalla proporciona al usuario detalles sobre su compra y confirma que el proceso se completó con éxito.

Flujo de datos:

- Del backend al frontend:

```
{  
  "id_pedido": integer,  
  "nombre_producto": string,  
  "cantidad_necesaria": integer,  
  "pallets_necesarios": integer,  
  "subtotal": double,  
  "descuento": double,  
  "total_con_descuento": double  
}
```

Productos **Mi cuenta**

Pedido Nro: xxxx

¡Gracias por tu compra!

Hemos recibido tu pago y tu pedido ha sido confirmado. Te mantendremos informado sobre el estado de tu envío.

Ladrillo Hueco x (unidades)

Subtotal: \$

Descuento: \$

Total a pagar: \$

Ver otros productos

8.2. INTERFACES APP MOBILE

Iniciar sesión:

En esta pantalla el usuario puede ingresar sus datos para ingresar a su cuenta en el caso de que ya tenga una existente, por el contrario, podrá registrarse.

Flujo de datos:

- Del frontend al backend:

El frontend hace una solicitud POST a la api en el endpoint: /api/login

```
{  
  "email": string,  
  "contrasena": string  
}
```

- Proceso en el backend:

El backend verifica el email con la base de datos. Si existe y la contraseña coincide, el backend genera un token de autenticación y lo envía de vuelta al frontend.

- Del backend al frontend:

```
{  
  "token": string  
}
```

El frontend guarda este token JWT en un lugar seguro (por ejemplo, en localStorage).

En el caso de la persona no se encuentre registrada, podrá tocar en “Crear Cuenta” y será redirigida a la pantalla para el registro.

Mockup of a mobile app login screen. The screen has a blue header with a hamburger menu icon. The main title is "INICIO SESIÓN". Below the title are two input fields: "Correo Electrónico:" and "Contraseña:". A blue button labeled "INICIAR SESIÓN" is positioned below the password field. At the bottom, there is a link that says "¿No tienes una cuenta aún? Crear cuenta".

Registro:

Aquí el usuario podrá registrarse ingresando sus datos personales.

Flujo de datos:

- Del frontend al backend:

Al rellenar los campos correspondientes y presionar “Registrarse” el frontend hace una solicitud POST a la api en el endpoint: /api/register

```
{  
  "nombre": string,  
  "apellido": string,  
  "email": string,  
  "teléfono": string,  
  "contrasena": string  
  "confirmar_contrasena": string  
}
```

El formulario de registro, titulado "REGISTRARME", se encuentra dentro de una interfaz con una barra superior azul que contiene un menú hamburguesa. El formulario mismo tiene un fondo gris claro y una sombra. Incluye los siguientes campos de entrada:

- Nombre:
- Apellido:
- Correo Electrónico:
- Teléfono:
- Contraseña:
- Confirmar Contraseña:

En la parte inferior derecha del formulario hay un botón azul con el texto "REGISTRARSE" en blanco.

- Proceso en el backend:

El backend recibe los datos y realiza las validaciones correspondientes (formato de datos válidos, correo electrónico no registrado anteriormente, contraseña coincidente).

Si los datos son correctos, se almacenan en la base de datos en la tabla de usuarios y devuelve una respuesta indicando el registro exitoso.

Si hubo un error, se devolverá un mensaje indicando el mismo. Por ejemplo, “Correo electrónico ya registrado”. La persona podrá intentar nuevamente el registro.

Pantalla de inicio:

En la pantalla de inicio, se mostrarán todos los productos disponibles en la tienda.

Flujo de datos:

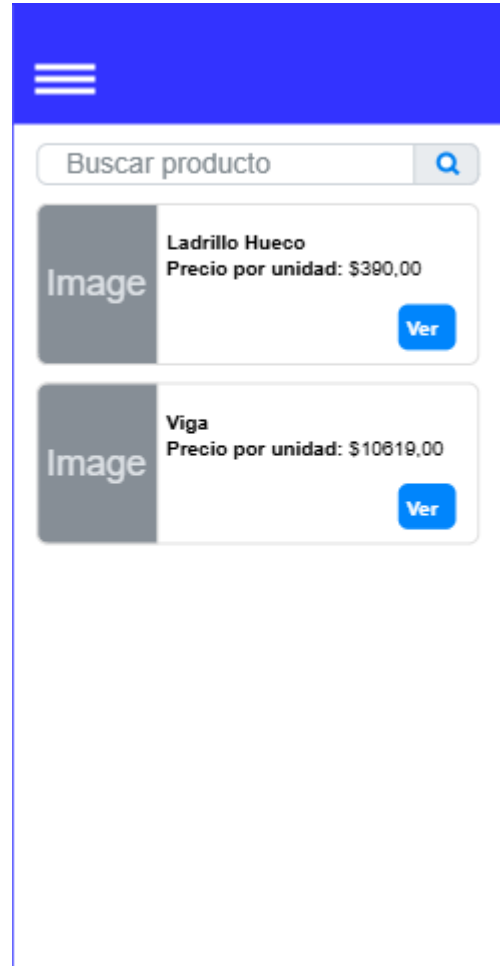
- Del frontend al backend:

El frontend realiza una solicitud GET a la API en el endpoint: /api/productos.

- Del backend al frontend:

El backend responde con una lista de productos, donde cada producto tiene la siguiente estructura:

```
{
  "producto": [
    {
      "id_producto": integer,
      "nombre": string,
      "descripcion": string,
      "uso": string,
      "precio": double,
      "largo": double,
      "alto": double,
      "ancho": double,
      "cantidad_palet": integer,
      "imagen_url": string
    }
  ]
}
```



En el frontend, solo se mostrarán los siguientes datos de cada producto:

- Nombre del producto.

- Precio por unidad del producto.
- Imagen del producto.

Además, cada producto tiene un botón “Ver” que nos permite visualizar toda la información de un producto en específico.

Al tocar el botón "Ver", la aplicación redirige al usuario a la pantalla de detalles del producto. La pantalla de detalles está asociada a una URL o ruta dinámica que incluye el ID del producto. Esto se maneja mediante un sistema de navegación como React Navigation, que es el estándar para gestionar la navegación en aplicaciones React Native.

Detalle de producto:

En esta pantalla se podrá visualizar los datos completos de un producto luego de presionar el botón “Ver” en la página de inicio.

Flujo de datos:

- Del frontend al backend:

El frontend realiza una solicitud GET a la API en el endpoint: `/api/productos/{id_producto}`.

- Del backend al frontend:

El backend responde con los datos del producto:

```
{  
  "id_producto": integer,  
  "nombre": string,  
  "descripcion": string,  
  "uso": string,  
  "precio": double,  
  "largo": double,  
  "alto": double,
```




```
"ancho": double,  
"imagen": url,  
"cantidad_palet": integer,  
"imagen_url": string  
}
```

En el frontend, se mostrarán los siguientes datos del producto:

- Nombre del producto.
- Precio por unidad del producto.
- Imagen del producto.
- Descripción.
- Uso.
- Cantidad por pallet.

También estará el botón “Cotizar” que permitirá realizar una cotización de tal producto.

Al tocar el botón “Cotizar”, la aplicación redirige al usuario a la pantalla de cotización, la cual está asociada a una ruta dinámica que incluye el ID del producto.

Cotización del producto:

En esta pantalla, se presentan dos opciones, la de ingresar las medidas de tu proyecto manualmente, o subir un plano de autocad para obtener automáticamente las medidas.

Flujo de datos:

- Del frontend al backend:

El frontend realiza una solicitud GET a la API en el endpoint: `/api/productos/{id_producto}`.

- Del backend al frontend:

El backend responde con los datos del producto.

```
{  
  "id_producto": integer,  
  "nombre": string,  
  "descripcion": string,  
  "uso": string,  
  "precio": double,  
  "largo": double,  
  "alto": double,  
  "ancho": double,  
  "imagen": url,  
  "cantidad_palet": integer,  
  "imagen_url": string  
}
```

En el frontend, se mostrarán los siguientes datos del producto:

- Nombre del producto.

The screenshot shows a web application interface. At the top, there is a blue header bar with a white hamburger menu icon on the left. Below the header, the main content area has a title "Cotizar 'Ladrillo Hueco'" in bold black text. Underneath the title, there is a white box containing two blue buttons with white text. The first button says "INGRESAR MEDIDAS" and the second button says "SUBIR PLANO DE AUTOCAD".

En el caso de presionar “Ingresar medidas” el usuario será redirigido a otra pantalla donde podrá ingresar las medidas, la cual esta asociada a una ruta dinámica que incluye el ID del producto.

Por el contrario, si el usuario decide “Subir plano de autocad” será redirigido a otra pantalla en la cual podrá subir el archivo correspondiente, esta pantalla está asociada a una ruta dinámica que incluye el ID del producto.

Cotización “Ingresar medidas”:

Aquí el usuario podrá ingresar las dimensiones de su proyecto en metros como largo, ancho y si desea alto.

Flujo de datos:

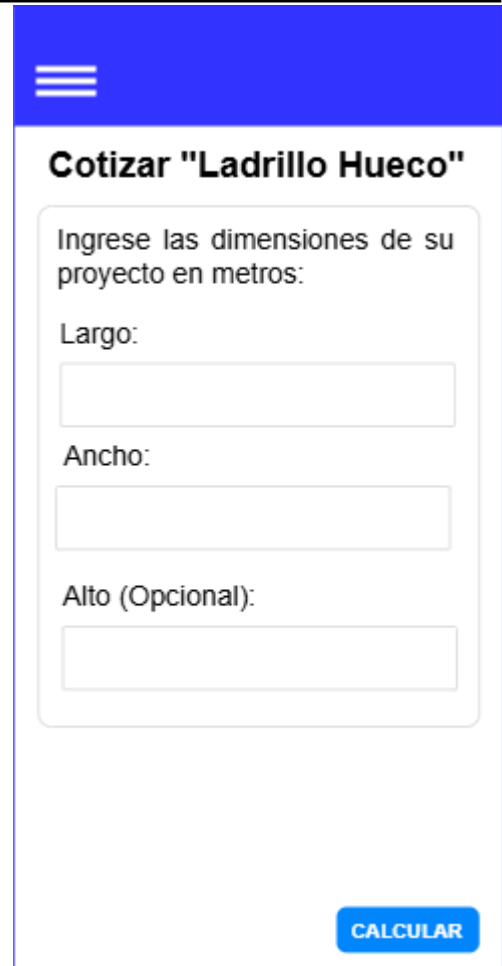
- Del frontend al backend:

El frontend realiza una solicitud GET a la API en el endpoint: /api/productos/{id_producto}.

- Del backend al frontend:

El backend responde con los datos del producto.

```
{  
  "id_producto": integer,  
  "nombre": string,  
  "descripcion": string,  
  "uso": string,  
  "precio": double,  
  "largo": double,  
  "alto": double,  
  "ancho": double,  
  "imagen": url,  
  "cantidad_palet": integer,  
  "imagen_url": string  
}
```



Cotizar "Ladrillo Hueco"

Ingrese las dimensiones de su proyecto en metros:

Largo:

Ancho:

Alto (Opcional):

CALCULAR

En el frontend, se mostrarán los siguientes datos del producto:

- Nombre del producto.

Una vez que el usuario ingresa las medidas y presiona el botón "Calcular", esto se procesará de la siguiente manera:

Flujo de datos:

- Del frontend al backend:

El frontend realiza una solicitud POST a la API en el endpoint: /api/cotizaciones/manual.

```
{
```

```
"id_producto": integer,  
"largo": double,  
"alto": double,  
"ancho": double  
}
```

- Proceso en el backend:

El backend recibe los datos, identifica el metodo de cotizacion y el producto a cotizar en la base de datos y realiza los cálculos necesarios para determinar:

- Unidades requeridas del producto.
- Pallets requeridos.
- Precio total de los productos.
- Descuento del 5% si la cantidad supera las 10.000 unidades.
- Precio total aplicando el descuento.

Y crea un registro en la tabla de cotizaciones. Los datos de la cotización se guardan en el contexto (cotizacionData).

- Del backend al frontend:

```
{  
"id_cotizacion": integer,  
"id_producto": integer,  
"nombre_producto": string,  
"cantidad_necesaria": integer,  
"pallets_necesarios": integer,  
"subtotal": double,  
"descuento": double,  
"total_con_descuento": double  
}
```

El frontend no muestra directamente los resultados en la misma pantalla. En lugar de eso, tras recibir la respuesta del backend con los datos de la cotización, el

usuario es redirigido a una pantalla de resultados de cotización, donde se muestran los detalles calculados.

Cotización “Subir plano de autocad”:

En esta pantalla, el usuario puede subir un archivo del plano de autocad (.dwg o cualquier otro formato aceptado por la API). El archivo será procesado para extraer las medidas necesarias (como largo, ancho y alto) que luego se usarán para calcular la cotización.

Flujo de datos:

- Del frontend al backend:

El frontend realiza una solicitud POST a la API en el endpoint: /api/cotizaciones/autocad

```
{  
  "id_producto": integer,  
  "archivo": "<archivo_dwg>"  
}
```

- Proceso en el backend:

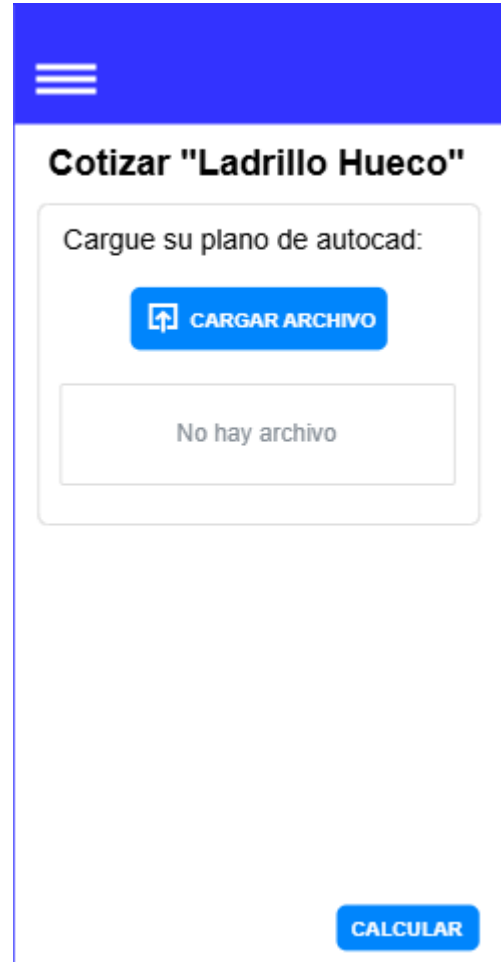
El backend recibe los datos, identifica el metodo de cotizacion y el producto a cotizar en la base de datos.

Recibe el archivo, lo procesa (usando una API externa como Autodesk Forge) y extrae las medidas del plano.

Autodesk Forge es un servicio que permite procesar archivos .dwg y extraer información como dimensiones, ubicaciones de objetos, etc.

Una vez que extrae las medidas, las envía a la API de cotizaciones junto con el ID del producto y realiza los calculos necesarios para determinar:

- Unidades requeridas del producto.
- Pallets requeridos.



- Precio total de los productos.
- Descuento del 5% si la cantidad supera las 10.000 unidades.
- Precio total aplicando el descuento.

Y crea un registro en la tabla de cotizaciones. Los datos de la cotización se guardan en el contexto (cotizacionData).

- Del backend al frontend:

```
{  
  "id_cotizacion": integer,  
  "id_producto": integer,  
  "nombre_producto": string,  
  "cantidad_necesaria": integer,  
  "pallets_necesarios": integer,  
  "subtotal": double,  
  "descuento": double,  
  "total_con_descuento": double  
}
```

El frontend no muestra directamente los resultados en la misma pantalla. En lugar de eso, tras recibir la respuesta del backend con los datos de la cotización, el usuario es redirigido a una pantalla de resultados de cotización, donde se muestran los detalles calculados.

Resultados cotización:

Aquí se mostrarán los resultados obtenidos de la cotización, ya sea por haber ingresado las medidas manualmente o por medio del plano de autocad.

Flujo de datos:

- Proceso en el frontend:

Se mostrarán los datos guardados en el contexto (cotizacionData).

- Del frontend al backend:

Al presionar el botón “Realizar pedido”, el frontend hará una solicitud POST al endpoint: /api/pedido

```
{  
  "id_cotizacion": integer,  
  "id_usuario": integer,  
  "estado": "en proceso"  
}
```

El id_usuario lo recupera del token generado al iniciar sesión.

- Proceso en el backend:

Recibe la solicitud y crea un nuevo registro en la tabla pedidos.

Si el registro del pedido fue exitoso, se redirige al usuario a la pantalla donde deberá completar los datos de envío, dicha pantalla está asociada a una ruta dinámica que incluye el ID del pedido.

Cotización "Ladrillo Hueco"

Unidades necesarias:
Pallets necesarios:

Subtotal:
Total con descuento:

Total a pagar:

REALIZAR PEDIDO

Información de envío:

En esta pantalla el usuario ingresará la información para el envío de su pedido.

Flujo de datos:

- Del frontend al backend:

Luego de que el usuario complete los datos y presione “Continuar”, se realiza una solicitud POST al endpoint: /api/envios

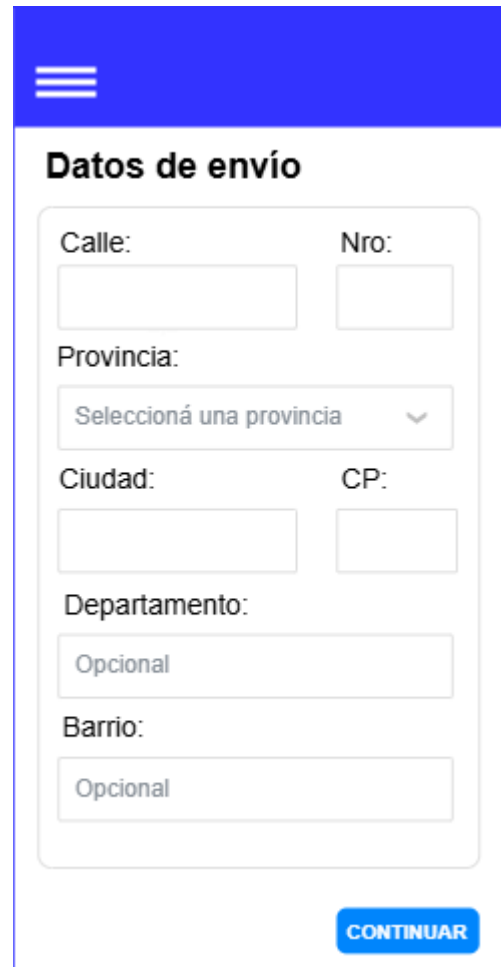
```
{  
  "id_pedido": integer,  
  "calle": string,  
  "nro": integer,  
  "provincia": string,
```

```
"ciudad": string,  
"código_postal": integer,  
"departamento": string,  
"barrio": string  
}
```

- Proceso en el backend:

El backend recibe los datos del envío, crea un nuevo registro en la tabla envíos y asocia el id_envio al pedido en la tabla de pedidos.

Si el proceso fue exitoso, se redirige al usuario a la pantalla donde podrá seleccionar el método de pago, la cual esta asociada a una ruta dinámica que incluye el ID del pedido.



Formulario de Datos de envío. Incluye campos para Calle, Nro, Provincia (seleccionar una provincia), Ciudad, CP, Departamento (Opcional) y Barrio (Opcional). Un botón CONTINUAR está en la parte inferior derecha.

Selección método de pago:

Los usuarios podrán elegir el método de pago de su preferencia, aquí se mostrarán las opciones disponibles.

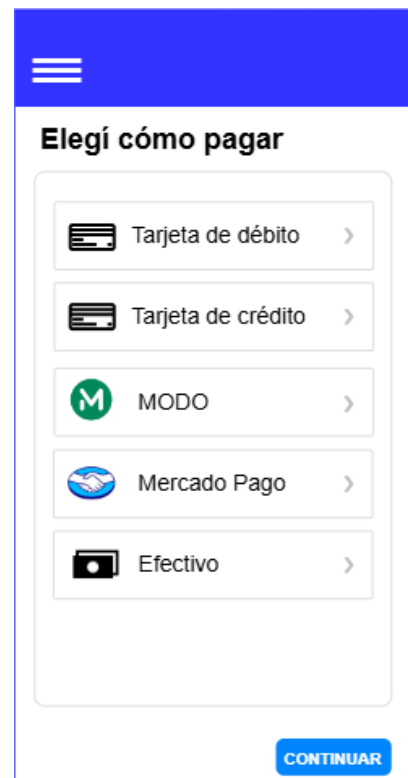
Flujo de datos:

- Del frontend al backend:

El frontend realiza una solicitud GET a la API en el endpoint: /api/metodo-pago

- Del backend al frontend:

```
{  
  "metodo-pago": [  
    {  
      "id": 1,  
      "metodo": "Tarjeta de débito",  
      "icon": "card-debit",  
      "activo": true  
    },  
    {  
      "id": 2,  
      "metodo": "Tarjeta de crédito",  
      "icon": "card-credit",  
      "activo": true  
    },  
    {  
      "id": 3,  
      "metodo": "MODO",  
      "icon": "modo",  
      "activo": true  
    },  
    {  
      "id": 4,  
      "metodo": "Mercado Pago",  
      "icon": "mercado_pago",  
      "activo": true  
    },  
    {  
      "id": 5,  
      "metodo": "Efectivo",  
      "icon": "efectivo",  
      "activo": true  
    }  
  ]  
}
```



Formulario de Elegí cómo pagar. Muestra opciones de pago: Tarjeta de débito, Tarjeta de crédito, MODO, Mercado Pago y Efectivo. Cada opción tiene un icono y un botón de selección. Un botón CONTINUAR está en la parte inferior derecha.


```
"id_metodo ": integer,  
  "nombre": string  
}  
]  
}
```

Una vez que el usuario selecciona un método de pago, es redireccionado a la pantalla que corresponda, esta pantalla está asociada a una ruta dinámica que incluye el ID del método de pago y el ID del pedido.

Tarjeta de crédito/débito:

Si el usuario seleccionó como método de pago tarjeta de débito o crédito, deberá ingresar los datos de la tarjeta.

Flujo de datos:

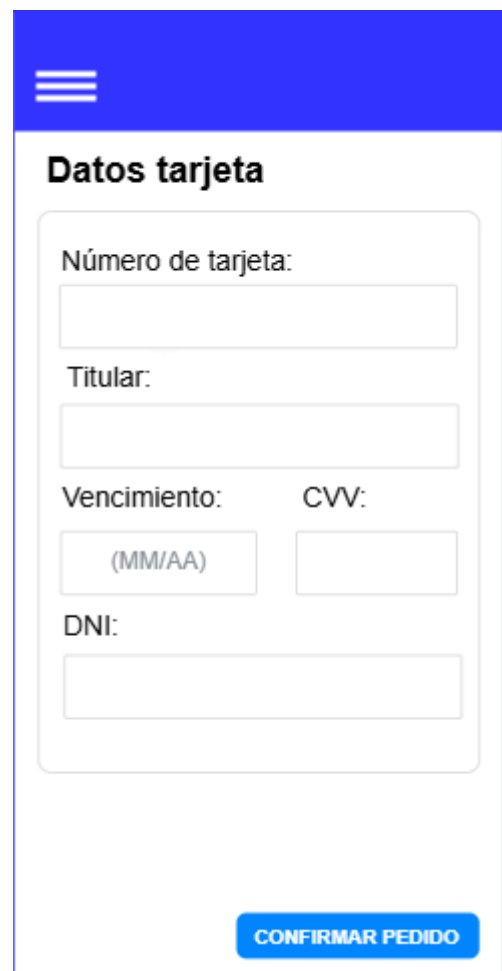
- Del frontend al backend:

Cuando el usuario presione “Confirmar pedido”, el frontend hace una solicitud POST al endpoint: /api/pago-tarjeta

```
{  
  "numero_tarjeta": string,  
  "titular": string,  
  "vencimiento": string,  
  "cvv": string,  
  "dni": string  
  "id_metodo ": integer,  
  "id_pedido ": integer  
}
```

- Proceso en el backend:

El backend recibe los datos y busca el pedido para obtener los datos de la cotización (monto total). Luego procesa dichos datos por medio de una api externa



Formulario de Datos tarjeta:

- Número de tarjeta:
- Titular:
- Vencimiento: CVV:
- DNI:
- Botón: CONFIRMAR PEDIDO

de pagos (por ejemplo: Stripe, PayU, etc.) y verifica el pago, creando un nuevo registro en la tabla pagos.

Si el pago fue exitoso, y se creó el registro de pago se cambia el estado del pedido a "Confirmado".

El backend consulta la tabla puntos para obtener la regla activa (valor del punto y multiplicador). Se calcula el monto de puntos a otorgar ($\text{puntos_nuevos} = \text{monto_total} * \text{valor_punto} * \text{multiplicador}$). Se actualizan los puntos del usuario.

El backend verifica si el usuario ya tiene un registro en la tabla puntos. Si tiene, se actualiza el total de puntos. Si el usuario no tiene un registro en la tabla puntos (es decir, es la primera vez que se le asignan puntos), entonces se crea un nuevo registro en la tabla puntos con el valor de puntos_totales igual a los puntos nuevos calculados.

Si la operación fue exitosa se redirige al usuario a la pantalla final, la cual está asociada a una ruta dinámica que contiene el ID de pedido.

Modo/Mercado Pago:

Si el usuario seleccionó como método de pago MODO o Mercado Pago, será redirigido a la aplicación correspondiente.

Flujo de datos:

- Del frontend al backend:

Al presionar el botón "Confirmar Pedido", se hace una solicitud POST al endpoint: `/api/pago-virtual`

```
{  
  "id_metodo ": integer,  
  "id_pedido ": integer
```



}

- Proceso en el backend:

Se buscarán los datos del pedido (monto total) y dependiendo de la plataforma elegida se utilizarán las APIS externas de las mismas para el procesamiento de los pagos. Una vez verificado el pago, se crea un nuevo registro en la tabla pagos.

Si el pago fue exitoso, y se creó el registro de pago se cambia el estado del pedido a "Confirmado".

El backend consulta la tabla puntos para obtener la regla activa (valor del punto y multiplicador). Se calcula el monto de puntos a otorgar ($\text{puntos_nuevos} = \text{monto_total} * \text{valor_punto} * \text{multiplicador}$). Se actualizan los puntos del usuario.

El backend verifica si el usuario ya tiene un registro en la tabla puntos. Si tiene, se actualiza el total de puntos. Si el usuario no tiene un registro en la tabla puntos (es decir, es la primera vez que se le asignan puntos), entonces se crea un nuevo registro en la tabla puntos con el valor de puntos_totales igual a los puntos nuevos calculados.

Si la operación fue exitosa se redirige al usuario a la pantalla final, la cual está asociada a una ruta dinámica que contiene el ID de pedido.

Efectivo:

Aquí se muestran los detalles en caso de que el usuario seleccionó como método de pago efectivo.

Flujo de datos:

- Del frontend al backend:

Al presionar el botón "Confirmar Pedido", se hace una solicitud POST al endpoint: /api/efectivo

```
{  
  "id_metodo ": integer,  
  "id_pedido ": integer
```

}

- Proceso en el backend:

El backend procesa la solicitud y cambia el estado del pedido a “Pendiente de pago”. Se le envía al usuario un email con los datos del pedido y la información del local.

Si la operación fue exitosa se redirige al usuario a la pantalla final, la cual está asociada a una ruta dinámica que contiene el ID de pedido.

Pantalla final:

Aquí se muestran los datos finales del pedido.

Flujo de datos:

- Del frontend al backend:

Se hace una solicitud GET al endpoint:
`/api/pedido/{id_pedido}`

- Proceso en el backend:

El backend devuelve los datos del pedido, y los puntos del usuario.

- Backend a frontend:

{

```
"id_pedido": integer,  
"id_cotizacion": integer,  
"fecha": date,  
"estado": string,
```

}

En el frontend, solo se mostrarán los siguientes datos

Pedido Nro: xxxx

¡Gracias por tu compra! Hemos recibido tu pago y tu pedido ha sido confirmado. Te mantendremos informado sobre el estado de tu envío.

Ladrillo Hueco x (unidades)

Subtotal:
Total con descuento:
Total a pagar:
Puntos sumados:

[IR A INICIO](#)

- Nombre del producto.
- Subtotal.
- Descuento.
- Total a pagar.
- Puntos sumados por el usuario.

Puntos:

Aquí el usuario podrá consultar sus puntos.

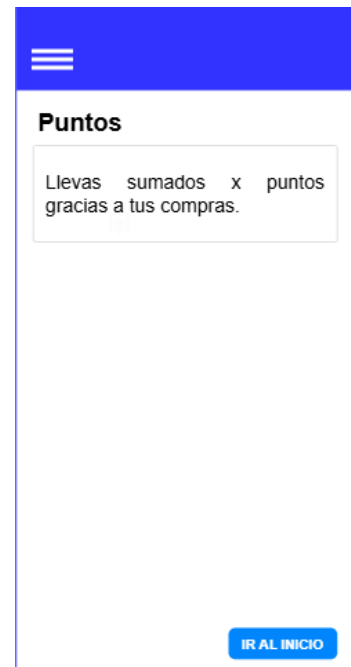
Flujo de datos:

- Del frontend al backend:

Se hace una solicitud GET al endpoint:
`/api/usuario/puntos/{id_usuario}`

- Del backend al frontend:

```
{  
  "puntos_totales": double,  
}
```



8.3. INTERFACES ADMINISTRADOR

Productos

En esta pantalla, el administrador puede consultar, agregar, modificar y eliminar productos.

Flujo de datos:

- Del frontend al backend (Ver productos):

El frontend realiza una solicitud GET a la API en el endpoint: `/api/productos`.

- Del backend al frontend (Lista de productos):

El backend responde con una lista de productos:

```
{
```

```
"productos": [  
  {  
    "id_producto": integer,  
    "nombre": string,  
    "descripcion": string,  
    "uso": string,  
    "precio": double,  
    "largo": double,  
    "alto": double,  
    "ancho": double,  
    "imagen": url,  
    "cantidad_palet": integer  
  }  
]  
}
```

En el frontend, se mostrarán los productos con los siguientes datos:

- Nombre del producto.
- Precio.
- Imagen.
- Descripción.
- Cantidad por pallet.
- Con botones de acción (modificar/eliminar).

- Del frontend al backend (Agregar producto):

El administrador puede completar un formulario con los datos del nuevo producto y enviar una solicitud POST a la API en el endpoint: /api/productos.

```
}  
  
"nombre": string,  
"descripcion": string,  
"uso": string,  
"precio": double,  
"largo": double,
```

```
"alto": double,  
"ancho": double,  
"imagen": url,  
"cantidad_palet": integer  
}
```

- Del frontend al backend (Modificar producto):

Si el administrador decide modificar un producto, se hace una solicitud PUT a la API en el endpoint: /api/productos/{id_producto} con los nuevos datos.

```
}  
  
"nombre": string,  
"descripcion": string,  
"uso": string,  
"precio": double,  
"largo": double,  
"alto": double,  
"ancho": double,  
"imagen": url,  
"cantidad_palet": integer  
}
```

- Del frontend al backend (Eliminar producto):


El administrador puede eliminar un producto mediante una solicitud DELETE al endpoint: /api/productos/{id_producto}.

```
{  
  "id_producto": integer
```


}

[Productos](#) [Pedidos](#) [Puntos](#) [Mi cuenta](#)

Consultar Productos [Agregar](#)



Ladrillo Hueco
Descripción: Ladrillo hueco cerámico 12x18x33 cm 9 tubos. Ladrillo de cerramiento.
Uso: Especiales para tabiques divisorios y cerramientos (ambientes interiores y muros de cierre).
Cantidad por pallet: 144 unidades.
Precio por unidad: \$390,00
[Modificar](#) [Eliminar](#)



Viga
Descripción: 4 mts.
Uso: Son utilizadas para techar en la construcción. Se colocan sobre las paredes y van acompañadas entre viga y viga por ladrillos para techo y malla sima.
Precio por unidad: \$10619
[Modificar](#) [Eliminar](#)

[Productos](#) [Pedidos](#) [Puntos](#) [Mi cuenta](#)

Nuevo Producto

Nombre:

Descripción:

Uso:

Precio:

Imagen:

Largo:

Ancho:

Alto:

[Crear producto](#)

Pedidos:

En esta pantalla, el administrador puede consultar los pedidos, ver sus detalles y cambiar su estado.

Flujo de datos:

- Del frontend al backend (Ver pedidos):

El frontend realiza una solicitud GET a la API en el endpoint: /api/pedidos.

- Del backend al frontend (Lista de pedidos):


```
[  
  {  
    "id_pedido": integer,  
    "id_usuario": integer,  
    "id_cotizacion": integer,  
    "estado": string,  
    "fecha": date  
  },  
  ...  
]
```

En el frontend, se mostrarán los pedidos con los siguientes datos:

- Id pedido.
 - Estado.
 - Botón para Ver los detalles.
- Del frontend al backend (Ver detalles de un pedido):

Cuando el administrador selecciona un pedido, realiza una solicitud GET a la API en el endpoint: `/api/pedidos/{id_pedido}`.

- Del backend al frontend (Detalles del pedido):

El backend responde con los detalles del pedido, que incluirán información adicional como la lista de productos, la dirección de envío y el estado del pago:

En el frontend, se mostrarán los siguientes detalles:

- Dirección de envío.
 - Producto del pedido (nombre, cantidad, precio).
 - Total del pedido.
 - Método de pago.
 - Fecha de compra.
 - Además, se presentará la opción para cambiar el estado del pedido (si corresponde, por ejemplo, "Pendiente", "Pagado", "Enviado").
- Del frontend al backend (Cambiar estado del pedido):

Si el administrador cambia el estado de un pedido (por ejemplo, de “Pendiente” a “Pagado”), realiza una solicitud PUT a la API en el endpoint: /api/pedidos/{id_pedido}.

```
{  
  "estado": "Pagado"  
}
```

ProductosPedidosPuntosMi cuenta

Consultar Pedidos

Pedido: xxxxEstado: estadoVer pedido

Pedido: xxxxEstado: estadoVer pedido

Pedido: xxxxEstado: estadoVer pedido

Pedido: xxxxEstado: estadoVer pedido

ProductosPedidosPuntosMi cuenta

Ver pedido:

Pedido: xxxxEstado: estadoCambiar estado

Información pedido:

Información envío:

Volver

Puntos (Configuración de reglas de puntos):

En esta pantalla, el administrador puede modificar la regla de conversión de puntos.

Flujo de datos:

- Del frontend al backend (Ver regla de puntos actual):

El frontend realiza una solicitud GET a la API en el endpoint: /api/puntos.

- Del backend al frontend (Regla de puntos):

El backend responde con la regla actual de puntos:

```
{  
  "valor_punto": integer,  
  "multiplicador": integer  
}
```

En el frontend, se mostrará el valor del punto y su multiplicador actual, con la opción de modificarlo.

- Del frontend al backend (Modificar regla de puntos):

El administrador selecciona el valor del punto (por ejemplo, 1 punto cada 1 peso o 2 puntos por peso) mediante un select o un formulario, y envía la nueva regla a la API en el endpoint: /api/puntos.

```
{  
  "valor_punto": integer,  
  "multiplicador": integer  
}
```

Productos

Pedidos

Puntos

Mi cuenta

Gestionar Puntos:

Regla puntos:

1



punto por

1



peso \$

Guardar cambios