**Evaluación Técnica – Sistema E-Commerce Modular**

**Autor:** Ing. MA. Bryan Josué Xol Muñoz

**Fecha:** *15 de octubre de 2025*

**1.1 Introducción general**

El presente documento describe de forma detallada la planificación, desarrollo e implementación de una arquitectura modular de comercio electrónico (E-Commerce Modular System) construida bajo principios de diseño limpio, desacoplamiento funcional y despliegue escalable.

El propósito de la evaluación técnica fue demostrar la capacidad para diseñar e implementar un sistema distribuido compuesto por microservicios independientes, un *reverse proxy* de integración, y dos interfaces de usuario: una aplicación web (Angular) y una aplicación móvil/PWA (Ionic).

El enfoque se basó en la creación de un ecosistema mantenible, reutilizable y seguro, en el cual cada componente cumple un rol claro dentro del flujo general de negocio: autenticación, catálogo de productos, y gestión de órdenes.

**1.2 Arquitectura del sistema**

La arquitectura sigue el paradigma de microservicios desacoplados, donde cada módulo tiene responsabilidad única.  
A nivel alto, el sistema se compone de:

* **AuthSvc:** Servicio de autenticación y autorización. Gestiona usuarios, roles, contraseñas y emisión de tokens JWT.
* **CatalogSvc:** Servicio de productos y categorías. Expone endpoints de búsqueda, filtrado y administración de inventario.
* **OrderSvc:** Servicio de carrito persistente y checkout. Gestiona las órdenes de compra, totales y pagos.
* **Gateway / Reverse Proxy:** Punto de entrada unificado. Implementa validación de tokens, enrutamiento, logging y CORS.
* **Frontends:**
  + **Angular Web App:** Aplicación web responsiva con dashboard y vista de productos.
  + **Ionic App:** Aplicación móvil/PWA conectada al Gateway con almacenamiento local y autenticación JWT.
* **Base de datos:** **Microsoft SQL Server**, con procedimientos almacenados (SPs) optimizados y normalización 3FN.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Esta arquitectura permite aislar fallos, desplegar módulos en contenedores independientes y escalar horizontalmente cada servicio según la demanda.

**1.3 Flujo general de comunicación**

El flujo principal de interacción se organiza en torno al Gateway:

1. El usuario accede mediante el frontend web (Angular) o la app móvil (Ionic).
2. Envía credenciales a /api/v1/auth/login; el AuthSvc valida los datos y devuelve un par de tokens (access y refresh).
3. El Gateway valida el token JWT y enruta la solicitud hacia el microservicio correspondiente.
4. CatalogSvc devuelve los productos disponibles con filtros de búsqueda y paginación.
5. OrderSvc mantiene el carrito de compras y procesa el flujo de checkout.
6. Todos los servicios persisten su información en SQL Server bajo diferentes esquemas de datos.

Pantalla de juego de computadora

El contenido generado por IA puede ser incorrecto.

Este flujo garantiza trazabilidad, validación centralizada y seguridad homogénea en todas las peticiones del sistema.

**1.4 Modelo de datos y procedimientos SQL**

El modelo de datos fue diseñado en torno a tres dominios principales:

* Autenticación: Usuarios, roles, tokens y auditoría.
* Catálogo: Categorías, productos e inventario.
* Órdenes: Carritos, ítems y pedidos.

Cada microservicio interactúa con la base de datos mediante procedimientos almacenados (usp\_...) que aseguran integridad referencial y encapsulan la lógica de negocio.

Ejemplos:

* usp\_Cart\_AddOrMergeItem: Agrega o actualiza ítems en el carrito.
* usp\_Order\_Checkout: Realiza el cierre de compra y genera encabezado/detalle de orden.
* usp\_Product\_GetAll: Retorna productos activos con filtros por categoría o texto.

Diagrama

El contenido generado por IA puede ser incorrecto.

Cada SP utiliza transacciones (BEGIN TRAN / COMMIT / ROLLBACK) para garantizar atomicidad, y parámetros tipados para mantener seguridad contra inyección SQL.  
El modelo fue normalizado hasta tercera forma normal, con claves foráneas y restricciones para asegurar consistencia en las relaciones.

**1.5 Seguridad y control de acceso**

La seguridad se implementó con una política de múltiples capas:

* **JWT (JSON Web Token):** tokens firmados con clave simétrica (HS256) que contienen claims estándar (sub, email, role, exp).
* **Refresh Tokens:** mecanismo de reautenticación sin exponer credenciales.
* **Hashing de contraseñas:** mediante **bcrypt/PBKDF2** con salt aleatorio.
* **CORS restringido:** solo dominios oficiales (https://app.ecommerce.gt, https://m.ecommerce.gt).
* **TLS/HTTPS:** cifrado extremo a extremo.
* **Validaciones a nivel de gateway:** middleware de autorización para rutas protegidas.

Estas medidas garantizan un cumplimiento con principios OWASP y prácticas seguras de autenticación.

**1.6 Pruebas y validaciones**

El ciclo de pruebas incluyó los siguientes niveles:

1. **Pruebas unitarias:** sobre endpoints críticos de AuthSvc, CatalogSvc y OrderSvc.
2. **Pruebas de integración:** flujo completo desde login → catálogo → carrito → checkout.
3. **Pruebas E2E (End-to-End):** ejecución con datos reales y validación del ciclo completo.
4. **Pruebas de carga:** mediante herramientas como Postman y k6 para validar rendimiento bajo concurrencia.

**1.7 Beneficios técnicos alcanzados**

* Arquitectura desacoplada y modular.
* Despliegue independiente de cada microservicio.
* Alta escalabilidad y mantenibilidad.
* Seguridad reforzada con JWT y tokens refresh.
* Reutilización de código entre frontend web y móvil.
* Base de datos optimizada mediante SPs transaccionales.

**2. Parte 2: Gestión y Metodologías**

**2.1 Principios y valores de Scrum**

El desarrollo siguió el marco **Scrum**, promoviendo entregas incrementales, autoorganización del equipo y mejora continua.

**Valores de Scrum:**

* Compromiso.
* Enfoque.
* Apertura.
* Respeto.
* Coraje.

**Roles principales:**

* **Product Owner (PO):** define la visión del producto y prioriza el backlog.
* **Scrum Master (SM):** facilita ceremonias y remueve impedimentos.
* **Equipo de desarrollo:** implementa los incrementos funcionales.

**Artefactos clave:** Product Backlog, Sprint Backlog e Incremento.  
**Ceremonias:** Sprint Planning, Daily Scrum, Sprint Review y Retrospective.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**2.2 Plan de Sprints**

|  |  |  |
| --- | --- | --- |
| **Sprint** | **Duración** | **Objetivos principales** |
| 1 | 2 semanas | Diseño de arquitectura, configuración del entorno, creación de repositorio y pipeline inicial. |
| 2 | 2 semanas | Implementación de AuthSvc, endpoints de autenticación y seguridad JWT. |
| 3 | 2 semanas | Desarrollo de CatalogSvc y vistas de productos en Angular. |
| 4 | 2 semanas | Desarrollo de OrderSvc y flujo de checkout. |
| 5 | 1 semana | Integración final, pruebas, despliegue y retrospectiva. |

Cada sprint finaliza con una entrega funcional y revisión conjunta con el Product Owner.

**2.3 Uso de Jira y gestión de tareas**

El flujo de trabajo se administró mediante **Jira Software**, estructurado bajo metodología Scrum con tableros dinámicos.

* **Columnas:** *Backlog → To Do → In Progress → Code Review → Done*.
* **Tipos de issues:** Epic, Story, Task, Sub-task, Bug.
* **Roles:** PO (prioriza), SM (monitorea), Dev (ejecuta).
* **Notificaciones:** integradas con Slack para alertas automáticas.

Diagrama, Aplicación

El contenido generado por IA puede ser incorrecto.

Este sistema permitió una trazabilidad total del progreso, reduciendo tiempos de seguimiento y asegurando cumplimiento de objetivos por sprint.

**2.4 Liderazgo y comunicación**

El liderazgo técnico se enfocó en motivar, coordinar y comunicar eficazmente entre los integrantes del equipo, siguiendo principios de liderazgo servicial.

* **Comunicación diaria (Daily):** sincronización rápida y resolución de bloqueos.
* **Revisiones semanales (Weekly Sync):** avance global con PO y stakeholders.
* **Comunicación asíncrona:** Slack y Jira para seguimiento constante.
* **Retrospectivas:** identificación de mejoras continuas y aprendizajes.

Diagrama

El contenido generado por IA puede ser incorrecto.

**2.5 Conclusiones y recomendaciones técnicas**

* Implementar contenedores Docker para aislar servicios y facilitar despliegue.
* Escalar infraestructura con Kubernetes o AWS ECS.
* Incorporar observabilidad (Prometheus + Grafana).
* Ampliar módulo de reportes y analítica.
* Aplicar políticas de versionado semántico en endpoints públicos.
* Mejorar el ciclo de pruebas E2E y automatización QA.