

**ASEGURAMIENTO CALIDAD SOFTWARE**  
**Proyecto Final**  
**Sistema de Gestión de Inventarios con QAS**

## Descripción General:

Desarrollar un **Sistema de Gestión de Inventarios** enfocado en pequeñas y medianas empresas, aplicando de forma rigurosa todas las **etapas del ciclo de vida del aseguramiento de la calidad del software (QAS)**. El proyecto debe garantizar funcionalidades sólidas, experiencia de usuario amigable y estándares profesionales en seguridad, pruebas, documentación y despliegue.

## Alcance Funcional

### ~~1. Gestión de Productos~~

- ~~a. **Agregar Producto:** Permitir a los usuarios agregar nuevos productos al inventario, incluyendo detalles como nombre, descripción, categoría, precio y cantidad inicial, stock mínimo.~~
- ~~b. **Editar Producto:** Permitir la edición de la información de un producto existente.~~
- ~~c. **Eliminar Producto:** Permitir la eliminación de productos del inventario.~~
- ~~d. **Visualizar Productos:** Mostrar una lista de todos los productos en el inventario con opciones de búsqueda y filtrado.~~

### ~~2. Control de Stock~~

- ~~a. **Actualizar Stock:** Permitir la actualización de la cantidad de productos en el inventario (entrada y salida de productos).~~
- ~~b. **Alertas por stock mínimo:** mostrar cuando un producto llega al stock mínimo establecido~~
- ~~c. **Historial de Movimientos:** Mantener un registro de todas las entradas y salidas de productos, incluyendo fechas y usuarios responsables.~~

### ~~3. Integración con Otros Sistemas~~

- ~~a. **API de Integración:** Proporcionar una API para integrar el sistema con otras aplicaciones (por ejemplo, sistemas de contabilidad o puntos de venta).~~

### ~~4. Interfaz de Usuario Amigable~~

- ~~a. **Dashboard:** Proveer un tablero de control con una visión general del estado del inventario y estadísticas clave.~~
- ~~b. **Usabilidad:** Asegurar que la interfaz sea intuitiva y fácil de usar, con una navegación clara y accesible.~~

# Roles y Niveles de Acceso

## 1. ~~Administrador~~

- ~~a. Descripción: Tiene acceso completo a todas las funcionalidades del sistema.~~
- ~~b. Permisos:~~
  - ~~i. Gestión de Productos: Agregar, editar, eliminar y visualizar productos.~~
  - ~~ii. Control de Stock: Actualizar stock y visualizar historial de movimientos.~~

## 2. Empleado

- ~~a. Descripción: Tiene acceso a las funcionalidades básicas para gestionar productos y actualizar el stock. No puede realizar configuraciones críticas.~~
- b. Permisos:
  - i. Gestión de Productos: Agregar, editar y visualizar productos (no puede eliminar).
  - ~~ii. Control de Stock: Actualizar stock y visualizar historial de movimientos.~~

## 3. Usuario Invitado/Cliente

- a. Descripción: Tiene acceso muy limitado, principalmente para visualizar información básica. Ideal para clientes que necesitan verificar el stock de ciertos productos.
- b. Permisos:
  - i. Visualización de Productos: Ver lista de productos y detalles básicos (sin agregar, editar o eliminar).
  - ii. Reportes de Inventario: Ver reportes públicos o básicos (sin acceso a reportes detallados o de movimientos).

Funcionalidad	Administrador	Empleado	Invitado
Gestión de Productos	CRUD	CRUD	R
Control de Stock	CRUD	CRUD	-
API de Integración	JWT	-	-

# Etapas del Ciclo de Vida del Aseguramiento de la Calidad del Software (QAS)

## 1. ~~Planificación y Gestión de Proyectos:~~

- ~~a. Plan de Proyecto: Definir claramente el alcance, los objetivos, los entregables y el cronograma del proyecto.~~
- ~~b. Gestión de Tareas: Utilizar herramientas como Jira, Trello o Asana para gestionar las tareas del proyecto y realizar un seguimiento del progreso.~~
- ~~c. Gestión de Riesgos: Identificar y documentar los posibles riesgos y planificar estrategias de mitigación.~~

## 2. Pruebas de Calidad:

- ~~a. Pruebas de Seguridad: Realizar pruebas de penetración y análisis de vulnerabilidades para asegurar que el sistema sea seguro contra amenazas externas.~~

- ~~b. **Pruebas de Usabilidad:** Evaluar la interfaz de usuario y la experiencia del usuario (UX) para asegurar que el sistema sea intuitivo y fácil de usar.~~
- ~~c. **Pruebas de Compatibilidad:** Asegurar que el sistema funcione correctamente en diferentes navegadores y dispositivos.~~
- ~~d. **Pruebas de Regresión:** Implementar pruebas automatizadas que aseguren que las nuevas modificaciones no introduzcan errores en el sistema existente.~~
- ~~e. **Pruebas de Estrés:** Utilizar herramientas como JMeter para evaluar el rendimiento del sistema bajo carga, asegurando que el sistema pueda manejar altos volúmenes de tráfico y transacciones sin degradarse.~~
- ~~f. **Pruebas de Aceptación:** Implementar pruebas de aceptación basadas en Cucumber para asegurar que el sistema cumple con los requisitos del usuario.~~
- ~~g. **Pruebas de Navegadores:** Utilizar Playwright para realizar pruebas automatizadas en diferentes navegadores y dispositivos, asegurando la compatibilidad.~~

### ~~3. Documentación:~~

- ~~a. **Documentación de Requisitos:** Crear un documento detallado de requisitos funcionales y no funcionales.~~
- ~~b. **Documentación Técnica:** Incluir diagramas de arquitectura, guías de instalación y manuales de mantenimiento.~~
- ~~c. **Guía de Pruebas:** Documentar los casos de prueba, los resultados y cualquier defecto encontrado.~~

### ~~4. Revisión y Validación:~~

- ~~a. **Revisiones de Código:** Implementar revisiones de código (code reviews) para asegurar la calidad y la consistencia del código.~~
- ~~b. **Validación de Requisitos:** Asegurar que los requisitos del sistema sean claros, completos y verificables.~~

### 5. Integración Continua y Despliegue Continuo (CI/CD):

- a. **Pipeline de CI/CD:** Configurar un pipeline robusto que incluya la construcción automática, pruebas unitarias, pruebas de integración, y despliegue a ambientes de prueba y producción. Utilizar GitHub Actions para gestionar el pipeline de CI/CD.
- b. **Entorno de Pruebas:** Configurar entornos de prueba que simulen de manera realista el entorno de producción.
- c. **Contenedorización:** Utilizar herramientas como Docker o Podman para crear contenedores del sistema que faciliten la consistencia entre diferentes entornos de desarrollo, prueba y producción.
- ~~d. **Migración de Base de Datos:** Implementar herramientas como Flyway o Liquibase para gestionar las migraciones de bases de datos de manera segura y automatizada.~~

### 6. Monitoreo y Mantenimiento:

- ~~a. **Monitoreo en Producción:** Implementar herramientas de monitoreo para verificar la salud del sistema en producción y detectar problemas en tiempo real.~~

- b. ~~Observabilidad:~~ Utilizar ~~OpenTelemetry~~ para instrumentar el sistema y recolectar métricas, trazas y logs, facilitando el monitoreo y la resolución de problemas.
- c. ~~Mantenimiento y Actualizaciones:~~ Planificar un ciclo regular de mantenimiento y actualizaciones para asegurar que el sistema permanezca seguro y funcional.

## ~~7. Gestión de Calidad:~~

- a. ~~Indicadores de Calidad:~~ Definir métricas de calidad del software (como cobertura de pruebas, densidad de defectos, tiempo de respuesta) y realizar un seguimiento de estas métricas.
- b. ~~Mejora Continua:~~ Implementar un proceso de mejora continua basado en retroalimentación y lecciones aprendidas.

## ~~8. Evaluación Post-Implementación:~~

- a. ~~Revisión Post-Mortem:~~ Realizar una revisión post-mortem del proyecto para identificar qué se hizo bien y qué se puede mejorar en futuros proyectos.

## Tecnologías Sugeridas:

- ~~Backend:~~ Spring Boot, Quarkus, Django, Node.js, Next.js
- ~~Frontend:~~ Hilla, Vaadin Flow, React.js, Vue.js, Angular, JHipster
- ~~Base de Datos:~~ MySQL, PostgreSQL, MariaDB
- ~~Auditoría:~~ Hibernate Envers
- ~~Autenticación y Autorización:~~ OAuth2, JWT, Keycloak
- Pruebas: JUnit/TestNG, Selenium, Cucumber, JMeter, Playwright, Microcks
- ~~CI/CD:~~ GitHub Actions, Jenkins, GitLab CI.
- ~~Contenedorización:~~ Docker, Podman.
- ~~Migración de Base de Datos:~~ Flyway, Liquibase.
- Monitoreo y Observabilidad: Prometheus, Grafana, Open Telemetry

## Evaluación y Reglas del Proyecto:

- El proyecto debe subirse a un repositorio GitHub público compartido con el docente.
- La participación será evaluada según los commits individuales, ramas, issues y pull requests.
- Ambos (máximos 2) integrantes deben demostrar participación equitativa.
- Se valorará la creatividad, calidad de código, uso adecuado de herramientas y calidad de documentación.
- Se debe hacer una presentación final funcional del sistema en clase.

## Conclusión:

Este proyecto es una oportunidad integral para aplicar los conceptos de calidad del software en un contexto realista y práctico. El objetivo es no solo construir una aplicación funcional, sino también garantizar que sea segura, confiable, mantenible y usable, tal como lo requiere el desarrollo profesional de software.