Título del Trabajo. Modificar la portada en Cascaras/cover.tex

Trabajo de Fin de Máster
Curso 2021–2022

Autor
Nombre Apellido1 Apellido2

Director
Director 1
Director 2

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Título del Trabajo. Modificar la portada en Cascaras/cover.tex

Trabajo de Fin de Máster en Ingeniería Informática
Departamento de XXXXXXXXXXXXX

Autor
Nombre Apellido1 Apellido2

Director
Director 1
Director 2

Convocatoria: *Febrero/Junio/Septiembre 2022*
Calificación: *Nota*

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

11 de octubre de 2022

Autorización de difusión

El abajo firmante, matriculado en el Máster en Ingeniería en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "TITULO DEL TRABAJO", realizado durante el curso académico CURSO bajo la dirección de DIRECTORES en el Departamento de XXXXXXXXXXXXXXXXXXXXXXX, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Nombre Del Alumno

11 de octubre de 2022

Dedicatoria

Texto de la dedicatoria...

Agradecimientos

Texto de los agradecimientos

Resumen

Resumen en español del trabajo

Palabras clave

Máximo 10 palabras clave separadas por comas

Abstract

Abstract in English.

Keywords

10 keywords max., separated by commas.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"Frase célebre dicha por alguien inteligente"
— Autor

1.1. Motivación

La educación escolar tiene como objetivo promover el desarrollo de ciertas habilidades, y el aprendizaje de ciertos contenidos necesarios para que los estudiantes se conviertan en miembros activos de la sociedad. Para ello, la escuela debe dar respuestas educativas que eviten la discriminación y promuevan la igualdad de oportunidades.

En el currículo escolar, todos los alumnos tienen necesidades educativas comunes. Sin embargo, no todos los alumnos se enfrentan con las mismas capacidades de aprendizaje, sino que cada alumno tiene necesidades individuales. La mayoría de las necesidades individuales de los alumnos se abordan a través de acciones simples: dar a los alumnos más tiempo para aprender determinados contenidos, diseñar actividades, etc. Sin embargo, también existen necesidades individuales que no se pueden atacar por estos medios, lo que requiere una serie de medidas didácticas especiales diferentes de las normalmente requeridas para la mayoría de los estudiantes. En este contexto, hablamos de necesidades educativas especiales, y para satisfacerlas, son necesarias adaptaciones curriculares. Estas adaptaciones son realizadas por los profesores, pero no se les facilita una herramienta para ello, por esa razón nació AdaptaMaterialEscolar 1.0. El objetivo de esta aplicación es facilitar a los docentes una herramienta para realizar ajustes curriculares no significativos de ACNEE (alumnos con necesidades educativas especiales)

Chapter 2

Introduction

Capítulo 3

Estado de la Cuestión

En el estado de la cuestión es donde aparecen gran parte de las referencias bibliográficas del trabajo. Una de las formas más cómodas de gestionar la bibliografía en LaTeX es utilizando **bibtex**. Las entradas bibliográficas deben estar en un fichero con extensión *.bib* (con esta plantilla se proporcionan 3, dos de los cuales están vacíos). Cada entrada bibliográfica tiene una clave que permite referenciarla desde cualquier parte del texto con los siguiente comandos:

- Referencia bibliografica con cite: **?**
- Referencia bibliográfica con citep: (**?**)
- Referencia bibliográfica con citet: **?**

Es posible citar más de una fuente, como por ejemplo (**???**)

Después, latex se ocupa de rellenar la sección de bibliografía con las entradas **que hayan sido citadas** (es decir, no con todas las entradas que hay en el .bib, sino sólo con aquellas que se hayan citado en alguna parte del texto).

Bibtex es un programa separado de latex, pdflatex o cualquier otra cosa que se use para compilar los .tex, de manera que para que se rellene correctamente la sección de bibliografía es necesario compilar primero el trabajo (a veces es necesario compilarlo dos veces), compilar después con bibtex, y volver a compilar otra vez el trabajo (de nuevo, puede ser necesario compilarlo dos veces).

Capítulo 4

Chicha

4.1. Metodología de desarrollo

4.1.1. Clases de servicios

Las clases de servicio que emplearemos serán las siguientes:

- Expedite: tareas que necesitan ser gestionadas de manera acelerada o urgente.
- Fixed Delivery Date: tareas con fecha fija que debemos cumplir.
- Standard: tareas que ya ha hecho antes el equipo y que no tienen un fecha fija.
- Intangible: tareas que son nuevas, se desconoce el tiempo que se le va dedicar, el riesgo que suponen.

4.2. Plan de Pruebas

4.2.1. Tipos de pruebas

Previamente hemos distinguido dos tipos de tareas que habrá en el proyecto, las tareas relacionadas con la memoria y las de implementación. Cada tipo de tarea se probará o revisará de manera diferente.

4.2.2. Pruebas relacionadas con la memoria

La revisión de la memoria consistirá en la búsqueda de faltas ortográficas y errores gramaticales, y comprobación de la claridad del texto. También, se confirmará que no falta información. La revisión de la memoria la realizarán todos integrantes del equipo siguiendo los siguientes pasos:

1. Cuando haya una tarea de memoria en la columna de *DEV:DONE*, el miembro del equipo que vaya a revisarla se la asignará y la llevara a la columna de *TESTING*.

2. Tras acabar de revisar la tarea de memoria, la moverá de vuelta a la columna de *DEV:DONE*.

3. Cuando todos los miembros del equipo hayan revisado la tarea, el ultimo revisor se encargará de mover la tarea a la columna de *VALIDATE*.

4.2.3. Pruebas de implementación

Las pruebas de implementación consistirán de dos tipos de pruebas: las pruebas unitarias y las pruebas de integración. Las pruebas de una tarea de implementación concreta las realizará algún miembro del equipo que no haya participado en el desarrollo de esta y las hará cuando la tarea se encuentre en la columna de *DEV:DONE*. También se encargará de mover la tarea a la columna de *TESTING*. La ventaja de que las pruebas las realice un miembro que no se haya visto involucrado en el desarrollo de la tarea es que puede sacar más casos de prueba que aquellos miembros que han implementado la tarea y conocen el código. Para ambos tipos de pruebas, unitarias y de integración, se utilizará la herramienta Jest (ver sección de herramientas).

4.2.3.1. Pruebas Unitarias

Las pruebas unitarias son pruebas, mayoritariamente automatizadas, que verifican la funcionalidad de una unidad software (componente, clase o método), de forma aislada. Esta verificación no se debe ver afectada por otros casos de prueba que se hayan ejecutado anteriormente o por dependencias con otros módulos de la aplicación. Para evitar que las dependencias de un módulo, por ejemplo el acceso a una base de datos o una petición a una API externa, afecten al resultado de la prueba se pueden usar *stubs* para simular esas dependencias. La razón por la que vamos a realizar pruebas unitarias es porque permiten detectar errores en fases tempranas y evitar que los errores se propaguen a fases posteriores, aumentando la calidad del software. Además, facilitan los cambios, ya que se puede comprobar rápidamente que los cambios no han afectado al funcionamiento esperado de la aplicación. Para aprovechar al máximo las ventajas que ofrecen las pruebas unitarias hay que asegurarse de que cumplen las siguientes características:

- Deben ser automáticas y repetibles, es decir, se deben poder ejecutar tantas veces como uno quiera sin necesidad de intervención manual durante las pruebas.

- Deben ser rápidas, ya que si el ejecutar las pruebas unitarias es un proceso lento, no se va a llevar a cabo tantas veces como sea necesario.

- Cada caso de prueba debe estar aislado completamente del resto de casos o pruebas de otros módulos. Si no se aísla correctamente, se pueden producir resultados no consistentes, complicando la detección de errores.

- Las pruebas deben ser relevantes para el futuro y deben tratar de cubrir la totalidad del código.

4.2.3.2. Pruebas de integración

Las pruebas de integración se utilizan para comprobar que las conexiones o interfaces entre los distintos módulos, ya probados individualmente (pruebas unitarias), funcionan correctamente. En este proyecto las pruebas de integración se realizarán de forma incremental[1], es decir, los módulos se integran uno a uno, a medida que estos se van desarrollando y probando. En esta estrategia, la integración se lleva a cabo de arriba abajo, siguiendo el flujo natural de la aplicación. Se seguirá una estrategia *Top-Down* para realizar las pruebas. Utilizaremos esta estrategia, en vez de *Bottom-Up*, porque nos queremos centrar en errores de diseño y tener una aplicación funcional probada lo antes posible. Puede darse el caso de que una prueba de integración no se pueda realizar debido a que alguno de los módulos a integrar está en desarrollo o todavía se está probando individualmente. En este caso, se utilizarán *stubs* para simular la dependencia entre módulos.

4.2.4. Integración continua

Para complementar las pruebas unitarias y de integración, en este proyecto se implementará integración continua. La integración continua[2] es la práctica de desarrollo software mediante la cual los miembros del equipo combinan su trabajo frecuentemente en un repositorio compartido. Cada integración se verifica mediante una serie de fases por las que va pasando el software y que se automatizan, esta serie de fases se conoce como *pipeline*[3]. El *pipeline* que se va a implementar en este proyecto constará de 3 fases principales: *Build*, *Test* y *Release*. Cuando un miembro integre una nueva versión del proyecto en el repositorio compartido, pasará primero por la fase *Build*, durante la cual se instalarán las dependencias necesarias en una maquina virtual y se preparará todo lo necesario para pasar a la fase *Test*. En la fase *Test*, se ejecutarán todas las pruebas especificadas en el plan de pruebas a la versión que se esta intentando subir al repositorio. En caso de que el resultado de las pruebas sea correcto, se pasara a la fase *Release*, en caso de que se produzca un error en alguna de las pruebas se abortará la ejecución de

[1] `https://www.softwaretestinghelp.com/incremental-testing/`
[2] `https://aws.amazon.com/es/devops/continuous-integration/`
[3] `https://www.redhat.com/en/topics/devops/what-cicd-pipeline`

cualquier prueba restante, no se pasara a la fase *Release*, por lo tanto no se subirán los cambios realizados en la versión que se estaba compartiendo. En la fase *Release*, se subirán los cambios automáticamente al repositorio. En la figura 4.1 se muestra un diagrama del *pipeline*. Las razones por las que creemos necesaria la implementación de integración continua en este proyecto son las siguientes:

- En este proyecto se seguirá una metodología Kanban, por lo que el trabajo es continuo y es una metodología más propensa al cambio que otras metodologías agiles, como por ejemplo Scrum. La integración continua nos permitirá asegurarnos de que en todo momento el software cumple los requisitos especificados y funciona como se espera.
- La integración continua mejora la productividad de desarrollo, ya que libera a los desarrolladores de tener que ejecutar las pruebas manualmente y tener que esperar a que se arreglen errores en el repositorio. Si al integrar el trabajo se produce un error en la verificación se notificará a los desarrolladores y el cambio que ha producido ese error no se subirá al repositorio compartido.
- Mejora la detección de errores debido a la ejecución de pruebas de forma automática y frecuente, lo que permite a los desarrolladores descubrir los errores y arreglarlos antes de que se conviertan en problemas graves. En este proyecto se utilizará la herramienta de GitHub Actions (ver sección de herramientas) para implementar la integración continua y definir el pipeline.

Figura 4.1: Diagrama del *pipeline*

Capítulo 5

Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.

Chapter 5

Conclusions and Future Work

Conclusions and future lines of work.

Apéndice A

Título

Contenido del apéndice

Apéndice B

Título

Bibliografía

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes