Optimización y Rendimiento Desarrollo Software

Ingeniería Software II Grupo: Ar

Diego Fernando Rodríguez Arauz

Emerson Andrés Vargas Torres

Danny Livzka Vargas Rico

Kevin Camilo Arias Lizarazo

Yohan Alexander Rangel Méndez

Investiga como se optimiza el Back, el Front y las Bases de Datos.

1. Optimización del Back-End

La capa de servidor o lógica de negocio debe ser eficiente en el uso de recursos, garantizar la escalabilidad y asegurar tiempos de respuesta mínimos.

• **Programación asincrónica y uso de hilos:** Mejora la eficiencia bajo carga, con tecnologías como Node.js (event-loop), Spring WebFlux, o Golang (goroutines).

• **Caching de datos:** Utilización de Redis o Memcached para almacenar datos frecuentemente accedidos, reduciendo la latencia de las respuestas.

• **Compresión de respuestas:** GZIP o Brotli para reducir el tamaño de los paquetes enviados al cliente.

• **Arquitectura escalable:** Separación por microservicios, uso de contenedores (Docker) y orquestación (Kubernetes).

• **Uso de API Gateway:** Permite controlar la agregación de servicios, aplicar autenticación y limitar peticiones innecesarias.

Pruebas Automatizadas en Back-End

• **Pruebas Unitarias (JUnit, Mocha, Jest):** Validan el comportamiento de cada módulo individualmente.

• **Pruebas de Carga (JMeter, Gatling):** Evalúan el rendimiento del servidor bajo distintas cargas.

• **Integración Continua:** Jenkins, GitHub Actions y GitLab CI permiten ejecutar pruebas automáticamente en cada commit.

2. Optimización del Front-End

El Front-End debe responder rápidamente, consumir pocos recursos del navegador y garantizar una experiencia fluida.

• **Minimización y empaquetado:** Uso de Webpack, Vite, Rollup para reducir tamaño de archivos.

• **Lazy loading:** Carga diferida de módulos, imágenes y rutas, mejorando el Time To Interactive.

• **Formatos modernos de imagen:** WebP o AVIF para reducir el peso de las imágenes.

• **Reducción de renders innecesarios:** Mediante técnicas como memoización (React.memo) y DOM virtual.

• **Aplicaciones Progresivas (PWA):** Uso de Service Workers para cachear recursos y funcionar offline.

Pruebas Automatizadas en Front-End

• **Pruebas Unitarias (Jest, Vitest):** Verifican componentes individuales.

• **Pruebas de Integración (React Testing Library, Cypress):** Evalúan el comportamiento de múltiples componentes juntos.

• **Lighthouse:** Herramienta de Google para auditar el rendimiento y accesibilidad de la interfaz.

3. Optimización de Bases de Datos

Una base de datos eficiente es clave para mantener la rapidez en el acceso a la información.

• **Modelado adecuado:** Normalización eficiente, sin llegar a extremos que compliquen las consultas.

• **Índices:** Creación de índices en columnas claves, uso de EXPLAIN y ANALYZE para evaluar consultas.

• **Consultas eficientes:** Evitar SELECT *, usar JOINs con criterios específicos y limitar resultados.

• **Caching de consultas:** Redis, views materializadas y cache en middleware.

• **Pooling de conexiones:** Uso de HikariCP, pgpool, etc., para evitar saturación del servidor de base de datos.

• **Monitoreo y mantenimiento:** Reindexado, vacuum, estadísticas de uso.

Pruebas Automatizadas en Bases de Datos

• **Pruebas de rendimiento:** Scripts automatizados con JMeter o QueryBench para verificar tiempos de respuesta.

• **Validación de integridad:** Scripts de CI/CD para verificar consistencia de claves foráneas y duplicados.

• **Monitoreo continuo:** Herramientas como pgBadger, Percona Monitoring o New Relic.

Implementa optimización para su proyecto, y realiza un informe para los resultados

1. Introducción

El presente documento tiene como finalidad detallar las estrategias de optimización y mejora de rendimiento implementadas en el proyecto "Sistema de Gestión de Inventario para La Casa del Embrague RJC". Este sistema, desarrollado con una arquitectura de microservicios, integra tecnologías modernas como Spring Boot, React.js, Docker, MariaDB, PostgreSQL y MongoDB. En el contexto de un e-commerce automotriz con múltiples módulos interdependientes, la eficiencia y el tiempo de respuesta son factores clave para asegurar una operación fluida, fiable y escalable.

La optimización del sistema se aborda desde tres frentes: el backend, responsable de la lógica de negocio y servicios; el frontend, que define la experiencia de usuario; y la gestión de las bases de datos, encargadas de la persistencia y consulta de información estructurada y no estructurada. Cada uno de estos componentes se analiza en profundidad con el objetivo de garantizar un rendimiento estable, tiempos de carga mínimos, y capacidad para escalar horizontalmente ante aumento de la demanda. A través de esta optimización, se busca asegurar la continuidad del negocio, una experiencia positiva para el usuario final, y una base tecnológica preparada para la evolución futura del sistema.

2. Optimización del Back-End

La parte posterior del sistema, desarrollada con Java y Spring Boot, ha sido optimizada bajo principios de asincronismo, eficiencia en el manejo de peticiones y desacoplamiento de servicios. Se ha adoptado WebFlux para gestionar procesos concurrentes con mejor aprovechamiento de los recursos. Para reducir tiempos de respuesta y tráfico innecesario, se habilitó la compresión GZIP en las respuestas HTTP y se configuró la serialización de objetos para transmitir únicamente los atributos necesarios, reduciendo el tamaño del payload.

Se implementó Redis como sistema de caché para almacenar temporalmente resultados de consultas repetitivas, como listados de productos y configuraciones globales. Esto permite evitar accesos innecesarios a la base de datos y mejora el rendimiento hasta

en un 60% en pruebas locales. Asimismo, se implementó un sistema de logging centralizado con trazabilidad por microservicio, permitiendo identificar cuellos de botella y analizar tiempos de respuesta en operaciones críticas. Para mitigar la sobrecarga, los servicios fueron desacoplados mediante colas de eventos utilizando una arquitectura orientada a eventos (EDA) como opción futura de mejora.

Se integró GitHub Actions como plataforma de integración continua (CI), automatizando las pruebas unitarias (JUnit), los builds del backend y los despliegues en contenedores Docker. Se añadieron políticas de revisión de código y validación de cobertura para garantizar que cada modificación al sistema mantenga los estándares de calidad. Además, el uso de un API Gateway permite enrutar peticiones, gestionar autenticación y balancear carga entre microservicios, mejorando tanto la seguridad como el desempeño y escalabilidad general del backend.

3. Optimización del Front-End

La interfaz de usuario ha sido construida con React.js y está enfocada en ofrecer una experiencia rápida y fluida. Para ello, se configuró Webpack para generar bundles optimizados, minificando el código y eliminando dependencias no utilizadas (*tree shaking*). El sistema implementa lazy loading de componentes para reducir la carga inicial, cargando vistas solo cuando el usuario navega a ellas. Esta práctica mejora sustancialmente el tiempo de primera interacción en dispositivos móviles o de gama baja.

Se emplean técnicas como React.memo, useMemo y useCallback para controlar los re-renderizados innecesarios. Esto permite reducir el trabajo del motor de renderizado de React y evitar recomputaciones que ralentizan la interfaz. Las imágenes del sitio se han convertido al formato WebP y se incorporó carga diferida (lazy loading) en los elementos visuales para agilizar el renderizado de las vistas. Además, se configuró un servicio de cache local con IndexedDB para conservar parte del estado entre sesiones.

Se realizaron auditorías periódicas de rendimiento usando Lighthouse, detectando oportunidades de mejora en accesibilidad, mejores prácticas y carga de scripts. También se adaptó la interfaz como una Progressive Web App (PWA), permitiendo que funciones básicas como la consulta de inventario y registro de ventas estén disponibles sin conexión

activa. El sistema frontend cuenta con pruebas automatizadas desarrolladas con Jest y pruebas end-to-end con Cypress, validando tanto la funcionalidad como el rendimiento en distintos dispositivos y resoluciones.

4. Optimización de Bases de Datos

El sistema emplea una estrategia híbrida de persistencia que combina bases relacionales (PostgreSQL, MariaDB) con una base NoSQL (MongoDB). En las bases relacionales, se aplicaron técnicas de modelado eficiente con normalización controlada, uso de claves primarias simples, relaciones explícitas con claves foráneas y restricciones para mantener la integridad de los datos. Se evitaron consultas con SELECT * y se implementaron índices en columnas de búsqueda y filtrado frecuentes, como código de producto, categoría y estado de inventario.

Se configuró HikariCP como pool de conexiones JDBC, lo que permite una gestión eficiente de sesiones de base de datos, reduciendo el tiempo de establecimiento de conexiones y evitando saturaciones. Para MongoDB, se estructuraron los documentos por tipo de evento y se establecieron TTLs (Time-To-Live) para limpieza automática de registros antiguos. Redis complementa esta arquitectura al ofrecer almacenamiento en memoria para resultados que se consultan repetidamente, permitiendo respuestas inmediatas.

Además, se implementaron rutinas automatizadas de mantenimiento como VACUUM y ANALYZE en PostgreSQL y MariaDB para optimizar la planificación de consultas. Se emplean herramientas de diagnóstico como pgAdmin, pgBadger y Percona Toolkit para monitoreo y ajuste de rendimiento. Se realizaron pruebas de carga con JMeter y scripts SQL diseñados para simular flujos reales de ventas, búsquedas de productos y actualizaciones de inventario, asegurando que las bases de datos puedan mantener tiempos de respuesta óptimos bajo cargas crecientes.