INFORME DE CONOCIMIENTOS PREVIOS A DISEÑO Y PRUEBAS II DE TESTING EN UN WIS

Imagen que contiene Logotipo

El contenido generado por IA puede ser incorrecto.

Grupo: C1.061

Repositorio: <https://github.com/Davidvt04/Acme-ANS-D01>

Miembros:

David Valencia Toscano *davvaltos@alum.us.es*

Eloy Sancho Cebrero *elosanceb@alum.us.es*

Ivo Raimondi *ivorai@alum.us.es*

María del Pino Pérez Domínguez *marperdom@alum.us.es*

Iván Fernández Limárquez *ivaferlim@alum.us.es*

Fecha: 18-02-202

Índice

[Executive Summary 2](#_Toc2068499094)

[Revision Table 3](#_Toc49799923)

[Introduction 4](#_Toc216610737)

[Contents 5](#_Toc890286588)

[Conclusions 7](#_Toc146029718)

[Bibliography 8](#_Toc94131075)

Executive Summary

En este informe se presentan los conocimientos adquiridos previamente sobre testing en un WIS, con un enfoque especial en las pruebas unitarias y el Test Driven Development. Se describen los distintos tipos de pruebas utilizadas en el desarrollo de software, incluyendo pruebas unitarias, de integración, end-to-end, de aceptación y exploratorias. Sin embargo, las pruebas unitarias han sido el eje central de nuestra formación, abordando su estructura, características y el uso de los distintos tipos de dobles que hay (y sus funciones) para simular dependencias en pruebas solitarias.

Asimismo, se detallan las buenas prácticas recomendadas, como la parametrización de pruebas, el uso de Fluent Assertions y la escritura de pruebas enfocadas en un solo comportamiento. También se resalta la importancia de estructurar las pruebas de manera clara, priorizando el principio DAMP sobre DRY sin dejar de lado la reutilización del código cuando sea pertinente.

Por último, hemos aprendido a realizar pruebas en la capa de presentación del WIS, lo que siempre hemos llamado “frontend”. Para ello, utilizamos la librería de Node.js Jest, que nos permite probar componentes de React en un entorno de prueba específico sin necesidad de ejecutarlos en el navegador.

Este conocimiento previo nos proporciona una base sólida para abordar la actual asignatura, permitiéndonos profundizar en el testing de manera más avanzada y aplicada.

Revision Table

|  |  |  |
| --- | --- | --- |
| **Revision Number** | **Date** | **Description** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Introduction

En este documento procedemos a explicar lo que los integrantes de este grupo sabían sobre la realización de pruebas en un Web Information System (WIS) antes de cursar la asignatura Diseño y Pruebas II. Nos basamos en los conocimientos transmitidos por los docentes en la asignatura Diseño y Pruebas I. A pesar de que hemos estudiado previamente el concepto de WIS, la parte relativa a las pruebas es algo que sólo hemos visto en la asignatura mencionada.

Contents

Entendemos que el testing en un WIS es la realización de las pruebas necesarias para confirmar que se han implementado los requisitos correctamente y, por ende, se ha construido lo que el cliente quería.

Conocíamos la existencia y la definición de diferentes tipos de pruebas de distintas granulidades, concretamente, las pruebas unitarias, las pruebas de integración, las pruebas end-to-end, las pruebas de aceptación y las pruebas exploratorias. Sin embargo, a pesar de que sí hemos llegado a conocer con algo más de detalle las pruebas de aceptación, las pruebas E2E y las pruebas de integración, las pruebas unitarias han sido las que más importancia han tenido en Diseño y Pruebas I. También cabe recalcar la importancia que se le ha dado al Test Driven Development, concepto que siempre hemos entendido como “implementar funcionalidades a la par que se van realizando pruebas sobre el correcto funcionamiento de las mismas”.

Hemos estudiado las pruebas unitarias para comprobar el correcto funcionamiento de componentes de la aplicación como lo son los controladores, los servicios, los repositorios y las entidades del modelo de dominio. Por ello, hemos conocido la definición de Subject Under Test (SUT) como la parte concreta del código a probar en cada caso y, además, entendemos que cada prueba unitaria debe ser rápida, independiente, repetible y debe evaluar por sí misma si los resultados son correctos o no. Pensamos que cada SUT debe tener suficientes pruebas como para probar casos negativos y positivos, así como los distintos casos de un condicional.

Hemos distinguido entre pruebas sociables y solitarias, siendo las primeras las pruebas en las que la unidad que se está probando interactúa con otras unidades y las segundas las pruebas en las que dicha interacción se produce con dobles, es decir, elementos que simulan el comportamiento de las unidades con las que interactúa el SUT. Conocemos varios tipos de dobles y sus funciones, concretamente los stubs (devuelven respuestas predefinidas), los mocks (verifican interacciones), los fakes (tienen lógica simplificada), los dummies (son objetos vacíos sin lógica) y los spies (registran llamadas para análisis). Desconocemos la existencia de cualquier otro tipo no mencionado anteriormente. Siempre se nos ha indicado que tanto las pruebas sociables como las solitarias se utilizan, pero las pruebas solitarias se realizan para probar el funcionamiento interno de un componente concreto y las pruebas sociables se realizan para comprobar que la interacción entre varios componentes es la esperada.

En cuanto a las buenas prácticas de testing, nos han enseñado que lo más importante es:

* Parametrizar las pruebas unitarias para reutilizar el código y probar distintos escenarios con menos esfuerzo, pero sin comprometer la claridad de los casos de prueba.
* Usar fluent Assertions para escribir afirmaciones más intuitivas y legibles, lo que entendemos que facilita la comprensión y el mantenimiento del código de prueba.
* Mantener las pruebas unitarias enfocadas en un único comportamiento para que sean más predecibles, fáciles de depurar y rápidas de ejecutar.
* Escribir pruebas en las que los efectos sigan inmediatamente a las causas.
* Priorizar el principio “Descriptive and Meaningful Phrases” (DAMP) sobre el principio “Don’t Repeat Yourself” (DRY), es decir, tratar de que el código relativo a las pruebas sea lo más claro posible, sin tratar de reducir mucho el código duplicado. Sin embargo, nos han advertido de que debemos seguir teniendo en cuenta el principio DRY.

Por último, cabe mencionar que también hemos aprendido cómo hacer pruebas con respecto a la capa de presentación del WIS, lo que nosotros hemos llamado siempre “frontend”. Nos han enseñado a realizar pruebas de componentes React gracias a una librería de node.js llamada “jest”. Esta librería creaba un entorno de prueba para nuestros componentes, es decir, no se ejecutaban en el navegador, sino en un contexto de prueba específico donde ya están importadas react-test y otras librerías de testing.

Conclusions

En conclusión, hemos recibido una base sólida sobre testing en un WIS, centrándonos especialmente en las pruebas unitarias y en el Test Driven Development. Hemos aprendido a distinguir distintos tipos de pruebas y a aplicar buenas prácticas para mejorar su claridad y eficacia. También hemos comprendido la importancia de los dobles en las pruebas solitarias y la diferencia con las pruebas sociables. Todo este conocimiento nos ha permitido desarrollar pruebas que pensamos que han sido estructuradas y eficientes en el proyecto de la asignatura Diseño y Pruebas I. Ahora partimos de esta base para profundizar en nuevos conceptos en la actual asignatura.

Bibliography

Intentionally blank.