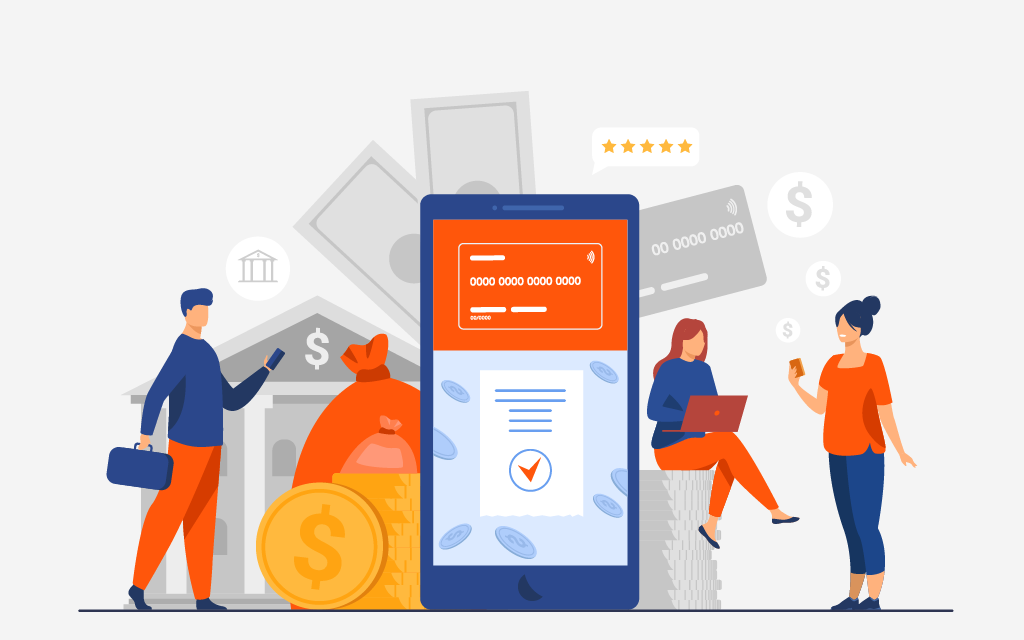
**PROYECTO FINAL**

**SISTEMA DE FACTURACIÓN**

Integrantes:

Montaña Martin

Sanchez Geronimo

Weinzettel Eduardo

Lich Cristian

Enunciado

Como miembros de una empresa que vende soluciones informáticas a diversos clientes, debemos generar una herramienta de gestión mediante la cual cada cliente tenga acceso a un panel de control para monitorear los servicios contratados y la información asociada a los mismos.

Habitualmente, estas herramientas se conocen con el nombre de Dashboards.

En particular, uno de nuestros principales productos es un sistema de facturación, por lo que este proyecto será un Dashboard para Sistema de Facturación.

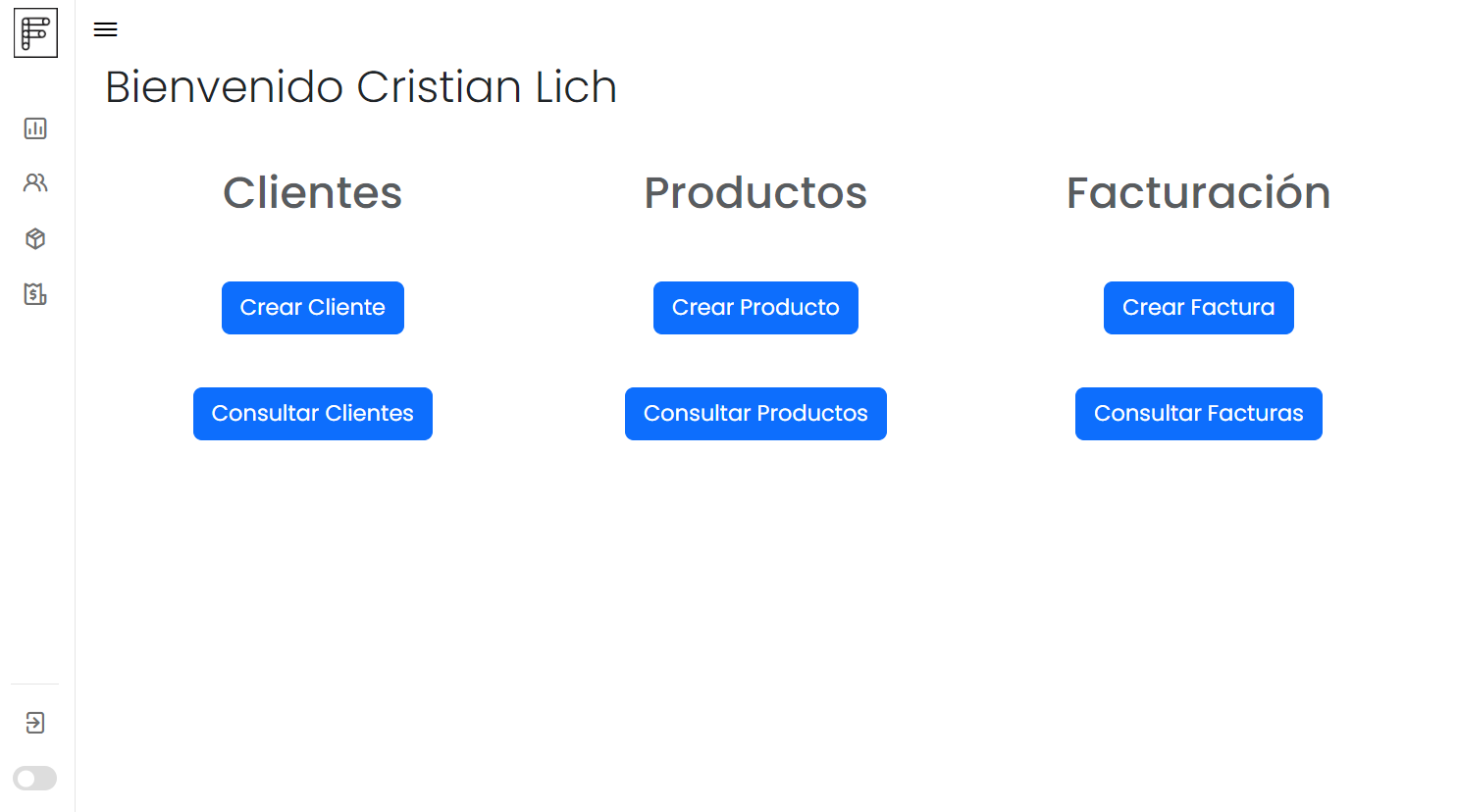
# Presentación de la página web

Lo primero que vemos al abrir la URL es la página de ingreso donde aparece la opción de Logueo.

Es necesario tener una cuenta para acceder a las funcionalidades del sistema.



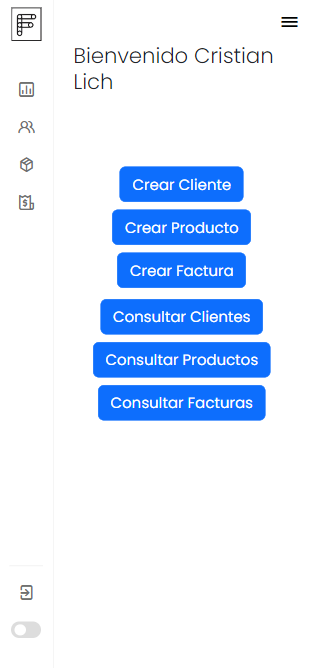
Luego de loguearse podemos comenzar a utilizar todas las funcionalidades que la página nos ofrece



# 

# Adaptación a diferentes dispositivos

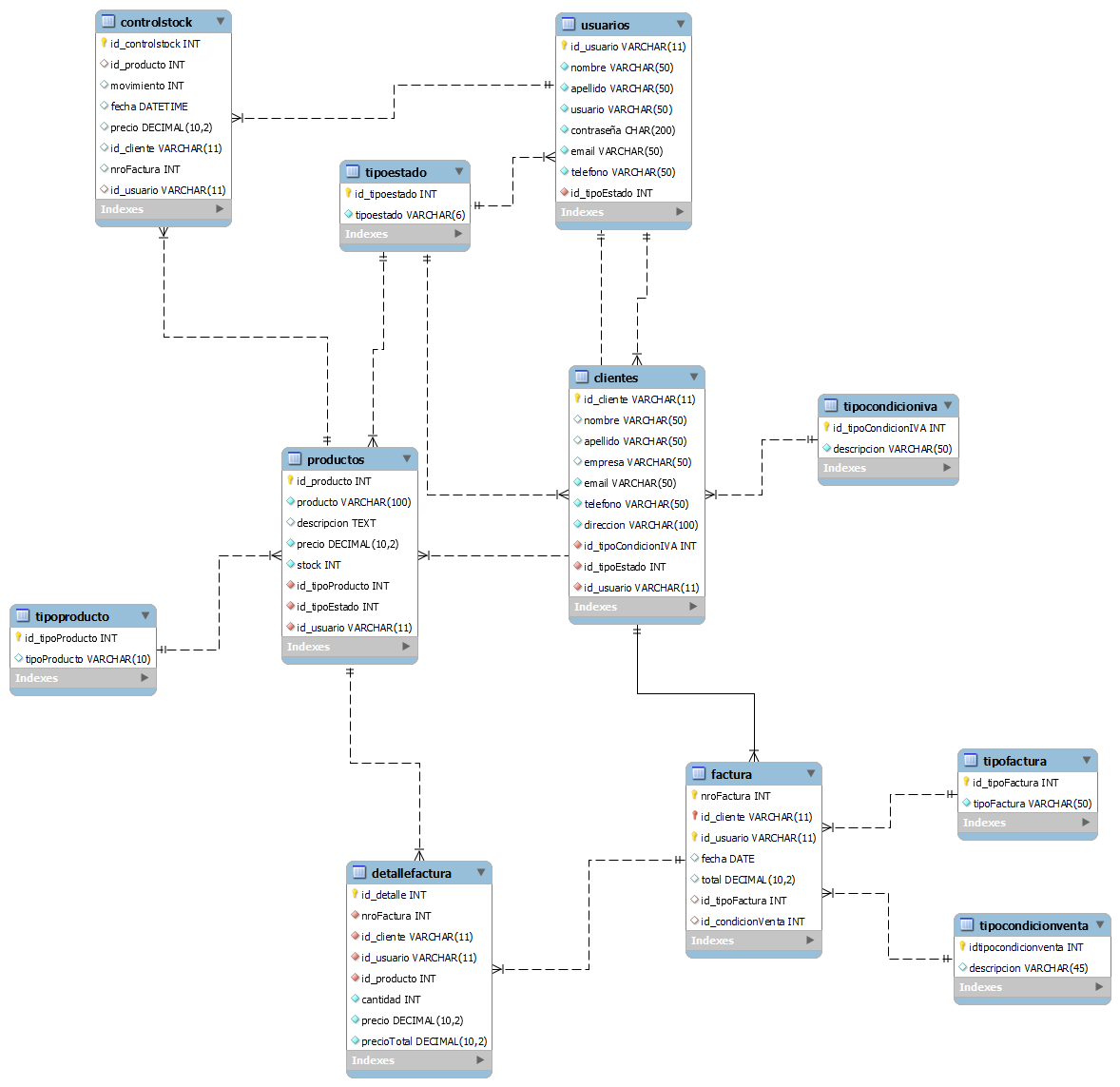
En esta instancia la página ofrece una adaptación para dispositivos móviles



# 

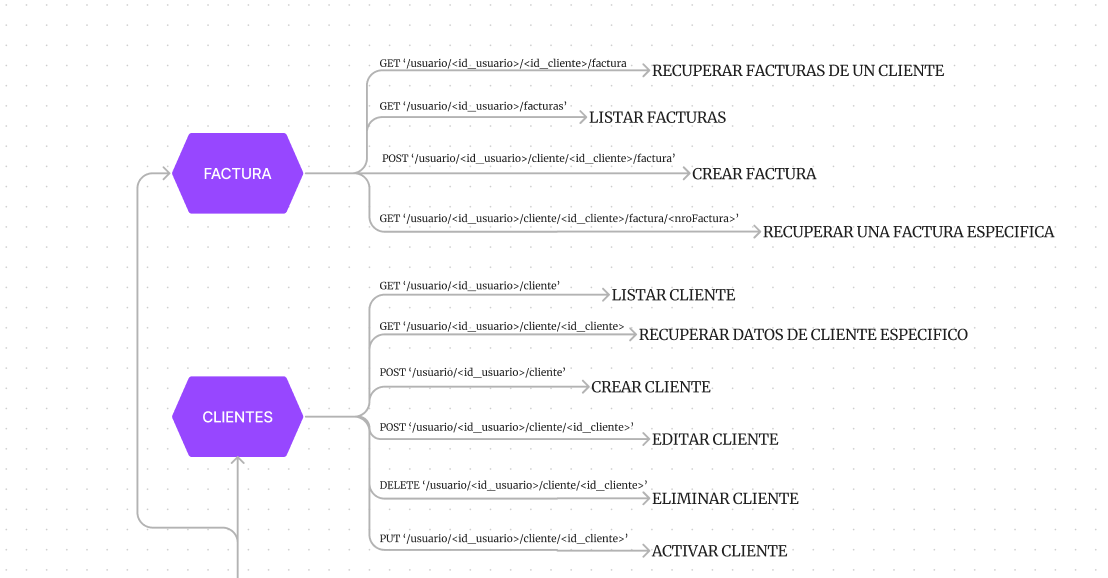
# 

# ESQUEMA UML DE LA BDD



Para nuestra app se decidió crear procedimientos almacenados para realizar las consultas a la base de datos por parte del backend.

**Listado de Store Procedures.**

`sp\_actualizarCliente`, `sp\_actualizarProducto`, `sp\_altaCliente`, `sp\_altaProducto`, `sp\_dashboard\_clientesActivos`, `sp\_dashboard\_clientesTotales`, `sp\_dashboard\_controlStock`, `sp\_dashboard\_historialVentas`, `sp\_dashboard\_movimientoStock`, `sp\_dashboard\_rankingVentasByCliente`, `sp\_dashboard\_rankingVentasByProducto`, `sp\_dashboard\_rankingVentasByServicio`, `sp\_dashboard\_ventasTotales`, `sp\_eliminarCliente`, `sp\_eliminarProducto`, `sp\_insertarCliente`, `sp\_insertarFactura`, `sp\_insertarFacturaDetalle`, `sp\_insertarProducto`, `sp\_listarTipoCondicionIVA`, `sp\_listarTipoCondicionVenta`, `sp\_listarTipoFactura`, `sp\_listarTipoProducto`, `sp\_loginUsuario`, `sp\_obtenerClienteById\_Cliente`, `sp\_obtenerClientesByUsuario`, `sp\_obtenerDetalleFactura`, `sp\_obtenerFacturaByCliente`, `sp\_obtenerFacturasByCliente`, `sp\_obtenerFacturasById\_Usuario`, `sp\_obtenerId\_UsuarioById\_Cliente`, `sp\_obtenerIDtipoCondicionIVAbyDescripcion`, `sp\_obtenerProductoByUsuario`, `sp\_obtenerProductosByUsuario`, `sp\_obtenerPWDByUsuario`, `sp\_obtenerStockPorProducto`

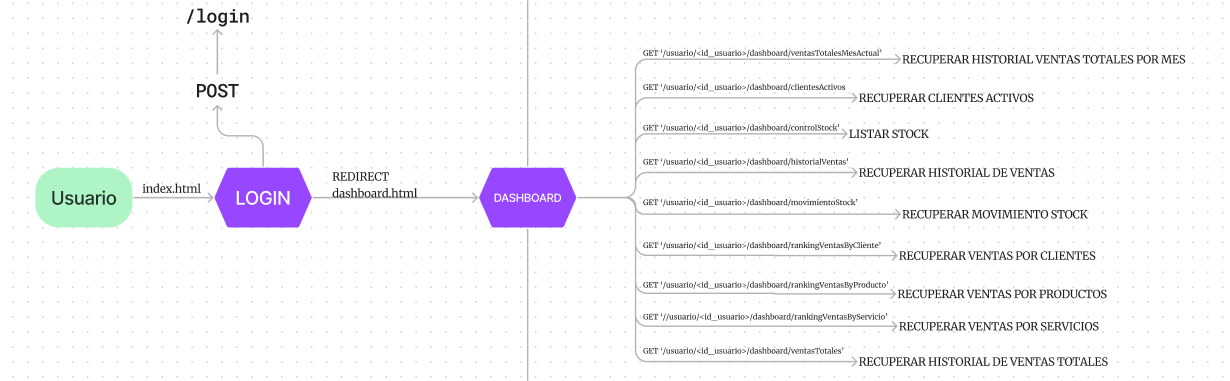
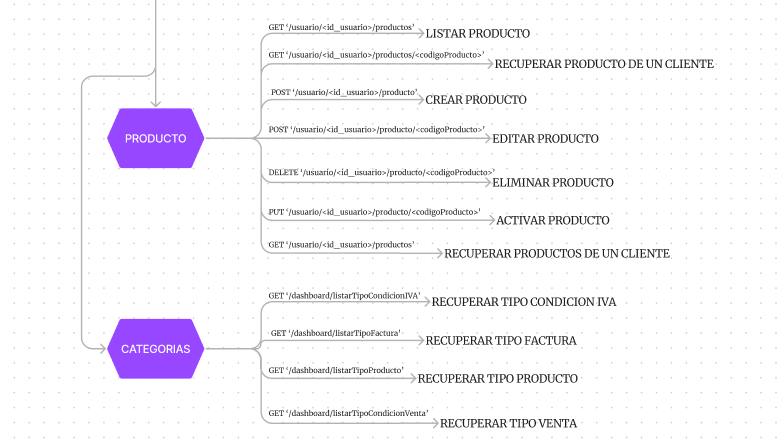
# Mapa de Backend

# 

# 

# 

# 



# 

# 

# Mapa de sitio

# 

# 

# Organización del proyecto

Dividimos el proyecto en dos ramas principales *Backend* y *Frontend* utilizando GitHub como herramienta de gestión de versionado*.* A partir de estas ramas se crearon las diferentes funcionalidades, tanto de la APIRestfull como de la página misma.

Una vez realizada la Base de datos se comenzó con las rutas y modelos de cada una de las entidades mientras que el se realizaba el esqueleto HTML para recibir y mostrar los datos al usuario. Terminada la Api se comprobaron las rutas usando Thunder Client como herramienta de testeo y se utilizó Javascript vanilla para la interacción del cliente con el servidor usando Promesas para comunicarnos con la Api y el backend.

# REVIEW

Durante la modelación de la base de datos en un comienzo, surgieron problemáticas con respecto a ¿cuáles datos son necesarios que persistan? y esto fue en la tabla factura, ya que hay datos como el “Precio” que debe estar en la tabla factura como en la tabla productos, superando este planteamiento surgieron muchos errores o necesidades que fueron subsanadas en el transcurso de este proyecto a medida que el Backend realizaba las rutas paralelamente se realizaba el Frontend, cuyo desarrollo presentó otras dificultades y en cierto sentido más variadas, realizado el esqueleto HTML y CSS se decidió por mudar el proyecto usando Bootstrap en su versión 5.3, lo cual resolvió gran parte del diseño por su flexibilidad a lo que sumando al Javascript. Otro gran desafío que afrontó el equipo fue la escasa experiencia con el lenguaje Javascript, lo que resultó en horas de estudio y tutoriales para poder llevar a cabo la unión del Front con el Back y las funcionalidades de cada sección.

Por último se procedió a testear manualmente, registrar errores y corregirlos para depurar la app, de los cuales solo algunos fueron agregados.

# Bugs encontrados

En la siguiente lista se puede observar el trabajo que realizó el equipo para testear la App y se reportaron los siguientes bugs, donde se muestran ordenados por severidad (de menor a mayor).

**EVIDENCIA DE PRUEBA**

| Fecha: | 27 Noviembre 2023 | Versión: | 1.01 | Plataforma: | App |
| --- | --- | --- | --- | --- | --- |
| Resumen: | Verificar opción “Editar cliente” | | | | |
| Defecto: | Al momento de repetir la acción se cargan datos erróneos en los campos | | | | |
| Tester | Cristian Lich | | | | |

| Se procede a testear cambiando campos de los datos de clientes y confirmarlos repetidamente de manera rápida. |
| --- |
|  |

# 

**EVIDENCIA DE PRUEBA**

| Fecha: | 27 Noviembre 2023 | Versión: | 1.01 | Plataforma: | App |
| --- | --- | --- | --- | --- | --- |
| Resumen: | Verificar Carga de cantidad en la creación de una factura | | | | |
| Defecto: | Permite cargar cualquier número positivo | | | | |
| Tester | Cristian Lich | | | | |

| Se procede a testear campos de entrada colocando valores fuera del rango válido o tipo diferentes a los necesarios |
| --- |
|  |

# 

**EVIDENCIA DE PRUEBA**

| Fecha: | 30 Noviembre 2023 | Versión: | 1.01 | Plataforma: | App |
| --- | --- | --- | --- | --- | --- |
| Resumen: | Verificar stock de Servicios | | | | |
| Defecto: | Muestra stock negativo en servicios | | | | |
| Tester | Cristian Lich | | | | |

| Se procede a testear campo stock de sección Productos, donde se generan nuevas facturas agregando servicios y sus cantidades. |
| --- |
|  |

# 

**EVIDENCIA DE PRUEBA**

| Fecha: | 30 Noviembre 2023 | Versión: | 1.01 | Plataforma: | App |
| --- | --- | --- | --- | --- | --- |
| Resumen: | Verificar stock en Facturas | | | | |
| Defecto: | Permite volver a agregar la cantidad máxima de stock | | | | |
| Tester | Cristian Lich | | | | |

| Se procede a testear control de stock en la creación de factura, donde coloca repetidamente la cantidad máxima en stock permitida. |
| --- |
|  |

# 

# 

# API Testing

Para este caso trabajamos con la herramienta THUNDER CLIENT a través del IDE Visual Studio Code, donde fuimos realizando distintos request, modificando endpoints y asignando valores a diferentes parámetros solicitados.

***PARÁMETROS SOLICITADOS.***

HEADERS

* id\_usuario
* Token

ROUTER

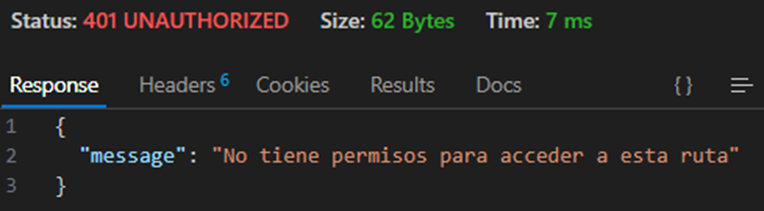
* id\_cliente
* id\_usuario
* CodigoProducto
* nroFactura

MANEJO DE ERRORES.

Utilizamos el estándar para el manejo de errores siendo estos los códigos de status HTTP.

* Error 401 – Status: 401 UNAUTHORIZED - Error de Autorización.

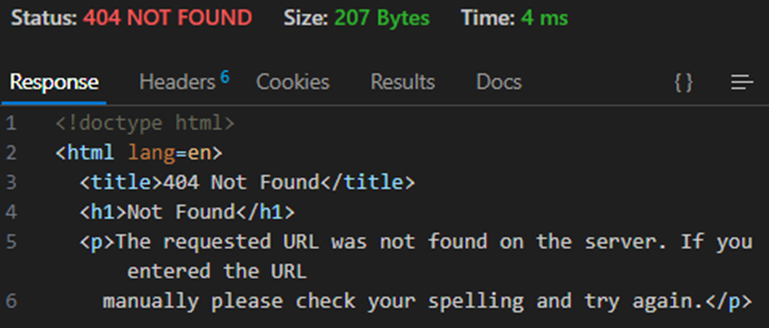
Denota inconveniente de Autorización al intentar realizar la acción requerida.



**Ejemplos**.

* Al intentar acceder al servidor y este no está levantado.
* Al intentar realizar alguna acción sobre un cliente, Factura, producto que no pertenezca a ese usuario, que falte algún dato de los que solicita el backend (Token, id-User,, id-Cliente, etc) para realizar esa acción.
* Al intentar realizar acciones sin tener el token, si este no es el correcto o haya expirado.
* Al actualizar un producto y el id-producto sea incorrecto
* Error 404 - Status: 404 NOT FOUND – No encontrado.

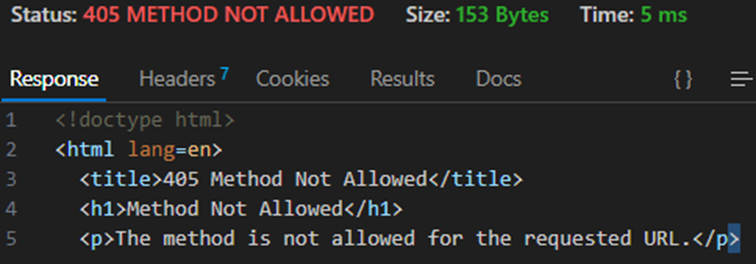
Ocurre cuando se intenta acceder a un cliente, factura, producto sin el dato de usuario en la ruta.



**Ejemplos**.

* Al intentar realizar una acción sobre un cliente, producto, factura sin pasar id-usuario. (En operativo y en funcionamiento normal, no se dará, ya que esos datos si o si, se pasan en la ruta, por haber realizado click en crear cliente/producto/factura. Surge en modo pruebas con thunder client,
* Error 405 - Status: 405 METHOD NOT ALLOWED – Acción No permitida

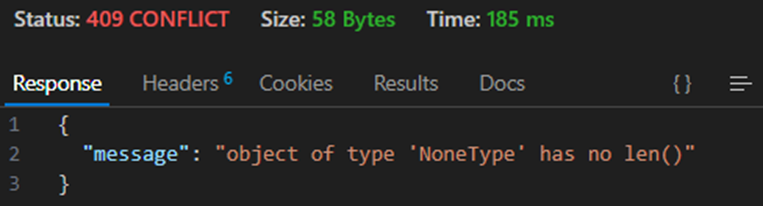
Muestra que esa acción no se puede realizar.



**Ejemplos**

* Al intentar eliminar un cliente de un usuario sin pasar el id-cliente. (En operativo y en funcionamiento normal, no se dará, ya que esos datos si o si, se pasan en la ruta, por haber realizado el click en el cliente a eliminar. Surge en modo pruebas con thunder client,
* Error 409 – Status: 409 CONFLICT - Error Retorno de Len().

La búsqueda retorna un Len(), vacío, No encontró lo solicitado.



**Ejemplos**.

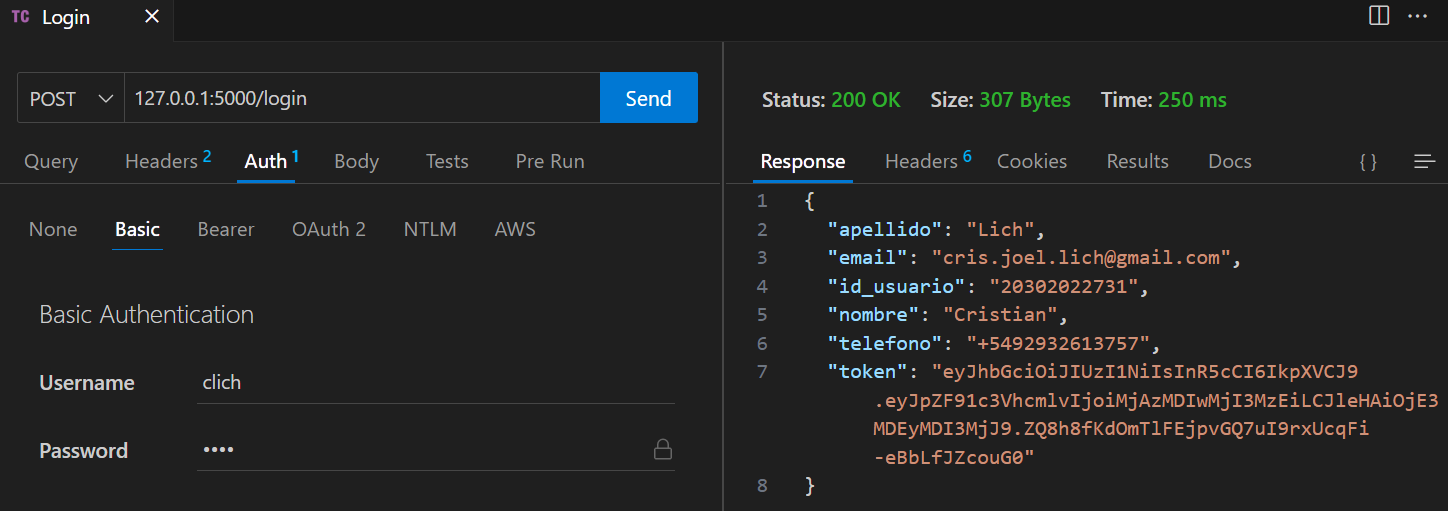
* Al Consultar una factura inexistente para el cliente.

***ROUTING***

LOGIN.-

'/login' methods = ['POST']

La ruta login implementa un tipo de autorización básica y la misma nos devuelve un token que será necesario para validar cualquier operación que modifique la base de datos.

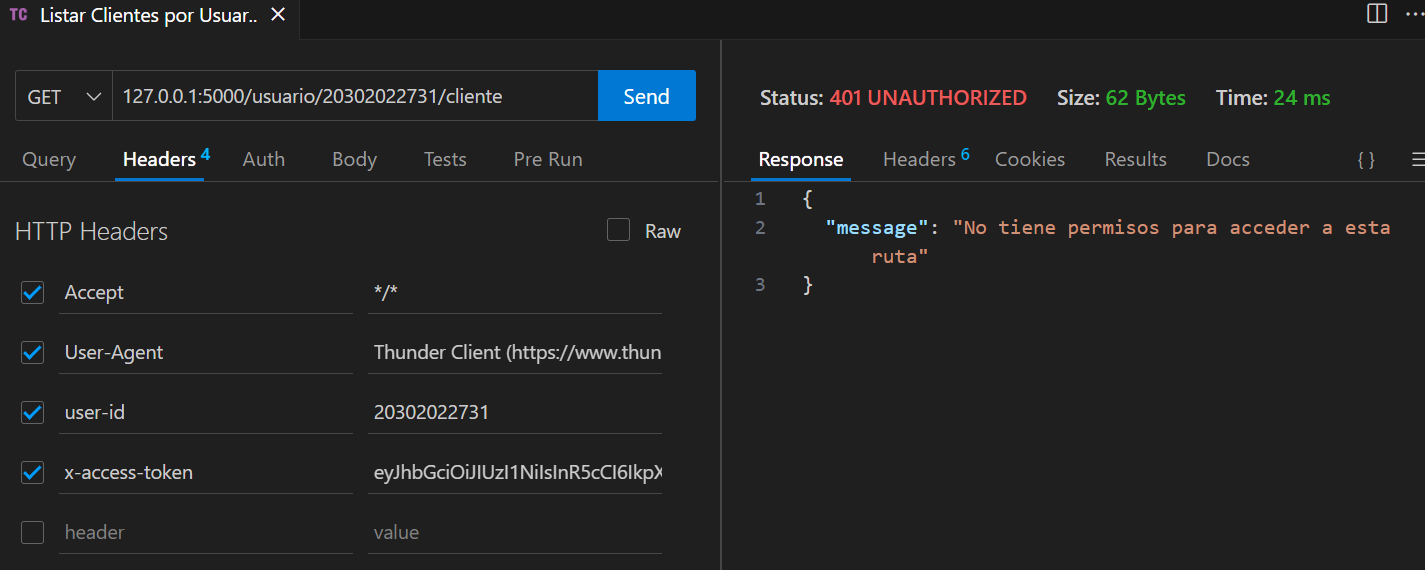


CLIENTES.-

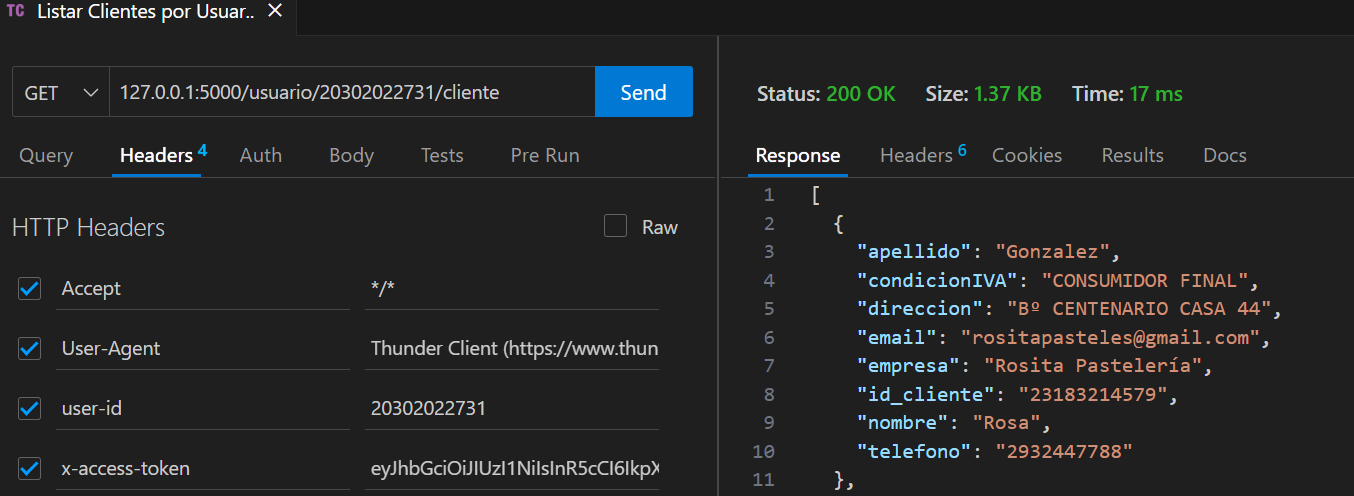
Obtener todos los clientes de un usuario

'/usuario/<id\_usuario>/cliente', methods = ['GET']

(en caso de querer ingresar a la ruta directamente la app nos devuelve el código de estado 401 con el mensaje en formato JSON)



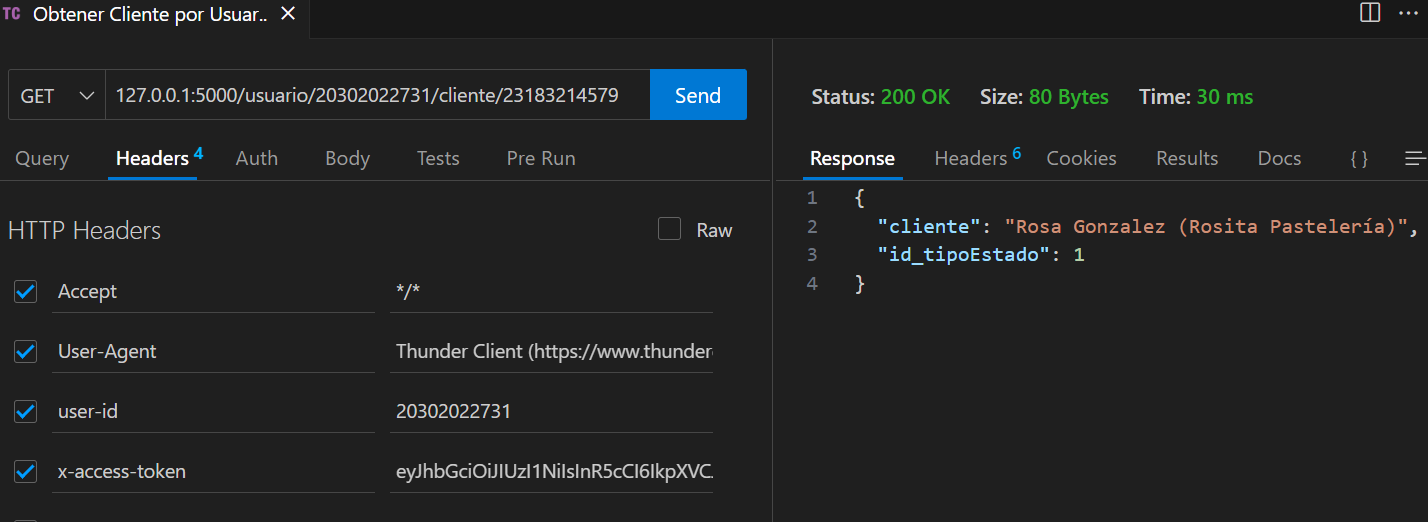
El Status 200 nos arroja un listado de clientes con el siguiente formato



Obtener datos de un cliente por usuario.-

'/usuario/<id\_usuario>/cliente/<id\_cliente>', methods = ['GET']

En esta ruta es necesario además de los datos en el header hay que pasar el id del cliente del cual necesitamos conocer los datos.

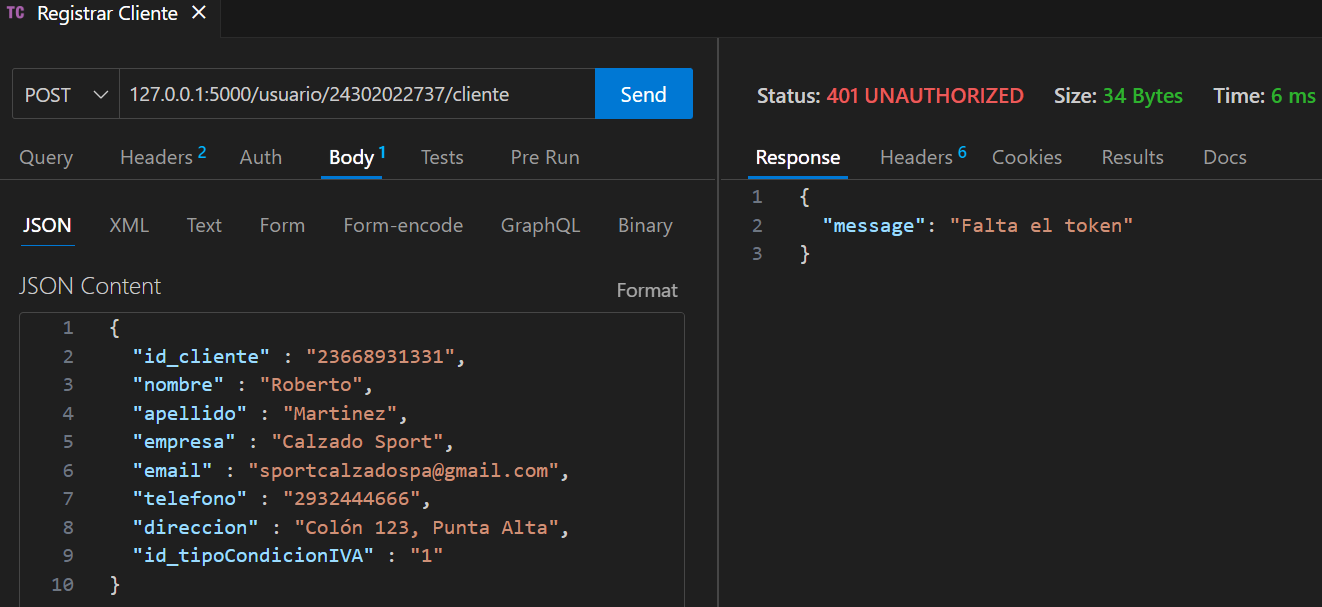


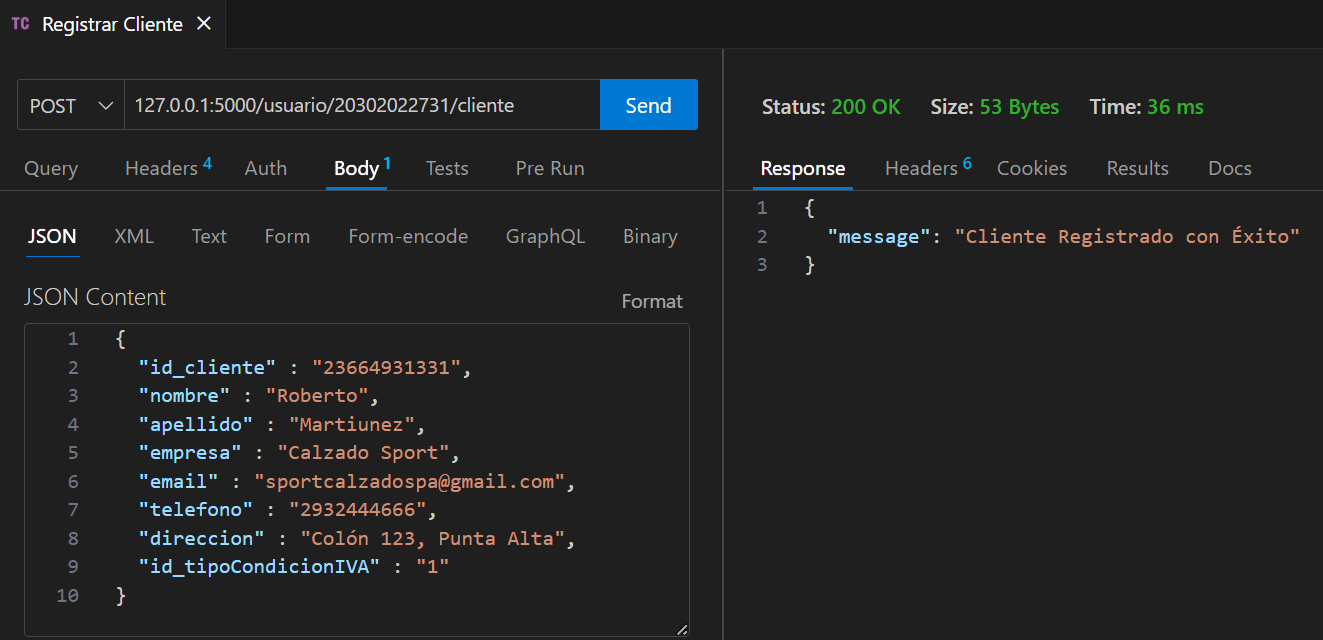
Registrar Cliente.-

'/usuario/<id\_usuario>/cliente', methods = ['POST']

Para enviar la solicitud de creación del elemento en este caso hay que enviar en el body del método el objeto completo con los siguientes datos.

(ejemplo de error cuando falta algún dato en el header, es decir, si no esta logueado)

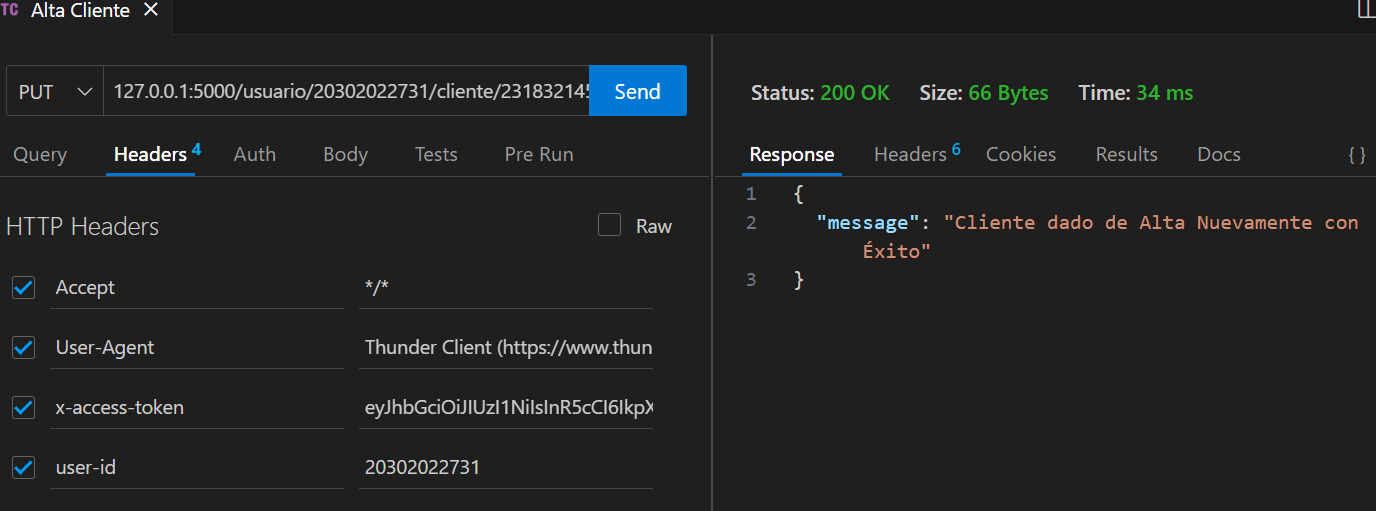




Alta de Cliente

'/usuario/<id\_usuario>/cliente/<id\_cliente>', methods = ['PUT']

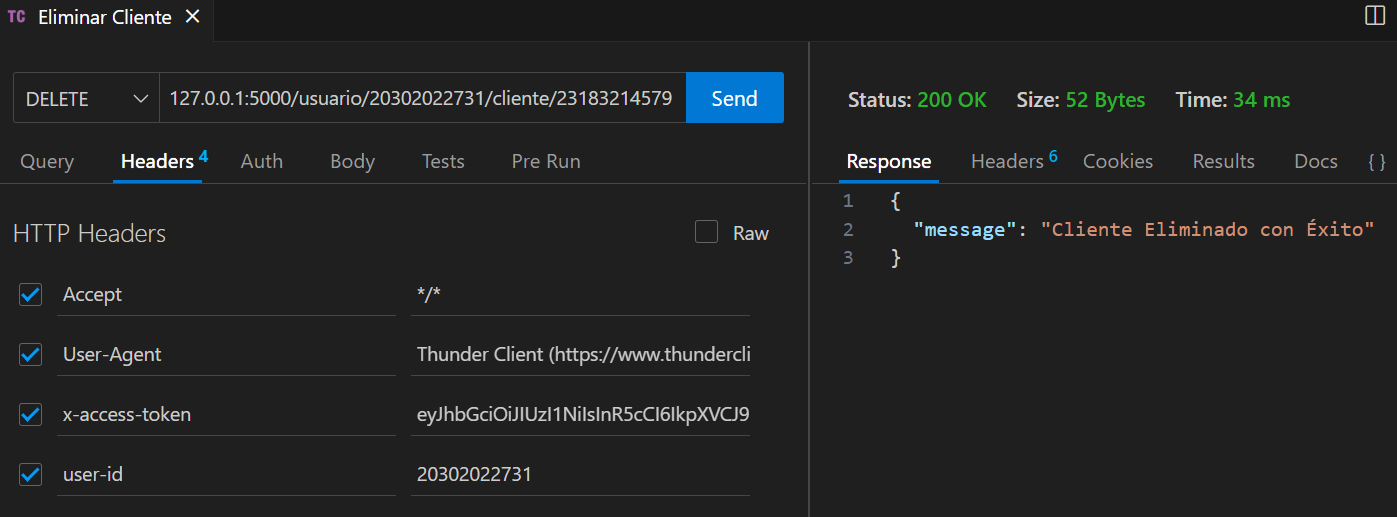
En la app se implementó el borrado lógico, lo que permite conservar todos los datos guardados sobre facturación, productos y clientes, por lo cual en caso de tener un estado de “baja” solo hace falta cambiar el estado a “alta”



Eliminar Cliente.-

'/usuario/<id\_usuario>/cliente/<id\_cliente>', methods = ['DELETE']

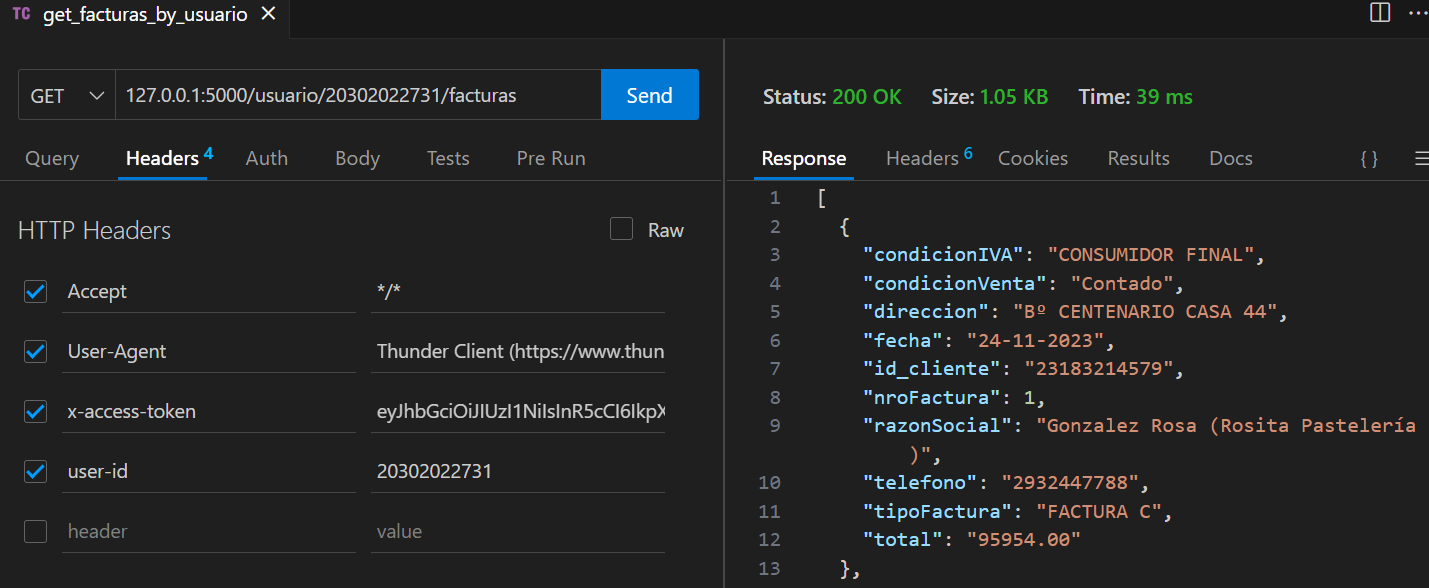
Como se explicó anteriormente, el método delete internamente se implementó como un estado binario de “alta” o “baja”.



**FACTURA**

Obtener facturas por usuario

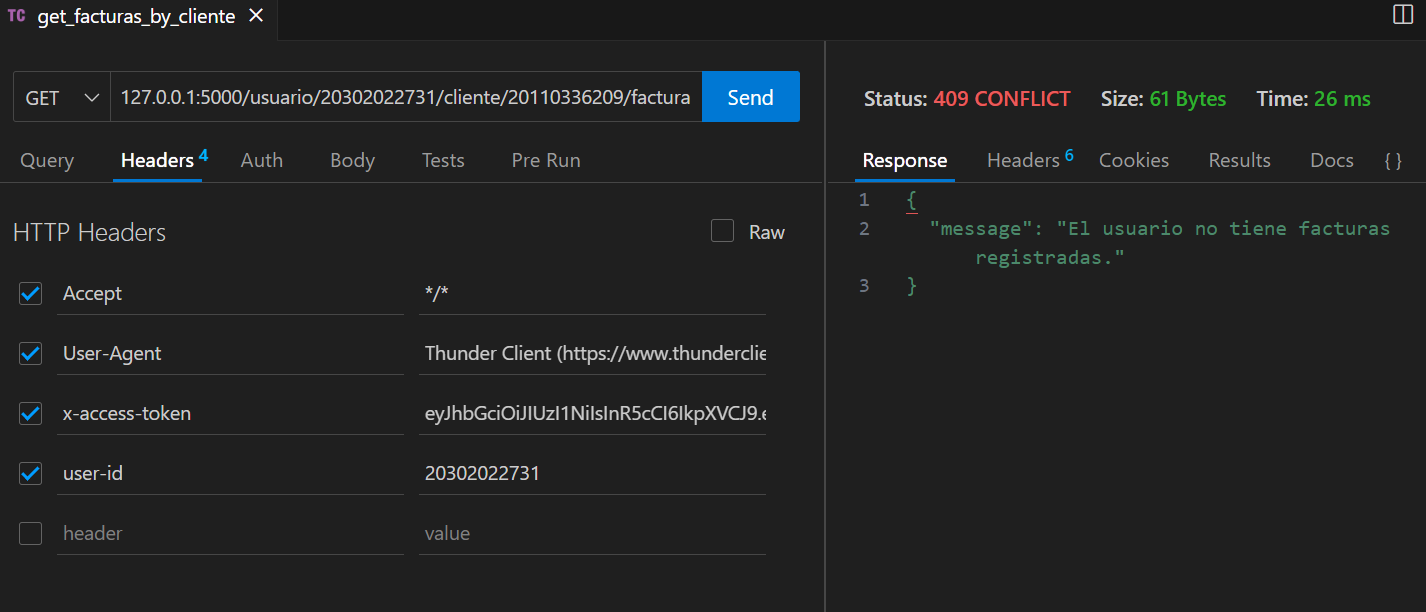
'/usuario/<id\_usuario>/facturas', methods = ['GET']

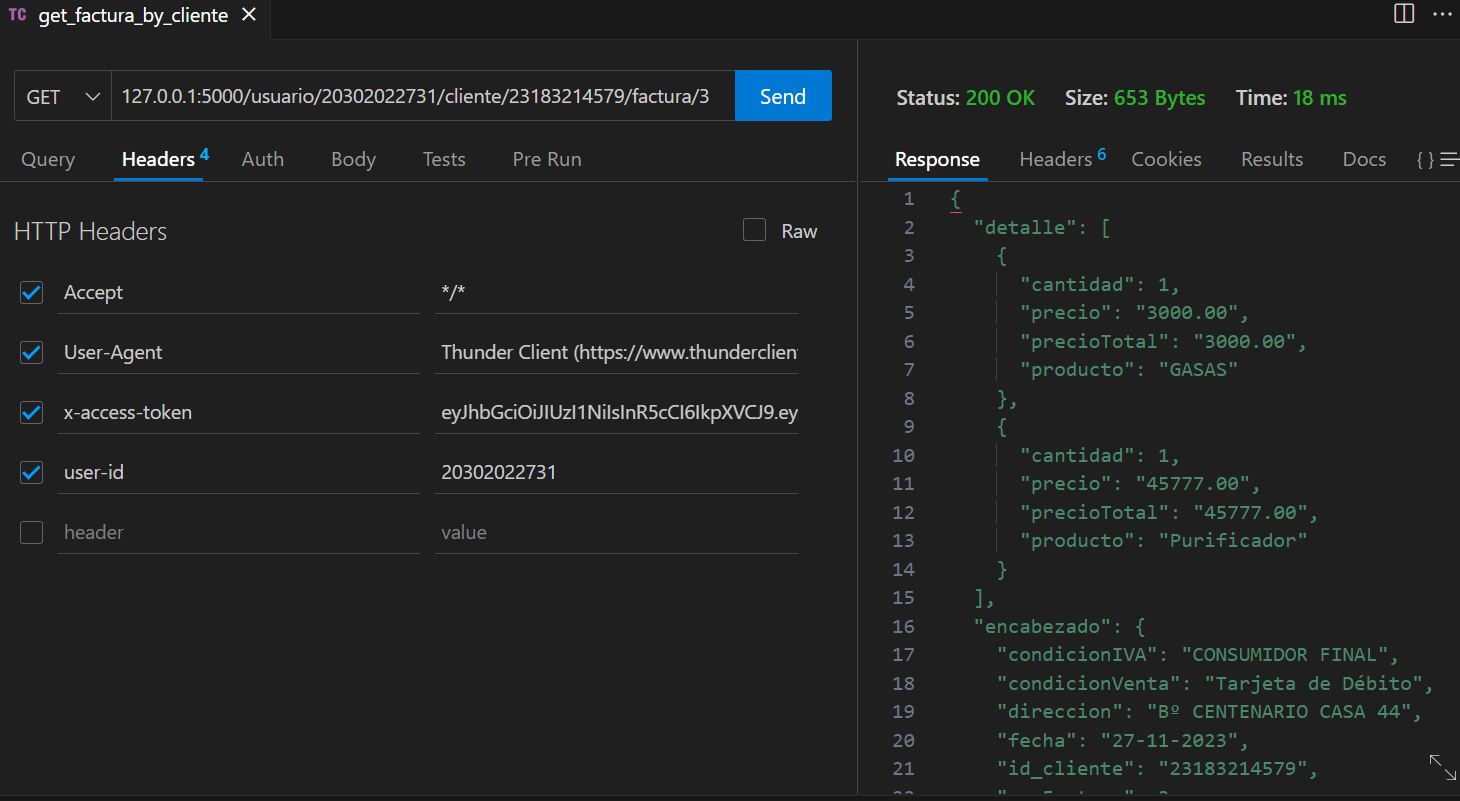


Obtener factura de un cliente.-

'/usuario/<id\_usuario>/cliente/<id\_cliente>/factura', methods = ['GET']

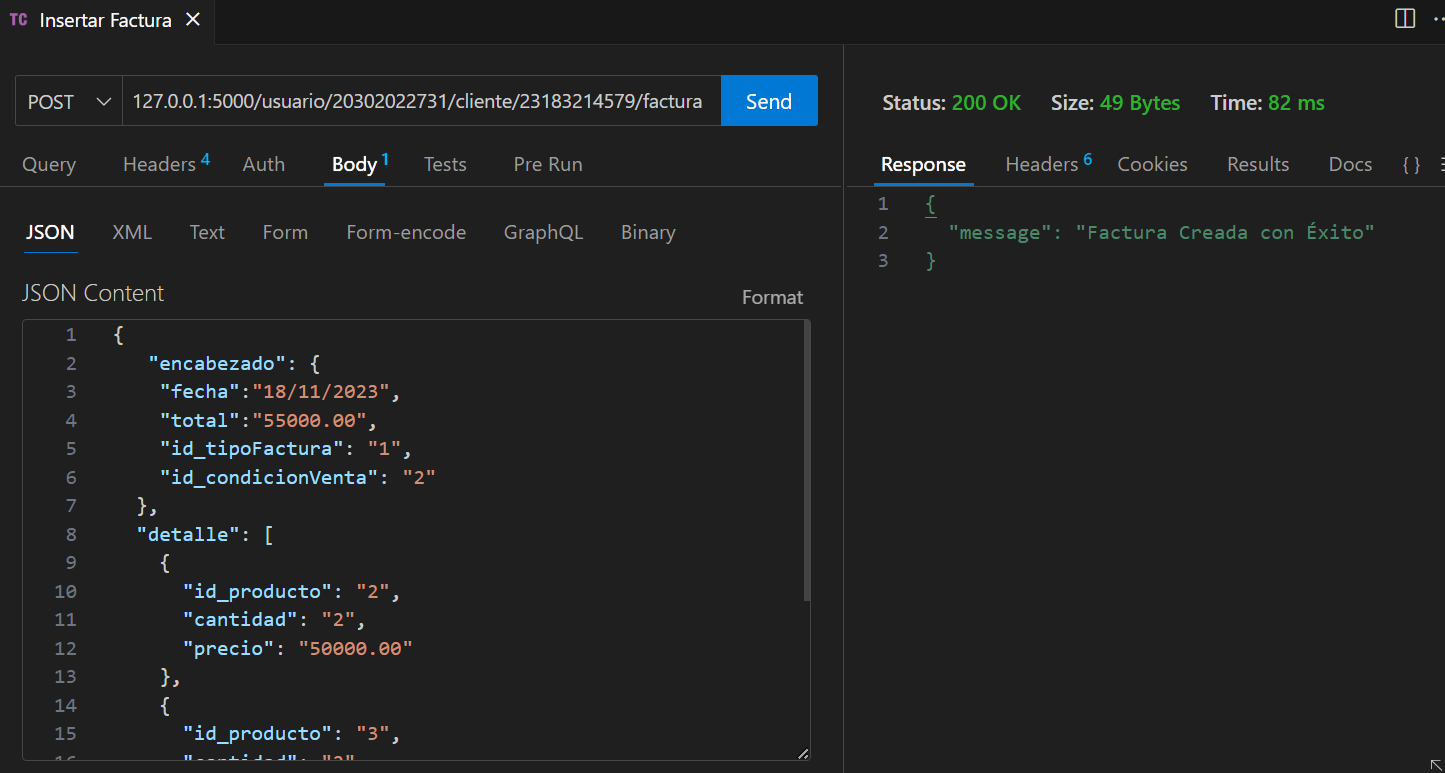
En caso de no tener facturas se mostrara el código de status 409





Insertar factura.-

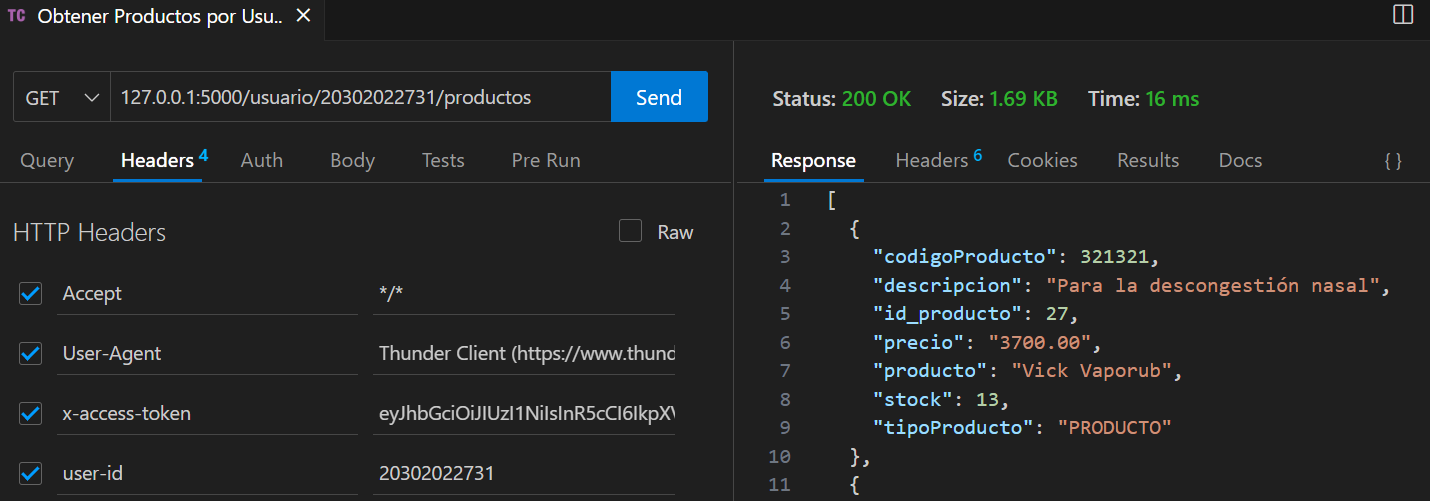
'/usuario/<id\_usuario>/cliente/<id\_cliente>/factura', methods = ['POST']



**PRODUCTOS**

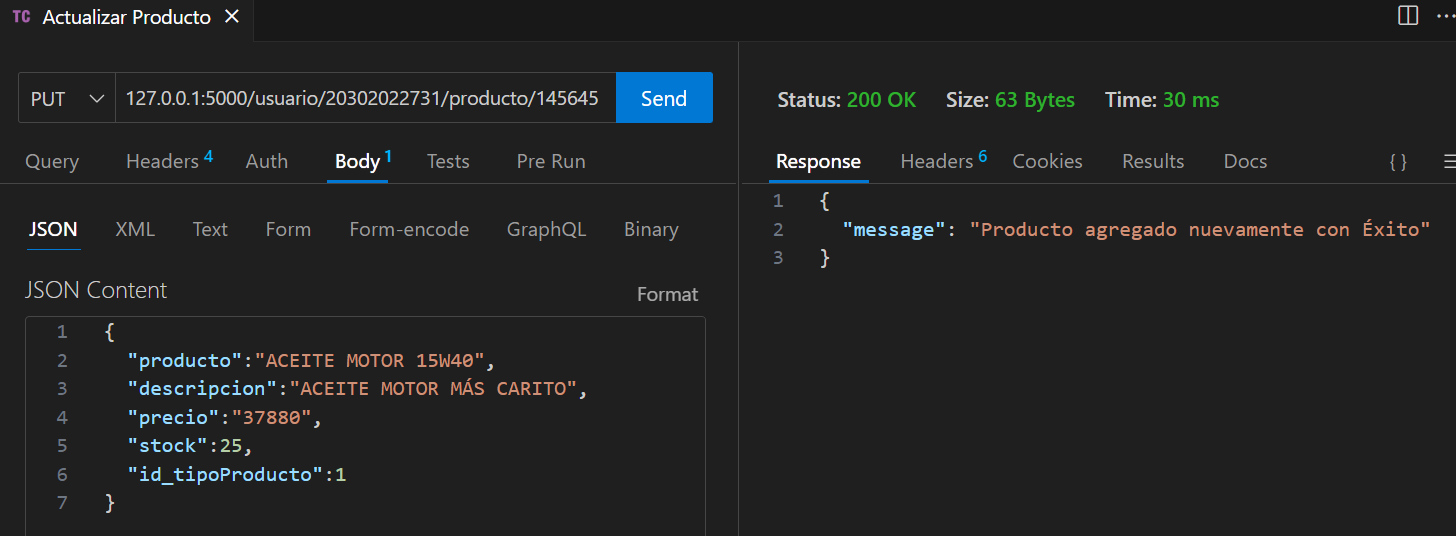
Obtener todos los productos por usuario.-

'/usuario/<id\_usuario>/productos', methods=['GET']



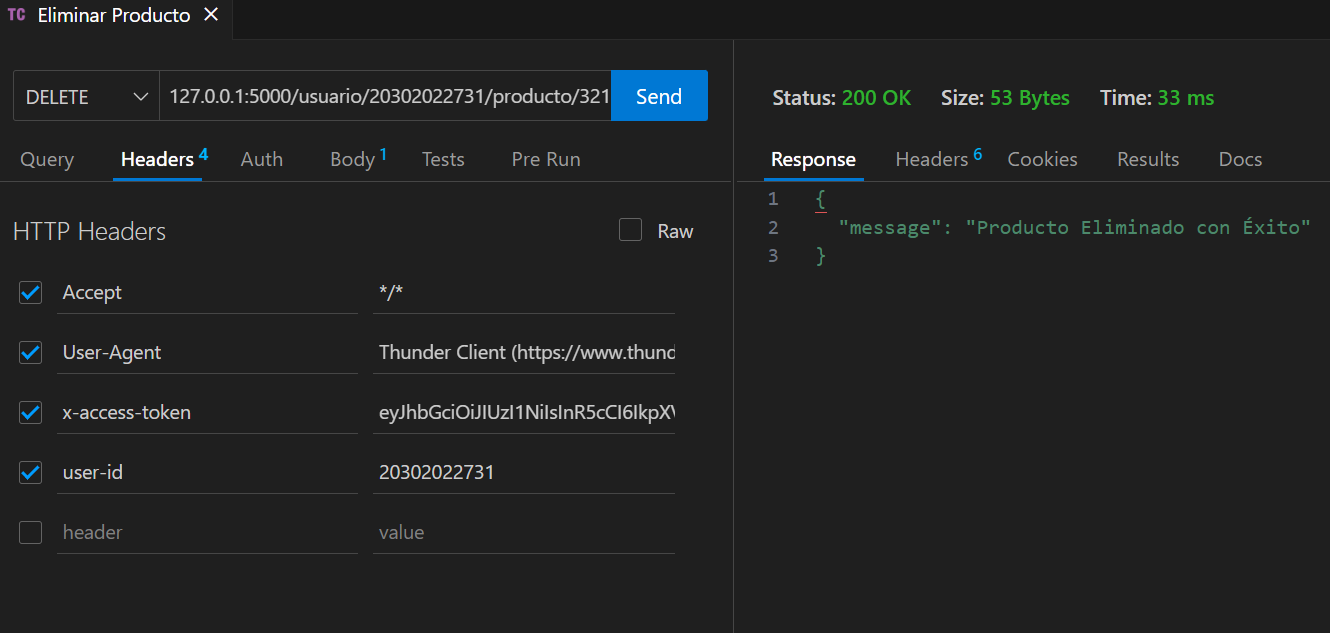
Actualizar producto.-

'/usuario/<id\_usuario>/producto/<codigoProducto>', methods=['POST']



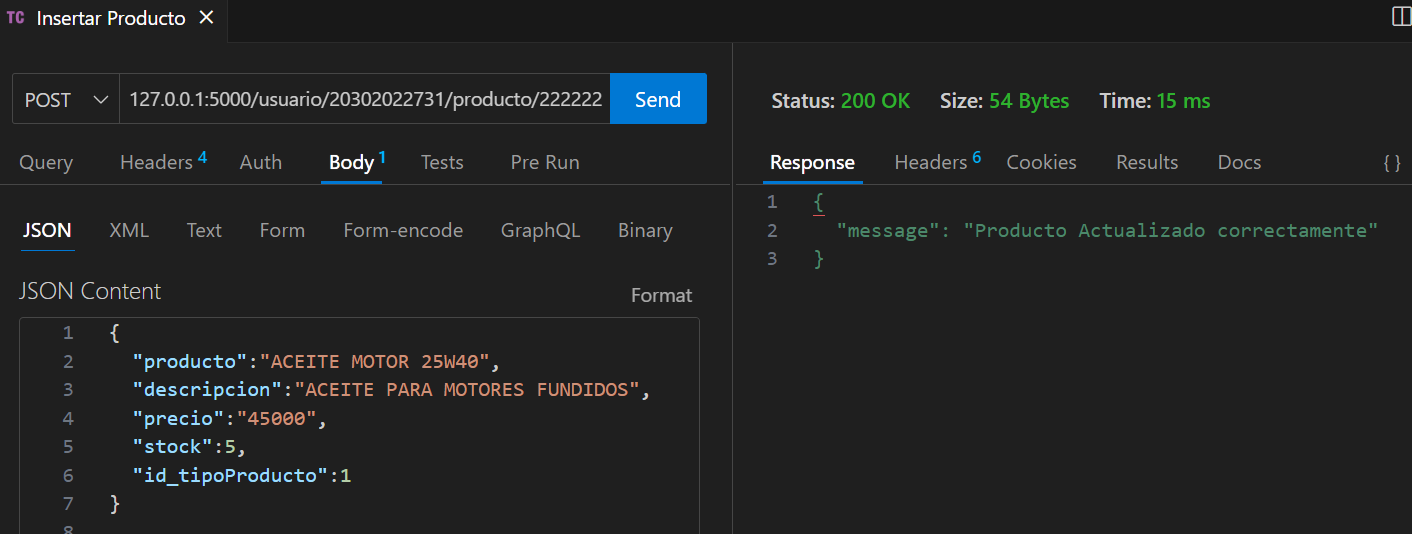
Eliminar Producto.-

'/usuario/<id\_usuario>/producto/<codigoProducto>', methods=['DELETE']



Insertar Producto.-

'/usuario/<id\_usuario>/producto/', methods = ['POST']



# UX/UI

*Dentro de esta sección haremos un análisis de las* ***Leyes UX****.*

**Ley de Hick** : Diseño sencillo e intuitivo, propone las funcionalidades básicas lo cual no añade confusión a la tarea para la cual fue realizada la app ().

**Ley de Fitts** : En cuanto a la accesibilidad de estas funcionalidades, la app propone diferentes rutas para utilizarlas.

**Ley de miller** : De igual manera que la ley de Hick podemos decir que la app contiene secciones claras y concisas que permiten al usuario utilizar toda la funcionalidad.

**Ley de Jakob** : Se muestra similar a otras app de su estilo y provee la misma experiencia de usabilidad.

**Ley de proximidad** : La app se encuentra agrupada en 3 secciones principales “Clientes”, “Productos” y “Facturas”, por otra parte agrupa datos relevantes para el usuario en una sección aparte llamada “Dashboard”.

**Ley de pregnancia** : A primera vista se encuentra el contenido principal de manera clara, los elementos están posicionados para que puedan distinguirse apropiadamente.

**Ley de similitud o semejanza** : La app no usa una paleta de colores amplia, se maneja entre botones de color azul y botones grises para realizar acciones, las advertencias de errores o de exito se muestran con los colores correctos y esperables.

# Mejoras propuestas

Para futuras versiones:

* Se propone restringir el tipo de factura en base a la condición con respecto al IVA del cliente.
* Modificar los gráficos o forma en la cual se muestra el stock de los productos teniendo en cuenta las generalidades de las reglas de negocio.
* Utilizar API’s de terceros para mejorar el ingreso de datos o guardado de los mismos.
* Crear Nota de Crédito para anulación de Facturas.
* Que los productos con Stock 0 no se muestren o no se puedan seleccionar, para la compra.
* En el dashboard en la parte de Movimiento de Stock, al cambiar a la página siguiente te desplaza la misma y te corre de la vista de Mov. de Stock, cuando debería quedar en esa posición.
* En Agregar Cliente, el campo domicilio, debería abrirse en Provincia, Ciudad, Calle, Altura, Piso, Dep. así el sistema puede ser utilizado por usuarios con clientes en varias provincias. Siendo más abarcativos podríamos agregar País, para aquellos que puedan vender al exterior.
* La Provincia, Ciudad y Calle (si agregamos País, también aplicaría) debería obtenerse de una Base de Datos, así evitamos errores de tipeo que distorsionen los datos de nuestra BD y ofrezca resultados de estadísticas poco confiables.
* Se podría implementar el envío de factura electrónica
* En la parte Factura, Método de Pago se podría implementar el pago por medios electrónicos o con Aplicaciones.

# 