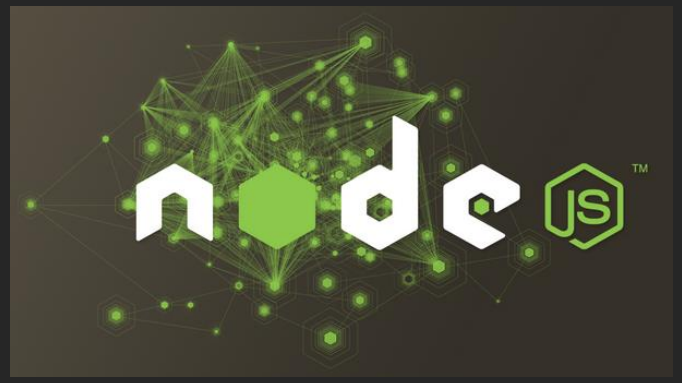
Programa de Entrenamiento Intensivo

Desarrollador Web Full Stack

Marzo de 2019



Microsoft SQL Server

Visual Studio Code

Jasmine

HTML

CSS

Día 09 – Aplicaciones Testeables

baufest

Sobre el instructor

Christian Smirnoff

Ingeniero en Informática – UADE

Microsoft Practice Manager

7 años en Baufest

Principales proyectos en los que participé…

- Comisiones (Falabella)
- Ajustes al Origen / Facturación por Módulos (La Nación)
- Evaluación de Productos de Desarrollo (Bunge)
- Transportation Management System (SC Johnson)
- Sincronización y Tareas de Gálpón (Don Mario)
- Control y Gestión de Obras (AySA)
- Club La Nación / TV / Órdenes de Publicación Web (La Nación)
- Rediseño portal Quiero! (Banco Galicia)
- BitCow (OpenBit)

Sobre el ayudante

Patricio Filice

Ingeniería en Sistemas de Información - UTN

.Net Developer

1 año en Baufest

Principales proyectos en los que participé...

- Rediseño portal Quiero! (Banco Galicia)

Objetivos del Módulo

- Entender el concepto de aplicaciones testeables (¿Qué?)
- Proveer a los desarrolladores las herramientas necesarias para diseñar y desarrollar aplicaciones testeables (¿Cómo?)
- Visualizar las ventajas de utilizar estas prácticas (¿Por qué?)

Agenda

- Testing del desarrollador
  - Introducción
  - Tests unitarios
  - Tests de integración
  - Ventajas
- ¿Cómo diseñamos aplicaciones testeables?
  - Tips de arquitectura
  - Ejemplos de arquitectura

Testing del desarrollador

Introducción

- ¿Qué es un test?
  - Es una prueba que compara el resultado esperado y el obtenido al ejecutar cierta funcionalidad de un sistema

- ¿Qué es un test de desarrollador?
  - Código escrito por el desarrollador para testear que lo desarrollado genera los resultados esperados
  - Es complementario a las pruebas funcionales, generalmente realizadas por un especialista en testing
  - Generalmente se ejecutan de forma automática mediante una herramienta

Tests unitarios

- ¿Qué es un test unitario?
  - Es un test que se realiza abstrayendo el objeto a testear de sus dependencias con otros componentes

Tests unitarios

- Un buen test unitario:
  - Documenta el diseño de la aplicación
  - Tiene control total de todos los componentes en ejecución
  - Puede ejecutarse en cualquier orden si es parte de muchos otros tests
  - Retorna consistentemente el mismo resultado
  - Prueba un único concepto lógico en el sistema
  - Tiene un nombre claro y consistente
  - Es legible
  - Es mantenible

Tests de integración

- Los tests de integración:
  - Testean la “integración” entre componentes
  - Son complementarios a los tests unitarios
  - Usan dependencias tales como una base de datos
  - Pueden ser utilizados para probar stored procedures y llamadas a aplicaciones externas
  - Son menos performantes que los tests unitarios y a veces se ejecutan menos frecuentemente
  - Se enfocan en métodos con dependencias, no pruebas de la aplicación de punta a punta

Tests de integración

- Un buen test de integración:
  - **Utiliza dependencias de forma controlada**
  - Documenta el diseño de la aplicación
  - Puede ejecutarse en cualquier orden si es parte de muchos otros tests
  - Retorna consistentemente el mismo resultado
  - Prueba un único concepto lógico en el sistema
  - Tiene un nombre claro y consistente
  - Es legible
  - Es mantenible

Ventajas

- Utilizar estas prácticas:
  - Minimiza el número de errores en el producto final
  - Hace el código más mantenible
  - Permite la detección temprana de errores en el ambiente de desarrollo
  - Reduce el tiempo de desarrollo y mantenimiento durante el ciclo de vida de un proyecto
  - Permite generar métricas de cobertura de código
  - **Mejora la calidad del producto final**

¿Cómo diseñamos aplicaciones testeables?

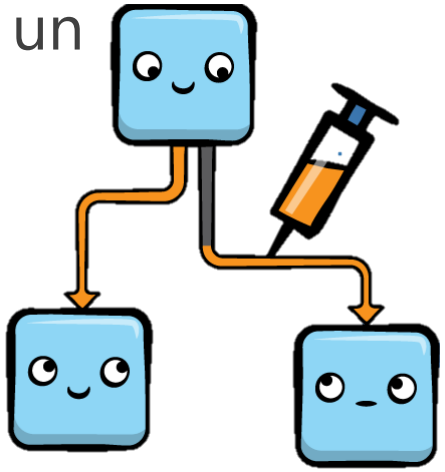
Tip 1: Programación orientada a abstracciones

- Mantenibilidad
  - Permite cambiar la implementación interna de las clases concretas sin modificar el código de la aplicación
- Extensibilidad
  - Permite la creación de diferentes clases concretas que implementen la abstracción sin modificar el código de la aplicación
- Testeabilidad
  - Es fundamental para testear componentes unitariamente
  - El código de la aplicación no depende de clases concretas

Tip 2: Inyección de dependencias por constructor

- Una inyección es el pasaje de una dependencia a un objeto dependiente, que no necesita crear ni buscar la dependencia

- Requiere que la dependencia se provea a través de un parámetro en un constructor
  - public constructor(Dependency dependency)
  - ¡Ojo con tener demasiadas dependencias!
    - public constructor(Class1 c1, Class2 c2, Class3 c3, Class4 c4, Class5 c5, ……)



- El objeto inyectado es parte del estado interno del objeto dependiente

- El objeto dependiente ya no necesita ningún conocimiento sobre la implementación concreta que va a utilizar

Tip 3: Favorecer la composición por sobre la herencia

- Permite que las subclases implementen nueva funcionalidad sin afectar otras subclases
- Permite cambios de comportamiento en tiempo de ejecución
- Ideal para casos donde una subclase implementa solamente una parte del comportamiento expuesto por la superclase
- *Recomendación: Elegir la composición por sobre la herencia ya que es más maleable y sencilla para la modificación de código, pero tampoco componer en todos los casos*

Tip 4: Generar tests unitarios

- Cómo escribir tests unitarios:
  - Setup de precondiciones
  - Ejecutar el código a testear
  - Realizar asserts sobre los resultados esperados

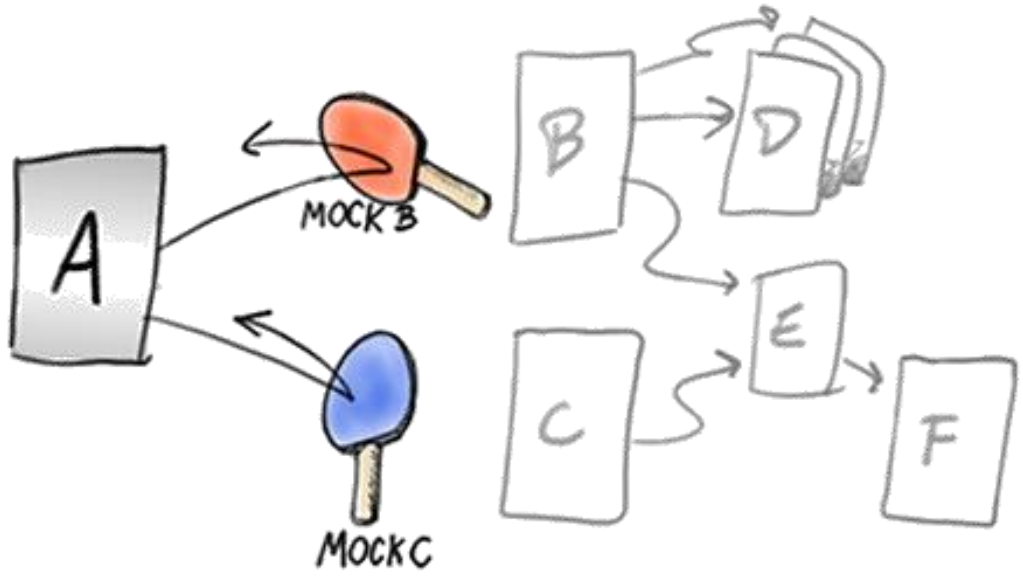
Jasmine

Tip 5: Mocking de dependencias

- Un test unitario debe probar el código sin probar las dependencias
- Ver Tips 1 y 2
  - Programación orientada a abstracciones
  - Inyección de dependencias por constructor

Tip 5: Mocking de dependencias

- Los objetos Mock son objetos simulados que imitan el comportamiento de objetos reales de forma controlada
- Permiten realizar verificación del comportamiento del objeto



Tip 5: Mocking de dependencias

- Cómo escribir tests unitarios usando Spies:
  - Setup de precondiciones incluyendo el setup de los objetos spy
  - Inyectar spies de dependencias
  - Ejecutar el código a ser testeado
  - Realizar asserts sobre los resultados esperados
  - Verificar que el spy fue llamado la cantidad de veces y con los parámetros esperados

Tip 6: Escribiendo código testeable

- No mezclar el grafo de instanciación de objetos con la lógica de la aplicación
- Pedir los objetos, no ir a buscarlos
- No escribir lógica en el constructor
- Tener cuidado con estado global y singletons
- Tener cuidado con métodos estáticos
- Elegir el polimorfismo por sobre los condicionales
- No mezclar objetos de servicio con objetos de valor
- No mezclar responsabilidades

¿Preguntas?

¡Muchas Gracias!