Universidad Nacional de Lanús

DESARROLLO DE SOFTWARE EN SISTEMAS DISTRIBUIDOS

Llamado a Procedimiento Remoto (RPC)

Departamento: Desarrollo Productivo y Tecnológico

Carrera: Licenciatura en Sistemas

Año: 2024

Cuatrimestre: 2° Cuatrimestre

Docentes:

Ing. Diego Andrés Azcurra
Lic. Marcos Amaro

Alumnos:

Fernando Jose Luis Camacho (DNI: 35.459.731)
Franco Alberto Astorga (DNI: 42.359.479)
Franco Ariel Figueroa (DNI: 42.684.291)
Juan Leandro Gauna (DNI: 31.654.945)

Índice

1. Código fuente

Enlace a repositorio público de Github: https://github.com/Franco2900/stockearteGrupo5.git

2. Roles y tareas desempeñadas

- Desarrollo de la base de datos: Todos los integrantes
- Desarrollo del servidor Node.js: Fernando Camacho y Franco Figueroa
- Desarrollo del cliente Python: Franco Astorga
- Desarrollo de la interfaz gráfica React.js: Leandro Gauna

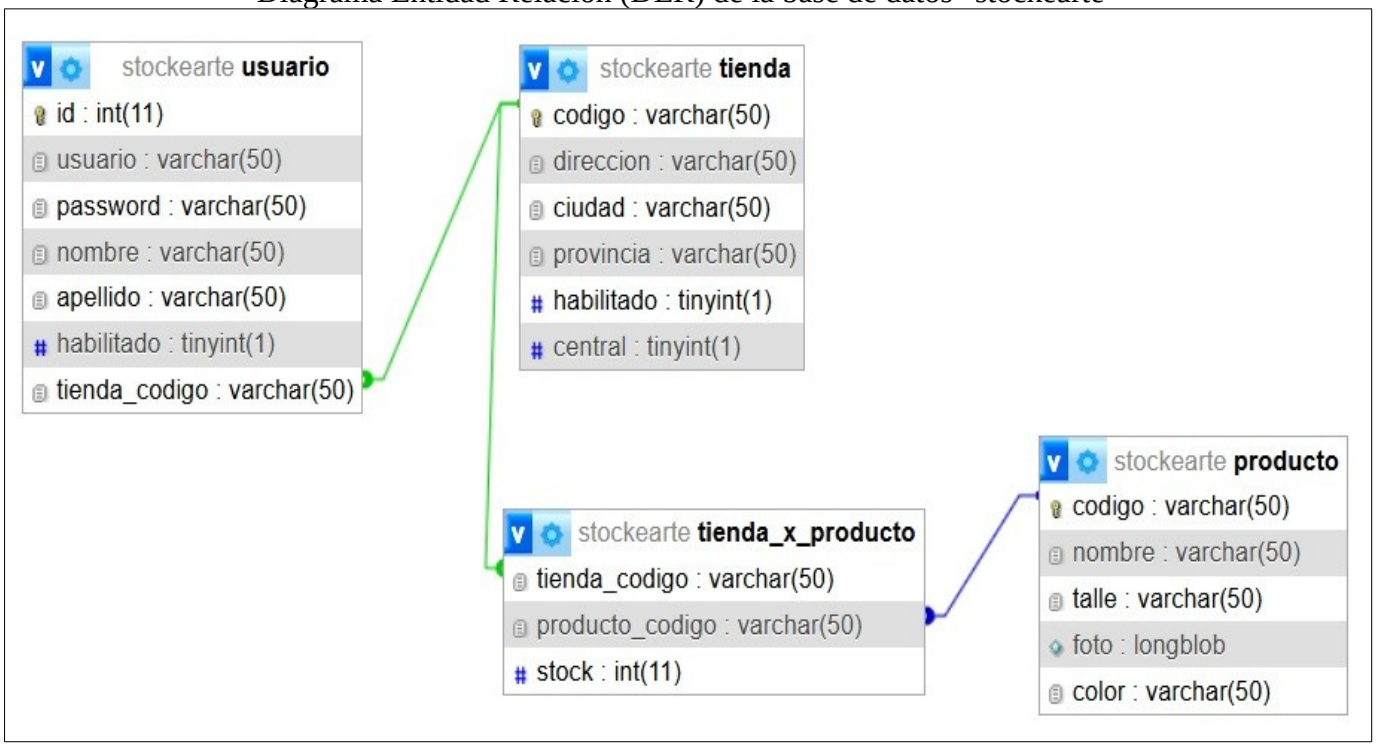
3. Estrategia de resolución

3.1. Base de datos

Para el almacenamiento de los datos se usa una base de datos SQL llamada “stockearte”. En la carpeta ‘Base de datos’ se encuentra el script sql para crear la base de datos y sus tablas usando la colación utf8mb4_general_ci. Para el manejo de la misma fue elegido XAMPP como herramienta de desarrollo.

La base de datos esta compuesta por 4 (cuatro) tablas: tienda, usuario, producto y tienda_x_producto. Como un usuario solo puede pertenecer a una tienda, las tablas tienda y usuario tienen una relación uno a uno. Como un producto puede estar en varias tiendas y una tienda puede tener varios productos, las tablas tienda y producto tienen una relación muchos a muchos.

Diagrama Entidad Relación (DER) de la base de datos “stockearte”



3.2. Servidor

El servidor esta desarrollado con Node.js. Su archivo principal es servidor.js, el cual se encarga de crear el servidor para la comunicación gRPC con el cliente, indicar que servicios atiende el servidor y de escuchar por intentos de conexión.

En la carpeta 'Logica' están los archivos que se encargan de la lógica del negocio. Hay un archivo para cada uno de los puntos de las consignas, esto se realizó con el fin de modularizar el código para un mejor mantenimiento.

No es necesario descargar ninguna librería para el funcionamiento de los archivos anteriormente mencionados ya que todas las librerías necesarias se encuentran en la carpeta 'node_modules'. Las librerías usadas se encuentran especificadas en el archivo package.json dentro de la carpeta Servidor. Se agregan endpoints extras al servidor Flask para poder tener una view al momento de hacer update sobre una tabla.

Debido a que el cliente omitía los campos con valores predeterminados (como false para booleanos) en las respuestas, esto hacia que no se pudieran recibir todos los campos vacios y false del cliente. Con lo cual se agregó en la carga del archivo proto:

Defaults: true

Esto indica que se deben establecer valores predeterminados para todos los campos en los mensajes. Si un campo no se establece explícitamente en un mensaje, se le asignará un valor predeterminado.

KeepCase: true

Esto indica que los nombres de los campos en los mensajes deben mantener el mismo formato de mayúsculas y minúsculas que tienen en el archivo .proto. Por defecto, los nombres de los campos en los archivos .proto se convierten a camelCase cuando se generan los objetos JavaScript.

Creación del servidor

```
var servidor = new gRPC.Server();
```

Carga y configuración del archivo proto

```
var archivoProto = path.join(__dirname, '/../Protos/serviciosStockearte.proto');
var packageDefinition = protoLoader.loadSync(archivoProto, {keepCase: true, defaults: true});
```

Definimos y cargamos el paquete stockearte, el cual contiene la definición de los servicios.

```
const stockearte = gRPC.loadPackageDefinition(packageDefinition).stockearte;
```

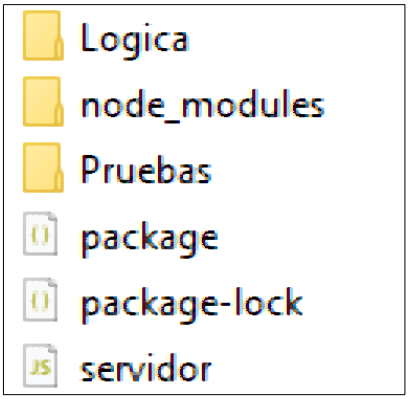
Indicación de servicios del servidor

```
servidor.addService(stockearte.stockearteService.service, {
```

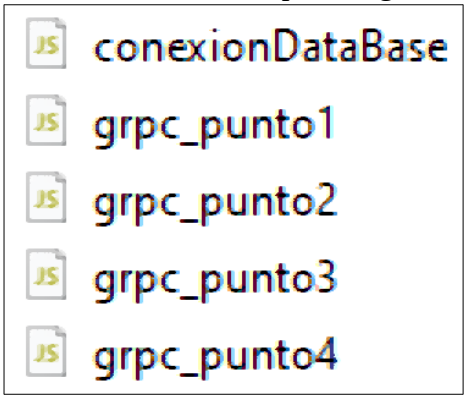
Servidor escuchando

```
servidor.bindAsync("0.0.0.0:8000", gRPC.ServerCredentials.createInsecure(), () => {
```

Archivos de la carpeta Servidor



Archivos de la carpeta Logica



3.3. Protos

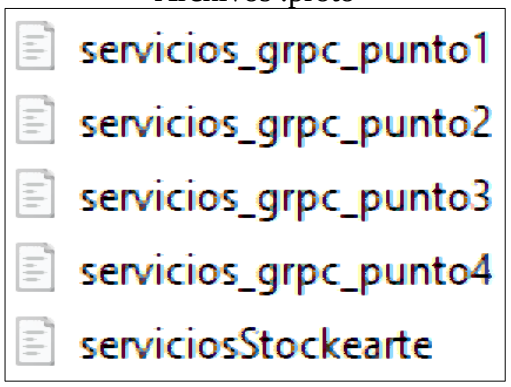
A pesar de que hay varios archivos protos en la carpeta ‘Protos’, al final se utiliza uno solo para la comunicación entre el servidor y el cliente, el archivo serviciosStockearte.proto.

Al principio se iba a usar un archivo proto para cada uno de los puntos de las consignas pero dado que el cliente Python usa 3 (tres) archivos nuevos por cada archivo proto cuando se compila, al final el cliente tendría que manejar 12 (doce) archivos nuevos al mismo tiempo; por lo que se decidió usar un solo archivo proto para hacer la comunicación más simple.

Los archivos protos ya hechos no fueron borrados para obtener una mejor idea de como funciona la comunicación entre el cliente y el servidor para cada punto.

Debido al problema mencionado anteriormente con los datos booleanos se llegó a la solución de agregar el campo optional a los campos con este tipo de dato, para que el cliente pudiese recibir correctamente los valores.

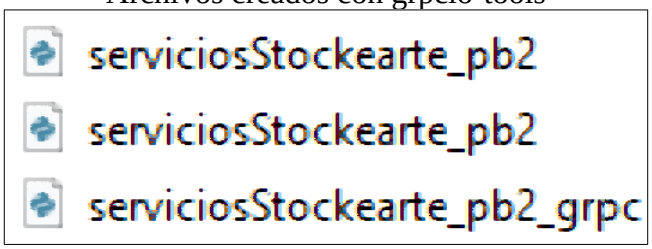
Archivos .proto



3.4. Cliente

El cliente esta desarrollado en Python. Para poder comunicarse con el servidor, el cliente Python usa la librería grpcio-tools de pip para crear 3 (tres) archivos que le permiten la comunicación mediante gRPC. Esos archivos están hechos a partir del archivo serviciosStockearte.proto, por lo que el cliente puede llamar a cualquier servicio definido en el archivo proto.

Archivos creados con grpcio-tools



El cliente crea una conexión con el servidor

```
canal = grpc.insecure_channel("localhost:8000")
```

El cliente crea un stub para usar los servicios gRPC

```
stub  = serviciosStockearte_pb2_grpc.stockearteServiceStub(canal)
```

El cliente actuá como intermediario entre el servidor y la interfaz gráfica, ya que recibe las peticiones de la interfaz gráfica para enviárselas al servidor y devuelve las respuestas del servidor a la interfaz gráfica.

Para recibir las peticiones de la interfaz gráfica, el cliente Python usa Flask para crear una aplicación web que al llamar a determinadas rutas se encarga de hacer la comunicación anteriormente mencionada. Tanto los mensajes entrantes que envía la interfaz, como las respuestas del servidor están en formato JSON, para una manipulación más sencilla. En cada endpoint se crea un stub con el mismo canal y se le carga la solicitud para que el servidor pueda procesarla.

Cliente usando Flask

```
app = Flask(__name__)
```

Una de las rutas que maneja el cliente con Flask

```
@app.route('/altaTienda', methods=['POST'])
def altaTienda():
```

No es necesario descargar ninguna librería para el funcionamiento del cliente Python ya que todas las librerías necesarias se encuentran en la carpeta ‘EntornoVirtualPython’, el cual se puede generar desde el .bat que tiene dentro. Esto permite que cualquier usuario pueda ejecutar el proyecto en cualquier PC sin necesidad de descargar a mano cada librería.

3.5. Interfaz Gráfica

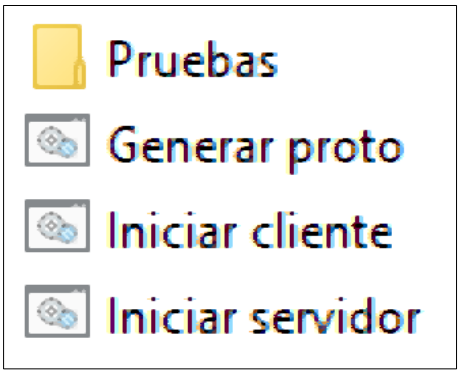
No llegamos a desarrollar la interfaz gráfica pero el objetivo en mente es que este sea en un nodo independiente desarrollado con Node, React y Bootstrap.

3.6. Ejecutables

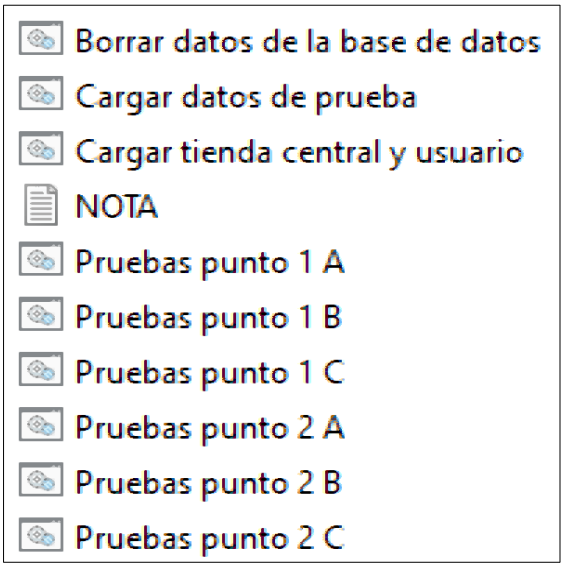
Esta carpeta tiene archivos .bat que sirven para ejecutar comandos automáticamente en la consola de comandos de Windows (cmd). Estos ejecutables sirven para automatizar distintos procesos: desde el proceso de pruebas hasta la iniciación del servidor y el cliente.

Los archivos para realizar las pruebas de los puntos de la consigna están en la carpeta ‘Pruebas’. También hay otros archivos .bat para cargar o borrar los datos de la base de datos “stockearte”.

Archivos de la carpeta Ejecutables



Archivos de la carpeta Pruebas



4. Pruebas realizadas

Todas las pruebas fueron realizadas en Windows 10 Home. Para poder realizar las pruebas es necesario tener Xampp, Node.js y Python instalados. Se uso Xampp para manejar la base de datos. Para iniciar el servidor Node.js y el cliente Python, en la carpeta Ejecutables hay que ejecutar los .bat “Iniciar servidor” e “Iniciar cliente” (también se puede ingresar manualmente los comandos que hay dentro de estos archivos).

En la carpeta Pruebas dentro de la carpeta Ejecutables están todas las pruebas que fueron realizadas para probar el buen funcionamiento del proyecto.

Enlace al video de las pruebas realizadas:
https://drive.google.com/file/d/1VzWitE3ttPbvsqcZ4poEIWaJbQtJM7Ca/view?usp=sharing