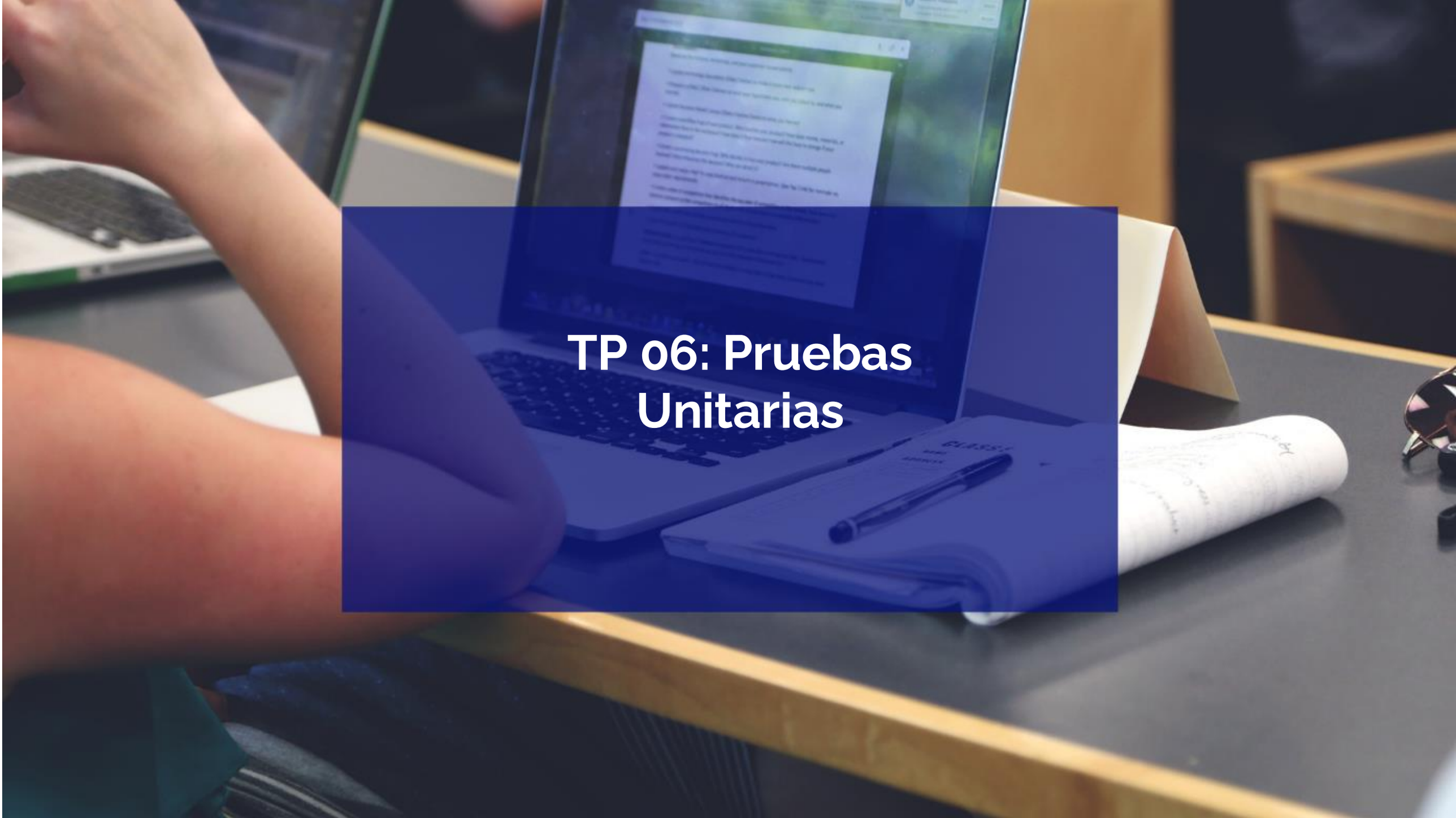


TP 06: Pruebas Unitarias

Pruebas Unitarias

Ing. Ariel Schwindt

https://www.linkedin.com/in/arielschwindt/

MS Certified DevOps Engineer Expert
MS Certified Azure Developer Associate
MS Certified Azure AI Engineer Associate

Objetivo de la Sesión

- Adquirir conocimientos sobre pruebas unitarias y sus beneficios en el ciclo de desarrollo.
- Desarrollar y ejecutar pruebas unitarias utilizando frameworks disponibles para backend (.NET Core) y frontend (Angular).
- Comprender la importancia de la automatización para reducir errores y asegurar la calidad del código a medida que evoluciona.

Resultados Esperados

Al final de la sesión, los participantes deben ser capaces de:

- Diseñar pruebas unitarias efectivas para validar funciones y métodos en un sistema.
- Aplicar herramientas de prueba automatizadas, como **xUnit** y **Jasmine**, para backend y frontend.
- Desarrollar pruebas unitarias que aseguren que el código funciona según lo esperado y detecten errores introducidos por cambios.
- Implementar mocks para aislar dependencias externas en las pruebas.
- Analizar resultados de pruebas para mejorar la calidad del código y evitar regresiones.

¿Qué son las pruebas de software?

Proceso automatizado o manual que se utiliza para evaluar la calidad de una pieza de código

Beneficios

- **Reducción de errores**: Al identificar problemas de código antes de que lleguen a producción.
- **Prevención de regresiones**: Aseguran que los cambios en el código no introduzcan errores en funcionalidades ya existentes.
- **Ahorro de tiempo**: Las pruebas automatizadas pueden ejecutarse repetidamente sin intervención humana, lo que facilita el desarrollo continuo y las actualizaciones rápidas.
- **Mejor calidad del código**: Aumentan la confianza en el sistema al garantizar que las distintas partes del software funcionen correctamente de forma independiente.

¿Qué validan las pruebas?

Estado esperado (validación de resultados):

- Verifican que el código produzca el resultado esperado para un conjunto de entradas específicas.
- Ejemplo: Si la función sumar(2,3) retorna 5, entonces la prueba es exitosa.

Comportamiento esperado (secuencia de acciones correctas):

- Aseguran que el código siga los pasos correctos para alcanzar el resultado.
- Ejemplo: Si al procesar una compra, primero se verifica la disponibilidad del producto, luego se genera la factura y finalmente se actualiza el inventario.

Áreas típicas a probar:

- **Funcionalidad**: Asegurar que el sistema cumple con los requisitos funcionales.
- **Rendimiento**: Verificar que el código responde eficientemente bajo carga.
- **Seguridad**: Comprobar que el sistema no sea vulnerable a ataques o uso indebido.
- **Compatibilidad**: Validar que el sistema funcione correctamente en diferentes plataformas, navegadores o dispositivos.

Tipos de Pruebas de Software

Etapas y Características Clave de las Diferentes Pruebas

Tipo de Prueba	Etapa en que se realiza	Características principales
Pruebas Unitarias	Desarrollo	Verifican unidades individuales de código (métodos, funciones). Aíslan dependencias con mocks.
Pruebas de Integración	Desarrollo/Integración	Validan que diferentes módulos o componentes del sistema funcionen bien en conjunto.
Pruebas Funcionales	Post-desarrollo	Verifican que el sistema cumpla con los requisitos funcionales. Evalúan características y flujos.
Pruebas de Sistema	Post-integración	Evalúan el sistema completo, validando que cumpla con los requisitos generales.
Pruebas de Regresión	Después de cambios	Aseguran que nuevas funcionalidades no rompan las existentes.
Pruebas de Rendimiento	Antes de la entrega	Miden la capacidad del sistema para manejar cargas de trabajo en diferentes condiciones.
Pruebas de Aceptación	Antes de la entrega	Validan que el sistema cumple con los requisitos acordados con el cliente. Última etapa de pruebas.
Pruebas de Seguridad	Después de desarrollo	Evalúan la vulnerabilidad del sistema ante ataques o amenazas de seguridad.

Pruebas Unitarias (Unit Tests)

Una prueba unitaria es un fragmento de código que valida pequeñas unidades individuales del programa, como funciones o métodos específicos.

Estas pruebas se enfocan en evaluar un componente aislado, verificando que cumpla con su funcionalidad esperada sin depender de otras partes del sistema.

Uso de Mocks:

- Para evitar que las pruebas dependan de componentes externos (como bases de datos o servicios web), se utilizan **mocks**. Los mocks son simulaciones de estos componentes, lo que permite probar una unidad en aislamiento.
- Esto asegura que las pruebas sean rápidas, confiables y no dependan del estado del sistema externo.

Cobertura de Pruebas (Code Coverage)

- La cobertura de pruebas se refiere al porcentaje del código que está cubierto por pruebas unitarias. Una mayor cobertura indica un mayor nivel de confianza en el código.
- Las pruebas unitarias aseguran que cada parte del código haya sido evaluada bajo distintos escenarios, minimizando el riesgo de errores ocultos.

¿Por qué son útiles las pruebas unitarias de software?

Mejora de calidad, detección temprana de errores y agilidad en el desarrollo.

Beneficios

1. **Verifican la lógica del código:** Aseguran que el código se comporte según lo esperado, ayudando a detectar errores lógicos y problemas en fases tempranas del desarrollo.
2. **Automatizan la detección de regresiones:** Detectan rápidamente errores que surgen cuando nuevas características rompen funcionalidades existentes, facilitando iteraciones rápidas y seguras.
3. **Incrementan la confianza en el desarrollo continuo:** Una cobertura de pruebas bien estructurada permite agregar nuevas funciones sin temor a introducir errores, lo que acelera el despliegue de nuevas versiones con mayor seguridad.
4. **Reducen los costos de mantenimiento:** Prevenir errores tempranos reduce significativamente los costos de corrección en producción, donde los errores suelen ser más costosos de resolver.
5. **Facilitan la colaboración en equipos de desarrollo:** Las pruebas permiten que los desarrolladores trabajen en paralelo sin temor a generar conflictos, lo que mejora la agilidad y colaboración en entornos de desarrollo continuo.
6. **Documentan el comportamiento esperado del código:** Las pruebas actúan como una guía viviente del comportamiento del sistema, ayudando a los nuevos desarrolladores a comprender cómo debe funcionar el código.

Las pruebas unitarias de software mejoran la calidad, reducen costos, incrementan la velocidad de desarrollo y facilitan la colaboración, ayudando a prevenir errores antes de que lleguen a producción.

Frameworks de Pruebas Unitarias

Un framework de pruebas unitarias es una herramienta diseñada para ayudar a los desarrolladores a crear, organizar y ejecutar pruebas sobre el código de manera estructurada.

- Facilitan la **automatización de las pruebas**, la verificación de que el código se comporta según lo esperado y la detección de errores en fases tempranas del desarrollo.
- Estos frameworks suelen incluir funciones para definir pruebas, ejecutar múltiples escenarios, y generar informes sobre los resultados de las pruebas.
- Existen numerosos frameworks orientados a distintos lenguajes de programación. Algunos de los más comunes incluyen: JUnit (Java), NUnit (C# / .NET), Mocha (JavaScript / Node.js), **Jasmine** (JavaScript / Angular) PyTest (Python), **xUnit** (Varios lenguajes, incluyendo .NET)

Frameworks que utilizaremos en nuestro proyecto:

- **Backend (API) - xUnit con Moq:**
  - En el backend basado en **.NET Core**, utilizaremos **xUnit** para escribir pruebas unitarias.
  - **Moq** será usado para simular dependencias externas, como bases de datos, con el fin de probar el código en aislamiento.

- **Frontend (Angular) - Jasmine y Karma:**
  - En el frontend basado en **Angular**, emplearemos **Jasmine** para escribir las pruebas unitarias, y **Karma** para ejecutarlas en diferentes navegadores.
  - **Jasmine** ofrece su propio mecanismo de simulación llamado **spies** para crear mocks, que permiten verificar las interacciones y el comportamiento de las dependencias de los componentes de Angular.
  - Este conjunto nos permitirá validar el comportamiento de los componentes de Angular en distintos entornos de manera eficiente.

Convenciones de nombre y patrón AAA

Las pruebas unitarias no solo deben asegurar que el código funcione correctamente, sino que también deben ser claras y fáciles de entender.

- Para lograr esto, se utilizan **convenciones de nomenclatura** que describen claramente lo que se está probando y lo que se espera como resultado.
- Además, las pruebas unitarias siguen el **patrón AAA** (Arrange, Act, Assert), una estructura simple pero efectiva para organizar y ejecutar las pruebas de manera coherente y predecible.

Convención de nombre en pruebas:

El nombre de las pruebas unitarias sigue una convención clara para describir qué se está probando y qué se espera.

- **Formato:** Metodo_Escenario_ResultadoEsperado
- **Ejemplo:** CanBeCancelledBy_UserIsAdmin_ReturnsTrue
- Esto mejora la legibilidad y facilita entender rápidamente qué escenario está cubriendo la prueba.

Patrón AAA (Arrange, Act, Assert):

- **Arrange:** Configuración inicial de la prueba, donde se preparan las dependencias, se crea el entorno necesario y se configuran los mocks.
- **Act:** Ejecución de la acción o funcionalidad que se va a probar.
- **Assert:** Verificación del resultado esperado. Se comparan los resultados obtenidos con los esperados, para confirmar que el código funciona correctamente.

¿Qué parte del código debe probarse?

Enfoque en unidades clave y escenarios críticos:

Unidad de Código:

Las pruebas unitarias deben enfocarse en **funciones, métodos o clases individuales**. Evalúan si una pequeña parte del sistema se comporta de manera correcta en distintos escenarios.

Caminos de Ejecución:

Es fundamental probar **todos los caminos de ejecución posibles** dentro de una unidad de código:

- **Casos de éxito:** Cuando todo funciona correctamente.
- **Casos de error:** Cuando el código debe manejar situaciones excepcionales o entradas inválidas.

Entradas y Salidas:

Verificar que las unidades de código manejen correctamente diferentes tipos de entradas, incluyendo valores límite o excepcionales, y que produzcan las salidas esperadas.

¿Qué parte del código NO debe probarse?

Limitaciones y áreas fuera del enfoque de las pruebas unitarias

Áreas fuera del alcance de las pruebas unitarias:

- **Interfaces de usuario complejas**:
  Las pruebas unitarias no son adecuadas para validar comportamientos o interacciones detalladas de interfaces gráficas. Para esto se necesitan **pruebas de integración** o **pruebas de interfaz de usuario (UI)**.

- **Sistemas externos**:
  No se deben probar conexiones directas con bases de datos, servicios externos o APIs de terceros. Para estas dependencias se utilizan **mocks** o simulaciones.

- **Frameworks y dependencias**:
  No es necesario probar el **funcionamiento interno** de los frameworks o librerías de terceros que ya están probados y testeados por sus desarrolladores. Las pruebas deben centrarse en el código propio.

- **Flujos de trabajo completos**:
  No se debe usar pruebas unitarias para verificar flujos de negocio que atraviesan múltiples componentes o sistemas. Esto corresponde a **pruebas de integración** o **pruebas funcionales**.

- **Rendimiento**:
  Las pruebas unitarias no son adecuadas para medir el rendimiento del sistema. El análisis de rendimiento requiere **pruebas de carga** o **pruebas de estrés**

Introducción a Mocking

Los mocks son versiones simuladas de dependencias externas, como bases de datos, servicios web o APIs. Permiten que las pruebas unitarias se enfoquen exclusivamente en la lógica interna del código, sin depender del estado de sistemas externos.

Frameworks de Mocks

Moq:

- **Simulación de dependencias**: Moq permite crear objetos simulados (mocks) que reemplazan las dependencias reales en una prueba unitaria.

- **Verificación de interacciones**: Con Moq, se pueden verificar si ciertos métodos fueron llamados y con qué parámetros, asegurando que la lógica de negocio interactúa correctamente con sus dependencias.

- **Aplicación de Moq**:
  - Moq se utiliza principalmente para probar la lógica de negocio sin necesidad de interactuar con componentes externos como bases de datos, servicios REST o APIs.
  - Esto aísla las pruebas de problemas externos y permite a los desarrolladores centrarse en la funcionalidad interna de la aplicación.

Otros frameworks de mocks:

- **NSubstitute**: Un framework de mocking para .NET que ofrece una sintaxis clara y fluida para simular dependencias.
- **FakeItEasy**: Un framework de mocks para .NET, fácil de usar y rápido para configurar simulaciones.
- **Rhino Mocks**: Un framework veterano de mocking para .NET, aunque ha sido reemplazado en popularidad por Moq y NSubstitute.
- **Jasmine (JavaScript / Angular)**: Jasmine incluye **spies** como su mecanismo integrado de mocking, que permite simular métodos y verificar interacciones en las pruebas unitarias.
- **Jest**: Popular en JavaScript y TypeScript, con soporte integrado para mocks, junto con pruebas unitarias.

slido

Please download and install the Slido app on all computers you use

Join at slido.com
#1989566

ⓘ Start presenting to display the joining instructions on this slide.

slido

Please download and install the Slido app on all computers you use

¿Cuál es el principal objetivo de las pruebas unitarias?

ⓘ Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use

¿Qué herramienta se usa comúnmente para pruebas unitarias en Angular?

ⓘ Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use

¿Qué patrón se sigue generalmente en una prueba unitaria?

ⓘ Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use

¿Qué NO debe probarse en una prueba unitaria?

ⓘ Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use

¿Cuáles de las siguientes son características de una prueba unitaria?
(Selecciona todas las que correspondan)

ⓘ Start presenting to display the poll results on this slide.

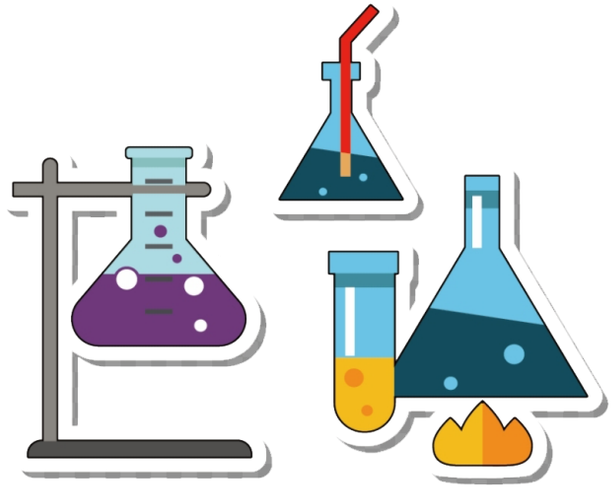
slido

Please download and install the Slido app on all computers you use

¿Cuáles son ventajas de las pruebas unitarias? (Selecciona todas las que correspondan)

ⓘ Start presenting to display the poll results on this slide.

Presentación del TP



Consigna y Desarrollo del Trabajo Práctico

Pruebas Unitarias en Backend y Frontend con Simulación de Dependencias.

Backend: Pruebas Unitarias con xUnit y Moq

- **Objetivo**: Crear un proyecto de pruebas unitarias para la API utilizando **xUnit** y **Moq**.
- **Desarrollo**:
  - Implementar pruebas sobre los métodos CRUD del controlador, simulando la base de datos con **Entity Framework Core InMemory**.
  - Probar operaciones como obtener empleados, agregar, actualizar y eliminar.
  - Validar que no se puedan agregar empleados duplicados y que los nombres y apellidos tengan el formato correcto.
- **Resultado esperado**: Las operaciones CRUD funcionen correctamente en aislamiento y las validaciones indicadas estén implementadas.

Frontend: Pruebas Unitarias con Jasmine y HttpClientTestingModule

- **Objetivo**: Crear pruebas unitarias para el frontend en **Angular** utilizando **Jasmine**.
- **Desarrollo**:
  - Verificar la correcta funcionalidad de los componentes y las validaciones de nombres sin caracteres especiales.
  - Mockear servicios HTTP usando **HttpClientTestingModule**.
  - Probar que los componentes manejen correctamente los datos y visualicen errores en un toast antes de enviar datos a la API.
- **Resultado esperado**: Las validaciones del frontend funcionen correctamente antes de interactuar con la API.

Espacio para preguntas, dudas y consultas