



**Semana 3**

**Taller Aplicado de Software**

Formato de respuesta

|  |  |
| --- | --- |
| **Nombre estudiante:** | **Giovanny Otarola**  **Matías Fuentes** |
| **Asignatura: Taller aplicado de software** | **Carrera: Ingeniería en desarrollo de software** |
| **Profesor: Carlos Valverde** | **Fecha:08-06-2025** |

# Descripción de la actividad En esta actividad sumativa en parejas deberás consolidar la información recabada durante las semanas 1 y 2 aplicando las mejoras en base a los comentarios realizados por tu docente. Adicionalmente deberás elaborar el plan de trabajo que detalle las actividades necesarias para la realización de tu proyecto.

## Instrucciones específicas

Copia y pega la información de la semana 1 y 2 en este formato de respuesta. Posteriormente, elabora el plan de trabajo de tu proyecto utilizando la tabla que se presenta a continuación.  
**Si tienes alguna consulta, no dudes en contactar a tu docente.**

# Semana 1

### Parte I

|  |  |
| --- | --- |
| **1. Nombre del proyecto** | ***“Gestión Inteligente de Recursos Tecnológicos” (GIRET)*** |

|  |
| --- |
| **2. Descripción Proyecto** |
| Este proyecto tiene como objetivo desarrollar una aplicación web para la gestión de recursos tecnológicos en instituciones, orientada a resolver las necesidades específicas de los departamentos de Tecnologías de la Información (TI). Estos equipos enfrentan desafíos en el control de inventario, seguimiento del estado de los activos y trazabilidad documental, muchas veces por depender de sistemas manuales o no integrados.  La plataforma permitirá un control centralizado del inventario, con seguimiento en tiempo real del estado de cada recurso (asignado, en bodega, prestado, etc.) y alertas automáticas sobre plazos críticos, como vencimiento de garantías o baja de stock. Además, facilitará la gestión documental mediante la carga de archivos PDF asociados a cada elemento.  Se integrará autenticación mediante AWS Cognito, asegurando que cada usuario del área TI acceda solo a la información correspondiente. La solución también incluirá tecnologías modernas como Azure Functions para gestión de eventos, AWS Api Gateway para protección de las apis y AWS S3 para el almacenamiento seguro y escalable de archivos. |

|  |
| --- |
| **3. Fundamentación Proyecto** |

|  |  |
| --- | --- |
| **Relevancia del proyecto** | La administración ineficiente de recursos tecnológicos es un problema recurrente en instituciones de todo tipo, provocando pérdidas materiales, duplicación de gastos y falta de trazabilidad. La propuesta nace como respuesta a esta problemática, con el objetivo de centralizar, automatizar y controlar el ciclo de vida de estos recursos. Esto es especialmente relevante en el contexto educativo, donde múltiples departamentos y usuarios comparten notebooks, proyectores, licencias, entre otros insumos tecnológicos.  Este sistema está pensado para ser implementado inicialmente en una institución de educación en donde la asignación y préstamo de recursos tecnológicos es frecuente y actualmente se gestiona en forma manual o con herramientas parciales (como Excel o correo electrónico). El grupo impactado incluye encargados de infraestructura, personal administrativo y estudiantes o docentes que hacen uso de los recursos.  El proyecto representa un aporte real al entorno organizacional al reducir los errores humanos, optimizar los tiempos de gestión, asegurar la trazabilidad completa de los equipos, y permitir auditorías eficientes mediante un historial de acciones. Además, se alinea con las tendencias actuales de digitalización y automatización en el campo profesional del desarrollo de software. |
| **Factibilidad de desarrollo del Proyecto** | Este proyecto es factible de realizar dentro del periodo académico establecido. Se utilizarán tecnologías ampliamente conocidas y disponibles como Angular (frontend), AWS Api Gateway, Spring Boot (backend), AWS Cognito , Azure Functions y AWS S3 almacenamiento de archivos. La base de datos será Oracle Cloud.  **Factores que facilitan el desarrollo:**   * Conocimientos previos del equipo en tecnologías Java y Angular. * Acceso a entornos gratuitos o académicos de Azure y AWS. * División clara de tareas y enfoque modular del sistema.   **Factores que facilitan el desarrollo:**     * La curva de aprendizaje de algunas integraciones como Azure AD o AWS S3. * Tiempo limitado del bimestre.   **Mitigación:**   * Enfocarse en un MVP (Producto Mínimo Viable) funcional * Uso de documentación oficial y soporte comunitario. |
| **Mercado Objetivo** | El mercado objetivo incluye instituciones educativas, áreas de TI de empresas medianas y organismos públicos que administran recursos tecnológicos compartidos.  Los **usuarios finales de la aplicación** se dividen en dos perfiles principales:   * **Encargados de Soporte Técnico o Infraestructura TI:** responsables del registro, seguimiento, préstamo y mantenimiento de los recursos tecnológicos. Utilizarán la plataforma para gestionar el inventario, actualizar el estado de los equipos y administrar documentos relacionados (boletas, garantías, manuales, etc.). * **Personal Administrativo:** encargado de coordinar solicitudes de préstamo, validar devoluciones, y generar reportes de uso. Podrán consultar información detallada sobre disponibilidad y estado de los recursos.   Se apunta a organizaciones que requieren una solución simple, centralizada y eficiente para llevar control de inventario, generar alertas automáticas, mantener documentación asociada y garantizar la trazabilidad de los recursos tecnológicos en entornos con múltiples usuarios y unidades operativas. La solución es escalable y puede adaptarse a distintas realidades institucionales. |

|  |
| --- |
| **4. Objetivos** |
|  |

|  |  |
| --- | --- |
| **Objetivo general** | Desarrollar una aplicación web que permita gestionar de forma eficiente los recursos tecnológicos institucionales, integrando funcionalidades de inventario, seguimiento por estado, alertas automatizadas, control de usuarios, generación y descarga de reportes y carga de documentos asociados. |
| **Objetivos específicos** | * Implementar un módulo de inventario con visualización, edición y registro de recursos tecnológicos. * Diseñar un sistema de seguimiento de estado de los recursos con historial de movimientos. * Configurar alertas automáticas para préstamos vencidos o préstamos por vencer. * Integrar un sistema de autenticación y autorización con AWS Cognito. * Desarrollar la funcionalidad de carga y visualización de archivos PDF asociados a los recursos. |

# Semana 2

|  |
| --- |
| **5. Metodología** |
| Para el desarrollo del proyecto **GIRET (Gestión Inteligente de Recursos Tecnológicos)** se utilizará la **metodología Waterfall(Cascada)**. |

|  |
| --- |
| **Descripción de la Metodología** |
| El proyecto **GIRET (Gestión Inteligente de Recursos Tecnológicos)** surge como respuesta a la necesidad de contar con un sistema eficiente para la gestión de recursos tecnológicos en instituciones educativas, áreas de TI de empresas medianas y organismos públicos que administran recursos tecnológicos compartidos. Actualmente, la falta de dicho sistema genera diversos problemas como el desconocimiento del inventario disponible, pérdida de tiempo en la búsqueda de equipos o insumos, y una gestión documental ineficiente.  La oportunidad radica en desarrollar una **aplicación web centralizada** que permita optimizar este proceso, mejorando la visibilidad, control y administración de los recursos tecnológicos.  Para el desarrollo del proyecto se utilizará la **metodología de desarrollo de software en cascada**, un enfoque secuencial que se basa en una serie de etapas lineales: **Análisis de Requisitos, Diseño del Sistema, Codificación, Pruebas, Despliegue y Mantenimiento**. Esta metodología es adecuada para proyectos con requerimientos bien definidos desde el inicio, ya que permite una planificación clara y una estructura ordenada y documentada. Cada fase debe completarse antes de iniciar la siguiente, lo que facilita el seguimiento del progreso y es ideal para contextos académicos donde se requiere una documentación rigurosa del proceso.  Se desarrolla el sistema de acuerdo al diseño técnico. Se codifican los distintos módulos utilizando las tecnologías seleccionadas (Angular, Spring Boot, Oracle Cloud, AWS Api gateway, AWS Cognito, Azure Functions). Cada componente se construye respetando las funcionalidades esperadas.  **Pruebas del sistema**  Se realizan pruebas unitarias, funcionales e integradas. Se valida que cada módulo funcione correctamente y que la interacción entre ellos sea coherente. También se prueba la autenticación, el sistema de alertas y la carga de archivos.  **Despliegue y entrega**  Una vez finalizado el desarrollo y validadas las pruebas, se entrega el sistema para su revisión. Se presenta una versión funcional (MVP) y la documentación correspondiente (manuales, presentación, etc.).  **Mantenimiento (si aplica)**  Aunque en un contexto académico esta etapa suele ser opcional, se identifican posibles mejoras y futuras funcionalidades para ampliar el sistema en futuras versiones. Roles y responsabilidades del equipo **Integrante 1 – Matias Fuentes**   * Responsable del backend (lógica de negocio,API REST, lógica de negocio, integración con Oracle Cloud y Azure Function).   **Integrante 2 – Giovanny Otárola**   * Responsable del frontend (lógica de negocio,interfaz Angular, subida de archivos, conexión con API,AWS).   Ambos comparten las tareas de documentación, pruebas, planificación de etapas y presentación final del proyecto. |
| **Tecnologías a utilizar** | | |
| La aplicación de gestión de recursos tecnológicos se construirá sobre una base de tecnologías robustas y escalables para asegurar su funcionalidad, seguridad y mantenibilidad:   * **Autenticación y Usuarios:** Se usará **AWS Cognito** para una gestión de identidad segura. * **Backend:** **Spring Boot (Java)** será el corazón de la lógica de negocio y las APIs RESTful, garantizando servicios sólidos y escalables. * **Frontend:** **Angular** desarrollará una interfaz de usuario moderna, interactiva y responsiva. * **Pruebas de API:** **Postman** asegurará la fiabilidad de los servicios backend antes de la integración. * **Lógica Serverless:** **Azure Functions** gestionará tareas en segundo plano como notificaciones y alertas, mejorando la eficiencia. * **Almacenamiento de Archivos:** **AWS S3 Buckets** almacenará documentos de forma segura y escalable. * **Protección de Api: AWS Api Gateway** garantizará que las APIs estén seguras y solo accesibles para las solicitudes autorizadas provenientes del frontend. * **Alojamiento y Despliegue:** **AWS EC2** proporcionará la infraestructura para el almacenamiento de los artefactos y[**Fly.io**](http://fly.io) para despliegue mediante contenedores. * **Control de versiones:** **GitHub** facilitará la colaboración y el seguimiento del código. * **Contenedorización:** **Docker** empaquetará las aplicaciones para entornos consistentes y despliegues simplificados. * **Gestión de Proyectos:** **Trello** organizará las tareas y el flujo de trabajo del equipo de desarrollo.   Esta combinación de herramientas optimiza el desarrollo, despliegue y operación de una aplicación integral y eficiente. | | |
| **Arquitectura** | |
| La arquitectura de software para el proyecto de gestión de recursos tecnológicos se basará principalmente en un **patrón de microservicios** combinado con un **enfoque basado en eventos**. Esta elección estratégica nos permitirá construir un sistema altamente escalable, resiliente, mantenible y flexible, capaz de evolucionar y adaptarse a los requisitos cambiantes del negocio. Componentes y sus Relaciones La aplicación se descompondrá en servicios atómicos, autónomos y acoplados de forma flexible, cada uno responsable de una capacidad de negocio específica y comunicándose a través de APIs.  **Frontend (Angular):**   * **Componente:** Una aplicación de una sola página (SPA) que proporcionará la interfaz de usuario. * **Relación:** Interactúa con los microservicios del backend a través de **APIs RESTful** para la visualización y gestión de datos. La autenticación se manejará a través de **AWS Cognito**, que redirigirá al frontend una vez que el usuario haya iniciado sesión.   **Microservicios del Backend (Spring Boot):**   * **Componentes:** Múltiples servicios Spring Boot, cada uno encapsulando una funcionalidad de negocio. Ejemplos incluyen:   + - **Servicio de Gestión de Inventario:** Responsable de los recursos tecnológicos, su estado y características.     - **Servicio de Gestión de Usuarios y Roles:** Aunque la autenticación primaria la maneja AWS Cognito, este servicio podría gestionar la autorización interna y los perfiles de usuario detallados.     - **Servicio de Notificaciones y Alertas:** Enviará avisos sobre préstamos atrasados o por vencer.     - **Servicio de Gestión Documental:** Manejará los enlaces a los archivos almacenados en AWS S3. * **Relaciones:**   + - **APIs RESTful:** Los microservicios expondrán APIs para ser consumidos por el frontend y por otros microservicios.     - **Bases de Datos Dedicadas:** Cada microservicio tendrá su propia base de datos (o esquema), garantizando independencia de datos y facilitando la evolución. La elección de la base de datos (relacional o NoSQL) dependerá de los requisitos específicos de cada microservicio.     - **Comunicación Asíncrona (Basada en Eventos):** Utilizarán un bus de eventos (por ejemplo, a través de **Azure Functions** como event handlers) para la comunicación entre servicios. Por ejemplo, un cambio en el "estado de un recurso" en el Servicio de Inventario podría emitir un evento que el Servicio de Notificaciones y Alertas consuma para enviar una alerta automática.   **API Gateway (AWS):**   * **Componentes:** Un servicio gestionado que actúa como puerta de enlace para exponer y proteger las APIs RESTful de los microservicios. * **Relaciones:**   + - Todos los microservicios del backend publicarán sus endpoints RESTful detrás de AWS API Gateway.     - API Gateway maneja la entrada de todas las solicitudes del frontend (Angular) y de otros consumidores externos autorizados.   **Azure Functions (Event Handlers):**   * **Componentes:** Funciones serverless diseñadas para reaccionar a eventos específicos. * **Relación:** Actuarán como consumidores de eventos generados por los microservicios. Por ejemplo, una Azure Function podría activarse ante un evento de "stock bajo" para enviar una notificación, o procesar un evento de "fecha de garantía vencida".   **Almacenamiento de Archivos (AWS S3 Buckets):**   * **Componente:** Un servicio de almacenamiento de objetos escalable. * **Relación:** Los microservicios (específicamente el Servicio de Gestión Documental) interactuarán con S3 para cargar, recuperar y gestionar archivos PDF asociados a los recursos. El frontend obtendrá los enlaces de descarga a través del backend.   **Autenticación y Autorización (AWS Cognito):**   * **Componente:** Un servicio de gestión de identidad como un proveedor externo. * **Relación:** El frontend redirigirá a los usuarios a **AWS Cognito** para realizar la autenticación. Una vez autenticados, se emitirá un token de acceso que el frontend incluirá con cada solicitud a los microservicios. Los microservicios validará este token con AWS Cognito para garantizar que solo los usuarios autenticados y autorizados puedan acceder a los recursos según sus roles y permisos.   **Contenedorización (Docker):**   * **Componente:** Tecnología para empaquetar aplicaciones y sus dependencias en contenedores portables. * **Relación:** Cada microservicio de Spring Boot y la aplicación Angular se dockerizarán. Esto asegurará la consistencia del entorno desde el desarrollo hasta la producción y simplificará el despliegue en **Fly.io**.   **Infraestructura (AWS EC2):**   * **Componente:** Máquinas virtuales escalables. * **Relación:** Los contenedores Docker de los microservicios de Spring Boot se desplegarán y ejecutarán en instancias de EC2.  Principios que Guían el Diseño  1. **Principio de Responsabilidad Única (SRP):** Cada microservicio se centrará en una única capacidad de negocio bien definida. 2. **Acoplamiento Débil y Alta Cohesión:** Los servicios serán independientes y se comunicarán a través de interfaces bien definidas, minimizando dependencias entre ellos. 3. **Escalabilidad Horizontal:** La arquitectura permitirá escalar servicios individualmente según la demanda, evitando cuellos de botella en el sistema. 4. **Resiliencia:** El fallo de un microservicio no debería afectar a todo el sistema. 5. **Autonomía de Equipos:** Los equipos podrán desarrollar, desplegar y operar sus microservicios de forma independiente. 6. **Descentralización de Datos:** Cada microservicio poseerá sus propios datos, promoviendo la autonomía y flexibilidad en la elección de la tecnología de persistencia. 7. **Comunicación Asíncrona Preferida:** La comunicación entre servicios será principalmente asíncrona mediante eventos(Azure Functions), lo que mejora la resiliencia y desacoplamiento.  Patrón de Arquitectura: Microservicios La elección del patrón de microservicios es fundamental para:   * **Desarrollo Ágil:** Permite equipos pequeños e independientes trabajando en paralelo. * **Flexibilidad Tecnológica:** Diferentes servicios pueden usar tecnologías óptimas para sus necesidades. * **Mayor Resiliencia:** El aislamiento de fallos mejora la disponibilidad del sistema. * **Escalabilidad Fina:** Se pueden escalar solo los componentes que lo necesitan, optimizando recursos. * **Mantenibilidad:** Servicios más pequeños son más fáciles de entender, modificar y mantener.   En resumen, esta arquitectura de microservicios, apoyada por un patrón basado en eventos, proporcionará una base sólida para un sistema de gestión de recursos tecnológicos que no solo cumple con los requisitos actuales, sino que también está preparado para futuras expansiones y cambios con un mínimo impacto en la operación. | |

**Planes**

**Plan de Calidad**

**Introducción**

Este documento presenta el Plan de Gestión de Calidad para el proyecto de la aplicación web de gestión de recursos tecnológicos. Su objetivo principal es asegurar que el producto final cumpla y supere los requisitos de los usuarios y stakeholders.

El plan establecerá los estándares, actividades y responsabilidades para garantizar la excelencia en cada fase del proyecto, desde la planificación hasta el despliegue. Buscamos una calidad total, entregando una solución robusta, segura y confiable que aporte valor significativo a la gestión de recursos tecnológicos.

**Objetivo del Plan de Gestión de Calidad**

El objetivo principal de este Plan de Gestión de Calidad es **garantizar que la aplicación web de gestión de recursos tecnológicos se entregue con la más alta calidad**, en línea con los objetivos del proyecto. Esto incluye:

* **Asegurar que cumpla con todos los requisitos** (funcionales y no funcionales).
* **Minimizar errores** mediante controles proactivos.
* **Optimizar la experiencia del usuario** para que sea intuitiva y eficiente.
* **Proteger la seguridad e integridad de los datos**.
* **Promover la mantenibilidad y escalabilidad** del código.
* **Fomentar la comunicación** para la mejora continua.
* Contribuir a que el proyecto se entregue **dentro del alcance, tiempo y presupuesto**.

En resumen, buscamos construir una aplicación confiable y de alto valor que optimice la gestión de recursos tecnológicos.

**Tiempos de entrega para paquetes de trabajo (Procesos / Entregables del proyecto sujetos a revisión de calidad)**

|  |  |  |
| --- | --- | --- |
| **Entregable / Proceso** | **Tiempo de entrega** | **Importancia para el proyecto** |

|  |  |  |
| --- | --- | --- |
| Definición del Proyecto . | Semana 1 | Crítico |
| Documentación. | Semana 2 | Crítico |
| Presentación Propuesta de proyecto. | Semana 3 | Crítico |
| Inicio de Desarrollos Front y Back. | Semana 4 | Crítico |
| Integración de Azure y Azure Functions. | Semana 5 | Crítico |
| Despliegue microservicios en AWS EC2 e integración con AWS Bucket S3. | Semana 6 | Crítico |
| Integración Capa Frontend con capa Backend y despliegue en Fly.io | Semana 7 | Crítico |
| Test unitarios capa FrontEnd y Backend y actualización de documentación. | Semana 8 | Alto |

**Asignación de responsables del seguimiento de la calidad**

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Rol** | **Responsabilidad** |
| Carlos Valverde | Gerente de proyecto | Controlar el seguimiento en el desarrollo del proyecto, en base a los tiempos asignados para cada una de las actividades a desarrollar. |
| Giovanny Otárola | Desarrollador Frontend | Validar interfaz, subida de archivos y pruebas de integración con API |
| Matias Fuentes | Desarrollador Backend | Validar integración de funcionalidades   implementadas en AWS y Azure |

**Estándares de Calidad**

Para asegurar la calidad del proyecto de gestión de recursos tecnológicos, documentaremos los requisitos y estándares utilizando un enfoque claro y medible.

|  |  |  |  |
| --- | --- | --- | --- |
| **Estándar** | **Descripción** | **Indicador** | **Fuente de datos** |
| Cumplimiento del plazo en el desarrollo y entrega de los hitos del proyecto | Cumplir con los plazos establecidos de acuerdo con el cronograma del proyecto | Índice de desempeño del cronograma | Cronograma del proyecto |
| Documentación de Requisitos Completos y Claros | Todos los requisitos funcionales y no funcionales deben estar especificados de manera unívoca, sin ambigüedades y ser verificables. | Porcentaje de requisitos ambiguos o incompletos identificados en revisiones. | Documento de Requisitos, Actas de Reunión, Herramienta de Gestión de Requisitos (si aplica). |
| Pruebas de Integración y API Exitosas | Las APIs del backend deben funcionar correctamente y sin errores al interactuar con el frontend y otros servicios. | Porcentaje de Casos de Prueba de API y de Integración Pasados. | Informes de Postman, Registros de Pruebas Automáticas. |
| Cobertura de Pruebas Unitarias del Backend | El código del backend debe tener una cobertura mínima de pruebas unitarias para asegurar su robustez. | Porcentaje de Cobertura de Código de Pruebas Unitarias (Backend). | Herramienta de Cobertura de Código (ej. JaCoCo para Java/Spring Boot). |
| Cumplimiento de Estándares de Codificación | El código fuente debe adherirse a guías de estilo y buenas prácticas de programación definidas. | Porcentaje de Detecciones de Violaciones de Estándares de Codificación. | Herramientas de Análisis Estático de Código (ej. SonarQube si se implementa, o revisiones de código en GitHub). |
| Rendimiento de la Aplicación (Backend) | La aplicación debe responder a las solicitudes del usuario dentro de los tiempos límite establecidos. | **Tiempo Promedio de Respuesta de APIs (ms)** en escenarios de carga definidos. | Registros de monitoreo de rendimiento, Informes de Pruebas de Carga (ej. JMeter). |
| Usabilidad del Frontend (UX/UI) | La interfaz de usuario debe ser intuitiva, fácil de navegar y satisfacer las necesidades del usuario. | Resultados de Pruebas de Usabilidad y encuestas de satisfacción del usuario (SUS - System Usability Scale). | Informes de Pruebas de Usabilidad, Cuestionarios SUS. |

**Atributos de Calidad**

Los siguientes atributos de calidad se considerarán esenciales para el proyecto de gestión de recursos tecnológicos, asegurando que la solución final sea de alto valor.

|  |  |
| --- | --- |
| **Atributo** | **Descripción** |
| Funcionalidad | La capacidad del software para proveer funciones que satisfacen las necesidades especificadas o implícitas cuando es usado bajo condiciones específicas. |
| Confiabilidad | La capacidad del software para mantener su nivel de rendimiento bajo condiciones definidas durante un periodo de tiempo determinado. |
| Usabilidad | La capacidad del software para ser entendido, aprendido, operado y atractivo para el usuario en condiciones especificadas. |
| Eficiencia | La capacidad del software para proveer un rendimiento adecuado, en relación con la cantidad de recursos usados, bajo condiciones definidas. |
| Mantenibilidad | La capacidad del software para ser modificado. |
| Seguridad | La capacidad del software para proteger la información y los datos de manera que personas o sistemas no autorizados no puedan acceder, modificar o destruir los datos, y que las personas o sistemas autorizados no se les niegue el acceso. |
| Escalabilidad | La capacidad del sistema para manejar una carga de trabajo creciente (más usuarios, más datos, más transacciones) o ser ampliado para satisfacer necesidades futuras sin una degradación significativa del rendimiento. |
| Portabilidad | La capacidad del software para ser transferido de un entorno a otro. |
| Interoperabilidad | La capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada. |

**Indicadores de Calidad**

Estos indicadores permitirán evaluar de manera precisa los atributos de calidad definidos, en coherencia con los objetivos del proyecto.

|  |  |  |
| --- | --- | --- |
| **Indicador** | **Descripción** | **Atributo** |
| Porcentaje de Requisitos Cubiertos por Pruebas | Proporción de requisitos del sistema que tienen al menos un caso de prueba asociado. | Funcionalidad |
| Tasa de Defectos por Módulo | Número de defectos encontrados y validados por cada módulo de desarrollo y por módulo funcional. | Confiabilidad, Mantenibilidad |
| Tiempo Medio entre Fallos (MTBF) | Promedio de tiempo que el sistema o un componente opera sin fallar. | Confiabilidad |
| Tiempo de Inactividad (Downtime) Anual | Suma total de las horas o minutos en que el sistema no estuvo disponible para los usuarios en un año. | Confiabilidad, Disponibilidad |
| Tiempo de Aprendizaje de Tareas Clave | Tiempo promedio que le toma a un nuevo usuario completar una tarea esencial en la aplicación por primera vez. | Usabilidad |
| Tiempo Promedio de Respuesta de APIs (ms) | Tiempo promedio que tarda el backend en procesar una solicitud API y devolver una respuesta. | Eficiencia |
| Uso de Recursos (CPU, Memoria) en Carga Máxima | Consumo de CPU y memoria de los servidores del backend y la base de datos bajo la carga máxima de usuarios concurrentes. | Eficiencia, Escalabilidad |
| Tiempo de Escalado de Instancias (ms) | Tiempo que toma el sistema para agregar nuevas instancias de servidores o contenedores en respuesta a un aumento de la carga. | Escalabilidad |

**Uso y Seguimiento de Métricas de Calidad**

Las métricas o KPI definidos en este plan serán utilizados como herramientas clave para el seguimiento del desempeño del proyecto en cada una de sus fases. Cada indicador se medirá en intervalos definidos (semanales, por iteración o por módulo) y será comparado con los valores objetivo establecidos para determinar el nivel de cumplimiento de los atributos de calidad. Los resultados serán documentados en las tablas de medición de control de calidad y revisados por el equipo responsable en reuniones de control de calidad.

En caso de detectar desviaciones o incumplimientos en las métricas establecidas, se generarán alertas y se tomarán medidas correctivas, ya sea en la planificación, diseño, desarrollo o pruebas. Este seguimiento sistemático permitirá tomar decisiones informadas, mejorar continuamente el proceso y asegurar que la aplicación cumpla con los estándares de calidad definidos.

|  |  |  |
| --- | --- | --- |
| **Aspecto de calidad** | **Métrica / KPI** | **Objetivo o umbral esperado** |
| Disponibilidad del sistema | Porcentaje de uptime | ≥ 99.5% mensual |
| Tiempos de respuesta | Tiempo medio de carga de interfaz | < 2 segundos |
| Integridad de los datos | % de registros consistentes | 100% de registros validados |
| Gestión de incidencias | Tiempo promedio de resolución de errores. | < 48 horas hábiles |
| Cobertura de pruebas | % de código cubierto por pruebas | ≥ 85% en pruebas unitarias |

**Instrumento de medición de Calidad**

Los siguientes instrumentos se utilizarán para medir y evaluar los estándares, atributos e indicadores de calidad definidos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instrumento** | **Descripción** | **Estándar** | **Atributos** | **Indicador** |
| Curva S | Instrumento para medir el progreso, evaluar el rendimiento y realizar previsiones del flujo del proyecto | Cumplimiento del plazo en el desarrollo y entrega de los hitos del proyecto | Tiempo definido | Índice de desempeño del cronograma |
| Cronograma del Proyecto | Documento detallado que establece las fechas de inicio y fin de las tareas, hitos y entregables. | Cumplimiento del plazo en el desarrollo y entrega de los hitos del proyecto | Tiempo definido | Índice de desempeño del cronograma |
| Herramientas de Gestión de Requisitos (ej. Trello) | Plataformas para documentar, organizar y rastrear los requisitos, permitiendo su validación. | Documentación de Requisitos Completos y Claros | Funcionalidad | Porcentaje de Requisitos Cubiertos por Pruebas |
| Informes de Cobertura de Código (ej. JaCoCo) | Herramientas que analizan el código para determinar qué porcentaje de este es ejecutado por las pruebas unitarias. | Cobertura de Pruebas Unitarias del Backend | Mantenibilidad, Confiabilidad | Porcentaje de Cobertura de Código de Pruebas Unitarias (Backend) |
| Postman / Herramientas de Pruebas de API Automatizadas | Plataformas para crear, ejecutar y automatizar pruebas de los endpoints de las APIs RESTful. | Pruebas de Integración y API Exitosas | Funcionalidad, Confiabilidad | Porcentaje de Casos de Prueba de API y de Integración Pasados |
| Herramientas de Análisis Estático de Código (ej. SonarQube) | Software que analiza el código fuente sin ejecutarlo para identificar errores, vulnerabilidades y violaciones de estilo. | Cumplimiento de Estándares de Codificación | Mantenibilidad, Seguridad | Porcentaje de Detecciones de Violaciones de Estándares de Codificación |
| Pruebas de Usabilidad (observación, entrevistas) | Sesiones con usuarios reales para observar cómo interactúan con la interfaz y recopilar su retroalimentación. | Usabilidad del Frontend (UX/UI) | Usabilidad | Puntuación del Cuestionario SUS, Tiempo de Aprendizaje de Tareas Clave |

**Mediciones del Control de Calidad**

El registro de los controles y el seguimiento del proceso de gestión de calidad se realizará de forma sistemática utilizando la siguiente tabla, la cual será actualizada periódicamente.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Fecha** | **Proceso / Producto medido** | **Valor requerido** | **Valor obtenido** | **¿Es aceptable?** | **Recomendación** | **Fecha de Resolución** |
| 2025-06-25 | Avance Hito 1: Backend CRUD Básico | SPI >= 0.95 | 0.98 | Sí | Continuar con la metodología actual. | 2025-06-30 |
| 2025-07-10 | Cobertura Pruebas Unitarias Módulo Inventario | >= 80% | 85% | Sí | Mantener el enfoque en la calidad del código. | 2025-06-12 |
| 2025-06-12 | Tiempo Promedio de Respuesta API (Listar Recursos) | <= 100 ms | 90 ms | Sí | Monitorear continuamente el rendimiento. | 2025-06-12 |
| 2025-07-12 | Pruebas de Integración - Flujo de Autenticación | 100% Pasados | 95% Pasados | No | Investigar y corregir las pruebas fallidas de inmediato. | 2025-07-12 |
| 2025-07-12 | Detecciones de Violaciones de Estándares de Codificación (Backend) | <= 5% | 7% | No | Implementar herramientas de linting y revisiones de código más estrictas. | 2025-07-13 |
| 2025-07-12 | Tasa de Defectos por Iteración | <= 3 defectos | 4 defectos | No | Reforzar las pruebas de control de calidad y las revisiones de código. | 2025-07-12 |
| 2025-07-5 | Cumplimiento de Requisitos Funcionales | >= 90% | 88% | No | Realizar una revisión exhaustiva de requisitos y funcionalidades implementadas. | 2025-07-07 |
| 2025-07-10 | Tiempo de Resolución de Incidencias Críticas | <= 24 horas | 18 horas | Sí | Mantener los tiempos de respuesta y escalamiento. | 2025-07-11 |
| 2025-07-05 | **CrearPrestamoComponent**: Tasa de Éxito de Creación de Préstamos | >= 98% | 99% | Sí | Mantener la validación estricta en el frontend y backend. | 2025-07-05 |
| 2025-07-05 | **CrearRecursoComponent**: Tiempo de Carga de Formulario de Recurso | <= 500 ms | 450 ms | Sí | Optimizar la carga de dependencias si es posible. | 2025-07-05 |
| 2025-07-05 | **DashboardComponent**: Tiempo de Carga del Dashboard | <= 1.5 segundos | 1.2 segundos | Sí | Monitorear el rendimiento con el crecimiento de datos. | 2025-07-05 |
| 2025-07-05 | **DetallesPrestamoComponent**: Tiempo de Carga de Detalles de Recurso (con historial y documentos) | <= 800 ms | 750 ms | Sí | Evaluar la paginación para historial muy extenso. | 2025-07-05 |
| 2025-07-06 | **EditarRecursoComponent**: Tasa de Éxito de Actualización de Recurso | >= 98% | 97% | No | Revisar logs de errores en el backend para fallos de actualización. | 2025-07-06 |
| 2025-07-06 | **HomeComponent**: Tiempo de Carga Inicial (Home) | <= 1 segundo | 0.8 segundos | Sí | Asegurar que los recursos críticos se carguen primero. | 2025-07-06 |
| 2025-07-06 | **InventarioComponent**: Tiempo de Carga de Tabla de Inventario (con filtros) | <= 2 segundos | 2.5 segundos | No | Optimizar consultas de base de datos o implementar paginación/virtualización. | 2025-07-06 |
| 2025-07-06 | **PrestamosComponent**: Tiempo de Carga de Tabla de Préstamos (con filtros) | <= 2 segundos | 1.8 segundos | Sí | Monitorear el rendimiento a medida que crece el volumen de préstamos. | 2025-07-06 |
| 2025-07-09 | **ReportesComponent**: Tiempo de Generación de Reporte (promedio) | <= 5 segundos | 6 segundos | No | Optimizar las consultas de datos para la generación de reportes. | 2025-07-09 |
| 2025-07-09 | **DashboardService**: Latencia de Servicio (Dashboard Data) | <= 150 ms | 130 ms | Sí | Mantener la eficiencia de las consultas de base de datos. | 2025-07-09 |
| 2025-07-10 | **HistorialService**: Latencia de Servicio (Historial por Recurso) | <= 100 ms | 110 ms | No | Optimizar la indexación de la base de datos para el historial. | 2025-07-10 |
| 2025-07-10 | **PrestamosService**: Latencia de Servicio (Guardar Préstamo) | <= 200 ms | 180 ms | Sí | Asegurar la eficiencia de las transacciones de base de datos. | 2025-07-10 |
| 2025-07-11 | **ResourceService**: Latencia de Servicio (Actualizar Recurso) | <= 200 ms | 220 ms | No | Revisar la lógica de actualización del recurso en el backend. | 2025-07-11 |

**Plan de Riesgo**

**Propósito y Alcance del Plan**

El presente plan tiene por principal objetivo establecer una estrategia de identificación, análisis, respuesta y monitoreo de los riesgos que puedan impactar negativamente el desarrollo y la calidad del proyecto GIRET. Su alcance incluye riesgos técnicos, de gestión, humanos y externos, desde la etapa de diseño hasta la entrega del sistema.

**Metodología**

Se utilizará un enfoque basado en análisis cualitativo para la identificación y evaluación de riesgos, utilizando escalas de probabilidad e impacto. Las actividades de riesgo se integrarán en reuniones semanales de control de avance. La identificación se realizará durante la planificación de cada fase y se documentará en un registro actualizado.

* Identificación de riesgos: lluvia de ideas, revisión de experiencias similares.
* Análisis: matriz de impacto y probabilidad.
* Respuesta: evitar, mitigar, transferir o aceptar.
* Monitoreo: reevaluación de riesgos al finalizar cada etapa del cronograma.

**Roles y Responsabilidades**

|  |  |  |
| --- | --- | --- |
| Rol | Responsable | Responsabilidad |
| Gerente de Proyecto | Carlos Valverde | Supervisar, documentar y controlar la ejecución del plan de riesgos. |
| Desarrollador Backend | Matias Fuentes | Identificar y gestionar riesgos técnicos en el backend. |
| Desarrollador Frontend | Giovanny Otarola | Identificar y gestionar riesgos técnicos en el desarrollo UI. |

**Calendario**

Evaluación inicial de riesgos: Semana 2

Revisión de riesgos técnicos: Semanas 4 y 6

Revisión de riesgos organizacionales y de calidad: Semana 7 y 8

**Categorías de Riesgo**

**Técnicos:** fallas de integración, incompatibilidad de tecnologías, pérdida de datos

**Humanos:** disponibilidad del equipo, errores de desarrollo

**Organizacionales:** falta de claridad en requisitos, tiempos limitados

**Externos:** problemas de conectividad, cambios en servicios cloud

**Estructura de Desglose de Riesgos (RBS)**

|  |  |
| --- | --- |
| **Tipo** | **Descripción** |
| Técnicos | -Fallos de configuración  -No disponibilidad de APIs o servicios externos (Azure, Oracle)  - Problemas de integración entre módulos |
| Humanos | -Falta de conocimientos específicos  -Problemas de coordinación del equipo |
| Organizacionales | -Cambios en requerimientos  -Demoras por parte del profesor o patrocinador |
| Externos | -Cambios en políticas de Azure u Oracle  -Cortes de red o energía  - Riesgos por permisos insuficientes o accesos restringidos a servicios cloud |
| Dependencias | - Uso de librerías o frameworks no soportados o inestables |

**Definiciones de Probabilidad e Impacto de Riesgos**

**Definiciones de Probabilidad**

|  |  |
| --- | --- |
| Muy Alta | ≥ 90% (ocurrencia prácticamente segura) |
| Alta | 70–89% |
| Media | 50–69% |
| Baja | 30–49% |
| Muy Baja | < 30% |

**Definiciones de Impacto**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Objetivo de Proyecto | Muy bajo | Bajo | Medio | Alto | Muy Alto |
| Alcance | Cambios mínimos | Ajustes menores | Cambio en un módulo | Cambio de enfoque | Afecta entrega completa |
| Cronograma | <1 día | 1–2 días | 3–5 días | 1 semana | >1 semana |
| Calidad | No afecta funcionalidad | Afecta interfaz | Afecta funcionalidades clave | Deja módulo incompleto | Proyecto inviable |

**Matriz de Probabilidad e Impacto**

La Matriz de Probabilidad e Impacto es una herramienta visual esencial para priorizar los riesgos, tanto amenazas como oportunidades, que podrían afectar nuestro proyecto de gestión de recursos tecnológicos. Al combinar la probabilidad de ocurrencia con el impacto potencial, podemos enfocar nuestros recursos y esfuerzos en los elementos más críticos, optimizando la gestión y el uso de los recursos.

**Amenazas (Riesgos)**

Identificar las amenazas es crucial para desarrollar planes específicos que las mitiguen o eviten, reduciendo su impacto potencial en el éxito del proyecto.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Impacto  Probabilidad | | Muy Bajo | Bajo | Medio | Alto | Muy Alto |
| 0,05 | 0,10 | 0,20 | 0,40 | 0,80 |
| Muy Alta | 0,90 | 0,045 | 0,09 | 0,18 | 0,36 | 0,72 |
| Alta | 0,70 | 0,035 | 0,07 | 0,14 | 0,28 | 0,56 |
| Media | 0,50 | 0,025 | 0,05 | 0,10 | 0,20 | 0,40 |
| Baja | 0,30 | 0,015 | 0,03 | 0,06 | 0,12 | 0,24 |
| Muy Baja | 0,10 | 0,005 | 0,01 | 0,02 | 0,04 | 0,08 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Amenaza (Riesgo Negativo)** | **Probabilidad** | **Impacto** | **Puntuación (P x I)** | **Plan de Respuesta (Mitigación/Contingencia)** |
| Retraso en la entrega de módulos clave (Backend/Frontend) | Alta (0.70) | Alto (0.40) | 0.28 | Plan de Respuesta: Comunicación diaria en Trello. Reuniones de sincronización diarias. Establecer hitos semanales pequeños y alcanzables. Redistribución de tareas si es necesario, con apoyo del profesor. Posibilidad de reducir el alcance de funcionalidades no críticas (ver "Funcionalidades a No Probar"). |
| Problemas de integración entre microservicios o con servicios externos (AWS Cognito, AWS Api Gateway, AWS S3) | Media (0.50) | Alto (0.40) | 0.20 | Plan de Respuesta: Pruebas de integración continuas y automatizadas desde fases tempranas. Definición clara de contratos de API. Uso de Postman para validación constante. Configuración de entornos de prueba estables y reproducibles. |
| Bajo rendimiento de la aplicación bajo carga esperada | Baja (0.30) | Alto (0.40) | 0.12 | Plan de Respuesta: Realización de pruebas de rendimiento con JMeter en fases tempranas. Monitoreo constante del consumo de recursos (CPU, memoria). Optimización de consultas a base de datos y algoritmos críticos. Considerar escalabilidad vertical u horizontal temprana si los resultados lo indican. |
| Falta de claridad o cambios frecuentes en los requisitos | Baja (0.30) | Medio (0.20) | 0.06 | Plan de Respuesta: Sesiones regulares de clarificación de requisitos con el profesor. Documentación detallada en Trello. Gestión formal de cambios de requisitos, evaluando impacto en cronograma y alcance. Priorización de requisitos en cada sprint. |
| Problemas de compatibilidad con navegadores/dispositivos principales | Baja (0.30) | Bajo (0.10) | 0.03 | Plan de Respuesta: Definir explícitamente los navegadores y dispositivos soportados (ya hecho). Realizar pruebas de compatibilidad en los navegadores principales. Usar CSS adaptable y frameworks responsivos como Angular Material. |
| Errores de integración entre módulos internos (backend y frontend) | Media (0.50) | Medio (0.20) | |  | | --- | | 0.10 |  |  | | --- | |  | | Pruebas de integración. Uso de Postman para contratos API. Validación continua entre desarrolladores. |
| Problemas con permisos o autenticación (Azure AD, JWT mal configurado, etc.) | |  | | --- | | Media (0.50) |  |  | | --- | |  | | |  | | --- | | Alto (0.40) |  |  | | --- | |  | | 0.20 | Revisión temprana de los flujos de autenticación y autorización. Tests automatizados de roles. Apoyo del profesor ante bloqueo. |
| Dependencia de librerías externas inestables o sin soporte | |  | | --- | | Baja (0.30) |  |  | | --- | |  | | |  | | --- | | Alto (0.40) |  |  | | --- | |  | | |  | | --- | | 0.12 |  |  | | --- | |  | | Verificación activa de versiones, mantenedores y documentación. Alternativas de respaldo si fallan dependencias clave. |
| Cambios o falta de permisos en servicios cloud utilizados (Azure, Oracle) | |  | | --- | | Baja (0.30) |  |  | | --- | |  | | |  | | --- | | Alto (0.40) |  |  | | --- | |  | | 0.12 | Solicitar accesos con anticipación. Documentar configuraciones. Contacto temprano con soporte institucional. Mantener entornos locales alternativos. |

**Oportunidades**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Impacto  Probabilidad | | Muy Alto | Alto | Medio | Bajo | Muy Bajo |
| 0,05 | 0,10 | 0,20 | 0,40 | 0,80 |
| Muy Alta | 0,90 | 0,045 | 0,09 | 0,18 | 0,36 | 0,72 |
| Alta | 0,70 | 0,035 | 0,07 | 0,14, | 0,28 | 0,56 |
| Media | 0,50 | 0,025 | 0,05 | 0,10 | 0,20 | 0,40 |
| Baja | 0,30 | 0,015 | 0,03 | 0,06 | 0,12 | 0,24 |
| Muy Baja | 0,10 | 0,005 | 0,01 | 0,02 | 0,04 | 0,08 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Oportunidad (Riesgo Positivo) | Probabilidad | Impacto | Puntuación (P x I) | Plan de Respuesta (Explotación/Mejora) |
| Descubrimiento de una herramienta/librería que optimice el desarrollo o rendimiento | Media (0.50) | Alto (0.40) | 0.20 | Plan de Respuesta: Investigación activa y constante de nuevas tecnologías. Realizar pruebas de concepto rápidas. Si se valida la eficiencia, integrar con aprobación del profesor, justificando el impacto positivo en el cronograma o la calidad. |
| Disponibilidad de tiempo adicional inesperado para el equipo | Baja (0.30) | Medio (0.20) | 0.06 | Plan de Respuesta: Utilizar el tiempo extra para refactorizar código, mejorar la documentación, realizar pruebas más exhaustivas (ej. E2E adicionales) o explorar funcionalidades no prioritarias previamente excluidas del alcance inicial. |
| Feedback temprano y positivo de los usuarios o del profesor sobre la usabilidad/funcionalidad | Alta (0.70) | Medio (0.20) | 0.14 | Plan de Respuesta: Capturar este feedback para reforzar las buenas prácticas y replicarlas en otros módulos. Utilizarlo como motivación para el equipo y como validación temprana del diseño. Podría permitir avanzar más rápido en otras funcionalidades. |
| Reutilización de código o componentes existentes de proyectos anteriores | Media (0.50) | Bajo (0.10) | 0.05 | Plan de Respuesta: Realizar una auditoría de código o componentes de proyectos previos que puedan ser adaptados. Evaluar la calidad y compatibilidad del código existente para asegurar que no introduzca deuda técnica. Si es viable, incorporarlo para acelerar el desarrollo. |

**Revisión de la tolerancia de los interesados (Stakeholders)**

Hemos analizado la tolerancia al riesgo de nuestros interesados clave para alinear la estrategia del proyecto:

* **Profesor (Supervisor/Mentor de Calidad):**
  + **Acepta:** Riesgos técnicos medios que ofrezcan aprendizaje o riesgos menores de alcance.
  + **No Compromete:** Funcionalidad central, seguridad del sistema, calidad del código y documentación. Un fracaso en la entrega o fallas críticas de seguridad son inaceptables.
* **Estudiantes (Desarrolladores Backend y Frontend):**
  + **Aceptan:** Riesgos técnicos medios a altos si implican un aprendizaje significativo, y riesgos menores de cronograma manejables.
  + **No Comprometen:** La calidad del código y la arquitectura que diseñen, ni la incapacidad de resolver problemas críticos que impidan el avance o resulten en una calificación deficiente.

**Alineación de la Estrategia**

Priorizaremos la mitigación de riesgos funcionales y de seguridad a través de pruebas exhaustivas. Mantendremos una comunicación constante con el profesor y equilibraremos la exploración tecnológica con el cumplimiento de los objetivos de entrega, asegurando siempre la calidad del código y la documentación.

**Formatos de los Informes**

* Bitácora de riesgos en Google Docs compartido con el equipo y el gerente del  proyecto.
* Revisión de riesgos en cada entrega intermedia (Semanas 4, 6 y 8).
* Informe resumen incluido en la presentación final del proyecto.

**Seguimiento**

* Reevaluación semanal de riesgos durante la planificación.
* Uso de checklist de validación de entregables.
* Control visual con etiquetas en Trello para marcar riesgos activos y mitigados.

**Plan de Pruebas**

**Resumen Ejecutivo**

Este Plan de Pruebas detalla la estrategia para verificar y validar la aplicación web de gestión de recursos tecnológicos. Su objetivo principal es asegurar que el software cumpla con todos los requisitos y sea robusto, seguro y fácil de usar antes de su implementación.

Es un plan detallado que cubre todas las partes críticas del sistema: el frontend (Angular), los microservicios del backend (Spring Boot), la autenticación (AWS Cognito), el almacenamiento de archivos (AWS S3) y la lógica serverless (Azure Functions). Se evaluará desde la funcionalidad básica hasta el rendimiento y la seguridad. Algunas pruebas de infraestructura no relacionadas directamente con la aplicación quedan fuera de alcance.

Las restricciones significativas incluyen el tiempo limitado y la disponibilidad de entornos de prueba. El esfuerzo requerido será considerable, implicando automatización y seguimiento riguroso para garantizar una entrega de alta calidad.

**Alcance de las Pruebas**

**Elementos de Pruebas**

Este plan detallado cubrirá pruebas exhaustivas de los siguientes elementos clave de la aplicación:

**Módulo de Autenticación y Gestión de Usuarios:** Se verificará el registro, inicio de sesión, recuperación de contraseña y AWS Cognito.

**Módulo de Gestión de Inventario:** Se probará toda la funcionalidad CRUD (Crear, Leer, Actualizar, Eliminar) de los recursos tecnológicos, su asignación y el seguimiento de estados e historiales.

**Módulo de Gestión Documental:** Se evaluará la carga, descarga y gestión de documentos adjuntos en AWS S3 Buckets.

**Módulo de Alertas y Notificaciones:** Se comprobará la correcta generación y envío de alertas automáticas (plazos, stock crítico) vía Azure Functions.

**Interfaz de Usuario (Frontend - Angular):** Se testeará la navegación, los componentes visuales, las validaciones de formularios, la responsividad y el manejo de errores.

**APIs RESTful (Backend - Spring Boot):** Se validarán los endpoints en cuanto a entradas, respuestas HTTP, formato de datos, autenticación y la integración entre los microservicios.

**Bases de Datos:** Se asegurará la coherencia e integridad de los datos almacenados.

**Contenedores Docker y Despliegue (fly.io):** Se verificará que las aplicaciones se construyan, ejecuten y desplieguen correctamente en los entornos de destino.

Cada uno de estos elementos será sometido a pruebas específicas, incluyendo unitarias, de integración, funcionales, de rendimiento, de seguridad y de usabilidad.

**Funcionalidades a No Probar**

Para optimizar el tiempo y los recursos, algunas funcionalidades no se probarán en esta fase, priorizando lo más crítico del proyecto:

**Integración con Sistemas Externos Futuros (ej. ERP):**

* **Razón:** Se pospone para fases futuras del proyecto.
* **Riesgos:** Posible incompatibilidad o errores al integrar más adelante.
* **Implicaciones:** La aplicación operará de forma independiente en esta etapa, sin comunicación directa con otros sistemas empresariales.

**Módulo de Reportes Analíticos Avanzados y Dashboards Personalizables:**

* **Razón:** Requiere un gran esfuerzo de desarrollo y no es prioritario en esta fase inicial.
* **Riesgos:** Análisis de datos limitado directamente en la aplicación.
* **Implicaciones:** Los usuarios solo tendrán acceso a listados básicos, no a gráficos o reportes complejos.

**Soporte para Navegadores Web Antiguos (ej. Internet Explorer 11):**

* **Razón:** Alta inversión de tiempo para una baja cuota de mercado.
* **Riesgos:** Problemas de visualización o funcionalidad para usuarios con navegadores desactualizados.
* **Implicaciones:** Se recomienda encarecidamente el uso de navegadores modernos para una experiencia óptima.

Estas exclusiones han sido acordadas para enfocar los recursos en las capacidades centrales de la aplicación, y los riesgos asociados serán gestionados mediante comunicación clara y planes futuros.

**Enfoque de Pruebas (Estrategia)**

Nuestra estrategia de pruebas asegura la calidad de la aplicación de gestión de recursos tecnológicos mediante un enfoque integral:

**Tipos de Pruebas**

* **Pruebas Unitarias:** Los desarrolladores validarán componentes pequeños del backend (Spring Boot) y frontend (Angular).
* **Pruebas de Integración:** Se verificará la comunicación entre microservicios, la base de datos, y servicios externos como AWS Cognito y AWS S3.
* **Pruebas Funcionales:** Se confirmará que todas las funcionalidades (inventario, documentos, alertas, autenticación) cumplen los requisitos.
* **Pruebas de Interfaz de Usuario (UI) y Usabilidad (UX):** Se evaluará la facilidad de uso y la responsividad del frontend.
* **Pruebas de Rendimiento:** Se medirá la velocidad y estabilidad del backend (AWS EC2) bajo carga.
* **Pruebas de Regresión:** Se ejecutarán periódicamente para asegurar que los cambios no introduzcan nuevos defectos.

**Requisitos Especiales**

Necesitaremos un entorno de pruebas aislado y herramientas de automatización como Postman y frameworks de pruebas unitarias.

**Configuraciones y Datos**

Evaluaremos la aplicación en navegadores modernos. Usaremos datos de prueba representativos (usuarios, recursos, documentos, alertas).

**Alcance de Regresión**

Las pruebas de regresión serán exhaustivas en hitos importantes y selectivas para cambios menores, priorizando la automatización.

Esta estrategia nos permitirá identificar defectos a tiempo y entregar una aplicación de alta calidad.

**Criterios de Aceptación o Rechazo**

Los siguientes parámetros determinarán cuándo el ciclo de pruebas del proyecto se considera completado satisfactoriamente, y por ende, cuándo la aplicación está lista para una posible fase de despliegue o aceptación por parte de los usuarios finales.

Cobertura de Pruebas Unitarias: El 85% de los componentes críticos del backend (microservicios principales y capas de persistencia) y frontend deben tener pruebas unitarias ejecutadas. Se exige un mínimo del 80% de cobertura de código para las clases y métodos clave del backend, medido con herramientas como JaCoCo.

**Casos de Prueba Ejecutados y Exitosos:**

* 95% de los casos de prueba funcionales y de integración definidos deben haber sido ejecutados.
* Al menos el 85% de los casos de prueba funcionales y de integración deben haber pasado exitosamente.
* **Defectos Críticos y Mayores Resueltos:** El 100% de los defectos catalogados como "Críticos" o "Mayores" (que impiden la funcionalidad principal o causan pérdida de datos) deben haber sido corregidos y validados.
* **Defectos Menores y de Baja Prioridad:** Un máximo del 10% de los defectos catalogados como "Menores" o "Baja Prioridad" pueden quedar abiertos al finalizar las pruebas, siempre que no afecten la usabilidad o la funcionalidad principal.
* **Resultados de Pruebas No Funcionales:**

**- Rendimiento:** El tiempo promedio de respuesta de las APIs críticas debe  estar por debajo del umbral establecido .

* **Documentación de Pruebas Completa:** Todos los entregables de pruebas   (casos de prueba, informes de ejecución, reportes de defectos, evidencias) deben estar completos y actualizados.

**Criterios de Suspensión**

La ejecución de un conjunto de casos de prueba se detendrá de manera controlada bajo las siguientes circunstancias, para evitar el desperdicio de recursos y asegurar que las pruebas se reanuden solo cuando sea productivo.

**Presencia de Defectos Críticos:** Si se detecta un defecto crítico que impide la ejecución continuada de pruebas en un módulo o funcionalidad central (ej. el inicio de sesión no funciona, la creación de recursos falla consistentemente), la ejecución de ese conjunto de pruebas o de la fase actual se suspenderá.

**Alto Porcentaje de Casos Fallidos:** Si más del 20% de los casos de prueba ejecutados en una fase o módulo específico fallan, se suspenderá la ejecución para permitir que el equipo de desarrollo investigue y corrija los problemas subyacentes.

**Inestabilidad del Entorno de Pruebas:** Si el entorno de pruebas deja de ser estable o accesible (ej. la base de datos se cae, los servidores de la aplicación no responden), la ejecución de pruebas se suspenderá hasta que el entorno sea restaurado.

**Cambios Mayores en los Requisitos:** Si se introducen cambios significativos en los requisitos funcionales o no funcionales que impactan directamente en las funcionalidades que se están probando, las pruebas se suspenderán hasta que los requisitos actualizados sean analizados y los casos de prueba pertinentes sean modificados o creados.

**Criterios de Reanudación**

Una vez que las pruebas han sido suspendidas, se podrán reanudar solo cuando se cumplan las siguientes condiciones, garantizando que el entorno y la aplicación estén en un estado adecuado para obtener resultados válidos y consistentes.

**Defectos Críticos Resueltos:** Todos los defectos críticos que causaron la suspensión deben haber sido corregidos y la corrección debe haber sido verificada mediante una prueba de confirmación (retesting).

**Estabilidad del Entorno de Pruebas Restablecida:** El entorno de pruebas debe ser completamente estable y accesible, con todos los servicios y componentes funcionando correctamente.

**Porcentaje de Casos Fallidos Reducido:** Si la suspensión fue por un alto porcentaje de fallos, las correcciones deben haber reducido ese porcentaje a un nivel aceptable (ej. menos del 5% de fallos iniciales en una nueva ejecución de los casos afectados).

**Actualización de Casos de Prueba (si aplica):** Si la suspensión se debió a cambios en los requisitos, los casos de prueba afectados deben haber sido actualizados o nuevos casos creados para reflejar los nuevos requisitos.

**Aprobación del Supervisor/Líder Técnico:** El profesor o líder técnico del proyecto debe aprobar la reanudación de las pruebas, asegurando que se han tomado todas las medidas correctivas necesarias.

**Entregables**

Los siguientes elementos se generarán y entregarán como parte de la ejecución y finalización del Plan de Pruebas. Estos documentos son cruciales para el seguimiento, la transparencia y la documentación del ciclo de vida de la calidad del proyecto.

**Documento del Plan de Pruebas:** Este mismo documento, actualizado con cualquier cambio o decisión relevante tomada durante la ejecución de las pruebas.

**Casos de Prueba y Especificaciones de Diseño de Casos:** Documentación detallada de los escenarios de prueba, pasos, datos de entrada esperados y resultados esperados para cada tipo de prueba (unitarias, integración, funcionales, etc.).

**Registros de Ejecución de Pruebas (Logs):** Evidencia detallada de la ejecución de los casos de prueba, incluyendo la fecha, hora, estado (pasado/fallido) y cualquier nota relevante.

**Reportes de Incidencias/Defectos:** Documentación formal de cada defecto encontrado, incluyendo su descripción, pasos para reproducirlo, severidad, prioridad, estado y asignación. Se utilizará una herramienta de gestión de proyectos como Trello o similar para el seguimiento.

**Evidencias de Pruebas:** Capturas de pantalla, grabaciones de video o registros de base de datos que demuestren el éxito o fracaso de un caso de prueba.

**Reportes de Cobertura de Código:** Informes generados por herramientas (ej. JaCoCo) que muestran el porcentaje de código cubierto por pruebas unitarias y de integración.

**Informes de Pruebas de Rendimiento:** Documentos que resumen los resultados de las pruebas de carga y estrés, incluyendo tiempos de respuesta, latencia y consumo de recursos.

**Informe Final de Pruebas:** Un resumen ejecutivo de todo el proceso de pruebas, incluyendo los resultados clave, la tasa de defectos, la cobertura de pruebas, las desviaciones encontradas y una recomendación sobre la aceptación o rechazo del software para su despliegue.

Estos entregables proporcionarán una visión completa del estado de la calidad de la aplicación y servirán como base para la toma de decisiones futuras del proyecto.

**Recursos**

**Requerimientos de Entornos – Hardware**

Para garantizar la ejecución eficiente y efectiva de las actividades de prueba, se establecen los siguientes requerimientos de hardware y red. Estos recursos aseguran un entorno de pruebas adecuado que replica, en la medida de lo posible, las condiciones del entorno de producción.

**1. Servidores de Aplicación (Backend y Azure Functions)**

Para el backend basado en Spring Boot y las Azure Functions, que se ejecutarán en AWS EC2, se necesitarán instancias con las siguientes especificaciones mínimas para el entorno de pruebas:

* **Capacidad de Procesamiento:** 2 vCPU.
* **Memoria RAM:** 8 GB.
* **Almacenamiento:** 50 GB SSD (para sistema operativo, Docker images y logs).
* **Sistema Operativo:** Ubuntu Server 22.04 LTS (o similar, compatible con Docker).
* **Configuración Adicional:** Docker instalado y configurado.

**2. Bases de Datos**

Dado que cada microservicio podría tener su propia base de datos (o esquema), se requiere un entorno de base de datos robusto para las pruebas.

* **Tipo de Base de Datos:** Oracle Cloud (u otro compatible con los servicios Spring Boot).
* **Software:** Versión compatible del gestor de base de datos.
* **Configuraciones:** Esquemas de base de datos específicos para cada microservicio, con datos de prueba adecuados para escenarios funcionales y de rendimiento. Asegurar copias de seguridad regulares para la integridad de los datos de prueba.

**3. Equipos de PC para los Testers (Desarrolladores)**

Los dos desarrolladores serán también los principales testers. Sus equipos deben cumplir con las siguientes especificaciones:

* **Procesador:** Intel Core i5 (8.ª generación o superior) o AMD Ryzen 5 (2.ª generación o superior).
* **Memoria RAM:** 16 GB DDR4.
* **Almacenamiento:** 512 GB SSD (para sistema operativo, IDEs, herramientas de desarrollo y testing, y datos de prueba locales).
* **Sistema Operativo:** Windows 10/11 o macOS (versión reciente), o una distribución de Linux compatible con Docker Desktop.
* **Software Adicional:**

**-** IDEs (ej., IntelliJ IDEA, VS Code).

- Docker Desktop.

- Postman (para pruebas de API).

- Navegadores web modernos (Chrome, Firefox, Edge, Safari) con sus últimas versiones.

- Herramientas de control de versiones (Git).

**4. Conectividad a la Red**

La conectividad es crucial para la integración y el acceso a los servicios en la nube.

* **Acceso a Internet de Banda Ancha:** Mínimo 50 Mbps de bajada y 20 Mbps de subida para cada tester.
* **Reglas de Firewall/Grupos de Seguridad:** Asegurar que los puertos necesarios estén abiertos en los firewalls de AWS (grupos de seguridad) y en las máquinas locales para permitir la comunicación entre servicios y el acceso de los testers.
* **DNS:** Configuración DNS adecuada para resolver los nombres de dominio de los servicios en la nube.

**5. Otros Recursos**

* **Cuenta en la Nube:** Credenciales de una cuenta de AWS activa con permisos para lanzar instancias EC2, configurar S3 buckets y acceso a AWS Api Gateway y Azure Functions.

Estos requerimientos garantizan que el equipo disponga de la infraestructura necesaria para desarrollar, desplegar y probar la aplicación de manera eficiente y con la fidelidad necesaria para el entorno de producción.

**Requerimientos de Entornos – Software**

Para asegurar un testing efectivo, se necesita el siguiente software:

**1. Sistemas de Pruebas**

Se requiere acceso a las versiones desplegadas de la aplicación (Frontend con Angular, Microservicios Spring Boot en Docker/AWS EC2 y Azure Functions) en un entorno de pruebas dedicado y bien configurado.

**2. Bases de Datos**

Necesitamos acceso a las bases de datos de prueba (ej., Oracle) con datos y herramientas de gestión (ej., SQL developer) para verificar la integridad y coherencia.

**3. Herramientas de Pruebas**

Se instalarán herramientas específicas para:

* **Pruebas Unitarias:** JUnit (Backend), Jasmine/Karma (Frontend).
* **Pruebas de Integración y API:** Postman
* **Análisis de Calidad de Código:** SonarQube..

**4. Software de Seguimiento de Incidencias**

Trello se usará para registrar y gestionar los defectos.

**5. Sistemas de Control de Versiones**

Git y GitHub para que los testers accedan a las versiones correctas del código.

**6. Entorno de Desarrollo Integrado (IDE)**

IntelliJ IDEA y Visual Studio Code para depuración y revisión de código.

**7. Software en Equipos de Testers**

Navegadores web actualizados, Docker Desktop y herramientas de captura de evidencia.

Estos requerimientos garantizan que el equipo tenga las herramientas y plataformas necesarias para ejecutar todas las pruebas de manera eficiente.

**Herramientas de Pruebas Requeridas**

Para asegurar la efectividad y eficiencia del proceso de pruebas, utilizaremos una combinación de herramientas y metodologías específicas, alineadas con la arquitectura de microservicios y las tecnologías seleccionadas.

**1. Herramientas de Automatización de Pruebas**

* **JUnit 5 (Java):** Será el framework principal para las pruebas unitarias y de integración del backend desarrollado con Spring Boot. Nos permitirá verificar la lógica de negocio y la interacción entre los servicios.
* **Jasmine / Karma (Angular):** Estos frameworks se utilizarán para las pruebas unitarias y de componentes del frontend en Angular. Asegurarán la correcta funcionalidad de la interfaz de usuario a nivel granular.
* **Postman:** Imprescindible para la ejecución manual y la automatización de colecciones de pruebas de las APIs RESTful. Esto nos permitirá validar la integración entre el frontend y el backend, así como la funcionalidad de los servicios de forma independiente.

**2. Software de Gestión de Pruebas**

* **Trello:** Nuestra herramienta principal para la gestión de tareas y seguimiento de incidencias (bugs). Utilizaremos tableros Kanban para organizar los casos de prueba, registrar resultados, asignar defectos y monitorear su resolución. Dada la naturaleza del proyecto, la flexibilidad de Trello se adapta bien a la gestión ágil.

**3. Herramientas de Integración y Continuidad**

* **GitHub Actions:** Integrado directamente con nuestro repositorio de código en GitHub, se configurará para la integración continua (CI). Esto automatizará la construcción del código, la ejecución de pruebas unitarias y la verificación de calidad en cada push o pull request, asegurando la identificación temprana de problemas.
* **Docker:** Más allá de la contenedorización, Docker Compose será una herramienta fundamental para establecer entornos de prueba consistentes y reproducibles a nivel local, permitiendo a los desarrolladores replicar el entorno de despliegue.

**Personal**

Para llevar a cabo las actividades de pruebas de manera efectiva y dada la estructura del equipo, se definen los siguientes roles y responsabilidades.

* **Profesor: Supervisor del Proyecto / Mentor de Calidad**

**- Responsabilidades:** Es el responsable último de la planificación general de la calidad y la supervisión del proceso de pruebas. Revisa la estrategia, los criterios de aceptación y la adherencia a los estándares. Proporciona orientación y toma decisiones clave sobre la aceptación o suspensión de las pruebas. Actúa como un garante de la calidad final del producto.

* **Matias Fuentes: Desarrollador Backend / Tester de API e Integración**

**- Responsabilidades:** Como desarrollador principal del backend, será el encargado de escribir y ejecutar las pruebas unitarias y de integración para los microservicios Spring Boot. Utilizará Postman para las pruebas de API.

* **Giovanny Otárola: Desarrollador Frontend / Tester Funcional y de Usabilidad**

**- Responsabilidades:** Como desarrollador principal del frontend, se encargará de escribir y ejecutar las pruebas unitarias y de componentes para Angular. Su rol incluirá la ejecución de pruebas funcionales exhaustivas de la interfaz de usuario y la validación de los flujos de negocio desde la perspectiva del usuario final. Será el responsable principal de las pruebas de usabilidad (UX/UI), recopilando feedback y asegurando una experiencia de usuario óptima. Participará en la validación de la integración entre frontend y backend.

**Dinámica de Colaboración**

Ambos estudiantes compartirán la responsabilidad de documentar los casos de prueba, registrar y hacer seguimiento de los defectos en Trello, y proporcionar evidencias de las pruebas. La supervisión del profesor garantizará que la calidad sea una prioridad constante y que los estándares definidos en este plan se cumplan rigurosamente. Este enfoque permite maximizar el aprendizaje y la responsabilidad de cada miembro del equipo, aprovechando sus conocimientos específicos en cada dominio.

**Planificación y Organización**

**Procedimientos para las Pruebas**

Nuestra estrategia de pruebas se basa en un enfoque ágil y continuo, integrando las pruebas en cada fase del desarrollo.

**Ciclo de Vida del Caso de Prueba**

Cada prueba pasará por un ciclo definido: diseño, ejecución, registro de defectos (en Trello) y re-ejecución tras las correcciones.

**Tipos y Enfoques de Pruebas**

Aplicaremos diversas técnicas:

* **Caja Blanca:** Los desarrolladores revisarán el código (**Spring Boot**, **Angular**) para calidad y lógica interna (con JUnit, SonarQube, Jasmine/Karma).
* **Caja Negra:** Se centrará en la funcionalidad externa de la aplicación, incluyendo:

1. **Funcionales:** Validar que cada característica cumpla los requisitos.
2. **Integración:** Probar la comunicación entre microservicios, y con AWS Cognito, AWS S3 y Azure Functions (usando Postman).
3. **Sistema:** Probar la aplicación completa.

* **Regresión:** Ejecutar pruebas existentes tras cada cambio para asegurar que no se introduzcan nuevos errores. Se priorizará la automatización.
* **Usabilidad:** Asegurar una experiencia de usuario intuitiva y satisfactoria en el frontend.
* **Compatibilidad:** Verificar el funcionamiento en navegadores modernos y diferentes dispositivos.
* **Basada en Riesgos:** Priorizar pruebas en las áreas más críticas o de mayor riesgo del sistema.

Este enfoque integral garantizará la calidad de la aplicación, adaptándose a la naturaleza ágil del proyecto.

**Matriz de Responsabilidades**

Para asegurar una clara asignación de tareas y responsabilidades en el proceso de pruebas, se ha elaborado la siguiente matriz RACI. Esta matriz define quién es **Responsable (R)** de ejecutar una tarea, quién **Aprueba (A)** su finalización, quién debe ser **Consultado (C)** antes de tomar una decisión o realizar una acción, y quién debe ser **Informado (I)** de los resultados o del progreso.

|  |  |  |  |
| --- | --- | --- | --- |
| Actividad Clave de Pruebas | Profesor (Supervisor / Mentor de Calidad) | Matias Fuentes  (Dev Backend / Tester API) | Giovanny Otárola(Dev Frontend / Tester Funcional) |
| Planificación del Plan de Pruebas | A | R | R |
| Diseño de Casos de Prueba (Backend) | I | R | C |
| Diseño de Casos de Prueba (Frontend) | I | C | R |
| Ejecución de Pruebas Unitarias (Backend) | I | R | I |
| Ejecución de Pruebas Unitarias (Frontend) | I | I | R |
| Ejecución de Pruebas de Integración (APIs) | I | R | C |
| Ejecución de Pruebas Funcionales (End-to-End) | I | C | R |
| Ejecución de Pruebas de Rendimiento | C | R | I |
| Ejecución de Pruebas de Usabilidad | C | I | R |
| Registro y Seguimiento de Defectos en Trello | I | R | R |
| Priorización y Asignación de Defectos | A | R | R |
| Actualización de Documentación de Pruebas | C | R | R |
| Generación de Informe Final de Pruebas | A | R | I |

**Cronograma**

El cronograma del plan de pruebas se ha diseñado con base en estimaciones realistas del equipo, considerando la naturaleza ágil del proyecto y la interacción continua entre desarrollo y pruebas. Las actividades de prueba se integrarán a lo largo de cada sprint de desarrollo, con hitos clave marcando la finalización de fases de prueba específicas.

Dado que estamos en un entorno ágil con iteraciones cortas, el cronograma se presentará de forma que muestre la dedicación a las pruebas en cada sprint, asumiendo sprints de dos semanas.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hito / Fase | Actividad Clave | Duración (Semanas) | Semanas del Proyecto (Ejemplo) | Responsable Principal | Entregables Clave |
| Sprint 1 | Configuración Inicial y Pruebas Unitarias/Integración (Backend) | 1 | Semana 7 | Desarrollador Backend | Casos de Prueba Unitarios (Backend), Registros de Ejecución |
|  | - Configuración del entorno de pruebas (Docker, EC2) |  |  |  |  |
|  | - Diseño de casos de prueba unitarios y de integración para microservicios base |  |  |  |  |
|  | - Ejecución de pruebas unitarias y de integración de APIs (Postman) |  |  |  |  |
|  | - Registro inicial de defectos |  |  |  |  |
| Sprint 2 | Pruebas Funcionales (CRUD Core) y Frontend Básico | 1 | Semana 7 | Desarrollador Frontend | Casos de Prueba Funcionales (CRUD), Registros de Ejecución, Defectos en Trello |
|  | - Diseño de casos de prueba funcionales para la gestión de inventario (CRUD) |  |  |  |  |
|  | - Ejecución de pruebas funcionales del frontend integrado con el backend (CRUD) |  |  |  |  |
|  | - Pruebas Unitarias/Componentes de Frontend (Jasmine/Karma) |  |  |  |  |
| Sprint 3 | Pruebas de Integración Avanzadas y Módulos Secundarios | 1 | Semana 7 | Desarrollador Backend | Informes de Pruebas de Integración, Actualización de Casos de Prueba |
|  | - Pruebas de integración con AWS Cognito y AWS S3 |  |  |  |  |
|  | - Diseño y ejecución de pruebas para el módulo de gestión documental |  |  |  |  |
|  | - Pruebas de regresión selectivas |  |  |  |  |
|  | - Continuar registro y seguimiento de defectos |  |  |  |  |
| Sprint 4 | Pruebas de Alertas, Rendimiento y Seguridad Inicial | 1 | Semana 7 | Desarrollador Backend | Registros de Pruebas de Alertas |
|  | - Diseño y ejecución de pruebas para el módulo de alertas (Azure Functions) |  |  |  |  |
| Sprint 5 | - Diseño y ejecución de pruebas de usabilidad (UX/UI) | 1 | Semana 7 | Desarrollador Frontend | Informe pruebas de usabilidad. |
|  | - Pruebas de compatibilidad en navegadores y dispositivos clave |  |  |  |  |
|  | - Resolución de defectos finales |  |  |  |  |
| Cierre de Pruebas | Análisis de Resultados y Generación de Informe Final | 1 | Semana 8 | Profesor / Ambos Desarrolladores | Informe Final de Pruebas, Resumen de Defectos |
|  | - Consolidación de todos los resultados de pruebas |  |  |  |  |

**Premisas**

Las siguientes premisas son condiciones o supuestos clave que se asumen al planificar y ejecutar las pruebas de software para este proyecto. Su cumplimiento es fundamental para el éxito del plan de pruebas. Cualquier desviación de estas premisas deberá ser comunicada y gestionada proactivamente.

* **Integración de las Pruebas con el Desarrollo:** Las actividades de pruebas se llevarán a cabo en paralelo con el desarrollo,con una integración continua de código y pruebas
* **Disponibilidad de Recursos Humanos:** Ambos desarrolladores estarán disponibles y dedicados a las actividades de desarrollo y pruebas según lo planificado en el cronograma, distribuyendo el 100% de su esfuerzo entre ambas tareas según la fase.
* **Disponibilidad de Entornos de Prueba:** El entorno de pruebas en AWS EC2 (para el backend) y la configuración de servicios como AWS Cognito, AWS S3, AWS Api Gateway y Azure Functions estarán disponibles, estables y correctamente configurados desde el inicio del Sprint 1, y se mantendrán así durante todo el ciclo de pruebas. Cualquier interrupción crítica será comunicada y resuelta en menos de 24 horas.
* **Requerimientos de Datos de Prueba:** Los datos de prueba necesarios para cada fase y tipo de prueba (datos de usuario, inventario, documentos, etc.) estarán disponibles y serán adecuados para simular escenarios reales y de borde. Los scripts para la carga y limpieza de datos de prueba serán desarrollados y probados con antelación.
* **Uso de Herramientas de Pruebas:** Las herramientas de pruebas seleccionadas (JUnit, Jasmine/Karma, Postman, Trello, GitHub) estarán instaladas, configuradas y operativas.
* **Alcance de las Pruebas Claramente Definido:** El alcance de las funcionalidades a probar y a no probar (según lo detallado en secciones previas) es claro y ha sido comunicado y aceptado por todas las partes interesadas (incluido el profesor).
* **Gestión de Defectos Efectiva:** Existirá un proceso definido y claro para la identificación, reporte, clasificación, asignación, corrección y verificación de defectos utilizando **Trello**. Se espera que los defectos críticos sean priorizados y resueltos rápidamente por el equipo de desarrollo.
* **Conocimiento del Equipo:** Los miembros del equipo (Giovanny y Matias) poseen la capacitación y las habilidades necesarias para ejecutar los tipos de pruebas planificados y para utilizar las herramientas seleccionadas.
* **Comunicación Continua:** Habrá una comunicación abierta y continua entre el equipo de desarrollo/pruebas y el profesor para discutir el progreso, los impedimentos y los resultados de las pruebas. Las reuniones de seguimiento de sprint servirán como punto de coordinación.

**Aprobación del Profesor:** La aprobación del profesor será requerida para la finalización de fases de prueba clave y para la aceptación final del producto antes del despliegue.

**Plan de Comunicaciones**

**Restricciones y Premisas**

Para asegurar una gestión de comunicaciones efectiva en nuestro proyecto de aplicación de gestión de recursos tecnológicos, es crucial definir las restricciones que limitarán nuestras opciones y las premisas que asumimos como verdaderas. Estas guiarán cómo, cuándo y con quién nos comunicaremos.

**Restricciones**

Las restricciones definen los límites dentro de los cuales operará nuestro plan de comunicación:

* **Restricciones de Tiempo:** El tiempo disponible para la comunicación es intrínsecamente limitado por el cronograma general del proyecto.
* **Restricciones de Recursos:** La disponibilidad de recursos para actividades de comunicación es limitada al equipo de proyecto y las herramientas ya existentes.
* **Restricciones de Herramientas de Comunicación:** El equipo utilizará un conjunto específico y limitado de herramientas ya familiarizadas, lo que influirá en el formato y la frecuencia de la comunicación.
* **Restricciones Geográficas:** Los miembros del equipo (dos estudiantes) y el profesor se encuentran en diferentes ubicaciones, lo que puede afectar la disponibilidad para reuniones síncronas.
* **Restricciones de Confidencialidad y Seguridad:** La información del proyecto, especialmente la relacionada con configuraciones de credenciales, acceso a servicios en la nube y datos de prueba, debe ser tratada con estricta confidencialidad.
* **Restricciones de Presupuesto:** El proyecto opera con un presupuesto de comunicación mínimo, lo que limita la adquisición de software o servicios de comunicación de pago.

**Premisas**

Las premisas son supuestos que consideramos verdaderos para que el plan de comunicación sea efectivo:

* **Compromiso de los Interesados:** Se asume que todos los interesados clave (profesor, estudiantes) están comprometidos con el éxito del proyecto y participarán activamente en las comunicaciones según lo planificado.
* **Habilidades de Comunicación del Equipo:** Se presume que los estudiantes desarrolladores poseen las habilidades básicas de comunicación necesarias para reportar el progreso, documentar problemas y participar en discusiones técnicas.
* **Disponibilidad de Conectividad:** Se asume que todos los miembros del equipo tendrán acceso a una conexión a internet estable y fiable para participar en reuniones virtuales y acceder a las plataformas de comunicación.
* **Adherencia a los Canales Definidos:** Se espera que la comunicación se realice preferentemente a través de los canales y formatos definidos en este plan para mantener la coherencia y la trazabilidad.
* **Claridad en los Roles y Responsabilidades:** Se asume que las responsabilidades de comunicación es clara y entendida por todos los involucrados.

**Requisitos de Comunicaciones de los Interesados**

Identificar los requisitos de comunicación de cada interesado es fundamental para garantizar que la información fluya de manera efectiva y satisfaga sus expectativas

**Tabla de Requerimientos de Comunicación del Proyecto**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Comunicación | Objetivo | Contenido | Formato | Medio | Frecuencia | Plazo para confirmar recepción | Responsable | Aprobador | Audiencia / Receptores |
| Reunión de Sincronización | Actualizar sobre progreso y discutir impedimentos. | Avance de tareas, problemas encontrados, próximos pasos, demos de funcionalidades. | Oral / Visual (Demo) | Videoconferencia | Bisemanal | N/A | Desarrolladores(ambos) | Profesor | Profesor, Desarrolladores(ambos) |
| Informe de Progreso (Resumen) | Mantener al profesor informado sobre el estado general del proyecto. | Avance general, hitos alcanzados, riesgos/problemas clave, % de completitud. | Documento PDF/PPT | Correo Electrónico | Bisemanal | 24 horas | Desarrollador 1 / Desarrollador 2 | Profesor | Profesor |
| Actualizaciones de Tareas y Defectos | Asegurar transparencia y seguimiento del trabajo y problemas. | Estado de tareas, asignaciones, descripciones de defectos, comentarios, adjuntos. | Texto / Kanban | Trello | Continuo | N/A (visibilidad directa) | Desarrolladores(ambos) | N/A | Desarrolladores (ambos), Profesor |
| Comunicación Operativa Rápida | Coordinación diaria, preguntas rápidas, avisos urgentes. | Dudas técnicas, notificaciones breves, confirmaciones rápidas. | Mensaje de texto | WhatsApp/Discord | Diario | Inmediata | Desarrolladores(ambos) | N/A | Desarrolladores(ambos) |
| Revisión de Código / Pull Requests | Asegurar la calidad del código y la implementación de funcionalidades. | Cambios en el código, comentarios, solicitudes de revisión. | Código / Comentarios | GitHub | Continua | 24 horas | Desarrolladores(ambos) | Desarrollador 1 / Desarrollador 2 | Desarrolladores (ambos) |
| Notificaciones de Entorno/Infraestructura | Informar sobre estado, incidentes o cambios en los entornos de desarrollo/pruebas. | Disponibilidad de servicios, problemas de despliegue, actualizaciones. | Texto / Correo | WhatsApp/Discord / Correo | Según necesidad | 4 horas | Desarrollador 1 | N/A | Desarrolladores 2, Profesor |
| Informe de Riesgos y Mitigaciones | Comunicar proactivamente los riesgos identificados y sus planes de respuesta. | Descripción del riesgo, probabilidad, impacto, plan de mitigación/contingencia. | Documento PDF | Correo Electrónico | Mensual/Según necesidad | 48 horas | Desarrollador 1 / Desarrollador 2 | Profesor | Profesor |

**Recursos asignados a actividades de comunicaciones**

Los recursos para las actividades de comunicación son esenciales para su buen funcionamiento y se gestionarán de la siguiente manera:

* **Herramientas de Comunicación:**
* **Trello:** Principal herramienta para la gestión de tareas, historias de usuario, seguimiento de defectos y documentación de las actividades diarias.
* **Discord:** Para la comunicación rápida, avisos urgentes y coordinación operativa entre los estudiantes.
* **Correo Electrónico:** Para comunicaciones formales, envío de informes y avisos importantes al profesor.
* **Teams - Discord:** Para las reuniones bisemanales de sincronización y cualquier reunión ad-hoc que requiera interacción en tiempo real.
* **GitHub:** Para la gestión de control de versiones del código y la revisión de pull requests, que incluye comentarios y discusiones técnicas.
* **Personal de Comunicación:** Las actividades de comunicación serán responsabilidad directa de los Estudiantes Desarrolladores, quienes actuarán como comunicadores del proyecto. El Profesor actuará como el principal receptor y aprobador de las comunicaciones formales.
* **Tecnologías y Software:**
* Computadoras personales de los estudiantes con acceso a internet.
* Cuentas de correo electrónico institucionales o personales.
* Acceso a las plataformas web de Trello, GitHub,Teams,Discord.

**Diagrama de Flujo de Información**

*Diagrama

El contenido generado por IA puede ser incorrecto.*

**Descripción del Flujo:**

1. **Desarrollo y Actualización de Tareas:** Los Desarrolladores 1 y 2 actualizan continuamente el estado de sus tareas y defectos en **Trello** a medida que desarrollan y prueban.
2. **Control de Versiones y Revisión de Código:** El código se gestiona en **GitHub**, donde los estudiantes suben sus cambios y realizan revisiones de código mutuamente.
3. **Comunicación Rápida:** Para consultas o avisos urgentes entre estudiantes, se utiliza **Discord**.
4. **Informes Formales:** Periódicamente, los estudiantes compilan la información de Trello y otras fuentes para generar informes de progreso y riesgos, que se envían al profesor por **correo electrónico**.
5. **Reuniones de Sincronización:** Las reuniones bisemanales se realizan por **videoconferencia** (Teams y Discord), donde se discute el progreso, se resuelven impedimentos y se planifican los siguientes pasos. El profesor participa para revisar y dar feedback.
6. **Revisión y Feedback del Profesor:** El profesor revisa los informes por correo electrónico y el progreso directamente en Trello y GitHub. Durante las reuniones, proporciona feedback y aprobaciones, que guían a los estudiantes en las siguientes acciones.

Este flujo asegura que la información clave sea accesible, que los canales adecuados se utilicen para cada tipo de comunicación y que haya una retroalimentación constante para mantener el proyecto en curso.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PLAN DE TRABAJO DEL PROYECTO | | | | | | |
| Nombre de Actividades/Tareas | **Descripción Actividades/Tareas** | **Recursos** | **Duración de la actividad** | **Responsable** | **Entregable** | **Observaciones** |
| Elaboración de objetivos del proyecto GIRET | Definir objetivos específicos, medibles, alcanzables, relevantes y temporales. | Google Docs, Reunión con equipo | 2 días | Ambos | Documento de objetivos proyecto GIRET | Puede haber dificultad para delimitar objetivos realistas en poco tiempo. |
| Redacción de la descripción funcional del sistema | Describir cómo funcionará el sistema desde el punto de vista del usuario. | Google Docs | 2 días | Ambos | Documento de descripción funcional | Requiere comprensión total del problema y alcance. |
| Identificación de requisitos funcionales y no funcionales | Listar las funciones que el sistema debe cumplir. | Entrevistas, Google Docs | 4 días | Ambos | Lista de requisitos funcionales | Es clave validar bien con el profesor para no omitir requerimientos. |
| Validación de requisitos con el profesor | Revisar y aprobar requisitos con el profesor. | Reunión, Documentos previos | 1 días | Ambos | Acta de validación | Depende de la disponibilidad del profesor para retroalimentar. |
| Diseño de arquitectura de módulos (diagrama) | Diseñar los componentes principales del sistema y sus interacciones. | Draw.io, Google Docs | 2 días | Ambos | Diagrama de arquitectura | Dificultad a la hora de definir límites precisos entre los módulos. |
| Asignación de tecnologías a cada módulo | Seleccionar las tecnologías específicas para cada parte del sistema. | Documentación técnica | 1 días | Ambos | Documento de tecnologías asignadas | Facilitado por experiencia previa en tecnologías usadas. |
| Configuración de Trello con columnas por sprint | Organizar el tablero de Trello según la metodología Scrum. | Trello | 0-5 días | Ambos | Tablero Trello configurado | Actividad sencilla si hay claridad en fases del proyecto. |
| Creación de repositorio GitHub con ramas base | Crear repositorio y ramas iniciales para control de versiones. | GitHub | 0-5 días | Ambos | Repositorio con estructura base | Facilitado por experiencias previas en el uso de la herramienta Git. |
| Diseño de mockups | Crear prototipos visuales de la interfaz. | Balsamiq | 2 días | Giovanny Otarola | Mockups | Facilitado al ser una herramienta visual moderna. |
| Implementación de componentes básicos UI | Programar los primeros elementos de interfaz gráfica. | Angular, Angular Material, VS Code | 3 días | Giovanny Otarola | Componentes UI funcionales | Dificultad a la hora de mantener una consistencia visual. |
| Creación de entidades y repositorios base | Definir clases de entidad y repositorios JPA en backend. | Spring Boot, JPA, Oracle Cloud. | 2 días | Matias Fuentes | Código fuente del modelo y repositorios | Facilitado al tener una estructura clara desde el diseño. |
| Exposición de APIs REST CRUD básicas | Crear controladores para exponer operaciones básicas sobre entidades. | Spring Boot, Postman | 3 días | Matias Fuentes | Endpoints funcionales | Requiere validación con postman. |
| Configuración de AWS Cognito | Integrar sistema de autenticación con Cognito. | Cuenta AWS, Documentación oficial | 2 días | Matias Fuentes | Autenticación integrada | Puede fallar la integración si faltan permisos. |
| Desarrollo de Azure Function | Programar función en la nube que modifiquen el estado. | Azure Portal, Java. | 2 días | Matias Fuentes | Azure Function funcional | Posibles errores en los triggers. |
| Integración de carga de archivos a AWS S3 | Permitir almacenamiento de archivos en la nube. | AWS S3. | 3 días | Giovanny Otarola | Subida de archivos funcional | Requiere gestión de credenciales y permisos. |
| Conexión frontend con backend por API Gateway | Establecer comunicación entre cliente y servidor a través de API Gateway. | Angular, Spring Boot. | 2 días | Ambos | Conexión establecida | Dificultad de configuración CORS y endpoints. |
| Ejecución de pruebas unitarias y de integración | Validar el correcto funcionamiento del sistema. | JUnit, Postman, Jasmine/Karma. | 2 días | Ambos | Reporte de pruebas | Facilitador: frameworks disponibles. |
| Corrección de errores críticos encontrados | Resolver bugs importantes detectados durante pruebas. | IDE, GitHub. | 2 días | Ambos | Código corregido | Dificultad: errores interdependientes entre capas. |
| Pruebas de rendimiento con JMeter | Evaluar la capacidad del sistema bajo carga. | Apache JMeter. | 2 días | Matias Fuentes | Reporte de rendimiento | Dificultad: configuración realista de escenarios. |
| Documentación técnica y manuales de usuario | Redactar documentación de instalación, uso y mantenimiento. | Google Docs, Word. | 2 días | Ambos | Documentación completa | Facilitador: trabajo colaborativo. |
| Preparación de presentación final | Elaborar los contenidos y materiales para presentar el proyecto. | PowerPoint. | 2 días | Ambos | Presentación final | Facilitador: experiencia previa en presentaciones. |
| Defensa y demostración del sistema | Exposición del sistema ante el profesor. | Videoconferencia, proyecto desplegado. | 1 días | Ambos | Evaluación final | Dificultad: manejo de preguntas imprevistas. |





Reservados todos los derechos Fundación Instituto Profesional Duoc UC. No se permite copiar, reproducir, reeditar, descargar, publicar, emitir, difundir, de forma total o parcial la presente obra, ni su incorporación a un sistema informático, ni su transmisión en cualquier forma o por cualquier medio (electrónico, mecánico, fotocopia, grabación u otros) sin autorización previa y por escrito de Fundación Instituto Profesional Duoc UC La infracción de dichos derechos puede constituir un delito contra la propiedad intelectual.