Esta hoja de referencia proporciona una guía rápida de los conceptos clave de HTTP y REST relevantes para los ingenieros de automatización de control de calidad. Comprender estos principios es crucial para probar eficazmente los servicios web y las API.I. Conceptos básicos de HTTP

HTTP (Protocolo de Transferencia de Hipertexto) es la base de la comunicación de datos para la World Wide Web.A. Métodos HTTP (Verbos)

Estos métodos indican la acción deseada que se realizará en el recurso identificado.

Método	Descripción	Idempotente	Seguro
GET	Recupera una representación del recurso especificado.	Sí	Sí
POST	Envía una entidad al recurso especificado, a menudo causando un cambio de estado o efectos secundarios en el servidor.	No	No
PUT	Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la solicitud.	Sí	No
DELETE	Elimina el recurso especificado.	Sí	No
PATCH	Aplica modificaciones parciales a un recurso.	No	No
HEAD	Solicita una respuesta idéntica a la de una solicitud GET, pero sin el cuerpo de la respuesta.	Sí	Sí
OPTIONS	Describe las opciones de comunicación para el recurso de destino.	Sí	Sí

- **Idempotente:** Una operación es idempotente si realizarla varias veces produce el mismo resultado que realizarla una sola vez. No tiene efectos secundarios adicionales después de la primera solicitud.
- **Seguro:** Una operación es segura si no altera el estado del servidor. Solo recupera datos sin causar ningún cambio.

Códigos de estado HTTP

Los códigos de estado indican si una solicitud HTTP específica se ha completado correctamente.

Categoría	Rango	Descripción	Códigos comunes
Informativo	1xx	La solicitud fue recibida, continuando el proceso.	100 (Continuar)
Éxito	2xx	La acción fue recibida, entendida y aceptada correctamente.	200 (OK), 201 (Creado), 204 (Sin contenido)
Redirección	3xx	Se necesita realizar una acción adicional para completar la solicitud.	301 (Movido permanentemente), 302 (Encontrado), 304 (No modificado)
Error del cliente	4xx	La solicitud contiene una sintaxis incorrecta o no se puede cumplir.	400 (Solicitud incorrecta), 401 (No autorizado), 403 (Prohibido), 404 (No encontrado), 405 (Método no permitido), 409 (Conflicto), 422 (Entidad no procesable)
Error del servidor	5xx	El servidor no pudo cumplir una solicitud aparentemente válida.	500 (Error interno del servidor), 502 (Puerta de enlace incorrecta), 503 (Servicio no disponible), 504 (Tiempo de espera de la puerta de enlace)

Encabezados HTTP

Los encabezados proporcionan información adicional sobre la solicitud o respuesta, como el tipo de contenido, las credenciales de autenticación o las instrucciones de almacenamiento en caché.

- **Encabezados de solicitud comunes:**
  - Content-Type: Tipo de medio del recurso (por ejemplo, application/json, application/xml).
  - Accept: Tipos de medios que son aceptables para la respuesta.
  - Authorization: Credenciales para autenticar un agente de usuario con un servidor.
  - User-Agent: Información sobre el software cliente.
  - Cache-Control: Mecanismos de almacenamiento en caché tanto en solicitudes como en respuestas.
- **Encabezados de respuesta comunes:**
  - Content-Type: Tipo de medio del cuerpo de la respuesta.
  - Content-Length: Tamaño del cuerpo de la respuesta en octetos.
  - Location: Se utiliza para la redirección o para apuntar a un recurso recién creado.
  - Set-Cookie: Envía cookies del servidor al agente de usuario.

II. Principios REST

REST (Representational State Transfer) es un estilo arquitectónico para construir servicios web. Es un conjunto de pautas sobre cómo deben comunicarse los clientes y los servidores.A. Principios clave de REST

1. **Cliente-Servidor:** Separación de preocupaciones. Los clientes no conocen la implementación del servidor y los servidores no conocen la interfaz de usuario del cliente.
2. **Sin estado:** Cada solicitud del cliente al servidor debe contener toda la información necesaria para comprender la solicitud. El servidor no debe almacenar ningún contexto del cliente entre solicitudes.
3. **Almacenable en caché:** Las respuestas deben definirse explícita o implícitamente como almacenables en caché o no almacenables en caché para evitar que los clientes reutilicen datos obsoletos o inapropiados.
4. **Interfaz uniforme:** Simplifica y desacopla la arquitectura, lo que permite la evolución independiente de clientes y servidores. Esto incluye:
   - **Identificación de recursos en las solicitudes:** Los recursos individuales se

- identifican en las solicitudes.
- **Manipulación de recursos a través de representaciones:** El cliente recibe una representación de un recurso que contiene suficiente información para modificar o eliminar el recurso.
- **Mensajes autodocumentados:** Cada mensaje incluye suficiente información para describir cómo procesar el mensaje.
- **Hipermedia como motor del estado de la aplicación (HATEOAS):** El cliente interactúa con la aplicación completamente a través de hipermedia proporcionada dinámicamente por las aplicaciones del lado del servidor. (A menudo, el principio más difícil de implementar y probar por completo).

5. **Sistema en capas:** Un cliente no puede saber normalmente si está conectado directamente al servidor final o a un intermediario en el camino.
6. **Código bajo demanda (Opcional):** Los servidores pueden extender o personalizar temporalmente la funcionalidad de un cliente transfiriendo código ejecutable.

B. Recursos y URI

- **Recurso:** Cualquier información que se pueda nombrar, como un documento, una imagen o una colección de otros recursos.
- **URI (Uniform Resource Identifier):** Una cadena de caracteres utilizada para identificar un nombre o un recurso en Internet. En REST, los URI identifican recursos.
- **Buenos ejemplos de URI:**
  - /users (colección de usuarios)
  - /users/123 (usuario específico con ID 123)
  - /products/electronics/laptops (recursos anidados)
- **Evitar URI que:**
  - Contengan verbos (por ejemplo, /getAllUsers, /createUser)
  - Sean demasiado complejos o profundamente anidados
  - Usen extensiones de archivo (por ejemplo, /users.json)

III. Consideraciones de prueba para ingenieros de automatización de control de calidadA. Escenarios de prueba comunes

- **Pruebas funcionales:**
  - Verificar que todos los métodos HTTP (GET, POST, PUT, DELETE, PATCH) funcionen como se espera.
  - Validar las estructuras del cuerpo de la solicitud y la respuesta (JSON, XML).
  - Probar la creación, recuperación, actualización y eliminación de datos.
  - Verificar la lógica de negocio y la integridad de los datos.
- **Pruebas de rendimiento:**
  - Medir los tiempos de respuesta bajo varias cargas.
  - Probar el rendimiento y la concurrencia.

    - Identificar cuellos de botella.
- **Pruebas de seguridad:**
    - Probar la autenticación (por ejemplo, claves API, tokens OAuth) y la autorización (acceso basado en roles).
    - Validar el saneamiento de la entrada para evitar ataques de inyección.
    - Asegurar la comunicación segura (HTTPS).
- **Pruebas de manejo de errores:**
    - Validar que se devuelvan los códigos de estado HTTP apropiados para varias condiciones de error (por ejemplo, 400 para solicitud incorrecta, 404 para no encontrado, 401 para no autorizado).
    - Verificar que el contenido del mensaje de error sea informativo y coherente.
- **Pruebas negativas:**
    - Enviar tipos de datos no válidos, campos obligatorios faltantes o valores fuera de rango.
    - Probar intentos de acceso no autorizados.
    - Simular problemas de red o fallas del servidor.

B. Herramientas y bibliotecas

- **Herramientas de prueba de API:**
    - **Postman/Insomnia:** Para pruebas de API manuales y automatizadas, creación de colecciones.
    - **cURL:** Herramienta de línea de comandos para realizar solicitudes HTTP.