**PROYECTO FINAL FUNDAMENTOS DE INGENIERIA DE SOFTWARE.**

desarrollar un e-business de una ferretería con herramientas de gestión de proyectos y software. La práctica incluye requisitos, configuraciones de Jira y GitHub, un diseño de software simple, pruebas, y mantenimiento.

**1. Requerimientos para el e-business de la ferretería (UTILIZA LA HERRAMIENTA DE JIRA PARA ADMINISTRAR LA FORMALIDAD DE LOS REQUERIMIENTOS)**

Define un **e-business** de una ferretería que permita a los usuarios:

1. **Registro y autenticación de usuarios**: Permitir a los clientes registrarse y autenticarse.
2. **Catálogo de productos**: Mostrar los productos de la ferretería con sus descripciones, precios y disponibilidad.
3. **Carrito de compras**: Facilitar la selección y compra de productos.
4. **Sistema de pagos**: Implementar un sistema de pagos en línea.
5. **Gestión de inventario**: Para el administrador, gestión y actualización del stock.
6. **Notificaciones**: Enviar confirmación de pedido y notificaciones de entrega.
7. **Historial de compras**: Permitir a los usuarios consultar sus pedidos anteriores.
8. **Opiniones y calificaciones**: Facilitar comentarios de usuarios sobre productos.

**2. Configuración de Jira para Gestión de Proyectos**

**Objetivo**: Configurar Jira para el seguimiento y gestión de tareas en el proyecto.

1. **Crear un proyecto en Jira**
   * Nombre: **E-business Ferretería**
   * Tipo de proyecto: Software Development
2. **Definir el backlog de tareas**: Crear historias de usuario en el backlog con subtareas. Ejemplos:
   * Como usuario, quiero poder registrar una cuenta para comprar productos.
   * Como administrador, quiero gestionar el inventario.
   * Como usuario, quiero poder añadir productos al carrito y proceder al pago.
3. **Configurar el tablero Kanban/Scrum**:
   * Divide el tablero en columnas: Backlog, Diseño, En Desarrollo, Pruebas, Finalizado y Mantenimiento.
   * Asigna responsables y estima el tiempo de cada tarea.
4. **Integrar tareas con GitHub**:
   * Configura Jira para vincular cada tarea a una rama específica en GitHub para que el progreso del código se refleje en el tablero de Jira.
5. **Establecer reuniones de equipo (Scrum diario, Sprint review y Sprint retrospective)** para la revisión de progreso (desde el día en que se otorga la práctica hasta el **21 de noviembre**.)

**3. Gestión del Código con GitHub**

**Objetivo**: Gestionar el código y coordinar el trabajo en equipo usando GitHub.

1. **Crear un repositorio en GitHub**
   * Nombre: **e-business-ferreteria**
   * Colaboradores: Añadir a todos los integrantes del equipo como colaboradores.
2. **Estructura del repositorio**:
   * main branch: para la versión de producción.
   * develop branch: para el desarrollo y pruebas.
   * Crear ramas de características (feature/nombre-de-la-funcionalidad) para el desarrollo de cada historia de usuario.
3. **Integración con Jira**: Vincular el repositorio con Jira para rastrear el progreso de cada tarea.
4. **Pull Requests**:
   * Todos los cambios deben pasar por una Pull Request y una revisión de código antes de ser fusionados en la rama principal.

**4. Instrucciones para un Diseño de Software Sencillo**

**Objetivo**: Crear una arquitectura de software básica que cubra los requisitos.

1. **Diagrama de arquitectura**:
   * Define una arquitectura MVC (Modelo-Vista-Controlador).
   * Componentes principales:
     + **Frontend**: HTML, CSS, JavaScript, con un framework como React o Vue.
     + **Backend**: API RESTful con Node.js o Django.
     + **Base de datos**: SQL (MySQL o PostgreSQL) o NoSQL (MongoDB).
2. **Diagrama de clases**:
   * Diseñar las clases o módulos básicos:
     + Producto, Usuario, Pedido, Inventario, Pago.
   * Define relaciones entre clases, como Usuario puede tener múltiples Pedidos.
3. **Diagrama de flujo de datos**:
   * Muestra el flujo de los datos desde el cliente (usuario) al servidor y la base de datos.
   * Incluye los procesos de registro de usuario, gestión de productos, proceso de compra y pagos.

**5. Instrucciones para Generar Escenarios de Pruebas de Software**

**Objetivo**: Crear escenarios de prueba para asegurar la calidad del software.

1. **Pruebas de unidad**: Cada componente debe tener pruebas unitarias. Ejemplos:
   * Probar el método de autenticación de usuario.
   * Probar el cálculo del total en el carrito de compras.
2. **Pruebas de integración**: Verifica que los módulos trabajen bien juntos.
   * Prueba la integración entre el carrito y el sistema de pagos.
   * Verifica que el inventario se actualice después de cada compra.
3. **Pruebas funcionales**: Basadas en los requisitos.
   * Probar el flujo de registro y login de usuario.
   * Probar el proceso completo de añadir un producto al carrito y realizar una compra.
4. **Pruebas de interfaz de usuario (UI)**: Verifica que los elementos se muestren y funcionen correctamente en diferentes navegadores y dispositivos.
   * Prueba el diseño responsivo y la funcionalidad de los botones.
5. **Pruebas de rendimiento**:
   * Carga: Prueba cómo el sistema se comporta con varios usuarios accediendo simultáneamente.
   * Respuesta: Mide el tiempo de respuesta del sistema.

**6. Gestión del Mantenimiento**

**Objetivo**: Planificar y documentar el mantenimiento para asegurar el funcionamiento continuo.

1. **Mantenimiento Correctivo**:
   * Documentar todos los errores identificados y el procedimiento para solucionarlos.
   * Asignar a un miembro del equipo para revisar y resolver los errores de manera regular (Establecer su gestión de resolución de errores y los momentos en que debe realizarlos.)
2. **Mantenimiento Preventivo**:
   * Programar revisiones periódicas de código para eliminar dependencias obsoletas.
   * Implementar medidas de seguridad para proteger la base de datos.
3. **Mantenimiento Perfectivo**:
   * Recoger retroalimentación de usuarios sobre mejoras posibles en la interfaz o en la funcionalidad (Establecer los artefactos para esta obtener esta información por parte de los usuarios.)
   * Realizar ajustes para mejorar la experiencia del usuario o la eficiencia del sistema (Generar escenarios de estas posibles mejoras.)
4. **Mantenimiento Adaptativo**:
   * Monitorear cambios en tecnologías (nuevas versiones de software y frameworks) y ajustar el sistema.
   * Preparar el sistema para adaptarse a nuevos requisitos legales (por ejemplo, normas de protección de datos).

**Entregables**

* **Documentación del proyecto** (**requerimientos** en la formalidad de su registro, **diseño**, **Reporte de pruebas de software** documentando los escenarios de las pruebas, **Plan de mantenimiento** documentado).
* **Configuración de Jira** con el tablero de tareas actualizado.
* **Repositorio de GitHub** solo nombres de los archivos y realizar actualizaciones cada día como si realizaran una nueva version.)