Instituto Politécnico Nacional

Escuela Superior de Computo

Sistemas Distribuidos

Tarea 7. Prototipo de un sistema de comercio electrónico utilizando un servicio web REST para Tomcat

Nombre del alumno:

García Quiroz Gustavo Ivan

Grupo: 7CV4

Nombre del profesor: Guerrero Carlos Pineda

Fecha de entrega: 29/11/2025

ÍNDICE

1 Resumen

Este documento presenta el desarrollo de un prototipo de sistema de comercio electrónico sencillo, compuesto por un servicio web REST (back-end) desplegado en Apache Tomcat y una aplicación web HTML/JavaScript (front-end) ejecutada en un navegador móvil. El prototipo extiende la funcionalidad previa de gestión de usuarios (implementada en la Tarea 2) para incorporar operaciones de administración de artículos (alta y consulta), manipulación de un carrito de compra y control de transacciones sobre una base de datos MySQL alojada en una máquina virtual de Azure (VM nombrada conforme a la boleta del alumno).

Se integran mecanismos básicos de autenticación mediante tokens y manejo de contraseñas con hash SHA-256 (calculado en el cliente). La arquitectura sigue un enfoque multicapa mínimo y agrupa las operaciones críticas (compra, modificación y eliminación) dentro de transacciones para garantizar consistencia entre el stock y el carrito. El resultado es una base sólida para evolucionar hacia soluciones más completas (pasarela de pagos, catálogo avanzado o refactor a frameworks modernos).

La herramienta de IA GitHub Copilot se utilizó como apoyo puntual en la generación de fragmentos de código y en la estructuración acelerada del back-end y partes del front-end, manteniendo supervisión humana para validación funcional y semántica.

> NOTA: La plataforma de Moodle no permitió subir el reporte en buena calidad debido al límite de 2 MB por archivo. Sugiero revisar el siguiente documento de Google drive con el reporte en buena calidad: https://drive.google.com/file/d/12RmsPrHCOPWLXeO1s6OF_BQuNRVdosLM/view?usp=sharing

2 Introducción

La Tarea 7 se orienta a la creación de un prototipo de comercio electrónico elemental que permita la administración de artículos y la simulación del proceso de compra mediante un carrito asociado a cada usuario registrado. Este trabajo parte del avance logrado en la Tarea 2, donde ya se disponía de un servicio REST para operaciones CRUD de usuarios y manejo de fotografías. Sobre esa base, se añadieron nuevas tablas (stock, fotos_articulos y carrito_compra) y se ampliaron los endpoints del servicio para cubrir las acciones de alta de artículos, búsqueda, compra, modificación y eliminación de elementos del carrito.

El contexto tecnológico se centra en:

1. Apache Tomcat como contenedor de servlets y punto de despliegue del servicio REST con Jersey + Jackson.
2. MySQL como gestor de persistencia, reforzado mediante el uso de transacciones para mantener la coherencia entre las cantidades del stock y las operaciones del carrito.
3. HTML + JavaScript (sin framework obligatorio) para el front-end responsive orientado a pruebas en dispositivo móvil.
4. Infraestructura en Azure for Students donde se aprovisiona la VM (nombre: T7-2022630278) en región de Canadá, siguiendo los lineamientos del curso.

Durante la implementación se empleó la inteligencia artificial de GitHub Copilot como herramienta de asistencia en la generación de código repetitivo, puntos de refactor y sugerencias de estructura, manteniéndose la revisión manual para garantizar:

- Correcta adecuación de cada endpoint a las reglas de negocio.
- Validez de las transacciones y manejo apropiado de errores HTTP (códigos 200 y 400).
- Coherencia entre las capturas del front-end y las pruebas unitarias mediante curl.

La arquitectura favorece una separación clara entre responsabilidades del cliente y del servidor: el front-end gestiona captura, interacción y presentación, mientras el back-end expone operaciones REST atómicas o transaccionales. El token de sesión (generado en el login) actúa como llave para la autorización posterior. La protección de contraseñas se realiza calculando el hash SHA-256 en el cliente antes del envío, reduciendo el riesgo de exposición de texto plano, aunque se reconocen mejoras futuras (salting, almacenamiento más robusto).

3 Objetivos

3.1 Objetivo general

Implementar un prototipo funcional de comercio electrónico con gestión de usuarios, artículos y carrito de compra, empleando un servicio REST sobre Tomcat y una interfaz HTML/JavaScript accesible desde móvil.

3.2 Objetivos específicos

- Integrar nuevas tablas (stock, fotos_articulos, carrito_compra) con llaves y restricciones adecuadas.
- Desarrollar endpoints REST que cubran alta, consulta y operaciones sobre el carrito, respetando reglas de negocio y validaciones.
- Asegurar integridad mediante transacciones en operaciones críticas (compra, modificación, eliminación).
- Adaptar el front-end para incluir pantallas de captura de artículo, compra y carrito.
- Probar cada método con curl (back-end) y cada requerimiento en dispositivo móvil (front-end).
- Incorporar autenticación basada en token y hash de contraseñas previo al envío.
- Documentar de forma clara los requerimientos funcionales y no funcionales.

4 Requerimientos

Esta sección detalla los requerimientos de la solución, organizados en tres grupos: funcionales del back-end, funcionales del front-end y no funcionales. Se parte del enunciado oficial, adaptando el lenguaje para claridad en el reporte.

4.1 Requerimientos funcionales del back-end

El servicio REST expone métodos que operan sobre usuarios (heredados de la Tarea 2) y sobre las nuevas entidades del comercio electrónico. Las operaciones nuevas necesitan validar el acceso mediante el token emitido en el login. Además, aplican reglas de negocio sobre inventarios y cantidades del carrito.

Principales requerimientos (nuevos para la Tarea 7):

- alta_articulo: Inserta un artículo (nombre, descripción, precio, cantidad y fotografía) en la tabla stock y opcionalmente la foto en fotos_articulos.
- consulta_articulos: Recupera artículos cuyo nombre o descripción contengan una palabra clave (búsqueda con LIKE). Regresa id_articulo, foto, nombre, descripción y precio.
- compra_articulo: Agrega un artículo al carrito del usuario si hay existencias suficientes; descuenta la cantidad del stock en una transacción (INSERT/UPDATE + UPDATE).
    - Si la cantidad solicitada supera existencias se retorna error (400) con mensaje “No hay suficientes artículos en stock”.
- elimina_articulo_carrito_compra: Remueve un artículo del carrito y regresa las unidades al stock en una transacción.
- elimina_carrito_compra: Vacía por completo el carrito del usuario, regresando cada cantidad al stock dentro de una transacción.
- modifica_carrito_compra: Ajusta (+1 / -1) la cantidad comprada de un artículo en el carrito, sincronizando stock y validando límites (no decrementar por debajo de 1; no incrementar sin existencias).

- consulta_carrito_compra: Devuelve los artículos en el carrito con foto, nombre, cantidad, precio y costo, además del total acumulado.

Cada método que modifica tablas relacionadas (stock y carrito_compra) utiliza transacciones para evitar inconsistencias. Los errores de acceso (token inválido o inexistente) retornan código 400 con objeto de error.

Lista complementaria (heredadas de la Tarea 2 y aún relevantes):

- login
- alta_usuario
- consulta_usuario
- modifica_usuario
- borra_usuario

4.2 Requerimientos funcionales del front-end

La interfaz HTML/JavaScript debe permitir un flujo claro y accesible desde un dispositivo móvil (layout simple, controles táctiles). Se agregan dos opciones al menú principal: “Captura de artículo” y “Compra de artículos”, además de la pantalla asociada al carrito.

Elementos clave:

- Pantalla “Captura de artículo”: Formulario para nombre, descripción, precio, cantidad y foto. Invoca alta_articulo.
- Pantalla “Compra de artículos”: Campo para palabra clave; muestra resultados con miniatura, nombre, descripción, precio y controles (cantidad, botones +/-, botón “Compra”).
- Botón “Carrito de compra”: Muestra pantalla “Artículos en el carrito” con cada artículo, su cantidad, precio individual, costo (cantidad × precio) y total general.
- Controles para modificar cantidad (+/-) sobre artículos ya en el carrito (modifica_carrito_compra).

- Botón “Eliminar artículo” para retirar un artículo del carrito (elimina_articulo_carrito_compra).
- Botón “Eliminar carrito” para vaciarlo por completo (elimina_carrito_compra).
- Botón “Seguir comprando” para volver a la pantalla de búsqueda.
- Manejo de autenticación: login previo y token persistente en sesión.
- Hash SHA-256 de contraseñas antes de enviar (ya existente de la Tarea 2).

4.3 Requerimientos no funcionales

Estos aseguran condiciones estructurales y de calidad:

Parámetros de infraestructura y despliegue:

- Máquina virtual Azure creada desde la imagen de la Tarea 2.
- Nombre estricto de la VM: T7-2022630278 (relacionado con la boleta).
- Región Canadá (Azure for Students).
- Exposición del puerto 8080 para acceso al front-end y pruebas con curl.

Base de datos y consistencia:

- Uso de transacciones para operaciones que combinan cambios en stock y carrito (compra, modificación de cantidad, eliminación de artículos, vaciado completo).
- Integridad referencial mediante llaves foráneas: fotos_articulos → stock, carrito_compra → usuarios y stock.
- Índice único (PRIMARY KEY compuesta) en carrito_compra (id_usuario, id_articulo) que evita duplicados del mismo artículo dentro del mismo carrito.

Calidad y mantenibilidad:

- Código Java organizado en POJOs (Usuario, Articulo, CarritoItem, HuboError).
- Serialización JSON consistente vía Jackson.
- Separación clara de responsabilidades (front-end ligero, back-end transaccional).

- Manejo de errores con códigos HTTP diferenciados (200 éxito / 400 error de lógica o acceso).

Seguridad básica:

- Token de sesión emitido en login.
- Hash de contraseñas en el cliente antes de envío (SHA-256).
- Validaciones de campos obligatorios y rangos (precio > 0, cantidad >= 0).

Pruebas y evidencia:

- Capturas de pantalla en dispositivo móvil para cada requerimiento funcional front-end.
- Salidas de curl por cada endpoint del back-end.

Aspectos potenciales de mejora (identificados, no obligatorios en esta fase):

- Paginación de resultados de consulta_articulos.
- Salt y almacenamiento de contraseñas con algoritmo robusto en servidor.
- Rol de administrador para separación de alta de artículos vs. compra.
- Cache o CDN para imágenes de productos.

5 Arquitectura del Sistema

La arquitectura del prototipo se compone de cuatro capas principales que interactúan de forma secuencial y transaccional: (1) Cliente Web móvil (HTML/JavaScript), (2) Servicio REST en Tomcat (Jersey + Jackson), (3) Base de Datos MySQL y (4) Infraestructura de despliegue en Azure (VM nombrada T7-2022630278). El flujo general parte de la autenticación (emisión de token), continúa con operaciones de artículo y carrito, y finaliza en la persistencia consistente de los datos.

La solución mantiene un enfoque minimalista de micro-arquitectura: cada endpoint encapsula una operación concreta y, cuando es crítico, agrupa múltiples sentencias SQL dentro de una transacción explícita (compra, modificación del carrito, eliminación de artículos, vaciado completo). El front-end es deliberadamente ligero para evidenciar la interacción y facilitar pruebas móviles.

5.1 Visión global por capas

Capa	Tecnología	Responsabilidad principal	Interacción destacada
Presentación	HTML + JavaScript (WSClient.js)	Captura de datos (usuarios, artículos, búsquedas, carrito), renderización y eventos táctiles	Emite peticiones HTTP (GET/POST/PUT/DELETE) al servicio REST
Lógica de negocio / API	Java (Servlet Container: Tomcat + Jersey)	Validación de parámetros, verificación de acceso (token), reglas de stock y carrito, control transaccional	Mapea JSON ↔ POJOs y coordina transacciones hacia la BD
Persistencia	MySQL	Almacenamiento de usuarios, artículos, fotos y carrito; integridad referencial	Recibe consultas y actualizaciones dentro o fuera de transacciones
Infraestructura	VM Azure (Linux/Windows según imagen base)	Hosting de Tomcat y MySQL, exposición de puerto 8080, seguridad básica (NSG)	Provee disponibilidad y acceso remoto para front-end y pruebas curl

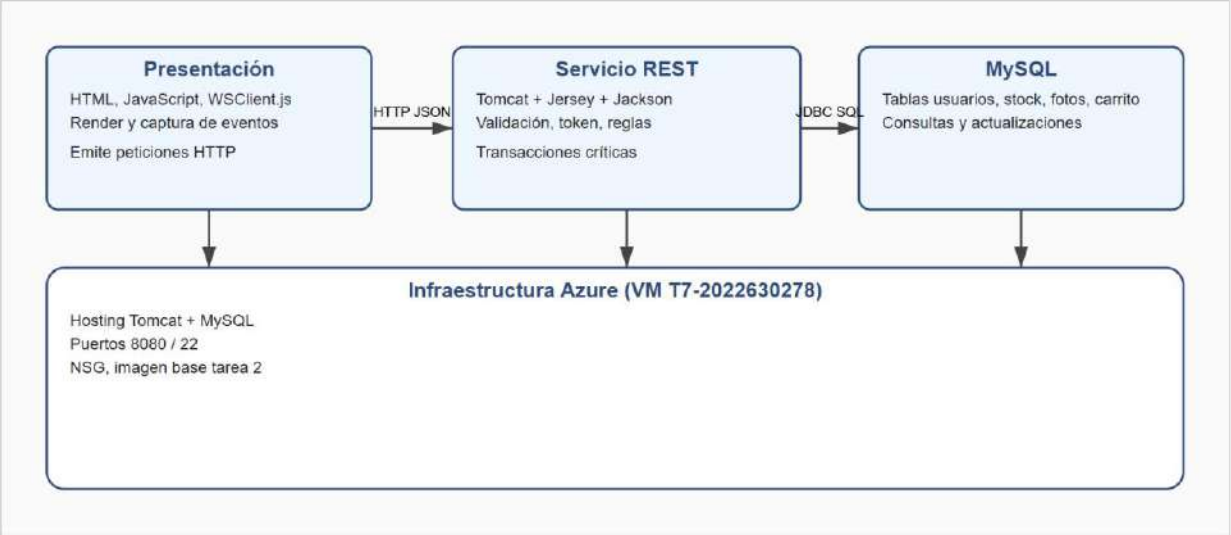


Imagen 6: Diagrama de arquitectura por capas (Móvil / Tomcat REST / MySQL / VM).

5.2 Componentes principales

- Front-end (prueba.html): Gestiona pantallas de Login, Perfil, Captura de artículo, Compra de artículos y Carrito. Invoca WSClient.js para emitir peticiones uniformes.
- WSClient.js: Wrapper simplificado sobre fetch que codifica parámetros y maneja JSON para métodos GET, POST, PUT y DELETE.
- Servicio.java: Clase central con endpoints REST; inicializa DataSource vía JNDI (context.xml). Orquesta verificación de token, queries y transacciones.
- Clases POJO (Usuario, Articulo, CarritoItem, HuboError): Definen estructura de datos, favoreciendo serialización/deserialización con Jackson.
- context.xml: Declara recurso de datasource para conexión a MySQL.
- web.xml: Configura servlet de Jersey, paquete “servicio” y soporte para JacksonFeature.

- Base de Datos: Tablas usuarios, fotos_usuarios, stock, fotos_articulos, carrito_compra (relaciones y llaves primarias/foráneas).

5.3 Flujo de autenticación y autorización

1. El usuario ingresa email y contraseña (plaintext sólo en el navegador).
2. Se calcula hash SHA-256 en el cliente antes del envío.
3. El endpoint login valida la combinación email + password en la BD.
4. Se genera un token aleatorio (20 caracteres) y se almacena en usuarios.token.
5. El front-end conserva token e id_usuario en memoria local (variables JS).
6. Cada operación protegida (artículos / carrito) envía id_usuario y token para verificación.

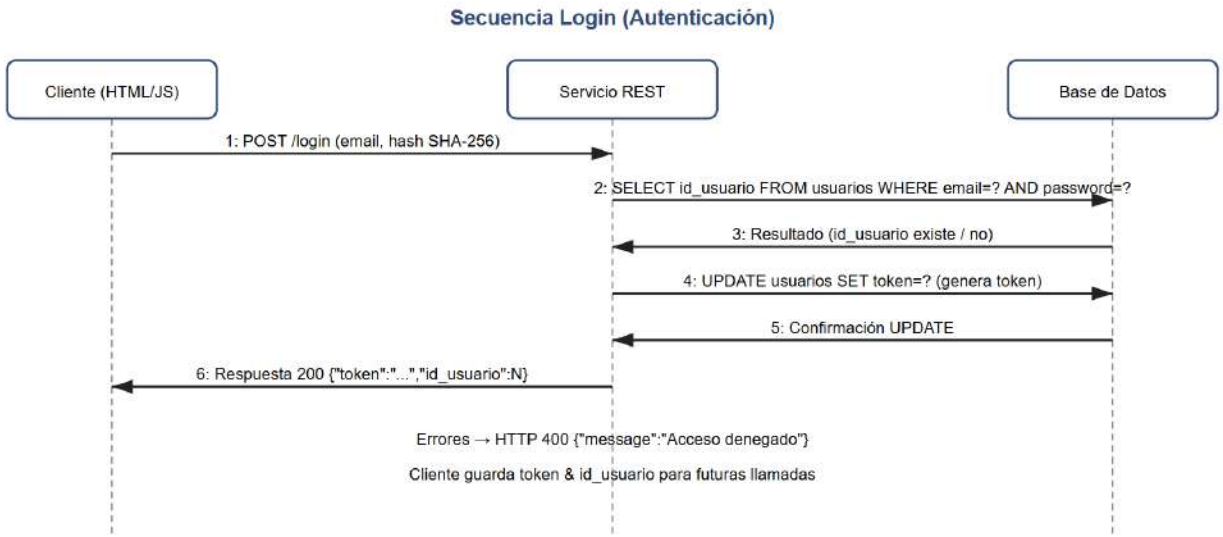


Imagen 7: Diagrama de secuencia del login (Cliente → Servicio → BD → Servicio → Cliente).

5.4 Flujo de compra y consistencia transaccional

La operación de compra (compra_articulo) ejemplifica el patrón transaccional:

1. Verificación de token e id_usuario.
2. Bloqueo (SELECT ... FOR UPDATE) o lectura directa del stock para confirmar existencias suficientes.
3. Inserción o actualización del carrito (carrito_compra).
4. Descuento de la cantidad en stock.
5. Commit; si ocurre error, rollback y respuesta HTTP 400 con mensaje detallado.

Igual mecanismo aplica en:

- modifica_carrito_compra: Ajuste (+/-) cantidad con control de límites y actualización inversa del stock.
- elimina_articulo_carrito_compra: Recupera unidades al stock y borra el registro del carrito.
- elimina_carrito_compra: Itera sobre todos los registros y devuelve las cantidades correspondientes al stock.

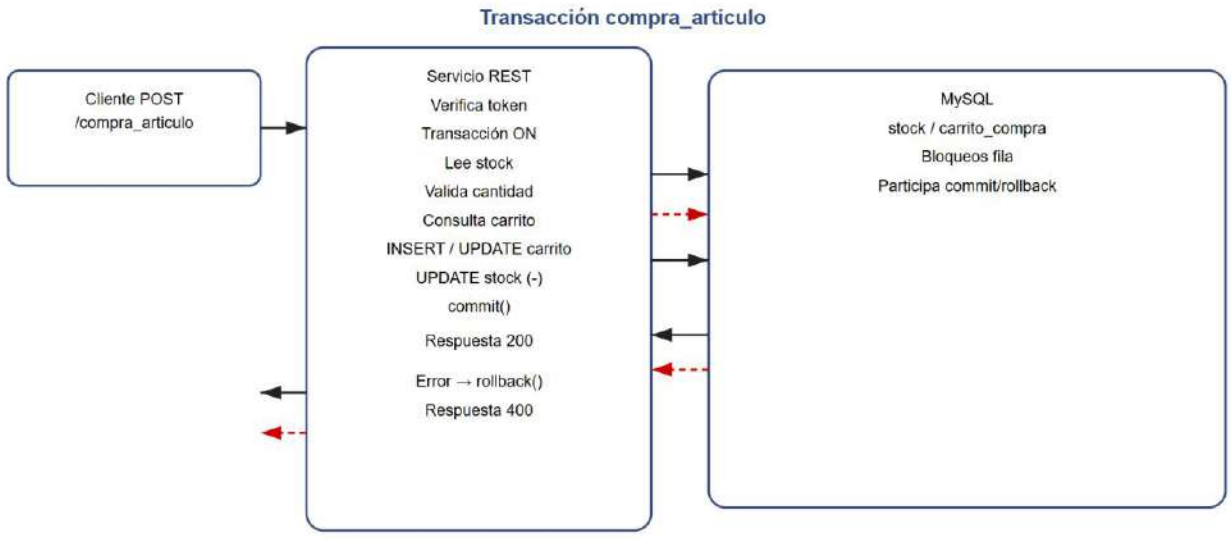


Imagen 8: Diagrama de transacción de compra (Stock ↔ Carrito ↔ Commit/Rollback).

5.5 Modelo lógico y relaciones (resumen conceptual)

Tabla	Llave primaria	Relaciones	Descripción
usuarios	id_usuario	fotos_usuarios (1:N), carrito_compra (1:N)	Información y credenciales (hash + token)
fotos_usuarios	id_foto	FK → usuarios.id_usuario	Foto opcional de perfil
stock	id_articulo	fotos_articulos (1:N), carrito_compra (N:M vía usuarios)	Inventario y datos de cada artículo
fotos_articulos	id_foto	FK → stock.id_articulo	Fotografía del artículo
carrito_compra	(id_usuario, id_articulo)	FK → usuarios, FK → stock	Asociación de artículos a usuario con cantidad

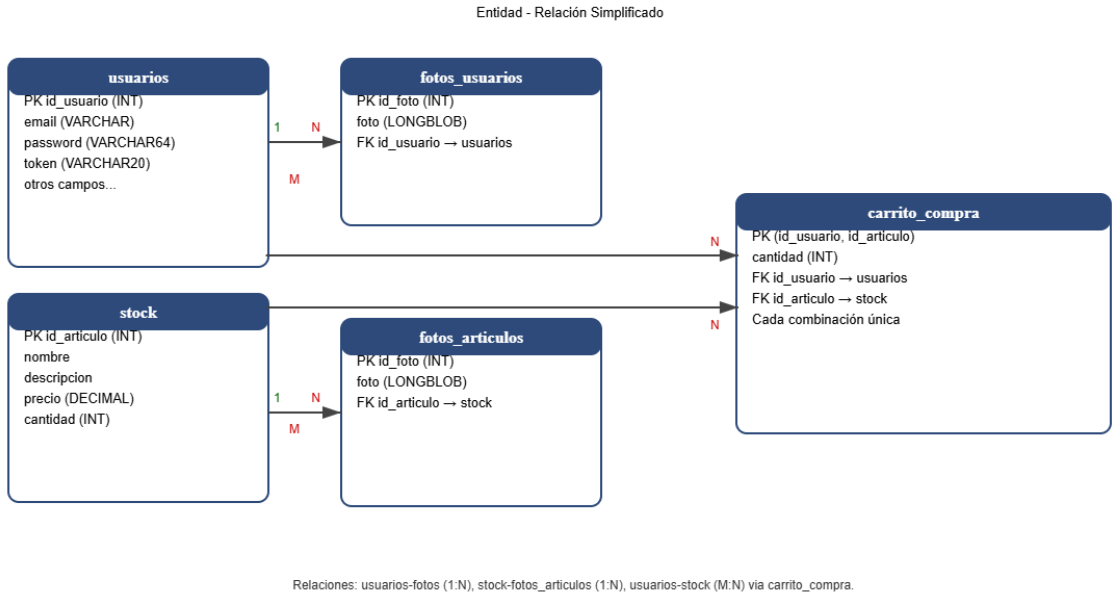


Imagen 9: Diagrama entidad-relación simplificado (Usuarios, Stock, Carrito, Fotos).

6 Desarrollo

7 Infraestructura en Azure

Esta sección documenta la preparación y validación de la infraestructura necesaria para ejecutar el prototipo: la máquina virtual (VM) creada a partir de la imagen de la Tarea 2, la configuración de puertos, los servicios (Tomcat y MySQL) y las evidencias solicitadas por el profesor. Todas las capturas deben ser pantallas completas (sin recortes) mostrando fecha y hora visibles, conforme a los lineamientos del curso.

La VM se nombra estrictamente según la boleta: T7-2022630278. Se selecciona una región en Canadá dentro del programa Azure for Students (Canada Central o Canada East) para cumplir el requerimiento no funcional. La reutilización de la imagen previa acelera la disponibilidad de Tomcat y de la base de datos, reduciendo pasos de instalación.

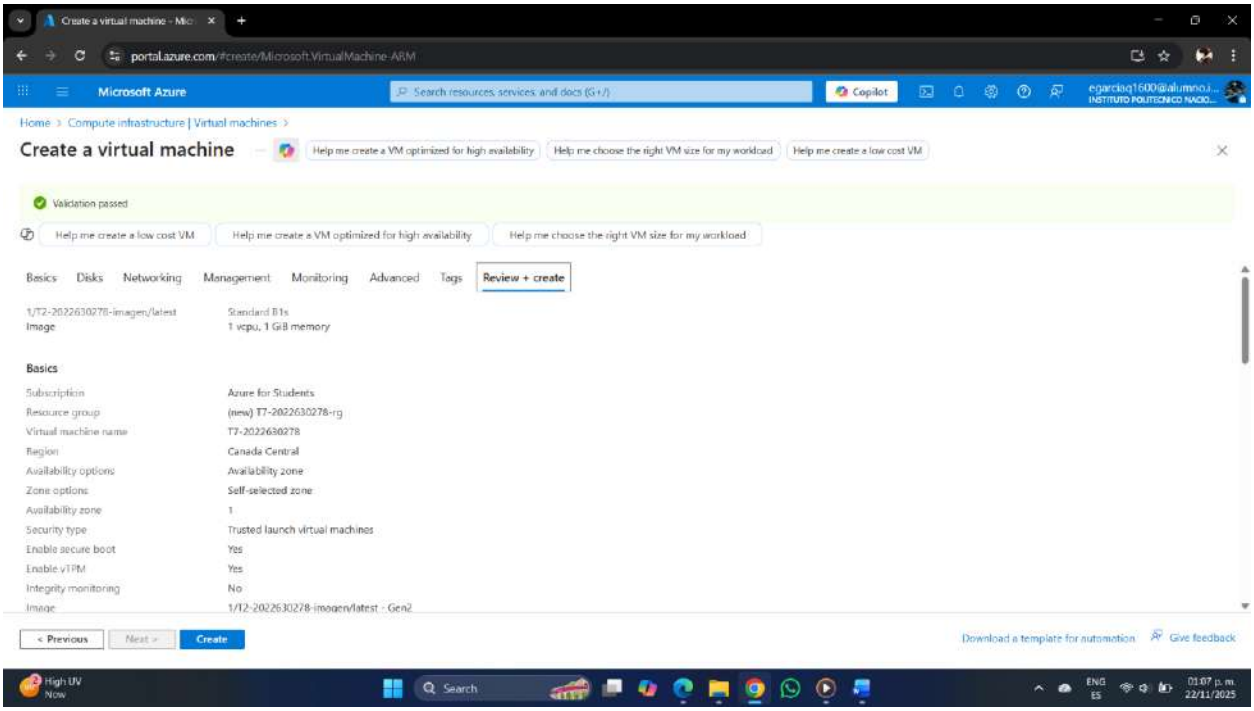
7.1 Creación de la máquina virtual

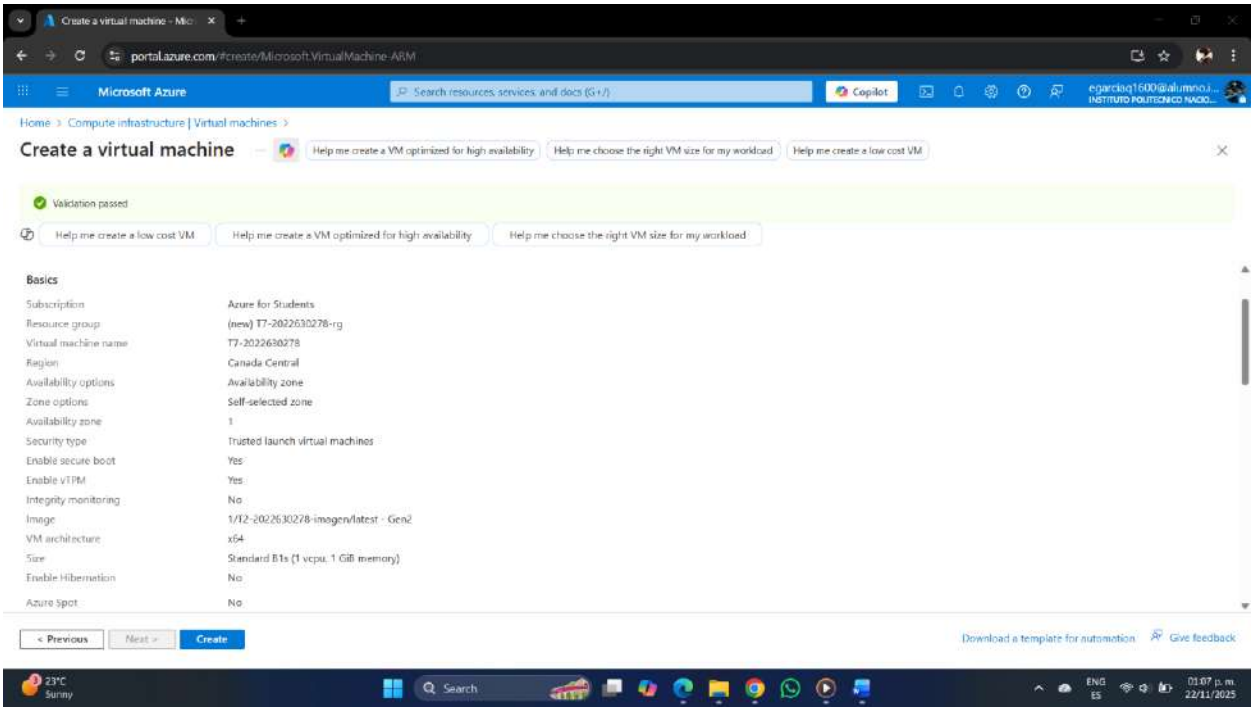
El proceso parte de la imagen generada en la Tarea 2 (que ya incluye entorno Java, Tomcat y MySQL configurado). En el portal de Azure se elige “Create a resource” → “Virtual machine” y se llenan los parámetros mínimos: nombre, región, tamaño, autenticación y red. Se verifica que el nombre coincida exactamente con el formato requerido para evitar rechazo de la tarea.

Pasos esenciales (los más relevantes para evidencia):

1. Seleccionar la suscripción (Students) y el grupo de recursos (reutilizar o crear uno específico p.e. rg-t7-ecommerce).
2. Definir nombre de la VM: T7-2022630278.
3. Elegir región: Canada Central (o Canada East si la primera no disponible).
4. Seleccionar tamaño (ej. B1s o B2s según costo / desempeño).
5. Método de autenticación: clave SSH (recomendado) o contraseña segura.
6. Disco: estándar SSD (suficiente para el prototipo).

7. Activar puerto HTTP (8080) mediante reglas de NSG personalizadas (no hay regla predeterminada para 8080).
8. Revisar pestañas “Networking” y “Management” antes de crear.
9. Confirmar en la pantalla final de validación y presionar “Create”.





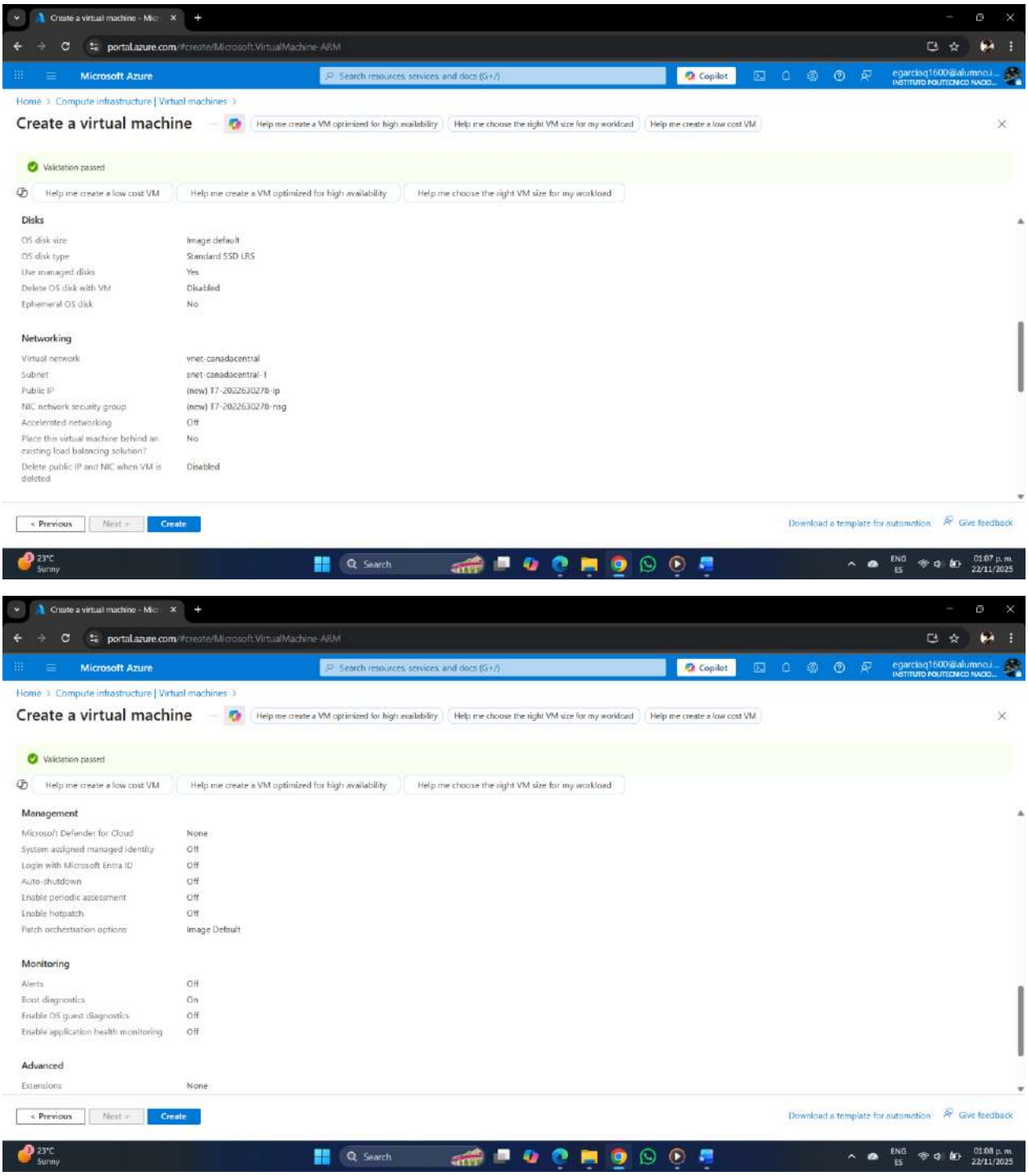


Imagen 11: Pantalla de resumen/Review + Create antes de lanzar la VM con validación “Passed”.

7.2 Características y configuración técnica

Una vez creada, se documentan las especificaciones principales para transparentar el entorno de ejecución del servicio:

Parámetro	Valor Propuesto / Observado	Justificación
Nombre	T7-2022630278	Cumple formato boleta requerido
Región	Canada Central	Requerimiento no funcional
Tamaño	B1s (1 vCPU, 1 GiB) o B2s	Suficiente para Tomcat + MySQL en pruebas
SO Base	Linux (Ubuntu Server) o Windows Server (según imagen T2)	Compatibilidad con stack Java/Tomcat
Disco OS	Standard SSD 30 GiB	Lectura/escritura adecuada para pruebas
Puertos abiertos	22 (SSH), 8080 (HTTP)	Acceso administrativo y aplicación web
Seguridad	NSG con reglas específicas, bloqueo de puertos no usados	Minimiza superficie de ataque
Servicios clave	Tomcat en :8080, MySQL local, Java JDK	Requisitos del prototipo
FQDN/Pública	IP pública asignada por Azure	Necesaria para acceso móvil y pruebas curl

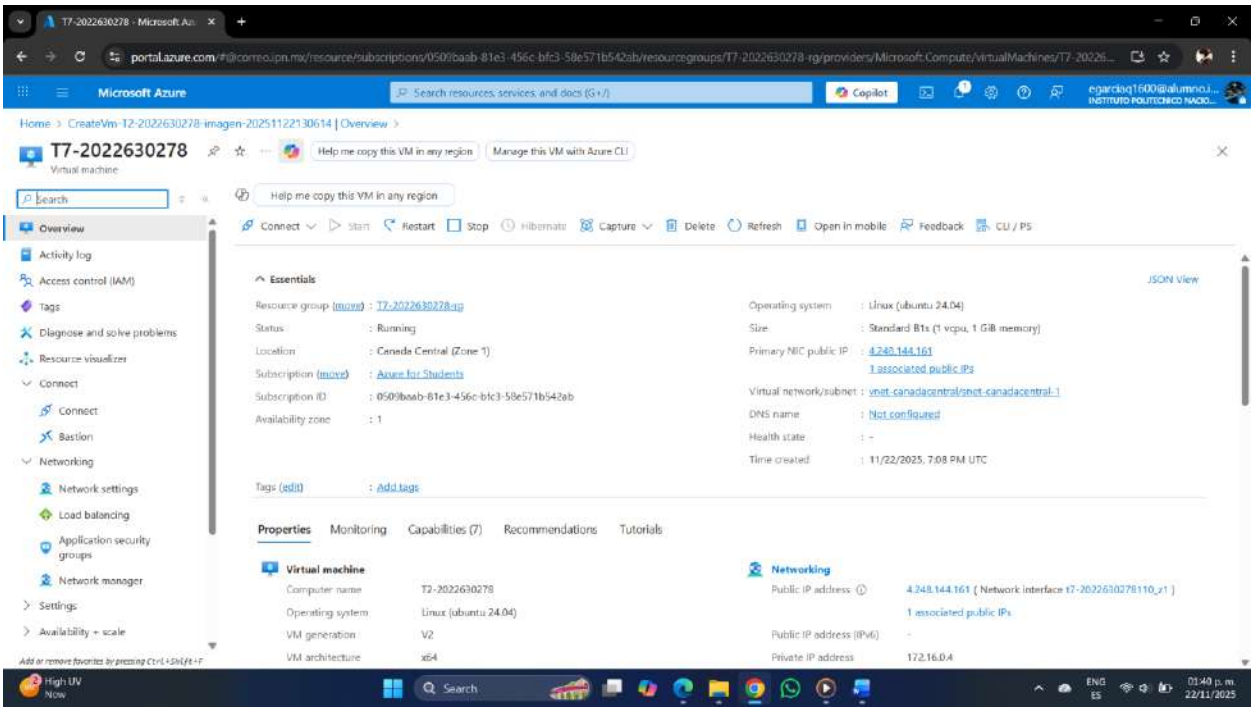


Imagen 12: Detalle de la VM en el portal (Overview con Status: Running, IP pública y región).

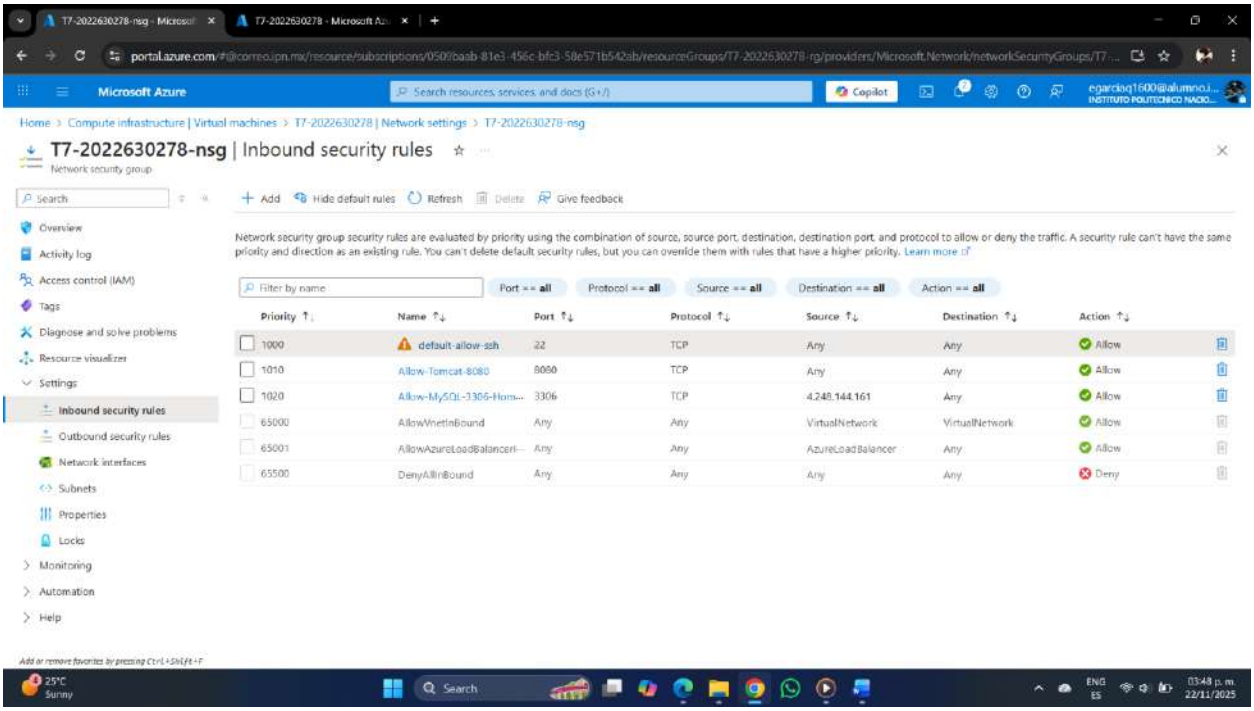


Imagen 13: Configuración de NSG mostrando reglas para puertos 22 y 8080 (toda la pantalla).

8 Configuración del Entorno

Esta sección documenta de forma detallada cada paso seguido para preparar el entorno de ejecución del prototipo: instalación/verificación de Java y Tomcat, configuración del datasource (MySQL), incorporación de librerías de Jersey/Jackson, compilación y despliegue del servicio REST, copia de archivos del front-end y validaciones finales. Todos los pasos incluyen comandos y archivos clave para garantizar reproducibilidad. Las capturas asociadas deben ser pantallas completas con fecha y hora (lineamientos del curso).

8.1 Verificación / Instalación de Java

Antes de compilar el servicio se comprueba la versión de Java disponible en la VM (imagen heredada de Tarea 2).

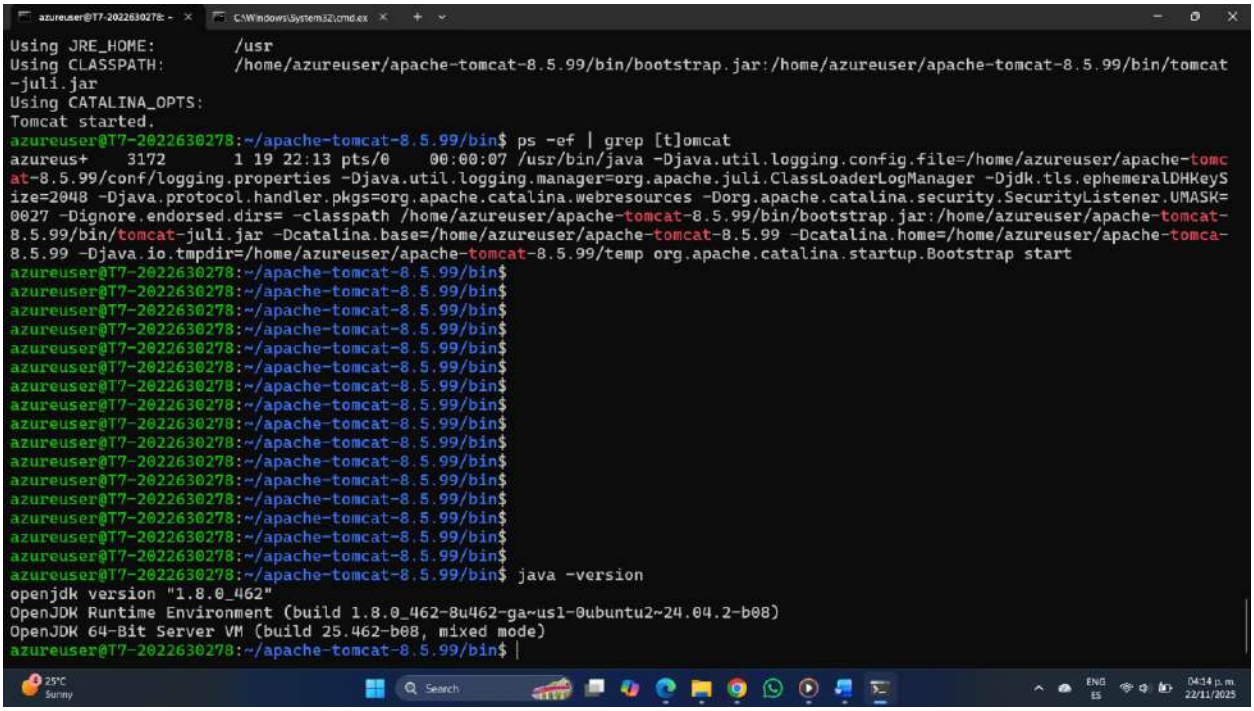


Imagen 16: Consola mostrando java -version con hora visible.

8.2 Verificación / Instalación de Tomcat

La imagen de la Tarea 2 ya incluye Tomcat. Se valida que el servicio esté activo.

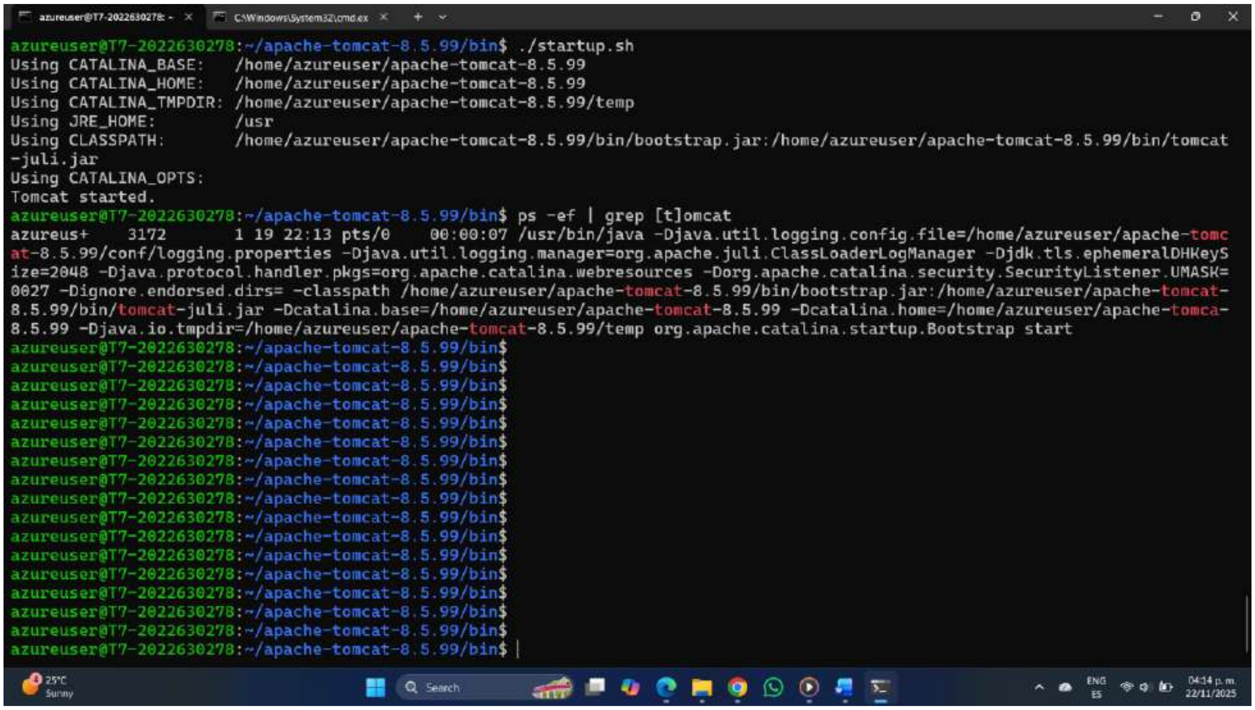


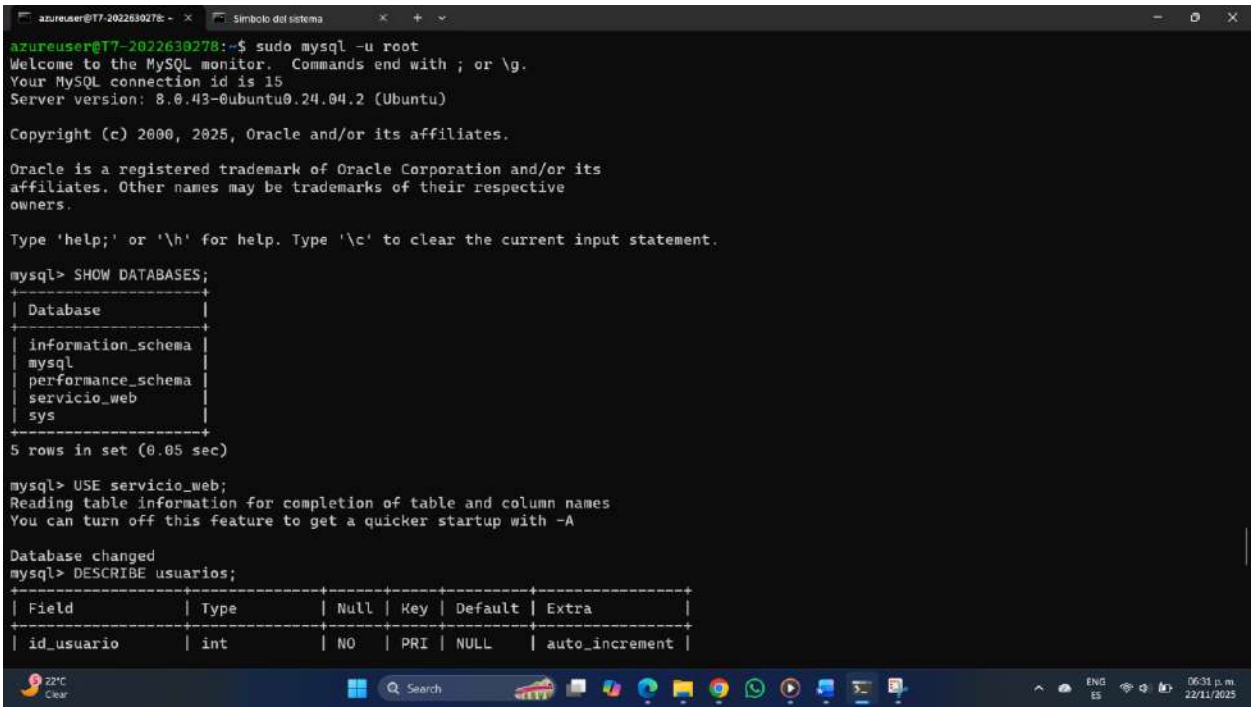
Imagen 17: comando ps -ef | grep tomcat de Tomcat activo.

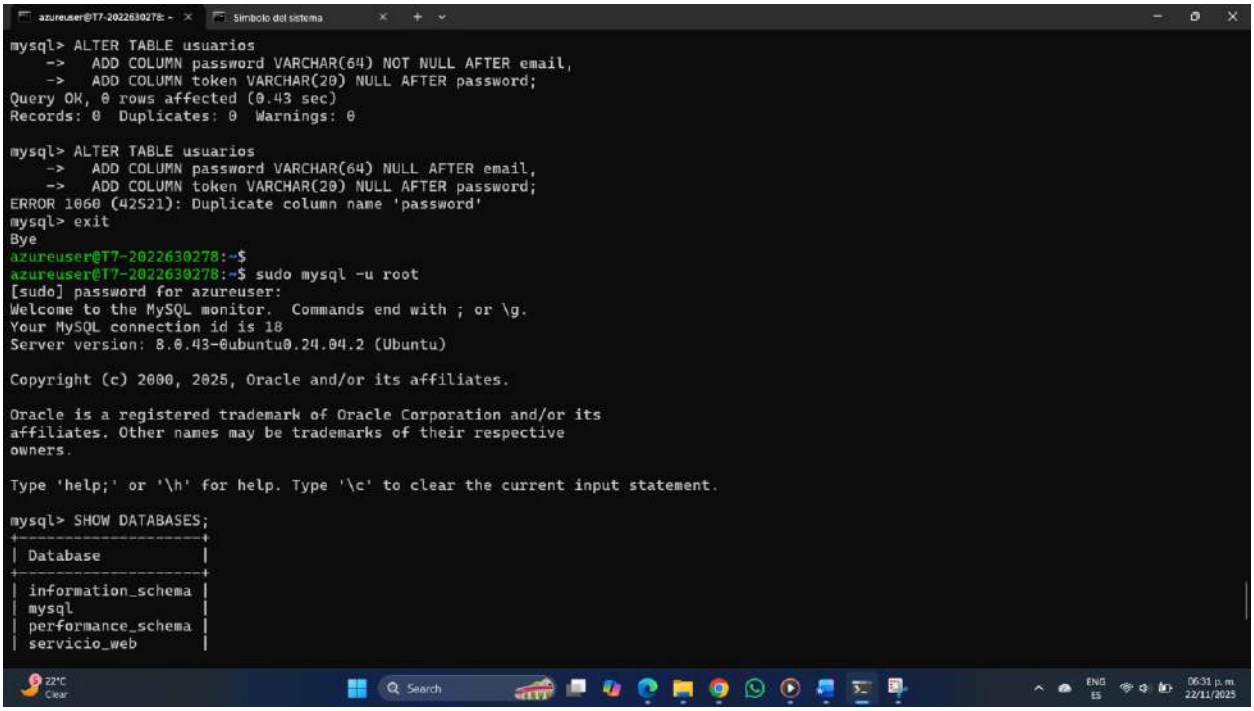
8.3 Base de datos (MySQL)

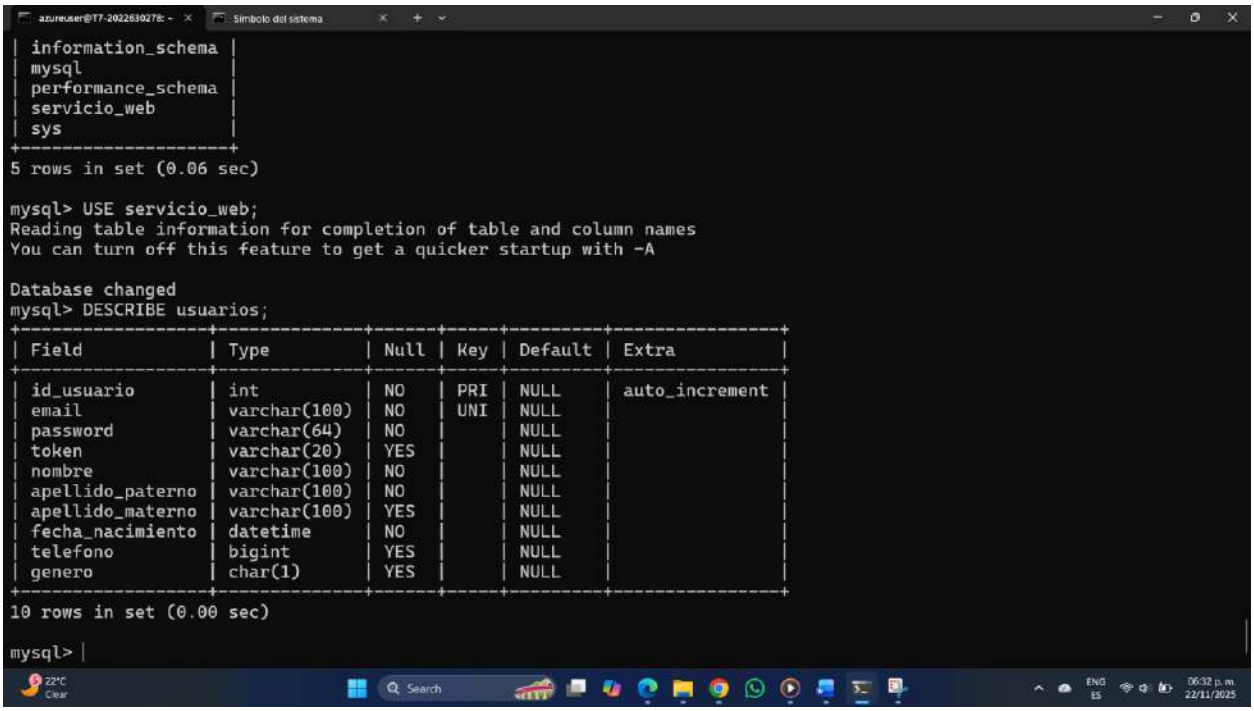
A continuación se explica con detalle cómo realizar la sección de la Base de Datos

8.3.1 Alterar tabla usuarios

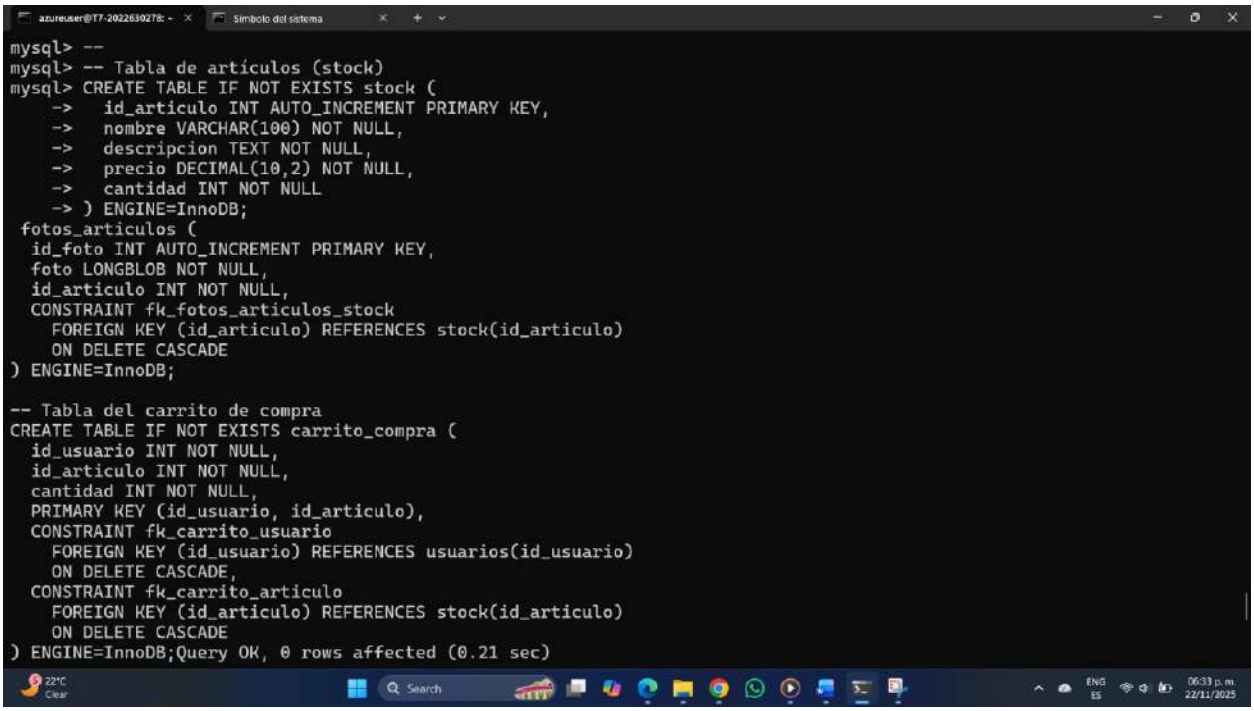
Se agregaron los campos password y token:


azureuser@T7-2022630278: ~
Símbolo del sistema
azureuser@T7-2022630278:~$ sudo mysql -u root
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.43-0ubuntu0.24.04.2 (Ubuntu)
Copyright (c) 2000, 2025, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SHOW DATABASES;
+--------------------+
| Database |
+--------------------+
| information_schema |
| mysql |
| performance_schema |
| servicio_web |
| sys |
+--------------------+
5 rows in set (0.05 sec)
mysql> USE servicio_web;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> DESCRIBE usuarios;
+------------------+--------------+------+-----+---------+----------------+
| Field | Type | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+----------------+
| id_usuario | int | NO | PRI | NULL | auto_increment |
22°C
Clear
Search
ENG
ES
06:31 p. m.
22/11/2025

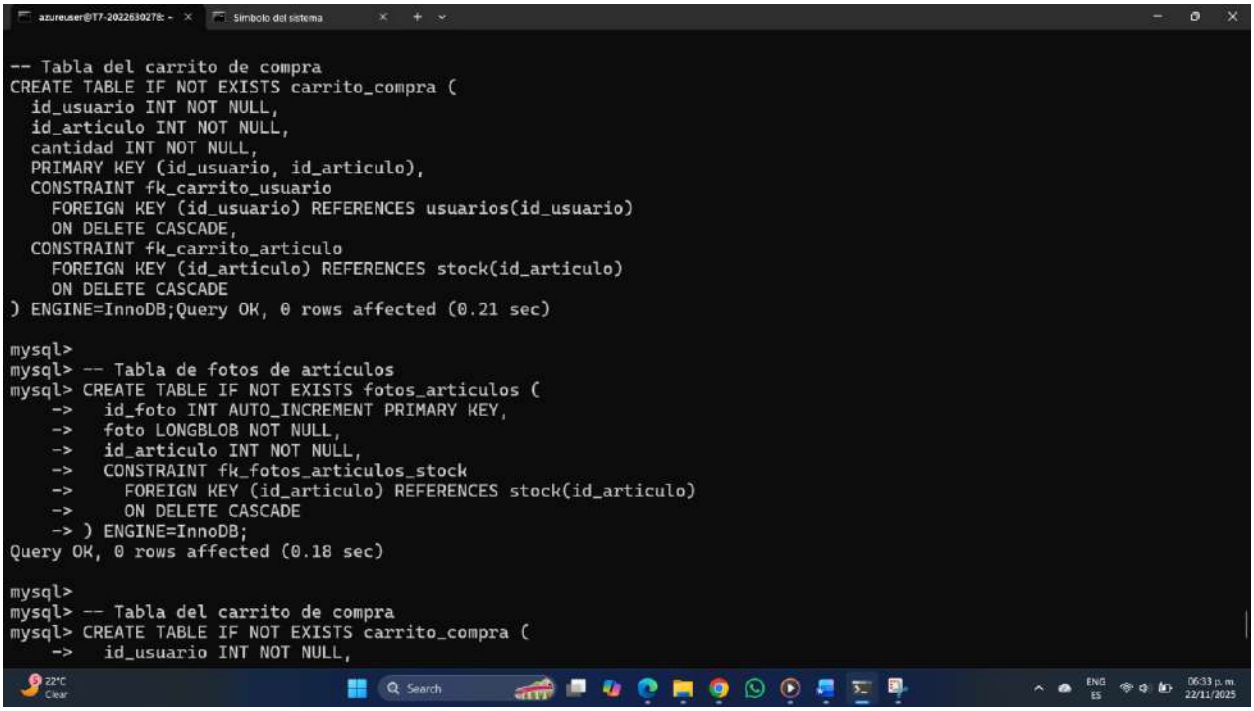

azureuser@T7-2022630278: ~
Símbolo del sistema
mysql> ALTER TABLE usuarios
-> ADD COLUMN password VARCHAR(64) NOT NULL AFTER email,
-> ADD COLUMN token VARCHAR(20) NULL AFTER password;
Query OK, 0 rows affected (0.43 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> ALTER TABLE usuarios
-> ADD COLUMN password VARCHAR(64) NULL AFTER email,
-> ADD COLUMN token VARCHAR(20) NULL AFTER password;
ERROR 1060 (42S21): Duplicate column name 'password'
mysql> exit
Bye
azureuser@T7-2022630278:~$
azureuser@T7-2022630278:~$ sudo mysql -u root
[sudo] password for azureuser:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 8.0.43-0ubuntu0.24.04.2 (Ubuntu)
Copyright (c) 2000, 2025, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SHOW DATABASES;
+--------------------+
| Database |
+--------------------+
| information_schema |
| mysql |
| performance_schema |
| servicio_web |
22°C
Clear
Search
ENG
ES
06:31 p. m.
22/11/2025



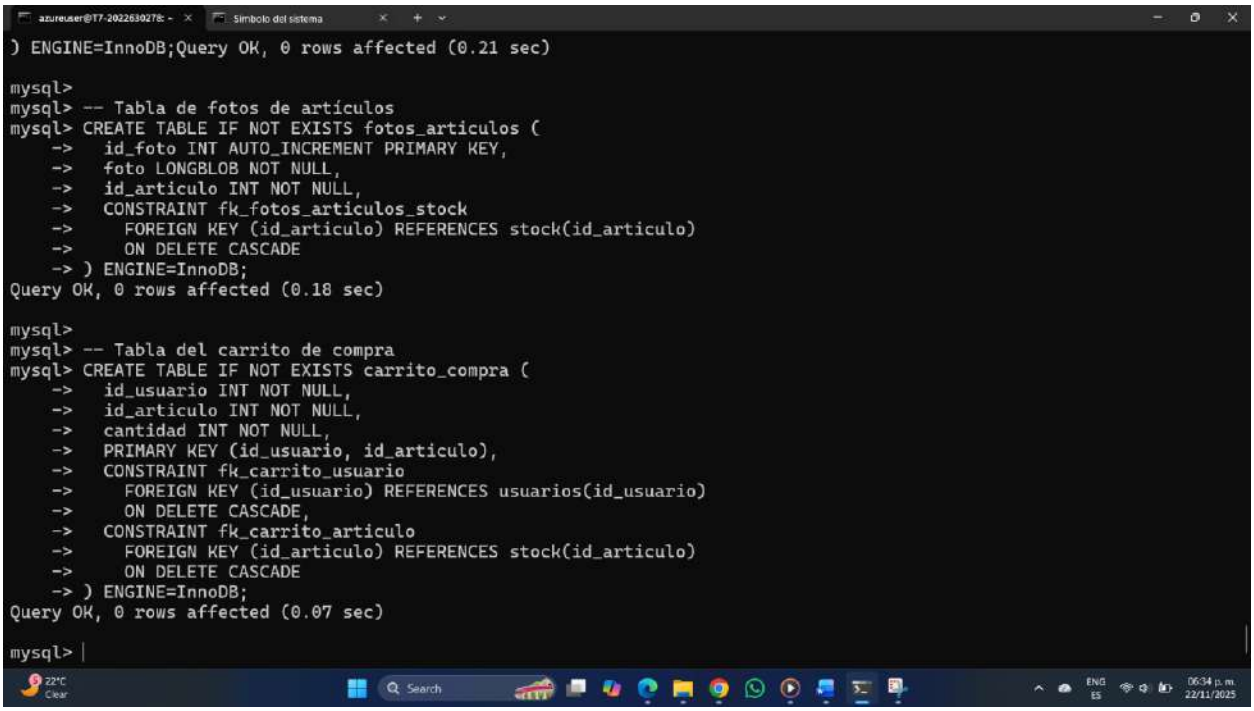
2.2 Crear tablas del e-commerce



```
mysql> --
mysql> -- Tabla de articulos (stock)
mysql> CREATE TABLE IF NOT EXISTS stock (
    ->   id_articulo INT AUTO_INCREMENT PRIMARY KEY,
    ->   nombre VARCHAR(100) NOT NULL,
    ->   descripcion TEXT NOT NULL,
    ->   precio DECIMAL(10,2) NOT NULL,
    ->   cantidad INT NOT NULL
    -> ) ENGINE=InnoDB;
 fotos_articulos (
  id_foto INT AUTO_INCREMENT PRIMARY KEY,
  foto LONGBLOB NOT NULL,
  id_articulo INT NOT NULL,
  CONSTRAINT fk_fotos_articulos_stock
    FOREIGN KEY (id_articulo) REFERENCES stock(id_articulo)
    ON DELETE CASCADE
) ENGINE=InnoDB;

-- Tabla del carrito de compra
CREATE TABLE IF NOT EXISTS carrito_compra (
  id_usuario INT NOT NULL,
  id_articulo INT NOT NULL,
  cantidad INT NOT NULL,
  PRIMARY KEY (id_usuario, id_articulo),
  CONSTRAINT fk_carrito_usuario
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario)
    ON DELETE CASCADE,
  CONSTRAINT fk_carrito_articulo
    FOREIGN KEY (id_articulo) REFERENCES stock(id_articulo)
    ON DELETE CASCADE
) ENGINE=InnoDB;Query OK, 0 rows affected (0.21 sec)
```



```
-- Tabla del carrito de compra
CREATE TABLE IF NOT EXISTS carrito_compra (
  id_usuario INT NOT NULL,
  id_articulo INT NOT NULL,
  cantidad INT NOT NULL,
  PRIMARY KEY (id_usuario, id_articulo),
  CONSTRAINT fk_carrito_usuario
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario)
    ON DELETE CASCADE,
  CONSTRAINT fk_carrito_articulo
    FOREIGN KEY (id_articulo) REFERENCES stock(id_articulo)
    ON DELETE CASCADE
) ENGINE=InnoDB;Query OK, 0 rows affected (0.21 sec)

mysql>
mysql> -- Tabla de fotos de articulos
mysql> CREATE TABLE IF NOT EXISTS fotos_articulos (
    ->   id_foto INT AUTO_INCREMENT PRIMARY KEY,
    ->   foto LONGBLOB NOT NULL,
    ->   id_articulo INT NOT NULL,
    ->   CONSTRAINT fk_fotos_articulos_stock
    ->     FOREIGN KEY (id_articulo) REFERENCES stock(id_articulo)
    ->     ON DELETE CASCADE
    -> ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.18 sec)

mysql>
mysql> -- Tabla del carrito de compra
mysql> CREATE TABLE IF NOT EXISTS carrito_compra (
    ->   id_usuario INT NOT NULL,
```



8.4 Incorporación de librerías (Jersey/Jackson)

Se descargan los JAR indicados (versiones exactas):

- jersey-media-json-jackson-2.24.jar
- jersey-entity-filtering-2.24.jar
- jackson-core-2.6.3.jar
- jackson-databind-2.6.3.jar
- jackson-annotations-2.6.3.jar
- jackson-jaxrs-json-provider-2.6.3.jar
- jackson-jaxrs-base-2.6.3.jar
- jackson-module-jaxb-annotations-2.6.3.jar