Desarrollo de Aplicaciones Distribuidas

Dispositivo con NFC

La idea parte de un supermercado cuyas etiquetas de los productos son compatibles con la tecnología de NFC. El proyecto consiste en el desarrollo de un dispositivo compatible con esta tecnología de manera que, se le podrán dar varias funcionalidades a dicho dispositivo.

Autores: Pérez López Sergio, Dan Gigi Marcel

Índice

1. IDEA (1ª Iteración)

La idea parte de un supermercado compatible con la tecnología *Near Field Communication* (NFC), que permite la comunicación inalámbrica de corto alcance para el intercambio de datos entre dispositivos. El proyecto se basará en el desarrollo de un dispositivo compatible con esta tecnología de manera que, se le podrán dar varias funcionalidades a dicho dispositivo.

Esto será posible debido a que cada etiqueta colocada en la estantería donde se encuentra un tipo de producto tendrá etiquetas especiales para transferir información sobre el producto a través del dispositivo mencionado anteriormente y que nos permitirá comprobar información nutricional y otros datos del producto como, por ejemplo, cuanto de saludable es un producto para determinadas personas. Esto quiere decir que el dispositivo se podrá configurar para las intolerancias alimentarias de una determinada persona.

1.1. Funcionalidades del dispositivo

- El dispositivo presentará un teclado de números, en el cual el usuario podrá marcar una **configuración** propia según sus **intolerancias**.

  Las características disponibles que seleccionar en el dispositivo son:

  - **Característica 1**: Diabético
  - **Característica 2**: Intolerante a la lactosa
  - **Característica 3**: Celíaco
  - **Característica 4**: Alergia al huevo
  - **Característica 5**: Alergia al marisco y al pescado
  - **Característica 6**: Alergia a frutos secos, legumbres y cereales
  - **Característica 7**: Intolerancia a la fructosa
  - **Característica 8**: Intolerante a la histamina
  - Etc.
- Otra funcionalidad beneficiosa para la empresa que implemente el sistema será que podrá obtener un **mapa de calor** con las **zonas más transitadas** del supermercado por los clientes. Aplicable después a estrategias marketing y publicitarias.

1.2. Ejemplo de aplicación

Para entender mejor el funcionamiento del dispositivo, vamos a contar una pequeña historia donde podremos entender la aplicación de este.
Pedro, un anciano de 76 años y con diabetes, acostumbra a ir al supermercado de debajo de casa a comprar la comida que necesita para el día a día. El problema que tiene nuestro protagonista es que no sabe leer, y en varias ocasiones ha tenido más de un susto por haber comprado alimentos con azúcar por equivocación. Además, Pedro, la mayoría de las veces está condicionado a comprar el mismo tipo de alimentos sin azúcar porque al no saber leer no sabe si el producto que desconoce puede o no tomarlo.
En el supermercado que hay debajo de la casa de Pedro se ha implantado un sistema cuya principal característica es el NFC. Cuando Pedro entra de nuevo en el supermercado como de costumbre, se para a hablar con Miguel, gerente de mantenimiento.

Miguel le ha dado un dispositivo a Pedro del tamaño de un móvil de 7 pulgadas y le ha preguntado que, si tenía algún tipo de intolerancia alimentaria, a lo que Pedro le contesta que sí, ya que es diabético desde pequeño. Miguel al escuchar esto, le pide a nuestro protagonista que pulse el primer botón y que cuando quiera saber si un producto es apto para él o no, tan solo tiene que acercar el dispositivo a la etiqueta de la estantería donde se encuentra el producto. Si la luz se pone de color verde significa que puede tomarlo sin ningún problema. De lo contrario, si la luz se pone roja significará que no debería de consumir ese producto porque contiene azúcar.
Pedro entra al supermercado y empieza a comprar con la ayuda del dispositivo. Al pasar por caja para pagar, se da cuenta de que lleva mucha más cantidad de productos de lo que está acostumbrado a comprar. La sorpresa llega cuando el precio por toda la compra no excede mucho más de la cuenta habitual.
El hecho de que la cuenta no se haya visto incrementada mucho ha sido porque, Pedro acostumbraba a comprar productos de marcas muy conocidas y desconocía la existencia de productos similares con precios más reducidos. El dispositivo ha causado que Pedro pueda comprar con mucha más libertad y sin miedo a comprar algo que no pueda tomar.
Por otro lado, Miguel ha ido haciendo lo mismo con los demás clientes, y poco a poco los clientes se han ido habituando al hecho de utilizar dicho dispositivo. Gracias a este dispositivo, el personal del supermercado ha sido capaz de ver que zonas han sido las más visitadas mediante un mapa de calor. Este mapa de calor se va a generar triangulando la posición de dicho dispositivo en todo momento y generando un archivo que indique dichas coordenadas al final del día, de modo que ahora van a ser capaces de anunciar nuevos productos en aquellas zonas que se saben con certeza que van a ser más visitadas que otras zonas y aplicar otras muchas estrategias de marketing útiles para la empresa.

2. BASE DE DATOS (2ª Iteración)

Las tablas de la base de datos realizadas para este segundo entregable y representadas en la Figura 1 del Anexo. dan solución para el almacenamiento de los datos necesario para llevar a cabo el proyecto.

Vamos a dar una visión general de cómo funciona, esto lo vamos a hacer dando una breve explicación de cada una de las tablas y de que datos se van a almacenar en ellas.

- **comercio:** en esta tabla se almacenan los datos que hacen referencia al supermercado en el cual se va a implantar el sistema. Como atributos tenemos:
    - **nombreComercio:** almacena el nombre del supermercado. (Por ejemplo: Mercadona).
    - **CIF:** almacena el CIF de la empresa. (Por ejemplo: A46103834)
    - **teléfono:** almacena el número de teléfono del establecimiento: (Por ejemplo: 963883333)
    - **idComercio:** clave primaria e identificativa de cada entrada de la tabla comercio.
- **redeswifi:** en esta tabla se almacena los datos que corresponden a las redes wifi que capta el dispositivo, necesarias para la triangulación de la ubicación.
    - **SSID:** almacena el nombre de la red wifi (*Service Set Identifier*). (Por ejemplo: WLAN_C990).
    - **PWR:** es el nivel de señal conforme nos acercamos con el dispositivo al punto de acceso de la red wifi. (Por ejemplo: -30).
    - **captureTime:** almacena el valor en segundos del momento en el que se realiza la captura de la señal wifi. (Por ejemplo: 1584921600).
    - **idredesWifi:** clave primaria e identificativa de cada entrada de la tabla redeswifi.
    - **idComercio:** clave ajena referente a la columna que tiene el mismo nombre de la tabla comercio.
- **ubicación:** en esta tabla se almacena la información referente a la ubicación de un cliente en un momento determinado.
    - **horaUbicación:** almacena la hora correspondiente a la ubicación del usuario. (Por ejemplo: 1584921300)
    - **margenError:** almacena el porcentaje de error que podría tener la ubicación almacenada con respecto a la realidad. (Por ejemplo: 0.02%)
    - **nombreZona:** almacena el nombre de la zona donde se sitúa la ubicación tomada. (Por ejemplo: pasillo central)
    - **idUbicacion:** clave primaria e identificativa de cada entrada de la tabla ubicación.
    - **idredesWifi:** clave ajena referente a la columna que tiene el mismo nombre de la tabla redeswifi.
    - **idUsuario:** clave ajena referente a la columna que tiene el mismo nombre de la tabla usuario.

- **intolerancia:** en esta tabla se almacena el nombre de las diferentes intolerancias alimentarias o intolerancia a los alimentos que puede tener una persona. Es decir, hace referencia al nombre de las posibles reacciones adversas del organismo hacia alimentos que no son digeridos, metabolizados o asimilados completa o parcialmente.
    - **nombreIntolerancia:** este campo almacena el nombre de las diferentes intolerancias. (Por ejemplo: intolerancia a la lactosa)
    - **idIntolerancia:** clave primaria e identificativa de cada entrada de la tabla intolerancia.
- **ingrediente:** en esta tabla se almacena el nombre de los diferentes ingredientes por los que está formado un producto alimenticio y a los cuales una persona puede ser intolerante porque contenga algún aditivo que no tolere.
    - **nombreIngrediente:** almacena el nombre del ingrediente. (Por ejemplo: E102 - Tartrazina)
    - **idIngrediente:** clave primaria e identificativa de cada entrada de la tabla ingrediente.
    - **idIntolerancia:** clave ajena referente a la columna que tiene el mismo nombre de la tabla intolerancia.
- **producto:** en esta tabla se almacena la información referente a cada uno de los productos que hay en el supermercado.
    - **nombreProducto:** almacena el nombre del producto. (Por ejemplo: LecheAsturiana1L)
    - **codigoBarras:** almacena la cadena de caracteres que hacen referencia al código basado en la representación de un conjunto de líneas paralelas de distinto grosor y espaciado que en su conjunto contienen una determinada información. (Por ejemplo: 7501086801046)
    - **fabricante:** almacena el nombre de la empresa que produce el producto. (Por ejemplo: Puleva)
    - **teléfono:** almacena el número de teléfono de atención al cliente del fabricante. (Por ejemplo: 957188753)
    - **idProducto:** clave primaria e identificativa de cada entrada de la tabla producto.
- **ingredientesProducto:** en esta tabla se almacena la información referente a los ingredientes que contiene un producto.
    - **idIngrediente:** clave ajena referente a la columna que tiene el mismo nombre de la tabla ingrediente.
    - **idProducto:** clave ajena referente a la columna que tiene el mismo nombre de la tabla producto.
    - **idIngredientesProductos:** clave primaria e identificativa de cada entrada de la tabla ingrediente.

- **usuario:** en esta tabla se almacena el id referente a la tabla que gestiona al usuario.
    - **idUsuario:** clave primaria e identificativa de cada entrada de la tabla usuario.
    - **idComercio:** clave ajena referente a la columna que tiene el mismo nombre de la tabla comercio.
- **intoleraciasUsuario:** en esta tabla se almacenan las referencias a intolerancias que tiene un usuario. Es decir, en esta tabla es donde debemos consultar si queremos saber las intolerancias de un usuario.
    - **idIntolerancia:** clave ajena referente a la columna que tiene el mismo nombre de la tabla intolerancia.
    - **idUsuario:** clave ajena referente a la columna que tiene el mismo nombre de la tabla usuario.
    - **idIntoleranciasUsuario:** clave primaria e identificativa de cada entrada de la tabla intoleranciasUsuario.

- **productosUsuario:** en esta tabla se almacena la información referente a las veces que un producto ha sido escaneado por un cliente.
    - **vecesEscaneado:** almacena el número de veces que un producto se escanea. (Por ejemplo: 4)
    - **idContador:** clave primaria e identificativa de cada entrada de la tabla contador.
    - **idUsuario:** clave ajena referente a la columna que tiene el mismo nombre de la tabla usuario.
    - **idProducto:** clave ajena referente a la columna que tiene el mismo nombre de la tabla producto.
- **intoleranciasIngredientes:** en esta tabla se almacenan las referencias a intolerancias que tiene un ingrediente. Es decir, en esta tabla es donde debemos consultar si queremos saber las intolerancias de un ingrediente.
    - **idIntoleranciasIngrediente:** clave primaria e identificativa de cada entrada de la tabla intoleranciasIngredientes.
    - **idIntolerancia:** clave ajena referente a la columna que tiene el mismo nombre de la tabla intolerancia.
    - **idIngrediente:** clave ajena referente a la columna que tiene el mismo nombre de la tabla ingrediente.

3. Esquema API Rest (3ª Iteración)

En este apartado se va a explicar cada una de las funciones de la API Rest, así como los métodos que sirven de comunicación entre el cliente y el servidor. A continuación, se va a detallar cada URL con sus correspondientes parámetros y definición de los métodos asociados.

1. **GET** http://localhost:8081/api/scan/:idProducto **→(getIntolerances)**

   Parámetros: Para realizar esta petición es necesario como podemos ver en la URL pasarle el parámetro "**:idProducto**". Con este parámetro obtendremos las intolerancias asociadas al producto correspondiente.

   Query: "SELECT idIntolerancia FROM intolerancia NATURAL JOIN ingrediente NATURAL JOIN ingredientesproducto NATURAL JOIN producto WHERE IdProducto = idProducto"

   Respuesta: la respuesta que recibimos del servidor en caso de haberse realizado correctamente la petición es un JSON que contiene todas las intolerancias que están asociadas al producto pasado por parámetro.

   Código: 200 OK (Figura 3)

   Contenido: JSON que contiene todas las intolerancias que están asociadas al producto pasado por parámetro.

```
[ {
  "idIntolerancia" : 1
 }, {
 "idIntolerancia" : 2
  } ]
```

   Código: 401 (Unauthorized)

   Contenido: Anexo de código de error 401. (Figura 2)

2. **PUT** http://localhost:8081/api/scan/put/wifi/values **→ (putWifiScan)**

   Cuerpo: Para realizar esta petición debemos pasarle un JSON que contiene los datos que definen una red wifi para insertarla en la tabla redeswifi de la base de datos.

   El JSON que debe enviarse tiene la siguiente forma:

```
{
  "ssid" : "WLAN_32",
  "power" : -43,
  "timestamp" : 1231231231,
  "idComercio" : 1
}
```


Query: "INSERT INTO redeswifi (SSID, PWR, captureTime, idComercio) VALUES (?,?,?,?)".

Respuesta:

Código: 200 OK. (Figura 4)

Contenido: JSON que contiene la información almacenada.

Código: 401 (Unauthorized).

Contenido: Anexo de código de error 401. (Figura 2)

3. PUT http://localhost:8081/api/scan/put/produs/values → **(putAfterScan)**

Cuerpo: Para realizar esta petición debemos pasarle un JSON que contiene los datos referentes al producto que el usuario acaba de escanear con el dispositivo para insertarlos en la tabla productosUsuario de la base de datos.

El JSON que debe enviarse tiene la siguiente forma:


```
{
 "idProducto": 2,
 "idUsuario": 3
}
```


Query: " INSERT INTO productosUsuario (idProducto, idUsuario) VALUES (?,?)".

Respuesta:

Código: 200 OK. (Figura 5)

Contenido: JSON que contiene la información almacenada.

Código: 401 (Unauthorized).

Contenido: Anexo de código de error 401. (Figura 2)

4. PUT http://localhost:8081/api/scan/put/usuario/values → (putUsuario)

Cuerpo: Para realizar esta petición debemos pasarle un JSON que contiene los datos referentes al usuario que se va a introducir en la tabla Usuario de la base de datos. En este caso el usuario solo necesita ser asociado a un comercio para crearse, además del id que la BD le asigne automáticamente.

El JSON que debe enviarse tiene la siguiente forma:

```
{
  "idComercio" : 1"
}
```

Query: " INSERT INTO usuario (idComercio) VALUES (?)".

Respuesta:

Código: 200 OK. (Figura 6)

Contenido: JSON que contiene la información almacenada.

Código: 401 (Unauthorized).

Contenido: Anexo de código de error 401. (Figura 2)

5. PUT http://localhost:8081/api/scan/put/usuint/values → (putIntoleranciasUsuario)

Cuerpo: Para realizar esta petición debemos pasarle un JSON que contiene los datos referentes a las intolerancias de un usuario. Estas intolerancias son las que el usuario introduce con el dispositivo una vez entra en el supermercado.

El JSON que debe enviarse tiene la siguiente forma:

{
"idIntolerancia": 1,
"idUsuario": 2
}

Query: " INSERT INTO intoleranciasusuario (idIntolerancia, idUsuario) VALUES (?,?) "

Respuesta:

Código: 200 OK. (Figura 7)

Contenido: JSON que contiene la información almacenada.

Código: 401 (Unauthorized).

Contenido: Anexo de código de error 401. (Figura 2)

4. Bot de Telegram (3ª Iteración)

En este apartado de la documentación vamos a hablar y explicar todo el funcionamiento de la parte que corresponde al Bot de Telegram (**adminNFC**).

El Bot ha sido creado con la finalidad de facilitar el trabajo al administrador del sistema. La pregunta es, ¿Cómo un Bot de Telegram puede facilitar el trabajo de un administrador?, la respuesta es sencilla: automatizando el proceso de inserción de datos en la base de datos. Nuestro Bot "adminNFC" ha sido creado para poder insertar todos los datos que son transparentes al usuario final. Entiéndase transparente como aquellos datos que son necesarios en la base de datos para que sistema funcione.

Por ejemplo, cuando un usuario entre al supermercado y escanea un producto, las intolerancias asociadas a dicho producto, así como la información del mismo deben de estar almacenadas en la base de datos. Es decir, se han tenido que introducir previamente para poder ser reconocidas. Pues este tipo de información es la que el administrador a través del Bot puede introducir en la Base de datos.

Para entender bien el funcionamiento del código correspondiente vamos a explicar a continuación una serie de historias de usuario que nos ayudarán a entender cómo funciona el Bot.

Hay que tener en cuenta que la principal lógica del sistema es que se van a introducir datos uno a uno en una tabla, lo que significa que hay una secuencia de comando que hay que seguir para introducir lo datos y por lo tanto una serie de

comandos anidados. Es decir, como bien se ha dicho la principal lógica es una anidación de comandos (i**f y else anidados**) así como el almacenamiento de variables temporales (**map**).

A continuación, se va a explicar cada una de las tablas en las que se puede insertar (**/insertar**) datos desde el Bot así como la secuencia a seguir para introducir los datos. Esto se va a realizar con unas historias de usuario a modo de ejemplo.

TABLA INGREDIENTE:

Historia de usuario: (Figura 8)

admin: /insertar
bot: ¿En qué tabla quiere usted insertar?
admin: ingrediente
bot: ¿Qué dato quiere introducir en nombreIngrediente?
admin: Sacarosa
bot: Se ha añadido la entrada correctamente

Función asociada → **"**void tablaIngrediente(Update handler)"

Query asociada:

- **"**INSERT INTO ingrediente (nombreIngrediente) VALUES (?)"

TABLA INTOLERANCIASINGREDIENTE:

Historia de usuario: (Figura 9)

admin: /insertar
bot: ¿En qué tabla quiere usted insertar?
admin: intoleranciasIngrediente
bot: ¿Qué dato quiere introducir en idIngrediente?
bot: se muestra un listado con los ingredientes y sus id que hay en la base de datos
admin: 2
bot: ¿Qué dato quiere introducir en idIntolerancia?
bot: se muestra un listado con las intolerancias y sus id que hay en la base de datos
admin: 1
bot: Se ha añadido la entrada correctamente

Función asociada → **"**void intoleranciasIngrediente(Update handler)"

Query asociada:

- "SELECT idIngrediente, nombreIngrediente FROM ingrediente".
- "SELECT idIngrediente, nombreIngrediente FROM ingrediente".
- **"**INSERT INTO ingrediente (nombreIngrediente) VALUES (?)".

TABLA INGREDIENTESPRODUCTO:

Historia de usuario: (Figura 10)

admin: /INSERTAR
bot: ¿En qué tabla quiere usted insertar?
admin: ingredientesproducto
bot: ¿Qué dato quiere introducir en idIngrediente?
bot: se muestra un listado con los ingredientes y sus id que hay en la base de datos
admin: 1
bot: ¿Qué dato quiere introducir en idProducto?
bot: se muestra un listado con los productos y sus id que hay en la base de datos
admin: 3
bot: Se ha añadido la entrada correctamente

Función asociada → **"**void ingredientesProducto(Update handler)"

Query asociada:

- "SELECT idIngrediente, nombreIngrediente FROM ingrediente".
- "SELECT idProducto, nombreProducto FROM producto".
- **"**INSERT INTO ingredientesproducto (idIngrediente, IdProducto) VALUES (?,?)".

TABLA INTOLERANCIA:

Historia de usuario: (Figura 11)

admin: /insertar
bot: ¿En qué tabla quiere usted insertar?
admin: intolerancia
bot: ¿Qué dato quiere introducir en nombreIntolerancia?
admin: Intolerancia a la lactosa
bot: Se ha añadido la entrada correctamente

Función asociada → **"**void tablaIntolerancia(Update handler)"

Query asociada:

- **"**INSERT INTO intolerancia (nombreIntolerancia) VALUES (?)".

TABLA PRODUCTO:

Historia de usuario: (Figura 12)

admin: /insertar
bot: ¿En qué tabla quiere usted insertar?
admin: producto
bot: ¿Qué dato quiere introducir en nombreProducto?
admin: Natilla con galletas
bot: ¿Qué dato quiere introducir en codigoBarras?
admin: 123456789123
bot: ¿Qué dato quiere introducir en fabricante?
admin: Hacendado
bot: ¿Qué dato quiere introducir en teléfono?
admin: 666666666
bot: Se ha añadido la entrada correctamente

Función asociada → **"**void tablaProducto(Update handler)"

Query asociada:

- **"**INSERT INTO producto (nombreProducto, codigoBarras, fabricante, telefono) VALUES (?,?,?,?)".

TABLA COMERCIO:

Historia de usuario: (Figura 13)

admin: /insertar
bot: ¿En qué tabla quiere usted insertar?
admin: comercio
bot: ¿Qué dato quiere introducir en nombreComercio?
admin: Mercadona
bot: ¿Qué dato quiere introducir en teléfono?
admin: 957222222
bot: ¿Qué dato quiere introducir en CIF?
admin: D55555555
bot: Se ha añadido la entrada correctamente

Función asociada → **"**void tablaComercio(Update handler)"

Query asociada:

- "INSERT INTO comercio (nombreComercio, telefono, CIF) VALUES (?,?,?)".

Otra de las funciones que implementa este Bot de Telegram es, poder visualizar información relevante de la base de datos. Por ejemplo, poder consultar los productos que han escaneado los usuarios con ciertas intolerancias es algo interesante en cuanto a estrategias de marketing para el supermercado se refiere. Este tipo de información se le podrá pedir al Bot (adminNFC) comenzado una conversación con el comando **"/info"**. Además del ejemplo comentado se le podrá pedir otro tipo de información que veremos a continuación con unas historias de usuario para entender mejor el funcionamiento.

INFORMACION DE PRODUCTOS ESCANEADOS FILTRADOS POR INTOLERANCIAS:

Historia de usuario: (Figura 14)

admin: /info
bot: A continuación, se indica la información disponible:
bot: 1. Productos escaneados por usuarios con ciertas intolerancias
bot: Selecciona la petición deseada, introduciendo su índice:
admin: 1
bot: ¿De qué intolerancias desea ver los productos escaneados por los usuarios?
bot: Introduzca alguno de los ID's mostrados a contiuación.
admin: 1 2 5
bot: A continuación, se mostrarán la lista de productos

Función asociada → **"**void infoProductosEscaneados(Update handler)"

Query asociada:

- **"**SELECT idProducto,nombreProducto, COUNT(idProducto) FROM productosusuario NATURAL JOIN producto WHERE productosusuario.idProducto in (SELECT idProducto from productosusuario NATURAL JOIN producto \r\n" + " NATURAL JOIN ingredientesproducto NATURAL JOIN ingrediente NATURAL JOIN intoleranciasingrediente NATURAL JOIN intolerancia WHERE idIntolerancia=" + Integer.parseInt(s) +") GROUP BY IdProducto".

COMPROBAR SI UN PRODUCTO ESTÁ EN LA BASE DE DATOS:

Historia de usuario: (Figura 15)

admin: /info
bot: A continuación, se indica la información disponible:
bot: 2. Comprobar si un producto está ya en la base de datos.
admin: 2
bot: Indique el nombre del producto a comprobar
admin: Natilla con galletas
bot: El producto ya está en la base de datos y posee el siguiente id.
bot: El producto no está en la base de datos

Función asociada → **"**void comprobarExistenciaProducto(Update handler)"

Query asociada:

- "SELECT nombreProducto FROM producto WHERE nombreProducto LIKE '%nombre_producto%'".

VER INFORMACIÓN PERTENECIENTE A UN PRODUCTO:

Historia de usuario: (Figura 16)

admin: /info
bot: A continuación, se indica la información disponible:
bot: 3. Visualizar información referente a un producto.
admin: 2
bot: Indique el nombre del producto del cual desea visualizar la información
admin: Natilla con galletas
bot: fabricante: "Hacendado"
bot:teléfono: "666666666"
bot: código de barras: 123456789123

Función asociada → **"**void informacionProducto(Update handler)"

Query asociada:

- "SELECT fabricante, telefono FROM producto WHERE idProducto= 'nombre_producto'".

Por otro lado, ya que desde el dispositivo no tendría sentido poder modificar datos como por ejemplo un producto del supermercado, o un ingrediente o algún tipo de intolerancia, hemos pensado que sería buena idea facilitar el trabajo al administrador del sistema si este tipo de modificaciones se hacen a través del Bot de Telegram (adminNFC). A continuación, se muestran las historias de usuario que simulan la conversación del administrador cuando se usa el comando **"/modificar".**

MODIFICAR INFORMACIÓN PERTENECIENTE A UN PRODUCTO:

Historia de usuario: (Figura 17)

admin: /modificar
bot: ¿Qué tabla desea modificar?
bot: a. Producto
admin: a
bot: Seleccione el id del producto a modificar (Se mostrarán lista de productos)
admin: 3
bot: Producto 3 contiene los siguientes datos (se mostrarán los datos de ese producto)
bot: Introduzca los datos del producto modificado separándolos por comas
admin: Leche hacendado,123456789123,El pozo,957484848
bot: La entrada se ha modificado correctamente

Función asociada → **"**void modificarProducto(Update handler)"

Query asociada:

- "SELECT idProducto, nombreProducto FROM producto".
- "SELECT * FROM producto WHERE idProducto= valor1"
- "UPDATE producto SET nombreProducto=valor1, codigoBarras=valor2, fabricante=valor3, telefono=valor4 WHERE idProducto=valor5"

MODIFICAR INFORMACIÓN PERTENECIENTE A UN INGREDIENTE:

Historia de usuario: (Figura 18)

admin: /modificar
bot: ¿Qué tabla desea modificar?
bot: b. Ingrediente
admin: b
bot: Seleccione el id del ingrediente a modificar (Se mostrarán lista de ingredientes)
admin: 2
bot: Ingrediente 2 contiene los siguientes datos (se mostrarán los datos de ese ingrediente)
bot: Introduzca los datos del ingrediente modificado separándolos por comas
admin: calcio
bot: la entrada se ha modificado correctamente

Función asociada → **"**void modificarIngrediente(Update handler)"

Query asociada:

- "SELECT idIngrediente, nombreIngrediente FROM Ingrediente".
- "UPDATE ingrediente SET nombreIngrediente= valor1 WHERE IdIngrediente=valor2"

MODIFICAR INFORMACIÓN PERTENECIENTE A UNA INTOLERANCIA:

Historia de usuario: (Figura 19)

admin: /modificar
bot: ¿Qué tabla desea modificar?
bot: c. intolerancia
admin: c
bot: Seleccione el id de la intolerancia a modificar (Se mostrarán lista de intolerancias)
admin: 4
bot: Intolerancia 4 contiene los siguientes datos (se mostrarán los datos de esa intolerancia)
bot: Introduzca los datos de la intolerancia modificada separándolos por comas
admin: Intolerancia a la sacarosa
bot: la entrada se ha modificado correctamente

Función asociada → **"**void modificarIntolerancia(Update handler)"

Query asociada:

- "SELECT idIntolerancia, nombreIntolerancia FROM Intolerancia".
- "UPDATE intolerancia SET nombreIntolerancia= valor1 WHERE IdIntolerancia=valor2"

Por último, el Bot de Telegram puede mediante el comando **"/eliminar"** eliminar una entrada referente a alguna tabla de la Base de Datos. Consideramos que los únicos datos que podría ser útil eliminar son los que hacen referencia a los productos de los supermercados. Esto es porque es posible que un supermercado deje de comercializar cierto producto y sea necesario eliminarlo de la Base de Datos. De nuevo, para facilitar el trabajo al administrador del sistema esta función se puede realizar desde el Bot de Telegram(adminNFC). A continuación, se muestra la historia de usuario y la información que hace referencia a esta funcionalidad.

ELIMINAR UN PRODUCTO:

Historia de usuario: (Figura 20)

admin: /eliminar
bot: ¿De qué tabla desea eliminar?
bot: a. producto
admin: a
bot: Seleccione el id del producto a eliminar (Se mostrarán lista de productos)
admin: 4
bot: la entrada se ha eliminado correctamente

Función asociada → **"**void eliminarProducto(Update handler)"

Query asociada:

- "SELECT idProducto, nombreProducto FROM producto".
- "DELETE FROM Producto WHERE idProducto=valor1"

5. MQTT

Este proyecto contiene una serie de funcionalidades que la mayoría han sido implementadas mediante la API Rest definida en los apartados anteriores. Sin embargo, hay alguna funcionalidad que no se ha implementado y es el envío por parte del cliente al servidor de los datos referentes a las redes Wifi.

CANAL PARA LAS REDES WIFI

Nombre del canal: TOPIC_WIFI

Propósito del canal: Este canal ha sido creado para que los clientes puedan mandar a través de él, al bróker, los datos que identifican a las redes Wifi. Estos datos se envían para que posteriormente se haga un procesamiento de estos, y mediante una triangulación de los distintos accesos wifi se puede dar una ubicación mas o menos exacta de la parte del supermercado en la que se encuentra el cliente.

Contenido del mensaje: El contenido del mensaje que el cliente enviará al bróker es un JSON como el que aparece a continuación**:**

```
{
    "ssid": "WLAN_32",
    "power": -43,
    "timestamp": 1231231231,
    "idComercio": 1
}
```

6. Conexión API Rest y MQTT con ESP32

En esta parte vamos a detallar las funciones que hacen posible que nuestro dispositivo ESP32 pueda realizar peticiones REST. Es decir, mediante las funciones que se detallan a continuación nuestro dispositivo es capaz de conectarse con la API Rest y realizar las peticiones que hacen posible la comunicación entre el servidor que en este caso está en local y nuestro dispositivo.
A continuación, como bien se ha dicho se van a detallar las funciones que hacen posible esa comunicación, se detallará el nombre de ellas, el funcionamiento y con qué función de la API Rest hacen referencia.

Esta parte del proyecto se ha realizado en otro entorno de desarrollo, en concreto en **Atom**. Dentro de este entorno se ha instalado un plugin llamado **PlatformIO.** Las funciones que se detallan a continuación pueden contener a otras para la simplificación del código, para su compresión se detallará la jerarquía de estas.

FUNCIONES

1.- → sendPutNuevoUsuario().

Definición: esta función es la función que se llama cuando un usuario coge el dipositivo. Es la encargada de que se cree un nuevo usuario en la base de datos, posteriormente es la que se encarga de mostrarle al usuario todas las intolerancias para que él seleccione las que el padece, y posteriormente guarde las seleccionadas en la base de datos asociadas a ese usuario recién creado. Todo esto se realiza a través de diferentes funciones que se detallan a continuación.

1.1.- → sendPutUsuario().

Definición: Esta función es la encargada de introducir un nuevo usuario en la base de datos.
Función API Rest asociada: putUsuario().
Petición http asociada: "/api/scan/put/usuario/values"

1.2.- → sendGetAllIntolerances().

Definición: Esta función es la encargada de rescatar todas las intolerancias disponibles para que el usuario elija las que padece.
Función API Rest asociada: getIntolerancesAll ().
Petición http asociada: "/api/scan/get/intolerances"

1.3.- → sendPutIntoleranciasUsuario().

Definición: Esta función es la encargada de enviar a la base de datos las intolerancias que el usuario padece, después de haberlas seleccionado.
Función API Rest asociada: putIntoleranciasUsuario().
Petición http asociada: "/api/scan/put/usuint/values".

2.- → sendGetIntolerancias(int ProductId).

Definición: esta función es la que se encarga de pedir los datos de cierto producto cuando este es escaneado. Cuando un producto es escaneado se deben comparar las intolerancias del usuario con las del producto, pues esta función se encarga de solicitar las intolerancias de un producto dado su id.

Función API Rest asociada: getIntolerances().

Petición http asociada: "/api/scan/:idProducto".

3.- → sendPutAfterScan(int ProductId).

Definición: esta función es la encargada de una vez el usuario ha escaneado un cierto producto insertar en la base de datos el producto escaneado con el usuario asociado a ese escaneo.

Función API Rest asociada: getIntolerances().

Petición http asociada: "/api/scan/:idProducto".

4.- → sendPutWifiRead().

Definición: esta función se encarga enviar las informaciones correspondientes a las redes wifi captadas para poder realizar la triangulación de la ubicación.

Función API Rest asociada: putWifiScan().

Petición http asociada: "/api/scan/put/wifi/values".

Con respecto a la parte de MQTT como bien hemos explicado anteriormete, en nuestro proyecto estamos usando esta tecnología para enviar datos al servidor sobre las redes wifi, aunque en la API Rest hay una función que puede realizar este mismo proceso, para usar y conocer la parte de MQTT, el envío de las redes wifi se realiza a través de una comunicación con el broker como se ve a continuación. Estos datos serán útiles para poder definir una ubicación aproximada de los usuarios. A continuación, se explica la función que se encarga de enviar los datos al servidor a través de un canal. Antes de profundizar en la función, el dispositivo ESP32 enviará información al servidor si esté a dando el aviso de que quiere recibir los mensajes.

FUNCIONES

1.- → conectarMQTT().

Definición: esta función es la que se encarga de conectar el dispositivo ESP32 al broker. Es por ese motivo por el cual en esta función tenemos que hacer uso de la función **client.connect()**, y pasarle el nombre con el cual el servidor verá a este dispositivo, el bróker y la contraseña. También es necesaria aquí la función **client.subscribe()**, a la cual le tenemos que pasar el nombre del TOPIC al cual queremos que nuestro dispositivo se conecte. Y por último llamar a **client.loop().**

Canal MQTT asociado → TOPIC_WIFI.

2.- → enviarMQTT().

Definición: esta función a diferencia de la anterior se encarga de publicar en el canal del broket el JSON que contiene los parámetros de las redes WIFI. Para publicar en el canal hay que hacer uso de la fusión **client.publish()**, a la cual tenemos que pasarle el mensaje que queremos que publique.

Canal MQTT asociado → TOPIC_WIFI.

Formato JSON:

```
{
  "ssid" : "WLAN_32",
  "power" : -43,
  "timestamp" : 1231231231,
  "idComercio" : 1
}
```

7. Funcionamiento del dispositivo

En este apartado se va a describir la lógica final del dispositivo. En ella se va a detallar el orden en el cual se va a ejecutar las tareas en el dispositivo para que todo funcione correctamente. Además, durante la creación de este último entregable nos hemos dado cuenta de cosas como por ejemplo la necesidad de ejecutar dos tareas a la misma vez y gracias a los dos núcleos que tiene el ESP32 ha sido posible ejecutar un código multihilo y que todo se ejecute correctamente en su momento. A continuación, se muestra ese paralelismo que se acaba de mencionar.

Orden de las tareas:

1. El dispositivo se conecta a una red WIFI.

2. Se crea un usuario nuevo.

3. El usuario debe introducir las intolerancias ayudándose del display y de los botones.

4. Se imprime por pantalla "Esperando escaneo".

5. Se ejecuta la función “leerNFC()”, búsqueda activa de un tag NFC, en caso de detectar un escaneo, manda una solicitud de las intolerancias del producto escaneado mediante la API Rest.

6. Se ejecuta la función “getCompatibilidad()”, la cual compara el nivel de incompatibilidad de las intolerancias del cliente con las del usuario y lo muestra por el display.

7. Por último se llama a “sendPutAfterScan()” que inserta en la BD el producto asociado al usuario que lo ha escaneado.

4. El dispositivo se conecta al servidor MQTT correspondiente mediante la función “conectarMQTT()”.

5. El mecanismo MQTT implementado manda las muestras de las redes Wifi necesarias cuando recibe por el canal una demanda con la palabra “enviar” mediante las funciones de “getRSSI()” y “de enviarMQTT()”.

A nivel físico el manejo del dispositivo por parte del usuario es bastante simple puesto que la mayoría de las tareas descritas anteriormente están programadas para que se realicen de manera transparente al usuario.
Cuando el usuario enciende el dispositivo, automáticamente se crea un nuevo usuario y el cliente tan solo tiene que introducir las intolerancias con la ayuda de dos botones y el display LCD. Las intolerancias se seleccionan con un número que representa a cada una de ellas y que se puede ver en la leyenda que muestro a continuación. Mediante el botón de UP el cliente puede moverse para elegir uno u otro número, una vez esté en el número de la intolerancia correspondiente para seleccionarla tiene que pulsar el botón RIGHT y se selecciona, si padece mas de una realiza el paso anterior tantas veces como sea necesario, una vez seleccionadas pulsa el botón RIGHT de nuevo y el dispositivo asociará esas intolerancias a ese usuario y será con las que una vez escaneé un producto calcule el nivel de incompatibilidad.

Leyenda para la selección de intolerancias.

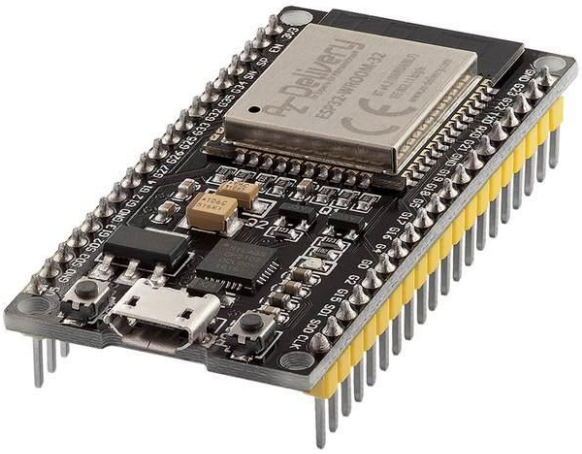
Nombre de intolerancia	Nº de selección
Intolerancia a la lactosa	1
Intolerancia a la histamina	2
Intolerancia al gluten	3
Intolerancia al huevo	4
Intolerancia al cereal	5

8. Hardware

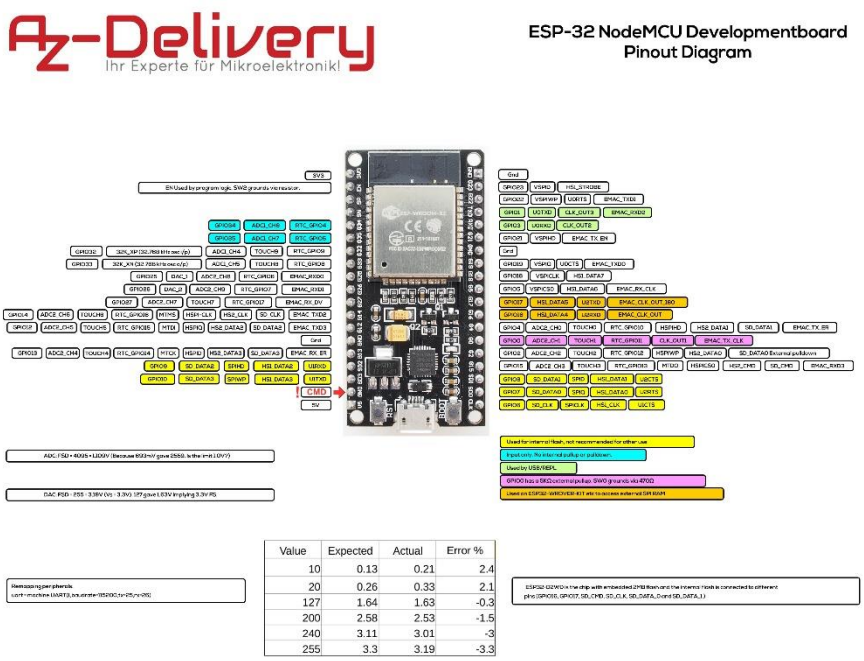
A continuación se van a explicar cada uno de los componentes hardware usado para la realización del proyecto, así como el PINOUT de aquellos de los que sea necesario.

ESP32 NodeMCU Módulo WLAN WiFi Development Board con CP2102

Imagen del componente:



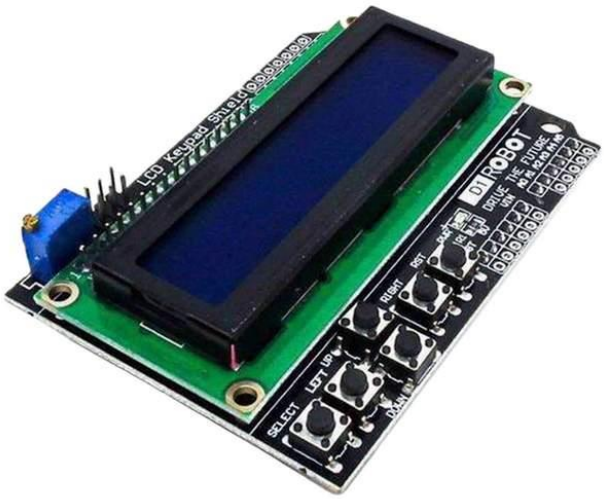
Pinout:



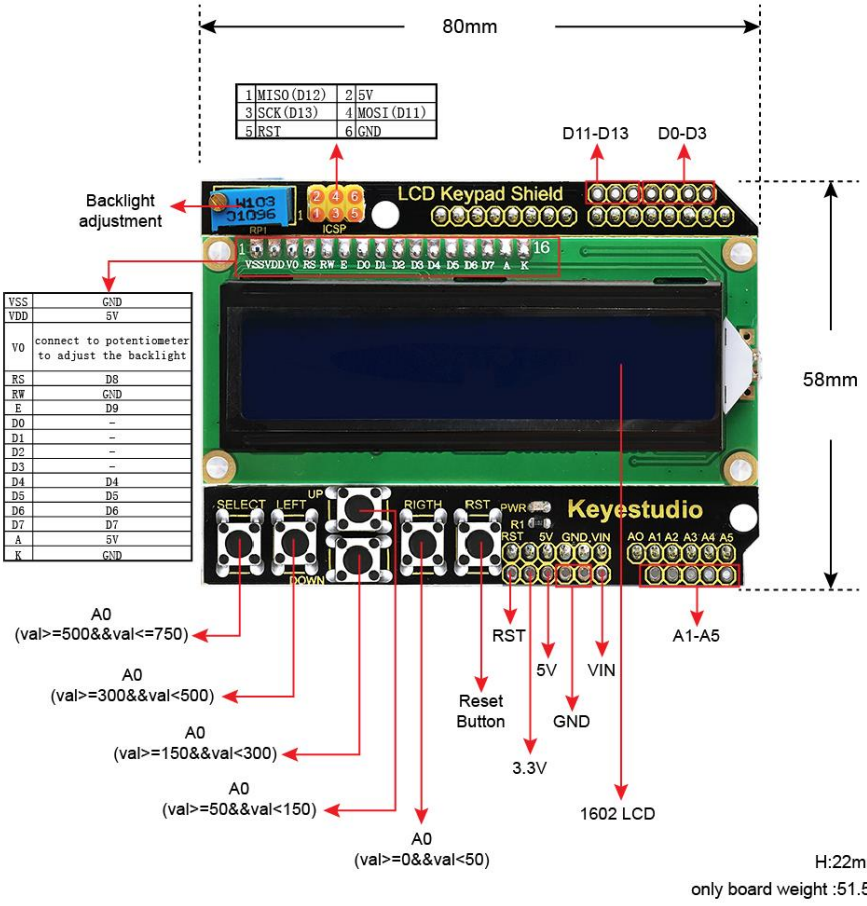
Value	Expected	Actual	Error %
10	0.13	0.21	2.4
20	0.26	0.33	2.1
127	1.64	1.63	-0.3
200	2.58	2.53	-1.5
240	3.11	3.01	-3
255	3.3	3.19	-3.3

LCD1602 Pantalla teclado Escudo HD44780 1602 Módulo con 2x16 caracteres

Imagen del componente:



Pinout:

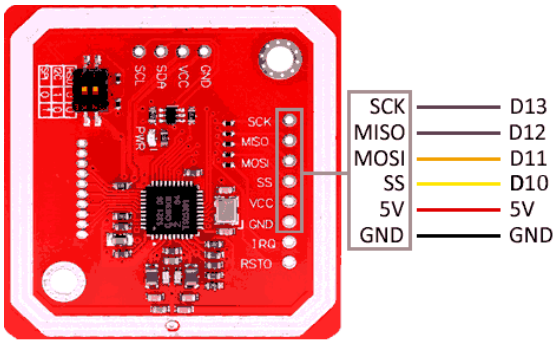


Juego Pn532 NFC MóDulo InaláMbrico RFID V3 Kits de Usuario Lector Modo de Escritura IC S50 Tarjeta PCB Attenna I2C Iic SPI HSU

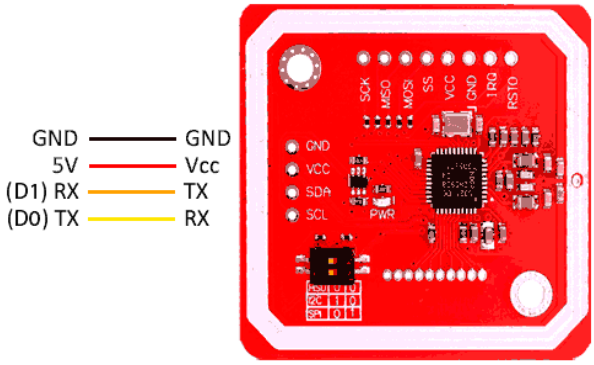
Imagen del componente:

Pinout:

Conexión por SPI (usada en este proyecto)



Conexión por I2C



9. Anexo

9.1. Diagrama UML

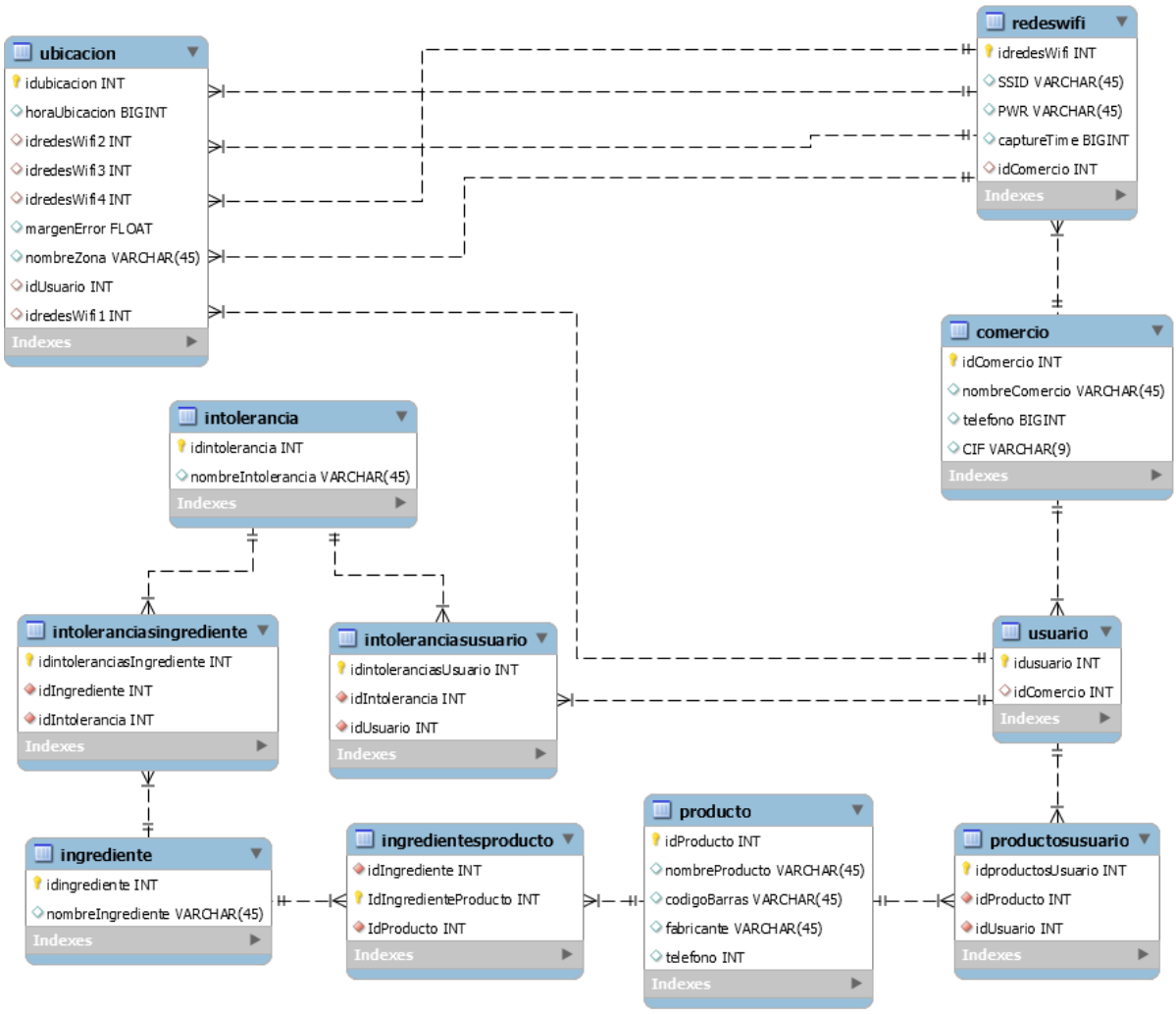


Figura 1

9.2 Ejemplo código error 401


```
{
  "cause": null,
  "stackTrace": [
    {
      "methodName": "handleErrorPacketPayload",
      "fileName": "CommandCodec.java",
      "lineNumber": 130,
      "className": "io.vertx.mysqlclient.impl.codec.CommandCodec",
      "nativeMethod": false
    },
    {
      "methodName": "handleInitPacket",
      "fileName": "ExtendedQueryCommandBaseCodec.java",
      "lineNumber": 27,
      "className": "io.vertx.mysqlclient.impl.codec.ExtendedQueryCommandBaseCodec",
      "nativeMethod": false
    },
    {
      "methodName": "decodePayload",
      "fileName": "QueryCommandBaseCodec.java",
      "lineNumber": 55,
      "className": "io.vertx.mysqlclient.impl.codec.QueryCommandBaseCodec",
      "nativeMethod": false
    },
    },
  ],
  "errorCode": 1452,
  "sqlState": "23000",
  "message": "Cannot add or update a child row: a foreign key constraint fails (`dad`.`usuario`, CONSTRAINT `idComercio7` FOREIGN KEY (`idComercio`) REFERENCES `comercio` (`idComercio`) ON DELETE CASCADE ON UPDATE CASCADE)",
  "localizedMessage": "Cannot add or update a child row: a foreign key constraint fails (`dad`.`usuario`, CONSTRAINT `idComercio7` FOREIGN KEY (`idComercio`) REFERENCES `comercio` (`idComercio`) ON DELETE CASCADE ON UPDATE CASCADE)",
  "suppressed": []
}
```


Figura 2

9.3 Capturas de pantalla POSTMAN

1. GET http://localhost:8081/api/scan/:idProducto

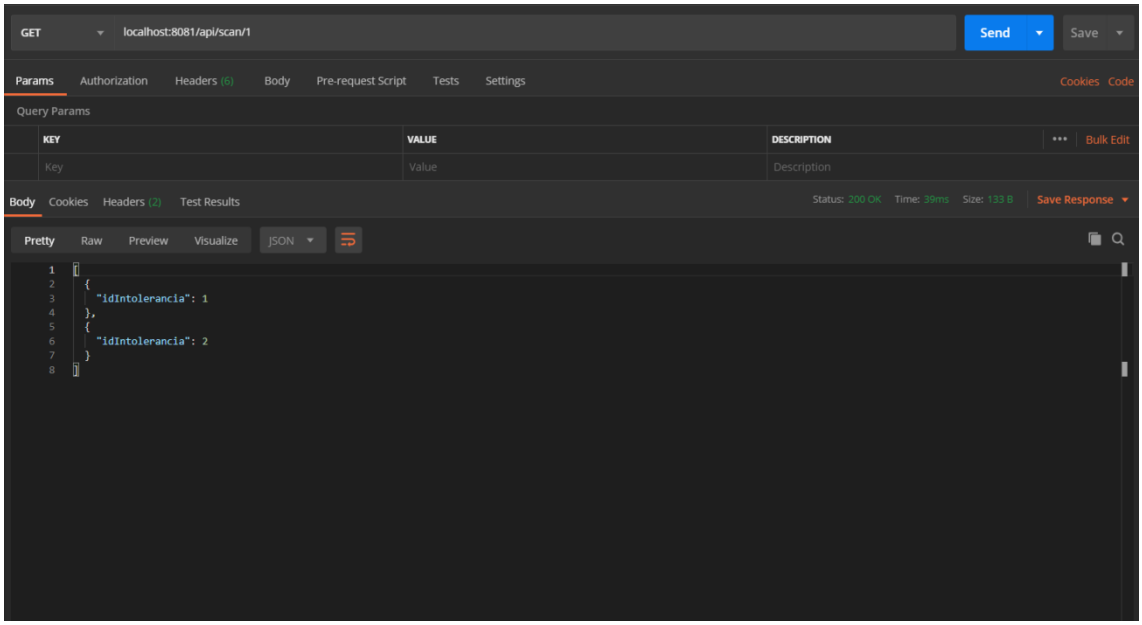


Figura 3

2. PUT http://localhost:8081/api/scan/put/wifi/values

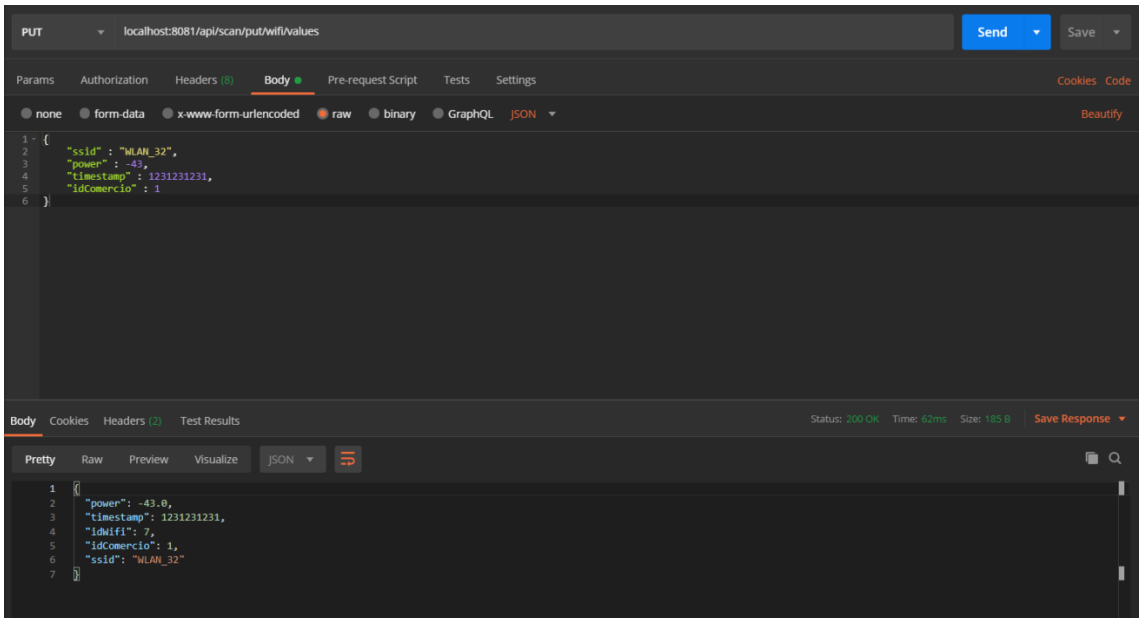


Figura 4

3. PUT http://localhost:8081/api/scan/put/produs/values

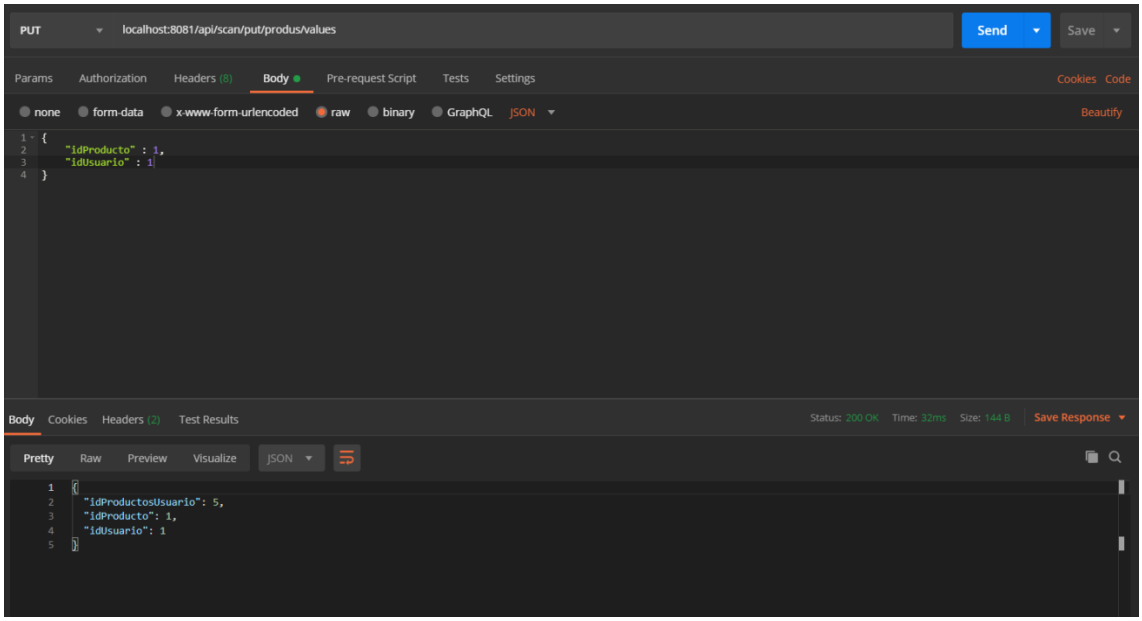


Figura 5

4. PUT http://localhost:8081/api/scan/put/usuario/values

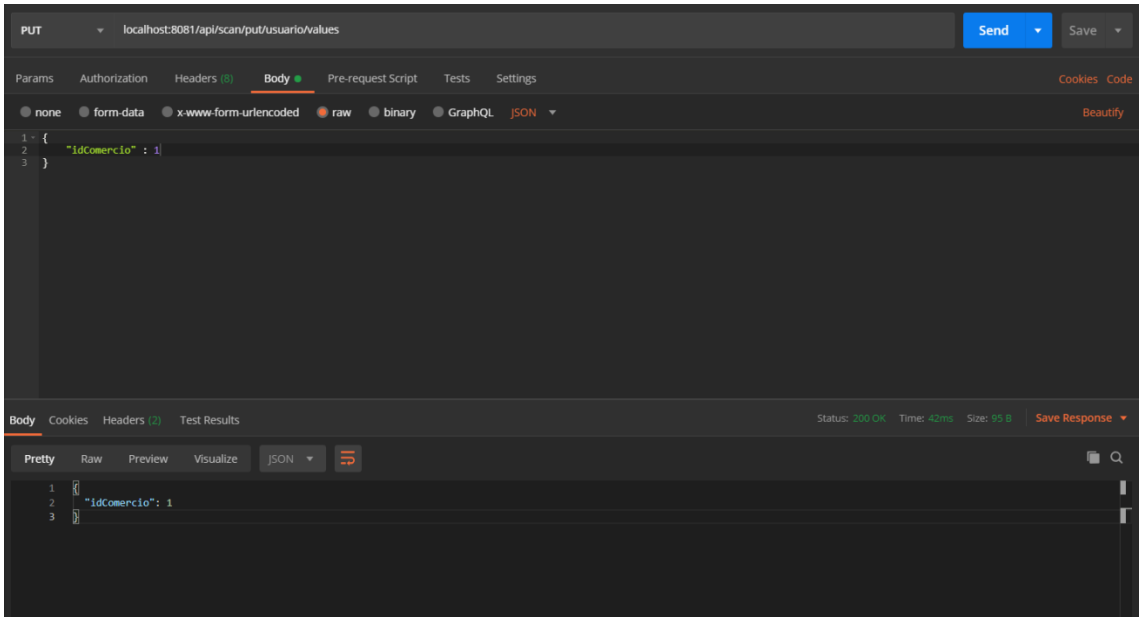


Figura 6

5. PUT http://localhost:8081/api/scan/put/usuint/values

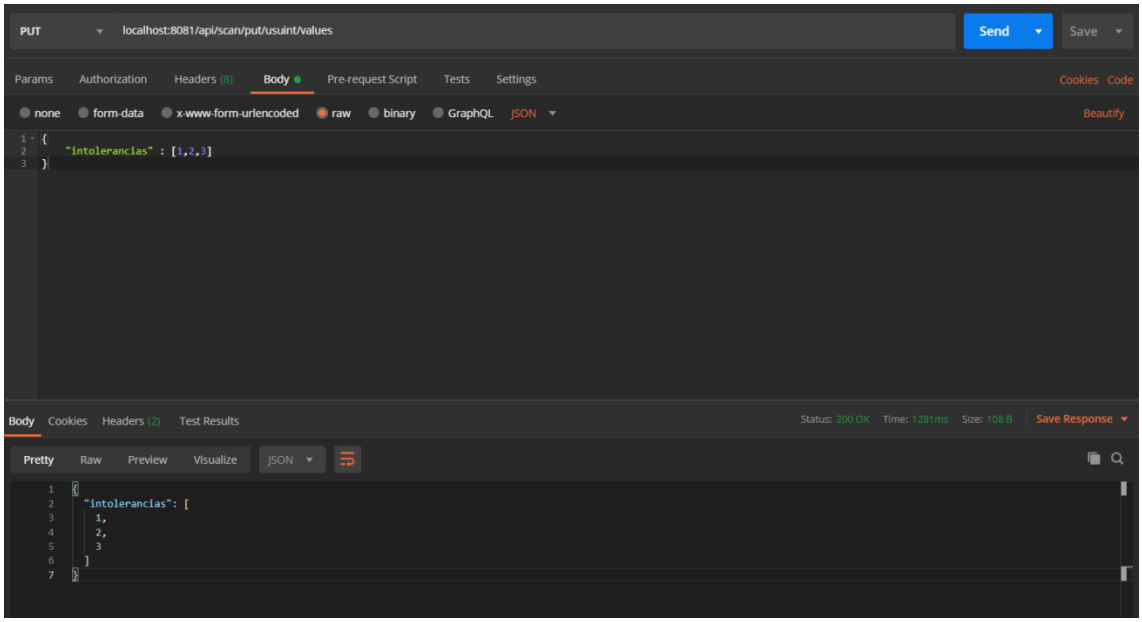


Figura 7

9.4 Capturas de pantalla Telegram (adminNFC)

- INSERTAR EN TABLA INGREDIENTE

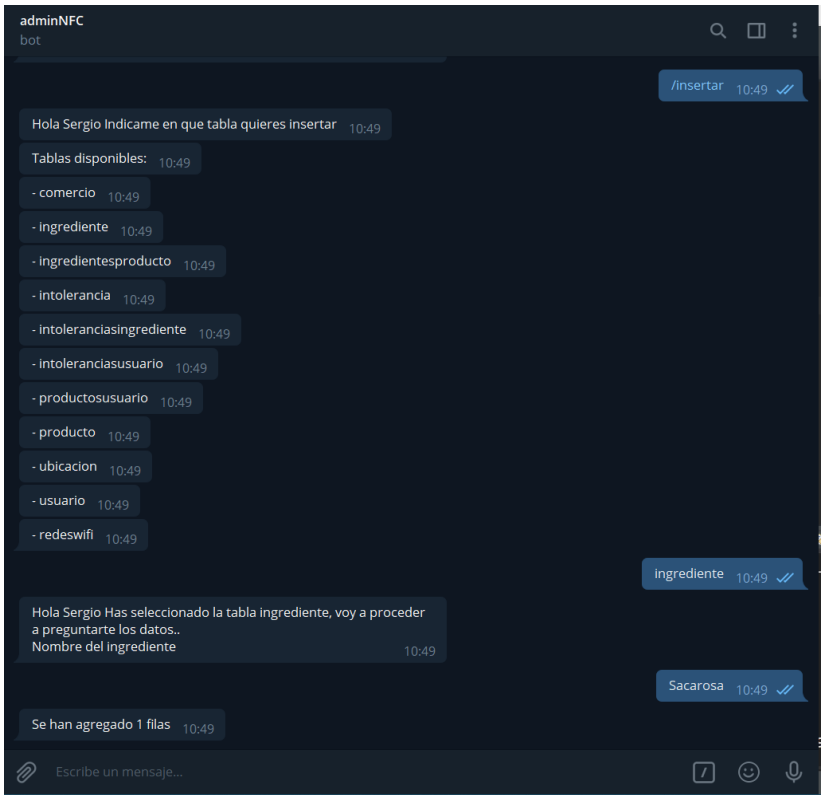


Figura 8

- INSERTAR EN TABLA INTOLERANCIASINGREDIENTE

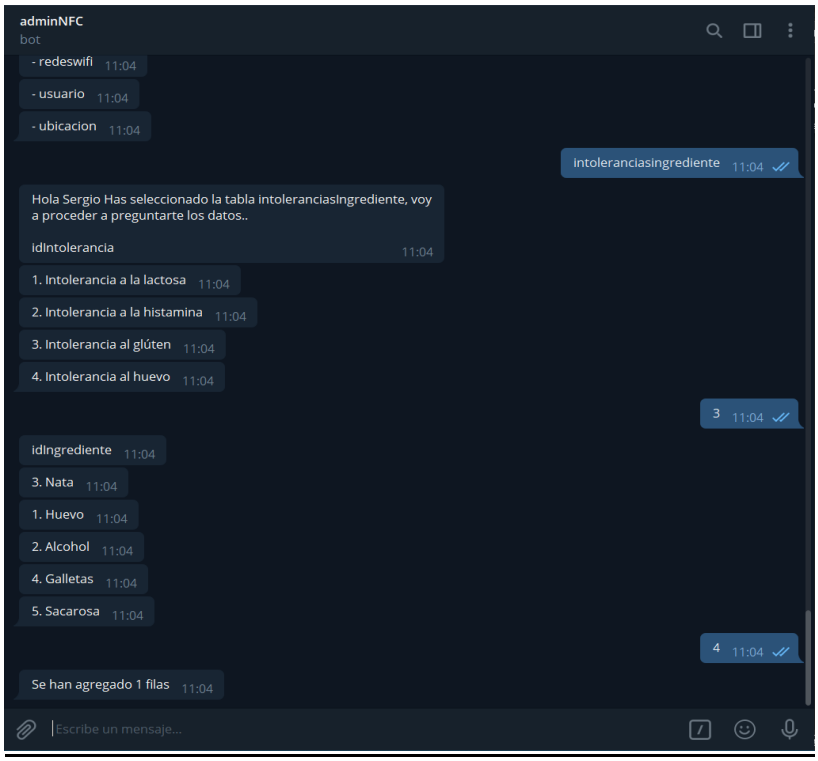


Figura 9

- INSERTAR EN TABLA INGREDIENTESPRODUCTO

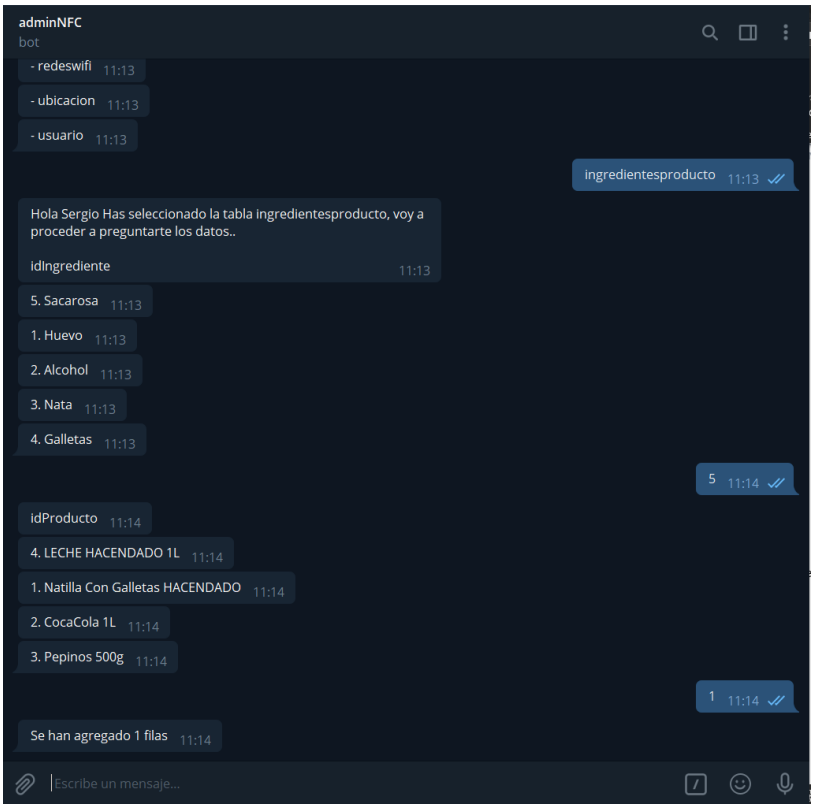


Figura 10

- INSERTAR EN TABLA INTOLERANCIA

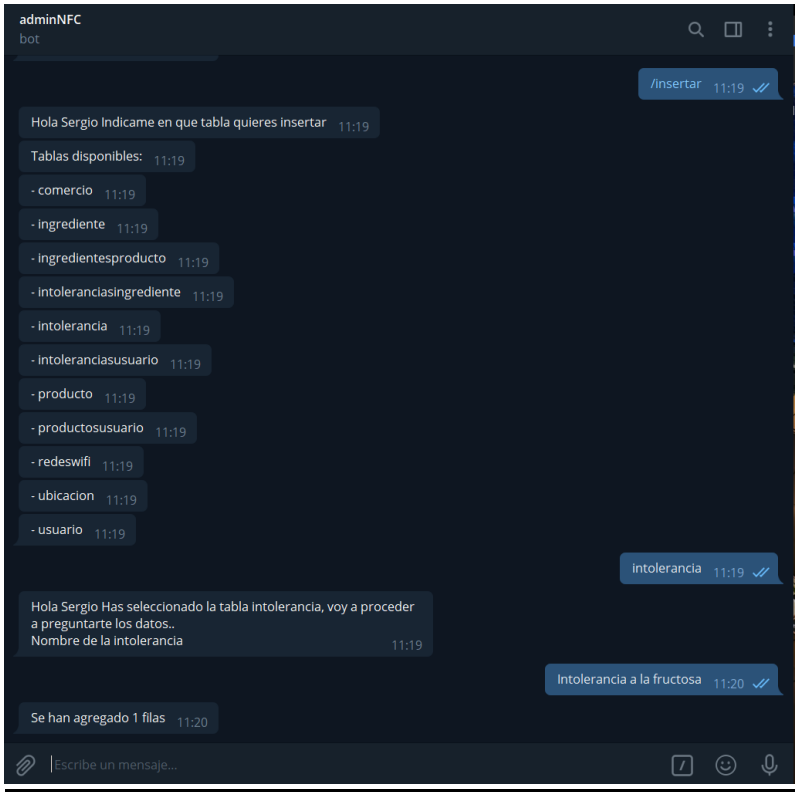


Figura 11

- INSERTAR EN TABLA PRODUCTO

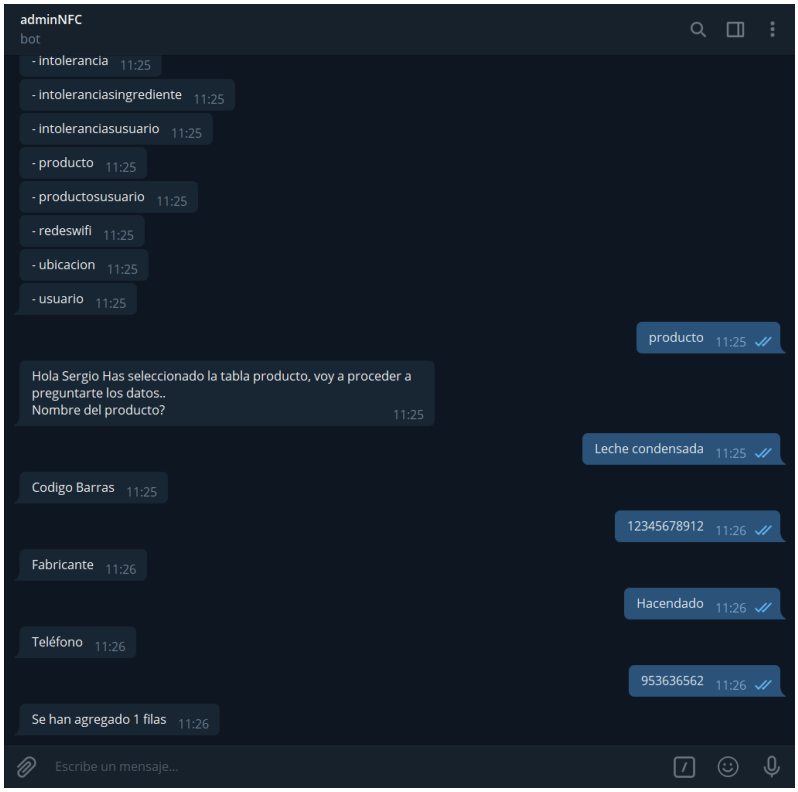


Figura 12

- INSERTAR EN TABLA PRODUCTO

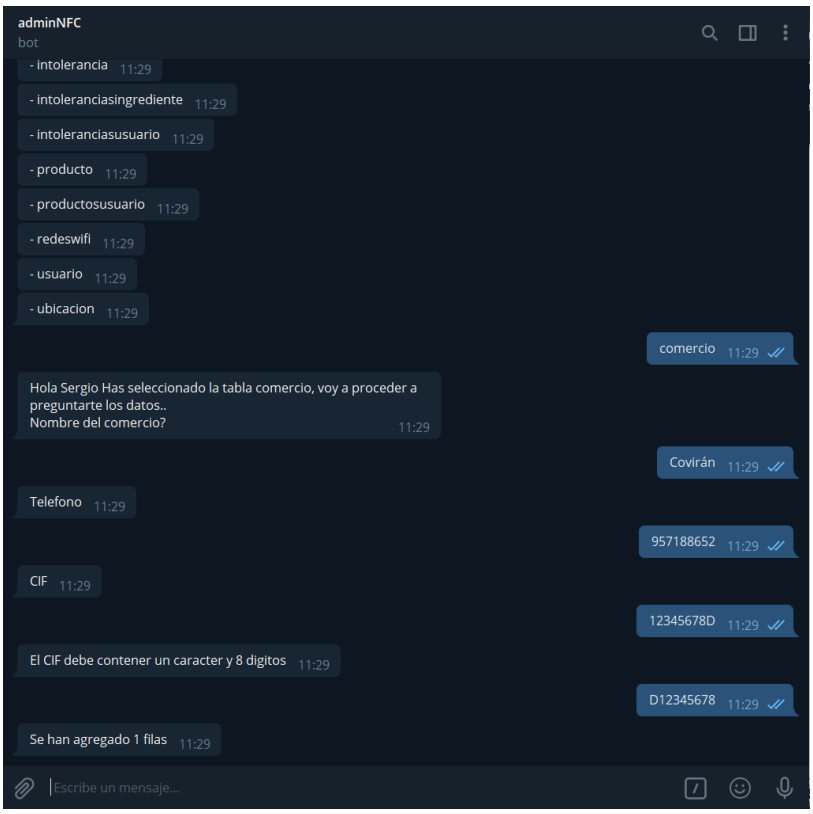


Figura 13

- INFORMACION DE PRODUCTOS ESCANEADOS FILTRADOS POR INTOLERANCIAS.

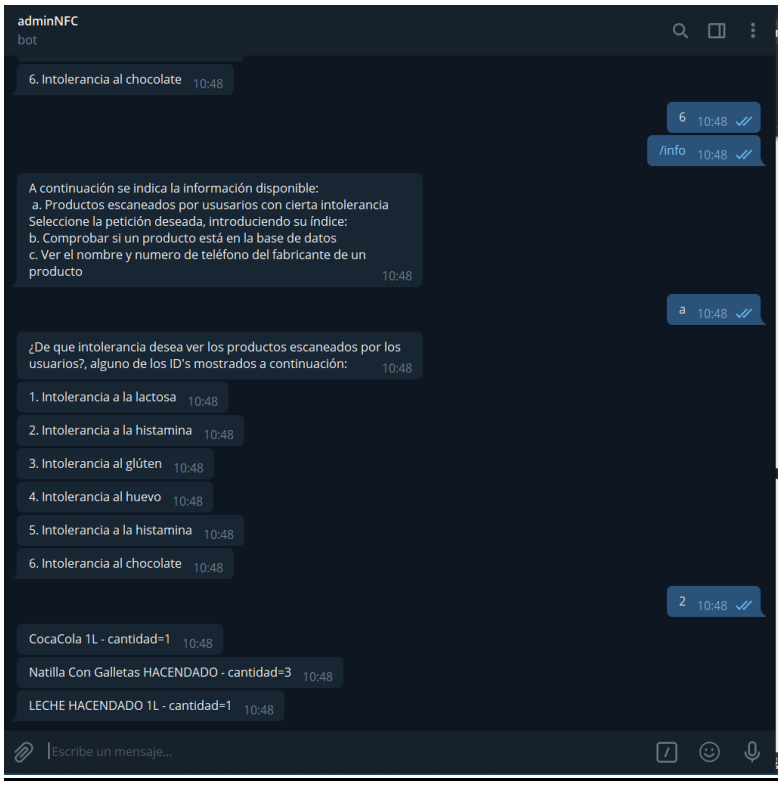


Figura 14

- COMPROBAR SI UN PRODUCTO ESTÁ EN LA BASE DE DATOS.

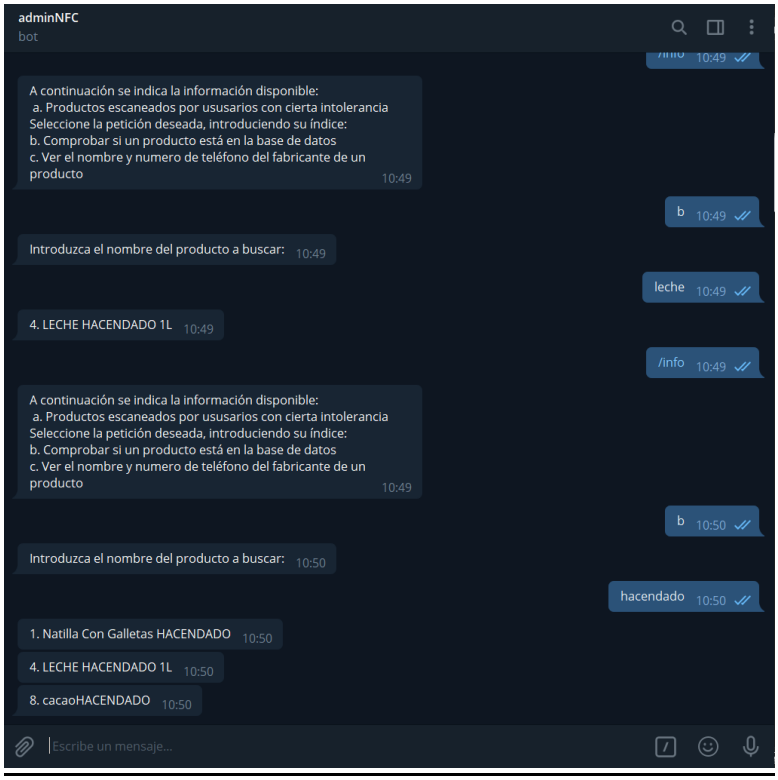


Figura 15

- VER INFORMACIÓN PERTENECIENTE A UN PRODUCTO.

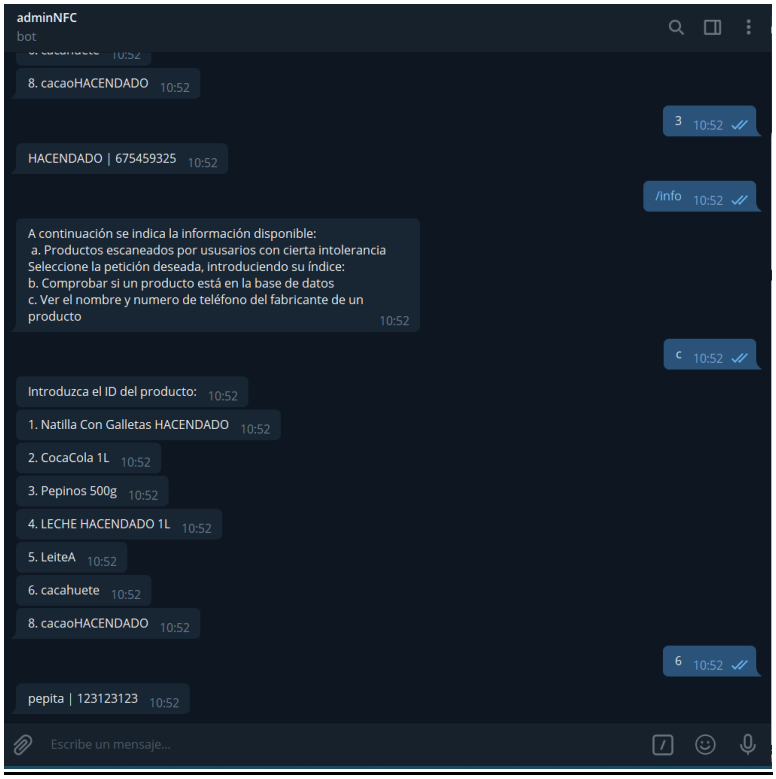


Figura 16

- MODIFICAR INFORMACIÓN PERTENECIENTE A UN PRODUCTO.

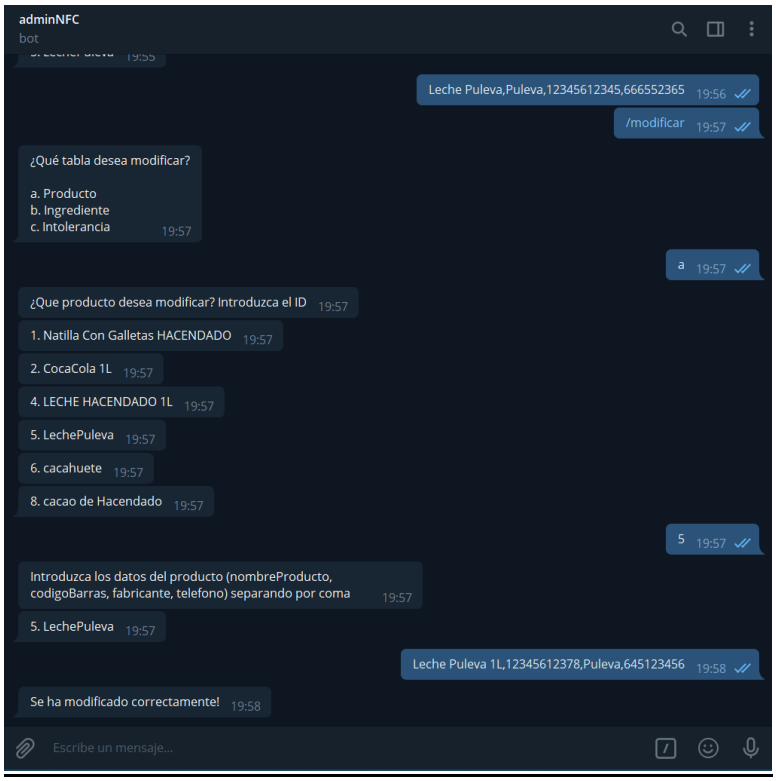


Figura 17

-MODIFICAR INFORMACIÓN PERTENECIENTE A UN INGREDIENTE.

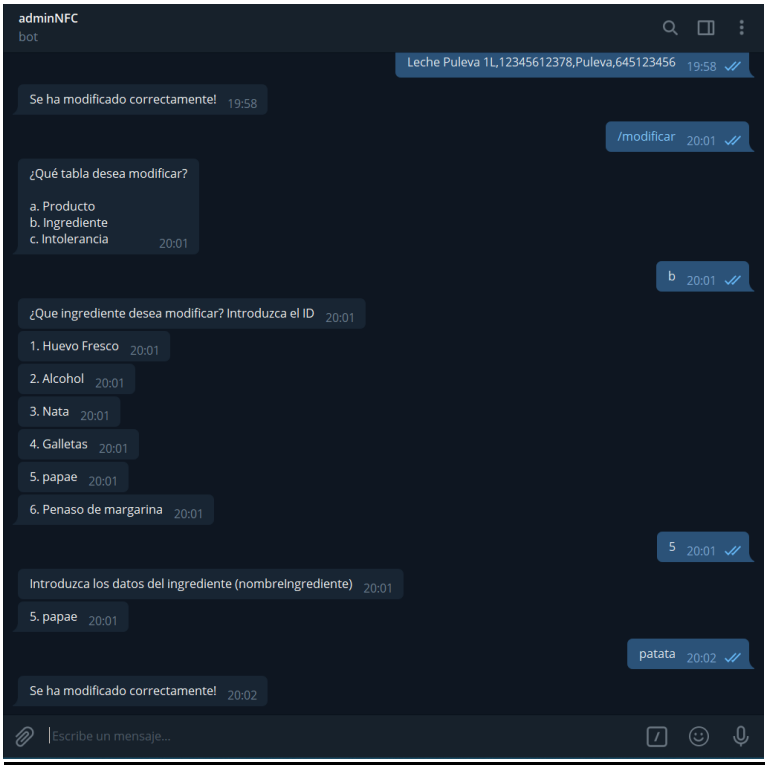


Figura 18

- MODIFICAR INFORMACIÓN PERTENECIENTE A UNA INTOLERANCIA.

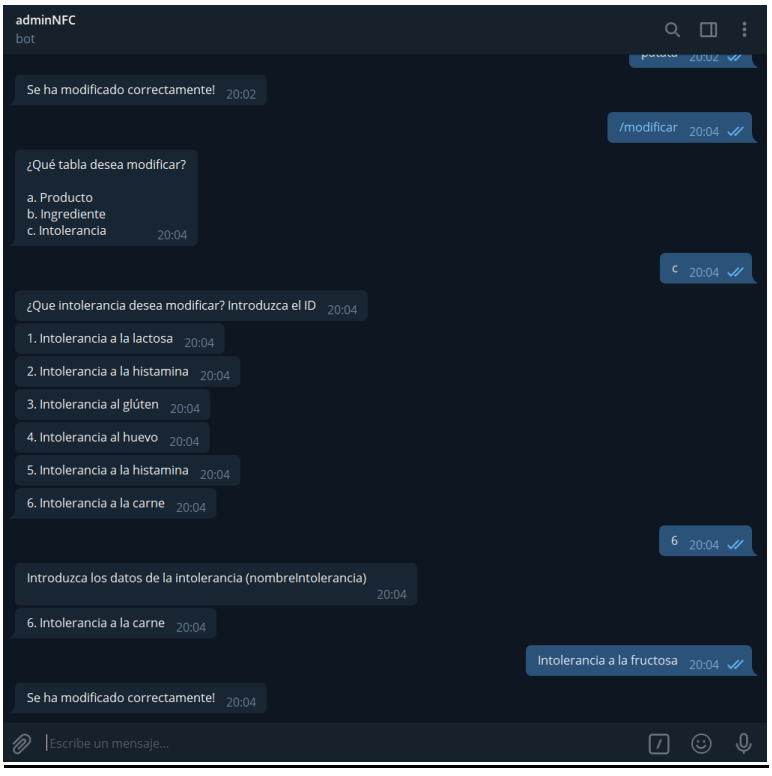


Figura 19

-ELIMINAR UN PRODUCTO.

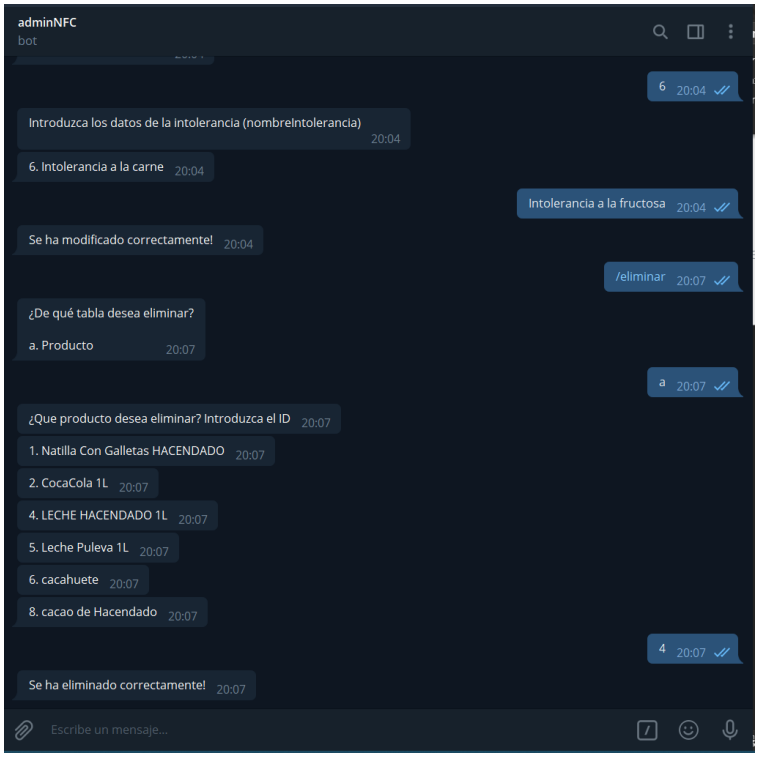


Figura 20