5. Componente Teórico (70 minutos)

Definir los siguientes conceptos desde su punto de vista y experiencia:

● ¿Cómo es el proceso de desarrollo de software utilizando metodologías tradicionales y ágiles?

Se debe tener en cuenta que todo proceso puede variar según el entorno en que se encuentre, y según las necesidades de cada organización.

En las metodologías tradicionales se suelen enfocar en una planificación bastante detallada en sus inicios, mientras que en las metodologías ágiles se va iniciando con una pequeña idea, puede ser, y a medida que van avanzando se van agregando mejoras y requerimientos, permitiendo que el proceso sea adaptativo y flexible.

● ¿Cuál es el rol de analista de calidad de software en el proceso de desarrollo de software en proyectos donde se utilice metodologías ágiles?

El analista de calidad debe garantizar siempre la entrega de un producto de calidad, confiable y funcional según se ha solicitado.

Siempre debe estar dispuesto a trabajar con todo el equipo para establecer criterios de calidad que sean claros para todos, teniendo en cuenta el o los alcances del proyecto.

Es muy importante su presencia en todas las reuniones desde el inicio del proyecto, con esto se aclaran dudas en una etapa temprana y se pueden evitar futuros fallos que pueden incurrir en altos costos.

Está en la obligación de ofrecer una retroalimentación detallada y clara del avance del proyecto, toda vez que esto permita avanzar con claridad, seguridad y evitando posibles fallos.

Debe participar siempre en las retrospectivas para identificar posibles áreas de mejora en cada uno de los procesos que son objetos de pruebas, bridando con esto un resultado de alta calidad, de manera iterativa y en forma continua.

● Cuál es la principal diferencia entre:
A. Verificar y Validar

La verificación se realiza con pruebas estáticas, es decir, sobre la documentación que se tenga, antes de iniciar ejecución sobre el producto funcional; con la verificación se busca asegurarse que se está realizando de manera adecuada el producto solicitado.

La validación se hace con pruebas dinámicas, es decir, ejecutando el producto; con la validación se busca asegurar de si en realidad se está construyendo lo correcto.

B. Issue y Bug

Básicamente un Issue es cualquier hallazgo que debe tener un seguimiento hasta que sea resuelto; y un Bug es un subgrupo de issues; hay varios tipos de issues: Bug, Task, Quetions, etc. Por lo tanto todos los bugs son issues, pero no todos los issues son bugs.

C. Novedad y Hallazgo

Una novedad hace referencia a algo que recientemente se encontró, a algo nuevo; mientras que un hallazgo se refiere a descubrir o encontrar algo después de una investigación. La novedad se enfoca en la actualidad, el hallazgo puede estar enfocado en algo antiguo.

● Explique e indique cuáles son los entregables de las siguientes etapas:

A. Planeación: La etapa de planeación es donde se deben definir los objetivos, las estrategias, los recursos y los cronogramas necesarios para llevar a cabo el proyecto con éxito y calidad.

Entregables:

1. **Plan de Pruebas**: Aquí se describe el alcance, enfoque, recursos, y cronograma de las actividades de prueba que se llevarán a cabo.
2. **Estrategia de Pruebas**: Menciona los tipos de pruebas a realizar y en qué momento de implementará cada uno, también define herramientas a utilizar, y métodos de evaluación.
3. **Cronograma del Proyecto**: Debe ser un calendario detallado que incluye las fechas de inicio y fin de cada actividad de prueba, teniendo en cuenta que puede haber cierto desfase en ello.
4. **Asignación de Recursos**: Listado de los recursos humanos y materiales necesarios que se utilizarán, incluyendo roles y responsabilidades de cada uno.
5. **Riesgos y Mitigaciones**: Identificación de posibles riesgos que pueden presentarse, y planes para mitigarlos.
6. **Criterios de Aceptación**: Define los criterios que determinarán si las pruebas han sido exitosas o no; con ello se determina su puesta en producción u otras alternativas a tomar en cuenta.

B. Diseño de Escenarios y Casos de Prueba: Aquí se diseñan los escenarios y casos de prueba teniendo en cuenta los requisitos solicitados.

Entregables:

1. **Escenarios de Prueba**: Describe una situación que el producto (software) debe representar de manera adecuada.
2. **Casos de Prueba**: Describe las condiciones de prueba, los pasos a seguir, datos de entrada, y resultados esperados para cada escenario, estado del caso de pruebas, se debe tener en cuenta que estos ítems pueden variar, o se pueden incluir más o quitar algunos.
3. **Matriz de Trazabilidad**: Es un documento donde se relaciona los casos de prueba con los requisitos del sistema para asegurar que todos los requisitos están cubiertos por las pruebas y los tiempos estimados.
4. **Ambiente de Prueba**: Entorno donde se llevarán a cabo las pruebas, se configurará según el producto a probar y siempre debe ser lo más similar al entorno productivo.

C. Ejecución: Aquí es donde se llevan a cabo las pruebas según lo planificado y documentado en las etapas anteriores. Los resultados se deben registran y se analizará el comportamiento del producto para asegurar su calidad y eficacia.

Entregables:

1. **Resultados de Pruebas**: Registro detallado de los resultados que se han obtenidos para cada caso de prueba ejecutado; incluyendo si fue exitoso o si falló.
2. **Registros de Incidentes**: Documentación de cualquier problema encontrado durante la ejecución de las pruebas, incluye descripción al detalle, imágenes o videos indicando dónde se encuentra la falla y pasos para reproducir el problema.
3. **Evidencias de Pruebas**: Capturas de pantalla, logs, videos y cualquier otra evidencia que respalde los resultados obtenidos.

4. **Reporte de Avances**: Informes periódicos sobre el progreso de las pruebas, estado de los casos de prueba, y cualquier bloqueo o desviación que se tenga en el momento.

D. Reporte de Hallazgos:
Esta etapa se centra en analizar y reportar los resultados de las pruebas, teniendo en cuenta los problemas encontrados y brindando recomendaciones para resolverlos de la manera adeucada.

Entregables:

1. **Informe de Bugs**: Documento que lista todos los bugs encontrados, incluyendo detalles como descripción, gravedad, prioridad, pasos para reproducir, y estado actual.
2. **Análisis de Hallazgos**: Evaluación de los resultados de las pruebas para identificar patrones, áreas problemáticas, y posibles causas de los fallos, con esto se busca evitar los mismos inconvenientes para futuros proyectos.
3. **Informe de Calidad del Producto**: Resumen general de la calidad del software basado en los resultados de las pruebas y los hallazgos reportados, teniendo en cuenta lo solicitado en un principio.
4. **Recomendaciones**: Sugerencias específicas y puntuales para solucionar los problemas encontrados y mejorar la calidad del software, con esto se pretende obtener confianza por parte de todo el equipo y por parte del cliente.

E. Feedback:
En esta etapa, se recopila y analiza la retroalimentación de las partes interesadas sobre el proceso de pruebas y los resultados obtenidos, con el objetivo de mejorar futuras iteraciones de pruebas y buscar siempre la confianza por parte de todo el equipo.

Entregables:

1. **Informe de Retroalimentación**: Documento que recopila toda la información durante el proceso de pruebas como comentarios y sugerencias de los miembros del equipo, stakeholders, y usuarios.
2. **Reuniones de Retrospectiva**: Actas de reuniones en las que se discuten las lecciones aprendidas, lo que funcionó bien, y las áreas de mejora, importante la participación de todo el equipo.
3. **Plan de Mejora Continua**: Propuestas de acciones puntuales para mejorar el proceso de pruebas en el futuro, teniendo en cuenta la retroalimentación recibida.
4. **Actualización del Plan de Pruebas**: Revisión, actualización y mejora del plan de pruebas inicial para incorporar las lecciones aprendidas y mejorar el proceso para futuros proyectos, garantizando siempre la calidad.

● ¿Indique con qué herramientas de gestión de pruebas ha trabajado y cuál es la de su preferencia y porque?

Según la solicitud de cada cliente, he trabajado con las siguientes herramientas de gestión de pruebas:

Jira: Lo he usado para reportar los hallazgos encontrados, desde allí se hace seguimiento y se validad su corrección con pruebas de regresión para poder cerrar el ticket, o de lo contrario abrir uno nuevo.

Mantis: Es otro buck tracker que he usado para reportar fallos y hacer el debido seguimiento de cada uno, hasta su solución definitiva y correcta

TestLink: Lo he usado para diseñar, ejecutar y sacar reportes de los casos de pruebas. Es una herramienta muy fácil de usar y aporta agilidad al momento de sacar informes del avance de ejecución.

RedMine: Lo he usado para el registro de mis actividades diarias, permite llevar un adecuado uso de los tiempos.

● ¿Qué se realiza en cada uno de los eventos Scrum?

1. **Sprint**

Los Sprints son el corazón de Scrum, aaquí es donde las ideas se convierten en valor.
Son eventos de duración fija de un mes o menos para crear consistencia. Un nuevo Sprint comienza inmediatamente después de la conclusión del Sprint anterior.
Todo el trabajo necesario para lograr el Objetivo del Producto, incluido la Sprint Planning, Daily Scrums, Sprint Review y Sprint Retrospective, ocurre dentro de los Sprints.
Un Sprint podría cancelarse si el Objetivo del Sprint se vuelve obsoleto. Solo el Product Owner tiene la autoridad para cancelar el Sprint.

Durante el Sprint:

● No se realizan cambios que pongan en peligro el Objetivo del Sprint;
● La calidad no disminuye;
● El Product Backlog se refina según sea necesario; y,
● El alcance se puede aclarar y renegociar con el Product Owner a medida que se aprende más.

Un Sprint podría cancelarse si el Objetivo del Sprint se vuelve obsoleto. Solo el Product Owner tiene la autoridad para cancelar el Sprint.

2. **Sprint Planning**

La Sprint Planning inicia el Sprint al establecer el trabajo que se realizará para el Sprint. El Scrum Team crea este plan resultante mediante trabajo colaborativo.
El Product Owner se asegura de que los asistentes estén preparados para discutir los elementos más importantes del Product Backlog y cómo se relacionan con el Objetivo del Producto. El Scrum Team también puede invitar a otras personas a asistir a la Sprint Planning para brindar asesoramiento.

La Sprint Planning aborda los siguientes temas:

Tema uno: ¿Por qué es valioso este Sprint?

El Product Owner propone cómo el producto podría Incrementar su valor y utilidad en el Sprint actual. Luego, todo el Scrum Team colabora para definir un Objetivo del Sprint que comunica por qué el Sprint es valioso para los interesados. El Objetivo del Sprint debe completarse antes de que termine la Sprint Planning.

Tema dos: ¿Qué se puede hacer en este Sprint?

A través de una conversación con el Product Owner, los Developers seleccionan elementos del Product Backlog para incluirlos en el Sprint actual. El Scrum Team puede refinar estos elementos durante este proceso, lo que aumenta la comprensión y la confianza.

Tema tres: ¿Cómo se realizará el trabajo elegido?

Para cada elemento del Product Backlog seleccionado, los Developers planifican el trabajo necesario para crear un Increment que cumpla con la Definición de Terminado. A menudo, esto se hace descomponiendo los elementos del Product Backlog en elementos de trabajo más pequeños de un día o menos. La forma de hacerlo queda a criterio exclusivo de los Developers. Nadie más les dice cómo convertir los elementos del Product Backlog en Increments de valor.

3. Daily Scrum

El propósito de la Daily Scrum es inspeccionar el progreso hacia el Objetivo del Sprint y adaptar el Sprint Backlog según sea necesario, ajustando el trabajo planificado entrante.
La Daily Scrum es un evento de 15 minutos. Para reducir la complejidad, se lleva a cabo a la misma hora y en el mismo lugar todos los días hábiles del Sprint. Si el Product Owner o Scrum Master están trabajando activamente en elementos del Sprint Backlog, participan como Developers.
Los Developers pueden seleccionar la estructura y las técnicas que deseen, siempre que su Daily Scrum se centre en el progreso hacia el Objetivo del Sprint y produzca un plan viable para el siguiente día de trabajo. Esto crea enfoque y mejora la autogestión.
Las Daily Scrums mejoran la comunicación, identifican impedimentos, promueven la toma rápida de decisiones y, en consecuencia, eliminan la necesidad de otras reuniones.
La Daily Scrum no es el único momento en el que los Developers pueden ajustar su plan. A menudo se reúnen durante el día para discusiones más detalladas sobre cómo adaptar o volver a planificar el resto del trabajo del Sprint.

4. Sprint Review

El propósito de la Sprint Review es inspeccionar el resultado del Sprint y determinar futuras adaptaciones. El Scrum Team presenta los resultados de su trabajo a los interesados clave y se discute el progreso hacia el Objetivo del Producto.
Durante el evento, el Scrum Team y los interesados revisan lo que se logró en el Sprint y lo que ha cambiado en su entorno. Con base en esta información, los asistentes colaboran sobre qué hacer a continuación. El Product Backlog también se puede ajustar para satisfacer nuevas oportunidades.
La Sprint Review es una sesión de trabajo y el Scrum Team debe evitar limitarla a una presentación.
La Sprint Review es el penúltimo evento del Sprint y tiene un límite de tiempo de máximo cuatro horas para un Sprint de un mes. Para Sprints más cortos, el evento suele ser de menor duración.

5. Sprint Retrospective

El propósito de la Sprint Retrospective es planificar formas de aumentar la calidad y la efectividad.
El Scrum Team inspecciona cómo fue el último Sprint con respecto a las personas, las interacciones, los procesos, las herramientas y su Definición de Terminado. Los elementos inspeccionados suelen variar según el ámbito del trabajo. Se identifican los supuestos que los llevaron por mal camino y se exploran sus orígenes. El Scrum Team analiza qué salió bien durante el Sprint, qué problemas encontró y cómo se resolvieron (o no) esos problemas.
El Scrum Team identifica los cambios más útiles para mejorar su efectividad. Las mejoras más impactantes se abordan lo antes posible. Incluso se pueden agregar al Sprint Backlog para el próximo Sprint.
La Sprint Retrospective concluye el Sprint. Tiene un tiempo limitado a máximo tres horas para un Sprint de un mes. Para Sprints más cortos, el evento suele ser de menor duración.

● Sobre Automatización y POO:
A. Mencione cuales son los pilares de Programación Orientada a Objetos que se deben considerar al automatizar pruebas?

Tener en cuenta que al automatizar pruebas con Programación Orientada a Objetos (POO), es muy importante considerar los cuatro pilares fundamentales de POO para asegurar un código eficiente, reutilizable y mantenible. Estos pilares son:

1. **Encapsulamiento**: Este principio se refiere a agrupar datos (atributos) y comportamientos (métodos) que operan sobre esos datos en una sola unidad, o clase. En el contexto de automatización de pruebas, el encapsulamiento ayuda a:

- **Proteger los datos**: Mantener los detalles internos de la implementación de los objetos ocultos para evitar acceso directo desde fuera de la clase, lo cual reduce errores y facilita el mantenimiento.
- **Modularidad**: Facilitar la creación de módulos de prueba independientes, cada uno con su propia responsabilidad, lo que simplifica el proceso de pruebas y la detección de errores.

2. **Abstracción**: Este pilar se enfoca en simplificar sistemas complejos al modelar clases con interfaces claras, ocultando los detalles de implementación innecesarios para el usuario final. En pruebas automatizadas, la abstracción permite:

- **Simplicidad**: Crear pruebas más legibles y comprensibles al enfocarse en las funcionalidades esenciales y omitir los detalles complejos de implementación.
- **Reutilización**: Utilizar interfaces y clases abstractas para definir estructuras de prueba que puedan ser extendidas y reutilizadas en diferentes contextos.

3. **Herencia**: La herencia permite crear nuevas clases basadas en clases existentes, compartiendo atributos y métodos comunes. En la automatización de pruebas, la herencia es útil para:

- **Reutilización de código**: Evitar la duplicación de código al crear clases de prueba derivadas de una clase base que contiene funcionalidades comunes.
- **Mantenimiento**: Facilitar las actualizaciones y correcciones al realizar cambios en una clase base, propagando automáticamente esas modificaciones a las clases derivadas.

4. **Polimorfismo**: Este principio permite que objetos de diferentes clases se traten como objetos de una clase común, generalmente a través de interfaces o clases base. En pruebas automatizadas, el polimorfismo facilita:

- **Flexibilidad**: Permitir que las pruebas sean más flexibles y escalables al tratar objetos de diferentes tipos de manera uniforme.
- **Extensibilidad**: Facilitar la integración de nuevas funcionalidades de prueba sin alterar el código existente, al definir comportamientos que pueden ser sobrescritos o extendidos en las clases derivadas.

B. ¿Qué papel juegan los principios SOLID en la automatización de pruebas?

Los principios SOLID son un conjunto de cinco principios de diseño orientados a objetos que ayudan a crear software más comprensible, flexible y mantenible. Aplicar estos principios en la automatización de pruebas es fundamental para asegurar que el código de prueba sea robusto y sostenible.

1. Principio de Responsabilidad Única (Single Responsibility Principle, SRP):

- Cada clase de prueba o módulo debe tener una única responsabilidad o propósito. Esto significa que una clase de prueba debe enfocarse únicamente en una parte específica de la funcionalidad del software bajo prueba.

2. Principio Abierto/Cerrado (Open/Closed Principle, OCP):

- Las clases de prueba deben estar abiertas para extensión pero cerradas para modificación. Esto significa que deberías poder agregar nuevas pruebas y funcionalidades sin modificar el código existente.

3. Principio de Sustitución de Liskov (Liskov Substitution Principle, LSP):

- Las clases derivadas deben ser sustituibles por sus clases base sin alterar el correcto funcionamiento del programa. En el contexto de pruebas, esto asegura que las clases de prueba que heredan de una clase base puedan ser utilizadas indistintamente sin romper la lógica de las pruebas.

4. Principio de Segregación de Interfaces (Interface Segregation Principle, ISP)

- Las clases de prueba deben depender solo de las interfaces que realmente utilizan. Este principio evita la creación de interfaces "gordas" con métodos que no son necesarios para todas las clases implementadoras. En la automatización de pruebas, aplicar el ISP significa definir interfaces específicas para diferentes aspectos de las pruebas, lo que facilita el mantenimiento y la evolución de las pruebas.

5. Principio de Inversión de Dependencia (Dependency Inversion Principle, DIP):

- Las clases de prueba deben depender de abstracciones (interfaces) en lugar de clases concretas. Esto significa que las dependencias deben ser inyectadas a través de la inversión de control (IoC) o inyección de dependencias (DI). En la automatización de pruebas, el DIP facilita la creación de pruebas más aisladas y configurables, donde las dependencias pueden ser sustituidas fácilmente por simulaciones (mocks) o stubs, permitiendo pruebas más eficientes y precisas.

C. ¿Cómo se definen los escenarios críticos a automatizar?

Se deben tener en cuenta aspectos muy importantes como lo son la identificación de los objetivos e identificación de posibles riesgos; se deben entender claramente los objetivos que se van a abordar y los requisitos que se deben tener previamente en el sistema para llevar a cabo toda la gestión.
Se deben evaluar los posibles riesgos que puedan tener impacto alto como puede ser pérdida de datos, brechas de seguridad o caída e intermitencia en el servicio.

Teniendo un enfoque en los escenarios de mayor impacto y riesgo, se permite asegurar que la automatización ofrece un beneficio máximo y eficiente, según se ha solicitado.

D. Defina la pirámide de automatización de Cohn

La pirámide de automatización de Cohn, propuesta por Mike Cohn, es un modelo que guía a los equipos de desarrollo y pruebas en la distribución de los diferentes tipos de pruebas automatizadas para maximizar la eficiencia y efectividad del proceso de testing.
La pirámide de automatización de Cohn es una guía valiosa para estructurar un conjunto balanceado de pruebas automatizadas. Al centrarse en una base sólida de pruebas unitarias, complementadas por pruebas de integración y un menor número de pruebas E2E, se asegura una cobertura efectiva de errores y una mayor eficiencia en el proceso de desarrollo y mantenimiento del software.

● Sobre servicios:
A. ¿Qué es un servicio?

Dentro de la automatización de pruebas, un "servicio" generalmente se refiere a un componente o módulo de software que proporciona una funcionalidad específica y está diseñado para ser utilizado por otras aplicaciones o servicios. Estos servicios son frecuentemente accesibles a través de una red, utilizando protocolos como HTTP/HTTPS, y expuestos mediante APIs (Application Programming Interfaces).

Un servicio es una unidad independiente de funcionalidad que puede ser probada de manera aislada o en conjunto con otros servicios. Los servicios son piezas clave en arquitecturas modernas como microservicios y servicios web, y su correcta operación es fundamental para el funcionamiento del sistema completo.

B. Diferencia entre SOAP y REST

Su diferencia está dada por su formato; SOAP es un protocolo de comunicación que utiliza XML exclusivamente, mientras que REST utiliza múltiples formatos para el intercambio de datos como puede ser JSON, XML, YAML o txt; por lo general JSON es el más usado.

SOAP es más adecuado para servicios empresariales que requieren seguridad y transacciones complejas, mientras que REST es preferido para aplicaciones web y móviles debido a su simplicidad y eficiencia.

C. Defina métodos GET, POST, PUT y DELETE

- **GET:** El método GET se utiliza para solicitar una representación de un recurso específico.
- Recupera datos del servidor, No modifica el recurso. Múltiples solicitudes GET al mismo recurso producen el mismo resultado.

- **POST:** El método POST se utiliza para enviar datos al servidor para crear un nuevo recurso. La solicitud POST puede tener efectos secundarios en el servidor.
- Crea nuevos recursos en el servidor. Múltiples solicitudes POST pueden crear múltiples recursos. Contiene los datos necesarios para crear el recurso.

- **PUT:** El método PUT se utiliza para actualizar un recurso existente con los datos proporcionados.
- Actualiza un recurso existente. Múltiples solicitudes PUT con el mismo conjunto de datos producen el mismo resultado. Contiene los datos completos del recurso.

- **Delete**: El método DELETE se utiliza para eliminar un recurso específico del servidor.
- Elimina recursos del servidor. Múltiples solicitudes DELETE al mismo recurso producen el mismo resultado (el recurso no existirá después de la primera solicitud).

D. Herramientas de ejecución de servicios trabajadas

- He usado POSTAM para realizar consultas, es decir para obtener ciertos datos los cuales se puedan modificar o eliminar, según la solicitud del caso de prueba, esto a partir de colecciones ya previamente creadas e importadas en la herramienta.

- Realicé un curso dentro de mi organización con SOAPIU, laboralmente no lo he usado.

E. ¿Cómo automatizar un servicio?

- **Configuración Inicial**: Importando las especificaciones de la API
- **Escribir Scripts de Prueba**: Crear las solicitudes para los endpoints y los métodos
- **Automatización y Ejecución**: Configurar colecciones de pruebas con casos de prueba definidos.
- **Análisis de Resultados**: Analizar los resultados de las pruebas para identificar fallos y problemas de rendimiento.

- Generar reportes detallados para informar sobre el estado de las pruebas y la calidad del servicio.

F. Validaciones que ha utilizado al momento de automatizar servicios

- **Códigos de Estado**: Verificar que los códigos de estado HTTP devueltos estén dentro de los rangos esperados (200 para éxito, 4xx para errores del cliente, 5xx para errores del servidor, etc.).
- **Headers**: Verificar la presencia y el contenido de headers importantes como Content-Type.
- **Estructura de Respuesta**: Validar la estructura y el formato de la respuesta, como JSON, XML o texto plano, para garantizar que cumpla con las especificaciones de la API.
- Verificar que las operaciones CRUD (Create, Read, Update, Delete) se realicen correctamente, creando, leyendo, actualizando y eliminando recursos según sea necesario.
- Confirmar que los endpoints y rutas de la API respondan correctamente y devuelvan los resultados esperados.

● Pruebas No Funcionales:
A. Explique las más utilizadas: Seguridad y Performance

Las pruebas de seguridad se centran en evaluar la capacidad de un sistema para proteger los datos y recursos sensibles contra amenazas y ataques maliciosos.

Las pruebas de performance se centran en evaluar cómo responde el sistema en términos de velocidad, escalabilidad y estabilidad bajo diferentes condiciones de carga y uso.

Tener en cuenta que las pruebas de seguridad y las pruebas de performance son críticas para garantizar que un sistema cumpla con los estándares de seguridad y rendimiento requeridos por los usuarios. Al realizar estas pruebas de manera regular y exhaustiva, los equipos pueden identificar y mitigar vulnerabilidades de seguridad, así como optimizar el rendimiento del sistema para proporcionar una experiencia de usuario óptima bajo diferentes condiciones de uso.

B. Herramientas utilizadas para ejecutar dichas pruebas

Entre las más conocidas están JMter y WAPT

C. Como realizar pruebas de carga con herramientas como JMETER, WAPT o BlazeMeter

Para cualquiera de las herramientas mencionadas, el deber ser para realizar pruebas de carga es: Configuración de pruebas y Ejecución de pruebas teniendo en cuanta las características específicas de cada una:

- Asegurarse de simular cargas realistas y representativas del tráfico esperado en producción.
- Monitorear activamente el rendimiento del servidor y ajustar la configuración de la prueba según sea necesario para optimizar el rendimiento y la estabilidad del sistema.
- Utilizar técnicas como la escalabilidad horizontal y vertical para aumentar la capacidad del sistema y manejar cargas de trabajo más grandes.

● Automatización de Móviles:
A. Explique los siguientes conceptos:app nativas, app híbridas, tipos de sistemas operativos.

1. Aplicaciones Nativas:
Las aplicaciones nativas son programas diseñados y desarrollados específicamente para funcionar en un sistema operativo particular, como iOS o Android. Estas aplicaciones están escritas utilizando lenguajes de programación. Las aplicaciones nativas generalmente tienen un alto rendimiento y pueden aprovechar al máximo las características y funcionalidades del dispositivo en el que se ejecutan. Además, suelen ofrecer una experiencia de usuario fluida y consistente.

2. Aplicaciones Híbridas:
Las aplicaciones híbridas son aquellas que combinan elementos de las aplicaciones web y las aplicaciones nativas. Las aplicaciones híbridas ofrecen la ventaja de poder desarrollar una sola base de código que se puede ejecutar en múltiples plataformas, lo que reduce el tiempo y los costos de desarrollo.

3. Tipos de Sistemas Operativos:
Los sistemas operativos pueden clasificarse en varias categorías, según su arquitectura y uso. Algunos de los tipos más comunes de sistemas operativos son:

a. Sistemas Operativos de Escritorio:

- Diseñados para su uso en computadoras personales y estaciones de trabajo.
- Ejemplos: Microsoft Windows, macOS de Apple y varias distribuciones de Linux.

b. Sistemas Operativos Móviles:

- Diseñados específicamente para dispositivos móviles como teléfonos inteligentes y tabletas.

c. Sistemas Operativos de Servidor:

- Optimizados para ejecutarse en servidores y manejar grandes cargas de trabajo y recursos compartidos.

d. Sistemas Operativos Embebidos:

- Diseñados para dispositivos específicos, como sistemas de control industriales, dispositivos médicos, y dispositivos IoT (Internet de las cosas).

B. ¿En qué difiere la automatización de pruebas web con Selenium y móviles con Appium?

Encontramos diferencias significativas en cuanto a herramientas, configuración y enfoque, aunque ambos frameworks comparten similitudes en cuanto a la filosofía y algunos conceptos de automatización de pruebas.

- **Selenium:** Diseñado para automatizar pruebas de aplicaciones web que se ejecutan en navegadores. Funciona con todos los navegadores modernos como Chrome, Firefox, Safari, Edge, e Internet Explorer.
- **Appium:** Diseñado para automatizar pruebas de aplicaciones móviles que se ejecutan en dispositivos Android e iOS. Soporta aplicaciones nativas, híbridas y web móviles.

C. ¿Que ha escuchado o trabajado sobre Granjas de dispositivos móviles?

En la organización para la que trabajo actualmente, estuve realizando un curso corto referente a uso de la granja BrowserStack.

BrowserStack es una plataforma de prueba entre navegadores, la cual permite integrar herramientas para la automatización, visualización y prueba en vivo de aplicaciones móviles y sitios web, según las necesidades presentadas.

BrowserStack permite integrar una amplia gama de herramientas para la realización de pruebas funcionales o automatizadas.

Nombre las granjas que ha escuchado y/o trabajado

BrowserStack

● Seguridad
A. Conceptos de seguridad en pruebas web y móvil?

La seguridad en las pruebas web y móviles es fundamental para asegurar que las aplicaciones sean resistentes a amenazas y vulnerabilidades que podrían comprometer la integridad, confidencialidad y disponibilidad de los datos y servicios.
Las pruebas de seguridad tanto para aplicaciones web como móviles son esenciales para identificar y mitigar vulnerabilidades que podrían ser explotadas por atacantes. Implementar una estrategia robusta de pruebas de seguridad ayuda a proteger los datos sensibles y a garantizar la confianza de los usuarios en las aplicaciones.

B. Herramientas utilizadas para pruebas de seguridad

1. Metasploit

- Una plataforma de pruebas de penetración que proporciona herramientas para descubrir, explotar y validar vulnerabilidades.

2. Wireshark

- Un analizador de protocolos de red que se utiliza para capturar y analizar tráfico de red en tiempo real.

3. Nessus

- Un escáner de vulnerabilidades que analiza sistemas y aplicaciones en busca de problemas de seguridad.

● Tecnología Cloud:

La computación en la nube se refiere a la entrega de servicios de computación (incluyendo servidores, almacenamiento, bases de datos, redes, software, análisis e inteligencia) a través de Internet ("la nube") para ofrecer una innovación más rápida, recursos flexibles y economías de escala.
Permite a las empresas escalar los recursos de TI hacia arriba o hacia abajo según sea necesario sin preocuparse por la capacidad del servidor físico.
Con una comprensión clara de los modelos de servicio y despliegue, y utilizando las herramientas adecuadas, las empresas pueden aprovechar al máximo los beneficios de la computación en la nube.

¿Ha trabajado con proyectos Cloud? ¿Herramientas utilizadas?

No.

● Integración Continua:
A. Defina las tareas de Integración y Despliegue continuo en la cual interviene un automatizador de pruebas

La Integración Continua se centra en la construcción y prueba del código de manera frecuente y automática, lo que permite detectar problemas lo antes posible, para ello se debe tener en cuenta las siguientes tareas:

Configuración del entorno de pruebas, Ejecución de pruebas unitarias, Ejecución de pruebas de integración, Ejecución de pruebas de regresión (Automatizadas), Generación de informe de pruebas.

El Despliegue Continuo se centra en la automatización del despliegue del software en entornos de producción, garantizando que cada cambio pase por un pipeline de pruebas automatizadas antes de ser desplegado para ello se debe tener en cuenta las siguientes tareas:

Ejecución de pruebas funcionales (Automatizadas), Ejecución de pruebas de seguridad, Ejecución de pruebas de Performance, Validación en entornos simulados, Monitoreo y validación Post-Despliegue, Rollback Automatizado

B. GIT: ¿Qué es, mencionar los comandos más utilizados, Ha trabajado con GitLab o GitHub?

Git es un sistema de control de versiones distribuido que permite a los desarrolladores rastrear y gestionar los cambios en el código fuente durante el desarrollo del software.

Personalmente he trabajo con ambos repositorios, en donde se comparten los proyectos y se van actualizando a medida que sea necesario.

Comandos Más Utilizados en Git

1. Repositorio

- **git init**: Inicializa un nuevo repositorio Git en el directorio actual.
- **git clone <url_del_repositorio>**: Clona un repositorio remoto a tu máquina local.

2. Estados de Archivos

- **git status**: Muestra el estado de los archivos en el directorio de trabajo y en el área de preparación.
- **git add <archivo>**: Añade un archivo al área de preparación.
- **git add .**: Añade todos los archivos modificados al área de preparación.

3. Commit

- **git commit -m "Mensaje del commit"**: Realiza un commit de los archivos que están en el área de preparación.

4. Ramas (Branches)

- **git branch**: Muestra todas las ramas en el repositorio.
- **git branch <nombre_de_rama>**: Crea una nueva rama.
- **git checkout <nombre_de_rama>**: Cambia a la rama especificada.
- **git merge <nombre_de_rama>**: Fusiona la rama especificada con la rama actual.

5. Actualización y Sincronización

- **git pull**: Actualiza el repositorio local con los cambios del repositorio remoto y fusiona esos cambios con la rama actual.
- **git push**: Envía los commits locales al repositorio remoto.

6. Deshacer Cambios

- **git reset --hard <commit>**: Restablece el índice y el árbol de trabajo para que coincidan con el commit especificado.
- **git revert <commit>**: Crea un nuevo commit que deshace los cambios realizados en el commit especificado.

7. Historial

- **git log**: Muestra el historial de commits.
- **git log --oneline**: Muestra el historial de commits en una sola línea por commit.

C. ¿Explique cómo es el proceso de integración continua con Jenkins?

1. **Desarrollador Realiza Cambios:** Un desarrollador realiza cambios en el código y hace un commit a la rama principal del repositorio.
2. **Webhook Dispara Build en Jenkins:** El repositorio (como GitHub o GitLab) notifica a Jenkins mediante un webhook que se ha realizado un cambio.
3. **Jenkins Clona el Repositorio:** Jenkins clona el repositorio en un entorno limpio.
4. **Ejecución de Pipeline:** Jenkins ejecuta el pipeline configurado, que incluye etapas como compilación, pruebas y despliegue.
5. **Publicación de Resultados:** Jenkins publica los resultados de las pruebas y otros artefactos generados.
6. **Notificaciones:** Jenkins envía notificaciones del estado del build a los desarrolladores y otros interesados.

D. ¿Explique cómo es el proceso de integración continua con Azure DevOps?

Configuración Inicial

1. Creación de un Proyecto en Azure DevOps

- **Descripción:** Crear un nuevo proyecto en Azure DevOps para alojar el código fuente y gestionar el proceso de CI/CD.
- **Pasos:** Iniciar sesión en Azure DevOps, seleccionar la organización y crear un nuevo proyecto. Se puede elegir entre varios modelos de proceso, como Ágil, Scrum o CMMI, según las necesidades del equipo.

2. Configuración del Repositorio

- **Descripción:** Configurar un repositorio Git o TFVC para almacenar el código fuente.
- **Pasos:** Crear un nuevo repositorio en Azure Repos y clonarlo en el entorno de desarrollo local. Se puede conectar el repositorio a un repositorio existente en GitHub si es necesario.

Proceso de Integración Continua

1. Creación de un Pipeline de CI

- **Descripción:** Definir un pipeline de CI que describa cómo se construirá, probará y desplegará el código.
- **Pasos:**

1. **Creación del Pipeline:** Navegar a la sección de Pipelines en Azure DevOps y crear un nuevo pipeline.
2. **Seleccionar Origen de Código:** Especificar el repositorio de código fuente que se utilizará para la construcción.
3. **Seleccionar Plantilla de Pipeline:** Elegir una plantilla predeterminada o crear un pipeline desde cero.
4. **Configurar Pasos de Build:** Definir los pasos de construcción, como la instalación de dependencias, compilación de código y generación de artefactos.
5. **Configurar Pruebas Automatizadas:** Integrar pruebas unitarias y de integración en el pipeline para asegurar la calidad del código.
6. **Publicar Artefactos:** Publicar los artefactos generados durante el proceso de construcción para su posterior despliegue.

2. Configuración de Triggers y Disparadores

- **Descripción:** Configurar triggers para iniciar automáticamente el pipeline en respuesta a eventos como cambios en el repositorio o la programación de ejecuciones regulares.
- **Pasos:** Configurar triggers basados en cambios de código (como push o pull request) o en horarios específicos para ejecuciones programadas.

3. Ejecución y Monitoreo del Pipeline

- **Descripción:** Ejecutar manual o automáticamente el pipeline y monitorear su progreso y resultados.
- **Pasos:** Iniciar una ejecución del pipeline y monitorear su progreso a través de la interfaz de usuario de Azure DevOps. Revisar los resultados de las pruebas y los artefactos generados.

4. Notificaciones y Retroalimentación

- **Descripción:** Configurar notificaciones para informar a los miembros del equipo sobre el estado del pipeline y proporcionar retroalimentación rápida.
- **Pasos:** Configurar notificaciones por correo electrónico, mensajes en equipos de Microsoft o integraciones con sistemas de mensajería para alertar a los desarrolladores sobre el éxito o el fracaso de las ejecuciones del pipeline.

5. Integración con Otros Servicios de Azure DevOps

- **Descripción:** Integrar el pipeline de CI con otros servicios de Azure DevOps, como Azure Boards para gestión de proyectos, Azure Repos para control de versiones y Azure Artifacts para gestión de paquetes.
- **Pasos:** Configurar integraciones entre los diferentes servicios de Azure DevOps para permitir un flujo de trabajo integrado y colaborativo.