

Universidad Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería Informática

Curso 2023-24

Trabajo Fin de Grado

**VSCODE4TEACHING: implementación de interfaz
web para el ecosistema para la docencia de la
programación en línea**

Autor | Diego Guerrero Carrasco
Tutor | Micael Gallego Carrillo

Julio de 2024



© 2024 Diego Guerrero Carrasco. Algunos derechos reservados.
Todas las figuras incluidas son de elaboración propia salvo indicación expresa.
El presente documento se distribuye bajo la licencia CC-4.0-BY-SA (*Atribución-CompartirIgual 4.0 Internacional* de *Creative Commons*). Se puede consultar más información acerca de esta licencia en:

<https://creativecommons.org/licenses/by-sa/4.0/deed.es>

Agradecimientos

A mis educadores vitales, mi familia, los que me han enseñado y allanado el camino, los que han dedicado su esfuerzo a formarme y a darme las facilidades que me han permitido crecer como persona. A mis abuelos, que se fueron muy pronto, gracias por dedicar vuestros últimos años a acompañarme durante mis primeros pasos; y en especial a Santos, el más longevo de ellos, el que vio con orgullo a su único nieto siendo el primer universitario de la familia. Gracias a ellos también por darme a mis padres, sempiternos baluartes, a quienes, a pesar de todos los contratiempos, les debo y les deberé siempre la vida tan fácil que me han regalado incondicionalmente.

A mis educadores intelectuales, a los profesores de los que he aprendido lo que sé durante los últimos veinte años. Gracias a los que me enseñasteis a razonar, analizar y aprender: a Antonio, a Teresa, a Silvia, a María Jesús y a tantos y tantos del Marillac. También a los de la Universidad, en especial a los que vivís con pasión vuestra gran profesión: a Sofía, a Mayte, a Ana y, en especial, a Mica.

A mis educadores sociales, a los que me habéis enseñado el valor de una buena amistad: a mis bachilleres, Miguel, Flor y Ana, que me habéis demostrado que, por mucho que diverjan los caminos, hay amistades que persisten; y a mis niños, José Luis, Andrea, Ismael, Debi y Flavia, gracias por todo lo que hemos disfrutado juntos dentro y, especialmente, fuera de la Universidad. Ojalá siga siendo así.

A mis educadores laborales, con los que he “estrenado” las ingenierías que todavía no tenía: al departamento de Aplicaciones Corporativas. Aunque este TFG no esté relacionado con nuestro cometido allí, de vosotros he aprendido cómo se aterriza nuestra fantástica profesión en el mundo real. Gracias a Juancar por nuestros debates sobre diseño y a todos los demás compañeros, en especial a mis coetáneos, a Gus, Julián y los Diegos, de los que he aprendido mucho. También a Santi, que hace que el trabajo sea mucho más llevadero.

Aun a riesgo de sonar petulante, termino agradeciéndome a mí las horas y el esfuerzo dedicados a cumplir el sueño que tengo desde hace quince años.

Gracias.

Resumen

Desde su origen, *VSCode4Teaching* ha venido siendo una extensión web para Visual Studio Code, un entorno de desarrollo integrado, que tiene como objetivo facilitar y potenciar la docencia de la programación informática para mejorar la educación en competencias digitales y en el ámbito de la informática, área en pleno crecimiento y promulgación a nivel global.

Para ello, *VSCode4Teaching* permite a los profesores crear y gestionar cursos con ejercicios de programación que se basan en una plantilla inicial propuesta por ellos y, opcionalmente, una propuesta de solución al ejercicio. Los alumnos inscritos en los cursos completarán los ejercicios descargándose la plantilla y realizando sobre ella su propuesta propia de resolución, sincronizándola con el servidor para guardarla e informar en tiempo real a sus profesores de los avances realizados hasta finalizarla.

La memoria de este Trabajo Fin de Grado describe en profundidad el ciclo de desarrollo relativo al cuarto hito evolutivo del proyecto *VSCode4Teaching*, en el que se implementa una aplicación web de navegador que incorpora los procesos de negocio que los usuarios ejecutaban en la extensión de Visual Studio Code, eliminando la obligatoriedad de uso de este entorno para alcanzar a un público objetivo mayor.

El *software* del proyecto se organiza en una arquitectura cliente-servidor. El servidor, encargado del suministro, persistencia e interpretación de los datos, intercambia información con dos clientes: la extensión para Visual Studio Code y la aplicación web de navegador, que disponen en consecuencia las interfaces gráficas necesarias para la interacción con la aplicación.

El proyecto *VSCode4Teaching* es *software* libre divulgado bajo licencia Apache 2.0 a través de un repositorio público en GitHub¹ que contiene, además, documentación sobre el proyecto para favorecer la libre ejecución, utilización y adaptación del proyecto a toda la comunidad de desarrolladores.

Palabras clave: educación, informática, programación, desarrollo de aplicaciones web, evolución del *software*, mantenimiento *software*.

¹ Repositorio: <https://github.com/codeurjc-students/2019-VSCode4Teaching>.

Abstract

Since its inception, *VSCode4Teaching* has been a web extension for Visual Studio Code, an integrated development environment, aimed at facilitating and enhancing the teaching of computer programming to improve education in digital skills and in the field of computer science, an area in full growth and promulgation at a global level.

To this end, *VSCode4Teaching* allows teachers to create and manage courses with programming exercises based on an initial template proposed by them and, optionally, a suggested solution to the exercise. Students enrolled in the courses will complete the exercises by downloading the template and providing their own developed solutions, synchronizing it with the server to save their work and inform their teachers in real-time of their progress until completion.

This Bachelor's Thesis report provides an in-depth description of the development cycle related to the fourth milestone of the *VSCode4Teaching* project, which involves implementing a web application that incorporates the business processes that users used to execute in the Visual Studio Code extension, thereby eliminating the mandatory use of this environment to reach a broader target audience.

The project's *software* is organized in a client-server architecture. The server, in charge of data supply, persistence and interpretation, exchanges information with two clients: the Visual Studio Code extension and the web browser application, which consequently provide the necessary graphical interfaces for interaction with the application.

The *VSCode4Teaching* project is free *software* released under the Apache 2.0 license through a public repository on GitHub², which also contains documentation about the project to encourage the free execution, use, and adaptation of the project by the developer community.

Keywords: education, computer science, programming, web applications development, *software* evolution, *software* maintenance.

² Repository: <https://github.com/codeurjc-students/2019-VSCode4Teaching>.

Índice de contenidos

Índice de figuras	XII
Índice de tablas	XIII
Índice de códigos	XV
1. Introducción	1
1.1. Contexto sociocultural	1
1.2. <i>VSCode4Teaching</i> : definición y cronología	3
1.3. Contexto técnico y motivación	8
1.4. Estructura del documento	10
2. Objetivos	11
3. Tecnologías, herramientas y metodología	13
3.1. Tecnologías	13
3.1.1. Aplicación web	13
3.1.1.1. Angular	14
3.1.1.2. <i>File System Access API</i>	16
3.1.1.3. RxJS	19
3.1.1.4. TypeScript	21
3.1.1.5. Node	21
3.1.1.6. Interfaz de usuario: SCSS, Bootstrap y Chart.js .	23
3.1.2. Servidor	24
3.1.2.1. Persistencia de la información: MySQL	24
3.1.2.2. Java y Spring	24
3.1.2.3. Maven	26
3.1.2.4. JUnit	27
3.1.3. Extensión para Visual Studio Code	28
3.1.4. Divulgación, despliegue y distribución	29
3.1.4.1. Divulgación del código: GitHub	29
3.1.4.2. Despliegue del servidor y la aplicación web: Docker	30

3.1.4.3. Distribución: Docker Hub y Visual Studio Code Marketplace	31
3.2. Herramientas	31
3.2.1. Herramientas para el producto	32
3.2.1.1. Edición de código	32
3.2.1.2. Sistema de control de versiones	32
3.2.1.3. CI/CD: GitHub Actions	33
3.2.2. Organización del proceso <i>software</i> : Trello	34
3.3. Metodología: proceso <i>software</i>	35
4. Descripción informática	39
4.1. Extracción de requisitos	39
4.1.1. Requisitos funcionales	40
4.1.2. Requisitos no funcionales	41
4.2. Diseño y arquitectura de la aplicación	42
4.2.1. Dominio de <i>VSCode4Teaching</i>	42
4.2.2. Arquitectura del proyecto: organización de componentes .	43
4.2.3. Diseño interno de cada componente	44
4.2.3.1. Aplicación web	45
4.2.3.2. Servidor	48
4.2.3.3. Extensión para Visual Studio Code	49
4.3. Implementación	50
4.3.1. Requisitos funcionales	50
4.3.1.1. RF-1: autenticación de usuarios y visualización de sus recursos asociados	50
4.3.1.2. RF-2: creación de nuevos ejercicios	52
4.3.1.3. RF-3: seguimiento en tiempo real de progreso de ejercicios	54
4.3.1.4. RF-4: obtención bajo demanda de ficheros de ejercicios	56
4.3.1.5. RF-5: configuración de publicación de solución de ejercicios	57
4.3.1.6. RF-6: gestión de matriculación de usuarios en cursos	58
4.3.1.7. RF-7: compartición de código asociado a un curso	59
4.3.1.8. RF-8: automatriculación mediante código de compartición	59
4.3.1.9. RF-9: visualización de un curso y sus ejercicios . .	60
4.3.1.10. RF-10: descarga de los ficheros de una propuesta propia de resolución de un ejercicio	61
4.3.1.11. RF-11: sincronización automática de las modificaciones en los ejercicios durante su realización . . .	63
4.3.1.12. RF-12: marcado de ejercicio como finalizado . . .	65

4.3.2. Requisitos no funcionales	65
4.3.2.1. RN-1: aviso a usuarios con navegadores no compatibles con la <i>File System Access API</i>	65
4.3.2.2. RN-2: aspecto visual de la aplicación web y coherencia entre interfaces	66
4.3.2.3. RN-3: encriptación de <i>tokens</i> JWT para autenticación	68
4.3.2.4. RN-4: adaptación de generación de imágenes Docker a nueva arquitectura	69
4.3.2.5. RN-5: migración a GitHub Actions del sistema de integración continua	71
4.4. Verificación de <i>software</i> y funcionalidad	74
4.4.1. Pruebas automáticas del servidor	74
4.4.2. Pruebas automáticas de la extensión	75
4.5. Distribución	76
4.5.1. Distribución del código fuente	76
4.5.2. Distribución de artefactos	77
5. Conclusiones y trabajos futuros	79
5.1. Cumplimiento de los objetivos estipulados	79
5.2. Trabajos futuros: hoja de ruta del proyecto <i>VSCode4Teaching</i> . . .	81
5.3. Aprendizajes personales	84
Bibliografía	85
Anexos	91
A. Detalle del algoritmo de sincronización de ejercicios	93

Índice de figuras

1.1.	Captura del <i>dashboard</i> para el seguimiento del progreso de los ejercicios en el primer TFG.	5
1.2.	Captura de la capacidad para comparación de ficheros en <i>VSCode4Teaching</i> en el primer TFG.	5
1.3.	Captura del <i>dashboard</i> para el seguimiento del progreso de los ejercicios en el segundo TFG.	6
1.4.	Captura de la barra lateral de la extensión con los nuevos iconos y menús contextuales.	7
1.5.	Captura del <i>dashboard</i> para el seguimiento del progreso de los ejercicios en el tercer TFG.	7
3.1.	Vista del tablero Trello empleado para la gestión del proceso <i>software</i>	36
4.1.	Dominio de <i>VSCode4Teaching</i> mediante notación UML.	42
4.2.	Disposición arquitectónica de los componentes de <i>VSCode4Teaching</i>	43
4.3.	Diagrama UML de las clases del modelo del dominio de <i>VSCode4Teaching</i>	44
4.4.	Diagrama de clases de las piezas <i>software</i> involucradas en el proceso de negocio para la gestión de estudiantes matriculados en un curso.	47
4.5.	Formulario de inicio de sesión para usuarios registrados en la aplicación web.	50
4.6.	Pantalla de inicio para usuarios autenticados, tanto estudiantes (arriba) como docentes (abajo).	51
4.7.	Interfaz principal para docentes acerca de cada uno de sus cursos impartidos.	52
4.8.	Ventana modal para la creación de nuevos ejercicios tras analizar los contenidos del directorio elegido.	53
4.9.	Instantánea de la ejecución del proceso de creación de nuevos ejercicios.	54
4.10.	<i>Dashboard</i> de un ejercicio en la aplicación web con métricas actualizadas en tiempo real para el seguimiento del progreso de los estudiantes.	55

4.11. Fragmento del <i>dashboard</i> que informa sobre la descarga de los ficheros de las propuestas de resolución de los estudiantes	56
4.12. Sección del <i>dashboard</i> de los ejercicios que permite configurar los parámetros para la configuración de la disponibilidad y comportamiento de la descarga de la propuesta de solución del docente.	57
4.13. Visualización de la lista de usuarios inscritos a un curso y de las capacidades para añadir nuevos usuarios o revocar inscripciones.	58
4.14. Información para la compartición del código único de automatrícu-la en un curso.	59
4.15. Página principal de los estudiantes en la aplicación web con un código de inscripción automática en un curso.	60
4.16. Situación inicial de los estudiantes al acceder al detalle de un curso y antes de elegir un directorio local.	61
4.17. Visualización detallada de un curso matriculado por un estudiante con ejercicios ubicados en distintos paneles según su estado.	62
4.18. Representación de la evolución entre los posibles estados visuales de los ejercicios en progreso según su disponibilidad local.	63
4.19. Captura de las señales de sincronización activa y estado de la sin-cronización de ejercicios en progreso.	63
4.20. Captura del modal explicativo de la sincronización de ficheros de un ejercicio.	64
4.21. <i>Dashboard</i> inicial de docentes en Firefox con aviso de incompati-bilidad para el uso de la <i>File System Access API</i>	66
4.22. <i>Dashboard</i> de un mismo ejercicio capturados a la vez en la apli-cación web (arriba) y en la extensión para Visual Studio Code (abajo).	67
4.23. Comparativa entre los formatos empleados para mostrar a los do-centes los ejercicios y configuraciones de las que disponen sus cursos.	68
4.24. Fragmento del listado de ejecuciones de los flujos de trabajo de integración continua con GitHub Actions.	73
4.25. Repositorio en GitHub del proyecto <i>VSCode4Teaching</i>	76
4.26. Extensión de <i>VSCode4Teaching</i> en el área de búsqueda y descarga de extensiones en Visual Studio Code.	77

Índice de tablas

1.1. Uso de los entornos de desarrollo más populares entre los estudiantes. 8

Índice de códigos

3.1.	Ejemplo de petición HTTP con el cliente de Angular (dependencia inyectada) y un observable de RxJS.	20
3.2.	Fragmento del documento <code>package.json</code> de la aplicación web Angular.	22
3.3.	Fragmento adaptado del fichero para la configuración de Maven aplicado en el servidor del proyecto (<code>pom.xml</code>).	27
4.1.	Fichero <i>Dockerfile</i> del proyecto, encargado de definir el proceso de generación de la imagen Docker del servidor y la aplicación web. .	69
4.2.	Fichero <i>docker-compose.yml</i> empleado para la orquestación de la imagen Docker del servidor con un contenedor para la base de datos.	70
4.3.	Fragmento del flujo de trabajo de acciones automáticas relativas a la integración continua del servidor.	72
A.1.	Método para el recorrido en profundidad de un directorio mediante la <i>File System Access API</i>	94
A.2.	Procedimiento para la interpretación de la lista de ficheros obtenida mediante el <i>ponyfill</i> como un árbol comparable.	94
A.3.	Algoritmo recursivo para la comparación de dos árboles de directorios dados dos nodos raíces <code>old</code> y <code>new</code>	95
A.4.	Algoritmo que remite trabajos de sincronización al servidor a partir de los contenidos de una cola de prioridad.	96

1

Introducción

La presente sección tiene como objetivo realizar una contextualización sociocultural del proyecto *VSCode4Teaching* y una cronología de la evolución del proyecto y sus iteraciones previas y, a continuación, exponer la motivación del presente Trabajo Fin de Grado y la necesidad que pretende resolver. Para concluir, se introduce brevemente el formato del presente documento y su división en epígrafes.

1.1. Contexto sociocultural

La *Declaración Universal de los Derechos Humanos* aprobada por la Asamblea General de la Organización de las Naciones Unidas en diciembre de 1948 recoge en su artículo 26 que “toda persona tiene derecho a la educación”, que “la instrucción elemental es obligatoria” y que “la educación tendrá por objeto el pleno desarrollo de la personalidad humana y el fortalecimiento del respeto a los derechos humanos y a las libertades fundamentales” [1]. Esta declaración continúa vigente y en plena expansión a nivel global, tal como demuestran los *Objetivos de Desarrollo Sostenible* contenidos en la *Agenda 2030* [2], un manifiesto de intenciones divulgado por Naciones Unidas en el año 2015 para el desarrollo integral del planeta y sus habitantes. Entre estos objetivos, el cuarto establece la necesidad de “garantizar una educación inclusiva, equitativa y de calidad”, destacando en su punto B la importancia del avance en los “programas técnicos, científicos, de ingeniería y de tecnología de la información y de las comunicaciones en los países desarrollados y otros países en desarrollo”.

Cabe conjugar este relevante papel de la educación como valor fundamental para el crecimiento y el desarrollo de la sociedad a nivel global con la incipiente relevancia que la informática tiene en la actualidad: todos los sectores sociales, culturales o laborales se han visto impregnados en mayor o menor medida de las capacidades que la digitalización puede brindarles para alcanzar nuevas cotas de desarrollo. El *Informe sobre la Conectividad Global* elaborado por la Unión Internacional de Telecomunicaciones [3], que es una agencia especializada de Naciones Unidas para las tecnologías de la información y las comunicaciones, refleja que el 63 % de la población global disponía de acceso a Internet en el año 2021, cifra que se reduce significativamente al explorar años anteriores: un 18 % en 2006, un 38 % en 2014 y un 46 % en 2017. Este crecimiento ha venido auspiciado en gran medida por la llegada de las tecnologías a las sociedades menos desarrolladas. Sin embargo, sigue existiendo una brecha en el acceso a Internet: mientras que el 87 % de los habitantes de Europa disfrutan de la red global descentralizada, solo un 33 % de los ciudadanos africanos la emplean. Además, de los habitantes europeos, el 94 % dispone de un acceso con una velocidad igual o superior a 10 Mbps³, mientras que en África este porcentaje se reduce hasta el 41 %, mientras que un 46 % disponen de un acceso de entre 2 y 10 Mbps, siendo el 12 % restante conexiones de peor velocidad.

Poniendo el foco en España, la digitalización de la educación es una de las grandes prioridades del marco educativo actual. Así lo refleja la actual Ley Orgánica para la Mejora de la Ley Orgánica de Educación [4] (LOMLOE), que trata en su preámbulo la “necesidad de tener en cuenta el cambio digital que se está produciendo y que afecta a la actividad educativa”, considerando que “el mundo digital es un nuevo hábitat en el que la infancia y la juventud viven cada vez más” y que, en consecuencia, “el sistema educativo debe adoptar el lugar que le corresponde en el cambio digital, con atención al desarrollo de la competencia digital de los estudiantes de todas las etapas educativas”. Tal es así que, ya en el año 2001, el escritor estadounidense Marc Prensky acuñó el término “nativos digitales” para aludir al conjunto de personas que han nacido durante la gran revolución tecnología que ha vivido la sociedad en las últimas décadas [5].

La educación de los “nativos digitales” debe producirse, por tanto, en consonancia con la sociedad en que se desarrollarán sus vidas, en la que deberán desenvolverse con soltura en la utilización de las herramientas informáticas que acompañen sus ámbitos sociocultural y laboral, lo que convierte a la informática en uno de los ejes fundamentales de la educación de los más pequeños de la sociedad. Intrínsecamente, este hecho conduce a la necesidad de la educación en la competencia digital, que toma un papel esencial en todos los currículos educativos de las enseñanzas obligatorias de las distintas comunidades autónomas españolas. Sirva como ejemplo Madrid, en cuyo currículo de enseñanzas para la Educación Secundaria Obligatoria incluye la palabra “digital” en 161 de las 321 páginas

³ Mbps. Abreviatura de “megabits por segundo”. Por ejemplo, una conexión de 1 Mbps permitirá recibir un máximo de 1000000 bits de información por segundo.

que lo componen y que establece la presencia de varias asignaturas relacionadas con este ámbito, tales como Tecnología y Digitalización, que es obligatoria en dos cursos de esta etapa educativa; y Ciencias de la Computación, una materia optativa ofrecida posteriormente que busca divulgar el pensamiento computacional, reflejando la vinculación esencial de la digitalización con la educación [6]. Esta educación orientada a la informática requiere que sus docentes dispongan de la capacidad y conocimientos adecuados para poder divulgarlos, por lo que se están ejecutando numerosos programas para la mejora de la competencia digital educativa, como *#CompDigEdu* [7], que es un plan de mejora articulado por el Instituto de las Tecnologías Educativas y de Formación del Profesorado (INTEF) que dispone el impulso de la competencia digital mediante la formación a los docentes y el diseño de planes para la expansión tecnológica en los centros educativos.

Este contexto de informatización de la educación y de situación de la competencia digital como un pilar esencial para el desarrollo de la sociedad en el que herramientas como *VSCode4Teaching* toman un papel crucial. Este proyecto busca facilitar la docencia de la computación a través de una aplicación que permite a los dos actores esenciales de la educación, docentes y estudiantes, interactuar en tiempo real a través de una aplicación informática que organiza y facilita las clases en torno a la programación informática.

1.2. *VSCode4Teaching*: definición y cronología

VSCode4Teaching es una herramienta que permite a docentes y estudiantes interactuar durante la enseñanza de la programación informática, facilitando su impartición y organización. Para ello, permite a los docentes crear cursos que, a su vez, quedan compuestos por ejercicios que están dotados de una plantilla inicial. Los estudiantes, por su parte, pueden visualizar los ejercicios de los cursos en los que estén matriculados y realizarlos, para lo que tendrán como punto de partida la plantilla propuesta por sus profesores y sobre la que realizarán modificaciones que se irán guardando progresivamente y quedando a disposición de los docentes. Una vez finalizados los ejercicios, los estudiantes los marcan como completados, bloqueando así la posibilidad de realizar nuevas ediciones. Los docentes disponen de información actualizada en tiempo real acerca del progreso de los estudiantes en la realización de los ejercicios y pudiendo visualizar los ficheros de cada propuesta de resolución en el momento que lo deseen.

El proyecto *VSCode4Teaching* se remonta al curso académico 2019-20, cuando Iván Chicano Capelo lo inició en el contexto de su Trabajo Fin de Grado [8]. Este trabajo sentó las bases del proyecto y dio lugar a una aplicación compuesta por dos componentes organizados en una arquitectura cliente-servidor. El cliente es

una extensión para Visual Studio Code⁴ que permite a docentes y estudiantes interactuar desde su entorno de desarrollo con el servidor, proporcionando una interfaz que usuario para consultar sus cursos impartidos o matriculados, los ejercicios de los que se componen y el estado de progreso en la realización de cada uno de ellos, permitiéndoles descargar sus ficheros asociados y visualizarlos dentro del propio entorno de desarrollo. Por otro lado, el servidor es el responsable de, a través de una API⁵, la interacción de la extensión con los sistemas de persistencia, ya que se ocupa de almacenar todas los ficheros de las propuestas de resolución de ejercicios y, además, todos los datos de la aplicación en una base de datos.

Tras este **primer** paso en el proceso evolutivo del proyecto, la aplicación ya disponía de los dos roles de usuarios —estudiantes y profesores— y de múltiples procesos de negocio implementados. Los docentes ya disponían de capacidades para: crear, actualizar y eliminar cursos, matricular y desmatricular estudiantes en sus cursos, añadir nuevos ejercicios cargando su plantilla asociada, modificar y eliminar los ejercicios, consultar en un *dashboard* información sobre su realización por parte de los estudiantes y obtener bajo demanda sus ficheros —tal como muestra la [figura 1.1](#)—, compartir un código para la automatización de estudiantes en los cursos, facilitar la comparación de las propuestas del estudiantado con la plantilla original de forma gráfica —reflejada en la [figura 1.2](#)— e introducir comentarios en las líneas de los ficheros de las propuestas de los estudiantes.

Los estudiantes, por otro lado, pueden: registrarse libremente en la aplicación, visualizar sus cursos matriculados, inscribirse en cursos nuevos a partir de un código de compartición, acceder a los ejercicios que componen los cursos y descargar los ficheros en su último estado de modificación (siendo este la plantilla inicial en caso de no haberlos iniciado), modificar y guardar su progreso en la realización de los ejercicios —sincronizándose con el servidor— y marcar los ejercicios como finalizados, impidiéndose la sincronización de sucesivas ediciones.

El **segundo** peldaño en la “escalera” evolutiva del proyecto *VSCode4Teaching* se produce durante el Trabajo Fin de Grado de Álvaro Justo Rivas Alcobendas [9]. Esta contribución potencia la usabilidad de la herramienta, incorporando numerosas características para lograr una experiencia de usuario más completa y que proporcione mejor información sobre todos los procesos de negocio realizados. A este respecto, destaca la incorporación de la actualización en tiempo real del *dashboard* de los docentes y una mejora en su apariencia y funcionalidad, dotándolo de un nuevo estado para los ejercicios “en progreso” y de botones para facilitar el acceso a la visualización de los ficheros que componen las propuestas de los estudiantes y sus diferencias respecto a la plantilla original, tal como se refleja en la [figura 1.3](#).

⁴ Visual Studio Code. Entorno de desarrollo integrado divulgado como *software* libre popularizado por su sencillez y gratuidad (más información en la [sección 3.2.1.1](#)).

⁵ API. Siglas de “interfaz de programación de aplicaciones” (del inglés *Application Programming Interface*). Es un componente *software* que permite la comunicación de agentes externos con el sistema que lo incorpora.

Capítulo 1. Introducción

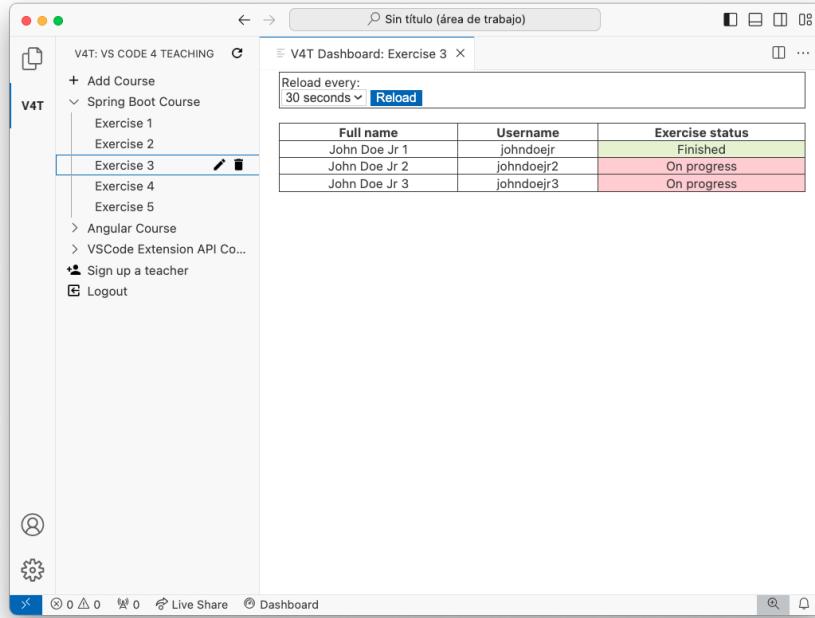


Figura 1.1: Captura del *dashboard* para el seguimiento del progreso de los ejercicios en el primer TFG.

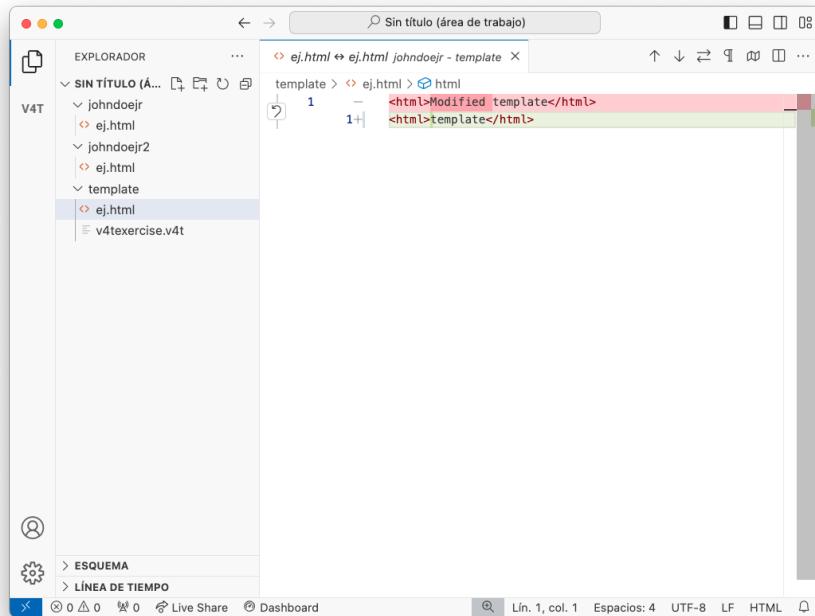


Figura 1.2: Captura de la capacidad para comparación de ficheros en *VSCode4Teaching* en el primer TFG.

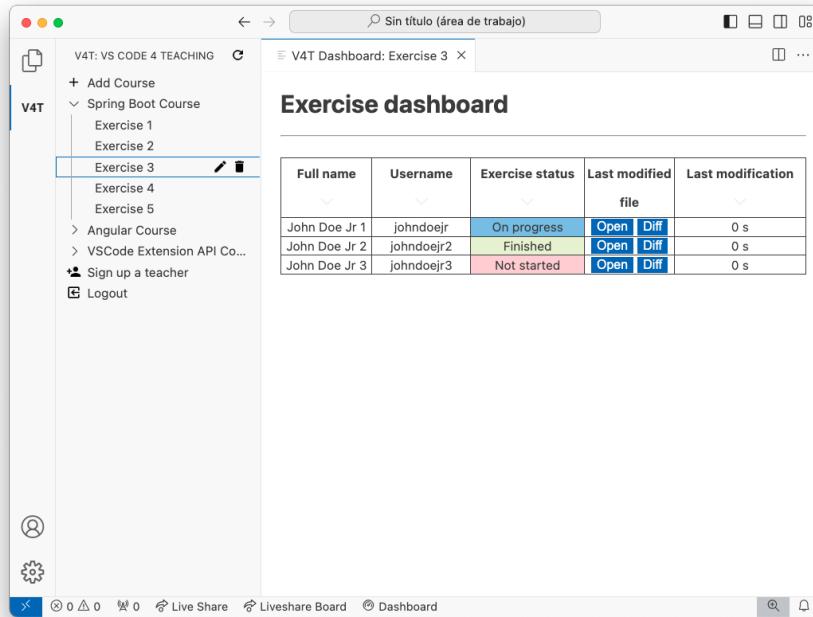


Figura 1.3: Captura del *dashboard* para el seguimiento del progreso de los ejercicios en el segundo TFG.

El **tercer** hito evolutivo del proyecto quedó materializado a través del TFG de Diego Guerrero Carrasco [10] (autor del presente Trabajo Fin de Grado). Esta iteración añade nuevas funcionalidades, tales como la capacidad para poder añadir a cada ejercicio una propuesta de solución elaborada por el docente, de modo que los estudiantes puedan descargarla cuando el profesor decida hacerla pública y compararla con su propia propuesta de resolución del ejercicio a través de una herramienta gráfica. A esta mejora se suma, además, la capacidad para poder crear múltiples ejercicios a partir de los subdirectorios existentes en una carpeta, la preservación del anonimato de los estudiantes en el sistema de ficheros de la aplicación, de modo que solo el docente pueda relacionar a un alumno con su propuesta de resolución; un nuevo proceso para el alta de docentes basado en un formato de invitaciones entre pares y numerosas mejoras en materia de usabilidad: un menú contextual disponible en los ejercicios y nuevos iconos para mostrar a golpe de vista el progreso en la ejecución de los ejercicios —tal como refleja la figura 1.4—, una nueva página de ayuda personalizada para cada curso compartido, con información específica del profesor que expide y divulga el código, y un rediseño del *dashboard* para hacerlo más visual e intuitivo —capturado en la figura 1.5—, añadiendo un nuevo gráfico del progreso de los estudiantes y otras métricas numéricas de utilidad sobre el ejercicio. En su faceta técnica, se añade durante esta fase, además, un nuevo componente al proyecto: una aplicación web que sirve como soporte para la implementación de algunas funcionalidades y que actúa de cliente en el modelo cliente-servidor descrito anteriormente.

Capítulo 1. Introducción

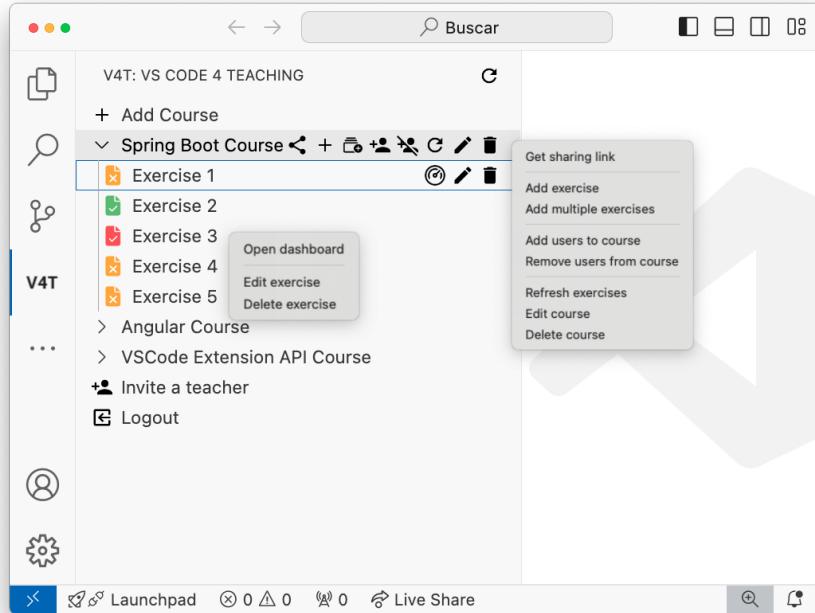


Figura 1.4: Captura de la barra lateral de la extensión con los nuevos iconos y menús contextuales.

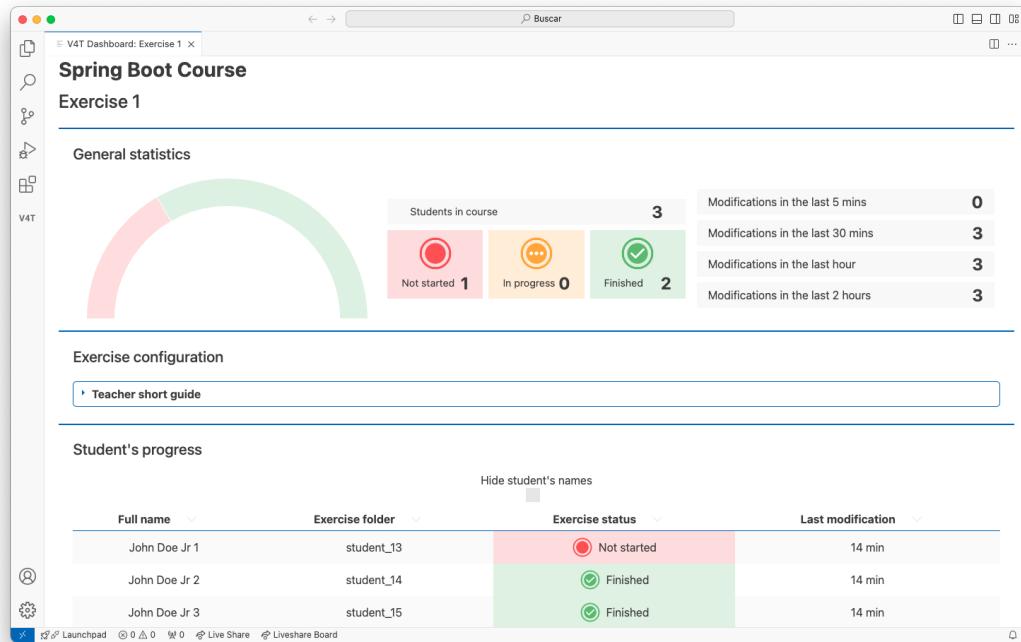


Figura 1.5: Captura del *dashboard* para el seguimiento del progreso de los ejercicios en el tercer TFG.

Tabla 1.1: Uso de los entornos de desarrollo más populares entre los estudiantes.

Entorno	Usuarios que lo utilizan		Usuarios que lo prefieren	
	Nº	%	Nº	%
Visual Studio Code	3512	67,67 %	1401	26,99 %
IntelliJ IDEA	2442	47,05 %	965	18,59 %
PyCharm	1779	34,28 %	424	8,17 %
Visual Studio	1063	20,48 %	239	4,61 %
Android Studio	953	18,36 %	145	2,79 %
CLion	806	15,53 %	164	3,16 %
Vim	745	14,35 %	87	1,68 %
WebStorm	725	13,97 %	144	2,77 %
Notepad++	595	11,46 %	24	0,46 %
Total	5190			

Nota: datos porcentuales respecto al total de estudiantes encuestados.

1.3. Contexto técnico y motivación

Una vez planteado el contexto en el que surge la necesidad general del proyecto *VSCode4Teaching* en la sección 1.1, y establecido qué es, cómo funciona y cómo se ha construido en la sección 1.2, se justifica a continuación la motivación que conduce a realizar el presente Trabajo Fin de Grado.

VSCode4Teaching es una aplicación que los usuarios únicamente pueden utilizar a través de un solo IDE⁶: Visual Studio Code.

A este respecto, la *Encuesta sobre el estado de los desarrolladores*, realizada anualmente por JetBrains, recoge en su edición de 2023 [11] las respuestas de 26 348 encuestados acerca del uso de distintas tecnologías y herramientas durante el ejercicio de su profesión. De entre ellos, cabe poner el foco en el 19,7% (5190) que afirman ser estudiantes de programación. Los datos recogidos para este grupo se incluyen en la tabla 1.1, que refleja las tasas de utilización y preferencia para varios entornos de desarrollo.

Entre los estudiantes destaca Visual Studio Code, siendo un entorno empleado por casi siete de cada diez estudiantes, aunque es el preferido por solo el 27% de este grupo. Sin embargo, la tabla permite constatar que existe un amplio espectro de estudiantes que prefieren utilizar otras opciones, sobresaliendo IntelliJ IDEA y PyCharm, entornos de desarrollo de la empresa JetBrains dedicados a la creación de proyectos sobre las plataformas Java y Python, respectivamente.

⁶ IDE. Siglas de “entorno de desarrollo integrado” (del inglés *Integrated Development Environment*).

Aunque un 67,67 % del público objetivo es un porcentaje adecuado, la tasa de estudiantes que afirman preferir Visual Studio Code es sensiblemente inferior, existiendo una gran heterogeneidad en la preferencia por los distintos entornos de desarrollo. De este modo, cabe poner sobre la mesa la posibilidad de ampliar el alcance del proyecto *VSCODE4Teaching*, tratando de llegar a la máxima cantidad de usuarios posible. La única alternativa que permitiría alcanzarlo es ofrecer la posibilidad de utilizar la aplicación fuera de Visual Studio Code.

Planteado este escenario, surgen diversas alternativas a ponderar para aumentar el alcance del proyecto. Una de ellas podría ser, por ejemplo, desarrollar una extensión paralela compatible con los entornos de JetBrains, que suman entre todos ellos —IntelliJ IDEA, PyCharm, CLion y WebStorm en la clasificación anterior— un 32,69 % de preferencia por el estudiantado encuestado. Sin embargo, tal como refleja la documentación de JetBrains al respecto [12], el conjunto de tecnologías empleado para ello es notablemente diferente del que utiliza la extensión para Visual Studio Code, ya que requieren utilizar Kotlin como lenguaje de programación y Gradle para la gestión de la configuración del *software*, hecho que dificultaría el mantenimiento del proyecto *software* al añadir un componente nuevo que, además, requiere formas de mantenimiento diferentes de los demás y, por otro lado, la cota de público objetivo alcanzado aumenta hasta el 59,69 % de usuarios, que es una cifra elevada pero no lo suficientemente como para ejecutar este planteamiento.

Por tanto, la decisión es generar un nuevo cliente distinto de la extensión para Visual Studio Code que se base en una plataforma que sea independiente del IDE que empleen estudiantes o profesores. Generar una aplicación de escritorio conduciría a la misma conclusión que la extrapolada al inicio de este planteamiento, y es que cada usuario puede estar empleando un sistema operativo distinto, hecho que exige la construcción de varias aplicaciones que permitan utilizar *VSCODE4Teaching*. Por tanto, de nuevo, la opción de trasladar la funcionalidad a una aplicación de escritorio queda descartada por su complejidad técnica y la dificultad para su mantenimiento.

Con esta motivación y tras el análisis de las opciones disponibles, surge la idea de desarrollar una aplicación web que permita realizar todos los procesos de negocio que *VSCODE4Teaching* ya tiene implementados a través de una interfaz. De este modo, los usuarios podrían emplear el entorno de desarrollo y el sistema operativo de su preferencia, dependiendo únicamente del uso de un navegador web actual para garantizar que el proceso más relevante, que es la sincronización bidireccional de ficheros ubicados en el sistema local de los usuarios, puede realizarse correctamente.

1.4. Estructura del documento

El presente documento describe detalladamente todos los trabajos realizados sobre el proyecto *VSCode4Teaching* en su cuarto hito evolutivo.

El siguiente apartado ([capítulo 2](#)) establece los objetivos que se buscará satisfacer mediante la realización del Trabajo Fin de Grado.

El tercer punto ([capítulo 3](#)) recoge el abanico de tecnologías y herramientas empleadas para la ejecución del trabajo, así como la metodología empleada y la organización para su realización.

El apartado cuarto ([capítulo 4](#)) incluye toda la descripción informática del trabajo ejecutado. Cabe reseñar que la aplicación queda compuesta por tres grandes componentes: el servidor, la extensión para Visual Studio Code y la aplicación web para navegadores, siendo este último el componente sobre el que se implementarán la práctica totalidad de los objetivos y requisitos.

Para finalizar, el capítulo quinto ([capítulo 5](#)) extrae conclusiones tras finalizar el desarrollo, analizando la completitud de las necesidades establecidas y analizando trabajos futuros tentativos a realizar en sucesivas iteraciones.

A continuación, una vez finalizado el cuerpo del documento, se introduce la enumeración completa de las referencias bibliográficas a las que aluden las citas introducidas durante el desarrollo del documento.

Nota: este documento contiene numerosos enlaces e hipervínculos a otras secciones y epígrafes del documento, así como a las referencias bibliográficas, quedando escritos en [color rojo](#). En su edición electrónica, el lector puede hacer *click* sobre estos enlaces para ser conducido al punto al que referencian.

2

Objetivos

El [capítulo 1](#) ha introducido pormenorizadamente la contextualización socio-cultural que constata la necesidad del proyecto *VSCode4Teaching*, una cronología de las distintas evoluciones que lo han ido conformando y la necesidad de generar una nueva interfaz para que los usuarios puedan utilizar esta herramienta sin necesidad de hacer uso de Visual Studio Code.

El presente Trabajo Fin de Grado busca alcanzar el cuarto hito en la evolución del proyecto, poniendo el foco en la necesidad anteriormente descrita con el fin de alcanzar a la mayor cantidad de potenciales usuarios posible, facilitando el acceso y la utilización de la herramienta. Esta necesidad es la piedra angular de este Trabajo Fin de Grado, que fija como su objetivo principal alcanzar al mayor público objetivo posible.

Traducido al plano técnico, este objetivo se va a materializar a través de la **implementación de un componente nuevo** en *VSCode4Teaching*: una aplicación web que actúe como cliente (esto es, un *frontend*), recogiendo toda la funcionalidad posible de la extensión y haciéndola independiente de Visual Studio Code, desapareciendo así el único requisito *software* que se imponía a los usuarios. De este modo, la nueva aplicación web permitirá a sus usuarios ejecutar todos aquellos procesos de negocio que venían pudiendo realizar en la extensión para Visual Studio Code y que no dependiesen específicamente del citado entorno de desarrollo. Este matiz conduce a la exclusión de características como, por ejemplo, la capacidad para introducir comentarios textuales asociados a las líneas de código de los ficheros, característica intrínsecamente vinculada al entorno de desarrollo.

El proyecto ya dispone de una pequeña aplicación web implementada mediante Angular que se emplea como componente auxiliar para la ejecución de funcionalidades como el registro en dos partes de nuevos docentes mediante invitación y la visualización de una página de ayuda personalizada para alumnos cuando reciben una invitación a un curso por parte de un profesor. Como el alcance pretendido es más amplio y se dirige a la obtención de un componente renovado y completo, el objetivo principal de este Trabajo Fin de Grado en el plano técnico es el de generar una nueva aplicación web desde cero que permita alcanzar el objetivo de dominio, reemplazando el anterior *frontend* existente.

Por todo lo anterior, cabe concluir que esta nueva evolución del proyecto *VSCode4Teaching* busca acoplar al proyecto una aplicación web de lado cliente que permita a estudiantes y profesores operar con sus cursos y ejercicios, pudiendo ejecutar todos los procesos de negocio posibles de los previamente existentes. Este hecho conduce a que objetivos de dominio de este Trabajo Fin de Grado se fundamenten en los objetivos de negocio establecidos en las tres iteraciones previas, entre los que cabe destacar los siguientes seis objetivos:

1. Autenticación y visualización personalizada de cursos y ejercicios: permitir a los usuarios autenticarse, de modo que cada usuario podrá visualizar sus cursos inscritos o impartidos y los ejercicios que los compongan.
2. Impartición de cursos y ejercicios por docentes: brindar a los docentes las herramientas para gestionar sus cursos y añadir en ellos ejercicios a partir de plantillas iniciales y, opcionalmente, incorporándoles propuestas de solución que estarán disponibles al estudiantado cuando el docente lo deseé.
3. Seguimiento activo de ejercicios para docentes: dotar a los docentes de las herramientas necesarias para seguir el progreso de los estudiantes al realizar los ejercicios de sus cursos, obteniendo información actualizada en tiempo real y pudiendo descargar sus propuestas de resolución.
4. Matriculación del estudiantado en cursos: permitir a los docentes gestionar los estudiantes que están matriculados en sus cursos y, además, compartirlos para permitir a los estudiantes automatricularse en ellos.
5. Realización de ejercicios en cursos matriculados por alumnos: otorgar a los estudiantes la capacidad para descargar los ficheros asociados a los ejercicios de los cursos de los que forman parte para realizarlos, sincronizando las modificaciones que vayan realizando según se produzcan.
6. Calidad del proyecto y de su software: realizar actuaciones para mantener y mejorar el propio proyecto *software*, migrando y ampliando el sistema de integración continua y, además, mejorando y automatizando la generación de los artefactos compilados de la aplicación para adecuar su despliegue y distribución al nuevo componente introducido.

3

Tecnologías, herramientas y metodología

Se presenta a continuación la diversidad de tecnologías ([sección 3.1](#)) y herramientas ([sección 3.2](#)) involucradas en el proyecto *VSCode4Teaching*, así como la metodología de trabajo empleada ([sección 3.3](#)) para realizar el presente trabajo.

3.1. Tecnologías

Esta sección recoge información acerca de las tecnologías empleadas en el proyecto *VSCode4Teaching*, describiéndolas en subsiguientes secciones dispuestas para cada uno de los componentes del proyecto (al respecto de los componentes y su arquitectura, véase la [sección 4.2](#)): la aplicación web ([sección 3.1.1](#)), que toma el papel protagonista en el presente hito evolutivo, la extensión para Visual Studio Code ([sección 3.1.3](#)) y el servidor ([sección 3.1.2](#)). A estas secciones se añade, además, una sección adicional acerca de las tecnologías para la divulgación del código de la herramienta y su distribución y despliegue ([sección 3.1.4](#)).

3.1.1. Aplicación web

La nueva aplicación web de *VSCode4Teaching* se basa en **Angular**, que emplea **Node** como plataforma subyacente, que se implementa utilizando el lenguaje **TypeScript** y que hace uso de bibliotecas para la comunicación con el servidor (**RxJS**), para la interacción con el sistema local de ficheros (mediante la **File System Access API**) y para la mejora de la **interfaz de usuario**.

3.1.1.1. Angular

Angular es un “framework⁷ web que facilita a los desarrolladores la creación de aplicaciones rápidas y fiables” [13]. Fue desarrollado y está activamente mantenido por una subdivisión de Google, es de código abierto divulgado bajo licencia MIT y permite la generación de aplicaciones web SPA⁸ orientadas a componentes.

Las aplicaciones web SPA son aquellas que requieren una única obtención de sus contenidos básicos (su estructura declarada mediante HTML, sus hojas de estilo en formato CSS y sus *scripts* implementados mediante JavaScript) en la primera petición efectuada, actualizando su interfaz durante la interacción de los usuarios ejecutando peticiones en segundo plano y actualizando en consecuencia el DOM⁹. Este tipo de aplicaciones aporta ventajas en materia de rendimiento, ya que en la primera petición se devuelve todo el código y los *scripts* necesarios, soliendo ser más costosa en tiempo y memoria, mejorando así la experiencia de usuario en las sucesivas interacciones dentro de la propia aplicación [14].

Otra de las principales características que suele acompañar al desarrollo de aplicaciones web SPA es su orientación a componentes. Por ejemplo, Angular define el concepto de componente como “el bloque fundamental para la creación de aplicaciones” [15]. Este *framework* fundamenta su arquitectura en la disposición de los componentes y las interrelaciones que establecen entre sí. Tal ha sido su popularización por parte de las bibliotecas de generación de aplicaciones SPA que este concepto está en vías de ser integrado en los estándares de HTML y JavaScript para su utilización sin necesidad de bibliotecas adicionales a través de los *Web Components* [16], siendo compatibles aún solo con las versiones más modernas de los navegadores más populares.

Tras exponer las principales ventajas aportadas por las aplicaciones web SPA, cabe analizar el estado del arte a este respecto. Son tres las tecnologías más recurrentemente empleadas: React, Vue y Angular. La *Encuesta de Desarrolladores* de Stack Overflow señala en su edición de 2023 que React es la tecnología para *frontend* favorita para el 42,87% de desarrolladores profesionales encuestados, frente a Angular (19,89%) y Vue (17,64%) [17].

Del citado artículo junto con otro que remarca las diferencias entre Angular y React [18], se extrapolan algunos puntos clave sobre las dos opciones más extendidas. Por un lado, Angular, aunque tiene una curva de aprendizaje más compleja,

⁷ *Framework*. Del inglés “marco de trabajo”, es un conjunto de conceptos, prácticas y estándares que facilitan la consecución de un determinado objetivo, como la creación de aplicaciones o la gestión de un proceso *software*.

⁸ SPA. Siglas de “aplicación de página única” (del inglés *Single Page Application*).

⁹ DOM. Siglas de “modelo de objetos del documento” (del inglés *Document Object Model*). Es uno de los objetos principales en el contexto de un navegador web y contiene una estructura arborescente mutable y navegable que representa los elementos dispuestos en la interfaz de una web y se genera a partir del código HTML recibido.

está bien mantenido y documentado, es escalable, favorece la modularidad y se basa en TypeScript; mientras que React mejora en rendimiento durante la renderización, es más fácil de aprender, únicamente se preocupa de la vista, requiriendo dependencias adicionales para dar soporte a las demás capas de la arquitectura; e introduce una sintaxis propia (JSX) para intercalar HTML y JavaScript.

Es necesario reseñar, además, una diferencia radical entre ellos: mientras que Angular es un *framework* “completo” que impone ciertos conceptos y prácticas durante el ciclo de desarrollo y que toma el control de la aplicación en tiempo de ejecución, React es una “biblioteca para interfaces de usuario web y nativas” [19] que requiere de la instalación de librerías adicionales para alcanzar una funcionalidad similar. Esta diferencia es fundamental, ya que permite a Angular brindar mecanismos como la inyección de dependencias, que facilita su aprendizaje a desarrolladores acostumbrados a la orientación a objetos, mientras que React utiliza objetos del ámbito global en el navegador para la compartición del estado entre componentes.

Al ser un *framework*, Angular está “opinionado” (por traducción literal del inglés “opinionated”); esto es, su uso viene marcadamente influenciado por las sugerencias que elaboran sus propios creadores para hacer las aplicaciones, pasando por la utilización de un determinado lenguaje —TypeScript—, de una determinada arquitectura —basada en componentes— y de, en definitiva, un método recomendado pero no obligado para el desarrollo de proyectos.

El *framework* utilizado en este proyecto es Angular. A todas las ventajas justificadas anteriores se suman, además, tres factores dependientes del contexto: es el *framework* empleado en la asignatura de Desarrollo de Aplicaciones Web del Grado en Ingeniería del *Software*, lo que facilita su posterior mantenimiento por otros autores en el contexto en el que se viene desarrollando este proyecto; ya se venía utilizando en *VSCODE4Teaching*, que ya contaba con una pequeña aplicación web basada en Angular como página de inicio y para la implementación del registro de nuevos profesores mediante invitación —tal como se introduce en la [sección 1.2](#); y su uso se basa en las mismas plataformas que la extensión para Visual Studio Code, que son TypeScript como lenguaje de programación y Node como plataforma subyacente, lo que hace que tenga una curva de aprendizaje compartida entre los dos componentes y, además, facilita la posibilidad de reutilización de código.

El ecosistema de Angular dispone de una amplia variedad de funcionalidades que se pueden emplear en estas aplicaciones, entre las que destacan: Angular CLI, una herramienta por línea de comandos que ayuda a los desarrolladores a generar *scaffoldings*¹⁰ de nuevas piezas *software*, a actualizar la versión de las dependencias Angular añadidas o a ejecutar, compilar y desplegar las aplicaciones, entre

¹⁰ *Scaffolding*. Del inglés “andamio”, alude al conjunto de ficheros que conforman una plantilla inicial sobre la que se asienta una nueva unidad *software*.

otros; Angular Forms, que permite controlar integralmente el funcionamiento de los formularios, con métodos avanzados para la validación de las entradas y proporcionando *two-way data binding* (es decir, la capacidad de cambiar el valor de variables a partir de la interfaz de usuario, y viceversa); o Angular Router, para hacer aplicaciones SPA con capacidad de establecer un sistema de enrutamiento virtual que permita diferenciar las distintas pantallas disponibles. Además de todas las citadas, que han sido utilizadas en la aplicación web, existen algunas más, como SSR, para habilitar *Server-Side Rendering*, de modo que la vista se renderice en el servidor y el cliente la reciba preparada para injectarla en el DOM; Material, que añade una amplia colección de componentes preparados para la interacción con los usuarios; o Localize, que facilita ofrecer una misma aplicación en múltiples idiomas.

Este proyecto utiliza Angular 16, publicado en mayo de 2023. Habitualmente, se libera una versión *major*¹¹ nueva del *framework* cada seis meses, hecho que dificulta mantener actualizados los proyectos a la última versión. Además, entre las versiones 16 y 17 se han producido numerosos cambios en el método de funcionamiento del *framework*, por lo que se ha pospuesto la actualización a alguna versión más reciente.

3.1.1.2. *File System Access API*

La *File System Access API* (del inglés, “interfaz para el acceso al sistema de ficheros”, y anteriormente llamada *Native File System API*) es una interfaz proporcionada por algunos navegadores web para la interacción de las aplicaciones con el sistema de ficheros local de los usuarios.

Antes de entrar en detalle acerca de esta tecnología, cabe hacer un breve apunte acerca de la amplia variedad de navegadores web disponible y sus tendencias de uso. Según StatCounter [21], los más utilizados en entornos de escritorio durante 2023 fueron Opera (3,23 %), Mozilla Firefox (6,64 %), Apple Safari (8,8 %), Microsoft Edge (13,13 %) y Google Chrome (64,88 %). Cada uno de estos navegadores ha sido creado y es mantenido por distintas corporaciones, hecho que ha conducido a una gran diversidad en los motores e intérpretes que utilizan. Los motores de renderizado son los componentes *software* que los navegadores emplean para generar elementos gráficos a partir de la conjunción de la estructura definida mediante HTML y las hojas de estilo declaradas en CSS, siendo los más destacados *WebKit* en el caso de Safari [22], *Gecko* en Firefox [23] y *Blink* en los navegadores basados en Chromium [24], Chrome y Edge [25, 26]. Por otro lado, los intérpretes de JavaScript, empleados para leer el código de los *scripts* y

¹¹ *Major*. En el sistema de versionado semántico¹², primera cifra en la nomenclatura de cada versión. Un salto de versión *major* puede no ser retrocompatible.

¹² Versionado semántico. Sistema para la nomenclatura de nuevas versiones que se basa en la concatenación de tres secuencias numéricas: *major*, *minor* y *patch*, ampliamente utilizado [20].

ejecutarlo, son también diferentes para cada navegador, destacando *SpiderMonkey* para Firefox [27] y *V8* en el caso de Chrome y Edge [28]. Como resultado de esta heterogeneidad, se hace necesario revisar la compatibilidad de los *scripts* implementados con las opciones disponibles en cada navegador, ya que un mismo código puede funcionar de forma diferente según el intérprete que lo ejecute.

La *File System Access API* sienta sobre su base sobre la *File System API* [29], que es un “estándar vivo” de WHATWG (siglas de *Web Hypertext Application Technology Working Group*), una comunidad integrada por desarrolladores encargados de actualizar muy frecuentemente varios estándares, entre los que destaca el que define HTML, promovida por grandes compañías de la informática como Apple, Mozilla Foundation u Opera. Este estándar define una interfaz para operar con sistemas de ficheros desde las aplicaciones web ejecutadas en el navegador a través de *scripts*, estableciendo los métodos disponibles en los manejadores de ficheros (*FileSystemHandle*) o directorios (*FileSystemDirectoryHandle*) y en el BOM¹³, tales como *window.showOpenFilePicker*, para mostrar un diálogo que permita al usuario elegir un fichero en su sistema de ficheros local y dotar al navegador de un manejador del fichero basado en su contenido y metadatos; o *window.showDirectoryFilePicker*, análogo para los directorios y sus contenidos; además de todos los métodos disponibles en las instancias concretas de las piezas *software* que define este estándar.

Aunque esta API está parcialmente implementada en todos los navegadores, algunos de los más utilizados, como Firefox o Safari, restringen su uso a un sistema de ficheros privado y encapsulado localizado dentro del navegador llamado OPFS (siglas de “sistema de ficheros de origen privado”, del inglés *Origin Private File System*), accesible desde el BOM mediante el objeto *navigator.storage*, de modo que estos navegadores no pueden operar con el sistema de ficheros local real del usuario que ejecuta una aplicación web que emplea esta API.

La *File System Access API* propone complementar la *File System API*. Esta API es actualmente un borrador divulgado por el WICG¹⁴, que es un grupo dentro del W3C¹⁵ que sirve como “lugar para la experimentación y discusión de nuevas características para la plataforma web” [30]. La propuesta de la *File System Access API* tiene como propósito “dotar a los desarrolladores de la capacidad para construir aplicaciones web que interactúen con ficheros en el dispositivo local del usuario [...], permitiendo modificar los archivos y trabajar con directorios” [31]. Para ello, propone la definición de los métodos *queryPermission*

¹³ BOM. Siglas de “modelo de objetos del navegador” (del inglés *Browser Object Model*). Es el conjunto de objetos ofrecidos por el intérprete de un navegador, habitualmente designado como *window*, ya que suele ceñirse al contexto de una ventana o pestaña. Dentro de él se encuentra el DOM y permite ejecutar todas las operaciones accesibles del intérprete de *JavaScript*, por lo que su funcionalidad depende del navegador empleado.

¹⁴ WICG. Siglas de “Grupo de la Comunidad para la Incubación de la Web”, del inglés *Web Incubator Community Group*.

¹⁵ W3C. Siglas de “Consorcio Web”, del inglés *World Wide Web Consortium*.

y *requestPermission*, que permiten solicitar permiso al usuario para ver o escribir (según se requiera) en el fichero o directorio que deseé, dando como resultado una ampliación sobre las clases que ya proponía la *File System API*. La definición de esta interfaz ampliada se preocupa de acotar los directorios que deben restringirse al implementarla en los navegadores, excluyendo los directorios críticos de los distintos sistemas operativos, aquellos que contienen el propio agente de usuario, la carpeta del usuario en sí misma y los directorios por defecto para el escritorio, los documentos o las descargas del usuario, entre otras.

Al no estar estandarizada, la compatibilidad de esta API queda restringida a cuantos navegadores la implementen. Los desarrolladores de Chrome son impulsores del uso de esta interfaz, lo que hizo que este navegador fuese el primero que dispusiese de ella, incorporándola en octubre de 2020. Además, publicaron un artículo divulgativo sobre ella en la que describen varios usos interesantes de la API acompañados de *snippets*¹⁶ de código [32]: lectura y escritura de ficheros, especificación para guardar ficheros en local con sugerencia de nombre y ubicación, manejo de permisos concedidos por el usuario para acceder a un directorio, listado de contenidos de un directorio, creación o exploración de ficheros y directorios dentro de directorios, resolución de ruta relativa de ficheros localizados en directorios, renombrado, reubicación o eliminación de ficheros o directorios, integración con *drag and drop*¹⁷ y métodos para el acceso al OPFS. Este artículo, además, va acompañado de una implementación de un editor web de ficheros localizados en el sistema local que, divulgado públicamente bajo licencia Apache 2.0, sirve como base y ejemplo de uso de gran parte de la funcionalidad de esta API [33].

La heterogeneidad existente en la implementación de las interfaces por parte de los distintos intérpretes de los navegadores y la restricción vigente en Firefox y Safari, que no implementan los métodos de la *File System Access API*, hace que el principal proceso de negocio de este proyecto, que es el intercambio bidireccional de ficheros entre el servidor y el cliente de *VSCODE4Teaching* para sincronizar las propuestas de los estudiantes y las plantillas de los ejercicios del profesorado, esté vigente únicamente en los navegadores Chrome, Edge y Opera, sumando un alcance de un 81,24 % de usuarios, aunque la restricción se puede ver fácilmente paliada al cambiar de navegador.

Una forma de mitigar parcialmente los efectos adversos derivados de esta divergencia en el funcionamiento de los navegadores es el uso de *polyfills*, que son “fragmentos de código que proveen la tecnología que se esperaría que un navegador proporcionase nativamente” [34]. En la práctica, los *polyfills* funcionan en tiempo de ejecución, y detectan si el intérprete del navegador que los ejecuta

¹⁶ *Snippet*. Traducido literalmente del inglés como “retal”, es un pequeño fragmento de código.

¹⁷ *Drag and drop*. Del inglés “arrastrar y soltar”, se refiere a la capacidad para poder seleccionar uno o varios ficheros o directorios y arrastrarlos con el cursor hasta un área habilitada para la recepción, en la que se suelta el cursor. Es una interacción de usuario sustituta de la capacidad para elegir ficheros o directorios en un diálogo específicamente dispuesto con ese fin.

dispone de los métodos deseados y, si no es así, añade una implementación efímera al vuelo, ya que quedará eliminada al cambiar de página.

Esta característica de sustitución en tiempo de ejecución que proponen los *polyfills* puede no ser adecuada si se producen incompatibilidades entre la especificación original y la sustitutiva, pudiendo producir errores inesperados o difíciles de controlar. Para solucionar este problema, surge el concepto de *ponyfill* [35], que comparte finalidad con el anterior pero, en lugar de añadir implementaciones en tiempo real sobre las clases que no dispongan de los elementos deseados, propone una implementación independiente invocada utilizando una llamada a un módulo distinto, una biblioteca, que, internamente, dispondrá de una implementación no acoplada al estándar que se buscaba complementar originalmente, evitando utilizar la API nativa no soportada en su totalidad para garantizar que, con independencia de si el navegador que ejecuta el *ponyfill* disponía de la funcionalidad deseada, funcione adecuadamente a través del mecanismo sustitutivo.

La contribución de los redactores del artículo de Chrome Labs anteriormente citado [32] no solo se ciñe al editor de texto, sino que, conscientes de la limitación para utilizar la API en algunos navegadores, implementaron y divulgaron un *ponyfill* llamado *browser-fs-access* [36], en el que también se basa el editor de texto que publicaron como caso de uso [33], que dispone de implementaciones alternativas de los métodos de la *File System Access API* basados en la utilización de elementos HTML como campos de tipo fichero o enlaces destinados a la descarga de documentos.

Este proyecto utilizó este *ponyfill* para la implementación del proceso de sincronización de ficheros de estudiantes con el servidor, tal como se detalla en la sección 4.3.1.11. Además, se aprovecha la capacidad de este *ponyfill* para advertir a los usuarios que utilizan navegadores no compatibles con la *File System Access API* para mostrar por pantalla una recomendación para que cambien de navegador a otro que sí disponga de ella, tal como se define en la sección 4.3.2.4.

3.1.1.3. RxJS

La programación reactiva es un paradigma de programación situado dentro del declarativo que basa su funcionamiento en el procesamiento de eventos asíncronos y en los flujos de datos [37]. Esta marcada presencia de la asincronía se asocia intrínsecamente a una ejecución de código no bloqueante, declarando qué acciones se desearán ejecutar en el momento en que se reciba un dato a través de un flujo. Una vez declarado, continúa la ejecución del código sin pausarlo, ejecutando las acciones especificadas en el momento de recibir algún dato nuevo y dándolo a conocer a aquellos elementos que estén interesados en obtener la información procesada para, a su vez, producir nuevos eventos o alimentar otros flujos de datos.

La anterior descripción permite vislumbrar la aplicación práctica de un patrón de diseño clásico: *Observer* [38], en el que un informante tiene constancia de un grupo de observadores que quedan inscritos para recibir notificaciones sobre la modificación de su estado para, como consecuencia, realizar acciones derivadas de estas alteraciones. La programación reactiva bautiza al informante como “observable”, a los observadores como “suscriptores”; y a la acción que realizan estos sobre los primeros, “suscripción”.

RxJS es una biblioteca para JavaScript y TypeScript que permite “componer programas asíncronos y basados en eventos mediante la utilización de secuencias observables” [39]. Divulgada como *software* libre bajo licencia Apache 2.0, es una de las implementaciones del proyecto *ReactiveX*, que propone múltiples implementaciones para llevar la programación reactiva a distintas plataformas *software* [40]. Así, RxJS pone a disposición de sus usuarios clases que modelan observables y suscriptores a través de suscripciones, además de una amplia variedad de operadores para poder generar nuevos observables o para fusionar, transformar, mezclar o gestionar errores en los flujos asíncronos producidos por observables.

El cliente HTTP de Angular, incorporado por defecto en el *framework*, permite preparar peticiones utilizando los observables de RxJS. Un uso típico estas tecnologías combinadas se ilustra en el [código 3.1](#), en el que se emplea un operador RxJS para convertir la información entrante en la respuesta de la petición en forma de DTO¹⁸ a una instancia del modelo y, además, un operador que permite cerrar automáticamente el observable al recibir la última respuesta (que, en una petición HTTP habitual, será la primera), finalizando el flujo asíncrono y desuscribiendo a sus observadores tras transmitir la respuesta procesada. Además, este último operador permite “aplanar” el flujo de datos: dado que se sabe que habrá una sola respuesta, es posible devolver una promesa¹⁹, lo que, dado que están estandarizadas desde ES6²⁰, facilita la codificación posterior.

```

1  public constructor(private http: HttpService) {}
2
3  public getCourseById = (id: number): Promise<Course> =>
4      lastValueFrom(this.http.get<CourseDTO>("/course/" + id.toString())
5          .pipe(map((courseDTO: CourseDTO) => new Course(courseDTO)))
6      );

```

Código 3.1: Ejemplo de petición HTTP con el cliente de Angular (dependencia inyectada) y un observable de RxJS.

¹⁸ DTO. Siglas de “objeto de transferencia de datos”, del inglés *Data Transfer Object*. Es un objeto sencillo que emplea tipos primitivos y que se utiliza para facilitar la transferencia de información entre componentes *software*.

¹⁹ Promesa. En JavaScript y TypeScript, un objeto declarado asíncrono que devolverá eventualmente un valor afirmativo (resolución) o negativo (rechazo). Es estándar en JavaScript desde 2015, por lo que existen facilidades sintácticas del lenguaje para su uso [41].

²⁰ ES6. Acrónimo de *ECMAScript 6*, estándar del lenguaje JavaScript publicado en 2015. Los estándares publicados por ECMA son los más extendidos y empleados para generar intérpretes de JavaScript [42].

Sin embargo, resulta interesante emplear el comportamiento estándar de un observable de RxJS —es decir, sin convertirlo en una promesa—, por ejemplo, al utilizar una conexión mediante *Web Sockets*. Este protocolo permite dejar una conexión abierta bidireccional entre emisor y receptor, de modo que un observable sirve como interfaz para la recepción de mensajes de un *web socket*, procesando las entradas e informando en el momento de la recepción a los suscriptores existentes, quienes pueden efectuar las acciones programadas como respuesta a la ocurrencia del citado evento asíncrono.

3.1.1.4. TypeScript

Según Microsoft, creadores e impulsores de este proyecto, TypeScript es un “lenguaje fuertemente tipado que transpila a JavaScript, brindando a los desarrolladores mejores herramientas a cualquier escala” [43]. Divulgado como código abierto bajo licencia Apache, TypeScript es un lenguaje de programación con tipado fuerte que proporciona una envoltura alrededor de JavaScript, aportando las principales ventajas del uso de tipos, como la detección de errores en tiempo de compilación como consecuencia de un mal uso de las variables según sus tipos definidos. Su uso se basa en un transpilador²¹, *tsc*, que convierte el código TypeScript en su correspondiente JavaScript.

De las tecnologías analizadas en la sección 3.1.1.1, Angular es el único *framework* de aplicaciones web para cliente que emplea TypeScript como lenguaje recomendado para su utilización durante el desarrollo. El propio *framework* está implementado con este lenguaje e incorpora *per sé* todas las herramientas necesarias para generar JavaScript automáticamente durante los procesos de construcción o ejecución de la aplicación con el fin de favorecer el uso de este lenguaje.

3.1.1.5. Node

Node.js (habitualmente denominado “Node”) es un “entorno de ejecución de JavaScript gratuito, de código abierto y multiplataforma que permite crear servidores, aplicaciones web, herramientas para línea de comandos y scripts” [44] que se basa en V8, el intérprete de JavaScript que emplean navegadores como Chrome o Edge.

Si bien la intención original del lenguaje JavaScript, sexto lenguaje más usado con una cuota de uso del 3,32 % según TIOBE [45] en junio de 2024, era la de dotar de dinamismo a las aplicaciones web mediante *scripts* ejecutados en los navegadores al cargar las páginas, sus prestaciones han aumentado y en la actualidad dispone de herramientas de forma nativa que permiten trabajar mediante

²¹ Transpilador. Tipo particular de compilador que genera código fuente en un lenguaje a partir de código fuente en otro lenguaje.

eventos asíncronos de forma sencilla, facilitando la creación de aplicaciones fundamentadas en el intercambio de información por red. La finalidad de Node es la traslación de estas ventajas a la posibilidad de ejecutar aplicaciones basadas en este lenguaje directamente como procesos en el sistema operativo para generar aplicaciones de escritorio o herramientas por línea de comandos.

Este último es el caso de Angular. Como se introduce en la [sección 3.1.1.1](#), este *framework* dota al desarrollador de una herramienta por línea de comandos que permite, entre otros procesos, ejecutar la aplicación o compilarla, para lo que se basa en herramientas asentadas en la plataforma Node. Además, por ejemplo, la capacidad para incorporar *server-side rendering* (SSR) hace uso de un pequeño proceso Node a instalar en el servidor que se encarga de la generación de las vistas renderizadas transmitidas al cliente.

Otra de las capacidades asociadas a los proyectos basados en la plataforma Node es la utilización de un gestor de dependencias instalado por defecto junto con la distribución habitual de Node, que es NPM (siglas de *Node Package Manager*). Además de un gestor de dependencias, NPM dispone de un repositorio público impulsado por GitHub con una gran cantidad de módulos reutilizables para los proyectos Node [46]. Este gestor de dependencias se basa en las declaraciones albergadas en el fichero `package.json` que existe en la raíz de los proyectos Node, instalando todas las dependencias declaradas y sus transitivas en el directorio `node_modules`. En el [código 3.2](#) se introduce un fragmento del fichero `package.json` de la aplicación web para Angular, en el que se observa la declaración de algunos atributos básicos, tales como el nombre o la versión, de algunos atajos para la ejecución de *scripts* que permiten ejecutar la aplicación o construirla y la declaración de dependencias, tanto las requeridas en tiempo de ejecución (`dependencies`) como las únicamente requeridas para el desarrollo (`devDependencies`).

```

1 {
2     "name": "vscode4teaching",
3     "version": "3.0.0",
4     "scripts": {
5         "start": "ng serve",
6         "build": "ng build",
7         // ...
8     },
9     "dependencies": {
10        "@angular/common": "^16.2.9",
11        "@angular/core": "^16.2.9",
12        "@angular/forms": "^16.2.9",
13        "@angular/router": "^16.2.9",
14        // ...
15    },
16    "devDependencies": {
17        "@angular/cli": "^16.2.6",
18        "typescript": "^5.1",
19        // ...
20    }
21 }
```

Código 3.2: Fragmento del documento `package.json` de la aplicación web Angular.

3.1.1.6. Interfaz de usuario: SCSS, Bootstrap y Chart.js

A todas las tecnologías desarrolladas durante las subsecciones anteriores, se añaden algunas otras bibliotecas y tecnologías empleadas para facilitar la generación de una interfaz de usuario visualmente atractiva, fácilmente operable e intuitiva. Para ello, se han empleado **SCSS** como preprocesador de hojas de estilo, **Bootstrap** como biblioteca para la disposición básica de los elementos y **Chart.js** como ejemplo de biblioteca para la generación de componentes visuales avanzados.

CSS (siglas en inglés de “hojas de estilo en cascada”) es el lenguaje declarativo empleado de forma estándar para la definición de los estilos que se deben aplicar sobre la definición de la estructura de una visualización web a través de una maqueta en HTML. Este estándar es mantenido por un grupo de trabajo del W3C [47], y es aplicado por todos los motores de renderizado de los navegadores más empleados.

De una forma similar a como TypeScript envuelve JavaScript, alrededor de CSS surgió **Sass** (“hojas de estilo sintácticamente increíbles”, del inglés *Syntactically Awesome Style Sheets*) [48], que introduce algunas características de las que CSS no disponía en sus versiones más antiguas, como el uso de variables, el anidamiento de selctores y reglas u operadores automáticos. Sass es de código libre, está divulgado bajo licencia MIT, y que ofrece dos sintaxis: SCSS, que emplea el mismo concepto sintáctico que CSS; o la sintaxis indentada, que elimina los caracteres de separación de ámbitos y reglas y los sustituye, respectivamente, por tabulaciones y saltos de línea. Proporciona un transpilador que permite generar el CSS compatible con los motores de renderizado.

Angular ofrece soporte a la transpilación de Sass, incorporándola al proceso de ejecución y construcción de aplicaciones de forma automática, pudiendo configurar en los proyectos si harán uso de CSS o de alguna de las dos sintaxis que propone Sass. La aplicación web de *VSCode4Teaching* define sus hojas de estilo haciendo uso de la sintaxis SCSS.

Bootstrap es un “conjunto de herramientas para *frontend* poderoso, extensible y empaquetado por características” que “está construido con Sass, utiliza un sistema de cuadrícula y aporta componentes a través de poderosas extensiones JavaScript” [49] publicado como *software* libre bajo licencia MIT. Ofrece una especificación de estilos básica para todos los elementos visuales, aportándoles una estética común, además de un conjunto de componentes funcionales JavaScript básicos, entre los que destacan las ventanas modales embebidas.

Chart.js es una biblioteca “simple y flexible” [50] para generar gráficos de diversa índole a partir de una configuración básica y una serie de datos. Específicamente, se utiliza para generar un gráfico semicircular representativo del progreso de los estudiantes al realizar un ejercicio (véase la sección 4.3.1.3).

3.1.2. Servidor

Si bien este Trabajo Fin de Grado pone el foco en la generación del nuevo *frontend* Angular (véase la sección 3.1.1), esto no es óbice para exponer a continuación las tecnologías empleadas para la generación del servidor, que es la base sobre la que funciona el conjunto del proyecto. Este servidor está implementado en **Spring Boot** (basado en el *framework* Spring), hace uso de un sistema de persistencia materializado en una **base de datos MySQL**, potencia su funcionalidad mediante bibliotecas gestionadas con **Maven** e incorpora una batería de pruebas automáticas sobre la plataforma **JUnit**.

3.1.2.1. Persistencia de la información: MySQL

VSCode4Teaching asienta su lógica de negocio sobre un modelo del dominio que introduce varias entidades interrelacionadas, tal como queda reflejado en la sección 4.2.1. Para la persistencia de la información asociada a cada entidad y sus interconexiones, se hace uso de un sistema gestor de bases de datos (SGBD) de tipo relacional. En estos sistemas, cada base de datos contiene una colección de tablas, que son estructuras bidimensionales que almacenan tuplas (registros) de información estructurada en atributos (columnas) comunes a todas, permitiéndose el establecimiento de relaciones lógicas entre los atributos de distintas tablas para dar lugar a un conjunto de datos coherente en el dominio especificado.

Concretamente, el sistema gestor empleado para la persistencia de la información en el proyecto *VSCode4Teaching* es MySQL. Este gestor es mantenido por Oracle Corporation y se divulga con licencia pública de GNU (GPL) en su versión *Community*, abierta a su uso no comercial para la comunidad [51]. Según la *Encuesta de Desarrolladores* realizada por Stack Overflow en su edición de 2023 [17], MySQL es el segundo sistema gestor de bases de datos más empleado, alcanzando una cuota de uso del 41,09 % según los más de 75 000 encuestados, viéndose superado por primera vez por PostgreSQL, su principal alternativa, también *software* libre divulgado bajo su propia licencia (PostgreSQL License, similar a la licencia MIT) [52], que alcanza una cuota de uso del 45,55 % y cuenta en la actualidad con un gran respaldo entre los desarrolladores profesionales.

3.1.2.2. Java y Spring

El servidor de *VSCode4Teaching* es una aplicación basada en el *framework* **Spring**, que hace uso de **Java** como plataforma para su implementación y ejecución.

Java

Java es “una plataforma informática de lenguaje de programación creada por Sun Microsystems en 1995” [53], actualmente parte de Oracle Corporation. Esta plataforma proporciona tres componentes principales: el propio lenguaje de programación, el entorno en tiempo de ejecución y las bibliotecas incorporadas por defecto.

El lenguaje de programación, también llamado Java, es uno de los más empleados en la actualidad, consolidado en la cuarta posición del índice TIOBE desde hace más de un año con una cuota de uso del 8,40 % (junio de 2024) [45]. Es orientado a objetos y destaca por su tipado fuerte y el soporte a las características esenciales de este paradigma: abstracción, encapsulación, jerarquía, modularización, herencia, polimorfismo y genericidad.

La ejecución de programas implementados en este lenguaje requiere como paso intermedio la generación de *bytecode*, que es un código objeto²² apto para su posterior ejecución mediante el entorno en tiempo de ejecución de Java, *Java Runtime Environment* (JRE), que basa su funcionamiento en la máquina virtual de Java (*Java Virtual Machine* o JVM). Este formato de compilación a un código intermedio posteriormente interpretado en ejecución mediante un compilador JIT (*just in time*) brinda a la plataforma Java una de sus capacidades más destacadas: la independencia del sistema operativo para la ejecución de las aplicaciones, ya que un mismo artefacto compilado puede ser ejecutado sobre cualquier JRE con independencia del sistema operativo en el que se generó.

A todas estas características se añade un tercer factor determinante: el conjunto de las bibliotecas incorporadas nativamente en Java, que proporcionan una gran cantidad de funcionalidad apta para todos los sistemas, permitiendo el acceso unificado a recursos para la gestión de la concurrencia, de las comunicaciones por red o de las interfaces gráficas de usuario, entre muchas otras.

Spring

Spring es el *framework* más empleado para la construcción de aplicaciones web basadas en la plataforma Java. Este dato está ratificado por la *Encuesta de Desarrolladores* de Stack Overflow que, en su edición de 2023 [17], afirma que el 11,95 % de los encuestados utiliza Spring, siendo la duodécima tecnología web más empleada según la anterior clasificación, que aglutina una amplia variedad de tecnologías para el lado cliente, el lado servidor y algunos CMS²³.

²² Código objeto. Es un paso intermedio producido al compilar un código fuente y que puede ser interpretado mediante programas específicos.

²³ CMS. Siglas de “sistema de gestión de contenidos” (del inglés *System Content Manager*). Habitualmente orientadas a la web, son aplicaciones que ponen a disposición de sus usuarios un conjunto de herramientas visuales que facilitan la generación de páginas y aplicaciones web con escasa o nula necesidad de implementación de código fuente.

Tal como introduce su propia información, Spring “proporciona un modelo de configuración y programación fácilmente comprensible para crear aplicaciones modernas basadas en Java” [54]. Este modelo está basado en el ecosistema alrededor del *framework*: existe una ingente cantidad de bibliotecas que se pueden añadir a las aplicaciones basadas en Spring para introducirles funcionalidades adicionales de interés. Tanto Spring como todas sus dependencias son *software* libre y se distribuyen en abierto bajo licencia Apache 2.0.

De entre todas las bibliotecas disponibles y empleadas para implementar el servidor, una de las más destacadas es **Spring Boot**, que facilita la utilización de Spring para “crear fácilmente aplicaciones autocontenidoas aptas para entornos de producción [...] que pueden simplemente ejecutarse” [55], ya que dispone de un servidor web embebido e introduce algunas dependencias de Spring por defecto para simplificar la configuración de la aplicación.

Otro proyecto del ecosistema Spring es **Spring Data**, que es un conjunto de bibliotecas que “proporciona un modelo de acceso a datos familiar, consistente y basado en Spring que facilita el acceso a bases de datos relacionales y no relacionales” [56] que, específicamente, es utilizado en el servidor para acceder a la base de datos MySQL empleada como sistema de persistencia de la información (véase la sección 3.1.2.1), relegando los detalles de la comunicación y utilización de este sistema a un ORM²⁴ que brinda la capacidad para la interacción bidireccional con la base de datos, encapsulando la necesidad de utilizar consultas y operaciones SQL para ello.

3.1.2.3. Maven

Los proyectos asentados sobre Java como plataforma tecnológica pueden hacer uso de Maven, que es un “*software* de gestión de proyectos basado en el concepto de un modelo de objetos de proyecto (*Project Object Model* o POM) centralizado que permite manejar las distintas etapas de su ciclo de vida (construcción, ejecución, verificación...) y la generación de informes y documentación” [57].

Creado por la *Apache Software Foundation* y divulgado bajo licencia Apache 2.0, Maven permite gestionar proyectos mediante la centralización de su configuración en un único fichero, el POM (habitualmente llamado `pom.xml`), que está situado en la raíz de los proyectos Maven y recoge aspectos básicos de la definición del componente, tales como su nombre, licencia o autoría, además de características del *software*, incluyendo la versión del componente, dependencias que emplea y otras configuraciones específicas para su compilación o ejecución.

²⁴ ORM. Siglas de “mapeador objeto-relacional” (del inglés *Object Relational Mapper*). Es una pieza *software* que permite interactuar con bases de datos mediante clases y objetos en el código con el fin de facilitar su uso.

Sirva como ejemplo el POM del servidor de *VSCode4Teaching*, del que se incluye un fragmento en el [código 3.3](#). Esta definición hace uso de algunos de los rasgos más importantes de la configuración mediante Maven: la declaración de la dependencia de Spring Boot como `parent` y de algunas de sus dependencias asociadas (`dependencies`), la declaración del nombre del artefacto generado y su versión, nombre y descripción y la versión de Java sobre la que se desarrolla (y, por tanto, la mínima requerida para su ejecución), entre otras configuraciones para su compilación y ejecución (`build`).

```

1 <project [...]>
2   <parent>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-parent</artifactId>
5     <version>2.7.16</version>
6     <relativePath/>
7   </parent>
8
9   <groupId>com.vscode4teaching</groupId>
10  <artifactId>vscode4teaching-server</artifactId>
11  <version>2.2.1</version>
12
13 <name>VSCode 4 Teaching</name>
14 <description>Server side of VSCode 4 Teaching extension.</description>
15
16 <properties>
17   <java.version>11</java.version>
18 </properties>
19
20 <dependencies>
21   <dependency>
22     <groupId>org.springframework.boot</groupId>
23     <artifactId>spring-boot-starter-data-jpa</artifactId>
24   </dependency>
25   <dependency>
26     <groupId>org.springframework.boot</groupId>
27     <artifactId>spring-boot-starter-web</artifactId>
28   </dependency>
29   [...]
30 </dependencies>
31
32 <build>
33   [...]
34 </build>
35 </project>
```

Código 3.3: Fragmento adaptado del fichero para la configuración de Maven aplicado en el servidor del proyecto (`pom.xml`).

3.1.2.4. JUnit

El servidor de *VSCode4Teaching* cuenta con una batería de pruebas automáticas que permite su verificación y aseguramiento de su calidad, tal como se detalla en la [sección 4.4.1](#). Estas pruebas se asientan sobre JUnit, que es una biblioteca muy extendida para la implementación y ejecución de pruebas o *tests* en proyectos basados en la plataforma Java [58]. Fue creada por Kent Beck y Erich Gamma y se distribuye como código libre con licencia Eclipse Public License.

JUnit brinda a los desarrolladores la posibilidad de implementar pruebas basadas en aserciones —esto es, comparaciones estrictas entre valores esperados y valores obtenidos tras la ejecución de las pruebas—, introduciendo soporte a la parametrización de pruebas (es decir, al uso de colecciones de distintos datos de prueba) y a los dobles, que son piezas *software* que permiten reemplazar el comportamiento que tendrán ciertos componentes relacionados y utilizados en el SUT²⁵ que quedan verificados mediante otras pruebas. Es posible ejecutar la batería de pruebas manualmente mediante Maven y, además, se integra fácilmente con sistemas de integración continua, tal como se hace en este proyecto *software* (véase la [sección 4.3.2.5](#) a este respecto).

3.1.3. Extensión para Visual Studio Code

Se introducen en esta sección las tecnologías empleadas para la construcción de la extensión de *VSCODE4Teaching* para el entorno de desarrollo integrado Visual Studio Code (véase la [sección 3.2.1.1](#) para más información sobre esta herramienta). La extensión hace uso de **Node** como plataforma subyacente, está implementada mediante el lenguaje **TypeScript**, se basa en la **Visual Studio Code Extension API** como biblioteca fundamental y utiliza **Jest** para la codificación y ejecución de las pruebas automáticas.

Node y TypeScript

Tal como se desarrolla en la [sección 3.1.1](#), este cuarto hito evolutivo del proyecto introduce una nueva aplicación web del lado cliente basada en Angular. Algunas de las tecnologías fundamentales empleadas para la implementación de este nuevo componente ya se venían utilizando con la extensión: en ambos casos se hace uso de Node como plataforma subyacente y de TypeScript como lenguaje de programación. Esta compartición reduce la curva de aprendizaje de las tecnologías empleadas para el proyecto, facilitando posteriores tareas de mantenimiento.

Visual Studio Code Extension API

Visual Studio Code es un entorno de desarrollo “extensible y personalizable” [70], siendo esta una de las características más destacadas por sus creadores. Esta versatilidad de la herramienta queda materializada en su integración con el *Marketplace*, punto para la descarga y divulgación de extensiones (véase la [sección 3.1.4.3](#) sobre este método de publicación). En su implementación, las extensiones hacen uso de la Visual Studio Code Extension API, interfaz integrada en el IDE para la interacción con sus elementos nativos propios, permitiendo numerosas características entre las que cabe destacar: implementación de nuevos comandos,

²⁵ SUT. Siglas de “sujeto bajo pruebas” (del inglés *Subject Under Test*. Respecto a una prueba automática de *software*, es la unidad del *software* que el *test* busca validar.)

modificación de la estética del editor de código, visualización de notificaciones y elementos gráficos para la interacción del usuario (tales como campos de texto o listas de selección múltiple, entre otros), mostración del estado o progreso de la extensión en la barra de actividad, introducción de vistas web personalizadas en pestañas nuevas, adición de funcionalidad para la depuración, acceso al control de las ventanas y pestañas abiertas, generación de nuevos ítems en la barra lateral y modificación de las carpetas que conforman el área de trabajo activa.

Jest

Jest es una herramienta para la implementación y posterior ejecución de *tests* en aplicaciones basadas en JavaScript que “pone el foco en la simplicidad”. Este proyecto, *software* libre divulgado bajo licencia MIT, fue originalmente creado por Facebook y es mantenido en la actualidad por la *Open JS Foundation* [59].

De una forma sencilla y sin necesidad de configuración adicional, Jest incorpora mecanismos para la generación de baterías de pruebas automáticas dando un soporte sencillo e intuitivo a las aserciones y los dobles, disponiendo de una utilidad para, mediante línea de comandos, ejecutar las pruebas y obtener informes sobre la cobertura del código y el detalle pormenorizado de las excepciones producidas en las pruebas erróneas.

3.1.4. Divulgación, despliegue y distribución

Si bien se confiere al diseño e implementación del *software* un papel preponderante dentro del ciclo de desarrollo, existen otras tareas fundamentales que las circundan y que cobran una especial importancia al finalizar cada iteración del proceso *software*: la divulgación del código fuente, especialmente reseñable en proyectos de código abierto como *VSCode4Teaching*; la distribución de las nuevas versiones publicadas al conjunto de usuarios finales y el despliegue de las actualizaciones son parte esencial del proceso *software*. Se introducen a continuación las principales tecnologías empleadas para la consecución de estas tareas, si bien a estas se suman algunas herramientas descritas en la sección 3.2, tales como GitHub Actions, el sistema empleado para la integración y entrega continuos.

3.1.4.1. Divulgación del código: GitHub

GitHub es la “plataforma empleada para almacenar, compartir y trabajar junto con otros usuarios para escribir código” [60] más extendida en la comunidad de desarrolladores de *software*, ya que alberga más de 420 millones de repositorios de código fuente creados por más de 100 millones de desarrolladores [61].

Si bien GitHub comenzó siendo un servidor remoto donde alojar repositorios de código fuente basados en el uso de *git* como sistema de control de versiones, ha venido incorporando numerosas herramientas asociadas que lo consolidan hoy como un ecosistema para no solo almacenar y divulgar proyectos *software*, sino también para trabajar sobre ellos en entornos remotos (GitHub Codespaces), facilitar la colaboración de los desarrolladores de los proyectos y gestionar la seguridad de los proyectos mediante mecanismos automatizados para la detección de vulnerabilidades, entre otras herramientas de utilidad para la gestión de proyectos.

GitHub viene siendo empleado como sistema de almacenamiento remoto del proyecto *VSCode4Teaching* desde su inicio. El presente hito evolutivo, además, propone un uso aún mayor de las ventajas que aporta GitHub mediante la utilización de GitHub Actions que, desarrollado en la sección 3.2.1.3, es la herramienta que se emplea para el nuevo sistema de integración continua del proyecto.

La sección 4.5.1 incluye más información acerca de su utilización como punto de divulgación del código fuente como código abierto y *software* libre.

3.1.4.2. Despliegue del servidor y la aplicación web: Docker

Docker es “una plataforma diseñada para ayudar a los desarrolladores a construir, compartir y ejecutar aplicaciones modernas” [62]. Alrededor de Docker existe un amplio abanico de tecnologías disponibles, entre las que cabe destacar su motor de ejecución, *Docker Engine*, que está disponible como código abierto bajo licencia Apache 2.0 y que dispone de un servidor capaz de ejecutar imágenes en contenedores y de una interfaz por línea de comandos para operar con él [63].

En este ámbito, se conoce como contenedor a una “unidad estandarizada para el desarrollo, entrega y despliegue de paquetes *software* que empaqueta el código y todas sus dependencias para ejecutar aplicaciones rápida y fiablemente” [64]. La principal ventaja derivada del uso de contenedores como entorno de ejecución de aplicaciones es que la posibilidad de dotar a cada contenedor de un entorno aislado que cumpla los requerimientos *software* necesarios para cada aplicación con independencia de la plataforma sobre la que se ejecute el motor de ejecución de Docker, proporcionando ventajas semejantes a la virtualización sin necesidades *hardware* tan altas como las que requiere este formato. La independencia de la plataforma viene dada por el uso de imágenes, que son las especificaciones sobre las condiciones *software* en las que debe ejecutarse un determinado código fuente al introducirse en un contenedor; esto es, las imágenes se especifican y generan para dar lugar a contenedores que las ejecutan. La especificación de imágenes Docker se realiza a través de los *Dockerfile*, que son los “ficheros de texto que contienen las instrucciones para generar una imagen a partir de un código fuente [...] empleando una sintaxis específicamente definida” [65].

Del ecosistema de tecnologías alrededor de Docker, también se utiliza Docker Compose en este proyecto. Docker Compose es una “herramienta que permite definir y ejecutar aplicaciones basadas en múltiples contenedores” [66], pudiendo así redactar instrucciones, declaraciones y especificaciones mediante una determinada sintaxis para orquestar el funcionamiento de distintos contenedores en un entorno compartido, tal como se requiere para ejecutar *VSCODE4Teaching* junto con una instancia de un sistema gestor de bases de datos empleado para la persistencia.

El detalle acerca del requisito **RN-4** incluye información acerca de la utilización del ecosistema Docker durante este nuevo hito evolutivo de *VSCODE4Teaching*.

3.1.4.3. Distribución: Docker Hub y Visual Studio Code Marketplace

Además de las tecnologías desarrolladas en la sección anterior, el proyecto *VSCODE4Teaching* también hace uso de Docker Hub, que es el “repositorio de imágenes más grande del mundo, incluyendo imágenes de la comunidad de desarrolladores y de proyectos de código abierto” [67]. La imagen Docker generada en el proyecto *VSCODE4Teaching*, que permite ejecutar el servidor y la aplicación web, se publica en Docker Hub.

Por otro lado, la extensión para Visual Studio Code queda divulgada a través del Visual Studio Code Marketplace, que es un repositorio público para la publicación de extensiones para este IDE disponible para la descarga e instalación de extensiones desde dentro del propio entorno, permitiendo así a los usuarios pueden buscar extensiones publicadas y descargarlas en su instancia local del editor de forma gratuita, rápida y sencilla [68].

La [sección 4.5.2](#) incluye información detallada sobre cómo se hace uso de estos repositorios en el proyecto *VSCODE4Teaching*.

3.2. Herramientas

Esta sección incluye un compendio de las herramientas empleadas durante el desarrollo del presente Trabajo Fin de Grado, clasificándolas según su ámbito de aplicación en: herramientas destinadas al producto ([sección 3.2.1](#)), que son aquellas que se emplean para modificar el código fuente o para mantenerlo y divulgarlo en tanto que proyecto *software*; y la herramienta destinada al propio proceso, **Trello**, que ha sido la empleada para la gestión, organización y jerarquización de las tareas ([sección 3.2.2](#)).

3.2.1. Herramientas para el producto

3.2.1.1. Edición de código

Los entornos de desarrollo integrados o IDE (siglas de *Integrated Development Environment*) son aplicaciones informáticas específicamente destinadas a los desarrolladores de proyectos *software*, ya que cuentan con una funcionalidad específicamente destinada al desarrollo con el fin de hacerlo más sencillo.

La composición arquitectónica del proyecto *VSCODE4Teaching*, que dispone de tres componentes heterogéneos, conduce intrínsecamente a la necesidad de utilizar varias herramientas especializadas distintas para cada uno de sus componentes.

WebStorm, de JetBrains [69], es el entorno de preferencia para ejecutar el presente hito evolutivo que, como se centra en la aplicación web Angular, requiere de un entorno especializado en el desarrollo web. Esta herramienta de JetBrains está específicamente enfocada a la plataforma web y facilita al desarrollador el trabajo con las tecnologías propias de este ámbito, como JavaScript o TypeScript y, en este caso concreto, Node y el *framework* Angular. Es *software* privativo de uso gratuito para aplicaciones educativas y de *software* libre.

La implementación de una extensión para **Visual Studio Code** hace imperativo el uso del susodicho entorno de desarrollo. Desarrollado por Microsoft y distribuido bajo licencia MIT como *software* libre [70], es el entorno necesario para la programación de extensiones, ya que brinda herramientas para la fácil ejecución y depuración de las extensiones.

Por otro lado, **IntelliJ IDEA**, también de JetBrains [71], es una herramienta muy popular para el desarrollo de aplicaciones basadas en la plataforma Java, como es el caso del servidor de *VSCODE4Teaching*. Tiene varias extensiones integradas en el propio entorno que facilitan el desarrollo de servicios web y de aplicaciones basadas en Spring. Al igual que WebStorm, es de uso gratuito cuando se destina al ámbito educativo o cuando se emplea para implementar *software* libre.

3.2.1.2. Sistema de control de versiones

Los sistemas de control de versiones son herramientas que ponen a disposición de los programadores partícipes de un proyecto la capacidad para observar y gestionar el cambio producido en el código a lo largo del tiempo, ofreciendo la posibilidad de conservar la historia de un proyecto a través de la sucesión de distintos cambios acontecidos sobre él. Además, estos sistemas incorporan capacidades para que los distintos integrantes de un proyecto puedan trabajar en paralelo, facilitando la posterior incorporación de los distintos cambios acontecidos de diversas formas.

El sistema de control de versiones más extendido en la actualidad es **git**, que es un “sistema de control de versiones distribuido, gratuito y de código abierto diseñado para manejar con velocidad y eficacia desde proyectos pequeños hasta muy grandes de forma rápida y eficiente” [72]. Distribuido bajo licencia de *software* libre GNU 2.0, git pone a disposición de sus usuarios una herramienta fácil de usar y aprender que se basa en el concepto de “ramas” de desarrollo para facilitar que distintos desarrolladores evolucionen el código, posteriormente fusionando sus cambios incorporados en forma de *commits*, que son los pequeños hitos registrados en la historia del proyecto y que hacen referencia a las variaciones realizadas por los desarrolladores en los distintos ficheros de un proyecto.

A pesar de su carácter distribuido, es habitual que los proyectos git tengan como punto de referencia un servidor remoto al que todos los integrantes del proyecto trasladan sus cambios particulares mediante una operación de subida (*push*) y del que todos sincronizan su versión local de la historia a partir de los cambios producidos por otros a través de una operación de actualización (*fetch*) y descarga (*pull*) de los cambios. En el caso de este proyecto, el servidor empleado es GitHub, detallado en la [sección 3.1.4.1](#).

3.2.1.3. CI/CD: GitHub Actions

CI/CD es una abreviatura formada a partir de tres conceptos: integración continua (del inglés *Continuous Integration*), entrega continua (del inglés *Continuous Delivery*) y despliegue continuo (del inglés *Continuous Deployment*). Forma parte de la filosofía DevOps, que es un “conjunto de prácticas, herramientas y filosofía cultural que busca automatizar e integrar los procesos que comparten los equipos de desarrollo de *software* y de operaciones” [73]. A esta filosofía, CI/CD contribuye proponiendo la automatización de los procesos de construcción, validación y despliegue del *software*, promoviendo la necesidad de realizar integraciones por parte del equipo de desarrollo de forma iterativa y ágil, evitando paralizar el ciclo de desarrollo para realizar estas operaciones [74, 75].

VSCode4Teaching utiliza **GitHub Actions**, una herramienta de GitHub que “facilita la automatización de todos los flujos de trabajo del *software*, [...] permitiendo construir, validar y desplegar el código desde GitHub” [76].

Anteriormente, el proyecto venía utilizando Travis CI como plataforma para ejecutar algunas tareas de la esfera propia de la integración continua, como las ejecución de las baterías de pruebas automáticas del servidor y la extensión y el despliegue de las nuevas versiones del servidor. Este Trabajo Fin de Grado establece en su sexto objetivo (véase el [capítulo 2](#)) la necesidad de mejorar la calidad del proyecto *software* y de, en consonancia, automatizar más tareas relativas a la entrega y despliegue continuos. Como GitHub es la tecnología que se emplea en el proyecto para la conservación y divulgación del código fuente ([sección 3.1.4.1](#)), se

ha decidido cambiar de sistema para CI/CD en favor de GitHub Actions, aprovechando, además, para aumentar su alcance. Esta migración queda materializada a través del requisito no funcional por el que se han implementado los nuevos flujos de trabajo de GitHub Actions, el requisito RN-5 (véase la [sección 4.3.2.5](#)).

3.2.2. Organización del proceso *software*: Trello

Este Trabajo Fin de Grado se ha realizado según un proceso *software* basado en la ejecución de iteraciones reducidas que dan lugar a pequeños incrementos cuya agregación permite alcanzar la nueva evolución pretendida para el proyecto *VSCCode4Teaching*. En particular, para la gestión de las tareas, se han tomado como referencia algunas de las técnicas y buenas prácticas que proponen los *frameworks* ágiles Kanban y Scrum, desembocando intrínsecamente en la creación y utilización de un tablero.

Este propósito se ha materializado en la utilización de **Trello**, que es una “herramienta visual que permite a los equipos gestionar cualquier tipo de proyecto y flujo de trabajo, así como supervisar tareas” [77]. Esta herramienta permite generar tableros personalizados, pudiendo incluir en cada uno tantas columnas y tarjetas como sea necesario para adecuarlos particularmente a las necesidades de cada proyecto informático.

Trello dota a cada tarjeta dispuesta en un tablero de un sinfín de funcionalidades: además de poder escribir en cada una de ellas un título y una descripción con texto enriquecido, es posible asociarlas ficheros adjuntos, enlaces de interés, listas de tareas anidadas o subyacentes, vínculos a ramas o *commits* de repositorios en GitHub, conexiones con otras tarjetas, clasificación mediante etiquetas de color, comentarios y asignación de autores y fechas de inicio y finalización, entre otras características. La herramienta es intuitiva y visual, lo que facilita la gestión del trabajo realizado, en progreso y pendiente de iniciar de forma rápida y ágil.

En particular, el tablero empleado para la gestión de este proceso toma de Kanban la idea de las tres columnas principales dispuestas: *to do*, que recoge las tareas pendientes de hacer en forma de pila —es decir, incluye en la parte más alta de la columna la siguiente tarea que será realizada—; *doing*, que alberga la tarea que está siendo ejecutada; y *done*, que agrupa en forma de cola las tareas ya ejecutadas —esto es, muestra en la parte superior la tarea completada más recientemente—. Por otro lado, se copia de Scrum la idea de disponer de un *product backlog*, que incluye en la primera columna todas las tareas que quedan pendientes de realizar sobre la totalidad del proyecto en forma de pila, extrayendo paulatinamente de él las tareas que se colocan en el *sprint backlog*, que recoge las que se cubrirán en la iteración en curso y que, en este caso, coincide con la columna de tareas pendientes de realización.

La figura 3.1 incluye una instantánea del tablero empleado que permite observar las tarjetas, su colocación en las columnas descritas y la utilización de los elementos visuales de Trello para dar lugar a una visualización colorida e intuitiva que facilita percibir a golpe de vista rasgos básicos del tablero, tales como la cantidad de trabajo pendiente, cuál es la tarea en ejecución o el estado de progreso del proyecto.

Conjugando el modelo de columnas con la visualización incluida en la citada figura, también cabe reseñar la facilidad para la utilización del tablero, ya que todas las tareas fluyen de izquierda a derecha: cuando la tarea en ejecución es finalizada (*doing*), se arrastra a la primera posición de la columna a su derecha (*done*), trasladando la primera de las pendientes de hacer (*to do*) al *doing* para iniciarla. Además, al finalizar cada iteración, se recogen nuevas tarjetas del *product backlog* para fragmentarlas en caso de ser necesario e incluirlas en la columna de tareas pendientes de hacer, ordenándolas por prioridad descendente. Esta sencillez en el uso del método organizativo es crucial, ya que, aunque requiere un esfuerzo inicial mayor para la configuración de la situación inicial del tablero, posteriormente reduce significativamente el esfuerzo y tiempo dedicados a las labores de gestión y planificación de las tareas.

La sección 3.3 describe en detalle el proceso *software*, desarrollando con mayor detalle el proceso en su integridad y situando en él la necesidad del tablero de gestión de tareas.

3.3. Metodología: proceso *software*

La historia de la ingeniería del *software*, aunque corta en el tiempo —ya que surgió hace poco más de medio siglo—, es muy prolífica, y ha dado lugar a una gran lista de técnicas, herramientas y métodos organizativos empleados por los equipos para la correcta organización de los proyectos *software*, favoreciendo la adecuada fragmentación de las tareas, su jerarquización y su planificación temporal para alcanzar un funcionamiento óptimo.

En el año 1986, Barry Boehm publicó un artículo en el que sentó las bases del **modelo en espiral** [78], que, de forma resumida, propone que el proceso *software* debe ser iterativo e incremental. Iterativo porque deben ejecutarse las mismas fases tantas veces como sea necesario (y no solo una, como proponía el método en cascada), e incremental porque cada iteración produce un nuevo aumento del *software* construido.

Aunque sienta sus bases en trabajos anteriores, en 1995, Grady Booch propuso el enfoque realizado en Rational (perteneciente posteriormente a IBM) de un proceso *software* basado en componentes, dirigido por casos de uso, centrado en la arquitectura y con un carácter iterativo e incremental. El posteriormente deno-

3.3. Metodología: proceso *software*

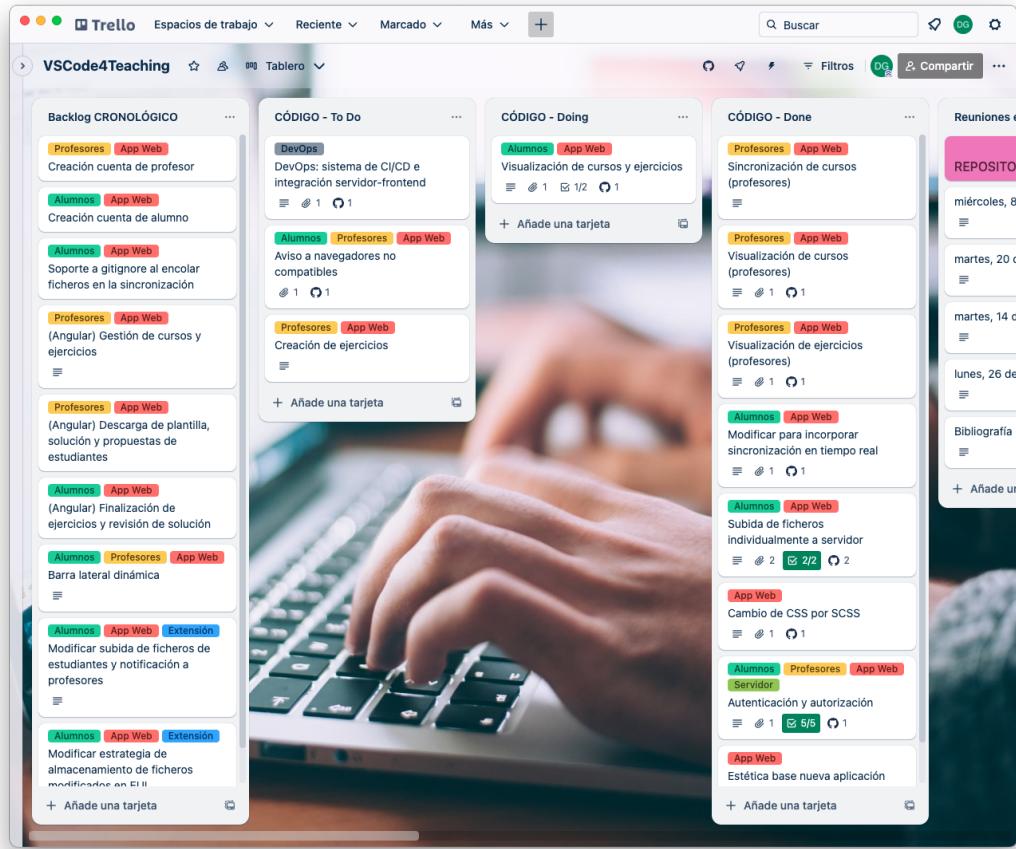


Figura 3.1: Vista del tablero Trello empleado para la gestión del proceso *software*.

minado **proceso unificado de desarrollo** tras la publicación realizada en 1999 por Booch, Jacobson y Rumbaugh [79], conocido por sus siglas RUP (del inglés *Rational Unified Process*), es un proceso que pone en el centro de su ejecución los casos de uso, que sirven como enlace entre los requisitos y el diseño e implementación; y la arquitectura, que es primordial para la posterior construcción y evolución del sistema desarrollado. Hace un uso intensivo del lenguaje unificado de modelado (UML, por sus siglas en inglés *Unified Model Language*) que, surgido en la misma corporación y divulgado por los mismos autores, establece una sintaxis gráfica para la expresión del funcionamiento y organización del *software* ampliamente consolidada en esta ingeniería [80].

También en 1999, Kent Beck publicó un libro acerca de un nuevo proceso *software*: **eXtreme Programming** (XP) [81]. Propone una modificación del enfoque, haciendo mayor énfasis en la adaptabilidad frente al cambio que en la previsibilidad, considerando que los cambios sobre los requisitos son parte intrínseca de la naturaleza de los proyectos. En consonancia con este principio,

XP propone nuevas técnicas para la ingeniería del *software*, como la programación por pares (*pair programming*), las buenas prácticas para la refactorización del código, el desarrollo guiado por las pruebas (TDD) o las historias de usuario, cuyo propósito es centrar cada requisito en el valor que aportará al usuario final.

Si bien los tres procesos desarrollados anteriormente sientan las bases de la ingeniería del *software* tal como es en la actualidad, en la que la iteratividad e incrementalidad son los dos principios básicos que rigen todos los procesos *software*, estos fueron opacados por la publicación del *Manifiesto Ágil* en 2001 [82], que afirma que los procesos *software* deben sentar sus bases en cuatro pilares: la preponderancia de individuos e interacciones sobre procesos y herramientas, el *software* funcional sobre la documentación extensiva, la colaboración con el cliente sobre la negociación contractual y la respuesta frente al cambio sobre la planificación exhaustiva. Dentro de estos principios encajan múltiples procesos de anterior publicación, como Scrum (1995), Kanban (década de 1940) o XP, cuyas buenas prácticas gozan de una gran popularidad en la actualidad.

En general, y en esta ingeniería en particular, la historia es una gran fuente de aprendizaje. La ingeniería del *software* ha aportado durante su aún breve historia una ingente cantidad de técnicas para facilitar la ejecución del proceso *software* en todas sus etapas: la ingeniería de requisitos, el diseño y análisis, la implementación, la validación del *software* y la entrega y el despliegue.

La mayoría de los procesos *software* están ideados para su utilización en equipos de desarrollo, por lo que su aplicación directa en proyectos con características tan específicas como los Trabajos Fin de Grado resulta ineficaz. Sin embargo, son muchas las buenas prácticas que se pueden tomar del histórico de conjuntos de técnicas y herramientas como ayuda y soporte para generar un nuevo proceso *ad hoc* para el contexto concreto del presente Trabajo.

Tal como propone el modelo en espiral desde hace cuarenta años, el proceso *software* de este Trabajo Fin de Grado se ha realizado de forma iterativa e incremental, buscando la completitud de cada uno de los requisitos antes de seguir avanzando: en lugar de extraerlos para diseñarlos e implementarlos todos, se han extraído, diseñado e implementado cada uno por separado, dando lugar a la ejecución reiterada de pequeñas iteraciones completas conducentes hacia pequeños incrementos hasta alcanzar la evolución deseada.

Por tanto, aunque durante la ejecución de este nuevo hito evolutivo de *VSCo-de4Teaching* no se ha seguido a rajatabla ningún proceso *software* consolidado, son numerosas las prácticas empleadas que se han extraído de los procesos más extendidos. Así, el proceso es iterativo e incremental, tal como propone el modelo en espiral, alcanzando un total de trece iteraciones; cuida la arquitectura y la sitúa en el centro, como propone RUP, haciendo uso de UML como lenguaje de modelado; y emplea el formato de historias de usuario para la especificación de los requisitos funcionales, inspirándose en XP.

Tal como queda reflejado en la [sección 3.2.2](#), que trata acerca de la herramienta para la planificación, durante este Trabajo se hace uso de un tablero inspirado en Kanban para la organización visual de las tareas, separándolas en tres columnas según su estado temporal: pendientes de hacer, en proceso y finalizadas. También inspirado en Kanban, se establece un límite al trabajo en progreso (WIP, del inglés *Work In Progress*), de modo que no se realiza más de un requisito en paralelo. El tablero empleado toma de Scrum la idea de añadir una columna adicional para albergar una pila de puntos restantes para completar los requisitos conocidos del producto, el *product backlog*, del que se van extrayendo tarjetas que, en este caso, representan historias de usuario (XP) que quedan fraccionadas e introducidas en otra pila específica para la actual iteración, coincidente en este caso con la columna de tareas pendientes de iniciar.

4

Descripción informática

Se recoge en esta sección la información detallada acerca del desarrollo ejecutado para la materialización en el proyecto *VSCode4Teaching* de los objetivos propuestos en el [capítulo 2](#). Se introduce siguiendo el orden propio de los procesos *software*: la [sección 4.1](#) introduce la enumeración de los requisitos derivados de los objetivos, la [sección 4.2](#) versa sobre el diseño y arquitectura del proyecto y las modificaciones realizadas a este respecto, la [sección 4.3](#) desarrolla pormenorizadamente la consecución de cada requisito establecido, la [sección 4.4](#) detalla los mecanismos para la garantía de la calidad del *software* y la [sección 4.5](#) abarca la distribución del código fuente y de los artefactos del proyecto.

4.1. Extracción de requisitos

Tomando como base el objetivo principal del proyecto (véase el [capítulo 2](#)), y los distintos objetivos articulados en torno a él, el primer paso para acometer la descripción informática del trabajo es la enumeración de sus requisitos, reflejando breve, clara y concisamente las tareas que se ejecutarán.

Este Trabajo Fin de Grado recoge dos tipos de requisitos: funcionales y no funcionales. Se asocia a cada requisito un identificador de la forma RX-Y, donde X designa el tipo de requisito (F para funcionales y N para no funcionales) e Y es un valor numérico correlativo. El uso de estos identificadores quedará extendido a todo el presente documento.

4.1.1. Requisitos funcionales

Se introducen a continuación los requisitos funcionales, que son aquellos que tienen como fin proporcionar mecanismos que permitan a los usuarios realizar procesos de negocio que les aporten valor; esto es, los conducentes a la incorporación de nueva funcionalidad en la aplicación.

Se recogen utilizando el formato típico de las “historias de usuario”, una práctica extraída de eXtreme Programming (XP) [83], redactándolos en un formato común preestablecido que es claro, conciso y fácilmente comprensible que sitúa en un papel preponderante las necesidades de los usuarios finales. Los requisitos son:

- **RF-1:** como usuario registrado en *VSCode4Teaching*, quiero poder acceder a mis cursos y ejercicios. Para ello, quiero poder iniciar sesión con mis credenciales para autenticarme y obtener acceso a mi información (**RF-1.1**) y, además, quiero poder visualizar mis cursos impartidos o matriculados y mis ejercicios (**RF-1.2**).
- **RF-2:** como profesor, quiero poder crear ejercicios nuevos en los cursos que imparto proporcionando una plantilla inicial y, opcionalmente, una propuesta de solución para que los alumnos puedan resolverlos.
- **RF-3:** como profesor, quiero poder visualizar en tiempo real información básica sobre el progreso de los estudiantes al realizar los ejercicios que he propuesto para poder conocer en todo momento la situación en que se encuentra cada ejercicio y cada estudiante.
- **RF-4:** como profesor, quiero poder descargar cuando lo desee los ficheros que compongan las propuestas de resolución de los ejercicios de los estudiantes para poder almacenarlas y visualizarlas, así como conocer de qué estudiante es cada una de ellas para poder asociar las distintas entregas con sus autores.
- **RF-5:** como profesor, quiero poder configurar los parámetros que determinan si la solución de los ejercicios es pública y si los estudiantes pueden editar sus propuestas tras descargarla para poder tener el control completo sobre cuándo se divulga la solución propuesta.
- **RF-6:** como profesor, quiero poder gestionar qué usuarios forman parte de mi curso para poder matricular o revocar el acceso de los estudiantes y para poder incorporar o eliminar a otros profesores.
- **RF-7:** como profesor, quiero tener la capacidad de compartir un código único de cada uno de mis cursos para poder dárselo a los estudiantes y que, de ese modo, se automatriculen en mi curso.

- **RF-8:** como alumno, quiero poder inscribirme en un cursos nuevos utilizando códigos proporcionados por los profesores para poder realizar sus ejercicios.
- **RF-9:** como alumno, quiero poder seleccionar un directorio de mi ordenador y usarlo para poder visualizar los ejercicios de un curso y sus ficheros.
- **RF-10:** como alumno, quiero poder iniciar un ejercicio nuevo y descargar su plantilla o reanudar un ejercicio y descargar mi progreso para poder realizarlo.
- **RF-11:** como alumno, quiero poder sincronizar automáticamente los ejercicios cuando los modifco y revisar el progreso de la sincronización para asegurarme de que se está almacenando correctamente.
- **RF-12:** como alumno, quiero poder marcar un ejercicio como finalizado para indicárselo al profesor y, de ese modo, consolidar el estado final de mi propuesta, ya que no podré volver a editarla.

4.1.2. Requisitos no funcionales

Se recogen a continuación los requisitos no funcionales, que son aquellos que tienen como fin la mejora de los atributos de calidad del proyecto y de su *software*, atendiendo a necesidades existentes alrededor de la funcionalidad que aportan valor al *software* en sí mismo y, en consecuencia, a los usuarios. Estos son:

- **RN-1:** mostrar un aviso en los navegadores no compatibles con la *File System Access API*.
- **RN-2:** generar un aspecto visual común para la aplicación web de *VSCode4Teaching* que sea intuitivo, fácil de comprender y coherente con la extensión para Visual Studio Code preeexistente.
- **RN-3:** mejorar el sistema de autenticación existente para encriptar el *token JWT*²⁶ mediante cifrado simétrico.
- **RN-4:** adaptar la generación de imágenes Docker del servidor para incorporar la aplicación web.
- **RN-5:** migrar el sistema de integración continua de Travis CI a GitHub Actions, adaptarlo a la nueva composición del proyecto y ampliar su alcance y tareas realizadas.

²⁶ JWT. Véase la sección 4.3.2.3.

4.2. Diseño y arquitectura de la aplicación

4.2.1. Dominio de *VSCode4Teaching*

VSCode4Teaching es, tal como constata la sección 1.2, una aplicación orientada al ámbito educativo, por lo que su dominio es un subconjunto del contexto educativo habitual. En consecuencia, aparecen en el dominio dos roles de usuarios, estudiantes y docentes, además de los elementos reales académicos más destacados: los cursos, compuestos por ejercicios que los estudiantes realizan dando lugar a propuestas de resolución.

La figura 4.1 refleja las distintas entidades del dominio de *VSCode4Teaching* y las relaciones que existen entre ellas mediante notación UML. Este diagrama representa gráficamente y en alto nivel qué realidades de la educación intervienen en el proyecto y cómo se relacionan entre sí, sirviendo como base para dar lugar al modelo del dominio, que es una traducción del dominio a un paquete de clases interconectadas similar en todos los componentes y sobre el que se fundamentan cada uno de ellos, actuando como eje vertebrador de cada componente específico y del proyecto en su conjunto.

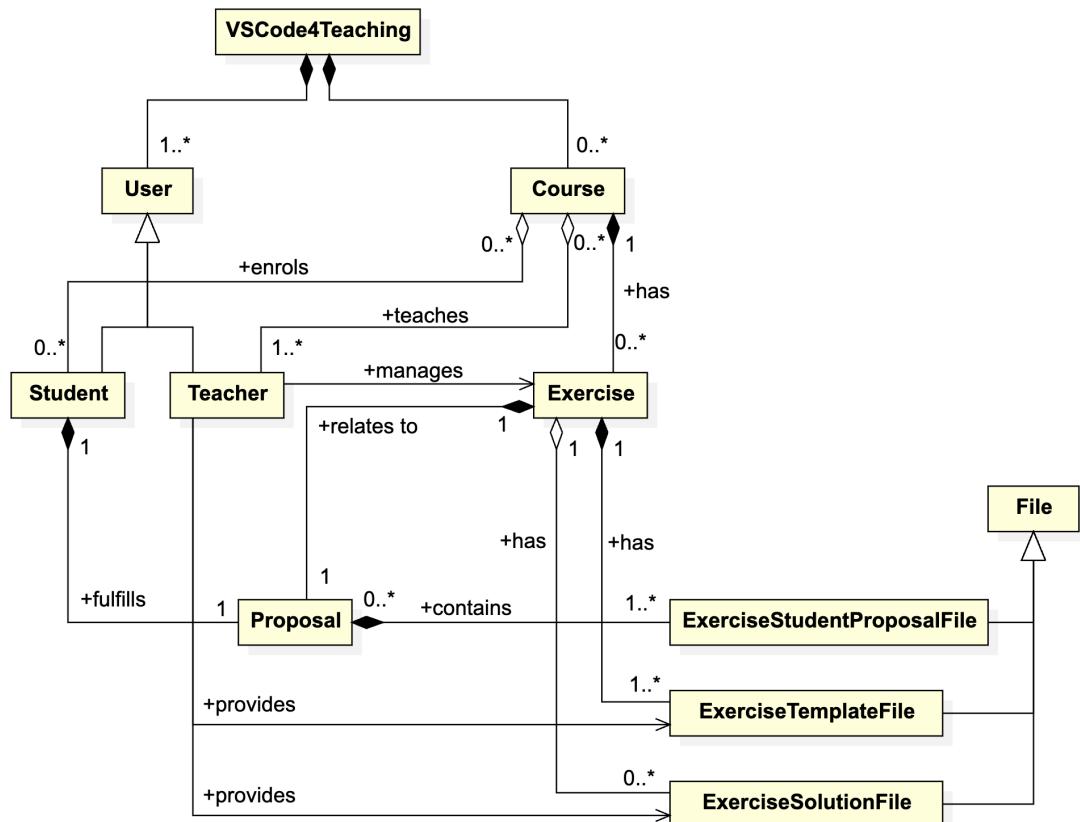


Figura 4.1: Dominio de *VSCode4Teaching* mediante notación UML.

4.2.2. Arquitectura del proyecto: organización de componentes

Previamente al inicio del actual hito evolutivo, el proyecto *VSCode4Teaching* contaba con tres componentes dispuestos en una arquitectura cliente-servidor prototípica, en la que el servidor, que basa la persistencia de los datos en una única instancia de un sistema gestor de bases de datos, responde las peticiones recibidas por HTTP²⁷ a través de su API REST^{28²⁹} desde dos clientes: la extensión para Visual Studio Code, que sirve para la ejecución de la práctica totalidad de los procesos de negocio; y la aplicación web Angular, que sirve como apoyo para la ejecución de algunos procesos, tales como el sistema de registro de docentes por invitación o la página de ayuda personalizada para estudiantes.

La fisonomía que presenta el proyecto tras la ejecución de las modificaciones necesarias queda reflejada en la figura 4.2. Esta disposición es la misma que ya tenía el proyecto, ya que la nueva aplicación web Angular sustituye a la anterior y, a pesar de que permite ejecutar una ingente cantidad de procesos de negocio, su situación en la arquitectura del proyecto no varía: la única modificación que comporta es que su comunicación con el servidor ya no solo se produce a través de la API REST sino que, además, al igual que la extensión, hace uso del *Web Socket*³⁰ gestionado por el servidor.

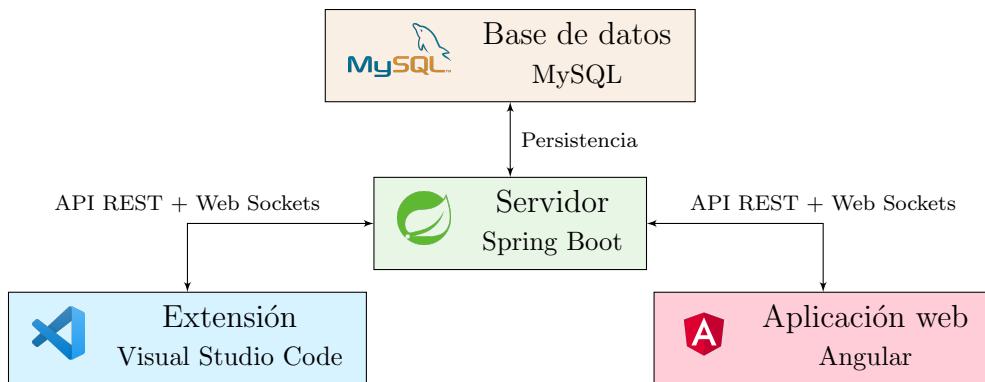


Figura 4.2: Disposición arquitectónica de los componentes de *VSCode4Teaching*.

²⁷ HTTP. Siglas de “protocolo de transferencia de hipertexto” (del inglés *HyperText Transfer Protocol*).

²⁸ REST. Siglas de “transferencia de estado representacional” (del inglés *Representational State Transfer*). Es un convenio acerca del estilo empleado para la representación de la información para su transmisión entre sistemas.

²⁹ API REST. Es una interfaz formada por varios puntos de entrada o *endpoints* que se basa en el estilo REST para el intercambio de información entre un servidor y sus clientes.

³⁰ *Web Socket*. Canal de comunicación entre dos extremos que permanece abierto y permite transmitir múltiples mensajes entre ambos extremos tanto tiempo como lo deseen, siendo habitualmente empleado para la implementación de comportamientos en tiempo real.

4.2.3. Diseño interno de cada componente

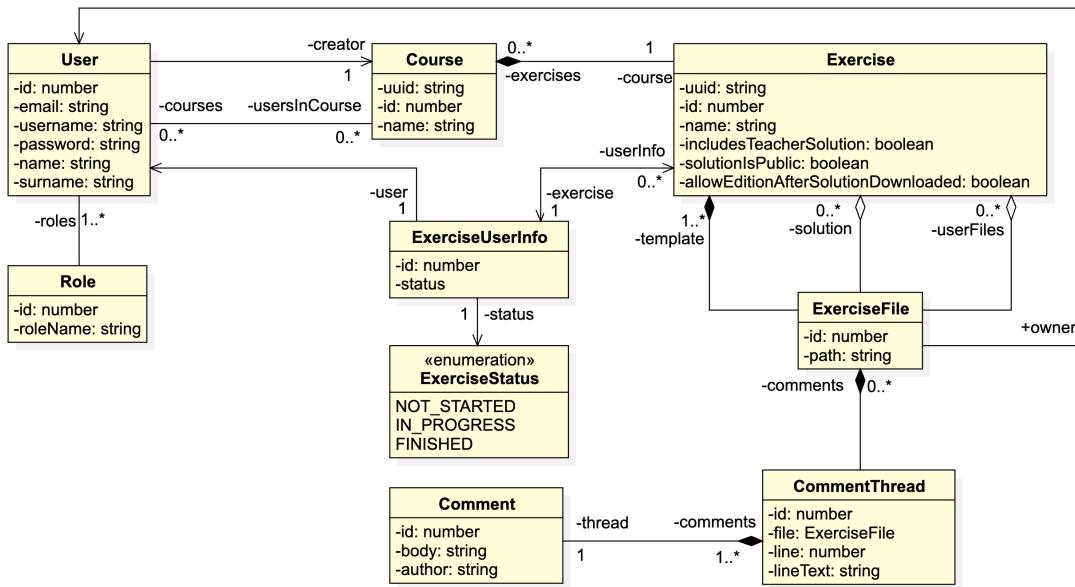


Figura 4.3: Diagrama UML de las clases del modelo del dominio de *VSCode4Teaching*.

Mientras que en la sección 4.2.2 se introducen los componentes que forman el proyecto *VSCode4Teaching* y su disposición arquitectónica, cada una de estas partes está internamente conformada por un conjunto de clases e interfaces agrupadas en paquetes según sus distintas responsabilidades. La divergencia en las tecnologías empleadas para la implementación de cada componente conlleva la particularización y adaptación de determinados patrones organizacionales o arquitectónicos para la ubicación e interrelación de sus piezas *software* de forma adecuada al contexto de cada uno.

Esta heterogeneidad, sin embargo, no es impedimento para que los tres componentes tomen como inspiración para sus arquitecturas internas el patrón Modelo-Vista-Controlador (MVC) [85], que establece la división de las piezas *software* en tres capas fundamentales: modelo, que es la agrupación de las clases y sus interrelaciones extrapoladas del dominio, tal como se desarrolla en la sección 4.2.1; vista, que se encarga del intercambio bidireccional de información con sus consumidores, sean los usuarios u otros componentes *software*; y controlador, que actúa como mediador entre el modelo y la vista y define intrínsecamente los procesos de negocio disponibles.

Este punto común existente entre los tres componentes se vislumbra en el modelo que, generado a partir del dominio, es común a todos los componentes, diferenciándose únicamente en el lenguaje de programación empleado para su codificación. La interpretación del dominio conduce a la generación de múltiples clases *software* con un estado definido por el conjunto de los atributos que tie-

nen, pudiendo ser primitivos, que son aquellos que permiten reflejar información independiente y específica de cada instancia concreta, o de los tipos de las demás clases del modelo, mecanismo empleado para traducir en *software* las interrelaciones existentes entre las distintas realidades del dominio.

La figura 4.3 es un diagrama de clases UML que representa el modelo del dominio generado para el servidor, aunque es igualmente válido como representación del modelo empleado en la extensión y en la aplicación web. Este diagrama incluye todas las clases necesarias para trasladar las realidades del dominio introducidas en la sección 4.2.1, y sirve como base fundamental sobre la que se erigen las demás piezas *software* de cada uno de los componentes implementados.

Las siguientes subsecciones introducen cuestiones específicas sobre el diseño de cada componente: de la nueva aplicación web (sección 4.2.3.1), del servidor (sección 4.2.3.2) y de la extensión (sección 4.2.3.3).

4.2.3.1. Aplicación web

Este nuevo hito evolutivo del proyecto *VSCode4Teaching* introduce como principal novedad en su arquitectura la implementación de una aplicación web basada en Angular que emplea las tecnologías desarrolladas en la sección 3.1.1.

La jerarquía de ficheros que componen la aplicación Angular incluye algunos archivos y carpetas indispensables para el correcto funcionamiento del proyecto:

- Directorio **src**. Contiene todo el código fuente propio de la aplicación específica. Alberga algunos ficheros básicos necesarios para el *framework*, como **index.html** y **main.ts**, que actúan como punto de entrada de la aplicación Angular.
- Fichero **angular.json**. Es la configuración básica del proyecto Angular. Contiene, entre otros, configuraciones relativas a su ejecución y compilación, tales como la definición de las hojas de estilo SCSS y las bibliotecas JavaScript disponibles globalmente en la aplicación, así como la declaración de la ubicación de la configuración del transpilador o del *proxy* inverso³¹, empleado para poder remitir peticiones a un servidor alojado en el mismo *host* que la aplicación.
- Fichero **package.json**. Es el manifiesto del proyecto Node, e incluye información relativa al propio componente: nombre, licencia, *scripts* ejecutables, las dependencias empleadas y sus versiones.

³¹ *Proxy* inverso. Mientras que un *proxy* convencional es un agente intermediario en las conexiones HTTP que permite, entre otras características, administrar el ancho de banda o aprovechar la caché para mejorar la velocidad de las conexiones, un *proxy* inverso recibe peticiones como si se tratase del servidor destinatario y las redirige hacia otros agentes *software*.

- Fichero `tsconfig.json`. Contiene la configuración propia del transpilador de TypeScript a JavaScript y es esencial para el correcto funcionamiento del proyecto, ya que TypeScript, lenguaje empleado para su codificación, no es directamente compatible con los navegadores web, por lo que el proceso de transpilación es necesario para dar lugar a una aplicación web operativa.

En este componente se hace uso de un único módulo, `app`, incluido dentro de `src` y que se organiza internamente según una adaptación del patrón MVC, pudiéndose distinguir las siguientes capas:

- **Modelo.** Agrupado dentro del directorio `model`, contiene las clases necesarias para incluir las realidades del dominio, sus correspondientes interfaces asociadas como implementación del patrón DTO³² (alojadas en `rest-api`) y las clases empleadas para modelar la sincronización de ficheros con el servidor en tiempo real (incluidas en `file-system`, véanse el [sección 4.3.1.11](#) y el [anexo A](#) para más información sobre su uso).
- **Controlador.** Es la capa que actúa como intermediaria entre el modelo y la vista que, en este caso, se corresponde con la capa encargada de pedir al servidor los datos que se convertirán en instancias del modelo para su visualización por la vista y, además, la interpretación de la información especificada por el usuario a través de la vista como instancias del modelo para su envío al servidor. Como consecuencia, esta capa queda articulada por los servicios, en la que recae la traslación al *software* de la lógica de negocio. Se implementan múltiples servicios, tales como los empleados para la autenticación de los usuarios (directorio `auth`), para el manejo del sistema local de ficheros (`file-system`) y para la interacción con el servidor a través de su API REST (`rest-api`) y de su *Web Socket* (`ws`).
- **Vista.** Está conformada por el conjunto de los componentes implementados (directorio `components`), que se organizan internamente de forma arborescente según su uso, destacando una primera subdivisión entre componentes de la parte pública de la aplicación (en `public`), tales como la página de inicio o la destinada a la autenticación; componentes de la parte privada de la aplicación (en `private`), subdivididos según los roles; los componentes básicos para conformar la visualización web (en `layout`), tales como la cabecera; y algunos componentes auxiliares multipropósito, localizados en `helpers`, tales como una barra de progreso o una visualización del tiempo transcurrido desde una marca temporal hasta la actualidad. De este modo, cada componente queda articulado por, al menos, un fichero HTML que determina su estructura visual para la interfaz de usuario y un fichero en TypeScript que contiene su lógica asociada.

³² DTO. Siglas de “objeto de transferencia de datos” (del inglés *Data Transfer Object*). Es una adaptación de una clase empleada específicamente para enviar peticiones y para interpretar el contenido de las respuestas empleado para la comunicación con otros componentes.

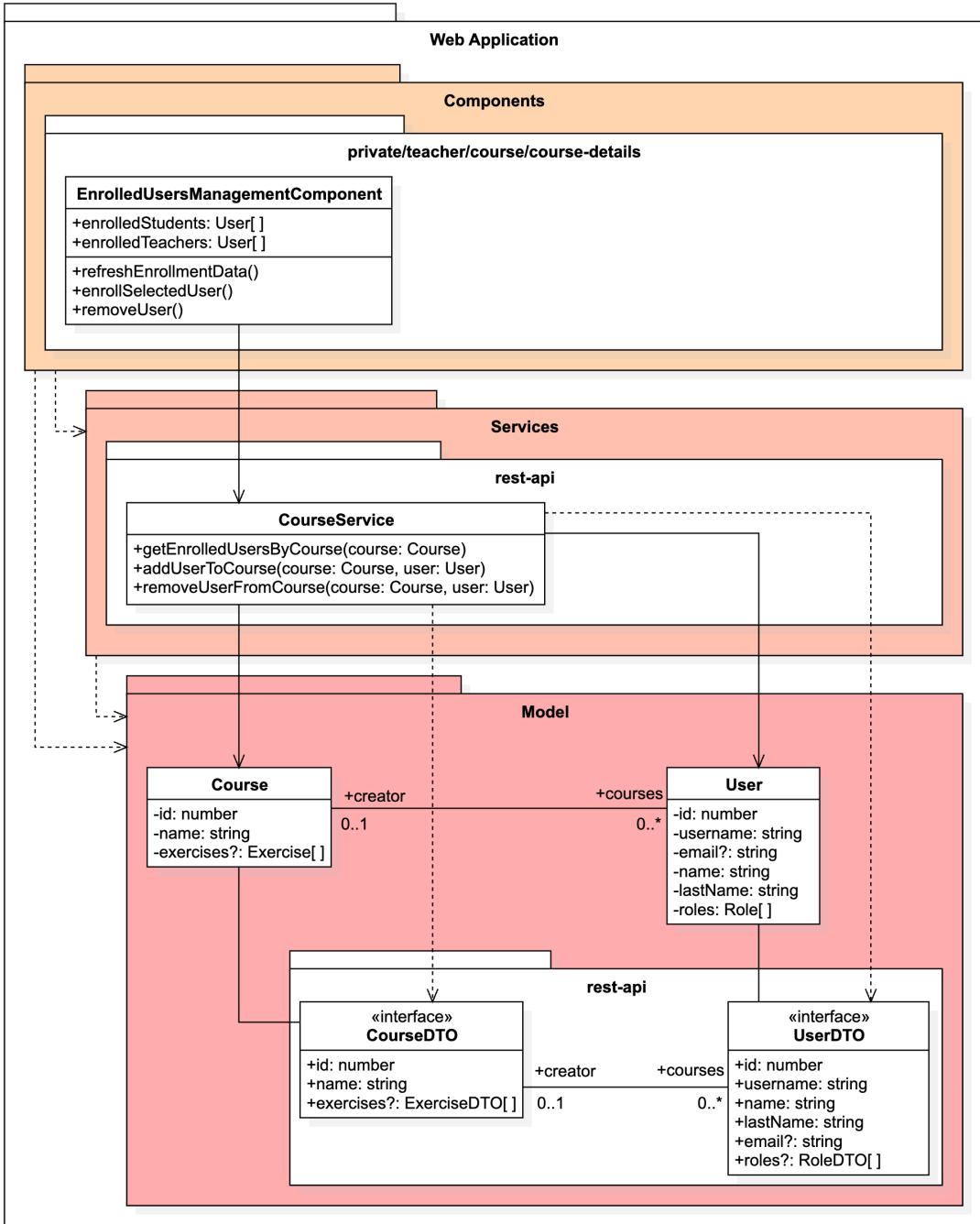


Figura 4.4: Diagrama de clases de las piezas *software* involucradas en el proceso de negocio para la gestión de estudiantes matriculados en un curso.

Supóngase la operativa por la que los profesores pueden visualizar los alumnos matriculados en uno de sus cursos impartidos y añadir un nuevo estudiante o revocar una inscripción (véase la sección 4.3.1.6). La figura 4.4 muestra un diagrama de clases que incluye el componente empleado para mostrar al usuario los elementos gráficos necesarios para la gestión de los estudiantes inscritos (`EnrolledUsersManagementComponent`), que dispone de métodos para solicitar al servidor los usuarios matriculados en un curso, dar de alta un nuevo estudiante y revocar una inscripción existente. Estos métodos hacen uso de la capa inmediatamente inferior, la de servicios, que incluye en `CourseService` los métodos encargados de la gestión de las peticiones HTTP que se requiere enviar y de la interpretación de las respuestas recibidas para, haciendo uso del modelo del dominio y de sus correspondientes interfaces DTO, generar instancias de las clases del modelo que el componente empleará para la generación de los elementos gráficos necesarios, periciéndose así a través de un ejemplo extrapolable al conjunto de la aplicación cómo funciona la arquitectura ejecutada, en la que todas las dependencias “fluyen” desde la capa superior hacia las capas inferiores y no existen conexiones entre capas no adyacentes ni realizadas en sentido ascendente, favoreciendo un diseño con alta cohesión y bajo acoplamiento.

4.2.3.2. Servidor

Tal como se introduce en la sección 3.1.2, el servidor es una aplicación basada en Spring Boot. Su disposición arquitectónica interna emplea una adaptación del patrón Modelo-Vista-Controlador al contexto en el que se utiliza, disponiendo las clases y paquetes que lo componen en tres capas:

- **Modelo.** Comprende las clases empleadas para la representación del dominio y sus conexiones, siendo aprovechadas por Spring Data como entidades para generar el esquema de base de datos y, además, como DTO con el sistema de persistencia. Esta capa, recogida en el paquete `model`, contiene, además, los repositorios empleados como DAO³³.
- **Vista.** Como este componente expone una API REST para la interacción con otros componentes en lugar de una interfaz gráfica de usuario, esta capa está comprendida por el conjunto de los *endpoints* que otros agentes *software* pueden emplear para comunicarse con el servidor. Estos puntos de acceso quedan introducidos a través de los controladores REST específicos de Spring, por lo que el paquete `controllers` es el que conforma la vista de este componente.

³³ DAO. Siglas de “objeto de acceso a datos” (del inglés *Data Access Object*). Es una instancia de una clase que dispone de métodos para la interacción con sistemas de persistencia. En este caso concreto, Spring Data define el aspecto y métodos básicos de los repositorios.

- Controlador. Es la capa intermedia entre el modelo y la vista, y contiene intrínsecamente la definición e implementación de la lógica asociada a los procesos de negocio disponibles en *VSCode4Teaching*. Está materializada en la carpeta **services**, que contiene los servicios empleados por la vista para la consulta y modificación de los sistemas de persistencia disponibles —sistema de ficheros y base de datos— haciendo uso de las clases que conforman el modelo.

4.2.3.3. Extensión para Visual Studio Code

La sección 3.1.3 recoge las tecnologías empleadas para la construcción, implementación y validación de la extensión para Visual Studio Code. Su punto de entrada es **extension.ts** que, localizado en la raíz y de uso imperativo, orquesta el funcionamiento de la extensión. Este componente es un proyecto Node, por lo que, al igual que en el caso de la aplicación web Angular, contiene un fichero **package.json** para su configuración esencial —esto es, nombre, autores, licencia y dependencias versionadas—, a la que se suma la configuración que Visual Studio Code introduce en él, ya que incluye los comandos implementados, las opciones de configuración del usuario incorporadas e información relativa a las vistas disponibles al usuario (elementos de la vista lateral, las acciones disponibles y la posición que deben ocupar).

El código fuente de la extensión distribuye sus clases y paquetes en una organización en cuatro capas:

- Cliente. Localizado en el directorio **client**, contiene la lógica destinada a la configuración y registro de todas las peticiones empleadas para la comunicación entre la extensión y el servidor. Configura aspectos como la autenticación o el tiempo de expiración de las peticiones.
- Componentes. La carpeta **components** incluye todos los elementos gráficos que se renderizan en la interfaz de Visual Studio Code: los ítems de la barra lateral (empleados para mostrar un árbol de cursos y sus ejercicios), el panel informativo empleado para la visualización del progreso en tiempo real de la ejecución de un ejercicio y los botones y elementos de estado mostrados en la barra de actividad del entorno (posición inferior).
- Servicios. Alojados en la carpeta **services**, implementan, entre otros, la configuración del sistema de registro de eventos y la capacidad para la actualización en tiempo real del *dashboard* de los profesores.
- Modelo. Análogamente a los demás componentes, el directorio **model** contiene las clases empleadas para la modelización en *software* de las realidades existentes en el dominio y sus interrelaciones.

4.3. Implementación

Tras el establecimiento de los requisitos abarcados durante este hito evolutivo de *VSCode4Teaching* en la sección 4.1 y desarrollados tanto el diseño de cada componente como la arquitectura del proyecto en la sección 4.2, se describe a continuación el proceso de implementación de los requisitos, desarrollando para cada uno la necesidad que busca resolver y el valor que aporta al usuario o al proyecto *software*.

4.3.1. Requisitos funcionales

4.3.1.1. RF-1: autenticación de usuarios y visualización de sus recursos asociados

Este requisito establece la necesidad de que los usuarios de *VSCode4Teaching* puedan autenticarse en la aplicación web para visualizar únicamente sus recursos legítimamente accesibles. Cabe dividir este requisito en dos partes: la autenticación (RF-1.1) y la visualización de los cursos asociados (RF-1.2).

El requisito RF-1.1 establece la necesidad de que los usuarios puedan iniciar sesión para autenticarse utilizando su nombre de usuario y contraseña asociada, garantizando así que son quien dicen ser. Para ello, la aplicación web introduce un formulario que permite a los usuarios introducir las credenciales que ya venían utilizando para la verificación de su identidad en la extensión para Visual Studio Code en versiones anteriores, tal como refleja la figura 4.5.

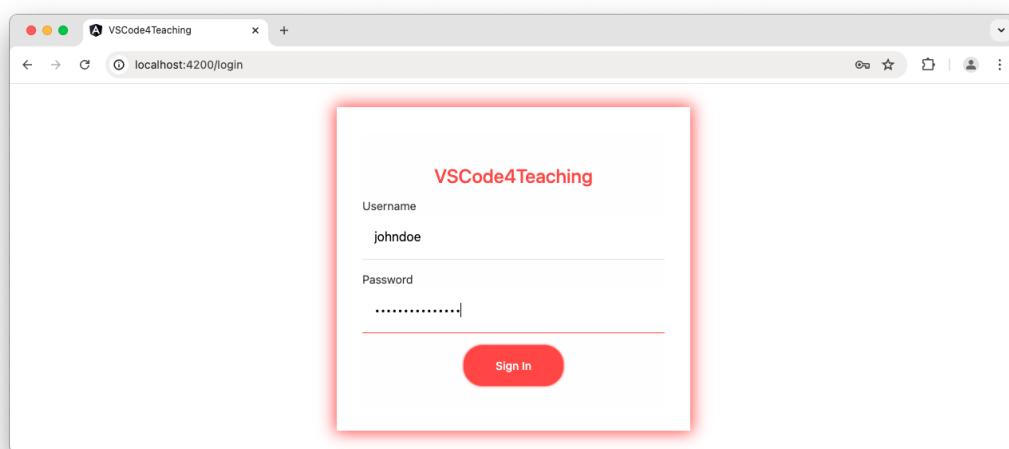


Figura 4.5: Formulario de inicio de sesión para usuarios registrados en la aplicación web.

Una vez cumplimentado, las credenciales se envían al servidor y, en caso de ratificarse su validez, se genera un *token* relativo a la sesión activa del usuario que queda almacenado en el contexto de la pestaña activa mediante el uso del *sessionStorage*, que es un almacenamiento del navegador que permite almacenar pares clave-valor que persisten únicamente al contexto de la página que los crea (por contraposición con el *localStorage*, que persiste a la finalización de la sesión activa salvo que se cierre la sesión ex profeso).

Por otro lado, el requisito RF-1.2 especifica que los usuarios autenticados deben poder visualizar los recursos que tenga asociados a su cuenta. Para ello, una vez que los usuarios inician sesión en la aplicación, visualizan una pantalla de inicio que muestra los cursos de los que forman parte (figura 4.6) que varía según el rol: mientras que los estudiantes tienen un botón para acceder a la pantalla en la que comienzan a realizar ejercicios (RF-9), los docentes acceden a una interfaz (figura 4.7) en la que pueden visualizar los ejercicios de los que dispone el curso con información básica sobre el progreso de cada uno y, además, crear nuevos ejercicios (RF-2), gestionar los alumnos matriculados (RF-6) y compartir el curso (RF-7).

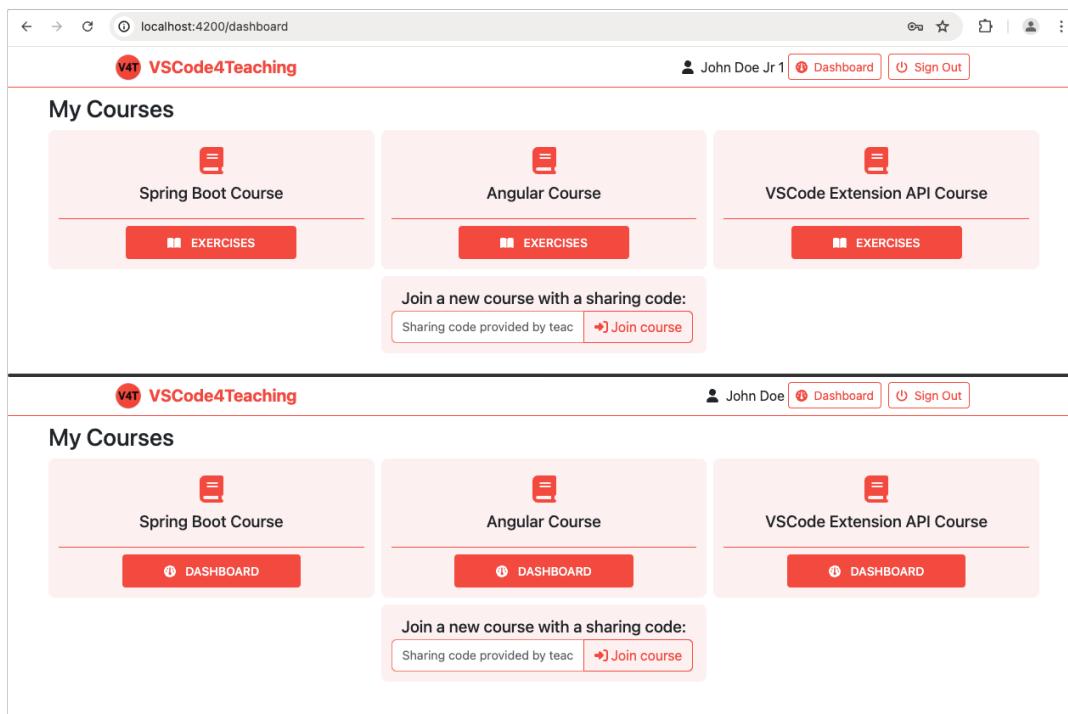


Figura 4.6: Pantalla de inicio para usuarios autenticados, tanto estudiantes (arriba) como docentes (abajo).

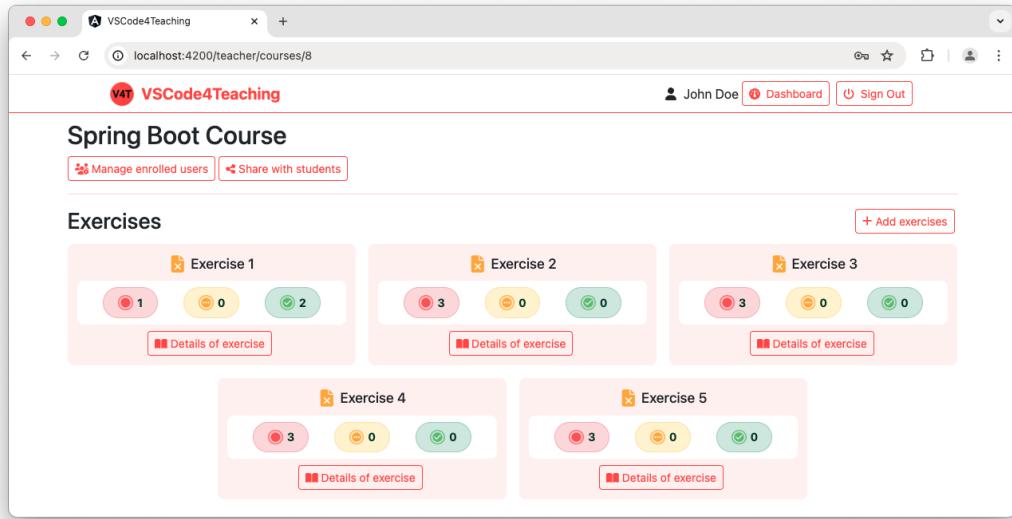


Figura 4.7: Interfaz principal para docentes acerca de cada uno de sus cursos impartidos.

4.3.1.2. RF-2: creación de nuevos ejercicios

Los docentes tienen la capacidad de añadir ejercicios nuevos en los cursos que imparten. En la extensión, disponen de sendos botones para añadir un solo ejercicio o varios contenidos en un mismo directorio. Para replicar este comportamiento en la aplicación web, en la pantalla que les permite visualizar el detalle de sus cursos (introducida en el [RF-1.2](#)) disponen de un botón “Add exercises” (añadir ejercicios) que despliega un modal con todos los elementos y controles necesarios para poder crear nuevos ejercicios.

En un primer momento, el modal solicita al profesor que seleccione un directorio disponible en su sistema local de ficheros. Una vez elegido el directorio, el navegador solicitará permiso de lectura de sus contenidos a través de la *File System Access API* (véase la [sección 3.1.1.2](#)) y, una vez concedido, se determinará automáticamente si el directorio contiene únicamente la plantilla o si tiene, además, propuesta de solución del docente. Se considera que un ejercicio dispone de plantilla de solución si su directorio raíz contiene, al menos, dos directorios *template* y *solution* cuyos contenidos serán tomados, respectivamente, como la plantilla y la propuesta de solución —formato ya empleado anteriormente en la extensión—; interpretándose en otro caso que todos los contenidos conforman la plantilla. Esta exploración puede realizarse directamente en el directorio elegido o en cada subdirectorio que contenga, considerando cada uno de ellos como un nuevo ejercicio y determinando en cada caso si tienen propuesta de solución o únicamente plantilla.

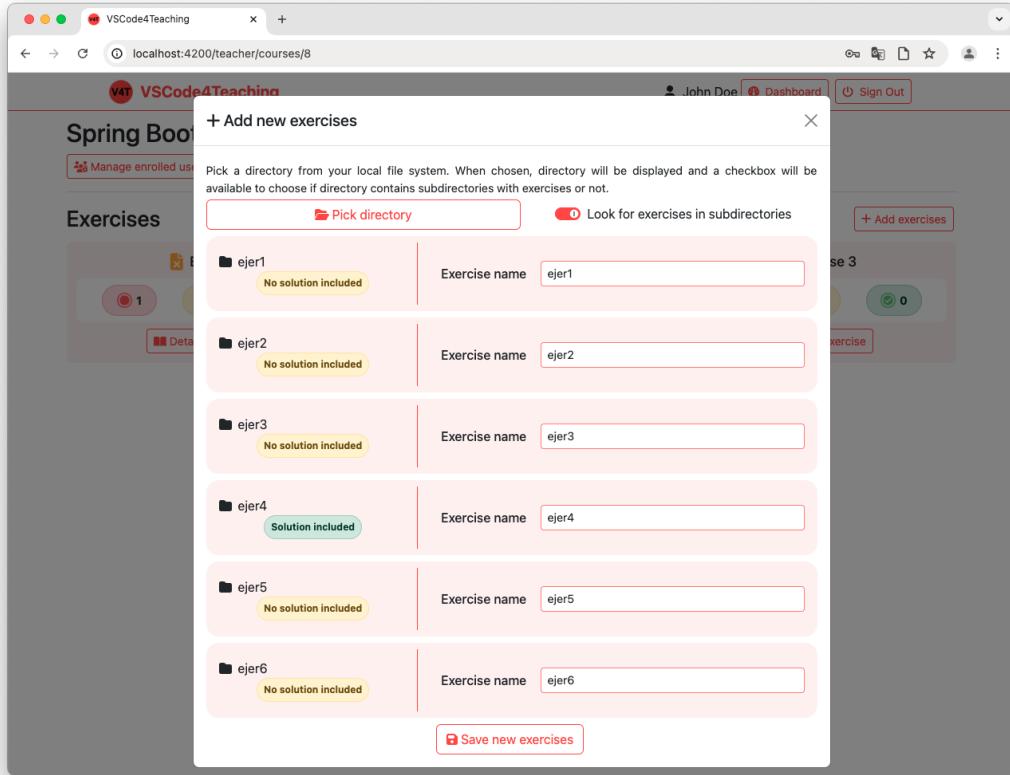


Figura 4.8: Ventana modal para la creación de nuevos ejercicios tras analizar los contenidos del directorio elegido.

Una vez elegido un directorio y determinado si se desea emplear directamente esta carpeta o sus contenidas, se muestra una lista con los ejercicios reconocidos, mostrando para cada uno el directorio local del que proceden y un campo que permite configurar su nombre, tomando por defecto el nombre del directorio que contiene los ficheros asociados. Se muestra un ejemplo en la figura 4.8, en la que se escoge revisar las carpetas contenidas en el directorio elegido, observando que únicamente una de ellas contiene una propuesta de resolución.

Cuando se han determinado los nombres para los ejercicios, es posible iniciar su creación. Este proceso, para cada ejercicio, ejecuta una primera petición para generar el ejercicio en el servidor y, cuando obtiene respuesta, se comprimen en el navegador los ficheros que conforman la plantilla, enviándola una vez comprimida en una nueva petición. En caso de incluir propuesta de solución, a este proceso se suma una nueva compresión y petición para guardar la propuesta de solución. Se ejecutan todos los procesos para la creación de ejercicios concurrentemente, tal como evidencia la figura 4.9, informando a través de una barra de progreso e indicadores de estado del avance del proceso para cada ejercicio. Solo cuando hayan terminado todos los procesos, el modal podrá ser cerrado.

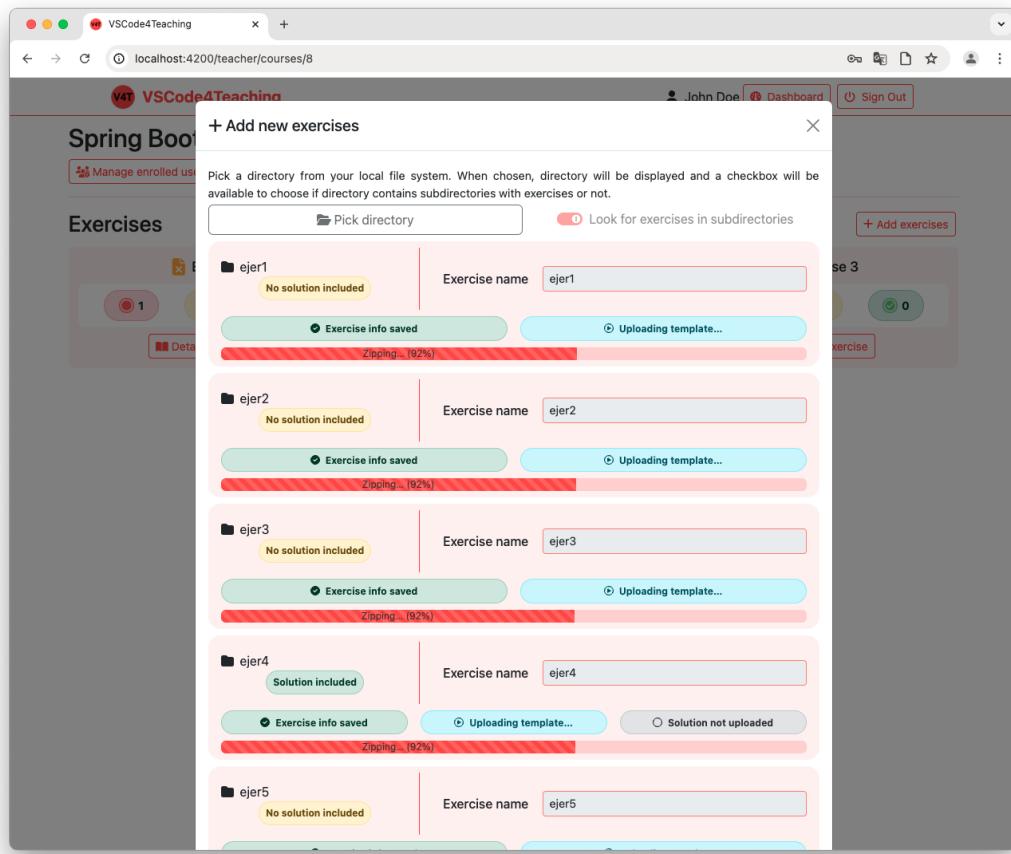


Figura 4.9: Instantánea de la ejecución del proceso de creación de nuevos ejercicios.

4.3.1.3. RF-3: seguimiento en tiempo real de progreso de ejercicios

Cuando utilizan la extensión para Visual Studio Code, los docentes disponen de la capacidad para visualizar el *dashboard* de un ejercicio, que muestra métricas actualizadas en tiempo real sobre el progreso de los estudiantes. Del mismo modo, la extensión incorpora una pantalla muy parecida que permite realizar un seguimiento en tiempo real sobre cada ejercicio de los cursos impartidos por un docente.

Se incluye una captura de esta visualización en la figura 4.10. Esta visualización permite ver a los docentes rápidamente cuántos estudiantes hay matriculados en el curso —y, por tanto, cuántos alumnos realizarán potencialmente el ejercicio—, desgranados debajo según el progreso, diferenciando los que no lo han iniciado, los que están realizándolo y los que lo han finalizado, incluyendo una gráfica semicircular representativa adyacente. Además, también se muestra cuántos estudiantes han modificado su propuesta en los últimos cinco y treinta

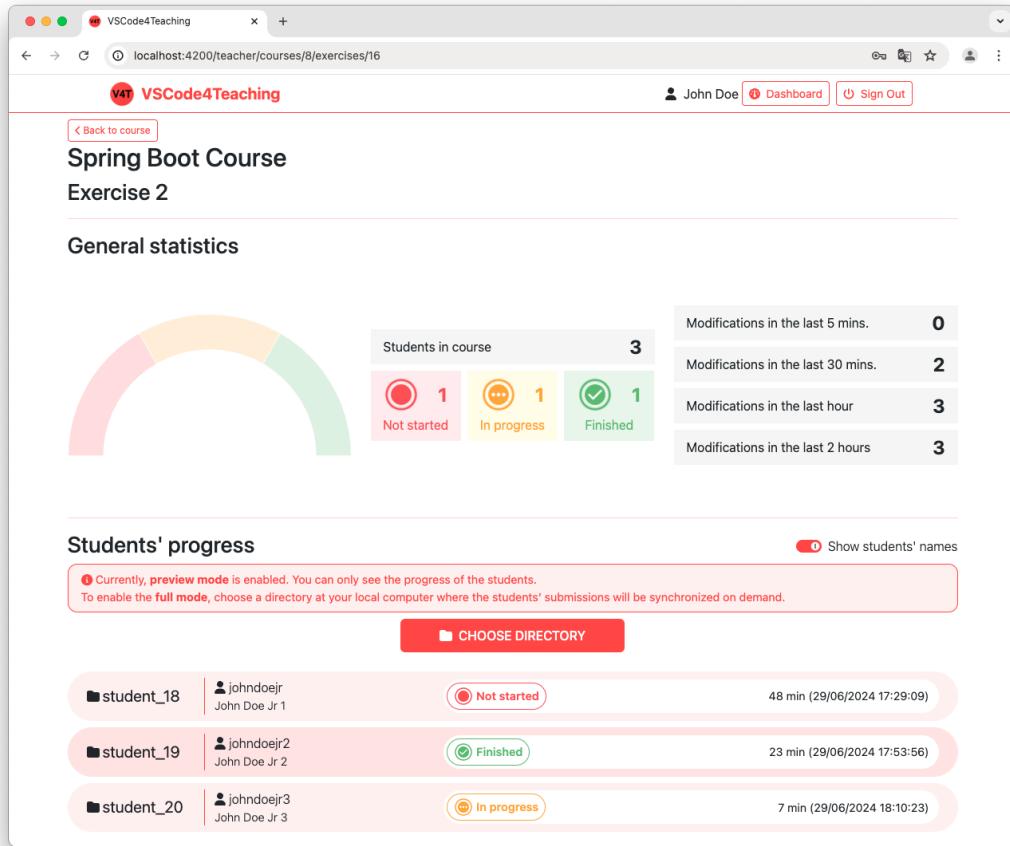


Figura 4.10: *Dashboard* de un ejercicio en la aplicación web con métricas actualizadas en tiempo real para el seguimiento del progreso de los estudiantes.

minutos, en la última hora y dos horas. Debajo de estas estadísticas generales, es posible visualizar el progreso individualizado de cada estudiante, que se muestra de forma anónima: junto a cada directorio de cada propuesta de resolución aparecen el estado del ejercicio y la fecha y tiempo transcurrido tras la última modificación. Es posible asociar cada propuesta de resolución a su autor al modificar el control “Show students’ names”, lo que hará que se muestre en cada fila, además, el apodo y nombre completo de cada estudiante.

La extensión permite disponer del *dashboard* en dos formatos: en forma de previsualización, mostrando únicamente las métricas anteriormente relatadas, o en formato completo, permitiendo abrir los ficheros que componen la propuesta de cada estudiante. Análogamente, la aplicación web permite visualizar únicamente las métricas en formato de previsualización y, además, da al docente la opción de escoger un directorio local para descargar en él bajo demanda las distintas propuestas de los estudiantes, tal como se detalla en el requisito RF-4.

4.3.1.4. RF-4: obtención bajo demanda de ficheros de ejercicios

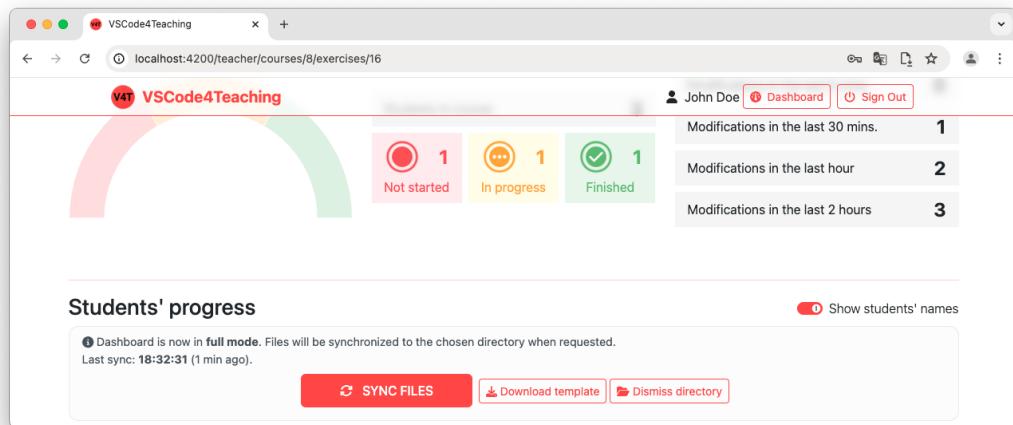


Figura 4.11: Fragmento del *dashboard* que informa sobre la descarga de los ficheros de las propuestas de resolución de los estudiantes.

El *dashboard* de cada ejercicio disponible en la extensión permite, en su formato completo, abrir los ficheros que conforman la propuesta de resolución de cada estudiante y, además, visualizar gráficamente en Visual Studio Code las diferencias existentes entre las propuestas y la plantilla original. Análogamente, y dentro de las capacidades que permiten las tecnologías empleadas (descritas en la sección 3.1.1), la aplicación web brinda a los docentes la capacidad de descargar bajo demanda todas las propuestas de resolución para poder visualizarlas en local en el editor de su preferencia.

El requisito RF-3 introduce el funcionamiento y las capacidades de las que dispone el *dashboard* de ejercicios implementado en la aplicación web. En la sección acerca del progreso de los estudiantes, tal como se puede apreciar en la figura 4.10, se dispone un botón “Choose directory” (elegir directorio) que permite a los docentes escoger una carpeta en su máquina sobre la que concederán permisos de lectura y escritura para descargar en ella las propuestas de resolución de los estudiantes bajo demanda.

Tal como refleja la figura 4.11, al escoger una carpeta local se habilitan nuevas opciones para los docentes, análogas al “modo completo” del *dashboard* de la extensión, permitiendo la sincronización de los ficheros de los estudiantes bajo demanda, registrando cuándo fueron descargados por última vez. Esta sincronización es unidireccional, ya que no se registran las modificaciones locales realizadas por los docentes y, además, la descarga sobrescribe los ficheros previamente existentes en local. Se permite al profesor, adicionalmente, descargar la plantilla del ejercicio (y la propuesta de solución si la tuviera) en el mismo directorio.

4.3.1.5. RF-5: configuración de publicación de solución de ejercicios

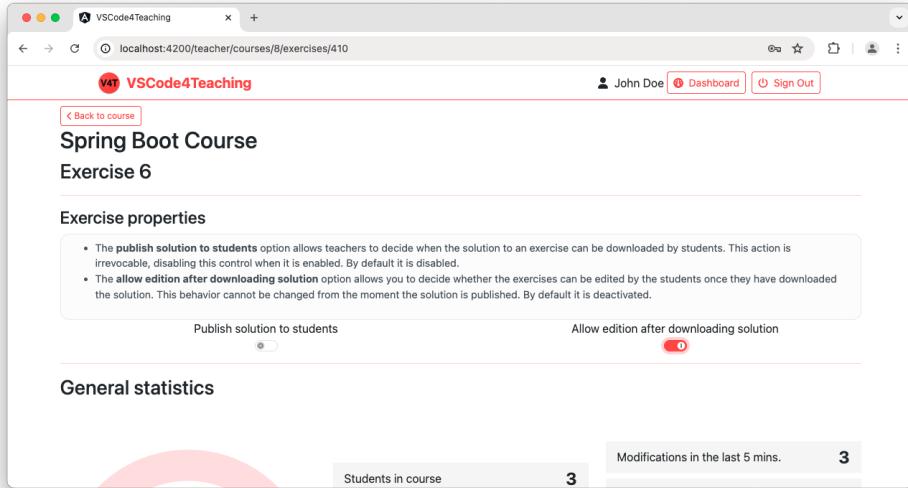


Figura 4.12: Sección del *dashboard* de los ejercicios que permite configurar los parámetros para la configuración de la disponibilidad y comportamiento de la descarga de la propuesta de solución del docente.

Tal como se especifica en el requisito **RF-2**, los ejercicios que los docentes añaden en sus cursos pueden tener asociada una propuesta de solución que los estudiantes pueden descargar en la extensión según dos parámetros de configuración de la disponibilidad de la solución que los docentes pueden ajustar en el *dashboard* de cada ejercicio, tal como muestra la figura 4.12. Los parámetros son:

- *Allow edition after downloading solution* (permitir edición tras descargar solución). Permite ajustar si los estudiantes pueden modificar sus propias propuestas tras descargar la solución o si, por el contrario, al descargar la propuesta de solución se deben impedir nuevas ediciones, marcándolo previamente como finalizado. Este parámetro se puede modificar mientras la solución no sea pública.
- *Publish solution to students* (publicar solución a estudiantes). Permite configurar la disponibilidad para la descarga de la solución a los estudiantes, quienes dispondrán de un elemento gráfico en la extensión que les permitirá descargar la solución en local para, además, poder compararla mediante una interfaz gráfica para visualizar las diferencias existentes entre su propia propuesta y la solución del docente. La activación de la publicación impide la edición del parámetro anterior y, una vez publicada, esta configuración ya no puede revertirse.

4.3.1.6. RF-6: gestión de matriculación de usuarios en cursos

Cuando utilizan la extensión para Visual Studio Code, los docentes tienen la posibilidad de inscribir y eliminar estudiantes y otros profesores de sus cursos. Equivalentemente, se añade a la aplicación web la posibilidad de gestionar las matrículas de un curso. Para ello, tal como introduce el requisito **RF-1.2**, los docentes disponen de un botón “Manage enrolled users” (gestionar usuarios inscritos) que despliega una ventana modal.

Este modal, tal como se muestra en la [figura 4.13](#), muestra una lista con todos los usuarios inscritos en el curso, diferenciando a los profesores de los estudiantes. A excepción del profesor creador, cada uno de estos usuarios puede ser eliminado del curso haciendo uso del botón dispuesto en cada fila. Adicionalmente, en la parte inferior se muestra un desplegable que contiene todos los usuarios potencialmente matriculables; esto es, a todos los usuarios salvo a aquellos que ya están matriculados. El profesor puede escoger uno de ellos y, mediante el botón adyacente, inscribirlo en el curso.

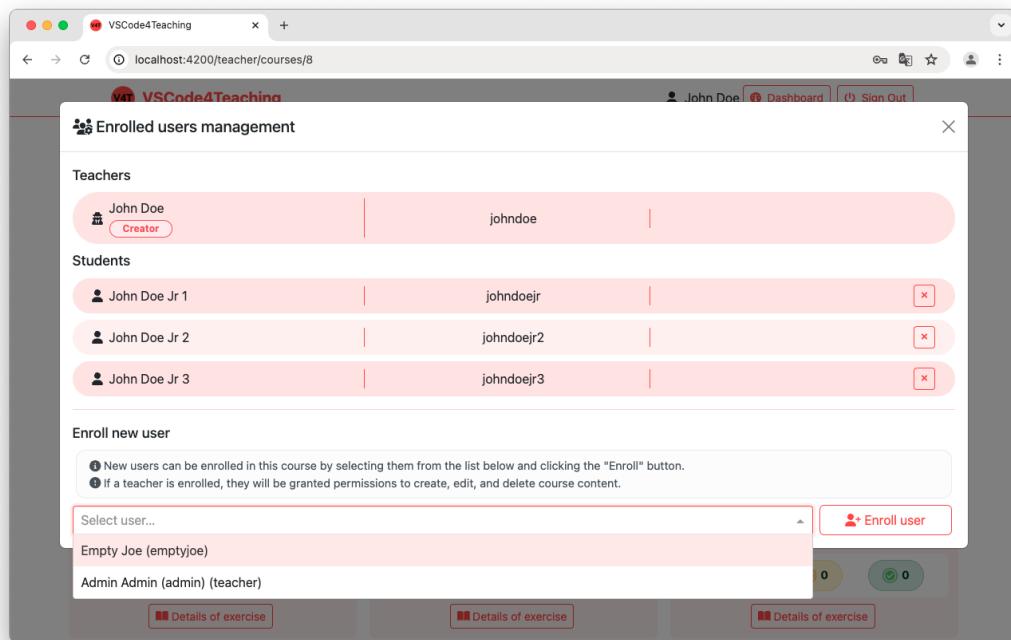


Figura 4.13: Visualización de la lista de usuarios inscritos a un curso y de las capacidades para añadir nuevos usuarios o revocar inscripciones.

4.3.1.7. RF-7: compartición de código asociado a un curso

Existen dos formas para hacer que un estudiante quede matriculado en un curso, tanto en la extensión como en la aplicación web: puede ser inscrito manualmente por un profesor (RF-6) o puede automatricularse (RF-8), que requiere de un código que deben ser proporcionado por el docente.

La extensión permite a los docentes obtener un enlace al servidor de *VSCode4Teaching* en uso que puede divulgar al estudiantado que desee que se automatricule. Este enlace incluye un código único que identifica biunívocamente cada curso existente en la plataforma. Para generararlo, la visualización de los cursos para docentes (RF-1.2) dispone de un botón “Share with students” (compartir con estudiantes) que, tal como queda plasmado en la figura 4.14, muestra a los docentes el código único para la automatrícula en el curso como, además, el enlace al servidor que los alumnos pueden abrir en el navegador para consultar información adaptada al caso del curso específicamente divulgado, incluyendo instrucciones acerca de cómo inscribirse en el curso compartido.

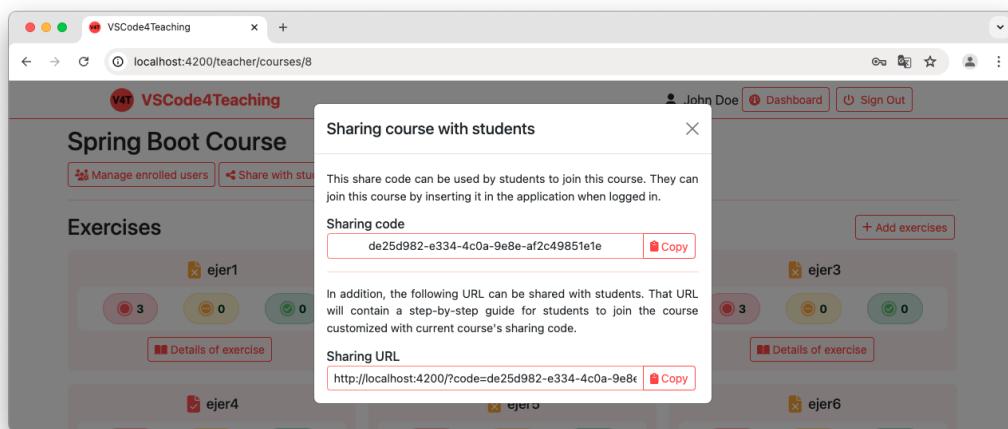


Figura 4.14: Información para la compartición del código único de automatrícula en un curso.

4.3.1.8. RF-8: automatriculación mediante código de compartición

Los estudiantes pueden ser matriculados en los cursos manualmente por sus docentes (RF-6) y, además, tienen la capacidad de inscribirse en cursos por sí mismos utilizando un código proporcionado por sus profesores, ya sea empleando la extensión para Visual Studio Code o a través de la aplicación web. Este requisito es complementario del RF-7, por el que los docentes disponen de la capacidad para generar códigos de compartición para permitir la autoinscripción de estudiantes en sus cursos.

En caso de querer hacerlo a través de la aplicación web, los estudiantes deben iniciar sesión ([RF-1.1](#)) y, dentro de su página principal, disponen de un campo de texto para introducir el código proporcionado por los docentes, tal como se aprecia en la [figura 4.15](#) y, presionando el botón adyacente, pueden inscribirse en un nuevo curso.

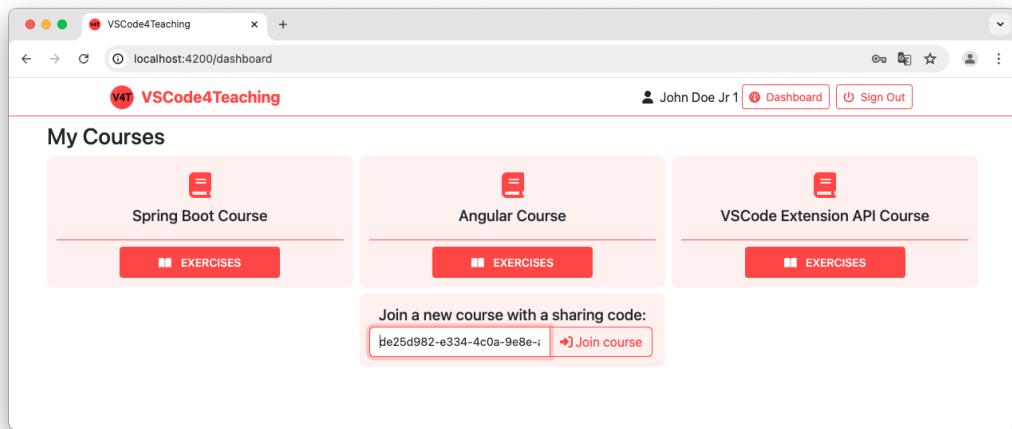


Figura 4.15: Página principal de los estudiantes en la aplicación web con un código de inscripción automática en un curso.

4.3.1.9. RF-9: visualización de un curso y sus ejercicios

Una vez autenticados en *VSCode4Teaching* ([RF-1.1](#)), los estudiantes visualizan una pantalla principal o *dashboard* en el que se muestran los cursos en los que están matriculados, tal como se desarrolla en el requisito [RF-1.2](#). En esta pantalla disponen de un botón “Exercises” (ejercicios) en cada curso que les permite acceder al detalle de los ejercicios de cada curso.

Cuando los estudiantes acceden a la visualización del detalle de un curso, la aplicación les solicita primeramente que escojan un directorio del sistema de ficheros de su computador local para alojar los contenidos de los ejercicios del curso, situación inicial que queda reflejada en la [figura 4.16](#). Si el estudiante ya realizó algún ejercicio, sea total o parcialmente, y mantuvo intacto el directorio que empleó —esto es, respetando el formato de subdirectorios empleado en local por *VSCode4Teaching*—, puede utilizar la misma carpeta para continuar trabajando, ya que coincidirá el estado de los ejercicios almacenados en local con el progreso sincronizado en el servidor y no se producirán desfases.

Una vez escogido el directorio, se muestra a los estudiantes el detalle de los ejercicios asociados al curso, sepárandolos en distintos paneles según su estado de ejecución: los ejercicios en progreso se sitúan en la parte superior, ubicando debajo

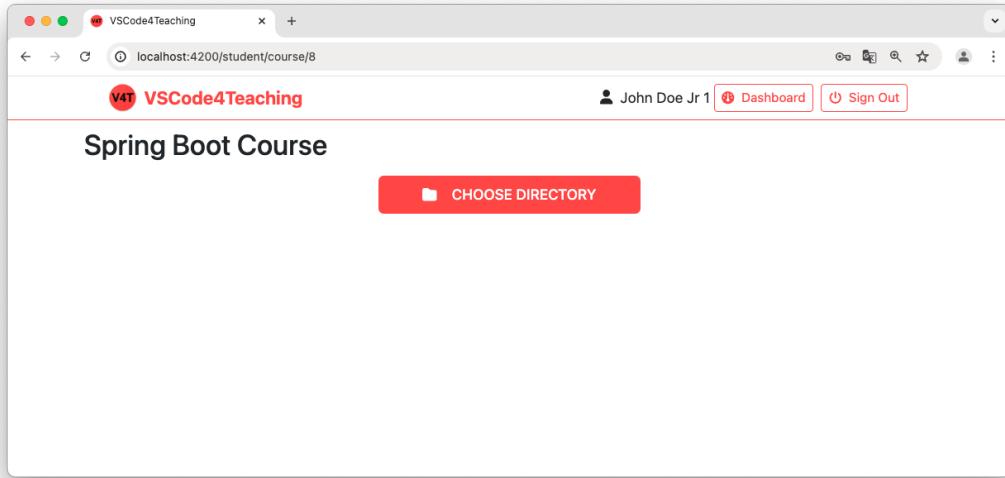


Figura 4.16: Situación inicial de los estudiantes al acceder al detalle de un curso y antes de elegir un directorio local.

los ejercicios no comenzados y los finalizados, tal como se refleja en la [figura 4.17](#). Esta ilustración permite apreciar que el estudiante tiene dos ejercicios pendientes de comenzar (“Exercise 2” y “Exercise 5”), ubicados dentro del panel sombreado en rojo con título “Not started” (no comenzado) y, además, ha finalizado un ejercicio (“Exercise 3”) que aparece dentro del panel sombreado en verde con título “Finished” (finalizado). A estos se suman dos ejercicios dentro del panel amarillo con título “In progress” (en progreso), que el estudiante podrá descargar o empezar a sincronizar, posibilidades implementadas en el requisito [RF-10](#).

4.3.1.10. **RF-10: descarga de los ficheros de una propuesta propia de resolución de un ejercicio**

Dentro de la visualización del detalle de un curso matriculado, y tras escoger un directorio local para sincronizar los ficheros, un estudiante puede visualizar todos los ejercicios del curso diferenciados en distintos paneles según su estado de ejecución, tal como especifica el [RF-9](#).

En el caso de los ejercicios en progreso, al escoger el directorio, la aplicación buscará si existe una versión local del ejercicio, transitando entre los distintos estados reflejados en la [figura 4.18](#). En caso de no existir ninguna carpeta hija del directorio elegido coincidente con un ejercicio en progreso, la aplicación informará al usuario de que deberá descargar el punto de progreso que haya almacenado en el servidor, tal como ocurre con el “Exercise 4” en la [figura 4.17](#). Si, por el contrario, existe un directorio local que coincide con el ejercicio en progreso, la aplicación pregunta al usuario si desea descargar la versión existente en remoto

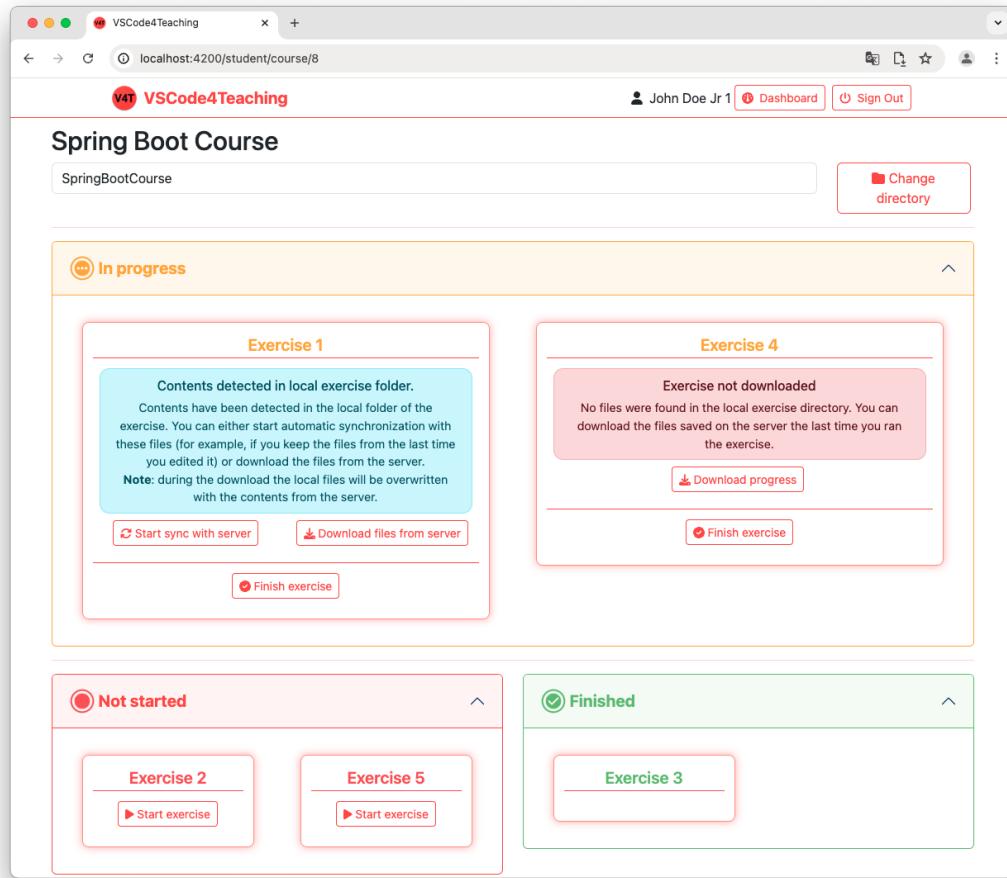


Figura 4.17: Visualización detallada de un curso matriculado por un estudiante con ejercicios ubicados en distintos paneles según su estado.

y sobrescribir los ficheros que colapsen respecto a la versión en local o si, por el contrario, desea iniciar de inmediato la sincronización con el servidor. Este primer escenario es el que sucede en el caso del “Exercise 1” de la figura 4.17.

Estos dos estados iniciales están reflejados en la parte izquierda de la figura 4.18. Al descargar el ejercicio, se transita al estado que se observa en la parte central de la representación, durante el que la aplicación descarga como fichero comprimido los contenidos del ejercicio proporcionados por el servidor y los descomprime en una carpeta propicia específica dentro del directorio elegido por el estudiante, informando de forma visual a través de dos indicadores del estado activo y una barra de progreso. Una vez preparado el directorio del ejercicio, sea porque ha finalizado la descarga y descompresión o porque se ha iniciado la sincronización de contenidos previamente existentes, se inicia la sincronización automática, reflejada visualmente como muestra la parte derecha de la figura 4.18. Esta sincronización queda detalladamente explicada en el requisito RF-11.

Capítulo 4. Descripción informática



Figura 4.18: Representación de la evolución entre los posibles estados visuales de los ejercicios en progreso según su disponibilidad local.

4.3.1.11. RF-11: sincronización automática de las modificaciones en los ejercicios durante su realización

Dentro de la visualización del detalle de los cursos, una vez se ha seleccionado un directorio (RF-9) y se ha descargado el estado de los ejercicios en progreso o se ha decidido utilizar la información previamente disponible en local (RF-10), estos ejercicios comienzan a sincronizarse automáticamente con el servidor.

En lo que a la interacción con el usuario respecta, la interfaz muestra para cada ejercicio en progreso dos señales de activación de la sincronización automática y de su estado. La figura 4.19 refleja la visualización de los dos estados posibles: o bien la aplicación se encuentra esperando a que ocurran nuevos cambios y la sincronización del ejercicio está finalizada porque no hay nuevos cambios a comunicar al servidor (izquierda), o bien se está produciendo la transmisión al servidor de las modificaciones de uno o más ficheros (derecha). Los indicadores mostrados en esta figura quedan encuadrados dentro del estado de cada ejercicio.



Figura 4.19: Captura de las señales de sincronización activa y estado de la sincronización de ejercicios en progreso.

El segundo elemento es el botón “Sync details” (detalles de la sincronización) que se muestra debajo del estado de la sincronización en cada ejercicio (visible en la figura 4.19). Cuando se pulsa, se despliega un modal que contiene una lista con el histórico de los diez ficheros sincronizados más recientemente, los diez siguientes ficheros que serán sincronizados y el que está siendo transmitido al servidor en el momento presente, que dispone de una barra de progreso indicativa del avance de la operación.

La figura 4.20 muestra dos ejemplos de este modal: uno durante la sincronización de un fichero muy pesado (arriba) y otro que refleja la sincronización de una ingente cantidad de ficheros (abajo), constando de una lista de más de 150 ficheros sincronizados y de más de 300 ficheros pendientes de sincronizar. Para cada fichero sincronizado o pendiente de subir, se incluyen dos iconos: uno para reflejar el estado de la subida (rojo si está pendiente, amarillo si está en progreso y verde si se finalizó) y otro que indica el tipo de sincronización, pudiendo tratarse de la creación de un fichero (verde), una modificación de un fichero previamente existente (amarillo) o una eliminación (rojo). Los renombramientos aparecen como eliminaciones y nuevas creaciones.

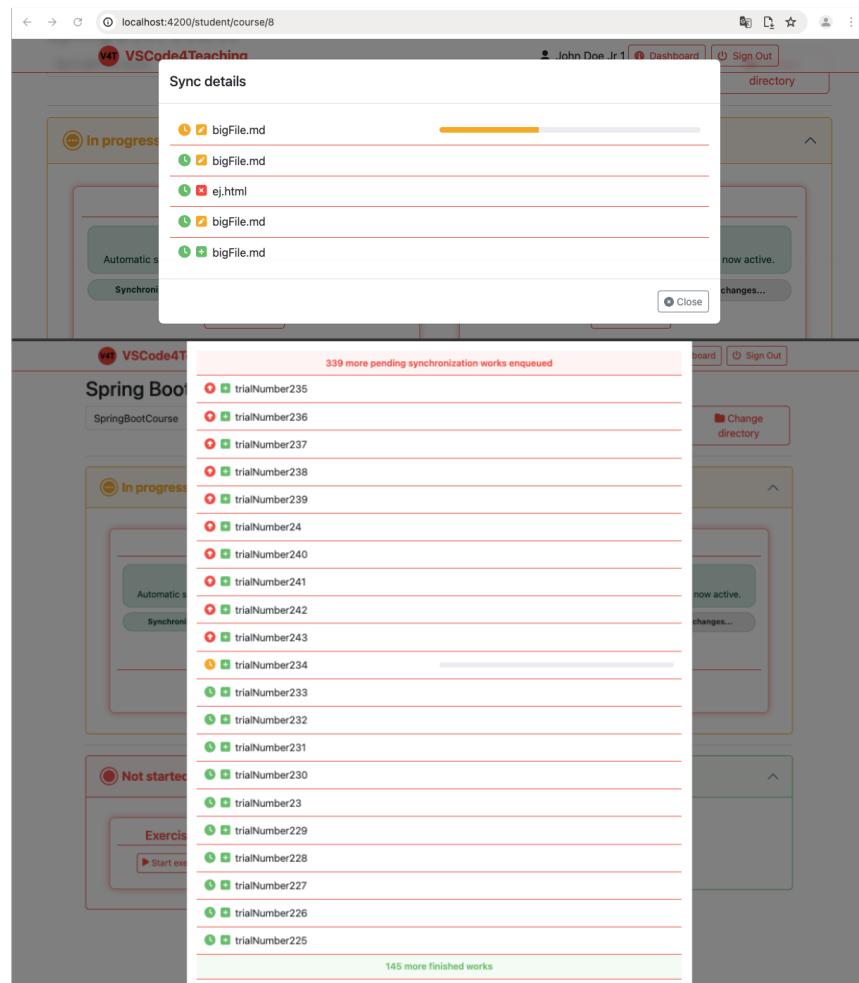


Figura 4.20: Captura del modal explicativo de la sincronización de ficheros de un ejercicio.

El diseño e implementación del algoritmo destinado a los procesos automáticos de cotejamiento y sincronización de cada ejercicio en progreso queda desarrollado pormenorizadamente en el [anexo A](#).

4.3.1.12. RF-12: marcado de ejercicio como finalizado

Tal como introduce el RF-9, los estudiantes disponen de la capacidad para visualizar el detalle de los cursos en los que están matriculados al seleccionar un directorio local, pudiendo ver los ejercicios que los componen, en qué etapa del progreso de su realización se encuentran y realizar distintas acciones con ellos según su estado: mientras que los no comenzados únicamente pueden ser iniciados y pasan al estado “en progreso” (RF-10), los ejercicios que se están realizando y sincronizando en tiempo real pueden ser finalizados. Para ello, los estudiantes pueden pulsar el botón “Finish exercise” (finalizar ejercicio) que aparece en la parte inferior de cada uno de los ejercicios en progreso, tal como sucede en la figura 4.16 con los ejercicios “Exercise 1” y “Exercise 4”.

Una vez acometida esta acción sobre un ejercicio, se guardará como finalizado, trasladándose en la interfaz de usuario al área destinada a mostrar estos ejercicios (panel sombreado en verde con título “Finished”). Esta acción comportará, además, la parada de la sincronización automática, ya que los ejercicios finalizados no pueden ser modificados, consolidando su último punto de progreso sincronizado en el servidor como propuesta final del estudiante para la resolución del ejercicio.

4.3.2. Requisitos no funcionales

4.3.2.1. RN-1: aviso a usuarios con navegadores no compatibles con la *File System Access API*

Tal como se introduce en la sección 3.1.1.2, la aplicación web hace uso de la *File System Access API*, que es la interfaz que permite que la aplicación web interactúe bidireccionalmente con las carpetas y ficheros del sistema local escogidos por los usuarios. Uno de sus mayores inconvenientes es la divergencia en su compatibilidad, ya que algunos navegadores no implementan esta interfaz, por lo que no permiten esta interacción.

Como consecuencia, algunos de los principales procesos de negocio incorporados en *VSCode4Teaching*, tales como la descarga bajo demanda de los ficheros de las propuestas de los estudiantes por parte de los docentes (RF-4) o, en el caso de los alumnos, la obtención de los ficheros de sus ejercicios (RF-10) y la sincronización automática de las modificaciones registradas en los ejercicios durante su realización por parte de los estudiantes (RF-11), no pueden ser realizados en navegadores no compatibles, mientras que el resto de procesos ya incorporados a la aplicación sí pueden ejecutarse, ya que no dependen del uso de la interfaz para la interacción con el sistema local de ficheros.

Para mejorar la interacción del usuario con la aplicación, este requisito establece la necesidad de informar desde el inicio a los usuarios autenticados acerca de la incompatibilidad de su navegador con la API en caso de acceder mediante, por ejemplo, Firefox o Safari, los navegadores más populares que no soportan esta característica. Con este fin, se introduce en todas las pantallas de la aplicación un aviso, tal como muestra la [figura 4.21](#), que aconseja el uso de un navegador compatible para poder aprovechar la completitud de las características de la aplicación.

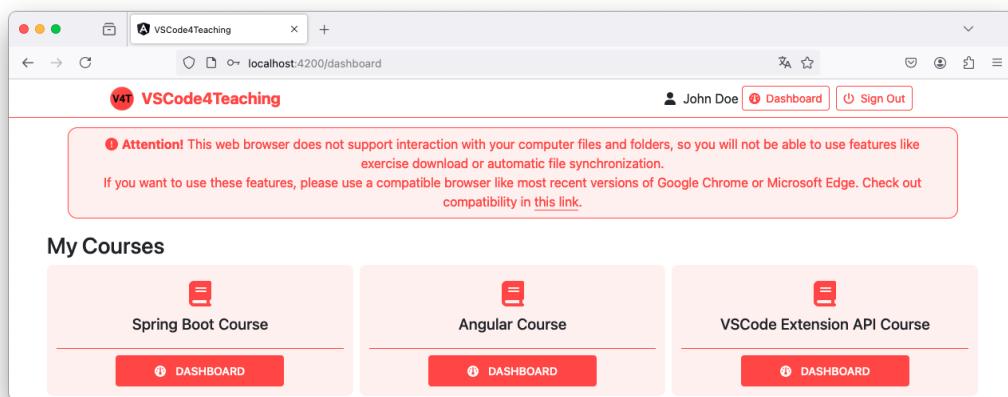


Figura 4.21: *Dashboard* inicial de docentes en Firefox con aviso de incompatibilidad para el uso de la *File System Access API*.

4.3.2.2. RN-2: aspecto visual de la aplicación web y coherencia entre interfaces

El diseño de la interfaz de usuario de la nueva aplicación web aspira a ser coherente con la estética que venía empleándose en la extensión para Visual Studio Code, pretendiendo formar un entono familiar para quien ya utilizase la extensión de *VSCode4Teaching* para Visual Studio Code.

La [figura 4.22](#) muestra los dos *dashboards* para el seguimiento del progreso de los ejercicios existentes: el de la extensión (abajo), que no se ha visto modificado y ya existía; y el de la aplicación web (arriba), que, inspirado en el diseño anterior, refleja la misma información haciendo uso de elementos gráficos de similar índole, ya que emplea el mismo esquema cromático y los mismos iconos informativos.

Esto también se observa en la [figura 4.23](#), que permite observar que los iconos empleados para la distinción de las tipologías de ejercicios son los mismos y que los dispuestos para las acciones de gestión de matriculados, compartición y adición de ejercicios son muy similares, potenciando la familiaridad del entorno a los usuarios.

Capítulo 4. Descripción informática

The figure displays two side-by-side screenshots of the V4T Dashboard interface. Both screenshots show the same exercise details: "Spring Boot Course" and "Exercise 6".

Top Screenshot (Web Browser):

- Exercise properties:** Publish solution to students (checkbox checked), Allow edition after downloading solution (checkbox checked).
- General statistics:** Students in course: 3. Breakdown: Not started (3), In progress (0), Finished (0). Modifications in the last 5 mins: 0.
- Students' progress:** Shows two student entries: student_411 (John Doe Jr 3) and student_412 (John Doe Jr 2). Both are listed as "Not started". A note indicates preview mode is enabled, and a "CHOOSE DIRECTORY" button is present.

Bottom Screenshot (VS Code Extension):

- Exercise configuration:** Teacher short guide (checkbox checked), Publish solution to students (checkbox checked), Allow edition after downloading solution (checkbox checked).
- Student's progress:** Shows three student entries: John Doe Jr 3, John Doe Jr 2, and John Doe Jr 1. All are listed as "Not started". A note indicates preview mode is enabled, and a "Hide student's names" button is present.

Figura 4.22: *Dashboard* de un mismo ejercicio capturados a la vez en la aplicación web (arriba) y en la extensión para Visual Studio Code (abajo).

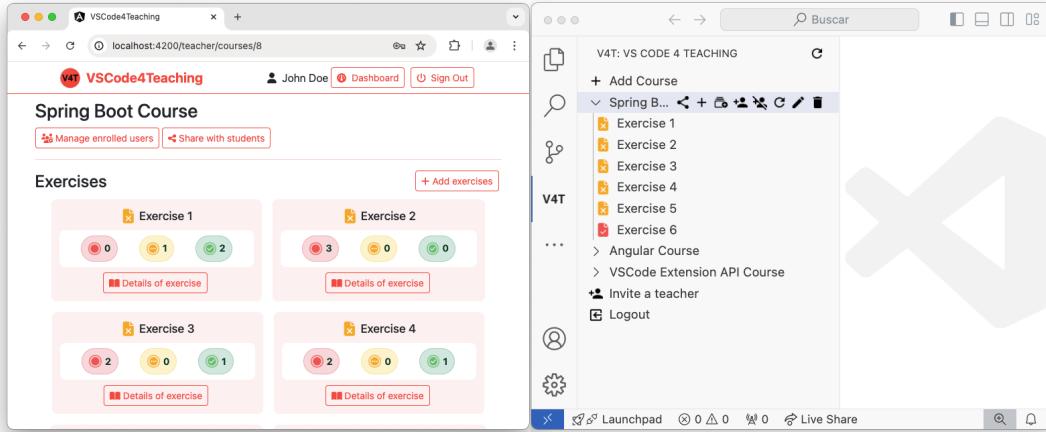


Figura 4.23: Comparativa entre los formatos empleados para mostrar a los docentes los ejercicios y configuraciones de las que disponen sus cursos.

4.3.2.3. RN-3: encriptación de *tokens* JWT para autenticación

El servidor de *VSCode4Teaching* es *stateless*; es decir, no almacena información sobre las interacciones realizadas por los usuarios con objeto de condicionar las siguientes, sino que cada una de las peticiones realizadas debe ser autocontenido e incluye en sí misma mediante sus cabeceras y cuerpo (si lo tiene) todo aquello que se debe tener en cuenta para proporcionar una respuesta, de modo que ninguna respuesta dependerá de las peticiones anteriormente realizadas [86].

Si *VSCode4Teaching* verifica esta característica es porque emplea *tokens* JWT (siglas de *JSON web token*, del inglés “*token* web en *JSON*”) para la autenticación de los usuarios. Estos *tokens* son piezas de información representadas como cadenas de caracteres que permiten autenticar a los usuarios de forma autocontenido, ya que incluyen en sí mismos un *payload* o carga útil que recoge qué usuario es el que está realizando una determinada petición. Se generan durante el inicio de sesión y quedan firmados mediante una clave privada, por lo que la suplantación de un usuario mediante un *token* distinto del expedido durante la autenticación es inviable.

Sobre este mecanismo de autenticación preexistente, el presente requisito implementa el soporte a *tokens* que, una vez generados, son encriptados nuevamente mediante un cifrado simétrico para añadir una capa adicional de seguridad. Este nuevo cifrado permite obtener cadenas no directamente comprensibles como *tokens* JWT, añadiendo al servidor la lógica necesaria para manejar la presencia de cabeceras personalizadas llamadas **Encrypted-Authorization** en las peticiones procedentes de la aplicación web mediante la API REST y, además, como parámetro en la llamada inicial de las conexiones mediante *Web Socket*.

4.3.2.4. RN-4: adaptación de generación de imágenes Docker a nueva arquitectura

El proyecto *VSCode4Teaching* ya contaba previamente con la capacidad ejecutar su servidor a través de una imagen Docker, embebiendo en él la aplicación web Angular auxiliar como recurso estático. Además, se disponía de un fichero en formato YAML³⁴, *docker-compose.yml*, para la orquestación de esta imagen con un contenedor para el sistema de persistencia basado en MySQL mediante Docker Compose.

Aunque la arquitectura del proyecto no ha variado al ejecutar la evolución realizada en este Trabajo Fin de Grado, se confiere mayor importancia a la aplicación web, que ahora busca ser sustitutiva o complementaria a la extensión para Visual Studio Code. La aplicación web auxiliar empleada anteriormente quedaba desplegada en el servidor en la ruta */app*. Para mostrarla directamente al acceder a la raíz y, además, hacerla completamente compatible con el sistema de *routing* propio de Angular (más información en la [sección 3.1.1.1](#)), se ha modificado el formato del despliegue e introducido un nuevo interceptor en el servidor que actúa como *middleware*; esto es, como una capa intermedia de lógica ejecutada antes que el cotejamiento de rutas del servidor que tiene como finalidad detectar si las peticiones entrantes deben ser respondidas por el propio servidor o si deben ser derivadas a la aplicación web Angular para que esta, a través de su propia lógica de enrutamiento, ofrezca como respuesta los recursos gráficos necesarios.

Esta modificación de la lógica ha permitido preservar el método previamente empleado para la generación de la imagen Docker del servidor y la aplicación web. Esta configuración está alojada en el fichero *Dockerfile* situado en la raíz del proyecto, en el que se define la construcción de la imagen en formato *multi-stage*³⁵, articulándola en tres pasos ejecutados secuencialmente, tal como evidencia el [código 4.1](#): compilación de la aplicación Angular en un contenedor Node, dando lugar a sendos recursos estáticos, compilación del servidor en un contenedor Maven (junto con los ficheros estáticos obtenidos en el paso anterior, que quedan copiados dentro del directorio del servidor destinado a este tipo de recursos) y generación de la imagen final sobre una base JDK que permite ejecutar la aplicación Java compilada en el paso anterior.

```
1 FROM node:18 AS angular
2 COPY vscode4teaching-webapp /usr/src/app
3 WORKDIR /usr/src/app
4 RUN ["npm", "install"]
5 RUN ["npm", "run", "build"]
```

³⁴ YAML. Siglas de “otro lenguaje de marcado más” (del inglés *Yet Another Markup Language*). Es un lenguaje de marcado de fácil lectura y comprensión que se aprovecha de la indentación para la jerarquización de declaraciones.

³⁵ *Multi-stage*. Del inglés “múltiples etapas”, se dice que un fichero de configuración es *multi-stage* cuando se ejecutan varias fases en contenedores aislados y diferentes para dar lugar a una imagen final [84].

```

7 FROM maven:3.9.7-eclipse-temurin-11 AS builder
8 COPY vscode4teaching-server /data
9 COPY --from=angular /usr/src/app/dist/vscode4teaching /data/src/main/resources/
  static/
10 WORKDIR /data
11 RUN ["mvn", "clean", "package"]
12
13 FROM eclipse-temurin:11
14 COPY --from=builder /data/target/vscode4teaching-server-*.jar ./app/
  vscode4teaching-server.jar
15 EXPOSE 8080
16 ENTRYPOINT [ "java", "-jar", "./app/vscode4teaching-server.jar" ]

```

Código 4.1: Fichero *Dockerfile* del proyecto, encargado de definir el proceso de generación de la imagen Docker del servidor y la aplicación web.

Además, se ha ejecutado un cambio de localización de ficheros: la definición de Docker Compose, *docker-compose.yml*, y el que contiene las variables de entorno que este último emplea (*.env*) se han trasladado a la raíz del proyecto, donde ya se localizaba previamente el *Dockerfile*.

Anteriormente se introducía en la imagen Docker un *script* de *shell* para organizar la sincronización de dependencias: preguntaba cada cierto tiempo si se disponía de una conexión válida con el sistema de persistencia configurado y solo cuando esta condición se cumplía, el *script* lanzaba la ejecución del servidor, que requiere necesariamente disponer de la base de datos configurada desde su mismo inicio. Esta implementación se ha visto reemplazada por el uso del mecanismo de *healthcheck*, que es una comprobación declarada como parte de la imagen empleada para la ejecución del contenedor de la base de datos, de modo que se relega en Docker Compose la responsabilidad de orquestar el funcionamiento de ambos contenedores. El [código 4.2](#) muestra un fragmento del fichero *docker-compose.yml* en el que se configura la dependencia entre contenedores y el mecanismo de espera mediante esta comprobación: la aplicación declara ser dependiente de la base de datos (*depends-on*), por lo que Docker Compose no ejecutará este contenedor hasta que la base de datos esté “sana”, introduciendo en su configuración el mecanismo para comprobar cuándo este contenedor alcanza esta condición tras su inicialización.

```

1 name: vscode4teaching
2
3 services:
4   app:
5     image: vscode4teaching/vscode4teaching:latest
6     depends_on:
7       db:
8         condition: service_healthy
9     env_file:
10      - path: .env
11        required: true
12     ports:
13       - ${SERVER_PORT}:${SERVER_PORT}
14     volumes:
15       - ./volume-v4t:${V4T_DIRECTORY}
16     restart: on-failure:6
17   db:
18     image: mysql:8.4.0

```

```

19    restart: on-failure:3
20    environment:
21      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
22      MYSQL_DATABASE: ${MYSQL_DATABASE}
23      MYSQL_USER: ${SPRING_DATASOURCE_USERNAME}
24      MYSQL_PASSWORD: ${SPRING_DATASOURCE_PASSWORD}
25    volumes:
26      - ./volume-mysql:/var/lib/mysql
27    healthcheck:
28      test: "mysqladmin ping -h 127.0.0.1 || exit 1"
29      interval: 5s
30      timeout: 5s
31      retries: 5

```

Código 4.2: Fichero *docker-compose.yml* empleado para la orquestación de la imagen Docker del servidor con un contenedor para la base de datos.

4.3.2.5. RN-5: migración a GitHub Actions del sistema de integración continua

Previamente al inicio del presente Trabajo Fin de Grado, el proyecto *VS-Code4Teaching* hacía uso de un sistema de integración y despliegue continuos a través de **Travis CI** en el que recaían tres tareas: cada vez que se producían modificaciones en la rama `master`, se ejecutaban las pruebas automáticas de servidor y extensión y, además, se publicaban nuevas etiquetas de la imagen del servidor en el repositorio en Docker Hub.

Este sistema se ha visto reemplazado por GitHub Actions, descrito en la [sección 3.2.1.3](#). Sin embargo, esta migración no se ha limitado a replicar la funcionalidad previamente utilizada en el sistema de integración, entrega y despliegue continuos sino que, además, se ha aprovechado para ampliar su alcance.

Como consecuencia, el proyecto *software* de *VSCode4Teaching* cuenta ahora con dos flujos de trabajo (*workflows*) diferenciados por componentes. El primero de estos, dedicado a las tareas relativas al servidor y la aplicación web, cuenta con tres trabajos definidos:

- **Test.** Es el trabajo encargado de la ejecución de la batería de pruebas automáticas implementada en el servidor. Se ejecuta cada vez que se produce una alteración en las ramas `master` o `develop` del proyecto y, además, cuando se produce un *pull request*³⁶ entre ellas para conocer el estado de las pruebas antes de fusionar cambios a la rama del código de producción.

³⁶ *Pull request*. En su origen, solicitud realizada al propietario legítimo de un código para realizar una incorporación de otro código implementado fuera del repositorio original. Actualmente, además, se utiliza como sistema organizativo que permite coordinar la fusión de las distintas ramas de un repositorio y dotar a este proceso de un entorno que permite debates entre programadores, solicitar aprobaciones y ejecutar validaciones automáticas, entre otros.

- **Publish.** Se encarga de la publicación de las nuevas versiones lanzadas al repositorio en Docker Hub y, por tanto, se ejecuta cada vez que se modifica la rama `master`. Genera una imagen Docker utilizando la declaración existente en el proyecto (véase la [sección 4.3.2.4](#)) y la publica en el repositorio del proyecto en Docker Hub, liberando dos etiquetas: actualiza la versión *latest* de la imagen y, además, genera una nueva específica para la versión liberada.
- **Deploy.** Una vez publicada una nueva versión en Docker Hub, se accede mediante SSH³⁷ a la máquina de producción, le transfiere una nueva versión del fichero Docker Compose en la que se utiliza la nueva versión puesta en producción, descarga la imagen desde Docker Hub y aplica los cambios al despliegue, dejando activa en producción la nueva versión automáticamente.

Análogamente, se declara un flujo de trabajo para la extensión que comprende únicamente los dos primeros trabajos descritos para el caso anterior, particularizándolos para la plataforma *software* que emplea este componente. De este modo, para la extensión se ejecuta la batería de pruebas automáticas tras cada *commit* en las ramas `master`, `develop` y cuando se produzca un *pull request* entre ellas y, si se ha lanzado una nueva versión, se publica automáticamente en el Visual Studio Code Marketplace.

Todos los flujos anteriormente descritos quedan declarados en sendos ficheros YAML dentro del directorio `.github/workflows` en el proyecto, siendo esta la ubicación requerida por GitHub Actions para ejecutarlos automáticamente. Sirva como ejemplo el [código 4.3](#), que es la especificación para GitHub Actions del trabajo que ejecuta las pruebas automáticas del servidor, que es el primero de los especificados en el fichero `server.ci.yml`.

```

1 name: "Server pipeline"
2
3 on:
4   push:
5     branches:
6       - master
7       - main
8       - develop
9   pull_request:
10    branches:
11      - master
12      - main
13      - develop
14
15 jobs:
16   test:
17     runs-on: ubuntu-latest
18     defaults:
19       run:
20         working-directory: ./vscode4teaching-server
21     steps:
22       - name: Checkout repository
23         uses: actions/checkout@v4

```

³⁷ SSH. Siglas de “terminal seguro” (del inglés *Secure SHell*). Es un protocolo utilizado para la administración remota de computadores.

Capítulo 4. Descripción informática

```
24      - name: Set up Java version
25        uses: actions/setup-java@v4
26        with:
27          java-version: 11
28          distribution: temurin
29    - name: Test
30      run: ./mvnw clean dependency:resolve test
31  publish:
32    if: ${{ github.event_name == 'push' && github.ref == 'refs/heads/main' }}
33    needs: test
34    # [...]
35  deploy:
36    if: ${{ github.event_name == 'push' && github.ref == 'refs/heads/main' }}
37    needs: publish
38    # [...]
```

Código 4.3: Fragmento del flujo de trabajo de acciones automáticas relativas a la integración continua del servidor.

En la figura 4.24 se muestra una parte del histórico de ejecuciones de los flujos de trabajo definidos para varios *commits* ejecutados sobre la rama `develop`. GitHub Actions posibilita acceder al detalle de cada ejecución, permitiendo visualizar gráficamente el estado y disposición de los trabajos realizados. Además, en caso de ser necesario, también se pueden obtener los registros en detalle de la ejecución de cada uno de los trabajos que componen un flujo.

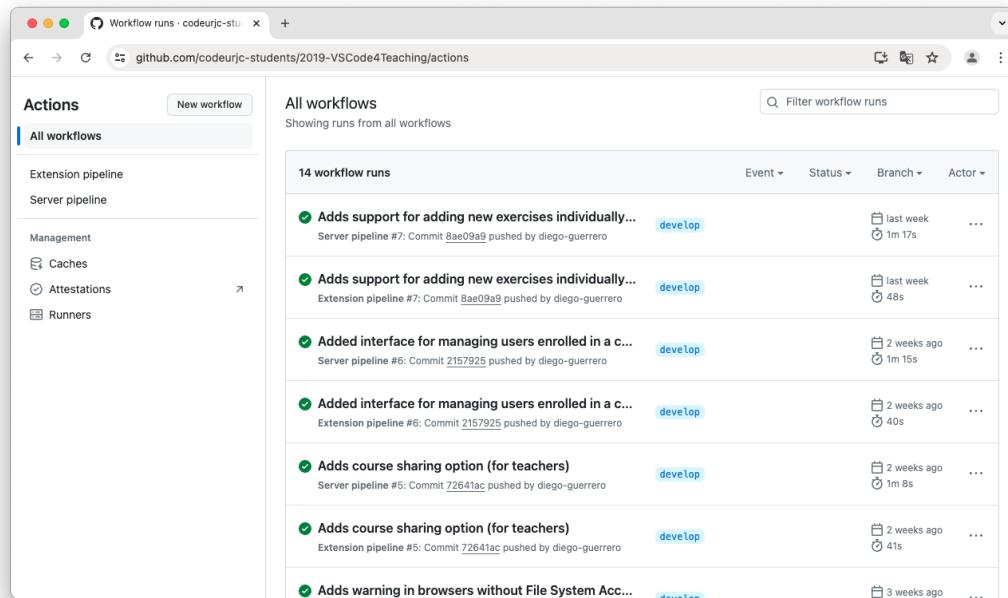


Figura 4.24: Fragmento del listado de ejecuciones de los flujos de trabajo de integración continua con GitHub Actions.

4.4. Verificación de *software* y funcionalidad

Muchos teóricos de la ingeniería del *software* afirman que la verificación del código mediante pruebas es indispensable para poder garantizar su calidad. Por ejemplo, Robert C. Martin afirma que “las pruebas unitarias son necesarias para garantizar que el código es flexible, mantenible y reutilizable, ya que eliminan el miedo a hacer modificaciones en el código” [87].

La nueva aplicación web implementada durante el presente hito evolutivo de *VSCode4Teaching* no incluye pruebas automáticas, ya que se ha primado alcanzar una versión funcional de la aplicación por encima de la verificación de su comportamiento, que requerirá de la implementación de pruebas de tres tipos: unitarias, para corroborar el funcionamiento de la lógica de negocio; de integración, para la comprobación de la correcta comunicación con el servidor; y *end-to-end* o de interfaz, con el fin de verificar que la interfaz de usuario se adecúa a los requerimientos específicos de cada situación; pudiendo verse complementadas por otros tipos de pruebas, como las de carga o rendimiento, las de accesibilidad o las pruebas de funcionamiento en múltiples navegadores, especialmente importantes en este caso por las divergencias de compatibilidad de las tecnologías empleadas. La incorporación de pruebas automáticas para la garantía de la calidad de la aplicación web es uno de los primeros trabajos a futuro del proyecto, tal como recoge la sección 5.2.

El servidor y la extensión cuentan con pruebas automáticas implementadas. Estas pruebas se basan en el uso de aserciones, que son comparaciones entre los valores obtenidos al ejecutar las pruebas y los valores deseados; y en dobles, que son piezas *software* que permiten sustituir las dependencias empleadas que queden fuera del ámbito de la prueba realizada para simular su comportamiento en un escenario real y poder ejecutar la prueba al completo proporcionando valores conocidos a la salida de la dependencia reemplazada.

Las pruebas implementadas son de dos tipos: unitarias, que son aquellas que tienen como alcance una sola capa del *software* y que emplean dobles para simular el funcionamiento de sus dependencias; y de integración, que verifican cómo interactúan entre sí las distintas capas del mismo componente o cómo se produce la comunicación entre el componente verificado y otros agentes *software*.

4.4.1. Pruebas automáticas del servidor

Tal como consta en la sección 3.1.2, el servidor hace uso de JUnit para la implementación y ejecución de las pruebas automáticas implementadas en el servidor. Esta biblioteca incorpora una amplia cantidad de aserciones y permite generar dobles de dependencias de forma sencilla para el implementador. El servidor incluye *tests* de tres tipos:

- Pruebas sobre controladores. Incluidas en el paquete `controllertests`, son pruebas unitarias que verifican el correcto funcionamiento de la capa de los controladores REST de Spring y que se implementan ejecutando llamadas HTTP a la aplicación y utilizando dobles que suplantan el funcionamiento de los servicios que emplean para la generación de la respuesta.
- Pruebas sobre servicios. Localizadas en el paquete `servicetests`, son pruebas unitarias que buscan verificar el correcto funcionamiento de la capa de los servicios Spring, que es la que incluye la traslación al *software* de la lógica de negocio, y se implementan mediante la suplantación con dobles de los DAO de la aplicación, proporcionando instancias basadas en un conjunto de valores conocidos.
- Pruebas de integración. Ubicadas en el paquete `integrationtests`, y al contrario que las anteriores, son pruebas que verifican el funcionamiento de la aplicación en su integridad, lanzando una petición HTTP y sin proporcionar ningún tipo de doble. Emplean un mecanismo para la inicialización de valores conocidos en una base de datos embebida en la aplicación e instanciada únicamente durante el lanzamiento de estas pruebas, hecho que posibilita la ejecución de pruebas que validan la correcta interacción entre las capas de la arquitectura y, además, con el sistema de persistencia.

Cuantitativamente, la batería de pruebas automáticas del servidor contiene un total de 96 pruebas que alcanzan un 78,3 % de las clases y un 79,4 % de los métodos que conforman este componente.

4.4.2. Pruebas automáticas de la extensión

Análogamente al caso anterior, y tal como enuncia la [sección 3.1.3](#), la extensión para Visual Studio Code también incluye una batería de pruebas automáticas implementada sobre Jest que permite verificar el correcto funcionamiento de este componente en su integridad.

Las pruebas incorporadas a la extensión permiten verificar el correcto funcionamiento de su arquitectura en su práctica totalidad, para lo que incorpora 129 pruebas automáticas. Entre ellas, cabe reseñar una cobertura del código superior al 80 % sobre el cliente empleado para el intercambio de peticiones con el servidor, sobre algunos de los elementos empleados en la interfaz de usuario (como los elementos mostrados en la barra de actividad) y sobre el modelo del dominio. El dato de cobertura total de la extensión es de un 61,7 % de las líneas de código del proyecto y de un 54,9 % de las funciones que incorpora.

4.5. Distribución

Se introduce a continuación la forma en que se divulga el proyecto *VSCode4Teaching* en sus distintas etapas: tanto como código fuente ([sección 4.5.1](#)) como los artefactos construidos de la extensión y el servidor con la aplicación web ([sección 4.5.2](#)).

4.5.1. Distribución del código fuente

El código fuente del proyecto *VSCode4Teaching* se encuentra íntegramente publicado en la red a través de un repositorio alojado en GitHub ([sección 3.1.4.1](#)), tal como ilustra la [figura 4.25](#).

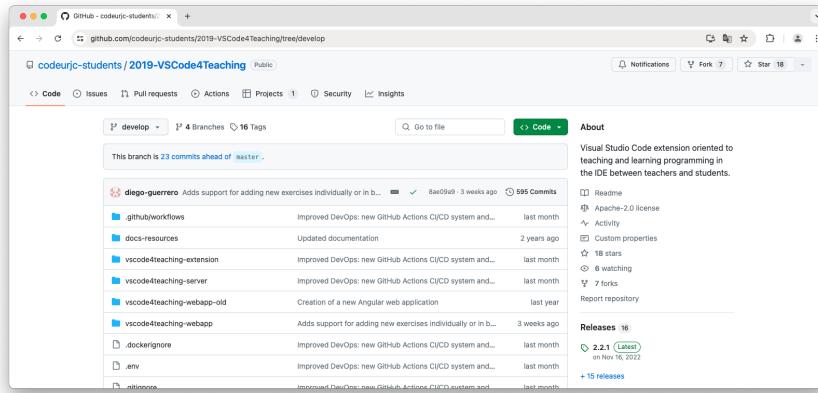


Figura 4.25: Repositorio en GitHub del proyecto *VSCode4Teaching*.

Este repositorio viene utilizándose desde el inicio del proyecto y contiene el histórico completo de cambios producidos en la aplicación durante su evolución. Se encuentra alojado en la siguiente dirección:

<https://github.com/codeurjc-students/2019-VSCode4Teaching>.

Tal como incluye en el fichero `LICENSE` alojado en su raíz, el código fuente de *VSCode4Teaching* está sujeto a la licencia Apache License 2.0 [88]. Esta licencia es permisiva y permite a otros desarrolladores utilizar el código del proyecto para su modificación y redistribución libremente siempre y cuando se mantenga la licencia sobre todas aquellas partes del fuente que no hayan sido adaptadas en las nuevas versiones generadas.

Esta licencia permite, además, aseverar que *VSCode4Teaching* es *software libre*, ya que otorga a sus usuarios las cuatro libertades esenciales: libertad de ejecutarlo como se desee y para lo que se desee, libertad de estudiar cómo funciona y poder adaptarlo para modificar su comportamiento, libertad para redistribuirlo y libertad para distribuir también las copias modificadas [89].

4.5.2. Distribución de artefactos

Además de la distribución de su código fuente, el proyecto *VSCode4Teaching* publica sus artefactos empaquetados para una utilización directa más sencilla de la aplicación. Tal como se introduce en la sección 3.1.4.3, se hace uso de dos repositorios públicos para la divulgación de la extensión para Visual Studio Code y del servidor empaquetado junto con la aplicación web en una imagen Docker (véase el requisito RN-4).

La extensión queda publicada mediante el sistema de integración continua (véase el requisito RN-5) en el Visual Studio Code Marketplace, lo que posibilita que esté disponible directamente para los usuarios a través de la herramienta integrada para la búsqueda e instalación de extensiones, tal como muestra la figura 4.26.

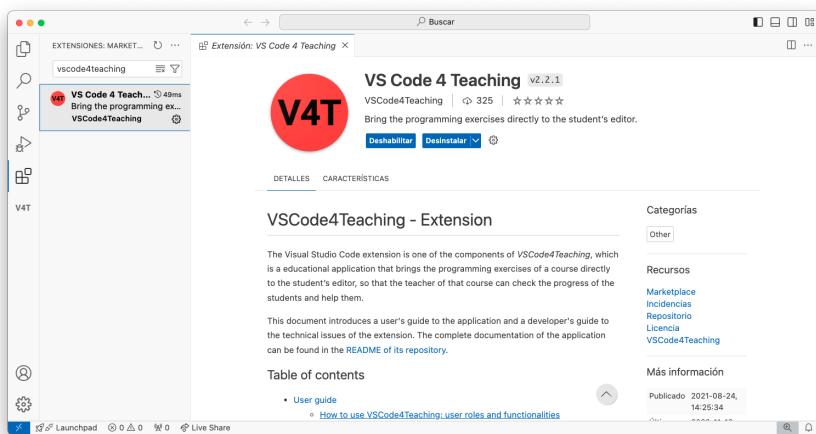


Figura 4.26: Extensión de *VSCode4Teaching* en el área de búsqueda y descarga de extensiones en Visual Studio Code.

El enlace a la extensión publicada es el siguiente:

<https://marketplace.visualstudio.com/items?itemName=VSCode4Teaching.vscode4teaching>.

Por otro lado, el servidor queda empaquetado junto con la aplicación web como *frontend* en una imagen Docker generada mediante el sistema de CI/CD que queda publicada en el Docker Hub, tal como se detalla en los requisitos RN-4 (acerca de la generación) y RN-5 (sobre la automatización), quedando disponible para su utilización mediante, por ejemplo, el fichero *docker-compose.yml* disponible en el repositorio.

El enlace a la imagen publicada del servidor con la aplicación web es el siguiente:

<https://hub.docker.com/r/vscode4teaching/vscode4teaching>.

5

Conclusiones y trabajos futuros

Esta última sección concluye el Trabajo Fin de Grado realizado, para lo que se analiza el cumplimiento de los objetivos estipulados —[sección 5.1](#)—, se introducen varios posibles trabajos futuros de interés para el proyecto *VSCode4Teaching* —[sección 5.2](#)— e incluyendo, para finalizar, una opinión personal final acerca del Trabajo Fin de Grado —[sección 5.3](#)—.

5.1. Cumplimiento de los objetivos estipulados

El actual Trabajo Fin de Grado supone un cuarto hito evolutivo en el proyecto *VSCode4Teaching*, basado en los objetivos iniciales estipulados recogidos en el [sección 2](#) sobre los que, a su vez, se asientan los requisitos especificados en la [sección 4.1](#) y cuyo diseño e implementación queda recogido en las subsiguientes secciones. Se confirma el cumplimiento de la totalidad de los objetivos establecidos, tal como se detalla a continuación:

- El objetivo primero establece la necesidad de que los usuarios registrados puedan autenticarse ([RF-1.1](#)) y, cuando hayan iniciado sesión de forma exitosa, puedan ver sus cursos matriculados, en el caso de los estudiantes; o impartidos, en el caso de los docentes, quienes también disponen de una visualización del detalle de un curso que refleja los ejercicios que contiene, aportando información básica adicional y acceso a las opciones de gestión del curso ([RF-1.2](#)).

Este objetivo también viene ratificado por la implementación del **RF-9**, por el que los estudiantes disponen de una pantalla en la que, una vez elegido un directorio de su sistema local de ficheros, pueden visualizar con detalle el estado de ejecución de los ejercicios de sus cursos. Además, cabe reseñar la utilidad del requisito **RN-2** en este objetivo, ya que proporcionar un aspecto visual coherente con la extensión para Visual Studio Code facilita a los usuarios previamente registrados la utilización y comprensión de la nueva aplicación web.

- El segundo objetivo especifica la necesidad de los docentes de disponer de la capacidad para añadir y gestionar los ejercicios de sus cursos. Este objetivo viene implementado por dos requisitos funcionales: **RF-2**, por el que los docentes pueden añadir nuevos ejercicios en sus cursos de forma individual o en lotes de ejercicios con o sin propuesta de solución; y el **RF-5**, que establece la necesidad de configurar la visibilidad de la solución propuesta por el profesor y la capacidad para realizar nuevas ediciones tras ser descargada por los estudiantes.

Este objetivo, además, viene complementado por el **RN-1**, por el que se muestra un aviso por pantalla en caso de utilizar un navegador no compatible con la interfaz para la interacción con los sistemas de ficheros locales, empleada para la creación de nuevos ejercicios.

- El objetivo tercero marca la necesidad de que los docentes dispongan de herramientas para el seguimiento en tiempo real del progreso de los estudiantes inscritos en sus cursos impartidos. Su intención queda materializada en la implementación de los requisitos **RN-3**, por el que los docentes disponen de un *dashboard* para cada ejercicio de sus cursos actualizado en tiempo real que contiene numerosas métricas acerca del progreso de los alumnos; y **RN-4**, que añade a la interfaz anteriormente citada la posibilidad de elegir un directorio de sistema local de ficheros para descargar en él bajo demanda los ficheros que integran las propuestas de resolución de ejercicios de los estudiantes. Este último requisito conduce a la mención del valor aportado del requisito **RN-1**, ya que la característica para la descarga no está disponible en navegadores no compatibles con la *File System Access API*.
- El cuarto objetivo complementa las necesidades de los docentes en *VSCODE4Teaching* especificando la necesidad para la gestión de la matriculación de estudiantes en los cursos que imparten. Con este fin, el requisito **RF-6** introduce la capacidad para la gestión de los estudiantes inscritos en los cursos a través de una interfaz visual que permite revocar inscripciones o matricular a nuevos usuarios. Adicionalmente, los docentes disponen de la capacidad para compartir un código único de inscripción asociado a sus cursos (**RF-7**) que puede ser empleado por los estudiantes para su automatriculación (**RF-8**).

- El objetivo quinto pone el foco en los estudiantes, quienes deben poder visualizar los ejercicios que componen sus cursos y realizarlos, sincronizando las modificaciones que realicen en sus propuestas de resolución. Este objetivo queda implementado a través de cuatro requisitos funcionales encadenados: la incorporación de la interfaz de usuario que permite a los estudiantes escoger un directorio del sistema de ficheros de su computador y obtener el detalle del progreso en la ejecución de los ejercicios del curso (**RF-9**), poder descargar las plantillas iniciales de los ejercicios recién comenzados o el último punto de progreso sincronizado en el servidor en el directorio escogido (**RF-10**), sincronizar automáticamente con el servidor el progreso de los ejercicios cada vez que se crea, modifica o elimina un fichero en el sistema local del estudiante (**RF-11**) y otorgar a los estudiantes la capacidad para marcar sus propuestas de resolución como finalizadas, impidiendo nuevas ediciones (**RF-12**). Cabe mencionar nuevamente la utilidad del requisito **RN-1** y el valor que aporta a la ejecución exitosa del presente objetivo.
- El sexto objetivo hace énfasis en la calidad del proyecto, procurando la ejecución de actuaciones para la mejora de sus atributos de calidad. Entre las actuaciones realizadas, destaca la adición de una capa de seguridad extra en el mecanismo de autenticación (**RN-3**), la adecuación de la generación de artefactos a la nueva arquitectura adoptada en el proyecto (**RN-4**) y la mejora y automatización integral de los procesos de integración, entrega y despliegue continuos (**RN-5**).

5.2. Trabajos futuros: hoja de ruta del proyecto *VSCode4Teaching*

VSCode4Teaching es un proyecto que, tal como se refleja en la [sección 1.2](#), es el resultado de, junto con el presente, cuatro Trabajos Fin de Grado que marcan sobre él cuatro hitos evolutivos.

El actual Trabajo Fin de Grado no ha completado la migración de la extensión al formato de aplicación *web*, aunque la funcionalidad principal de la aplicación ya puede ser ejecutada a través de esta nueva interfaz de usuario. La actual situación del proyecto permite plantear una extensa batería de posibles trabajos futuros de gran interés y valor para el usuario.

Para la mejora de la aplicación *web* de *VSCode4Teaching*, cabe considerar:

1. Finalizar algunas de las características más complejas, tales como la gestión pormenorizada de cursos y ejercicios, así como perfeccionar el funcionamiento de las características existentes, realizando sobre la aplicación nuevos y constantes trabajos de mantenimiento correctivo y perfectivo.

2. Incorporar la posibilidad de visualizar los contenidos de los ejercicios y de modificarlos a través de un editor de código integrado ligero que quede embedido en la aplicación *web*. Aunque puede ser muy complejo dotar a la herramienta de la capacidad para ejecutar los proyectos de los alumnos, es interesante disponer un editor mínimamente interactivo que permita visualizar los contenidos de los ejercicios o hacer ediciones rápidas sobre ellos sin necesidad de utilizar un directorio en el sistema local de ficheros, permitiendo así, además, ver o editar ejercicios en navegadores no compatibles con la *File System Access API*. Por ejemplo, el editor que emplea Visual Studio Code, llamado *Monaco* [90], está disponible como *software* libre bajo licencia MIT y puede ser integrado en cualquier aplicación *web*.
3. Implementar una amplia batería de pruebas automáticas de diversa índole: pruebas unitarias asociadas a la lógica de negocio, pruebas de integración para garantizar la correcta comunicación con el servidor y pruebas de sistema o *end to end* (E2E) para ratificar el correcto funcionamiento de los elementos dispuestos en la interfaz de usuario y la adecuación de la implementación de los distintos procesos de negocio disponibles para los usuarios.

Por otro lado, en el actual punto de evolución de *VSCode4Teaching* como proyecto *software*, cabe analizar si es pertinente continuar manteniendo la extensión para Visual Studio Code en paralelo a la aplicación *web* y disponer de dos clientes plenamente funcionales disponibles para los usuarios. Esta decisión debe dirimirse tomando en consideración varios factores, entre los que cabe destacar la dificultad y el coste temporal que conlleva el mantenimiento paralelo de dos clientes con la misma funcionalidad y finalidad.

En particular, si se decidiese continuar manteniendo ambos clientes de forma paralela, podría ser interesante plantear una reestructuración de su código para abstraer la funcionalidad común a ambos componentes a una biblioteca propia del proyecto. Como ambos clientes se basan en la plataforma Node y en la gestión de paquetes realizada a través de NPM, tal como se detalla en la [sección 3.1.1](#) y la [sección 3.1.3](#), puede resultar conveniente generar un paquete que abstraiga toda la lógica de negocio y la interacción con el servidor, que es compartida por ambas aplicaciones y que actualmente se realiza utilizando bibliotecas y mecanismos diferentes, logrando así eliminar gran parte de la duplicidad de código entre ambos clientes y, por tanto, facilitando el esfuerzo de mantenimiento, de modo que cada uno deberá contener únicamente el código necesario para la mediación entre la interfaz de usuario —necesariamente específica de cada uno— y la lógica de negocio, disponible a través de servicios comunes.

Otro trabajo futuro conveniente a realizar en *VSCode4Teaching* es implementar una mejora que permita extender el uso del *Web Socket* existente para ampliar su funcionalidad y dotar a los clientes de una interacción integral en tiempo real. Para ello, se puede establecer una política de uso del *Web Socket* que permita

identificar qué eventos son de interés para cada usuario según sus cursos impartidos o matriculados y, en consecuencia, notificar la ocurrencia de eventos para mantener actualizadas de inmediato todas las interfaces gráficas que los usuarios conectados estén empleando.

Adicionalmente, es susceptible de mejora el sistema de transmisión de los ficheros entre servidor y clientes. Actualmente, la extensión para Visual Studio Code genera un fichero comprimido con la totalidad del contenido de los ejercicios cada vez que se persiste una sola modificación de un archivo, lo que hace que esta operación sea costosa en tiempo y en recursos. Se puede unificar la forma en que se envían las modificaciones parciales realizadas por los estudiantes y remitirlas igual que en el caso de la aplicación *web*, que ciñe la transmisión al envío de cada fichero creado, modificado o eliminado. Esta “atomización” de las tareas permitirá potenciar la eficiencia de la comunicación entre cliente y servidor, pudiendo preservar el formato de transmisión de archivos comprimidos cuando se requiera el envío de una ingente cantidad de ficheros —por ejemplo, al subir la plantilla de los ejercicios o cuando un docente descarga por primera vez los ficheros de las propuestas de los alumnos—.

A parte de estos objetivos, la hoja de ruta de *VSCODE4Teaching* recoge propuestas de nueva funcionalidad para incorporar en el proyecto, tales como:

1. Dotar a los ejercicios de mayor funcionalidad, permitiendo a los docentes características como: escoger su visibilidad hacia el estudiantado, configurar un único ejercicio como “activo”, permitir escoger el orden en que se muestran, determinar la fecha máxima de finalización o estipular un periodo de tiempo máximo permitido para su realización.
2. Añadir soporte a las calificaciones, de modo que los docentes puedan puntear las propuestas de resolución de ejercicios elaboradas por los estudiantes y aportarles realimentación, permitiendo al alumnado revisar las calificaciones otorgadas. Además, se considera la posibilidad de introducir estrategias para la detección de plagio que permitan determinar de forma automática la calificación de los ejercicios por comparación con la propuesta de solución del docente o con las demás propuestas del estudiantado.
3. Introducir un nuevo rol de administración que disponga de capacidades específicas para la gestión completa de cursos, ejercicios y usuarios.
4. Explorar la integración del proyecto con entornos LMS³⁸ y, en particular, con el Aula Virtual de la Universidad Rey Juan Carlos, basado en Moodle [91], mediante el uso de un interfaz LTI³⁹ [92].

³⁸ LMS. Siglas de “sistema de gestión de aprendizaje” (del inglés *Learning Management System*).

³⁹ LTI. Siglas de “interoperabilidad entre herramientas de aprendizaje” (del inglés *Learning Tools Interoperability*).

5.3. Aprendizajes personales

Permítaseme redactar en primera persona la sección final de la memoria de mi segundo Trabajo Fin de Grado, en la que busco plasmar las conclusiones personales más destacadas de esta experiencia.

Tal como ya dije al concluir mi primera memoria, los Trabajos Fin de Grado que he escogido son una excelente forma de trasladar al plano práctico tantos conocimientos teóricos recibidos durante las asignaturas del itinerario formativo de mi doble grado: después de estar cuatro años recibiendo una formación teórica amplia y de calidad sobre la ingeniería informática y la ingeniería del software, trabajar en *VSCode4Teaching* me ha permitido trasladar a la práctica

Mis Trabajos Fin de Grado me han permitido acercarme a la rama de mis ingenierías que más disfruto, que es el desarrollo de aplicaciones web y todo lo que conlleva: la toma de decisiones de diseño y arquitectura, la utilización de un enorme abanico de tecnologías disponibles, el necesario continuo aprendizaje de los estándares más elementales, que están en constante evolución; la filosofía DevOps y la automatización de los procesos del software, la implementación de nuevos requisitos en servidor y cliente... un sinfín de tareas que envuelven la rama a la que ya me dedico profesionalmente y de la que quiero seguir aprendiendo más y más.

Toca ahora seguir aprendiendo, ya que me dedico con auténtica vocación a una disciplina muy exigente, en plena expansión, que cada vez dispone de más y más opciones para que los desarrolladores generemos mejores herramientas y, al fin y al cabo, para que cada vez ayudemos mejor a los usuarios generando *software* que aporte más valor y sea de mejor calidad.

Si los planes salen adelante, el punto que termina este párrafo no será mi punto final en el proyecto *VSCode4Teaching*, sino que supondrá un punto y aparte que cierra el cuarto peldaño en la escalera evolutiva de *VSCode4Teaching* y sienta la base de un nuevo trabajo, ya que será la aplicación sobre la que edificaré mi Trabajo Fin de Máster.

Bibliografía

- [1] Asamblea General de la Organización de las Naciones Unidas. (Diciembre de 1948). *Declaración Universal de los Derechos Humanos* (217 [III] A). París, Francia. <https://un.org>.
- [2] Asamblea General de la Organización de las Naciones Unidas. (Octubre de 2015). *Transformar nuestro mundo: la Agenda 2030 para el Desarrollo Sostenible* (A/RES/70/1). Nueva York, Estados Unidos de América. <https://undocs.org>.
- [3] Unión Internacional de Telecomunicaciones (ITU). (2022). *Global Connectivity Report 2022*. <https://itu.int>.
- [4] Ley Orgánica 3/2020, por la que se modifica la Ley Orgánica 2/2006, de 3 de mayo, de educación. (29 de diciembre de 2020, publicado en BOE núm. 340, de 30 de diciembre de 2020, páginas 122868 a 122953). <https://boe.es>.
- [5] Prensky, M. (2001). *Digital Natives, Digital Immigrants*. <https://marcprensky.com>.
- [6] Decreto 65/2022, por el que se establecen para la Comunidad de Madrid la ordenación y el currículo de la Educación Secundaria Obligatoria. (20 de julio de 2022, publicado en BOCM núm. 176, de 26 de julio de 2022, páginas 396 a 716) <https://bocm.es>.
- [7] Instituto Nacional de Tecnologías Educativas y de Formación del Profesorado. (2024). *¿Qué es #CompDigEdu? - INTEF*. <https://intef.es/>.
- [8] Chicano Capelo, I. (2020). *VS Code 4 Teaching: Los ejercicios directos al editor*.
- [9] Rivas Alcobendas, Á. J. (2021). *VSCode4Teaching 2.0: Seguimiento de ejercicios de alumnos en el IDE del profesor*.
- [10] Guerrero Carrasco, D. (2024). *VSCode4Teaching: mantenimiento y evolución de la herramienta para la enseñanza de la programación en línea*.
- [11] JetBrains, Inc. (2023). *Developer State Survey 2023*. <https://jetbrains.com>.

- [12] JetBrains, Inc. (2024). *Developing a Plugin - IntelliJ Platform Plugin SDK*. <https://jetbrains.com>.
- [13] Google, Inc. (2024). *What is Angular? - Angular*. <https://angular.dev>.
- [14] Davidson, T. (28 de febrero, 2023). Single Page Application (SPA) vs Multi Page Application (MPA): Which is the best? *CleanCommit*. <https://cleancommit.io>.
- [15] Google, Inc (2024). *Composing with Components - Angular*. <https://angular.dev>.
- [16] Mozilla Developer Network (2024). *Web Components*. <https://mozilla.org>.
- [17] Stack Overflow. (2023). *Stack Overflow Developer Survey 2023*. <https://subsec:tecAppAngular.co>.
- [18] Reis, J; Figueiredo, R. (15 de abril, 2024). Angular vs React: a comparison of both frameworks. *ImaginaryCloud*. <https://imaginarycloud.com>.
- [19] Meta Open Source (2024). *React*. <https://react.dev>.
- [20] Preston-Werner, T. (2023). *Semantic Versioning 2.0.0*. <https://semver.org>.
- [21] StatCounter. (2023). *Desktop browser market share worldwide*. <https://statcounter.com>.
- [22] Apple, Inc. (2024). *WebKit*. <https://webkit.org>.
- [23] Mozilla Foundation. (2024). *Gecko - Firefox Source Docs documentation*. <https://mozilla.org>.
- [24] Google, Inc. (2024). *Chromium*. <https://chromium.org>.
- [25] Google, Inc. (2024). *Blink (Rendering Engine)*. <https://chromium.org>.
- [26] Rajaa, R. (17 de enero, 2020). Microsoft Edge 79: What's new in the Chromium-based Edge? *BrowserStack*. <https://chromium.org>.
- [27] Mozilla Foundation. (2024). *SpiderMonkey - Firefox Source Docs documentation*. <https://mozilla.org>.
- [28] Google, Inc. (2024). *V8 JavaScript engine*. <https://v8.dev>.
- [29] Web Hypertext Application Technology Working Group. (24 de enero, 2024). *File System API living standard*. <https://whatwg.org>.
- [30] Web, Incubator Community Group. (2024). *Web, Incubator Community Group*. <https://wicg.io>.

BIBLIOGRAFÍA

- [31] Web, Incubator Community Group. (20 de marzo, 2024). *File System Access* (draft community group report). <https://wicg.github.io>.
- [32] LePage, P., Steiner, T. (2020). *The File System Access API: simplifying access to local files*. <https://developer.chrome.com>.
- [33] Lepage, P., Kruisselbrink, M. (2020). *GoogleChromeLabs/text-editor - HTML5 Text Editor*. (Repositorio de código en GitHub). <https://github.com>.
- [34] Sharp, R. (8 de octubre, 2010). *What is a Polyfill?* <https://remysharp.com>.
- [35] Sorhus, S. (29 de septiembre, 2016). *sindresorhus/ponyfill - Ponyfill*. (Repositorio de código en GitHub). <http://ponyfill.com>.
- [36] Steiner, T. et al. (2020). *GoogleChromeLabs/browser-fs-access - Browser-FS-Access: File System Access API with legacy fallback in the browser*. (Repositorio de código en GitHub). <https://github.com>.
- [37] Otta, M., Martin, E. (23 de mayo, 2024). *What is Reactive Programming? Baeldung*. <https://baeldung.com>.
- [38] Gamma, E., Helm. R et al. (1995). *Observer*. En *Design Patterns* (293-303). Addison-Wesley.
- [39] ReactiveX. (2024). *RxJS - Introduction*. <https://rxjs.dev>.
- [40] ReactiveX. (2024). *ReactiveX - Intro*. <https://reactivex.io>.
- [41] Mozilla Foundation. (8 de agosto, 2023). *Promise - JavaScript*. <https://mozilla.org>.
- [42] ECMA International. (Junio, 2015). *ECMA-262 Standard: ECMAScript 2015 Language Specification*. <https://ecma-international.org>.
- [43] Microsoft, Inc. (2024). *TypeScript: JavaScript with Syntax for Types*. <https://typescriptlang.org>.
- [44] OpenJS Foundation. (2024). *Node.js - Run JavaScript Everywhere*. <https://nodejs.org>.
- [45] Tiobe Software BV. (1 de junio, 2024). *TIOBE Index - June 2024 - TIOBE*. <https://tiobe.com>.
- [46] npm, Inc. (2024). *npm About*. <https://npmjs.com>.
- [47] World Wide Web Consortium (2024). *Cascading Style Sheets - Specs*. <https://w3.org>.

- [48] The Sass Team (2024). *Syntactically Awesome Style Sheets - Syntax*. <https://sass-lang.com>.
- [49] The Bootstrap Team (2024). *Bootstrap*. <https://getbootstrap.com>.
- [50] Downie, N. (2013). *Chart.js*. <https://www.chartjs.org>.
- [51] Oracle Corporation. (2024). *MySQL Community Edition*. <https://mysql.com>.
- [52] PostgreSQL Global Development Group. (2024). *PostgreSQL: About*. <https://postgresql.org>.
- [53] Oracle Corporation. (2024). *What is Java and why do I need it?* <https://java.com>.
- [54] Broadcom, Inc. (2024). *Spring Framework*. <https://spring.io>.
- [55] Broadcom, Inc. (2024). *Spring Boot*. <https://spring.io>.
- [56] Broadcom, Inc. (2024). *Spring Data*. <https://spring.io>.
- [57] Apache Software Foundation. (2024). *Maven - Welcome to Apache Maven*. <https://apache.org>.
- [58] JUnit Team. (2024). *JUnit 5 User Guide*. <https://junit.org>.
- [59] Open JS Foundation. (2024). *Jest - Delightful JavaScript Testing*. <https://jestjs.io>.
- [60] GitHub, Inc. (2024). *About GitHub and Git - GitHub Docs*. <https://github.com>
- [61] GitHub, Inc. (2024). *About GitHub*. <https://github.com/about>
- [62] Docker, Inc. (2024). *Docker: Accelerated, Containerized Application Development*. <https://docker.com>
- [63] Docker, Inc. (2024). *Docker Engine overview*. <https://docker.com>
- [64] Docker, Inc. (2024). *What is a Container?* <https://docker.com>
- [65] Docker, Inc. (2024). *Packaging your software: Dockerfile* <https://docker.com>
- [66] Docker, Inc. (2024). *Docker Compose overview*. <https://docker.com>.
- [67] Docker, Inc. (2024). *Docker Hub overview*. <https://docker.com>.
- [68] Microsoft, Inc. (2024). *Managing extensions in Visual Studio Code: Visual Studio Code Marketplace*. <https://visualstudio.com>.

BIBLIOGRAFÍA

- [69] JetBrains, Inc. (2024). *WebStorm: the JavaScript and TypeScript IDE.* <https://jetbrains.com>.
- [70] Microsoft, Inc. (2024). *Visual Studio Code: Code editing. Redefined.* <https://visualstudio.com>.
- [71] JetBrains, Inc. (2024). *IntelliJ IDEA: the Leading Java and Kotlin IDE.* <https://jetbrains.com>.
- [72] Software Freedom Conservancy. (2024). *Git.* <https://git-scm.com>.
- [73] Atlassian. (2024). *What is DevOps.* <https://atlassian.com>.
- [74] Red Hat, Inc. (December 12, 2023). *What is CI/CD?* <https://redhat.com>.
- [75] GitLab B.V. (2024). *What is CI/CD?* <https://gitlab.com>.
- [76] GitHub, Inc. (2024). *GitHub Actions.* <https://github.com>.
- [77] Atlassian. (2024). *What is Trello: learn features, uses & more.* <https://trello.com>.
- [78] Boehm, B. (1 de agosto, 1986). “*A spiral model of software development and enhancement*”. *ACM SIGSOFT Software Engineering Notes*, vol. 11 (nº 2), pp. 22-42. <https://acm.org>.
- [79] Booch, G., Jacobson, I., Rumbaugh, J. (1999). *The Unified Software Development Process.*
- [80] Booch, G., Jacobson, I., Rumbaugh, J. (2005). *The Unified Modeling Language User Guide (2nd edition).*
- [81] Beck, K. (1999). *Extreme Programming explained: embrace change.*
- [82] Beck, K., et al. (2001). *Manifesto for Agile Software Development.* <https://agilemanifesto.org>.
- [83] Sergeev, A. (26 de mayo de 2016). *Extreme programming user stories.* <https://hygger.io>.
- [84] Docker, Inc. (2024). *Multi-stage builds* <https://docker.com>.
- [85] Fowler, M. (2002). *Patterns of enterprise application architecture.* Addison-Wesley Professional. <https://oreilly.com>.
- [86] Red Hat, Inc. (21 de diciembre, 2023). *Stateful vs stateless.* <https://redhat.com>.
- [87] Martin, R. C. (2009). *Unit Tests.* En *Clean Code* (121-134). Prentice Hall.

- [88] Apache Software Foundation. (2004). *Apache License 2.0.* <https://apache.org>.
- [89] Free Software Foundation, Inc. (1996). *¿Qué es el software libre?* <https://gnu.org>.
- [90] Microsoft, Inc. (2024). *Monaco Editor.* <https://microsoft.github.io>.
- [91] Moodle. *Moodle - Open-source learning platform.* <https://moodle.org>.
- [92] 1EdTech. (16 de abril de 2019). *Learning Tools Interoperability Core Specification* (versión 1.3). <https://imsglobal.org>.

Anexos

A

Detalle del algoritmo de sincronización de ejercicios

Descendiendo al bajo nivel, la sincronización automática del directorio de cada ejercicio se basa en un algoritmo que realiza un recorrido en profundidad de la carpeta para dar lugar a una estructura arborescente que es comparada con su versión anterior para detectar las divergencias, que se interpretan como creaciones, modificaciones o eliminaciones, añadiéndolas en una cola de prioridad que alimenta un procedimiento que lanza los trabajos encolados para remitirlos al servidor. Se analizan a continuación las distintas partes de este algoritmo.

El primer punto del algoritmo es el recorrido en profundidad del directorio para dar lugar a una estructura arborescente comparable. Tal como aborda la sección 3.1.1.2, la *File System Access API* es la interfaz que permite utilizar el directorio asignado por el estudiante para acceder a los contenidos específicos de un ejercicio y poder explorarlos en profundidad. La estructura que se obtiene cuando un estudiante proporciona un directorio local en un navegador compatible con la API es directamente recorrible de forma recursiva, ya que tiene aspecto de árbol. El código A.1 es el procedimiento implementado para esta casuística. Sin embargo, cuando se emplea el mecanismo sustitutivo para navegadores no compatibles procedente del *ponyfill* empleado —de utilización justificada en la citada sección—, la estructura de ficheros contenidos recursivamente dentro del directorio se obtiene de forma plana, y debe ser interpretada para obtenerse un árbol basado en las rutas relativas de cada fichero obtenido, tal como se hace en el código A.2.

Sin embargo, este doble algoritmo tiene una limitación: cuando se utiliza un navegador compatible con la *File System Access API*, se permite al navegador acceder en tiempo real a la completitud de ficheros y directorios incluidos recursivamente dentro de la carpeta autorizada con independencia de cuándo el usuario dio su autorización, mientras que el mecanismo alternativo del *ponyfill* únicamente concede acceso a una instantánea, devolviendo los ficheros existentes en el momento en el que el usuario autorizó la visualización de los contenidos de una carpeta. Como consecuencia de esta limitación, se ha decidido permitir realizar la sincronización únicamente en el caso de los navegadores compatibles y mantener la implementación alternativa para activar eventualmente un mecanismo de sincronización manual de los ficheros de un ejercicio.

```

1  public async fsaAPI(curDirectory: FileSystemDirectoryHandle, parent?:  
2   DirectoryNode): Promise<DirectoryNode | undefined>  
3 {  
4   const childrenNodesList: Node[] = [];  
5   const directoryNode: DirectoryNode = new DirectoryNode({  
6     name: curDirectory.name,  
7     children: childrenNodesList,  
8     parentDirectoryNode: parent  
9   });  
10  for await (const entry of curDirectory.values()) {  
11    if (entry instanceof FileSystemFileHandle) {  
12      let fileInformation = await entry.getFile();  
13      childrenNodesList.push(new FileNode({  
14        name: entry.name,  
15        lastModifiedTime: fileInformation.lastModified,  
16        fileBlob: fileInformation,  
17        parentDirectoryNode: directoryNode  
18      }));  
19    } else if (entry instanceof FileSystemDirectoryHandle) {  
20      const subdirInfo = await fsaAPI(entry, directoryNode);  
21      if (subdirInfo !== undefined) childrenNodesList.push(subdirInfo);  
22    }  
23  }

```

Código A.1: Método para el recorrido en profundidad de un directorio mediante la *File System Access API*.

```

1  public noFsaAPI(recursive fileList: File[]): DirectoryNode | undefined  
2 {  
3   if (recursive.length === 0) return undefined;  
4   const root = new DirectoryNode({  
5     name: recursive[0].webkitRelativePath.split(/\/|\\/) [0],  
6     children: []  
7   });  
8   for (const handler of recursive) {  
9     if (!handler.webkitRelativePath || handler.webkitRelativePath === "")  
10       return undefined;  
11     noFsaAPIRecursive(handler, root);  
12   }  
13   return root;  
14 }  
15  
16 private noFsaAPIRecursive(file: File, dirNode: DirectoryNode, relPath?:  
17   string[])  
18 {  
19   let path = relPath ?? file.webkitRelativePath.split(/\/|\\/).slice(1);  
20   if (path.length === 1) {  
    dirNode.children.push(new FileNode({

```

```

21         name: path[0],
22         lastModifiedTime: file.lastModified,
23         fileBlob: file,
24         parentDirectoryNode: dirNode
25     }));
26 } else {
27     const children = dirNode.children.filter(child => child.name ===
path[0]);
28     if (children.length === 1) {
29         noFsaAPIRecursive(file, children[0] as DirectoryNode,
path.slice(1));
30     } else {
31         const subdirNode = new DirectoryNode({
32             name: path[0],
33             children: [],
34             parentDirectoryNode: dirNode
35         });
36         dirNode.children.push(subdirNode);
37         noFsaAPIRecursive(file, subdirNode, path.slice(1));
38     }
39 }
40 }
```

Código A.2: Procedimiento para la interpretación de la lista de ficheros obtenida mediante el *ponyfill* como un árbol comparable.

La divergencia estructural anterior ha conducido a la generación de varios tipos abstractos para modelar un árbol de ficheros y directorios mediante una interfaz `Node` y las clases derivadas `FileNode` (cuyas instancias representan los ficheros y componen la frontera del árbol) y `DirectoryNode` (para directorios), siendo los tipos devueltos por los anteriores algoritmos. Estos tipos permiten, además, implementar un algoritmo de comparación de dos árboles dados por sus nodos `DirectoryNode` raíces de modo que, dados dos árboles A y B, se obtengan tres listas: los nodos de A no presentes en B (eliminaciones), los nodos de B no presentes en A (creaciones) y los nodos que, estando presentes en ambas estructuras, tengan datos diferentes (modificaciones). Así, se implementa un algoritmo que permite comparar dos estructuras realizadas sobre un mismo directorio en instantes temporales diferentes. Este algoritmo queda plasmado en el [código A.3](#).

```

1 public dirDiff(old: DirectoryNode, new: DirectoryNode): Diff {
2     let diff: Diff = new Diff();
3     let o, n: number = 0;
4     while (o < old.children.length || n < new.children.length) {
5         if (o < old.children.length && n < new.children.length
6             && old.children[o].name === new.children[n].name
7         ) {
8             if (old.children[o] instanceof FileNode
9                 && new.children[n] instanceof FileNode
10            ) {
11                 const oldFileNode = old.children[o] as FileNode;
12                 const newFileNode = new.children[n] as FileNode;
13                 if (oldFileNode.lastModifiedTime < newFileNode.lastModifiedTime)
14                     diff.addModified(newFileNode);
15             } else if (
16                 old.children[o] instanceof DirectoryNode
17                 && new.children[n] instanceof DirectoryNode
18            ) {
19                 const subDirDiff = dirDiff(
20                     <DirectoryNode>old.children[o],
21                     <DirectoryNode>new.children[n]
```

```

22         );
23         diff.addCreated(...subdirDiff.created);
24         diff.addDeleted(...subdirDiff.deleted);
25         diff.addModified(...subdirDiff.modified);
26     } else {
27         diff.addDeleted(old.children[o]);
28         diff.addCreated(new.children[n]);
29     }
30
31     o++;
32     n++;
33 } else {
34     if (o >= old.children.length) {
35         diff.addCreated(new.children[n]);
36         n++;
37     } else if (n >= new.children.length) {
38         diff.addDeleted(old.children[o]);
39         o++;
40     } else {
41         if (old.children[o].name.localeCompare(new.children[n].name)) {
42             diff.addCreated(new.children[n]);
43             n++;
44         } else {
45             diff.addDeleted(old.children[o]);
46             o++;
47         }
48     }
49 }
50
51 return diff;
52 }

```

Código A.3: Algoritmo recursivo para la comparación de dos árboles de directorios dados dos nodos raíces `old` y `new`.

Una vez ejecutado el anterior algoritmo, se obtienen las creaciones, modificaciones y eliminaciones producidas entre los dos árboles comparados. Estas listas de elementos a crear, modificar y eliminar dan lugar a una lista de “trabajos de sincronización” que se añaden a una cola de prioridad. La cola empleada es de implementación propia y dispone de dos niveles de prioridad. Los trabajos de eliminación tienen prioridad sobre los de creación y modificación porque no requieren del envío del cuerpo de un fichero completo sino de, únicamente, la ruta relativa del fichero que debe ser eliminado. Una vez encolados los trabajos, da comienzo el algoritmo que toma trabajos de la cola y los envía al servidor, tal como se aprecia en el [código A.4](#). Este procedimiento viene controlado por un booleano que actúa como mecanismo de exclusión, previniendo su ejecución paralela más de una vez para evitar posibles duplicidades en el envío de peticiones. Una vez ejecutado, envía todos los trabajos encolados pendientes, desencolándolos uno a uno y gestionando su transmisión según su tipo. Es un algoritmo bloqueante, deteniendo la ejecución del código hasta ver resuelta cada petición enviada al servidor, garantizando la no concurrencia de peticiones para tratar de evitar saturar la comunicación con el servidor. Emplea, además, un método auxiliar que permite manejar los eventos HTTP ocurridos durante el transcurso de la petición para reflejar en la interfaz de usuario el progreso en la ejecución de cada petición, permitiendo “aplanar” los observables empleados para utilizarlos como promesas.

```

1  public async serverSync(): Promise<void> {
2      if (!activeSyncSemaphore) {
3          while (syncJobs.pending.pendingElements() !== 0) {
4              activeSyncSemaphore = true;
5              let newSyncWork = syncJobs.pending.dequeue();
6              if (newSyncWork === undefined) break;
7              let current = newSyncWork;
8              currentSyncStatus = "SENDING_FILES";
9              let fileReqPromise: Promise<any> = Promise.resolve();
10             if (current.type === "CREATION") {
11                 fileReqPromise = httpEventObservableAsPromise(
12                     createFileByExerciseIdRelativePath(
13                         eui.exercise.id, current.relativePath, current.fileBlob
14                     )
15                 );
16             } else if (current.type === "MODIFICATION") {
17                 fileReqPromise = httpEventObservableAsPromise(
18                     fileExchangeService.editFileByExerciseIdRelativePath(
19                         eui.exercise.id, current.relativePath, current.fileBlob
20                     )
21                 );
22             } else if (current.type === "DELETION") {
23                 fileReqPromise = httpEventObservableAsPromise(
24                     fileExchangeService.deleteFileByExerciseIdRelativePath(
25                         eui.exercise.id, current.relativePath
26                     )
27                 );
28             }
29             if (fileReqPromise !== undefined) {
30                 eui.modifiedFiles = [current.node.relativePath];
31                 await euiService.editInfoByExercise(eui.exercise, eui);
32                 await httpEventPromiseFinalizationHandler(fileReqPromise);
33             }
34         }
35         currentSyncStatus = "WAITING_FOR_CHANGES";
36     }
37     activeSyncSemaphore = false;
38 }

```

Código A.4: Algoritmo que remite trabajos de sincronización al servidor a partir de los contenidos de una cola de prioridad.

Este algoritmo queda orquestado para su ejecución completa de forma automática cada 500 milisegundos, obteniendo un nuevo árbol, encolando trabajos de sincronización según el tipo de diferencias encontradas respecto al árbol anterior y lanzando el procedimiento para la sincronización con el servidor. Cada ejecución reemplaza el árbol anterior por el nuevo obtenido, minimizando las diferencias encontradas en cada ejecución.