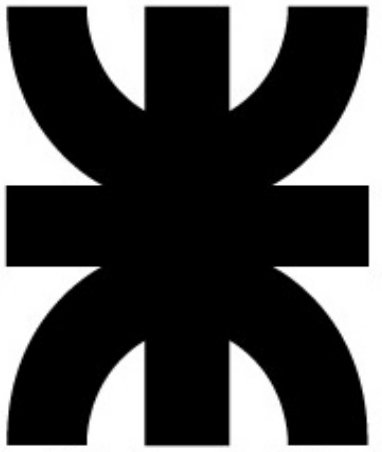

Universidad Tecnológica Nacional
Facultad Regional Resistencia
Ingeniería en Sistemas de Información

Cátedra: **Desarrollo de Software**
Año: **2025**

Trabajo Práctico Integrador

Grupo N° 10

Integrantes:

Nombre y Apellido	**Correo electrónico**
Almirón, Sabugo Alejo Manuel	alejo.almiron2018@hotmail.com
Bruzzo, Juan Ignacio	juanignaciobruzzo@ca.frre.utn.edu.ar
Decoud, Santiago	santinodecoud10@gmail.com
Landaeta Martínez, María Gracia Esther	mariagelandaetam@gmail.com
Ruiz Diaz, Agostina	agusruizdiaz08@gmail.com
Sánchez, Rosalía	rosancheza2@gmail.com
Sanchez, María Eugenia	eugemasanchez@gmail.com
Simonek, Joaquín	joaquinsimonek@gmail.com
Vega, Lourdes Rocío	lourdesvega2002@gmail.com
Villalva, Joel Tomás	joelvillalva06@gmail.com

Profesores:

Villaverde, Jorge Eduardo
Fernández, Jose Alejandro

INDICE

1. Introducción

1.1. Contexto del Proyecto

El presente informe documenta el desarrollo del **Portal de Compras**, que constituye uno de los tres subsistemas interdependientes requeridos para el Trabajo Práctico Final (TPF) de la cátedra Desarrollo de Software. Este ejercicio académico tiene como objetivo la aplicación práctica de conocimientos en la construcción de un sistema de comercio electrónico completo.

Este documento detalla la arquitectura, diseño e implementación del *backend* del sistema, concebido como una API RESTful que servirá como núcleo central de servicios para la plataforma.

1.2. Problema y Objetivos

La consigna principal (Actividad 1) requiere la creación y desarrollo de un Portal de Compras que debe proveer a los usuarios finales la funcionalidad esencial de un *e-commerce* moderno.

El objetivo central de nuestro trabajo es diseñar e implementar una API RESTful que no solo soporte todas estas operaciones de manera segura y eficiente, sino que también cumpla con su rol de orquestador. Específicamente, la API debe comunicarse activamente con los subsistemas externos del TPF para tareas clave, como la reserva de *stock* en el servicio de Bienes y Servicios y el cálculo de costes de transporte en el servicio de Logística.

La metodología de desarrollo se centró en la aplicación de herramientas recomendadas por la cátedra. Se estableció Git como el sistema de control de versiones para la gestión colaborativa del código, con una estrategia de integración centralizada para asegurar la calidad y la trazabilidad de las ramas. Finalmente, la validez de los servicios fue verificada sistemáticamente utilizando Postman para confirmar que todos los *endpoints* estuvieran listos para la correcta integración con el *frontend* del Portal y los otros servicios del ecosistema.

1.3. **Arquitectura y Stack Tecnológico**

Para cumplir con los requisitos, se ha optado por una arquitectura de microservicios o servicios desacoplados, donde una API central maneja toda la lógica de negocio y el acceso a datos.

El stack tecnológico seleccionado para el desarrollo del *backend* es:

- Framework: .NET (utilizando Visual Studio 2022) para construir la API RESTful.
- Base de Datos: MySQL (gestionada con MySQL Workbench 8.0 CE), sirviendo como almacén principal de datos para productos, usuarios y pedidos.
- Autenticación y Seguridad: Keycloak, desplegado en un contenedor de Docker, para

manejar la autenticación de usuarios (Identity Management).

- Pruebas de API: Postman, como herramienta principal para el testeo y validación de *endpoints*.

Esta API está diseñada para ser consumida por el *frontend* del portal (desarrollado en Angular) La justificación detallada de cada elección se presenta a continuación en la sección de Desarrollo

2. Desarrollo

Decisiones de Diseño y Soluciones Implementadas

A lo largo del trabajo se necesitó la implementación de un *stack* tecnológico robusto y moderno, donde cada herramienta cumple una finalidad específica. Las elecciones se basaron en el conocimiento con el que contaba el equipo, la necesidad de construir una Arquitectura de Servicios Desacoplados para la integración con los subsistemas externos, y el aporte de rendimiento y seguridad al sistema.

2.1. Arquitectura y Framework Principal

(Backend)

Para el desarrollo de la lógica de negocio y la exposición de servicios, se seleccionó un *framework* de alto rendimiento, cumpliendo la decisión de separar la presentación del procesamiento de datos.

.NET 9.0 (C#)

Microsoft .NET fue elegido como *framework* principal para construir el *Backend* y la API RESTful. La elección se centró en su robustez, su capacidad para manejar grandes cargas de trabajo y la compatibilidad con un modelo de Arquitectura en Capas bien definido

- ¿Por qué usamos .NET? Es un *framework* moderno y de alto rendimiento que nos permite implementar la API RESTful que sirve como el núcleo central de servicios. Esto garantiza que las operaciones críticas (como el registro de compras y la orquestación de servicios) se ejecuten de manera segura y eficiente. A su vez, algunos miembros tenían conocimiento de este y lo consideramos apropiado.

- Uso en el Proyecto: Utilizamos Visual Studio 2022 para la codificación. El Backend se encarga de recibir las peticiones del Frontend, aplicar la lógica de negocios y, crucialmente, realizar las peticiones a los servicios de Stock y Logística, actuando como el módulo orquestador del TPF.

Angular

Angular fue el *framework* seleccionado para el desarrollo del Frontend del Portal de Compras.

- ¿Por qué lo escogimos? Angular es un *framework* completo para la interfaz de usuario, lo cual nos permite construir una Aplicación de Página Única (SPA) con una estructura de componentes clara y un sistema de enrutamiento robusto.
- Uso en el Proyecto: Angular consume directamente los *endpoints* de la API de .NET. Es el responsable de la Gestión de Usuarios del lado del cliente, la Visualización de Productos y toda la experiencia interactiva del usuario.

2.2. Persistencia de Datos y ORM

Para el almacenamiento de información crítica (productos, usuarios, pedidos), se implementó una solución relacional y se utilizó una herramienta de mapeo.

MySQL

MySQL (gestionado con MySQL Workbench 8.0 CE) fue elegido como el sistema de gestión de bases de datos relacionales.

- ¿Por qué usamos MySQL? Es una base de datos estable, ampliamente utilizada y de código abierto, ideal para el manejo de datos estructurados como los requeridos para un *e-commerce*, además, ya estábamos familiarizados con MySQL debido a que la materia de Bases de Datos nos proporcionó los conocimientos básicos para su uso.
- Uso en el Proyecto: Sirve como almacén principal de datos para todas las entidades del Portal de Compras.

ORM (Entity Framework Core)

Como solución implementada para el acceso a datos, se utilizó un ORM dentro de la capa Data del Backend.

- Uso de ORM: Permite a los desarrolladores interactuar con la Base de Datos utilizando clases de C# en lugar de código SQL, lo que reduce errores, acelera el desarrollo y permite la fácil migración entre bases de datos relacionales.

2.3. Seguridad e Integración de Servicios

La seguridad y la comunicación entre los tres subsistemas fueron abordadas con herramientas dedicadas, cumpliendo con los estándares de arquitectura distribuida.

Keycloak (Autenticación y Seguridad)

Keycloak fue la herramienta seleccionada para manejar la Autenticación y Seguridad (*Identity Management*).

- Decisión de Diseño: Delegar la gestión de usuarios y la autenticación a un servidor de identidad externo (Keycloak) es una decisión clave en arquitecturas distribuidas.

Esto permite centralizar la identidad y asegurar que todos los subsistemas puedan validar a los mismos usuarios.

- Funcionamiento: Keycloak emite JSON Web Tokens (JWT) tras un *login* exitoso. El Backend de .NET utiliza estos *tokens* para autorizar el acceso a sus *endpoints*.

- Despliegue: Fue desplegado en un contenedor de Docker, asegurando un ambiente de ejecución aislado y reproducible para el servicio de identidad.

OpenAPI Specification (OAS)

OpenAPI es la solución implementada para la comunicación entre grupos, estableciendo un contrato de interfaz formal.

- Uso en el Proyecto: Se genera una especificación en formato YAML para definir los *endpoints*, los parámetros de entrada y los resultados de salida de la API de Compras. Esto asegura que la interacción con los servicios de Stock y Logística sea clara y estandarizada.

- Beneficio: Permite a los otros grupos consumir la especificación y, potencialmente, usar el OpenAPI Generator para automatizar la creación de código cliente, lo cual es una práctica profesional avanzada para la integración de servicios.

2.3.1 Integración Operativa entre los Subsistemas

La integración del Portal de Compras con los servicios de Stock y Logística requirió la coordinación de tres entornos independientes, cada uno ejecutado en contenedores Docker y autenticado mediante instancias propias de Keycloak. El objetivo fue establecer un flujo distribuido en el cual Compras actúa como orquestador central, consumiendo los servicios externos para completar sus operaciones.

Para garantizar un ambiente homogéneo, cada subsistema fue desplegado en su propio conjunto de contenedores, incluyendo su backend, su base de datos correspondiente y su servidor de identidad. Una vez en ejecución, los servicios quedaron disponibles localmente para permitir la comunicación entre ellos, respetando las interfaces definidas en sus respectivas especificaciones OpenAPI.

Durante el proceso de integración se verificó que el módulo de Compras pudiera autenticarse correctamente contra su instancia de Keycloak y utilizar los tokens generados para autorizar solicitudes hacia los subsistemas externos. De igual manera, se comprobó que los servicios de Stock y Logística aceptaran peticiones válidas, procesaran la información y devolvieran respuestas acordes a los contratos establecidos.

La validación funcional se llevó a cabo mediante las páginas Swagger de cada servicio, lo que permitió inspeccionar sus endpoints, formatos de entrada y estructura de respuestas. Esto facilitó la identificación y resolución de diferencias entre implementaciones, garantizando una compatibilidad total durante las pruebas.

Finalmente, se validó la persistencia de los datos generados durante la integración, revisando las bases internas de cada subsistema. Con ello se confirmó que las operaciones de reserva de stock, registro de envíos y generación de órdenes se propagaran correctamente entre los distintos módulos del ecosistema distribuido.

2.4. Herramientas de Proceso y Calidad

Para el proceso de desarrollo se utilizaron herramientas enfocadas en la organización y la calidad del código.

Postman (Pruebas de API)

Postman fue la herramienta principal para el testeo y validación de *endpoints*.

- Uso en el Proyecto: Permitió al equipo enviar peticiones de prueba (POST, GET, etc.) a la API de .NET de forma sistemática. Esto fue crucial para verificar que el Backend funcionara correctamente (ej. el registro de usuarios) antes de la integración con el Frontend, asegurando la calidad de los servicios.

2.5. Flujo de Integración entre Subsistemas

El flujo de integración implementado permite que el Portal de Compras actúe como orquestador central, consumiendo los servicios de Stock y Logística para completar todas las operaciones necesarias de un proceso de compra distribuido.

El proceso comienza cuando el usuario agrega un producto al carrito. Dichos productos provienen directamente de la base de datos del subsistema de Stock. Una vez incorporado al carrito, el usuario puede eliminarlo o continuar con la compra. Para continuar, se requiere seleccionar o ingresar una dirección existente en la base de datos del subsistema de Logística, ya que esa información es necesaria para calcular el costo del envío.

El cálculo del envío se realiza mediante una solicitud al servicio de Logística, y posteriormente se habilita la opción de finalizar la compra. Cuando la compra se confirma, el pedido queda registrado en el sistema y puede visualizarse en el historial del usuario. Para esto, el módulo de Compras realiza una consulta interna al subsistema de Logística con el fin de obtener los datos asociados al envío del pedido.

Finalmente, el portal también permite la búsqueda y filtrado de productos mediante la barra superior, enviando consultas directamente a la API de Stock para obtener coincidencias en tiempo real.

3. Desafíos Técnicos Superados

Durante el desarrollo del proyecto, se identificaron diversos desafíos que afectaron el progreso del equipo. A continuación, se detallan los principales problemas enfrentados:

3.1. Dificultad para Familiarizarse con .NET y Angular

La elección de **.NET** para el *Backend* y **Angular** para el *Frontend* supuso una **curva de aprendizaje compleja** para varios integrantes. Como no estábamos familiarizados con estos *frameworks* de gran escala, tuvimos que dedicar mucho tiempo extra a investigar y entender sus estructuras internas, los patrones de código que usan y cómo debíamos trabajar con C# y TypeScript. Esto hizo que las primeras tareas fueran más lentas hasta que logramos adaptarnos a la lógica de estos entornos.

3.2. Complejidad en la Integración y Flujo de Autenticación con Keycloak

El obstáculo más grande fue la migración y la implementación completa del sistema de autenticación, que pasó de una solución simple a una distribuida con Keycloak.

- **Migración de Lógica Local:** Al inicio, habíamos creado nuestro propio controlador de *login* y *register* local para generar los *tokens* **JSON Web Token (JWT)**. Tuvimos que **cambiar completamente** esta lógica cuando se nos pidió usar **Keycloak**. Esto significó borrar todo lo que habíamos hecho y reconfigurar la API para que delegara el registro y el *login* a Keycloak.
- **Problemas con los Tokens y el Flujo:** Uno de los mayores desafíos fue la **obtención y validación de los *tokens***. Era complicado hacer que el **Frontend** se comunicara correctamente con **Keycloak** para obtener el *token*, y luego asegurar que nuestra **API de Compras** lo aceptara para autenticar a los usuarios. El flujo de autenticación completo (Front -> Keycloak -> API -> Base de Datos) era muy sensible y generó muchos errores hasta que logramos hacerlo funcionar de manera segura.

3.3. Errores Críticos de Despliegue y Comunicación con Servicios Externos

La etapa final de *dockerización* e integración presentó problemas de tiempo y configuración.

- **Dificultades con Docker:** Al intentar generar la imagen del contenedor de nuestra API de Compras, surgieron una gran cantidad de errores que no podíamos solucionar fácilmente. Pasamos **casi un día entero** buscando y corrigiendo estos problemas de configuración de Docker para que la imagen se pudiera cargar correctamente.

- **Tokenización para otros grupos:** Tuvimos que asegurarnos de que el *token* de Keycloak funcionara bien para autorizar las peticiones no solo de nuestro Frontend, sino también de las **otras APIs (Stock y Logística)**, lo que implicó una configuración de seguridad compleja en el entorno de contenedores.

- **Problemas de integración con Stock:** Tuvimos dificultades específicas con la comunicación externa. Nos costó levantar correctamente el contenedor de la API de Stock y su base de datos, y hacer que ese subsistema se comunique sin fallos con nuestro subsistema de Compras.

- **Conflictos de puertos entre los subsistemas:** Tuvimos inconvenientes debido a que los archivos docker-compose.yaml de Stock y Logística utilizaban puertos distintos a los nuestros, generando superposiciones con puertos locales. Fue necesario ajustar varias veces estos valores para evitar colisiones y garantizar que todos los servicios pudieran ejecutarse simultáneamente.

- **Adaptación de modelos para la interoperabilidad:** Se presentaron dificultades al integrar los modelos de nuestra API de Compras con las estructuras utilizadas por las APIs de Stock y Logística. Fue necesario modificar y adecuar nuestros modelos existentes para que coincidieran con los formatos esperados por los otros subsistemas, lo cual añadió complejidad al proceso de integración.

- **Validación individual y prueba del flujo completo:** Una vez que los tres servicios estuvieron en funcionamiento (Compras, Stock y Logística), debimos verificar cada uno de manera independiente utilizando Postman y Swagger, asegurándonos de que los endpoints devolvieran las respuestas correctas. Tras esa validación individual, se procedió a probar el flujo de integración completo, primero desde Postman y finalmente desde nuestro frontend.

4. Conclusiones

En retrospectiva, la ejecución de este proyecto ha representado una experiencia formativa invaluable. Su amplio alcance, que implicó la creación de un subsistema funcional dentro de una Arquitectura Distribuida de Servicios, ofreció una oportunidad única para la aplicación práctica de los conocimientos adquiridos.

La necesidad inherente de investigar, seleccionar y complementar tecnologías diversas —como .NET, Angular, y el estándar OpenAPI— para resolver los múltiples desafíos de integración y seguridad, consolidó un aprendizaje práctico profundo para cada miembro del equipo.

A lo largo del proceso, enfrentamos desafíos significativos que trascendieron lo meramente técnico. Tuvimos dificultades iniciales en la coordinación de tareas y la gestión eficiente del flujo de comunicación con un equipo de mayor tamaño. Asimismo, la curva de aprendizaje en herramientas como Keycloak para la gestión de identidad requirió esfuerzo adicional.

No obstante, la capacidad de superar colectivamente estos obstáculos, reajustar la organización del trabajo y establecer un contrato de interfaz común (OpenAPI) para la integración con los otros subsistemas, se convirtió en el logro más relevante. Este trabajo práctico integrador no solo resulta en un producto funcional, sino que también suma experiencia esencial en la colaboración, la resolución de problemas en entornos distribuidos y el dominio de un *stack* tecnológico acorde a las demandas de la industria del software actual.

5. Bibliografías

Angular. (n.d.). *Introduction to the Angular docs*. Angular. Recuperado en noviembre de 2025, de https://angular.io/docs

Docker. (n.d.). *Docker Documentation*. Docker. Recuperado en Noviembre de 2025, de https://docs.docker.com/

GitHub. (n.d.). *Configuración de Git*. GitHub Docs. Recuperado en Noviembre 2025, de https://docs.github.com/es/get-started/getting-started-with-git

Keycloak. (n.d.). *Documentation*. Keycloak. Recuperado en Noviembre de 2025, de https://www.keycloak.org/documentation

Microsoft. (n.d.). *ASP.NET documentation*. Microsoft Learn. Recuperado en Noviembre de 2025, de https://learn.microsoft.com/es-es/aspnet/core/

OpenAPI Initiative. (n.d.). *The OpenAPI Specification*. OpenAPI Initiative. Recuperado en Noviembre de 2025, de https://www.openapis.org/

Oracle. (n.d.). *MySQL Documentation*. MySQL. Recuperado en Noviembre de 2025, de https://docs.oracle.com/cd/E17952_01/index.html

Postman. (n.d.). *Postman documentation overview*. Postman Learning Center. Recuperado en Noviembre de 2025, de https://learning.postman.com/docs/introduction/overview/