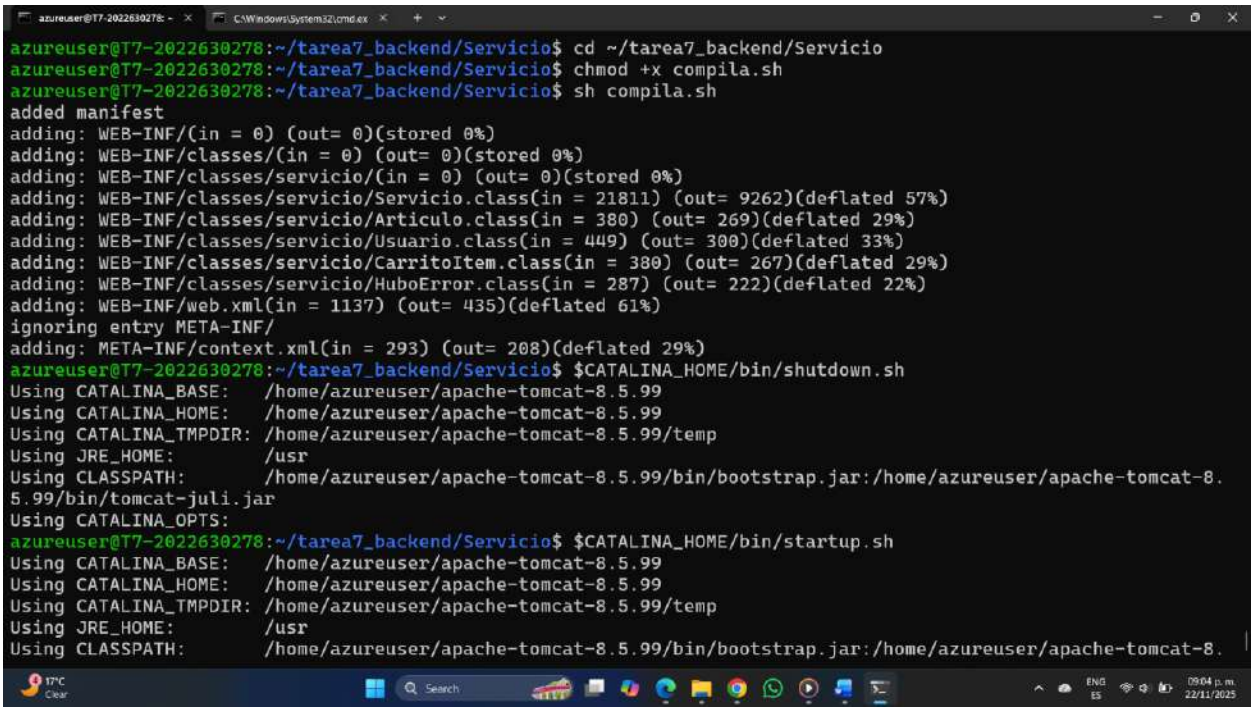
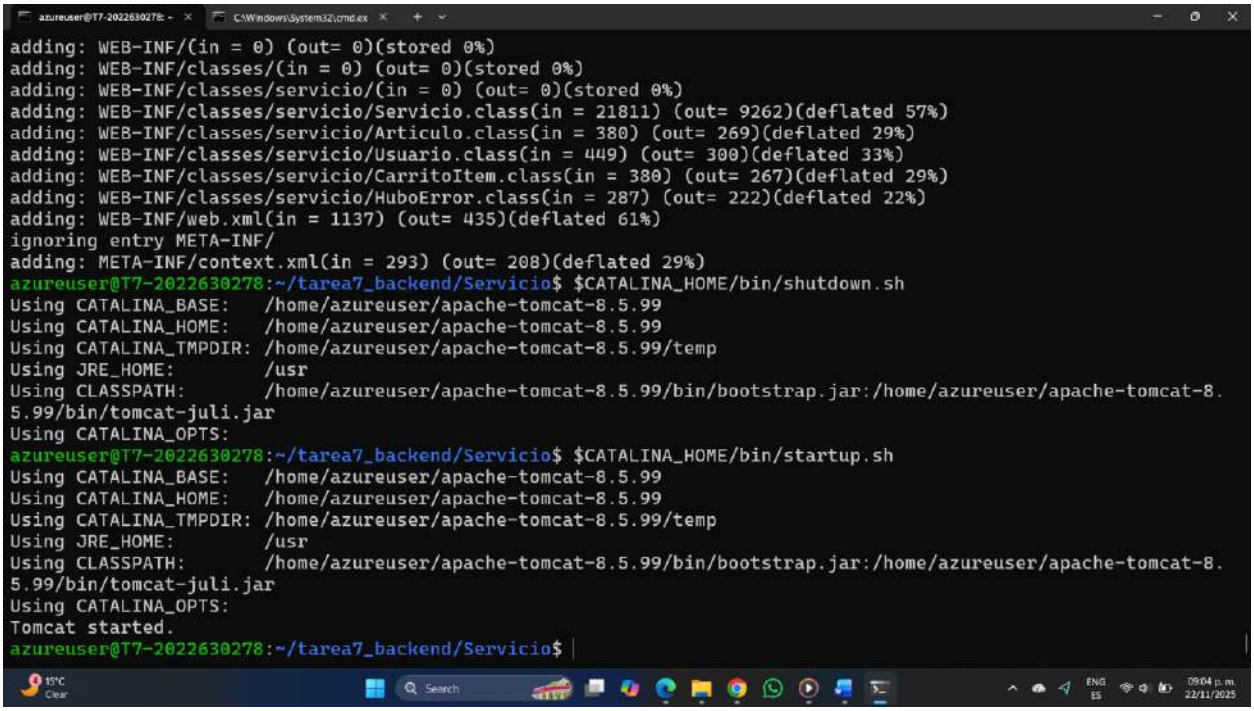

azureuser@T7-2022630278:~/tarea7_backend/Servicio$ cd ~/tarea7_backend/Servicio
azureuser@T7-2022630278:~/tarea7_backend/Servicio$ chmod +x compila.sh
azureuser@T7-2022630278:~/tarea7_backend/Servicio$ sh compila.sh
added manifest
adding: WEB-INF/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/servicio/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/servicio/Servicio.class(in = 21811) (out= 9262)(deflated 57%)
adding: WEB-INF/classes/servicio/Articulo.class(in = 380) (out= 269)(deflated 29%)
adding: WEB-INF/classes/servicio/Usuario.class(in = 449) (out= 300)(deflated 33%)
adding: WEB-INF/classes/servicio/CarritoItem.class(in = 380) (out= 267)(deflated 29%)
adding: WEB-INF/classes/servicio/HuboError.class(in = 287) (out= 222)(deflated 22%)
adding: WEB-INF/web.xml(in = 1137) (out= 435)(deflated 61%)
ignoring entry META-INF/
adding: META-INF/context.xml(in = 293) (out= 208)(deflated 29%)
azureuser@T7-2022630278:~/tarea7_backend/Servicio$ $CATALINA_HOME/bin/shutdown.sh
Using CATALINA_BASE: /home/azureuser/apache-tomcat-8.5.99
Using CATALINA_HOME: /home/azureuser/apache-tomcat-8.5.99
Using CATALINA_TMPDIR: /home/azureuser/apache-tomcat-8.5.99/temp
Using JRE_HOME: /usr
Using CLASSPATH: /home/azureuser/apache-tomcat-8.5.99/bin/bootstrap.jar:/home/azureuser/apache-tomcat-8.
5.99/bin/tomcat-juli.jar
Using CATALINA_OPTS:
azureuser@T7-2022630278:~/tarea7_backend/Servicio$ $CATALINA_HOME/bin/startup.sh
Using CATALINA_BASE: /home/azureuser/apache-tomcat-8.5.99
Using CATALINA_HOME: /home/azureuser/apache-tomcat-8.5.99
Using CATALINA_TMPDIR: /home/azureuser/apache-tomcat-8.5.99/temp
Using JRE_HOME: /usr
Using CLASSPATH: /home/azureuser/apache-tomcat-8.5.99/bin/bootstrap.jar:/home/azureuser/apache-tomcat-8.


adding: WEB-INF/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/servicio/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/servicio/Servicio.class(in = 21811) (out= 9262)(deflated 57%)
adding: WEB-INF/classes/servicio/Articulo.class(in = 380) (out= 269)(deflated 29%)
adding: WEB-INF/classes/servicio/Usuario.class(in = 449) (out= 300)(deflated 33%)
adding: WEB-INF/classes/servicio/CarritoItem.class(in = 380) (out= 267)(deflated 29%)
adding: WEB-INF/classes/servicio/HuboError.class(in = 287) (out= 222)(deflated 22%)
adding: WEB-INF/web.xml(in = 1137) (out= 435)(deflated 61%)
ignoring entry META-INF/
adding: META-INF/context.xml(in = 293) (out= 208)(deflated 29%)
azureuser@T7-2022630278:~/tarea7_backend/Servicio$ $CATALINA_HOME/bin/shutdown.sh
Using CATALINA_BASE: /home/azureuser/apache-tomcat-8.5.99
Using CATALINA_HOME: /home/azureuser/apache-tomcat-8.5.99
Using CATALINA_TMPDIR: /home/azureuser/apache-tomcat-8.5.99/temp
Using JRE_HOME: /usr
Using CLASSPATH: /home/azureuser/apache-tomcat-8.5.99/bin/bootstrap.jar:/home/azureuser/apache-tomcat-8.
5.99/bin/tomcat-juli.jar
Using CATALINA_OPTS:
azureuser@T7-2022630278:~/tarea7_backend/Servicio$ $CATALINA_HOME/bin/startup.sh
Using CATALINA_BASE: /home/azureuser/apache-tomcat-8.5.99
Using CATALINA_HOME: /home/azureuser/apache-tomcat-8.5.99
Using CATALINA_TMPDIR: /home/azureuser/apache-tomcat-8.5.99/temp
Using JRE_HOME: /usr
Using CLASSPATH: /home/azureuser/apache-tomcat-8.5.99/bin/bootstrap.jar:/home/azureuser/apache-tomcat-8.
5.99/bin/tomcat-juli.jar
Using CATALINA_OPTS:
Tomcat started.
azureuser@T7-2022630278:~/tarea7_backend/Servicio$

# 9 Implementación del Back-End

Esta sección detalla la ampliación del servicio REST desarrollado en la Tarea 2 (gestión de usuarios) para construir el prototipo de comercio electrónico solicitado en la Tarea 7. Se incorporan artículos, fotos de artículos, un carrito de compra y operaciones transaccionales, cumpliendo los requerimientos funcionales y no funcionales establecidos.

## 9.1 Estructura del proyecto

Después de descomprimir Servicio.zip y agregar las nuevas clases, la estructura queda:

Servicio/

├── META-INF/

│ └── context.xml

├── WEB-INF/

│ ├── classes/

│ │ └── servicio/ (archivos .class generados tras compilar)

│ └── web.xml

├── compila.sh

├── compila.bat

└── servicio/

 ├── HuboError.java

 ├── Servicio.java

 ├── Usuario.java

 ├── Articulo.java (nuevo)

 ├── CarritoItem.java (nuevo)

**Nota**: Al ejecutar compila.sh se empaqueta Servicio.war que Tomcat despliega en $CATALINA_HOME/webapps/Servicio.

- **9.2 Clases añadidas o modificadas**

| Clase | Rol | Comentarios |
|---|---|---|
| **Servicio.java** | Controlador REST principal | Se agregan endpoints de artículos y carrito; maneja transacciones. |
| **Usuario.java** | POJO de usuario | Sin cambios estructurales salvo uso extendido del token. |
| **Articulo.java** | POJO de artículo | Incluye id_articulo, datos básicos y foto (byte[]). |
| **CarritoItem.java** | POJO de ítem en carrito | No representa una tabla directa; se arma desde joins y calcula costo. |
| **HuboError.java** | Contenedor de error | Permite respuestas uniformes JSON con mensaje. |

## 9.2 Modificación del método login

Para optimizar el flujo se cambió el SELECT (antes devolvía sólo existencia). Ahora se obtiene también id_usuario, evitando consultas posteriores.

Consulta original (Tarea 2):

SELECT 1 FROM usuarios WHERE email=? AND password=?

Consulta nueva:

SELECT id_usuario FROM usuarios WHERE email=? AND password=?

La respuesta JSON ahora incluye:

```
{
  "token": "AB12CD34EF56GH78IJ90",
```

"id_usuario": 5

}

## 9.3 Endpoints añadidos y mapeo a requerimientos funcionales

| Endpoint (RF Back-End) | Método | Ruta relativa (/Servicio/rest/ws/...) | Query Params | Body JSON | Autenticación (token) | Descripción / Observaciones |
|---|---|---|---|---|---|---|
| **RF1 alta_articulo** | POST | alta_articulo | id_usuario, token | Articulo (sin id_articulo) | Sí | Inserta artículo en stock y foto en fotos_articulos. |
| **RF2 consulta_articulos** | GET | consulta_articulos | palabra, id_usuario, token | — | Sí | Búsqueda con LIKE en nombre o descripcion. Devuelve lista con campos requeridos. |
| **RF3 compra_articulo** | POST | compra_articulo | id_usuario, token | {id_articulo, cantidad} | Sí | Verifica stock, inserta/actualiza carrito y descuenta stock. Errores RF3.3. |
| **RF4 elimina_articulo_carrito_compra** | DELETE | elimina_articulo_carrito_compra | id_usuario, id_articulo, token | — | Sí | Regresa cantidad al stock y elimina ítem del carrito. |
| **RF5 elimina_carrito_compra** | DELETE | elimina_carrito_compra | id_usuario, token | — | Sí | Restituye stock de cada artículo y limpia el carrito. |
| **RF6 modifica_carrito_compra** | PUT | modifica_carrito_compra | id_usuario, id_articulo, incremento, token | (vacío) | Sí | Incrementa/decrementa cantidad en carrito; valida stock o límites. |
| **(Apoyo) consulta_carrito_compra** | GET | consulta_carrito_compra | id_usuario, token | — | Sí | Devuelve ítems del carrito y total. Facilita Front-End RF5–RF8. |

**Imagen 38. Tabla de endpoints en el editor.**

## 9.4 Correspondencia con requerimientos funcionales

| Requerimiento original | Implementación en código |
|---|---|

| | |
|---|---|
| **RF1 Alta artículo** | Método alta_articulo |
| **RF2 Consulta artículos** | Método consulta_articulos |
| **RF3 Compra artículo** | Método compra_articulo (transacción + validaciones) |
| **RF4 Elimina artículo carrito** | Método elimina_articulo_carrito_compra |
| **RF5 Elimina carrito completo** | Método elimina_carrito_compra |
| **RF6 Modifica cantidad carrito** | Método modifica_carrito_compra |
| **(Front-End soporte) Carrito con total** | Método consulta_carrito_compra |

**Imagen 42. Tabla de correspondencia en el editor.**

# 10 Implementación del Front-end (prueba.html y JS) para la Tarea 7

Se extiende el prueba.html de la Tarea 2 para incorporar las tres nuevas vistas relacionadas con el comercio electrónico: Captura de artículo, Compra de artículos y Carrito de compra, cumpliendo los requerimientos funcionales del front-end (1–8).

## 10.1 Enfoque y organización

1. Página única (SPA ligera) mediante mostrar/ocultar  (patrón ya existente).
2. Reutilización de variables globales: email, token, y nueva id_usuario.
3. Reutilización de WSClient.js para todas las peticiones (uniformidad).
4. Estructura clara de secciones: login, alta_usuario, consulta_usuario, menu, captura_articulo, compra _articulos, carrito.
5. Manejo de imágenes base64 aprovechando la misma función readSingleFile. (Se podría separar en foto_usuario y foto_articulo como mejora futura.)

**Imagen 44. Secciones nuevas en prueba.html.**

6. **10.2 Correspondencia con requerimientos front-end**

| Requerimiento Front-End | Implementación |
|---|---|
| **1 Captura de artículo** | Vista captura_articulo + función alta_articulo() |
| **2 Compra de artículos (búsqueda)** | Vista compra_articulos + consulta_articulos() |
| **3 Mostrar datos y controles por artículo** | Render dinámico: imagen, nombre, descripción, precio, cantidad, botones + / - / Compra |

| | |
|---|---|
| **4 Botón Compra** | Botón “Compra” → comprar_articulo(id) |
| **5 Carrito de compra con total** | Vista carrito + consulta_carrito() calcula y muestra total |
| **6 Eliminar artículo del carrito** | Botón “Eliminar artículo” → elimina_articulo_carrito(id) |
| **7 Eliminar carrito completo** | Botón “Eliminar carrito” → elimina_carrito() |
| **8 Seguir comprando** | Botón “Seguir comprando” regresa a compra_articulos |

**Imagen 45. Captura de la vista “Compra de artículos” mostrando lista y botones.**

7. **10.3 Flujo de interacción**

1. Login (obtiene token + id_usuario).
2. Menú principal: acceso a perfil, captura y compra.
3. Captura de artículo: envía POST y regresa al menú.
4. Compra: búsqueda → listado → compras individuales → acceso al carrito.
5. Carrito: ver artículos + total, modificar cantidades, eliminar ítem o vaciar.
6. Seguir comprando retorna a la búsqueda sin perder sesión.

Imagen 46. Diagrama simple del flujo (opcional).

## 10.2 Funciones JavaScript clave

| Función | Endpoint | Propósito | Observaciones |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **login()** | login | Autentica y obtiene token + id_usuario. | Hash SHA-256 en cliente. |
| **alta_articulo()** | alta_articulo | Envía datos y foto de nuevo artículo. | Valida campos numéricos. |
| **consulta_articulos()** | consulta_articulos | Obtiene artículos filtrados. | Renderiza dinámicamente. |
| **incrementa()/decrementa()** | — | Ajusta cantidad antes de compra. | No llaman al servidor. |
| **comprar_articulo()** | compra_articulo | Agrega al carrito y descuenta stock. | Maneja mensajes de error. |
| **consulta_carrito()** | consulta_carrito_compra | Lista ítems y total. | Reutiliza layout de resultados. |
| **elimina_articulo_carrito()** | elimina_articulo_carrito_compra | Borra ítem devolviendo stock. | Refresca carrito. |
| **elimina_carrito()** | elimina_carrito_compra | Limpia todo el carrito. | Confirmación previa. |
| **modifica_carrito()** | modifica_carrito_compra | Ajusta +1/-1 artículo en carrito. | Valida errores límite y stock. |

| | | | |
|---|---|---|---|
| **limpia_captura_articulo()** | — | Resetea campos y imagen al entrar. | Evita arrastrar foto previa. |

**Imagen 47. Fragmento JS mostrando comprar_articulo().**

# 11 Endpoints REST

Esta sección describe cada endpoint del servicio REST publicado sobre la base URL:

http:// 4.248.144.161:8080/Servicio/rest/ws

Incluye: propósito, método HTTP, ruta, parámetros (query / body), ejemplo de petición con curl, respuestas exitosas, errores frecuentes y notas sobre transacciones, autenticación (token) y validaciones. La numeración de imágenes continúa la secuencia del reporte (todas las capturas deben ser completas con fecha y hora, según lineamientos).

## 11.1 Convenciones generales

- Autenticación: los métodos protegidos verifican token y email o id_usuario mediante las funciones verifica_acceso.
- Códigos HTTP:
    - 200 Éxito.
    - 400 Error de lógica / validación / acceso denegado.
- Formato de errores: { "message": "<Descripción del error>" }
- Base de rutas: todos los endpoints agregan su nombre al final de /Servicio/rest/ws.
- Parámetros:
    - Query string para login, token, ids e incremento.
    - Cuerpo JSON para datos de usuario, artículo o compra.
- Transacciones: Alta/modificación/eliminación que involucran varias tablas usan setAutoCommit(false) y commit/rollback.

**Imagen 46: Captura del listado de métodos en el código fuente (scroll mostrando anotaciones @Path).**

### 11.2 Endpoint: login

| Aspecto | Detalle |
|---|---|
| **Método** | GET |
| **Ruta** | /login |
| **Autenticado** | No (emite token) |
| **Query Params** | email, password (hash SHA-256 ya generado en cliente) |
| **Body** | No aplica |
| **Transacción** | No |

### 11.3 Endpoint: alta_usuario

| Aspecto | Detalle |
|---|---|
| **Método** | POST |
| **Ruta** | /alta_usuario |
| **Autenticado** | No (registro) |
| **Body JSON** | Datos del usuario (email, password hash, nombre, apellidos, fecha_nacimiento ISO, teléfono, género, foto base64 opcional) |
| **Transacción** | Sí (inserta usuario y foto) |

### 11.4 Endpoint: consulta_usuario

| Aspecto | Detalle |
|---|---|
| **Método** | GET |
| **Ruta** | /consulta_usuario |
| **Autenticado** | Sí (email, token) |

| | |
|---|---|
| **Query Params** | email, token |
| **Transacción** | No (lectura) |

### 11.5 Endpoint: modifica_usuario

| Aspecto | Detalle |
|---|---|
| **Método** | PUT |
| **Ruta** | /modifica_usuario |
| **Autenticado** | Sí |
| **Query Params** | email, token |
| **Body JSON** | Campos modificables (password opcional no vacío para cambio) |
| **Transacción** | Sí (actualiza usuario y foto) |

### 11.6 Endpoint: borra_usuario

| Aspecto | Detalle |
|---|---|
| **Método** | DELETE |
| **Ruta** | /borra_usuario |
| **Autenticado** | Sí |
| **Query Params** | email, token |
| **Transacción** | Sí (borra fotos, luego usuario) |

### 11.7 Endpoint: alta_articulo

| Aspecto | Detalle |
|---|---|
| **Método** | POST |
| **Ruta** | /alta_articulo?id_usuario&token |

| | |
|---|---|
| **Autenticado** | Sí |
| **Query Params** | id_usuario, token |
| **Body JSON** | nombre, descripcion, precio, cantidad, foto (opcional) |
| **Transacción** | Sí (inserta stock y foto) |

### 11.8 Endpoint: consulta_articulos

| Aspecto | Detalle |
|---|---|
| **Método** | GET |
| **Ruta** | /consulta_articulos?palabra&id_usuario&token |
| **Autenticado** | Sí |
| **Query Params** | palabra, id_usuario, token |
| **Body** | No |
| **Transacción** | No (lectura) |

### 11.9 Endpoint: compra_articulo

| Aspecto | Detalle |
|---|---|
| **Método** | POST |
| **Ruta** | /compra_articulo?id_usuario&token |
| **Autenticado** | Sí |
| **Query Params** | id_usuario, token |
| **Body JSON** | id_articulo, cantidad |
| **Transacción** | Sí (carrito + stock) |

### 11.10 Endpoint: elimina_articulo_carrito_compra

| Aspecto | Detalle |
|---|---|
| **Método** | DELETE |
| **Ruta** | /elimina_articulo_carrito_compra?id_usuario&id_articulo&token |
| **Autenticado** | Sí |
| **Query Params** | id_usuario, id_articulo, token |
| **Transacción** | Sí (stock + borrado carrito) |

### 11.11 Endpoint: elimina_carrito_compra

| Aspecto | Detalle |
|---|---|
| **Método** | DELETE |
| **Ruta** | /elimina_carrito_compra?id_usuario&token |
| **Autenticado** | Sí |
| **Query Params** | id_usuario, token |
| **Transacción** | Sí (recorre artículos y revierte cantidades) |

### 11.12 Endpoint: modifica_carrito_compra

| Aspecto | Detalle |
|---|---|
| **Método** | PUT |
| **Ruta** | /modifica_carrito_compra?id_usuario&id_articulo&incremento&token |
| **Autenticado** | Sí |
| **Query Params** | id_usuario, id_articulo, incremento (1 o -1), token |
| **Body** | Vacío ({}) |

| | |
|---|---|
| **Transacción** | Sí (stock y carrito) |

### 11.13 Endpoint: consulta_carrito_compra

| Aspecto | Detalle |
|---|---|
| **Método** | GET |
| **Ruta** | /consulta_carrito_compra?id_usuario&token |
| **Autenticado** | Sí |
| **Query Params** | id_usuario, token |
| **Transacción** | No (lectura) |

# 12 Pruebas Unitarias del Back-End (curl)

En esta sección se documentan de forma sistemática las pruebas unitarias realizadas sobre cada endpoint REST del back-end. Se incluyen: preparación de datos, comandos curl, respuestas esperadas y ejemplos de errores controlados. Todas las capturas deben ser pantallas completas (sin recortes), con fecha y hora visibles (barra de tareas o terminal con marca temporal). La numeración de imágenes continúa desde la anterior sección (sección 11 terminó en Imagen 61); aquí partimos en la Imagen 62.

## 12.1 Metodología y Preparación

Las pruebas se ejecutaron directamente contra la VM T7-2022630278 en Azure, usando su IP pública (reemplazar 4.248.144.161 en ejemplos). Se emplearon comandos curl desde:

- Terminal Linux (SSH dentro de la VM) y/o
- Terminal local (Git Bash en Windows 11), asegurando conectividad a puerto 8080.

Pasos previos:

1. Verificar que Tomcat y el servicio Servicio.war estén desplegados y funcionando.
2. Confirmar que la base de datos servicio_web contenga las tablas: usuarios, fotos_usuarios, stock, fotos_articulos, carrito_compra.
3. Generar hash SHA-256 de la contraseña en el cliente (ejemplo “ClavePrueba2025”):
    - Uso rápido en Linux:
    - echo -n "x" | sha256sum | cut -d ' ' -f1
4. Utilizar ese hash en las peticiones de registro (alta_usuario) y login.

Consideraciones de captura:

- Mostrar comando ejecutado y respuesta completa JSON.

- Incluir evidencia de casos exitosos y fallidos (códigos 200 y 400).
- Mantener coherencia de email, id_usuario, token y id_articulo entre pruebas (documentar valores reales).

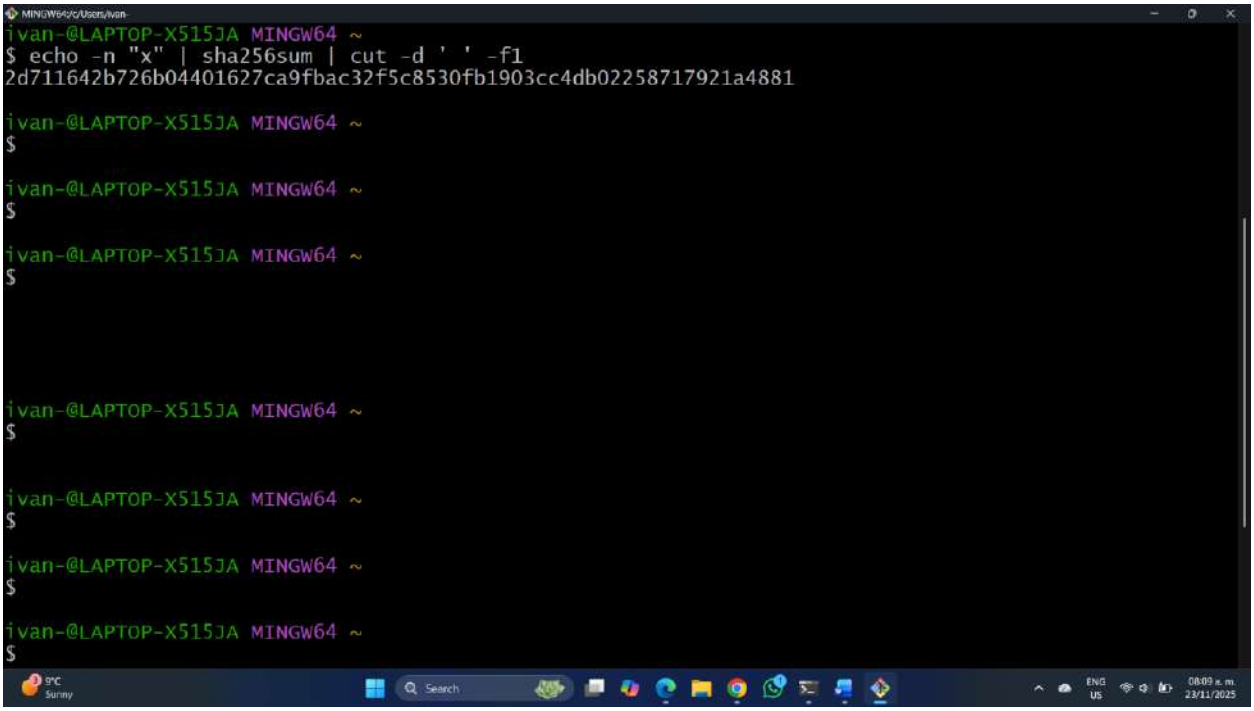



**Imagen 62: Terminal mostrando hash generado para la contraseña.**

## 12.2 Evidencias por Endpoint

Para claridad, se usa el siguiente usuario de pruebas (ejemplo):

- Email: x@x.com
- Hash de password: 2d711642b726b04401627ca9fbac32f5c8530fb1903cc4db02258717921a4881
- Después del login: id_usuario = 5 (ejemplo), token = <TOKEN_REAL>

Se describe cada endpoint (12.2.1 a 12.2.12) con:

- Preparación específica si aplica.
- Comandos curl (éxito y error).
- Respuesta esperada y observada.
- Imagen asociada (pantalla completa).

### 12.2.1 login

Comando (tras haber registrado el usuario):

curl "http://4.248.144.161:8080/Servicio/rest/ws/login?email=x@x.com&password=2d711642b726b04401627ca9fbac32f5c8530fb1903cc4db02258717921a4881"

Respuesta esperada (código 200):

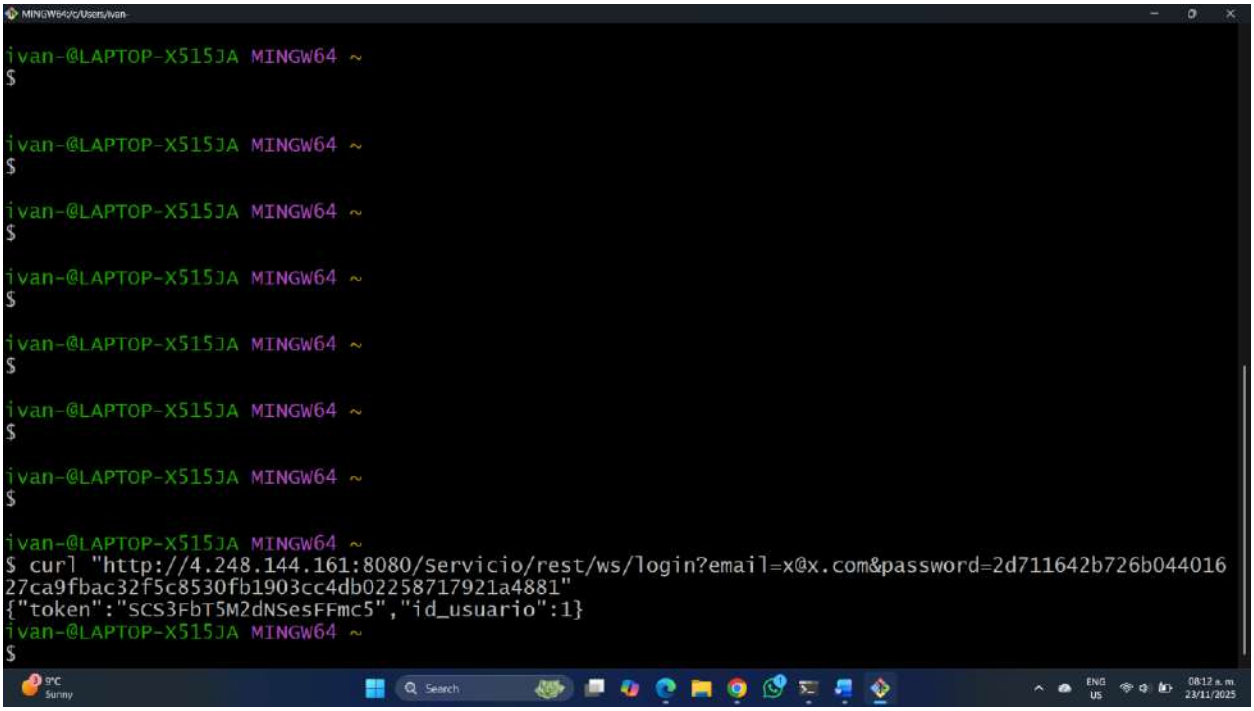



**Imagen 63: Login exitoso (con token e id_usuario).**

Caso de error (password incorrecto):

curl "http://4.248.144.161:8080/Servicio/rest/ws/login?email=x@x.com&password=0000"

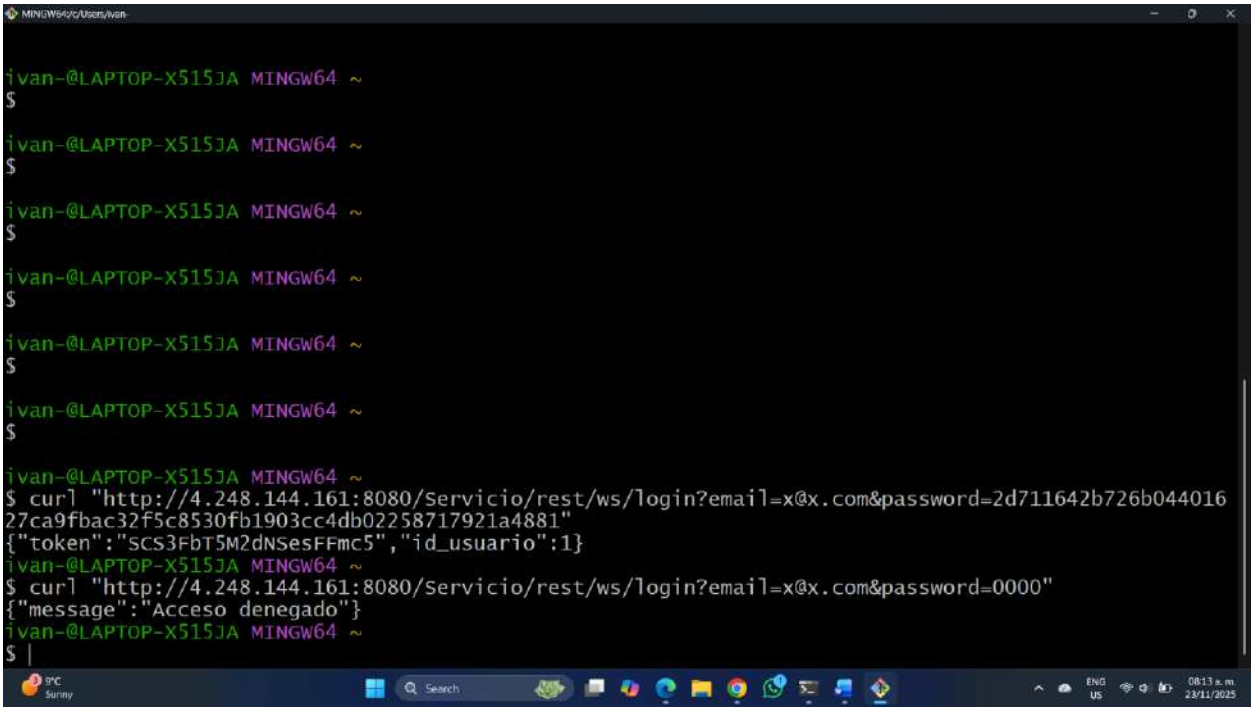



**Imagen 64: Login erróneo (mensaje Acceso denegado).**

### 12.2.2 alta_usuario

Comando (registro inicial):

curl -X POST "http://4.248.144.161:8080/Servicio/rest/ws/alta_usuario" \ -H "Content-Type: application/json" \ -d '{ "email":"x2@x2.com", "password":"x2", "nombre":"Test", "apellido_paterno":"Ejemplo", "apellido_materno":null, "fecha_nacimiento":"2024-05-01T12:00:00.000Z", "telefono":5512345678, "genero":"M", "foto":null }'

Respuesta esperada (200):

{ "mensaje": "Se dio de alta el usuario" }

Error (falta nombre):

curl -X POST "http://4.248.144.161:8080/Servicio/rest/ws/alta_usuario" \ -H "Content-Type: application/json" \ -d '{"email":"u@d.com","password":"<HASH>","nombre":"","apellido_paterno":"X","fecha_nacimiento":"2024-01-01T10:00:00.000Z"}'

Respuesta esperada (400):

{ "message":"Se debe ingresar el nombre" }

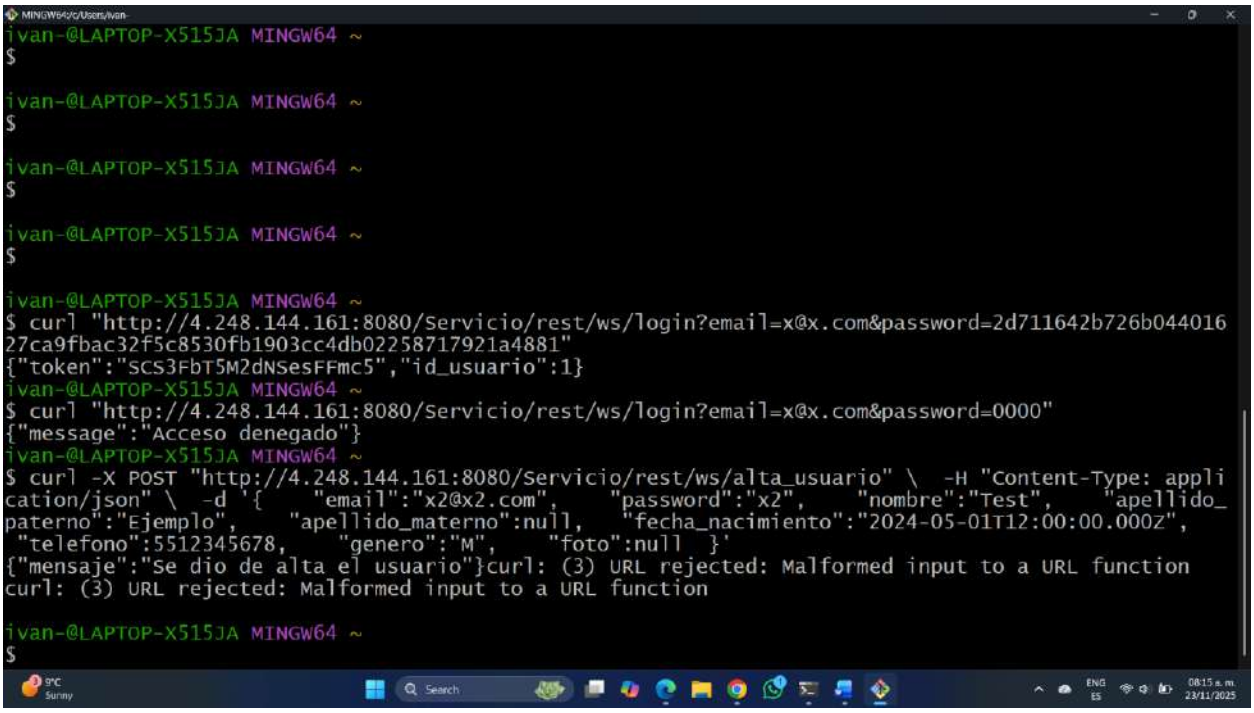



**Imagen 65: Alta exitosa.**

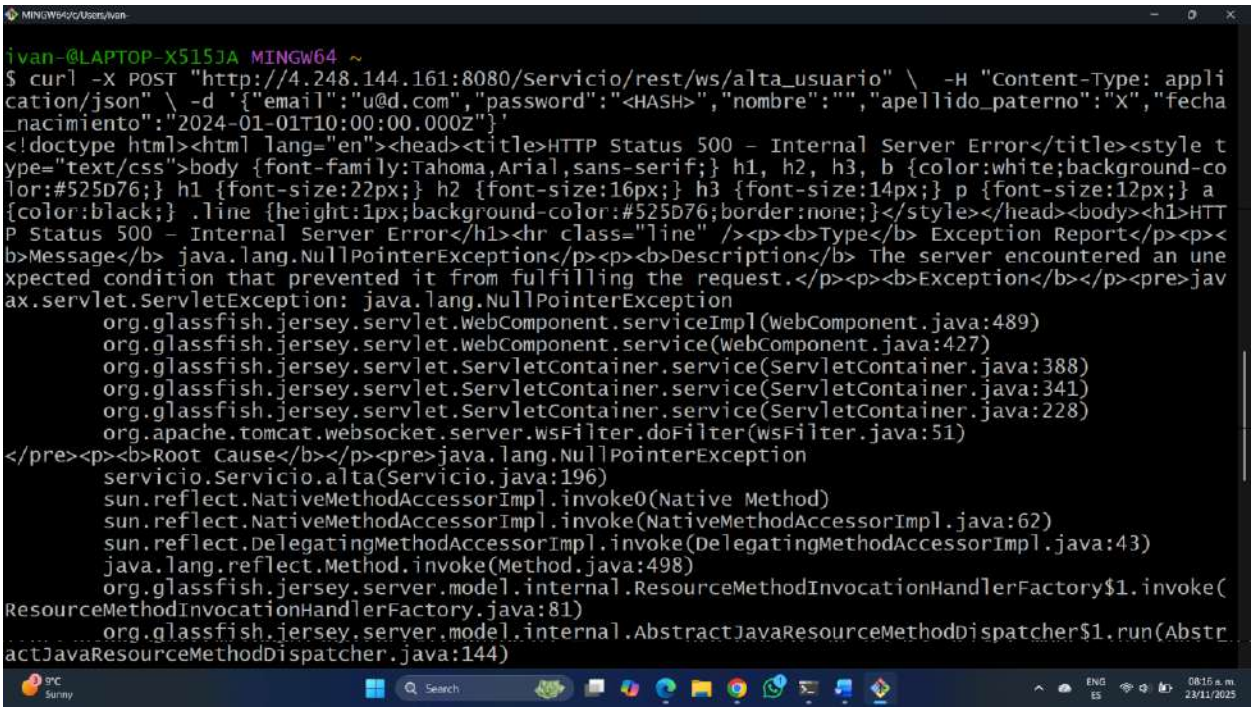



**Imagen 66: Error por campo obligatorio faltante.**

### 12.2.3 consulta_usuario

Con token obtenido del login:

curl "http://4.248.144.161:8080/Servicio/rest/ws/consulta_usuario?email=x@x.com&token=SCS3FbT5M2dNSesFFmc5"

Respuesta esperada (200): Datos del usuario y foto (null o base64). Error (token inválido):

curl "http://4.248.144.161:8080/Servicio/rest/ws/consulta_usuario?email=x@x.com&token=ZZZZ"

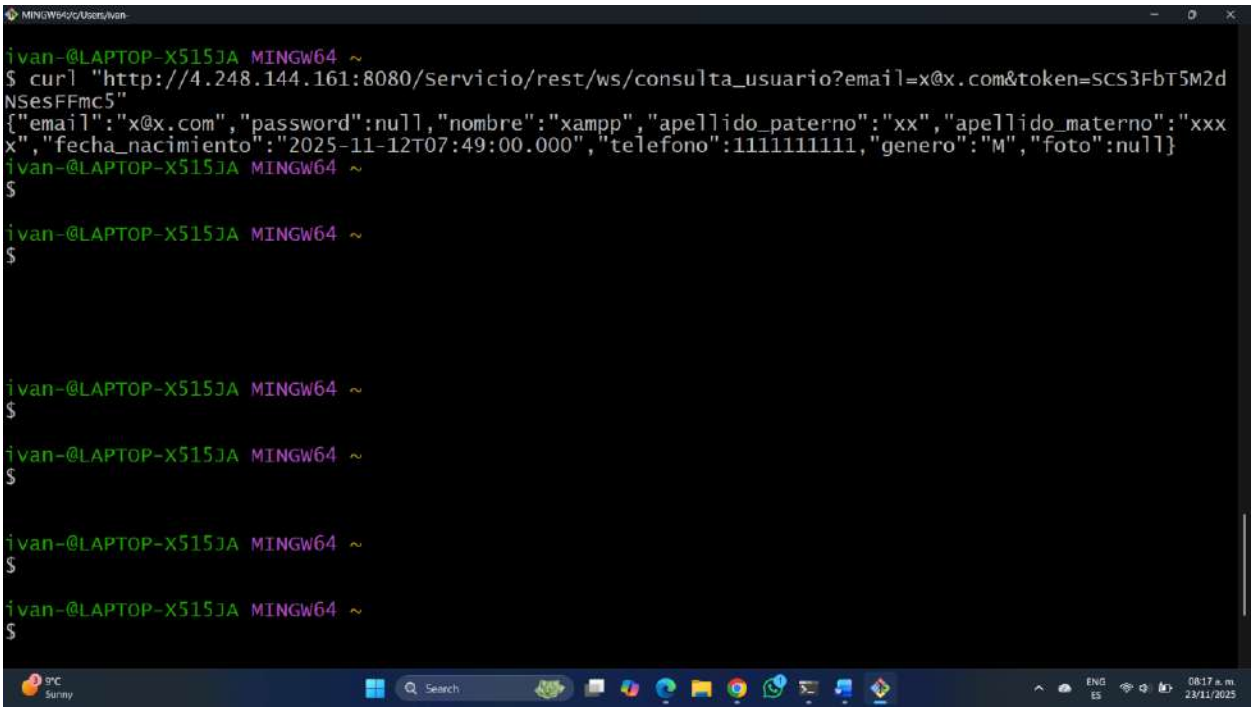



**Imagen 67: Consulta exitosa.**

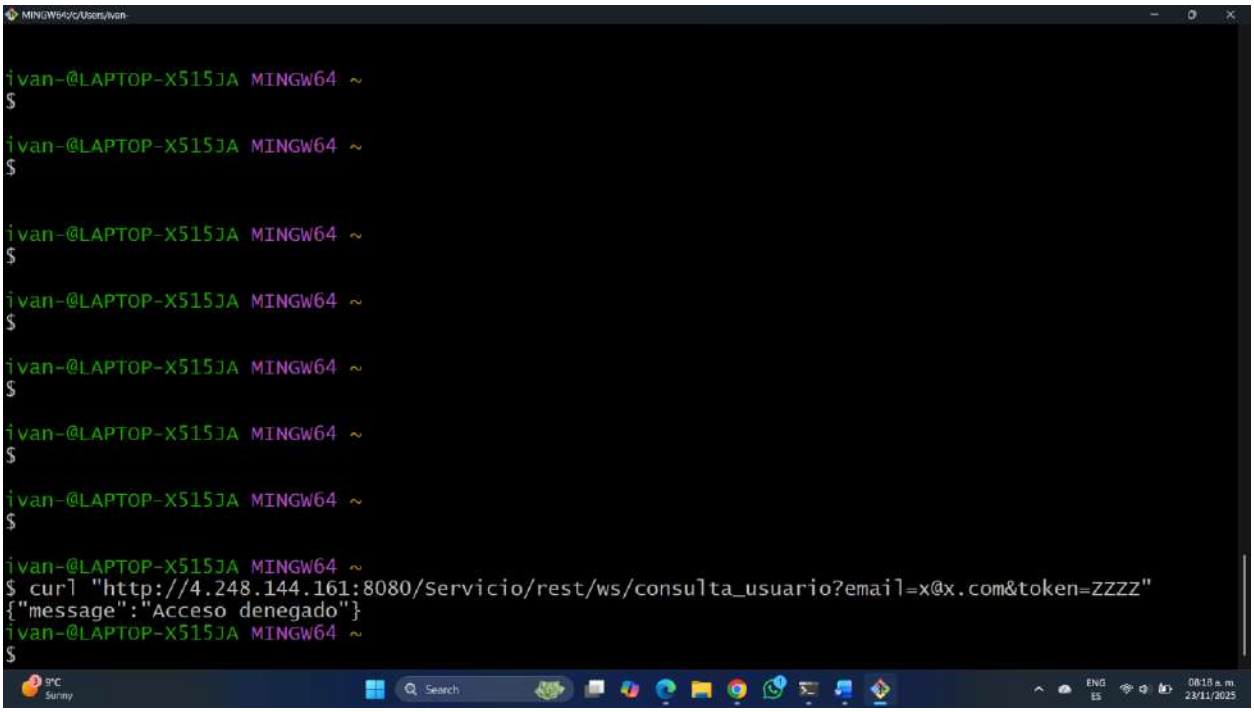



**Imagen 68: Error acceso denegado (token incorrecto).**

### 12.2.4 modifica_usuario

Cambiar nombre y teléfono (mantener token vigente):

curl -X PUT "http://4.248.144.161:8080/Servicio/rest/ws/modifica_usuario?email=x@x.com&token=SCS3FbT5M2dNSesFFmc5" \ -H "Content-Type: application/json" \ -d '{ "password":"", "nombre":"TestMod", "apellido_paterno":"Ejemplo", "apellido_materno":null, "fecha_nacimiento":"2024-05-01T12:00:00.000Z", "telefono":5511111111, "genero":"M", "foto":null

}'

Respuesta esperada (200):

{ "mensaje": "Se modificó el usuario" }

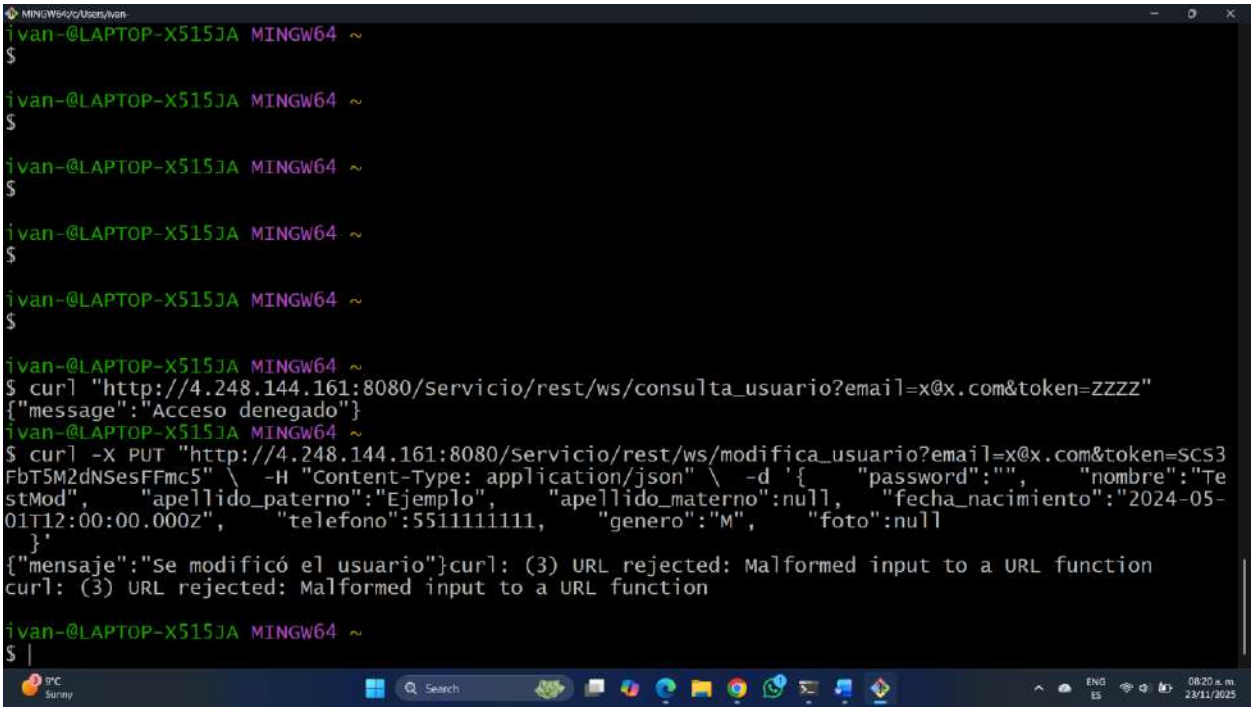



**Imagen 69: Modificación exitosa.**

### 12.2.5 borra_usuario (opcional, si se prueba al final)

curl -X DELETE "http://4.248.144.161:8080/Servicio/rest/ws/borra_usuario?email=x2@x2.com&token=sKXJbMGDN90EW3scAyNw"

Respuesta esperada (200):

{ "mensaje": "Se borró el usuario" }

Error (email no existente):

curl -X DELETE "http://4.248.144.161:8080/Servicio/rest/ws/borra_usuario?email=noexiste@dominio.com&token=SCS3FbT5M2dNSesFFmc5"

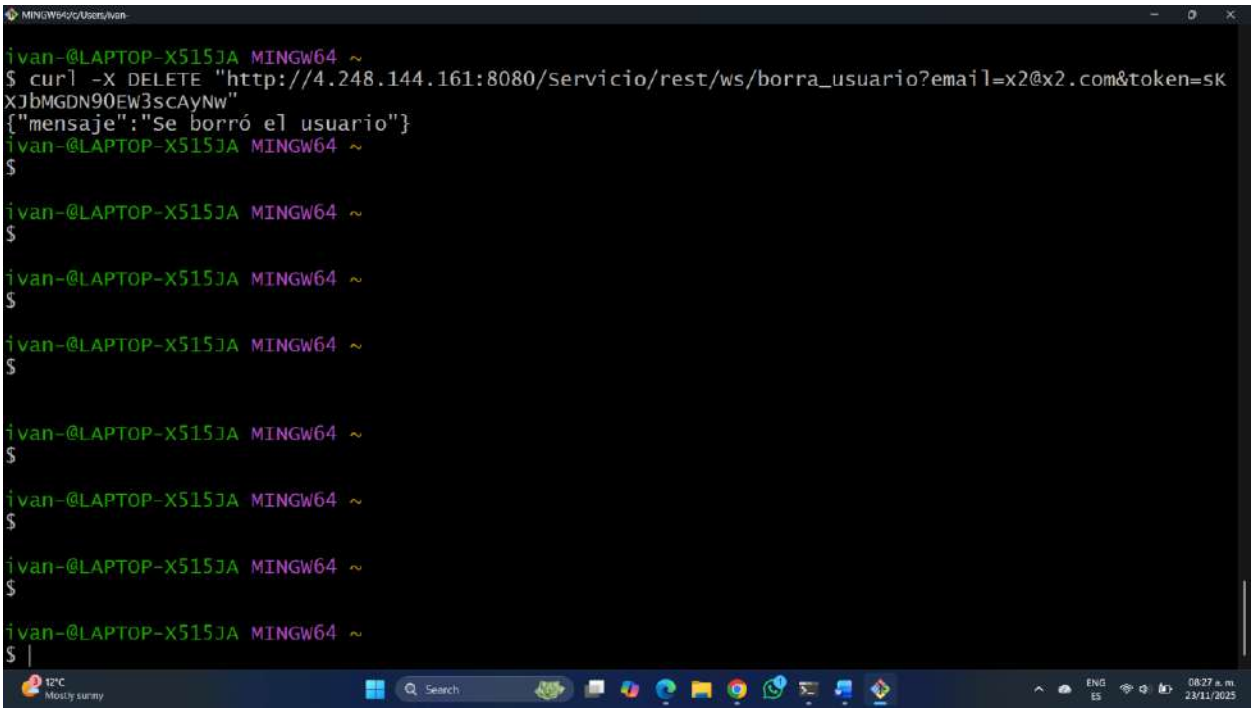



**Imagen 71: Borrado exitoso.**

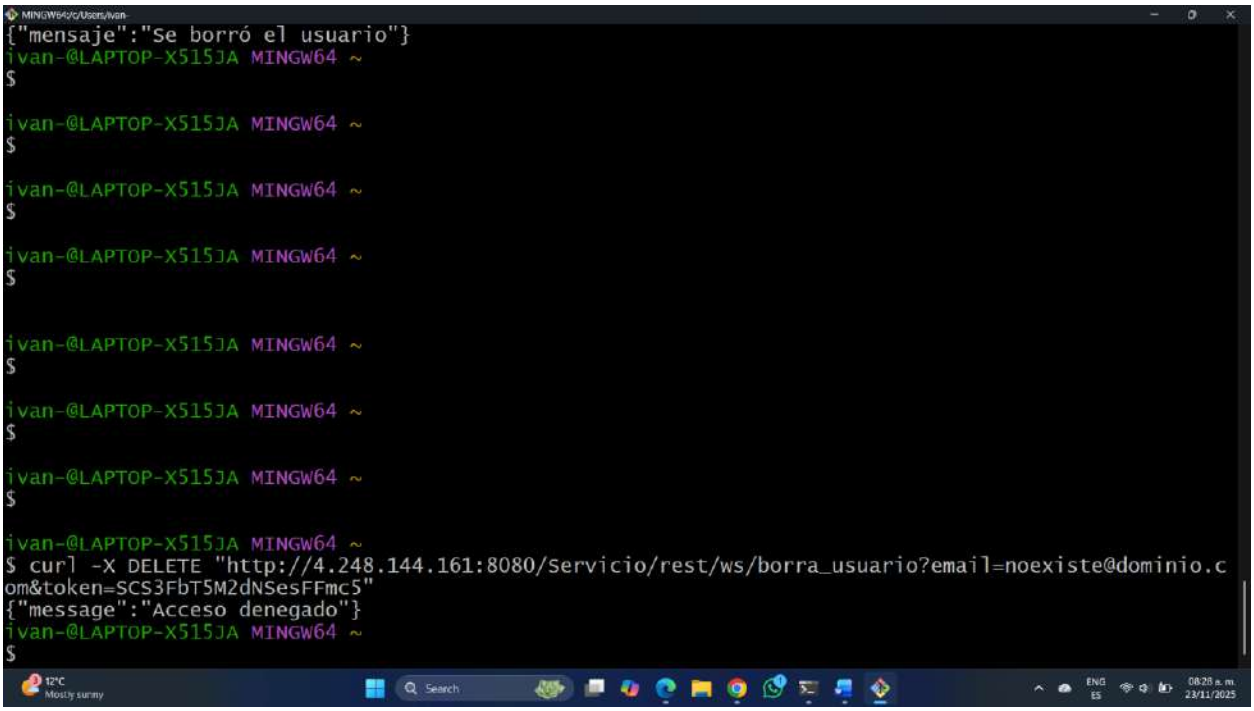



**Imagen 72: Error email inexistente.**

### 12.2.6 alta_articulo

Registrar artículo:

curl -X POST "http://4.248.144.161:8080/Servicio/rest/ws/alta_articulo?id_usuario=1&token=SCS3FbT5M2dNSesFFmc5" \ -H "Content-Type: application/json" \ -d '{ "nombre":"Mayonesa 500g", "descripcion":"Envase familiar", "precio":35.50, "cantidad":100, "foto":null }'

Respuesta esperada (200):

{ "mensaje":"Alta artículo OK" }

Error (precio inválido):

curl -X POST "http://4.248.144.161:8080/Servicio/rest/ws/alta_articulo?id_usuario=1&token=SCS3Fb

T5M2dNSesFFmc5" \ -H "Content-Type: application/json" \ -d '{"nombre":"X","descripcion":"Y","precio":0,"cantidad":10,"foto":null}'

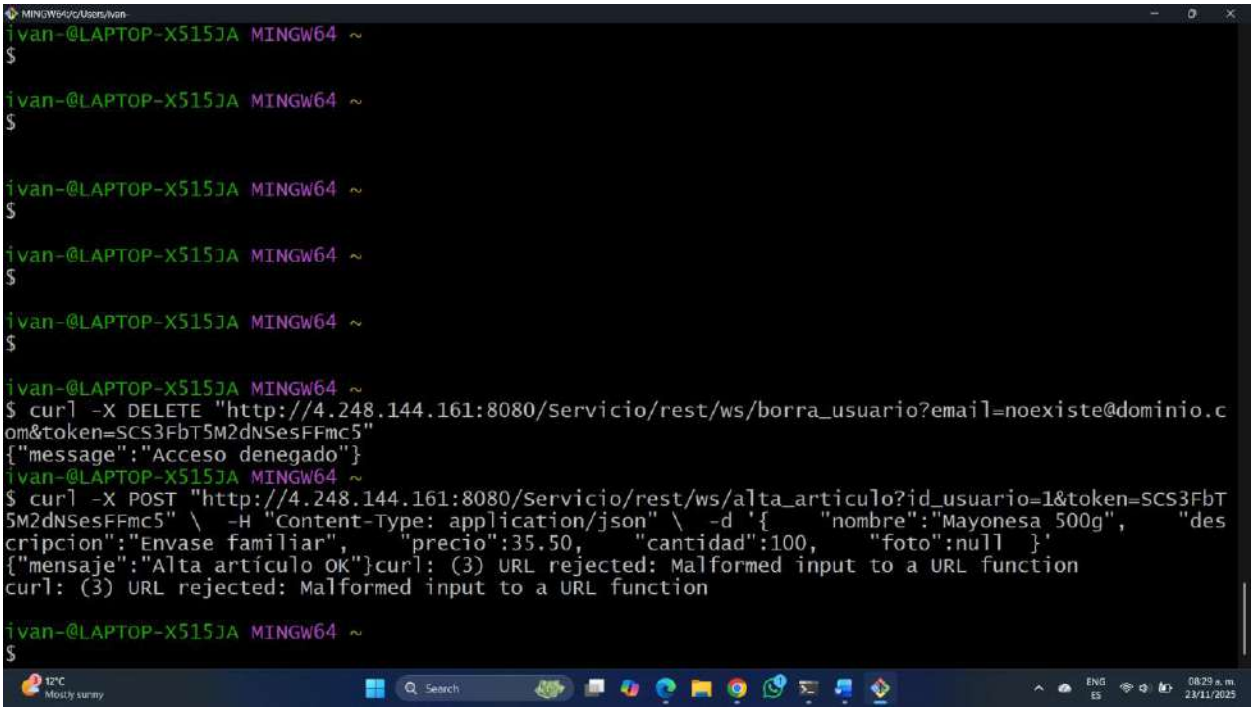



**Imagen 73: Alta artículo exitosa.**

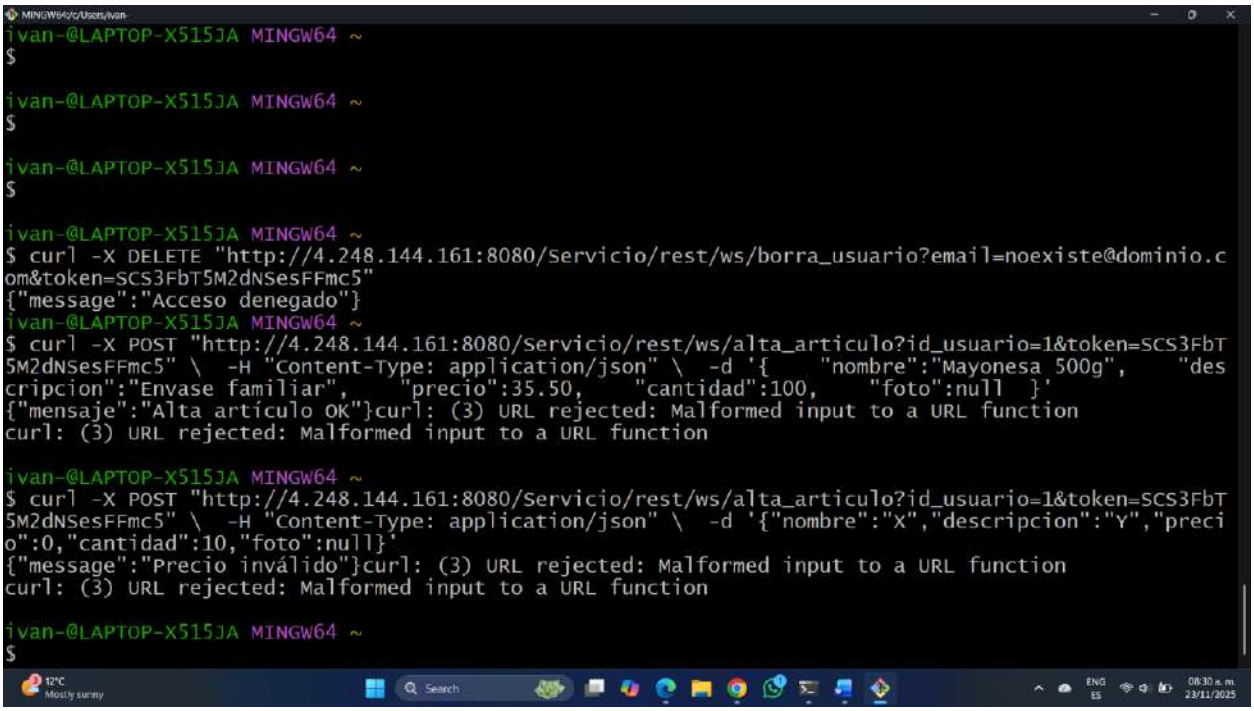



**Imagen 74: Error precio inválido.**

### 12.2.7 consulta_articulos

Búsqueda por palabra:

curl "http://4.248.144.161:8080/Servicio/rest/ws/consulta_articulos?palabra=mayonesa&id_usuario=1&token=SCS3FbT5M2dNSesFFmc5"

Respuesta esperada (200): Arreglo con artículos que hacen match.

Error (token inválido):

curl "http://4.248.144.161:8080/Servicio/rest/ws/consulta_articulos?palabra=mayonesa&id_usuario=5&token=ZZZZ"

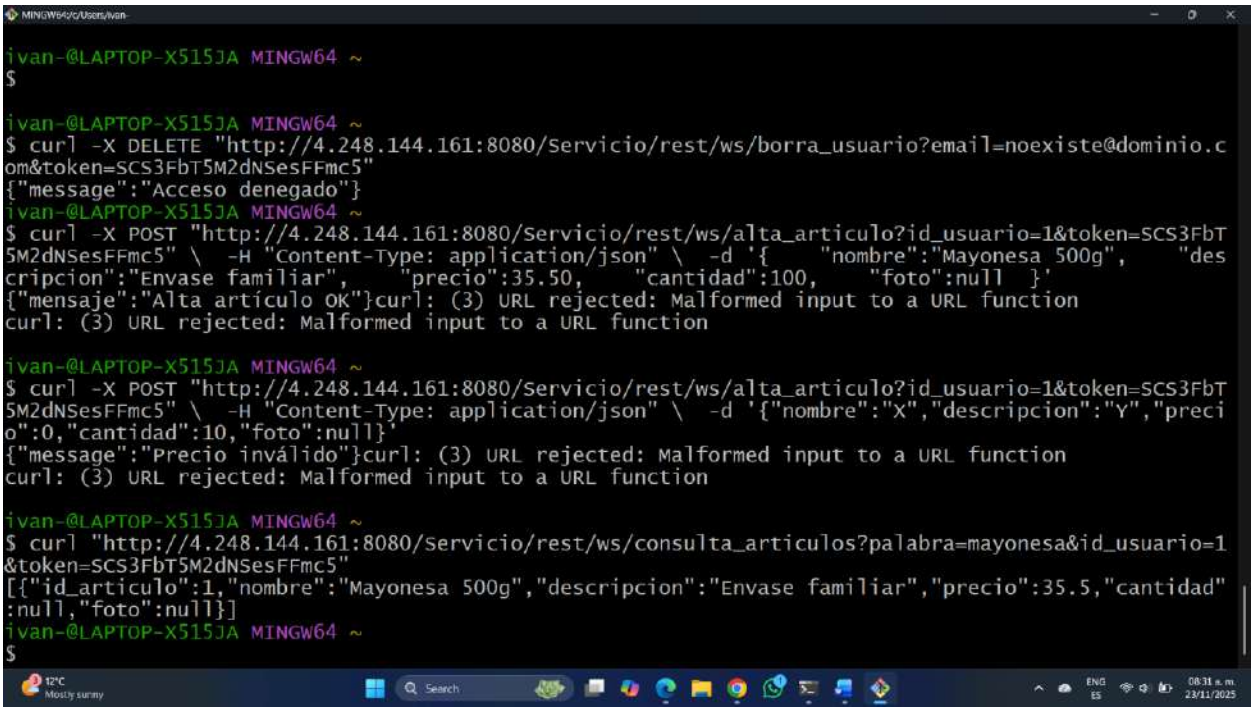



**Imagen 75: Lista con artículo encontrado.**

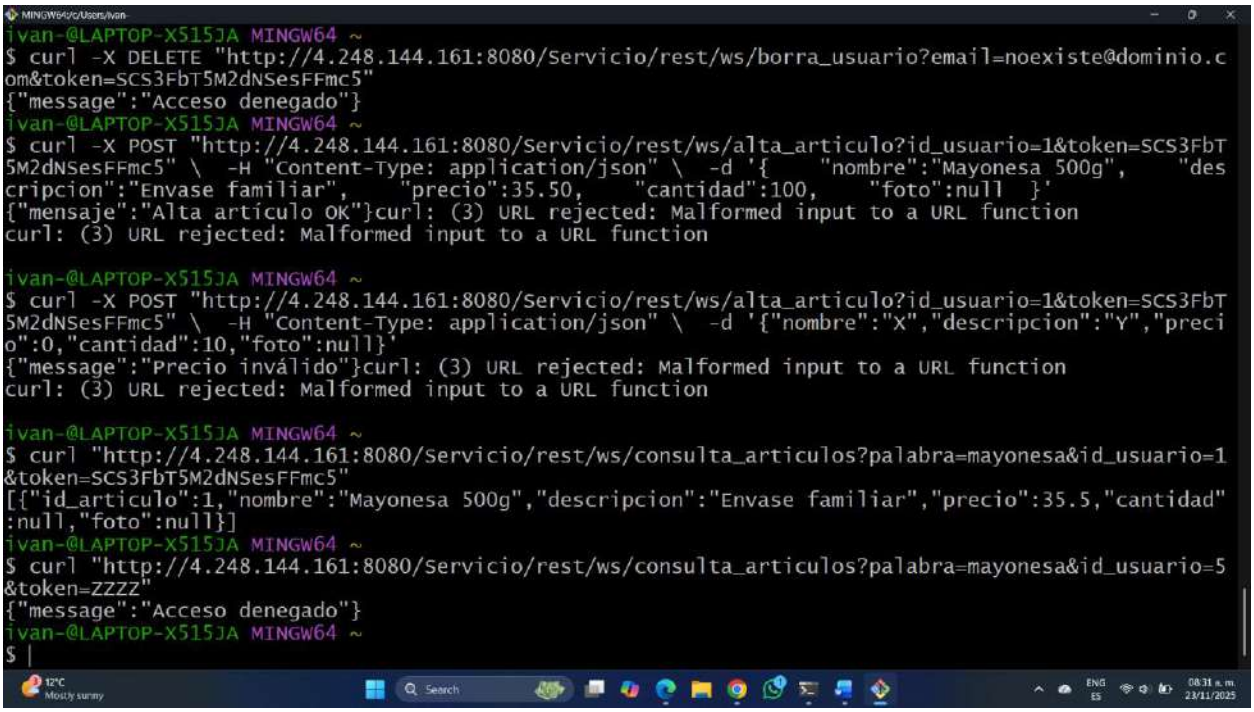



**Imagen 76: Error acceso denegado.**

### 12.2.8 compra_articulo

Comprar 3 unidades (asumiendo id_articulo=1):

curl -X POST "http://4.248.144.161:8080/Servicio/rest/ws/compra_articulo?id_usuario=1&token=SCS3FbT5M2dNSesFFmc5" \ -H "Content-Type: application/json" \ -d '{"id_articulo":1,"cantidad":3}'

Respuesta esperada (200):

{ "mensaje":"Compra realizada" }

Error (cantidad mayor al stock remanente):

curl -X POST "http://4.248.144.161:8080/Servicio/rest/ws/compra_articulo?id_usuario=1&token=SCS3FbT5M2dNSesFFmc5" \ -H "Content-Type: application/json" \ -d '{"id_articulo":1,"cantidad":1000}'

Respuesta (400):

{ "message":"No hay suficientes artículos en stock" }

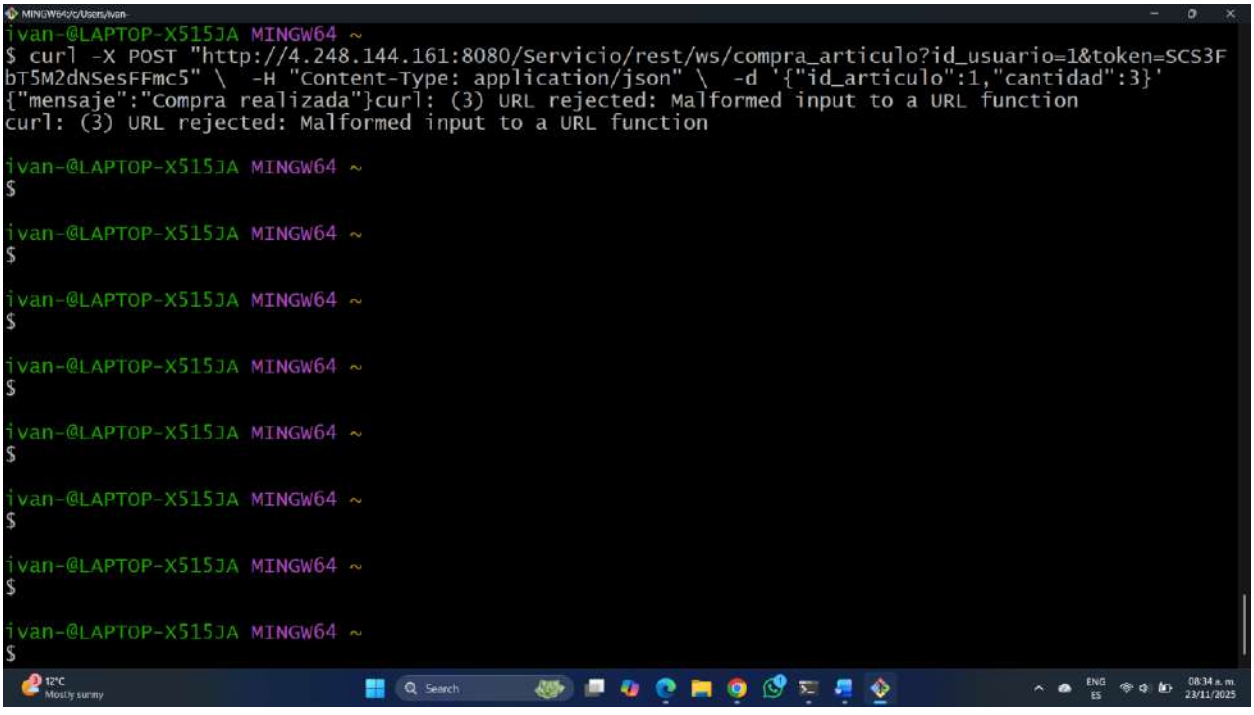


**Imagen 77: Compra exitosa.**

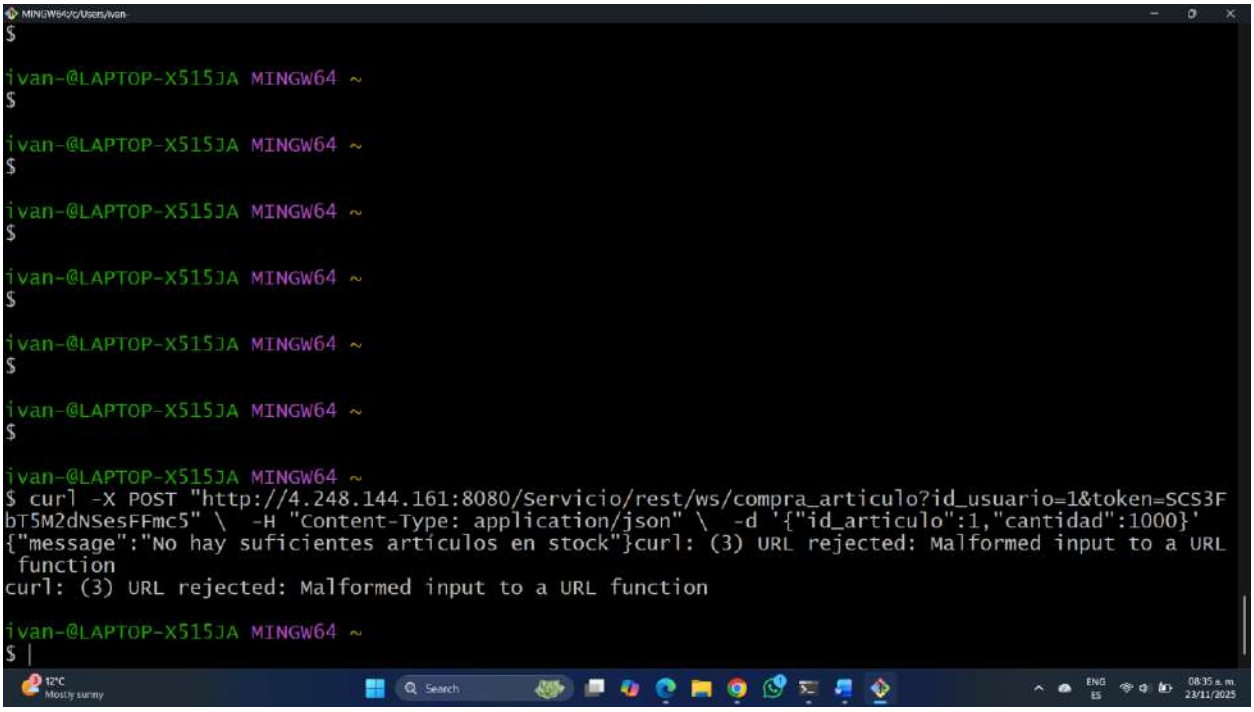



**Imagen 78: Error stock insuficiente.**

- **12.2.9 modifica_carrito_compra**

Incrementar 1 unidad:

curl -X PUT "http://4.248.144.161:8080/Servicio/rest/ws/modifica_carrito_compra?id_usuario=1&id_articulo=1&incremento=1&token=SCS3FbT5M2dNSesFFmc5" \ -H "Content-Type: application/json" -d '{}'

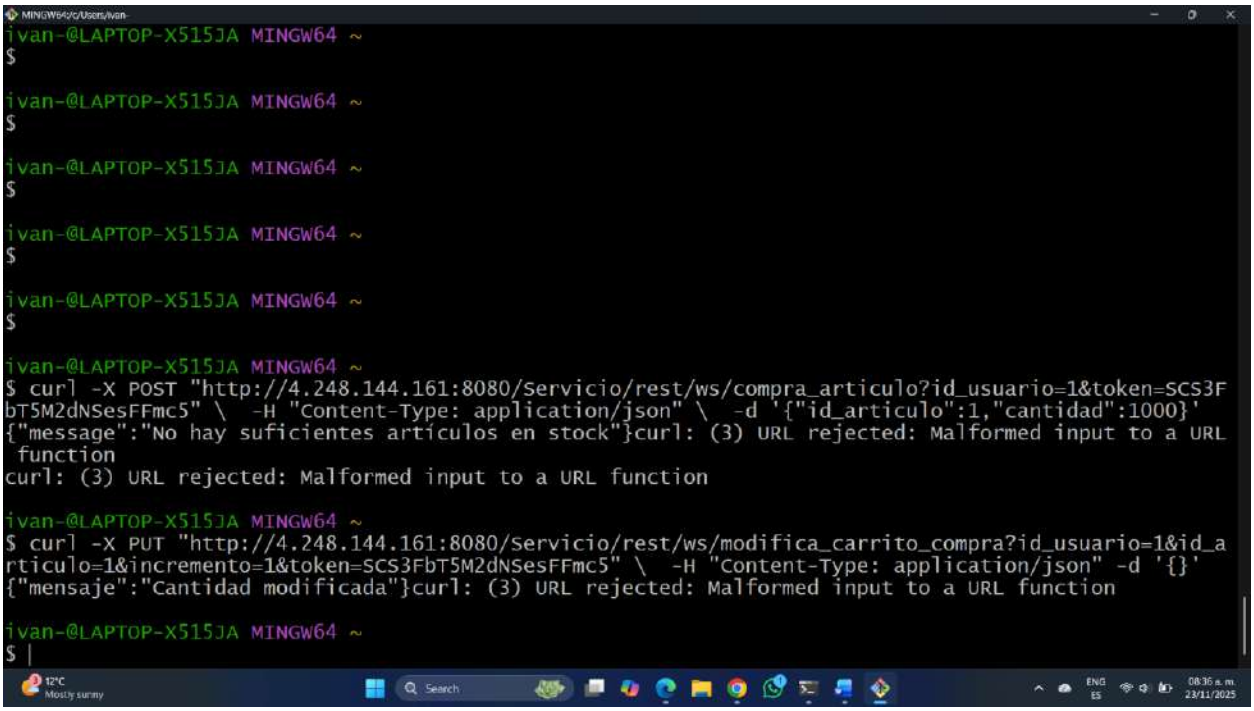



**Imagen 79: Incremento exitoso.**

### 12.2.9 consulta_carrito_compra

curl "http://4.248.144.161:8080/Servicio/rest/ws/consulta_carrito_compra?id_usuario=1&token=SCS3FbT5M2dNSesFFmc5"

Respuesta esperada (200) ejemplo:


{

"total": 142,

"items": [

{

"id_articulo": 1,

"nombre": "Mayonesa 500g",

"precio": 35.5,

```
      "cantidad": 4,
      "foto": null,
      "costo": 142
    }
  ]
}
```

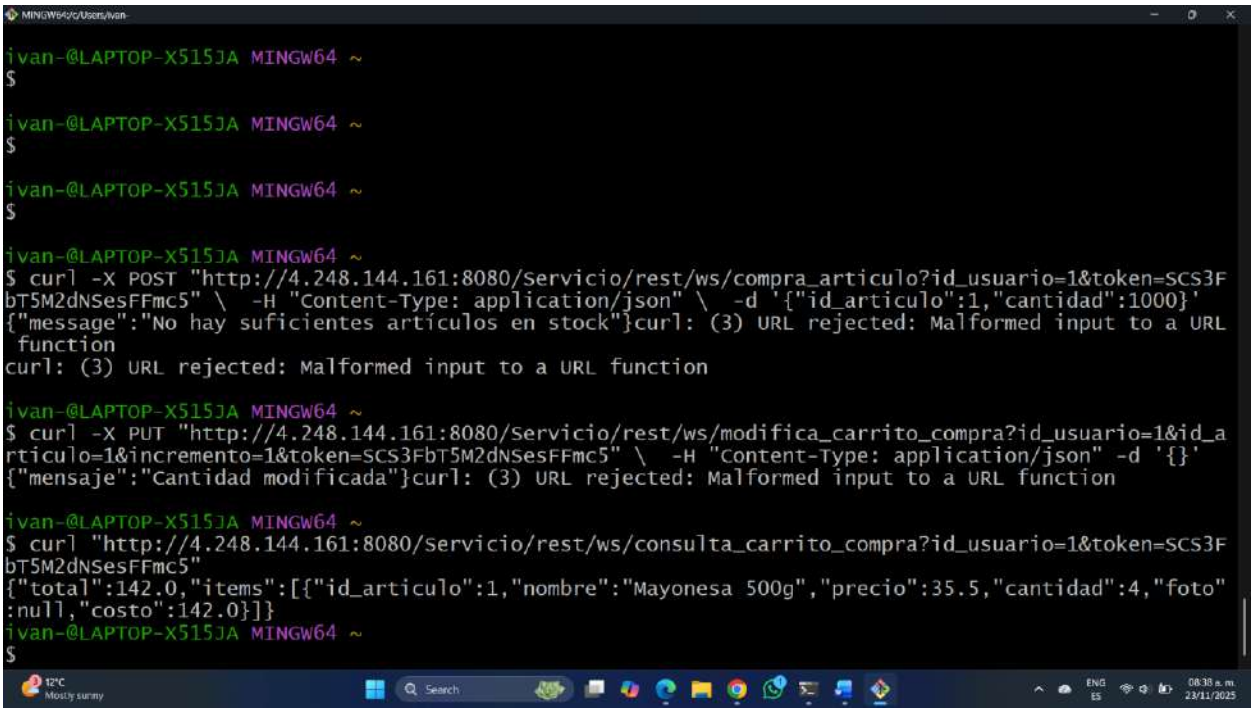




**Imagen 81: Carrito con total y items.**

### 12.2.10 elimina_articulo_carrito_compra

curl -X DELETE "http://4.248.144.161:8080/Servicio/rest/ws/elimina_articulo_carrito_compra?id_usuario=1&id_articulo=1&token=SCS3FbT5M2dNSesFFmc5"

Respuesta esperada (200):

{ "mensaje":"Artículo eliminado del carrito" }

Error (artículo ya no está en el carrito):

curl -X DELETE "http://4.248.144.161:8080/Servicio/rest/ws/elimina_articulo_carrito_compra?id_usuario=5&id_articulo=1&token=SCS3FbT5M2dNSesFFmc5"

Respuesta (400):

{ "message":"Artículo no está en el carrito" }

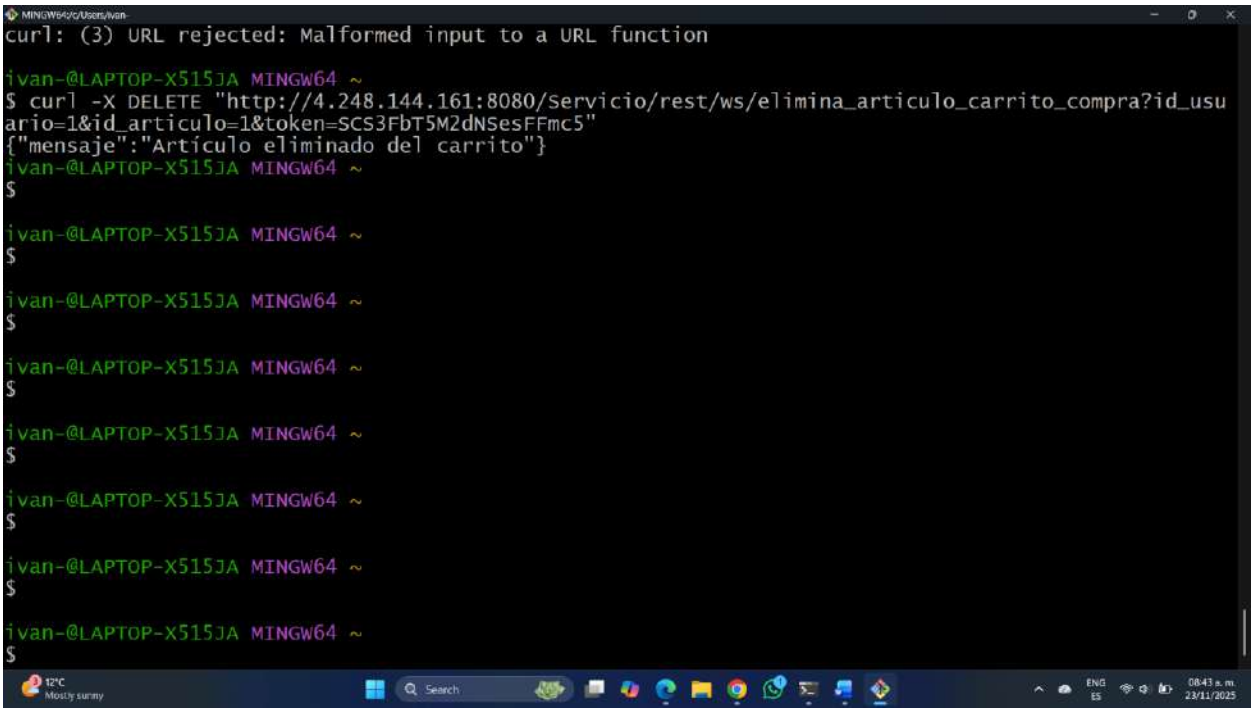



**Imagen 83: Eliminación exitosa.**

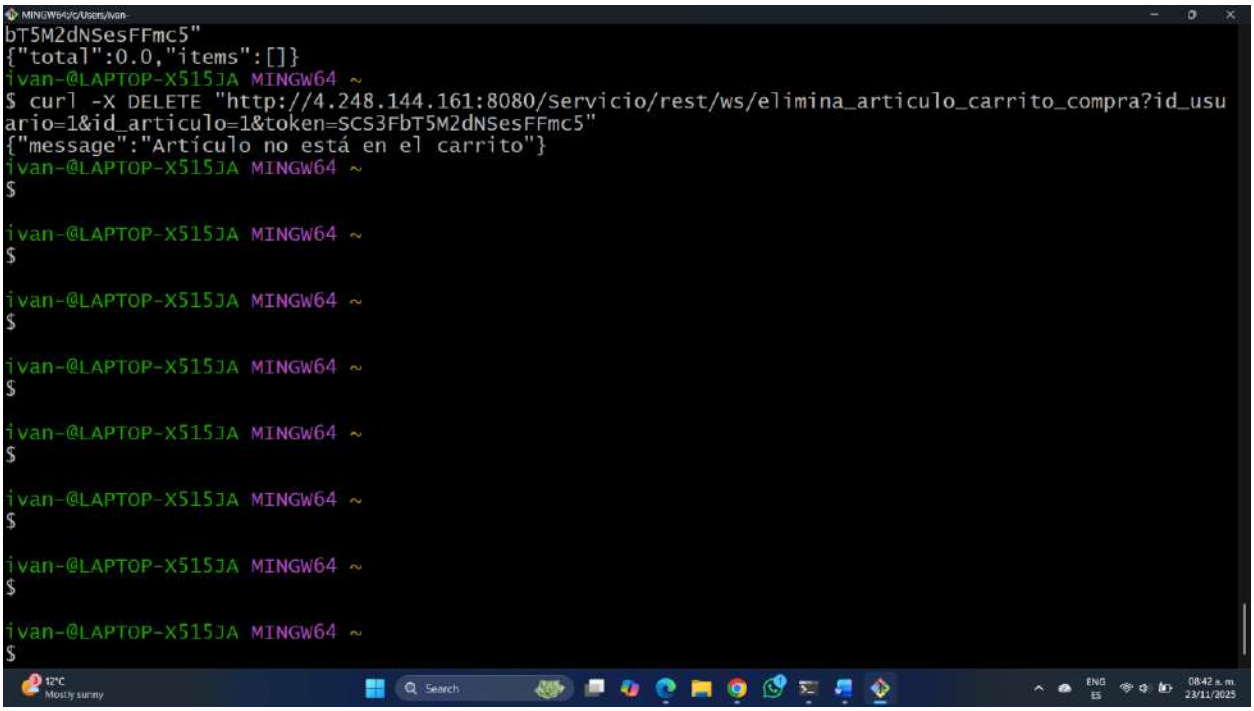



**Imagen 84: Error artículo inexistente en carrito.**

### 12.2.11 elimina_carrito_compra

curl -X DELETE "http://4.248.144.161:8080/Servicio/rest/ws/elimina_carrito_compra?id_usuario=1&token=SCS3FbT5M2dNSesFFmc5"

Respuesta esperada (200):

{ "mensaje":"Carrito eliminado" }

Consulta posterior (debería devolver total 0 e items vacíos o carrito sin elementos):

curl "http://4.248.144.161:8080/Servicio/rest/ws/consulta_carrito_compra?id_usuario=1&token=SCS3FbT5M2dNSesFFmc5"

Respuesta ejemplo:

{ "total": 0, "items": [] }

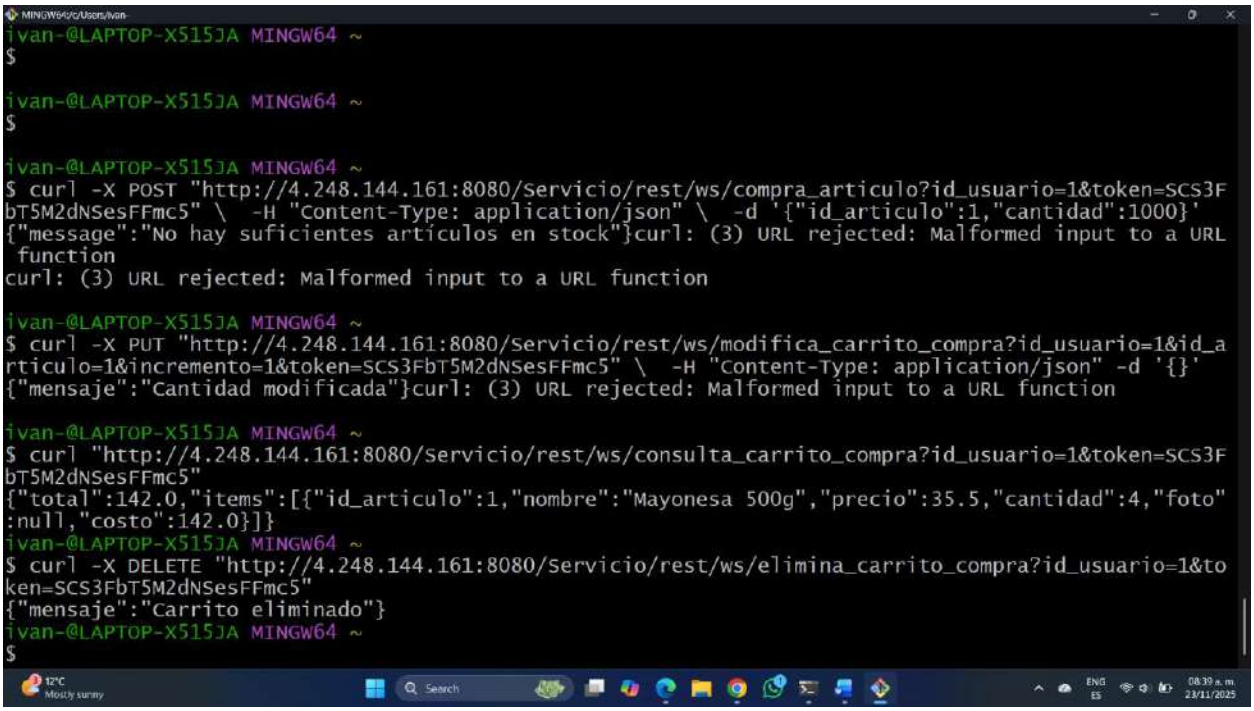



**Imagen 85: Vaciar carrito exitoso.**

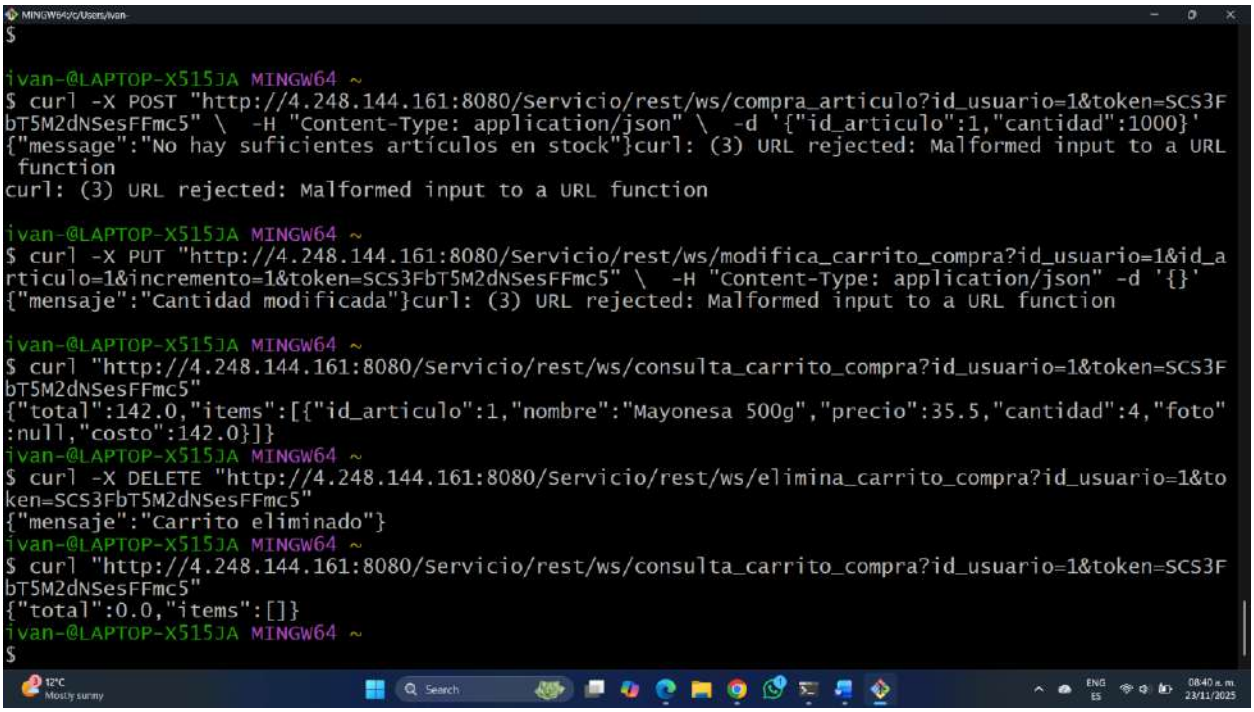



**Imagen 86: Carrito consultado vacío después de eliminación.**

## 12.3 Tabla Resumen de Pruebas

| Endpoint | Caso Éxito | Caso Error Validado | Código Éxito | Código Error |
|---|---|---|---|---|
| **login** | Credenciales correctas | Password incorrecta | 200 | 400 |
| **alta_usuario** | Registro completo | Campo obligatorio vacío | 200 | 400 |
| **consulta_usuario** | Token válido | Token inválido | 200 | 400 |
| **modifica_usuario** | Cambios válidos | Fecha nula | 200 | 400 |
| **borra_usuario** | Usuario existente | Email inexistente | 200 | 400 |
| **alta_articulo** | Datos válidos | Precio=0 | 200 | 400 |
| **consulta_articulos** | Palabra válida | Token inválido | 200 | 400 |
| **compra_articulo** | Stock suficiente | Stock insuficiente | 200 | 400 |
| **modifica_carrito_compra** | Incremento permitido | Incremento sin stock / Decremento a 0 | 200 | 400 |
| **consulta_carrito_compra** | Token válido | Token inválido | 200 | 400 |
| **elimina_articulo_carrito_compra** | Artículo presente | Artículo ausente | 200 | 400 |
| **elimina_carrito_compra** | Carrito con artículos | Token inválido | 200 | 400 |

- **12.5 Verificación de Consistencia**

Además de las respuestas JSON, se verificó el estado real de las tablas tras operaciones clave (compra, modificación, eliminación):

Ejemplo comprobación MySQL:

mysql -u usuario_mysql -p -e "SELECT * FROM carrito_compra WHERE id_usuario=5;"

mysql -u usuario_mysql -p -e "SELECT id_articulo,cantidad FROM stock;"

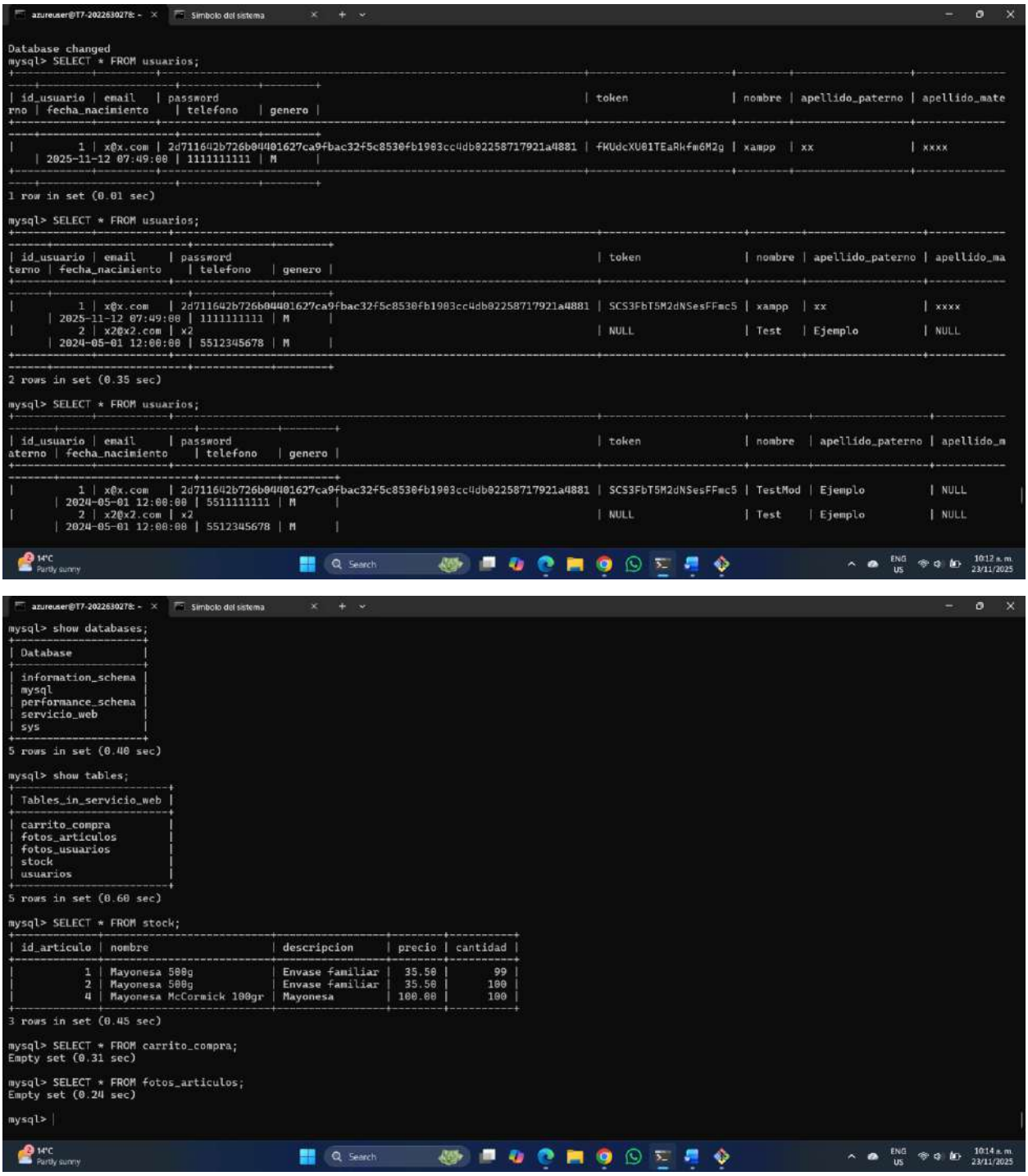



**Imagen 89: Consola MySQL mostrando contenido de carrito_compra y stock tras varias operaciones.**

# 13 Pruebas Funcionales del Front-End (Móvil)

Las pruebas funcionales del front-end se realizaron empleando un dispositivo móvil (teléfono inteligente) para verificar que cada requerimiento del apartado de comercio electrónico se ejecuta correctamente mediante la interfaz HTML/JavaScript. El objetivo principal fue constatar que las pantallas diseñadas (“Captura de artículo”, “Compra de artículos” y “Carrito de compra”) responden adecuadamente al flujo de usuario previsto: autenticación, gestión de artículos y operaciones sobre el carrito, incluyendo incrementos, decrementos, compras y eliminaciones. Se utilizó el navegador móvil (Chrome) sobre conexión Wi-Fi, accediendo directamente mediante la URL pública: http://<IP_VM>:8080/prueba.html. Las capturas registran el aspecto visual, la secuencia de pasos y los mensajes de retroalimentación (alertas, cambios instantáneos en listado y totales).

La metodología consistió en:

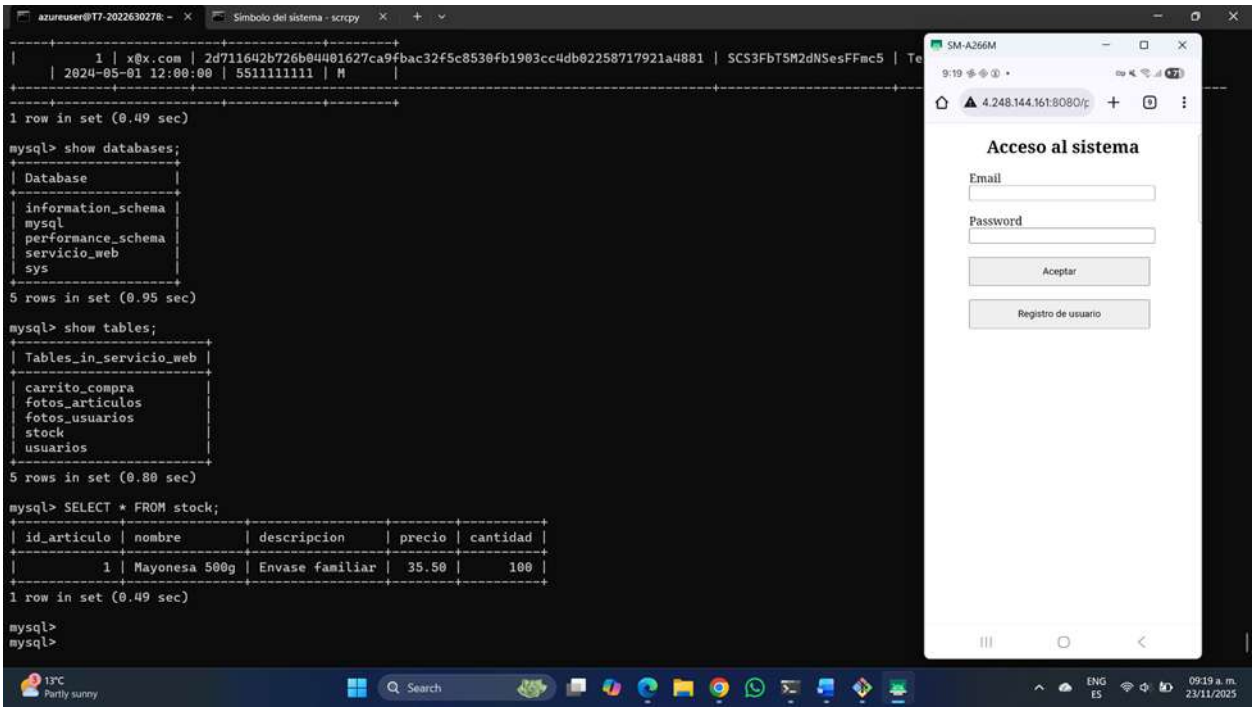
1. Preparar previamente en el back-end un usuario válido, artículos iniciales y estado limpio de carrito.
2. Ingresar al sistema desde el móvil, ejecutar cada acción funcional de manera aislada y luego en combinación (por ejemplo: buscar artículo → cambiar cantidad → comprar → verificar carrito → modificar +/- → eliminar).
3. Registrar capturas antes y después de cada evento relevante (botón presionado y resultado).
4. Validar mensajes de error esperados (stock insuficiente o decremento indebido) manipulando cantidades extremas en la pantalla de compra o carrito.
5. Confirmar la visualización correcta de imágenes (foto por defecto o foto cargada en artículos) y la actualización dinámica del total en el carrito.

Además, se comprobó la adaptabilidad mínima de la interfaz en orientación vertical (portrait), descartando problemas de desplazamiento horizontal y confirmando que el tamaño restringido (250px de ancho central) facilitó el uso táctil sin superposiciones. Aunque no se aplicó un framework de diseño responsivo avanzado, el prototipo cumple la accesibilidad básica de interacción para la tarea.

## 13.1 Dispositivo y Entorno

- Dispositivo de prueba: Teléfono inteligente.
- Navegador: Chrome móvil actualizado.
- Conectividad: Wi-Fi estable.

- URL utilizada: http://4.248.144.161:8080/prueba.html.




## 13.2 Casos de Prueba por Requerimiento Funcional

A continuación se listan los casos funcionales del front-end y las evidencias requeridas. Cada captura sugerida posee un identificador de imagen incremental para mantener coherencia con la numeración previa (continuando después de la imagen 35 de la sección anterior).

1. Registro y Login de usuario
    - Paso: Introducir email, contraseña, datos obligatorios, registrar y luego iniciar sesión.
    - Verificación: Aparición de menú principal con nuevas opciones (“Captura de artículo” y “Compra de artículos”).

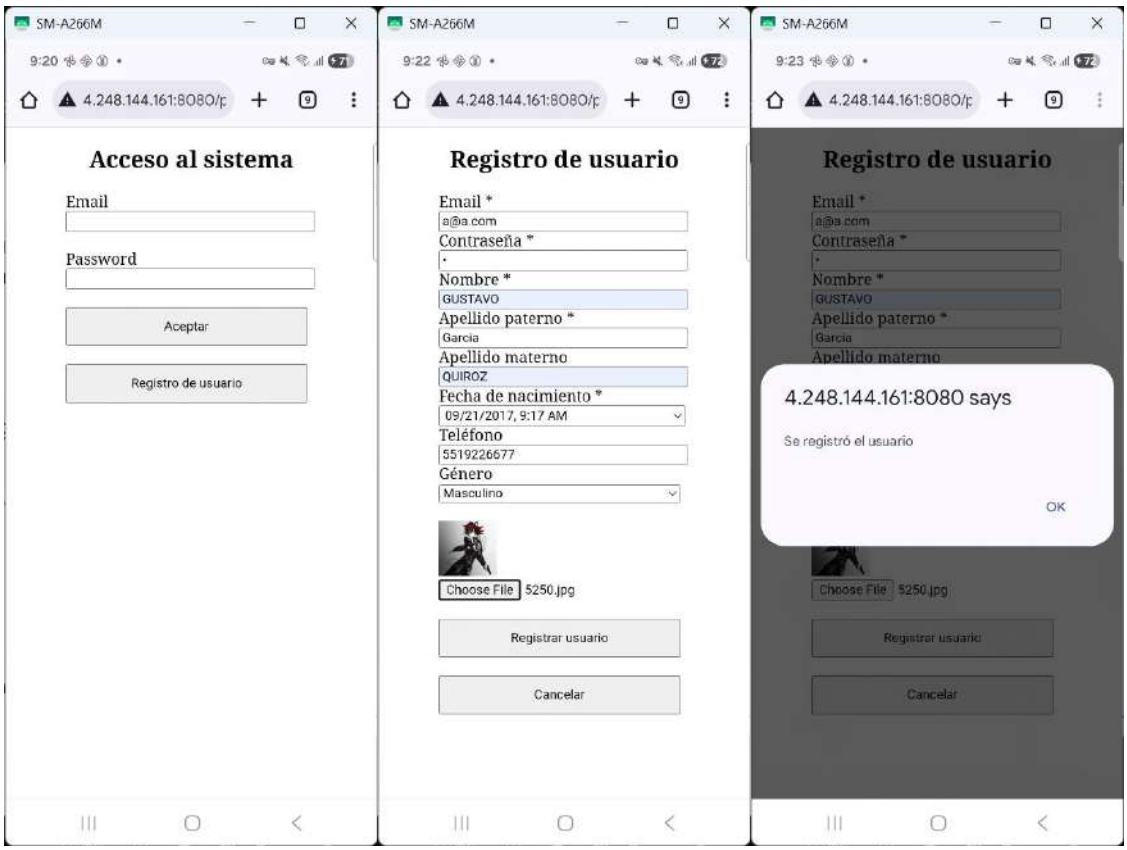



**Imagen 36: Pantalla de registro completa antes de enviar.**

**Imagen 37: Pantalla de login mostrando resultado exitoso (ingreso al menú).**

2. Acceso al Menú Principal
    - Verificación de botones: Perfil, Captura de artículo, Compra de artículos, Salir.


**Imagen 38: Menú principal en dispositivo móvil (vista clara de opciones).**

3. Captura de artículo
    - Paso: Abrir pantalla, llenar nombre, descripción, precio, cantidad, cargar foto opcional y guardar.
    - Resultado: Alerta de confirmación y retorno a menú (opcional).


**Imagen 39: Formulario “Captura de artículo” completo antes de presionar “Guardar artículo”.**

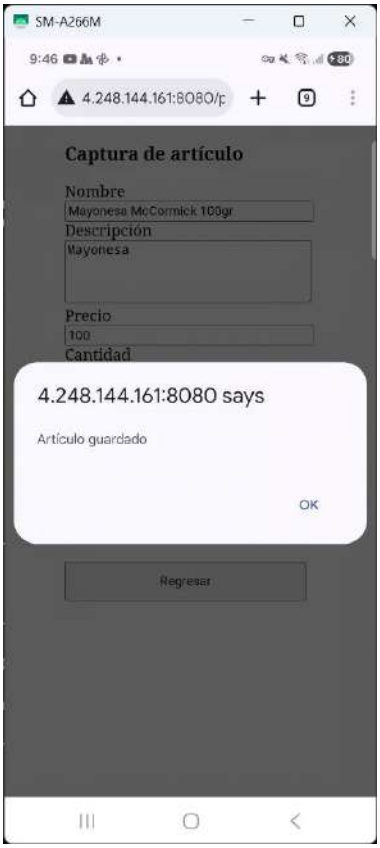



**Imagen 40: Mensaje de confirmación (alert) o retorno al menú tras alta.**

4. Búsqueda de artículos
    - Paso: Ingresar palabra clave y ejecutar búsqueda.
    - Verificación: Listado de resultados con miniatura, nombre, descripción, precio, controles de cantidad y compra.

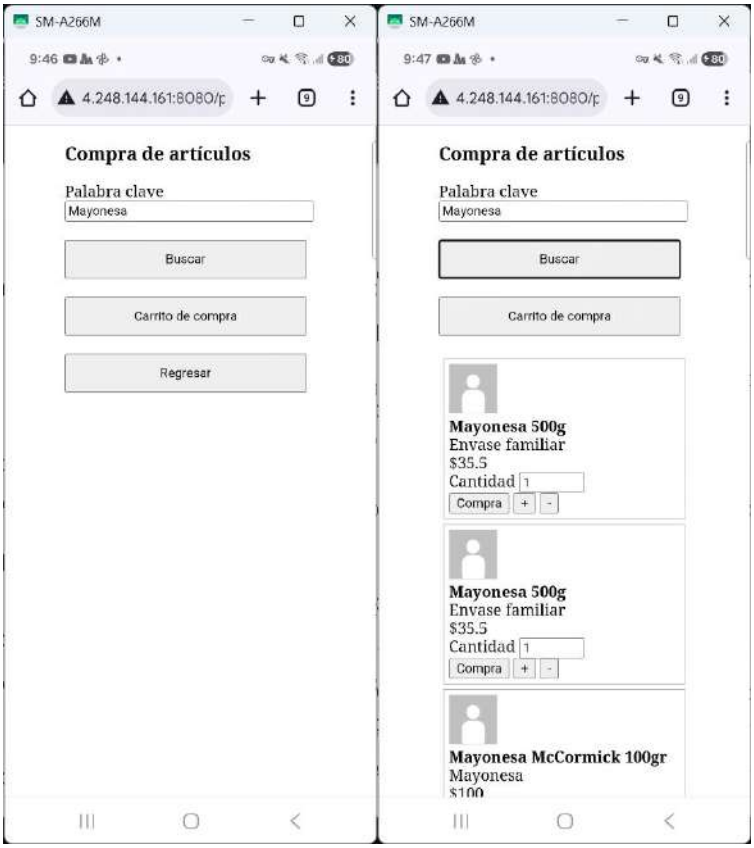



**Imagen 41: Pantalla “Compra de artículos” con resultados tras la búsqueda.**

5. Ajuste de cantidad previo a compra
    - Paso: Usar botones +/- o campo numérico para cambiar cantidad inicial (por defecto 1).


**Imagen 42: Ejemplo de artículo con cantidad modificada (campo numérico actualizado).**

6. Compra de artículo
    - Paso: Presionar “Compra” (cantidad > 0).
    - Resultado: Confirmación y actualización interna del stock (no visible directamente pero reflejada en futuras operaciones).

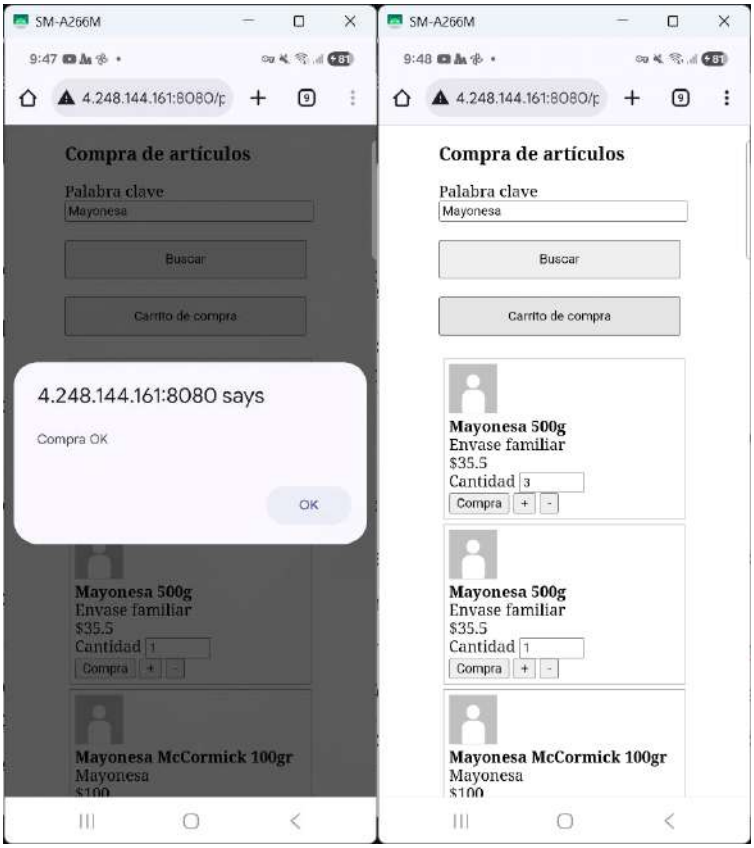



**Imagen 43: Alerta de compra exitosa (mensaje de confirmación).**

7. Visualización del carrito de compra
    - Paso: Presionar botón “Carrito de compra”.
    - Verificación: Lista de artículos con miniatura, precio, cantidad, costo (cantidad × precio) y total acumulado.


**Imagen 44: Pantalla “Artículos en el carrito” mostrando uno o más artículos.**

8. Incremento de cantidad dentro del carrito
    - Paso: Botón + en un artículo del carrito (llama a modifica_carrito_compra).
    - Resultado: Cantidad y costo actualizados, total recalculado.


**Imagen 45: Carrito después de incrementar la cantidad (resaltando cambio numérico).**

9. Decremento de cantidad dentro del carrito
    - Paso: Botón – en un artículo (sin llegar a invalidar las reglas, es decir no provocar mensaje de error).
    - Resultado: Cantidad disminuida, costo recalculado.


**Imagen 46: Carrito tras decremento válido de cantidad.**

10. Decremento invalidado (opcional para error)
    - Paso: Intentar reducir la cantidad por debajo del mínimo permitido (según lógica del prototipo).
    - Resultado: Alerta con mensaje “No hay más artículos en el carrito”.


**Imagen 47: Alerta de error de decremento.**

11. Eliminación de artículo específico
    - Paso: Presionar “Eliminar artículo” sobre uno de los ítems.
    - Resultado: Ítem desaparece y el total se ajusta.


**Imagen 48: Carrito tras eliminar un artículo (lista reducida).**

12. Eliminación total del carrito
    - Paso: Press “Eliminar carrito”, confirmar.
    - Resultado: Lista vacía y total igual a 0.

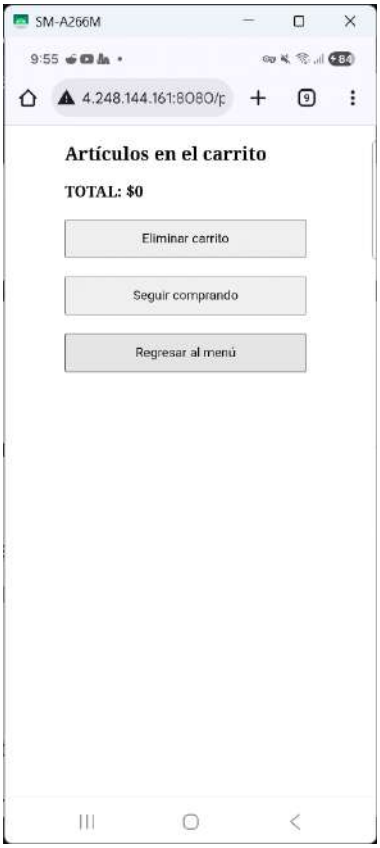



**Imagen 49: Carrito vacío después de la operación.**

13. Regresar a la pantalla de compra
    - Paso: “Seguir comprando” desde el carrito.
    - Resultado: Retorno a la pantalla de búsqueda sin perder token.


**Imagen 50: Pantalla “Compra de artículos” reabierta tras navegar desde carrito.**

14. Carga y visualización de imagen de artículo
    - Paso: Capturar nuevo artículo con foto; luego buscar y verificar miniatura en resultados y en carrito.


**Imagen 51: Resultado de búsqueda mostrando artículo con foto personalizada.**


**Imagen 52: Carrito mostrando la miniatura del artículo comprado.**

15. Flujo completo encadenado (opcional para evidencia global)
    - Paso: Registro → Login → Captura artículo → Compra → Carrito → Modificar → Eliminar.

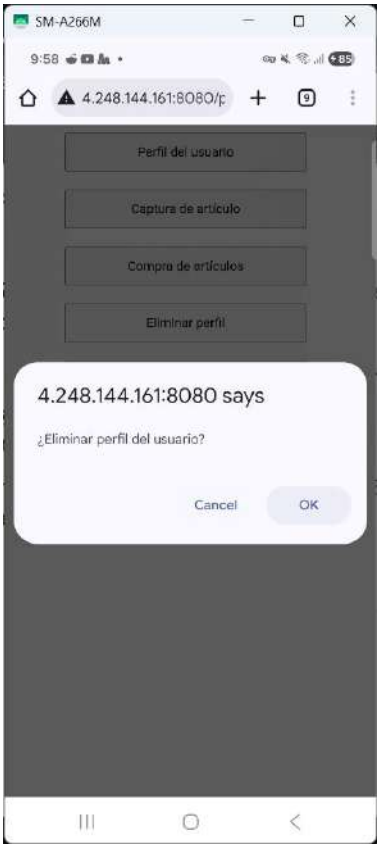



**Imagen 53: Collage o secuencia final demostrando el recorrido completo (puede ser un montaje o captura extendida).**

## 13.3 Validación de Mensajes y Errores

Se forzaron condiciones para generar mensajes de error:

- Stock insuficiente (compra de cantidad exagerada) desde pantalla de compra: alerta “No hay suficientes artículos en stock”.
- Decremento indebido en carrito: alerta “No hay más artículos en el carrito”.
- Token incorrecto (simulación manual en código no mostrada en interfaz común): interfaz muestra alert genérico de acceso denegado cuando se intercepta respuesta 400 con dicho mensaje.

Estos mensajes confirman que el front-end interpreta correctamente la estructura uniforme de errores JSON retornados por el back-end.

# 14 Enlace del chat de la IA generativa

Enlace: https://github.com/copilot/share/80155284-4a44-8c87-a112-324f043a68d3

# 15 Conclusiones

El desarrollo del prototipo de comercio electrónico logró integrar de forma coherente las funcionalidades solicitadas: alta y consulta de artículos, manejo completo del carrito de compra (agregar, modificar, eliminar y vaciar), y ampliación del servicio REST previo de usuarios sin romper compatibilidad con la arquitectura inicial de la Tarea 2. La adopción de transacciones en las operaciones críticas aseguró la consistencia entre el inventario (stock) y las reservas temporales (carrito), evitando desbalances comunes en escenarios de compra simultánea. La simplicidad estructural (un único WAR y un front-end HTML/JavaScript plano) permitió un aprendizaje enfocado en los principios esenciales de servicios REST, persistencia relacional y comunicación cliente-servidor.

El uso de GitHub Copilot aceleró la generación de plantillas repetitivas (endpoints, POJOs y patrones de manejo de errores), pero la validación manual fue imprescindible para garantizar la correcta aplicación de reglas de negocio y el empleo adecuado de transacciones y bloqueos (SELECT ... FOR UPDATE). El enfoque de seguridad aplicado (token + hash SHA-256 en cliente) es suficiente a nivel académico, aunque se identificaron varias oportunidades de robustecimiento para una versión productiva (expiración de token, hashing con sal adaptativa, HTTPS obligatorio, control de roles, rate limiting).

En el front-end se confirmó que cada requerimiento funcional se cumple de forma clara desde un dispositivo móvil: la navegación por pantallas, la captura de artículos, la compra y el ajuste de cantidades en el carrito resultaron intuitivos y consistentes. Las pruebas con curl reforzaron la verificación de cada método del back-end, mostrando respuestas uniformes (códigos 200 y 400) y mensajes de error explícitos. Esta dualidad de evidencia (capturas móviles + consola) robustece el reporte y cumple con los lineamientos de entrega.

El prototipo cumple los objetivos académicos y deja un terreno fértil para modularizar componentes, integrar capas adicionales (historial de pedidos, pagos simulados) y elevar el estándar de seguridad. Se alcanzó así un equilibrio entre simplicidad y funcionalidad

que facilita la comprensión del ciclo completo “usuario autenticado → artículo → carrito → consistencia transaccional”.

# 16 Referencias (Formato IEEE)


[1] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Ph.D. dissertation, Univ. California, Irvine, 2000.

[2] Oracle, “Java Platform Standard Edition 17 API Specification – Class SecureRandom,” 2025. [Online]. Available: https://docs.oracle.com/

[3] ISO/IEC 9075-1:2016, “Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework).”

[4] Oracle, “InnoDB Storage Engine,” MySQL 8.0 Reference Manual, 2025. [Online]. Available: https://dev.mysql.com/doc/

[5] Apache Software Foundation, “Apache Tomcat 10 Documentation,” 2025. [Online]. Available: https://tomcat.apache.org/

[6] Eclipse Foundation, “Jakarta RESTful Web Services (JAX-RS) Specification,” 2025. [Online]. Available: https://jakarta.ee/specifications/

[7] FasterXML, “Jackson Databind and Core Libraries,” 2025. [Online]. Available: https://github.com/FasterXML/jackson

[8] GitHub, “GitHub Copilot: AI Pair Programmer,” 2025. [Online]. Available: https://github.com/features/copilot

[9] NIST, “Secure Hash Standard (SHS),” FIPS PUB 180-4, Aug. 2015.

[10] W3C, “File API,” W3C Recommendation, 2015. [Online]. Available: https://www.w3.org/TR/FileAPI/

[11] W3C, “Web Cryptography API,” W3C Recommendation, 2017. [Online]. Available: https://www.w3.org/TR/WebCryptoAPI/

[12] OWASP Foundation, “OWASP Top 10: Web Application Security Risks,” 2023. [Online]. Available: https://owasp.org/

[13] D. Thomas et al., “Bcrypt Password Hashing,” OpenBSD Project Documentation, 2025. [Online]. Available: https://man.openbsd.org/

[14] P. Wuille et al., “Argon2: The Memory-Hard Function for Password Hashing,” RFC Draft (argon2), 2025.

[15] Google, “Fetch API Living Standard,” WHATWG, 2025. [Online]. Available: https://fetch.spec.whatwg.org/

[16] Mozilla, “Content Security Policy (CSP) Guide,” MDN Web Docs, 2025. [Online]. Available: https://developer.mozilla.org/

[17] Git, “Distributed Version Control System,” Git Project, 2025. [Online]. Available: https://git-scm.com/

[18] Azure, “Azure Virtual Machines Documentation,” Microsoft Learn, 2025. [Online]. Available: https://learn.microsoft.com/azure/virtual-machines/