UOC Submission System

Plugin para IDEs de JetBrains

Alberto Lago Mollinedo

UOC Submission System
Aplicaciones Multimedia de Nueva Generación

Tutor/a de TF
David García Solórzano
Profesor/a responsable de la asignatura
David García Solórzano

Junio 2023

Ficha del Trabajo Final

Título del trabajo:	UOC Submission System
Nombre del autor:	Alberto Lago Mollinedo
Nombre del Tutor de TF:	David García Solórzano
Nombre del/de la PRA:	David García Solórzano
Fecha de entrega:	06/2023
Titulación o programa:	Ingeniería de Tecnologías y Servicios de Telecomunicación
Área del Trabajo Final:	Aplicaciones Multimedia de Nueva Generación
Idioma del trabajo:	Castellano
Palabras clave	JetBrains, plugin, UOC

Resumen del Trabajo

El objetivo de este proyecto es desarrollar y documentar un *plugin* para los IDEs de JetBrains, que permitirá exportar los proyectos asignados a los alumnos de la UOC, ya sea a un servidor remoto o al disco duro local. Este proyecto también incluye el desarrollo del servidor.

Además, las acciones más relevantes realizadas por el usuario durante su trabajo deben ser registradas automáticamente y almacenadas en un archivo para su posterior análisis por parte del profesorado. Para garantizar el uso del *plugin*, se establecerá un mecanismo que consiste en cifrar los proyectos con el propio *plugin*, los cuales solo podrán ser descifrados por el mismo al abrirlos en el IDE. Al cerrar el proyecto, este volverá a cifrarse automáticamente.

El desarrollo se realizará haciendo uso de la API IntelliJ Platform Plugin SDK.

Abstract

The aim of this project is to develop and document a plugin for JetBrains IDEs, which will allow exporting of projects assigned to UOC students, either to a remote server or local hard drive. This project also includes server development.

In addition, the most relevant actions performed by the user during their work should

be automatically recorded and stored in a file for subsequent analysis by the teaching staff. To ensure the use of the plugin, a mechanism will be established that involves encrypting the projects with the plugin itself, which can only be decrypted by it when opened in the IDE. Upon closing the project, it will be encrypted again automatically.

The development will be carried out using the IntelliJ Platform Plugin SDK API.

Tabla de contenido

Lista de Figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

La versatilidad de los IDEs de JetBrains se debe en gran parte a su capacidad de admitir *plugins*[1]. Estos permiten una mayor personalización y expansión de las funcionalidades del entorno de desarrollo. Pueden brindar soporte para una variedad de lenguajes de programación, integrarse con herramientas para pruebas unitarias, proporcionar asistencia para la refactorización de código y facilitar el análisis del código.

Además de estas funcionalidades, existen *plugins* enfocados en promover la programación colaborativa y la educación. Se ha observado un crecimiento notable en el uso de *plugins* en el sector educativo en los últimos años. Ofrecen numerosas ventajas, como la facilitación del aprendizaje y la colaboración en proyectos de programación.

Para el desarrollo de estos *plugins*, JetBrains ha desarrollado una herramienta denominada IntelliJ Platform Plugin SDK[1]. Esta API ofrece una gama amplia de posibilidades, haciendo más accesible y diversa la creación de *plugins*.

Con este Trabajo Final de Grado (TFG) se diseñará un *plugin*, para IDEs de JetBrains, que pretende facilitar una tarea habitual de estudiantes y profesores, por esta razón se puede considerar de gran interés y relevancia en el ámbito académico. En la actualidad, la enseñanza online y a distancia se ha convertido en una tendencia en crecimiento, especialmente en un mundo cada vez más digitalizado. En este contexto, la UOC, universidad líder en el campo de la educación *online*, cuenta con un campus muy avanzado que es capaz de soportar una alta carga de trabajo. Para ofrecer una educación de calidad en este entorno, es crucial contar con tecnologías de vanguardia que permitan a los estudiantes y profesores llevar a cabo sus tareas de manera efectiva y eficiente.

El desarrollo de este TFG podría suponer un impacto directo en la mejora de la educación en el campo de la programación, además de producir un efecto muy positivo en el desarrollo de las habilidades tecnológicas de los estudiantes. Por otro lado, esta tecnología podría ser utilizada como referencia para futuros proyectos y contribuir así a la innovación en un campo relativamente poco desarrollado y con escasa documentación.

Se puede destacar, respecto a las opciones similares presentes en el mercado, que este plugin será el primero que realice un seguimiento y un registro de las acciones del usuario para su posterior análisis.

1.2. Objetivos del Trabajo

El objetivo consiste en el desarrollo de un plugin plenamente operativo. Además, se requiere el desarrollo de un servidor para habilitar todas las funcionalidades previstas. El plugin deberá tener la capacidad de:

- Exportar los proyectos al disco duro.
- Exportar proyectos al servidor.
- Implementar un sistema de cifrado en los proyectos que garantice su funcionalidad solo cuando el *plugin* esté instalado.
- Registrar y almacenar la actividad de los usuarios para su posterior análisis.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

Se consideran las tres dimensiones:

- Sostenibilidad.

  El TFG no tiene impacto directo relacionado con aspectos relacionados con la sostenibilidad medioambiental. Sin embargo, indirectamente si lo tiene debido a que se trata una herramienta que apoya la formación a distancia, hecho que ayuda a reducir la huella ecológica al evitar las consecuencias derivadas de la formación presencial cuando ello implica desplazamientos, cambios de residencia, etc.

- Comportamiento ético y responsabilidad social.

  El tema del Trabajo no afecta en ningún sentido a aspectos ético-sociales.

- Diversidad y derechos humanos.

  Este proyecto no tiene impacto negativo alguno en aspectos relacionados con este ámbito. Se trata de una pieza de software accesible a usuarios de todo tipo. En ningún momento se le solicitan al usuario datos personales como la ideología, raza, religión, etc.

1.4. Enfoque y método seguido

La metodología seguida a lo largo del proyecto comprende los pasos descritos a continuación:

- Inicialmente, se realizó un estudio detallado de la estructura básica de un *plugin*, un paso crucial debido a la escasa documentación y ejemplos existentes en este campo.
- Se desarrollaron los paneles gráficos correspondientes al menú.
- Se implementó la capacidad de empaquetar el proyecto abierto por el usuario y de guardarlo en el disco duro.
- En la siguiente etapa se añadieron las funciones de autentificación del usuario, permitiendo que tenga un compartimento determinado por el tipo de usuario, administrador o usuario estándar.
- Incorporación de *listeners* para registrar eventos relacionados con la actividad del usuario.
- Se añadieron capacidades de cifrado y de identificación de proyectos para determinar cuando el plugin debe actuar o no.
- Implementación de funciones relacionadas con la lectura y escritura del archivo de registro de actividad del usuario.
- Se realizaron pruebas de compatibilidad. Hasta este punto solamente se habían realizado pruebas con IntelliJ, por lo tanto, antes de añadir más funciones, fue necesario verificar el funcionamiento con otras IDEs de JetBrains.
- Se añadió la capacidad de registrar tipos de eventos adicionales.
- Finalmente, se refinó el código, se realizaron más pruebas y se mejoraron los comentarios explicativos en el propio código.

1.5. Planificación del trabajo

- PEC 1: Definición de la temática principal del trabajo, el alcance del proyecto y los objetivos que se quieren lograr con su realización.

- PEC 2: Realización del estado del arte del proyecto. Se lleva a cabo la recolección de datos sobre el área de interés.
- PEC 3: Diseño e implementación del proyecto.
- PEC 4: Elaboración de la memoria del Trabajo, para ello se tienen en cuenta los trabajos realizados en las PECs anteriores.
- PEC 5: Presentación.
- PEC 6: Defensa.

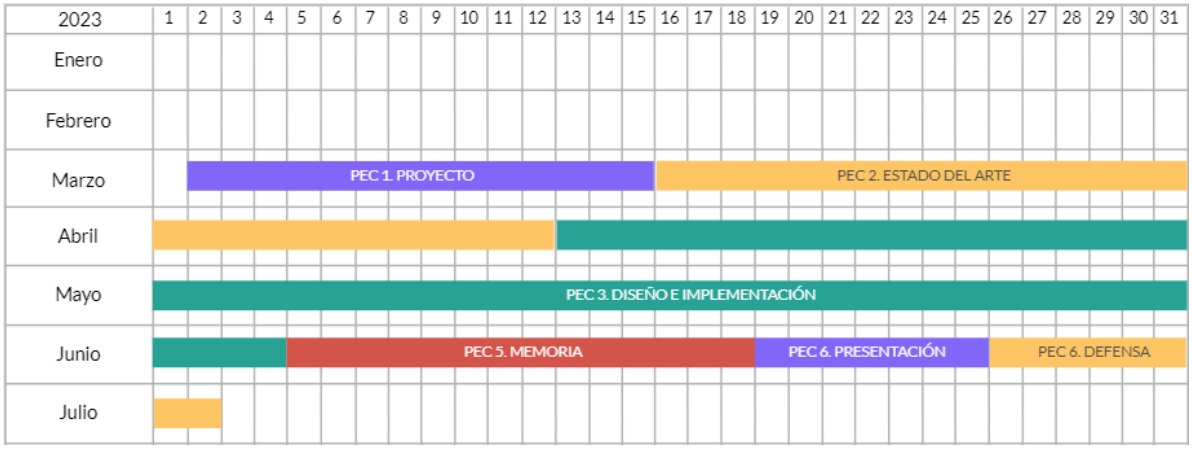

Figura 1: Diagrama de entregas.

1.6. Breve sumario de productos obtenidos

Elementos de la entrega final:

- Código fuente y compilación del *plugin*.
- Código fuente y compilación del servidor.
- Documentación del código en formato Javadoc.
- Memoria del TFG.

- Diagrama de clases.

1.7. Breve descripción de otros capítulos de la memoria

- Capítulo 2

  Estado del arte. Se realiza un repaso del *software* similar que existe actualmente en el mercado.

- Capítulo 3

  Se expone el diseño elegido y la implementación realizada.

- Capítulo 4

  Se describen los resultados obtenidos. Para ello se detalla el uso por parte de los dos posibles tipos de usuario.

- Capítulo 5

  Conclusiones. Se mencionan las funcionalidades descartadas y las futuras líneas de trabajo.

2. Estado del arte

2.1. Perspectiva general

Los *plugins* de JetBrains representan un significativo avance en la forma en que profesionales y estudiantes interactúan con los lenguajes de programación y los entornos de desarrollo. Muchos incluyen funcionalidades que simplifican y enriquecen el proceso de enseñanza y aprendizaje en programación, como el resaltado de la sintaxis del código, apoyo en tiempo real y corrección automática. Algunos facilitan la compartición de código en tiempo real, un recurso de gran valor para la programación colaborativa.

En la misma línea, existen *plugins* específicamente orientados hacia la educación, que pueden ofrecer la posibilidad de crear cursos interactivos de programación, realizar pruebas y seguimientos del progreso de los estudiantes. Estas herramientas se encuentran en constante evolución para responder a las demandas cambiantes del sector educativo.

Gracias a la IntelliJ Platform Plugin SDK, una API desarrollada por JetBrains, se ha logrado un significativo crecimiento y expansión en el ámbito de los *plugins*, lo cual ha permitido a los desarrolladores extender y personalizar las funcionalidades de los IDEs de JetBrains, generando un sinfín de *plugins* cada vez más especializados. Así, el panorama actual de los *plugins* para los IDEs de JetBrains en el sector educativo es un campo joven, pero en constante desarrollo, lleno de posibilidades de innovación y mejoras.

2.2. Revisión de plugins relevantes

Al examinar el mercado actual orientado al ámbito educativo, encontramos pocos casos que guarden similitudes con el plugin desarrollado en este proyecto. A continuación, se analizan varios *plugins* relevantes y sus aplicaciones en el contexto académico.

- PyPhanon[2]

  Desarrollado por la Universidad Estatal de Utah (USU). Registra el progreso de los estudiantes en sus tareas. Recurre a archivos CSV para almacenar información sobre los puntos más problemáticos para los estudiantes, de modo que, posteriormente, el profesorado pueda actuar en consecuencia. Esta función está relacionada con una de las funciones que incorporará el plugin desarrollado en este TFG.

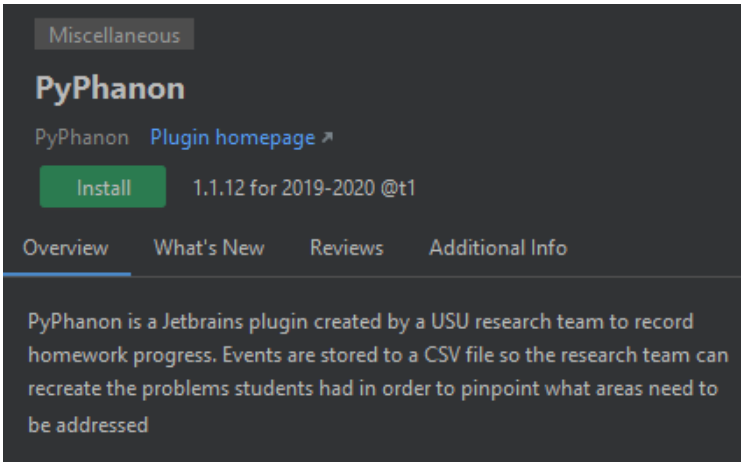

Figura 2: PyPhanon en el instalador de *plugins* de IntelliJ.

- ShowYourWork[3]

  Desarrollado por la Universidad Estatal de Utah (USU) para PyCharm, tiene como objetivo registrar el progreso en las tareas escolares. Realiza un seguimiento de los avances por parte de los alumnos y los guarda en forma de eventos, de este modo facilita a los docentes la compresión sobre cómo sus alumnos desarrollan el código. Implementa una base de datos SQLite.

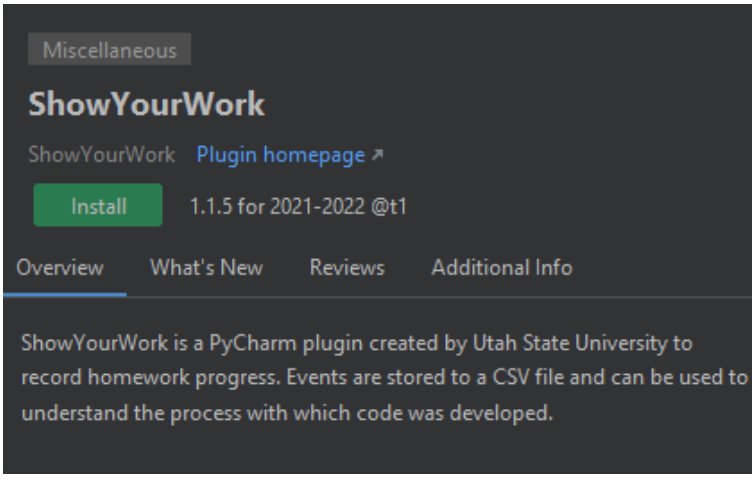

Figura 3: ShowYourWork en el instalador de *plugins* de IntelliJ.

- CSE335 Course Extensions[4]

  La Universidad del Estado de Míchigan (MSU) ofrece cursos de formación de desarrollo de software mediante programación orientada a objetos. Como soporte para estos cursos este interesante *plugin* facilita la entrega de proyectos creados en el IDE CLion, en concreto, es posible exportar los proyectos a formato ZIP o, alternativamente, realizar la entrega al servidor de forma remota desde el mismo IDE, para ello, los alumnos deben introducir previamente sus credenciales. Adicionalmente, el plugin realiza una revisión del código en busca de errores habituales.

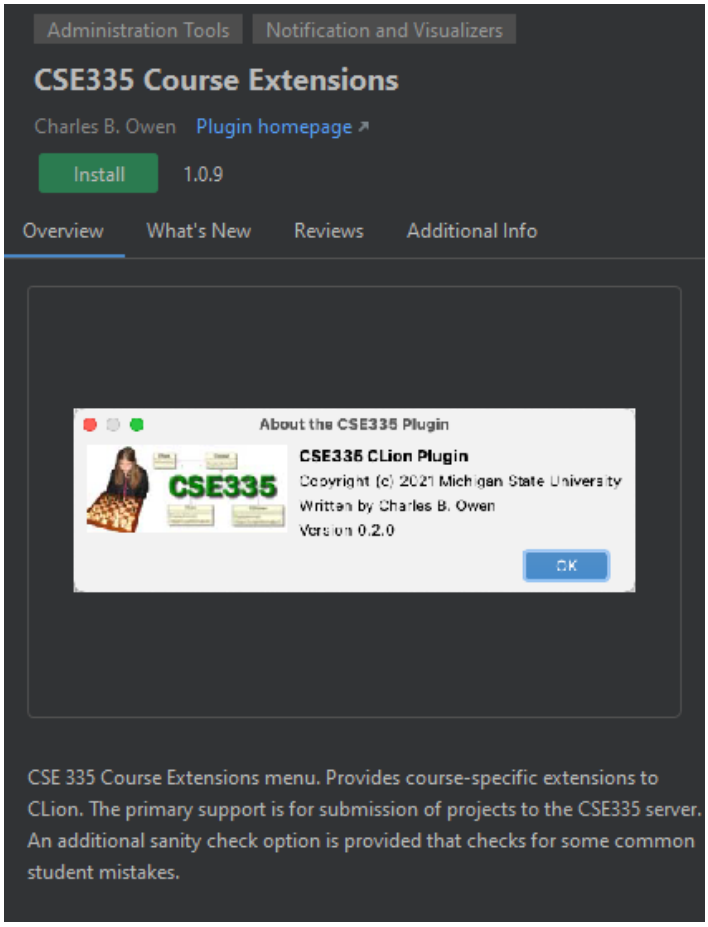

Figura 4: CSE335 Course Extension en el instalador de *plugins* de IntelliJ.

3. Diseño

3.1. Comportamiento y descripción

Para el desarrollo de un nuevo *plugin* relacionado con el ámbito educativo y universitario, es importante tener en cuenta que debe ofrecer una gran facilidad de uso y una buena experiencia de usuario. Estudiar y aprender de los ejemplos mencionados anteriormente, puede proporcionar una base sólida para el diseño y desarrollo de un nuevo plugin que mejore la experiencia educativa.

Uno de los principales objetivos es realizar un registro de la actividad del usuario y almacenarlo en el archivo `uoc.data`. Serán registradas las siguientes acciones:

- Apertura del proyecto.
- Cierre del proyecto.
- Modificaciones de archivos indicando el número de fila modificada.
- Código pegado, mostrando en el registro el código que ha sido pegado.
- Eliminación de archivos.
- Creación de archivos.

Para todas ellas, se indicará la fecha y la hora.

Otras funciones incorporadas:

- Posibilidad de enviar el proyecto directamente desde el IDE al servidor.
- Comprimir el proyecto en formato ZIP.
- Cifrado de los proyectos.
- Descifrado de los proyectos.

3.2. Estructura e implementación

La estructura del proyecto se define en el siguiente esquema:

```
UOCSubmissionSystem
├── .run
│   ├── Run IDE with Plugin.run.xml
│   ├── Run IDE for UI Tests.run.xml
│   ├── Run Plugin Tests.run.xml
│   ├── Run Plugin Verification.run.xml
│   └── Run Qodana.run.xml
├── gradle
│   ├── wrapper
│   │   ├── gradle-wrapper.jar
│   │   └── gradle-wrapper.properties
│   └── libs.version.toml
├── src
│   ├── main
│   │   ├── java
│   │   └── resources
│   │       └── META-INF
│   │           ├── plugin.xml
│   │           └── pluginIcon.svg
│   └── test
│       ├── java
│       └── resources
├── .gitignore
├── build.gradle.kts
├── gradle.properties
├── gradlew
├── gradlew.bat
├── qodona.yml
├── README.md
└── CHANGELOG.md
```

Inicialmente, el plugin detecta si el IDE se encuentra en modo LightEdit y, si es el caso, no realiza ninguna de sus funciones. Adicionalmente, para identificar si debe actuar, realiza una búsqueda del archivo `uoc.data` en el proyecto.

Para registrar las acciones del usuario, se hace uso de *listeners* y métodos proporcionados por el IntelliJ Platform Plugin SDK.

En la tabla 1 podemos observar las clases más relevantes del proyecto, entre las cuales destacan *ProjectOpenedManager, ProjectClosedManager* y *UserActionLogger.*

Clase	Función principal
AppSettingsState	Almacenamiento persistente
UserActionLogger	Escritura en el buffer y el archivo de registro
CipherTools	Cifrado del proyecto y del archivo de registro
ProjectOpenedManager	Listeners para eventos de apertura de proyectos y de modificaciones de archivos
ProjectClosedManager	Listeners para eventos de cierre de proyectos
DirToZip	Empaquetado de proyectos en zip
ExportAction	Exportar proyectos. Superclase de ProjectToZipAction y ProjectToServerAction
ProjectToZipAction	Exportar proyectos al disco duro. Extiende ExportAction
ProjectToServerAction	Exportar proyectos al servidor. Extiende ExportAction

Tabla 1: Resumen de las clases más relevantes del proyecto.

3.3. Clase ProjectOpenedManager

La clase que gestiona la mayoría de los eventos de actividad relevantes para el plugin es la denominada *ProjectOpenedManager*. Es una de las clases más elaboradas e implementa la interfaz *StartupActivity*, una parte integral del API de IntelliJ. Esta clase también redefine el método *RunActivity*. Precisamente, es en este método donde se manejan los eventos asociados con la

apertura de proyectos y la modificación de archivos. Además, este es el lugar donde se invocan los métodos responsables de desencriptar el proyecto cada vez que se abre.

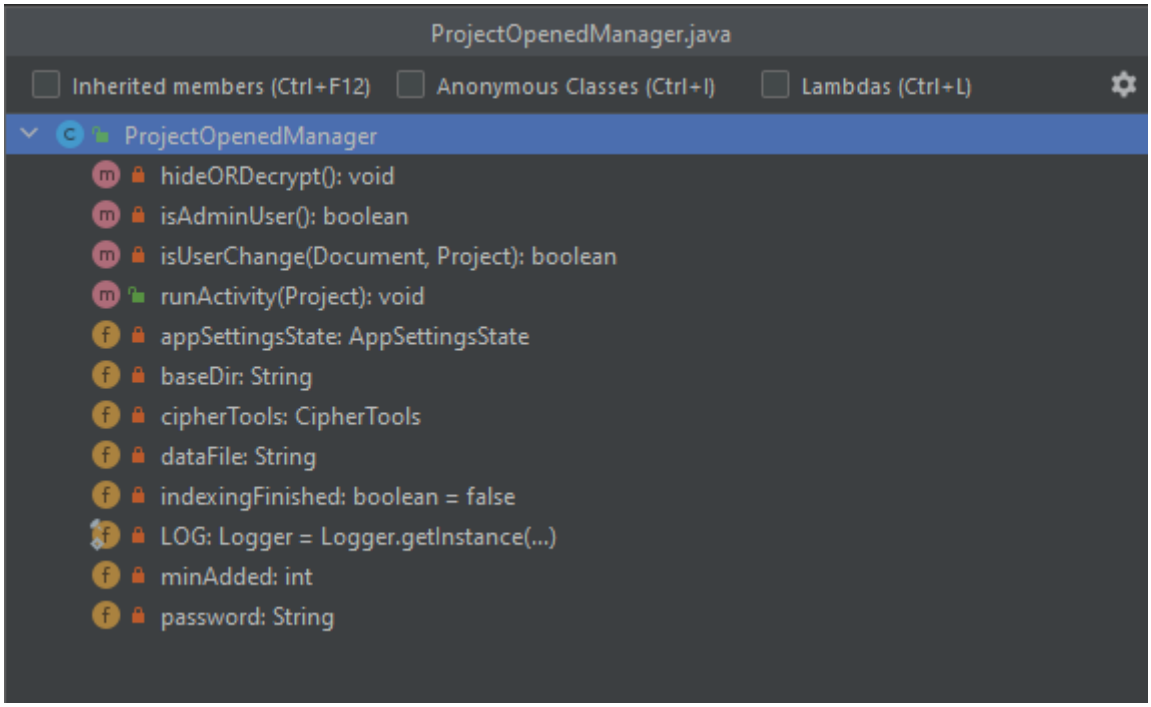

Figura 5: Variables y métodos de la clase *ProjectOpenedManager*.

Para realizar de forma correcta el registro de modificaciones de archivos, se han tenido que tomar medidas para discriminar entre las modificaciones realizadas por el IDE como parte de su funcionamiento interno, y las realizadas por el usuario.

Respecto a la detección de acciones de pegado de código por parte del usuario, al no existir en el API métodos que indiquen directamente cuando se producen, el desarrollo ha sido más complicado y se ha tenido que recurrir a un proceso de detección heurístico. Gracias al API se pueden detectar eventos de inserción de código, pero sin indicaciones sobre si ese código ha sido insertado por el usuario o por el IDE, por ejemplo, al autocompletar. Se han creado parámetros para distinguir el origen de estas inserciones y, además, en la clase *UserActionLogger* se aplican medidas adicionales para lograrlo. El objetivo es registrar las acciones de pegado cuando se trata de una cantidad relativamente grande de código. Si se cambian los parámetros para registrar cantidades menores de código se registrarán inserciones de código por parte del IDE que, no obstante, serían fácilmente reconocibles al revisar el archivo de registro. Sería recomendable un periodo de pruebas más amplio para pulir y ajustar esta función.

Considerando la gran variedad de archivos y tecnologías con las que se puede trabajar en las IDEs de JetBrains, lo más recomendable sería un ajuste conservador de los parámetros

relacionados con el registro de actividad del usuario. Por ejemplo, los archivos XML son ampliamente utilizados para configuraciones internas de los IDEs, incluirlos como archivos para tener en cuenta a la hora de registrar modificaciones por parte del usuario podría llevar a registrar archivos internos del IDE.

3.4. Clase ProjectClosedManager

La clase *ProjectClosedManager* implementa la interfaz *ProjectManagerListener* proporcionada por la API de IntelliJ IDEA. Esta clase se encarga de gestionar los eventos asociados con el cierre de proyectos. En su núcleo, *ProjectClosedManager* redefine el método *projectClosed*, que se invoca cada vez que se cierra un proyecto. Este método tiene varias responsabilidades, como el manejo de la encriptación de los archivos y el registro de los eventos de cierre de proyecto por parte del usuario si el usuario no es administrador. En el caso de que el usuario sea administrador no se realizarán dichas funciones.

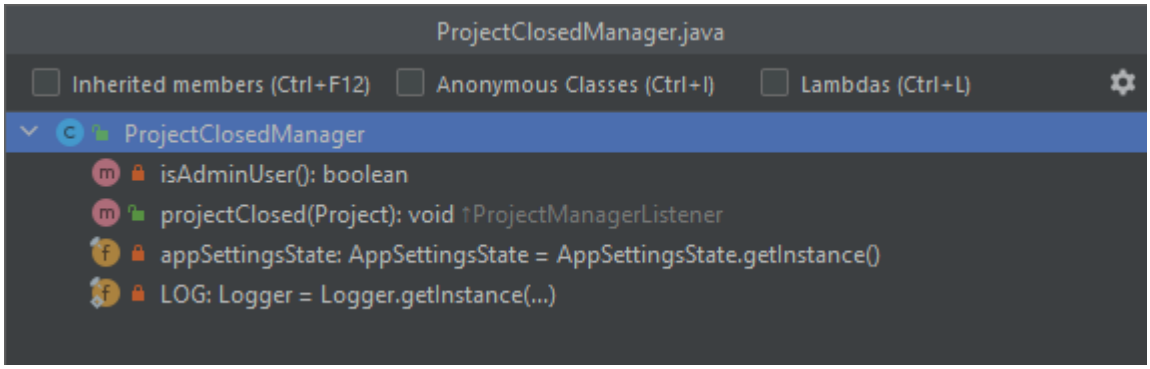

Figura 6: Variables y métodos de la clase *ProjectClosedManager*.

El registro de las acciones de apertura y de cierre de proyecto se ha podido implementar con relativa facilidad.

3.5. Clase UserActionLogger

La clase *UserActionLogger* recibe eventos que deben ser registrados y que son enviados por las clases *ProjectOpenendManager* y *ProjectClosedManager*, descritas anteriormente. Para lograrlo, los eventos se van almacenando en un *buffer* y son escritos en archivo de registro de forma periódica. El *buffer* es una variable estática, por lo tanto, solamente hay uno por cada instancia del IDE. En el caso de que se abriesen a la vez varias instancias de un IDE con distintos proyectos de la UOC, cada una tendría su propio *buffer* independiente debido a que cada instancia de un IDE de JetBrains se ejecuta en su propia JVM.

En esta clase también se realizan tareas de discriminación de los eventos que deben ser registrados.

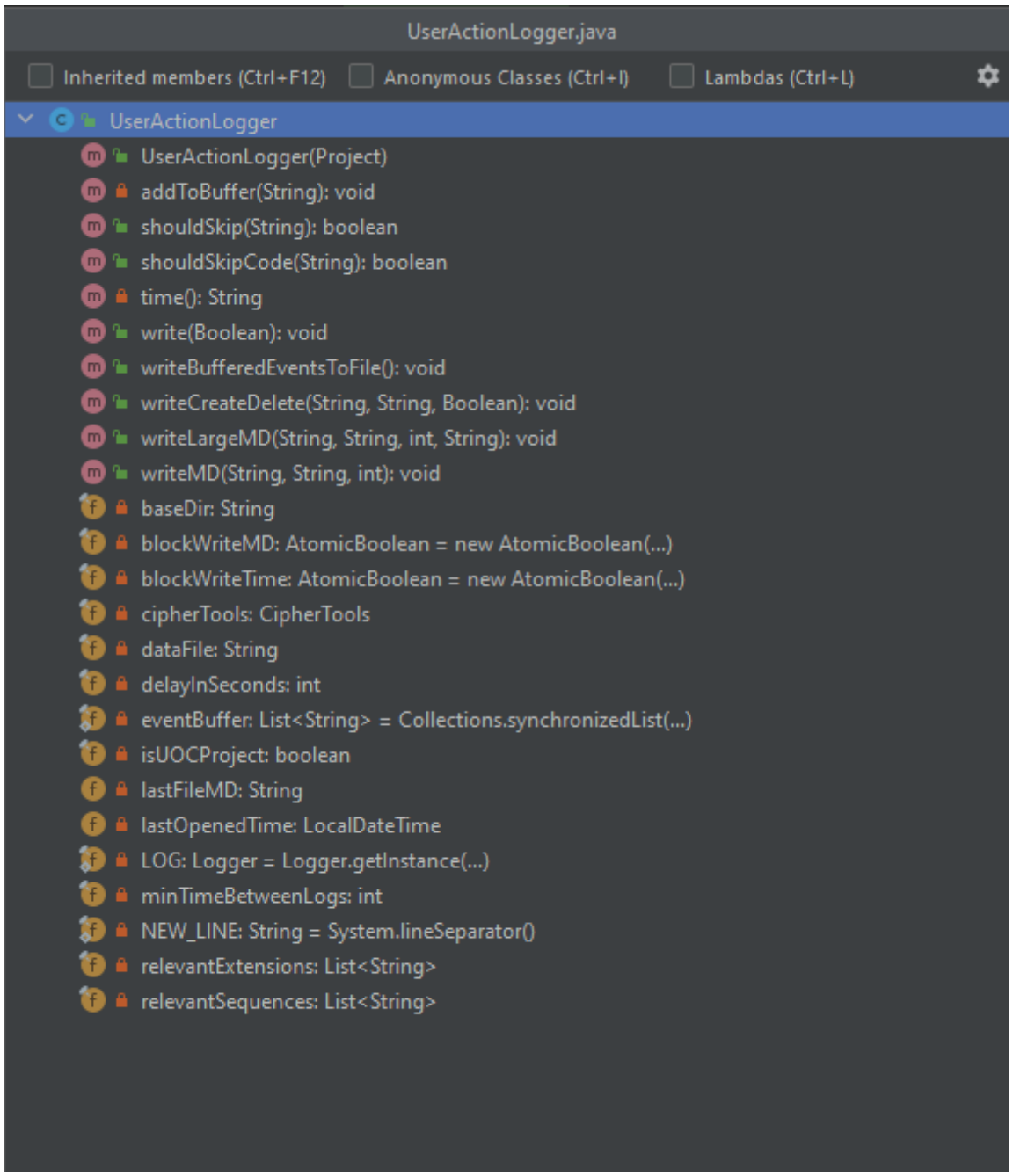

Figura 7: Variables y métodos de la clase *UserActionLogger*.

Los métodos más relevantes son los que se detallan a continuación:

- *public void write(Boolean opened)*

  Se encarga de añadir al *buffer* los eventos de apertura y de cierre del proyecto.

- *writeMD(String fileMD, String fileNameWithPath, int lineNumber)*

  Encargado de añadir al *buffer* los eventos de modificación de archivos.

- *public void writeLargeMD(String fileMD,String fileNameWithPath, int lineNumber, String addedCode)*

  Encargado de añadir al *buffer* los eventos de pegado de código.

- *public void writeCreateDelete(String fileMD, String fileNameWithPath, Boolean delete)*

  Se encarga de añadir al *buffer* los eventos de creación y eliminación de archivos.

- *shouldSkip(@NotNull String filePath)*

  Indica si cierto tipo de archivo debe ser tenido en cuenta en el registro de modificaciones por parte del usuario.

- *shouldSkipCode(String code)*

  Este método sirve de filtro para la detección de código pegado por el usuario. Evita que las inserciones de código por parte del IDE sean registradas como una acción realizada por el usuario.

3.6. Clase CipherTools

Esta clase que proporciona funcionalidades de cifrado y descifrado de archivos utilizando el algoritmo de cifrado AES. Para ello se recurre a la API de criptografía Java Cryptography Architecture (JCA)[5], específicamente las clases *Cipher*, *SecretKey*, *SecretKeyFactory*, *PBEKeySpec*, y *SecretKeySpec*.

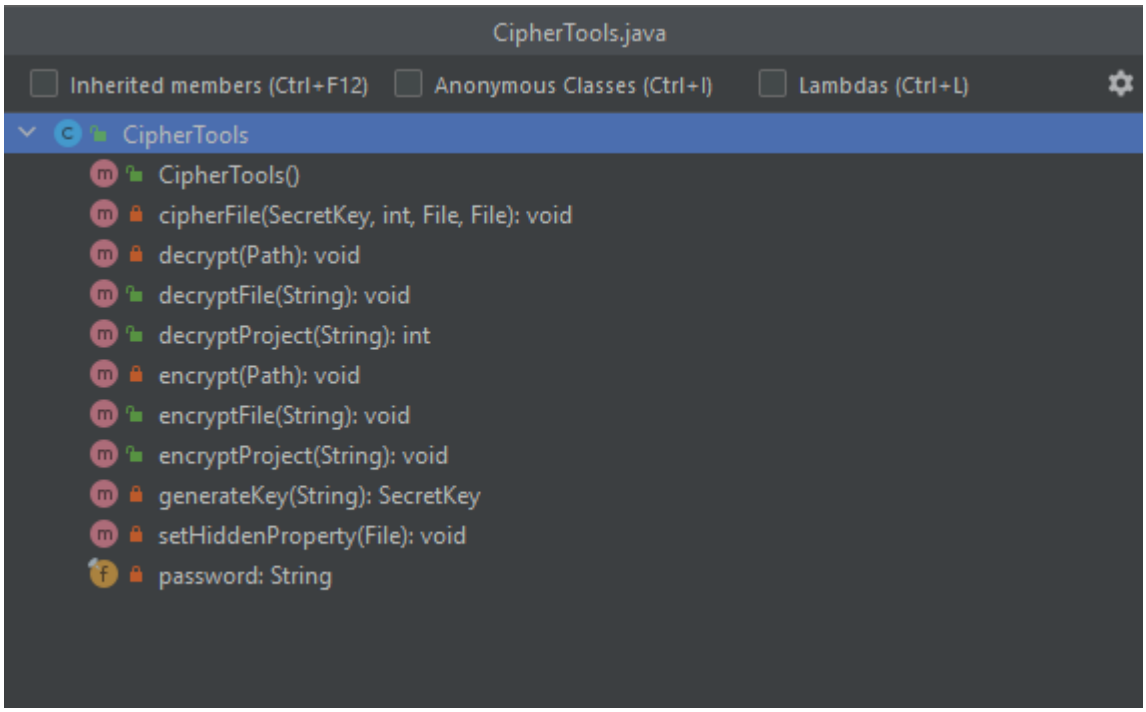

Figura 8: Variables y métodos de la clase *CipherTools*.

3.7. Clase ExportAction

La clase *ExportAction* es la superclase de las clases *ProjectToServerAction* y *ProjectToZipAction.* Contiene múltiples métodos y variables que son heredadas cuya función es la de proporcionar las herramientas comunes para realizar las exportaciones de proyectos, tanto al servidor como al disco duro local.

Al igual que otras clases, su comportamiento varía en función del tipo de usuario.

- En el caso de un usuario administrador, se añade al proyecto el archivo `uoc.data`, además, opcionalmente, añade información sobre el servidor y sobre el *pool* de entrega de proyectos al archivo.

- Si se trata de un usuario estándar, no se podrá exportar un proyecto que no incluya el archivo `uoc.data`. Además, se añadirá a este archivo el nombre del alumno y el nombre de usuario.

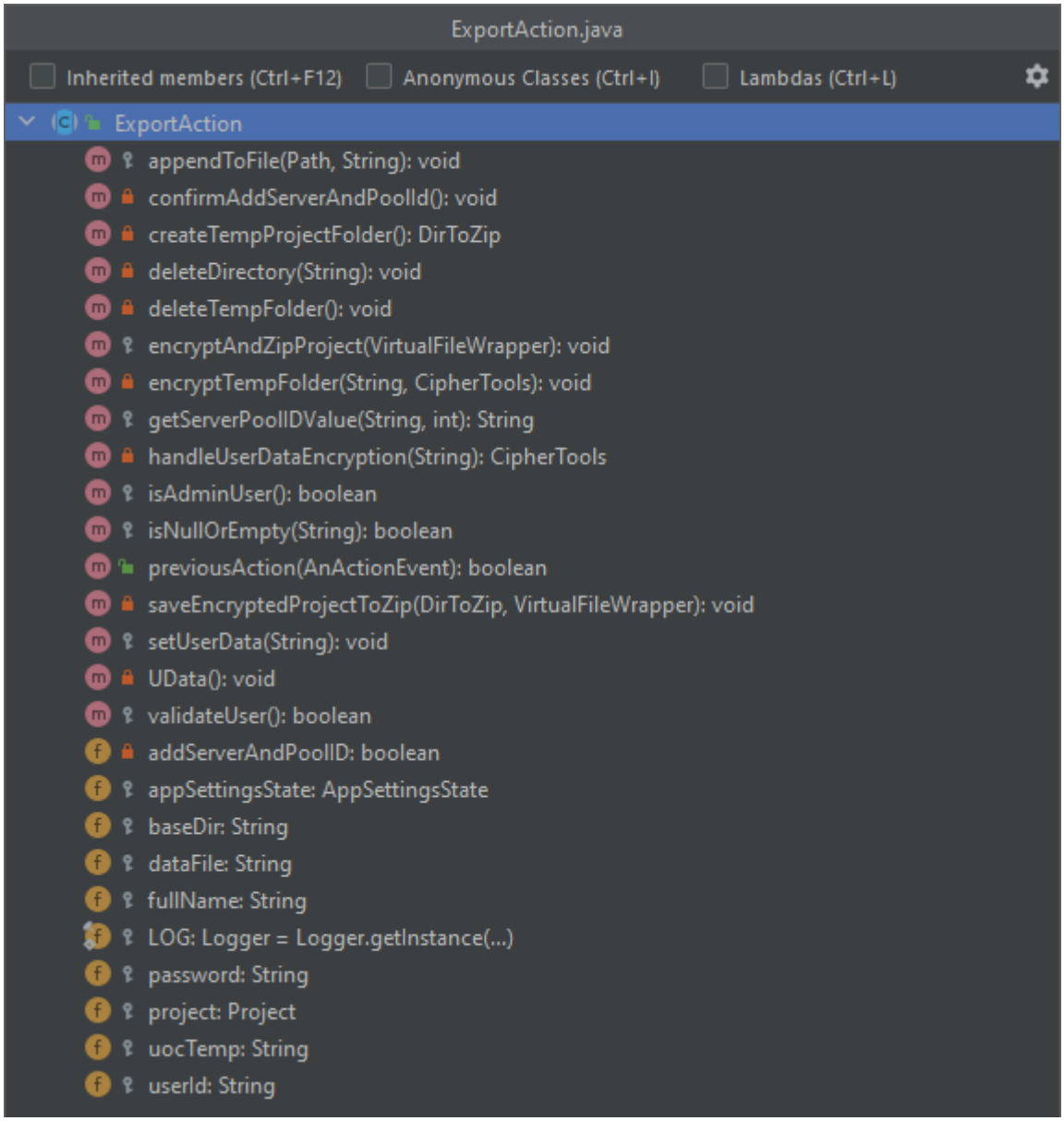

Figura 9: Variables y métodos de la clase *ExportAction*.

El método principal de la clase, denominado *encryptAndZipProject*, es empleado por dos clases descendientes, *ExportToZipAction* y *ExportToServerAction*. Su propósito es preparar el proyecto para ser exportado. Este método acepta un objeto de la clase *VirtualFileWrapper* como parámetro, el cual está vinculado a un objeto de tipo *File*. Este último contiene el proyecto ya empaquetado y listo para exportar. Su código es el siguiente:

```
    protected void encryptAndZipProject(VirtualFileWrapper virtualFileWrapper)
{

    // Cancel if the project is not from UOC
    if(!isAdminUser()){
        // Check if the project is a UOC project
        File inputFile = new File(baseDir + "/" + dataFile + ".uoc");
        if(!inputFile.exists()) {
            ApplicationManager.getApplication().invokeLater(() -> {
                Messages.showMessageDialog(
                        "The current project is not from UOC.",
                        "Warning",
                        Messages.getWarningIcon()
                );
            });
            return;
        }
    }


    // Prepare the project to be exported
    confirmAddServerAndPoolId();

    UserActionLogger userActionLogger = new UserActionLogger(project);
    userActionLogger.writeBufferedEventsToFile();

    DirToZip dirToZip = createTempProjectFolder();

    CipherTools cipherTools =
                        handleUserDataEncryption(uocTemp + "/" + dataFile);

    encryptTempFolder(uocTemp + "/" + dataFile, cipherTools);

    saveEncryptedProjectToZip(dirToZip, virtualFileWrapper);

    deleteTempFolder();
}
```

Dependiendo del tipo de usuario, se lleva a cabo un proceso de verificación inicial. Para un usuario estándar, el método comprueba si el proyecto está afiliado a la UOC. Si no es el caso, el proceso se interrumpe. No obstante, si el usuario es administrador, esta verificación se omite. Esto se debe a que un proyecto debe ser exportado por un administrador para que se le pueda considerar perteneciente a la UOC.

A continuación, el método ejecuta la siguiente secuencia:

- *confirmAddServerAndPoolId();*

  Si el usuario es de tipo administrador, se solicita la confirmación de inclusión en el archivo `uoc.data` los datos sobre el servidor y el *pool*.

- *userActionLogger.writeBufferedEventsToFile();*

  Se escriben en el archivo `uoc.data` los eventos sobre la actividad del usuario que se encuentran en el *buffer.* Esto no tiene efecto si el usuario es administrador.

- *createTempProjectFolder();*

  Se copia en un archivo temporal el proyecto para ser exportado.

- *handleUserDataEncryption(uocTemp + "/" + dataFile);*

  Se añaden al archivo `uoc.data` los datos sobre el usuario. Nombre del alumno y nombre de usuario. También, se buscarán en este archivo el nombre del *pool* y la dirección del servidor en el caso de que el usuario no los haya introducido manualmente.

- *encryptTempFolder(uocTemp + "/" + dataFile, cipherTools);*

  Se encripta el directorio temporal que contiene la copia del proyecto creado anteriormente.

- *saveEncryptedProjectToZip(dirToZip, virtualFileWrapper);*

  Se empaqueta en formato ZIP el directorio temporal.

- *deleteTempFolder();*

  Se elimina el directorio temporal.

3.8. Diagrama de clases

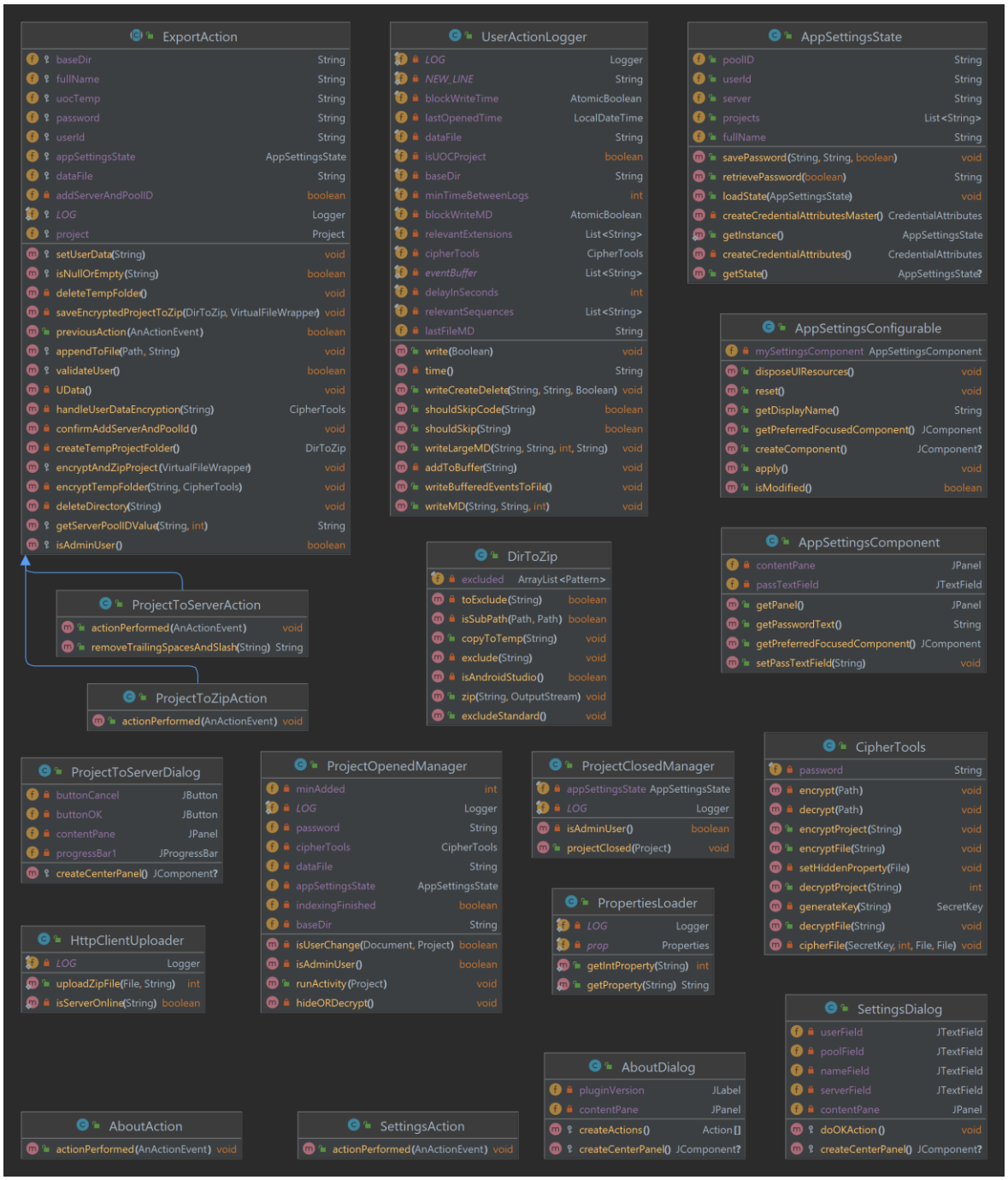

Figura 10: Diagrama de clases del proyecto.

3.9. Sistema de logs del plugin

Los *logs* son un aspecto importante a la hora de desarrollar y mantener una aplicación, en este caso, como sistema de *logs*, se ha implementado el *logger* proporcionado por el IntelliJ Platform Plugin SDK (com.intellij.openapi.diagnostic.Logger), por lo tanto, son almacenados en el archivo de logs de la plataforma (`idea.log`). En Windows la ruta es la siguiente:

```
C:\Users\nombre_de_usuario\AppData\Local\JetBrains\nombre_del_IDE\log\idea.log
```

3.10. Parámetros

En la carpeta `/src/main/resources` se encuentra el archivo de propiedades `config.properties`:

```
# Admin password
key = uoc31416

# Data file
dataFile = .uoc.data

# Minimum time in seconds between modified file log entries
minTimeBetweenLogs = 60

# Time from project opening until recording of created/deleted files begins
delayInSeconds = 35

# Delay the logging process until the indexing process is completed.
loggingDelayIndexing = yes

# Minimum characters for a pasted text (or generated by the IDE) to be logged
minAdded = 100

# If pasted code is detected, it will be discarded if it starts with any of the
following words.
# The goal is to register code generated by the IDE as pasted by the user
relevantSequences = package,//
// Created by,import

# Number of characters of the added code to cancel the relevantSequences
parameter
cancelRelevantSequences = 500

# Files to be considered for logging
relevantExtensions =
java,py,cpp,hpp,c,h,js,ts,html,css,scss,jsx,tsx,json,php,phtml,php4,php5,php3,ph
p2,phps,phpt,pht,phar,rb,rbw,rake,gemspec,rbx,duby,jbuilder,sql,ddl,dml,go,kt
```

```
# Time in minutes between writes to the hard disk, updating data.uoc logs
eventsToFilePeriod = 10
```

En este archivo encontramos algunos parámetros desde los que podemos modificar ciertos comportamientos:

- key

  Contraseña de administrador utilizada, además, en los procesos de cifrado y descifrado.

- dataFile

  Archivo en el que se almacenará la actividad del usuario.

- minTimeBetweenLogs

  Cantidad mínima de segundos que debe haber transcurrido desde el registro de un evento de modificación de un archivo para que se registre otro evento de ese mismo archivo de forma consecutiva. La finalidad es evitar que se registren una cantidad excesiva de eventos.

- delayInSeconds

  Parámetro que se utiliza para dar un margen de tiempo hasta que el IDE realice sus operaciones internas iniciales antes de empezar a registrar eventos de borrado o creación de archivos. El objetivo es prevenir que se registren operaciones realizadas por el IDE y no por el usuario, como la creación de la carpeta `.idea` con sus archivos correspondientes.

- logginDelayIndexing

  Indica si se debe esperar a que el proceso de *indexado* de archivos del IDE haya finalizado, para empezar o continuar registrando eventos de modificación, borrado o creación por parte del usuario. Si este parámetro está activado, puede provocar que

no se registren estos eventos durante los primeros segundos previos a la carga del proyecto.

- minAdded

  Cifra que representa la cantidad mínima de caracteres que debe tener una porción de código insertada en un archivo, para que sea registrado como código pegado por el usuario, en este caso 100, espacios en blanco incluidos. La finalidad es evitar que acciones por parte del IDE, como las de autocompletar, sean registradas como código pegado por el usuario.

- relevantSequences

  Parámetro que tiene un objetivo similar al anterior, en este caso, se pretende evitar que el código insertado por el IDE al crear un nuevo archivo sea registrado por el plugin como pegado por el usuario. Este parámetro admite secuencias de código separadas por comas. Se puede observar que está introducida la palabra “package”, la razón es que al crear un nuevo archivo en java el IDE añade automáticamente esa palabra en primer lugar.

- cancelRelevantSequences

  Si el código insertado tiene más caracteres de los indicados por este parámetro, se considerará pegado por el usuario y no generado por el IDE. Por lo tanto, esto anularía el parámetro anterior. Por ejemplo, si el usuario pega una clase entera, considerando que posiblemente supere los 500 caracteres indicados, espacios en blanco y saltos de línea incluidos, aunque empiece por la palabra “package”, la acción quedará registrada.

- relevantExtensions

  Determina los archivos que se deben tener en cuenta para el registro de acciones del usuario en función de sus extensiones. Con este parámetro se pueden excluir, por ejemplo, los archivos XML.

- eventsToFilePeriod

  La actividad del usuario se va almacenando en un buffer en la memoria, este parámetro indica cada cuanto tiempo se actualiza el archivo `uoc.data` con los nuevos registros almacenados en el *buffer*.

3.11. Archivo Plugin.xml

El archivo `plugin.xml` es un elemento crucial en el desarrollo de *plugins* para IDEs de JetBrains. Este archivo es esencialmente la declaración del *plugin*, donde se describen sus características, se definen sus dependencias y se registran sus componentes y extensiones. A continuación, se describen algunas de sus funciones clave:

- Identificación

  Mediante las etiquetas <*id*> y <*name*>, se establece la identidad única del plugin.

- Proveedor

  La etiqueta <*vendor*> proporciona información sobre el desarrollador o la organización que ha creado el *plugin*. Esta información puede incluir un correo electrónico y una URL.

- Dependencias

  La etiqueta <*depends*> permite especificar otros *plugins* de los que depende este *plugin* para funcionar correctamente. Estas dependencias deben estar presentes para que el *plugin* funcione.

- Extensiones

  La sección <*extensions*> permite registrar extensiones, que son los puntos donde el *plugin* puede interactuar con el IDE. Las extensiones pueden proporcionar una variedad de funcionalidades, como nuevos servicios o acciones.

Esta sección es donde se registran las clases *ProjectOpenedManager, AppSettingsState* y *AppSettingsConfigurable.*

- *Listeners* de eventos

  Los *listeners* de eventos, como los definidos en <*applicationListeners*>, permiten reaccionar a eventos específicos dentro del IDE, como el cierre de un proyecto.

  En esta sección es donde se registra el *listener ProjectManagerListener* de la clase *ProjectClosedManager.*

- Acciones

  La sección <*actions*> permite registrar acciones, que son las funcionalidades que los usuarios pueden invocar. Estas acciones se pueden agrupar en menús y submenús en la interfaz de usuario.

- Configuración

  A través de varias etiquetas y atributos se define la configuración del *plugin*, incluyendo si requiere un reinicio después de la instalación o actualización.

3.12. Compatibilidad

El plugin ha sido probado en Windows y en Linux, también en las IDEs IntelliJ IDEA, PyCharm y CLion. Adicionalmente, JetBrains ofrece una pequeña aplicación llamada intellij-plugin-verifier[6] que sirve para verificar la compatibilidad de los *plugins* con sus IDEs. Al comprobar la compatibilidad con las versiones de IntelliJ IDEA 2023.1, IntelliJ IDEA 2023.1.2, CLion 2023.1 y PyCharm 2023.1, se ha obtenido como resultado que el *plugin* es compatible. Esta aplicación está alojada en GitHub[7].

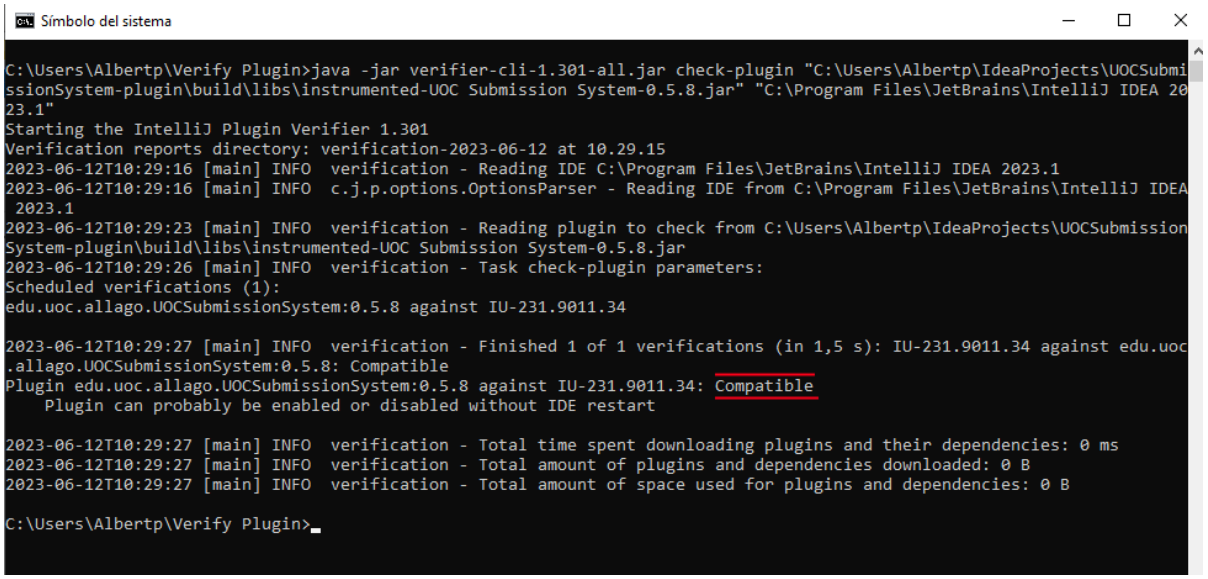

Figura 11: Verificación de compatibilidad contra IntelliJ IDEA 2023.1.

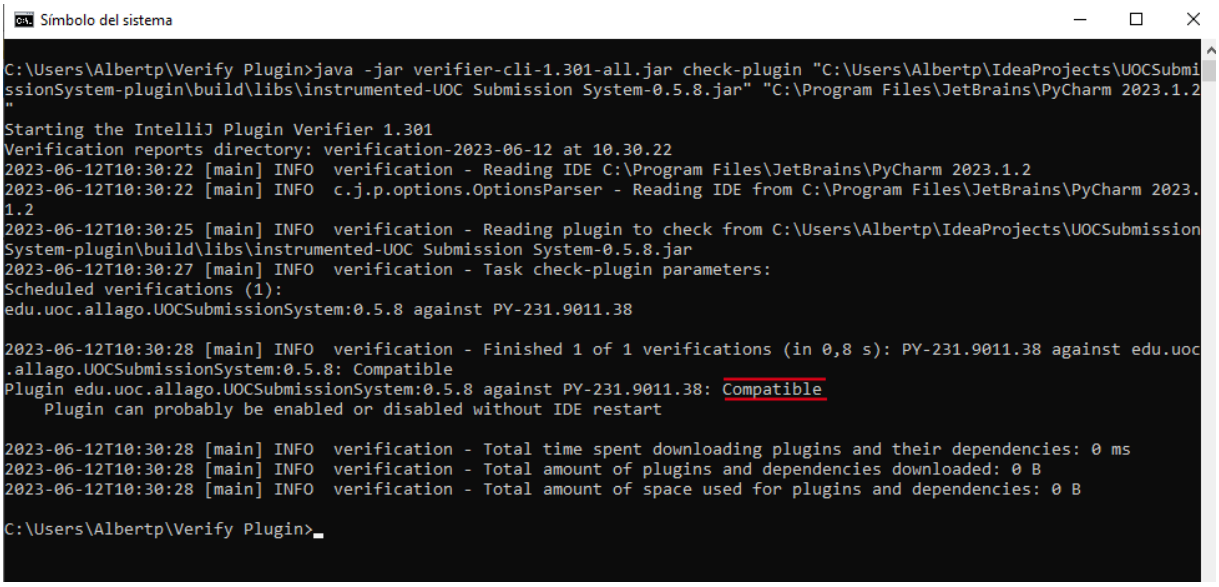

Figura 12: Verificación de compatibilidad contra PyCharm 2023.1.2.

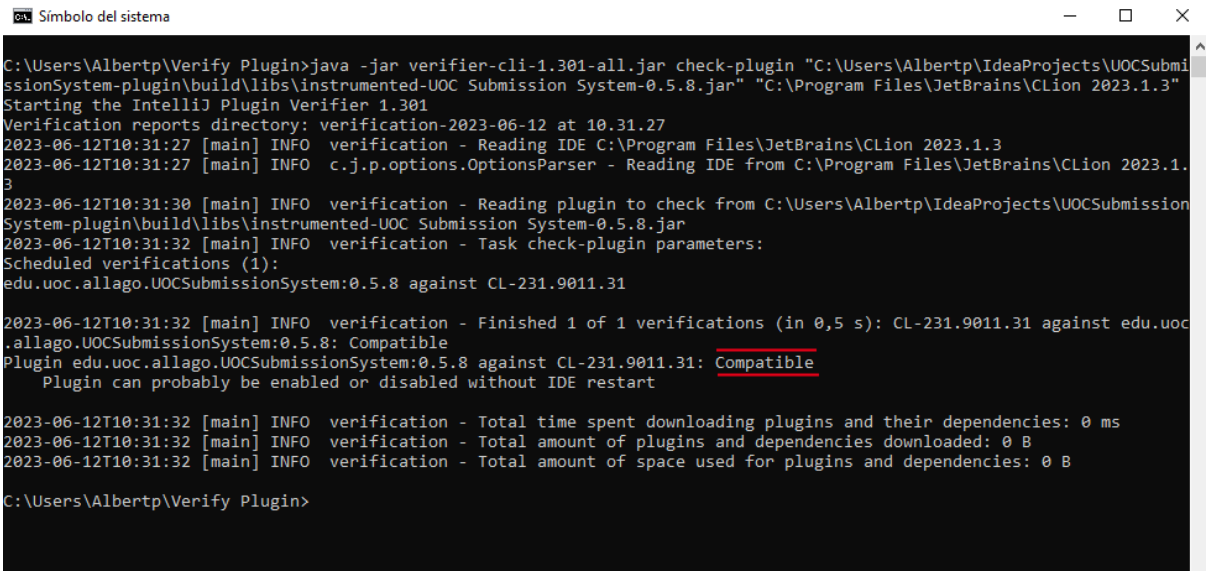

Figura 13: Verificación de compatibilidad contra CLion 2023.1.3.

También, es posible realizar esta verificación desde el IDE, para realizarla existe una *task* de Gradle.

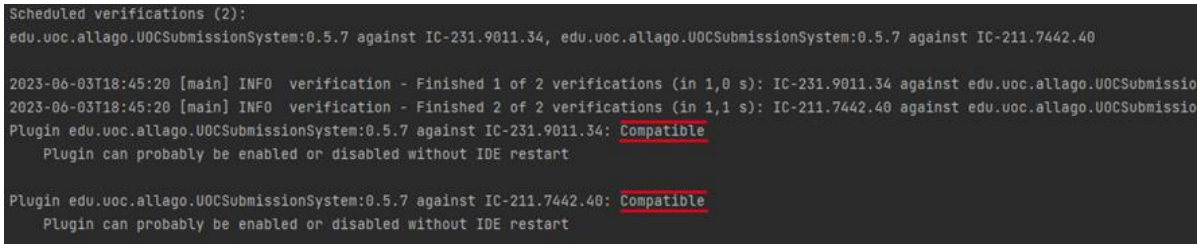

Figura 14: Verificación de compatibilidad en IntelliJ por medio de la *task* de Gradle.

En la última versión del *plugin* se ha activado el soporte para Android Studio Flamingo | 2022.2.1 Patch 2 y se han realizado pruebas con éxito, sin embargo, sería recomendable realizar más pruebas.

A lo largo del desarrollo, se ha hecho un esfuerzo consciente para evitar el uso de clases marcadas como obsoletas (deprecated) durante el proceso de compilación. Cualquier inclusión de dichas clases también habría sido señalada por la verificación de compatibilidad.

3.13. Servidor

El servidor incorpora las funciones básicas, su desarrollo está basando en el patrón de diseño MVC implementado por medio de SpringBoot. Para lanzarlo, como es habitual, es posible hacerlo desde el propio IDE o ejecutarlo desde la terminal del sistema. Si se desea ejecutar el servidor especificando algún puerto en particular (por defecto es el 50000) puede ser incluido como parámetro de lanzamiento:

```
java -jar UOCSubmissionSystem-server-0.0.3-SNAPSHOT.jar --sever.port=8081
```

La seguridad y el sistema de autenticación y de control de acceso, se implementa a través de SpringSecurity. El formulario para iniciar sesión se muestra en la figura 13. SpringSecurity es altamente personalizable, los parámetros de configuración se pueden encontrar en la clase *SecurityConfig*.

La aplicación utiliza una combinación de vistas generadas en el servidor mediante Thymeleaf y una API REST para la comunicación entre el controlador y la vista. Thymeleaf se utiliza para generar las páginas HTML que se envían al cliente.

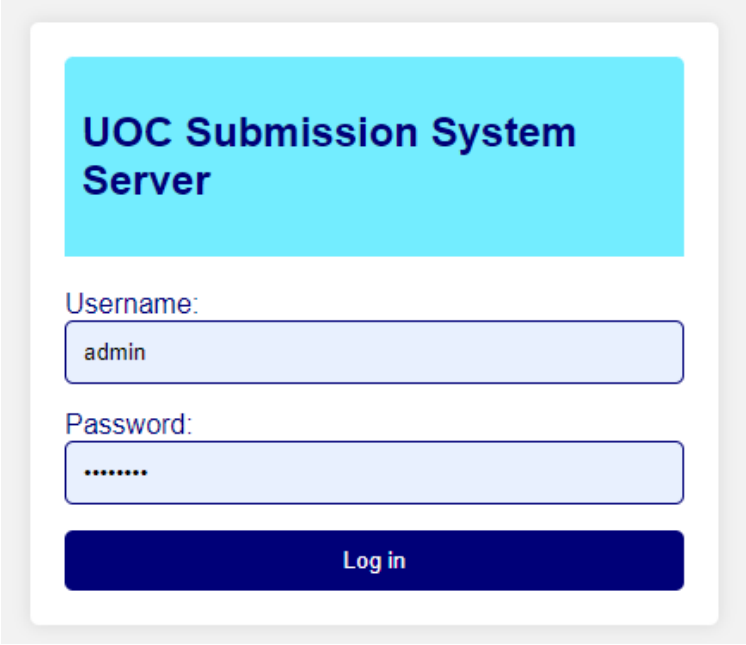

Figura 15. Formulario para el inicio de sesión.

El proceso se fundamenta en la creación de *pools* donde se almacenarán los proyectos subidos por los alumnos. Por ejemplo, para la PEC 1 de la asignatura DPOO del aula 1, se podría

crear un *pool* llamado DPOO_PEC1_AULA1 y en dicho *pool* se irán recibiendo y almacenando las entregas de la PEC 1 de los distintos estudiantes de esa asignatura y de esa aula. En la figura 16 se observa un ejemplo de lista de *pools*.

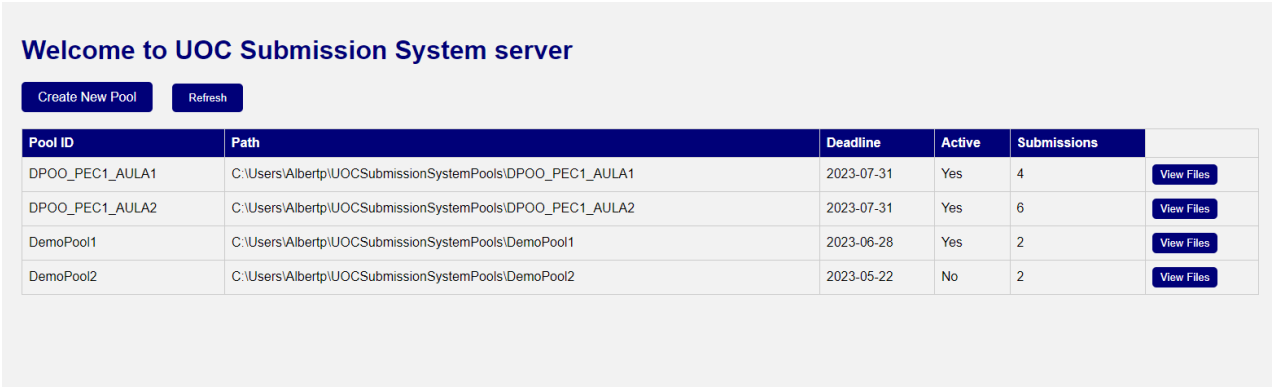

Figura 16. Ejemplo de una lista de *pools*.

En la figura 17 se observa el formulario para la creación de un *pool*. Al crearlo se debe indicar una fecha límite para las entregas y, si un alumno tratase de subir su proyecto una vez sobrepasada, la subida no tendría éxito. Esta información es persistente.

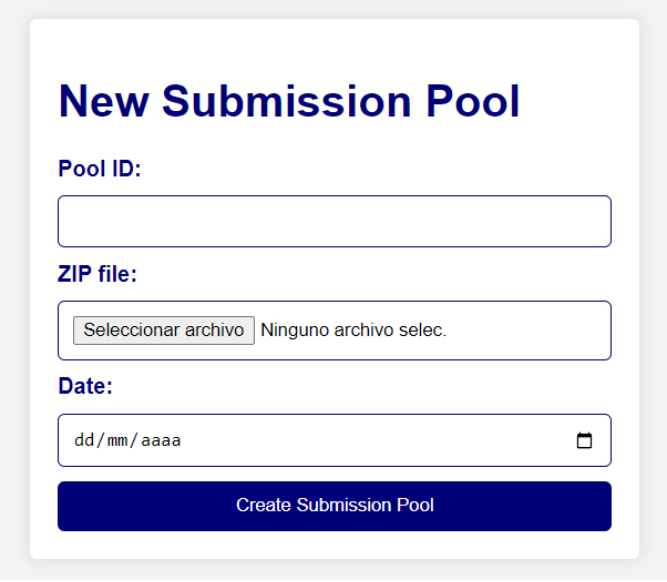

Figura 17. Formulario para la creación de un *pool* de entregas.

Los *pools* serán creados en la carpeta `UOCSubmissionSystemPools` que se generará automáticamente en la carpeta home del usuario.

- Windows: `C:\Users\nombre_de_usuario\UOCSubmissionSystemPools`
- Linux: `/home/nombre_de_usuario/UOCSubmissionSystemPools`

Cuando un estudiante exporte un proyecto, el nombre del archivo ZIP donde se empaquetará será el nombre de usuario del estudiante que, obligatoriamente, debe introducir previamente en el IDE o no podrá realizar la exportación.

Las entregas realizadas pueden ser descargadas por el administrador directamente desde la interfaz gráfica. Es posible descargar juntas todas las entregas que contiene un *pool* o realizarlo de forma individual. En la figura 18 se observa la página de descarga de los proyectos entregados en un *pool*.

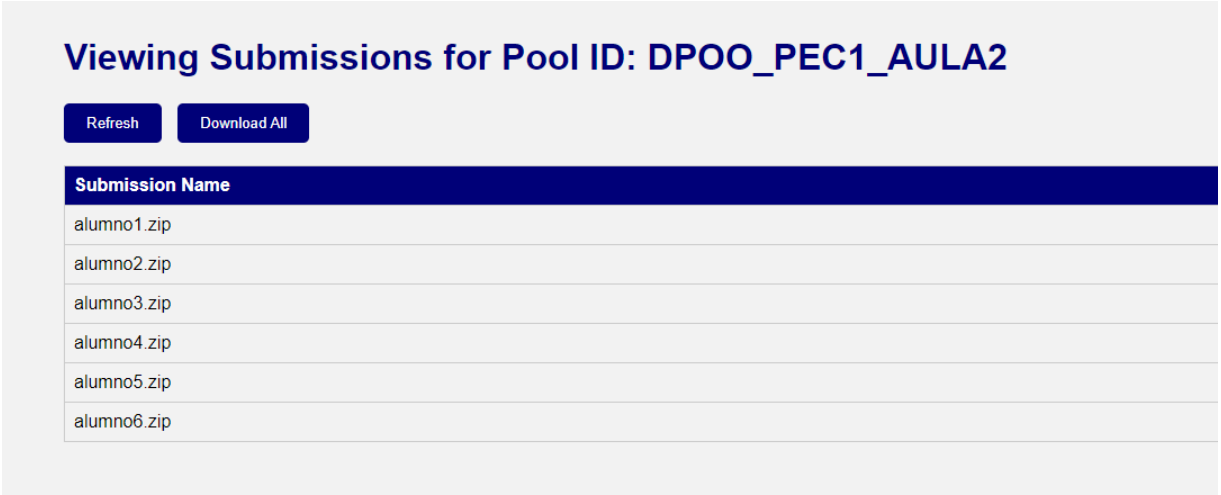

Figura 18. Ejemplo de una lista de entregas almacenadas en un *pool*.

4. Resultados

Se ha logrado un nivel de desarrollo completamente operativo, cuyo comportamiento ha resultado satisfactorio al realizar las pruebas correspondientes.

Se ha desarrollado un servidor que complementa al *plugin* con la capacidad de recibir los proyectos. En las siguientes secciones, para ilustrar el resultado del trabajo previamente descrito, se detallará su uso desde la perspectiva de los dos posibles tipos de usuario.

4.1. Usuario administrador

Cuando se introduce la contraseña de administrador en los *settings* generales del IDE, el *plugin* actúa de un modo diferente.

- Al abrir un proyecto cifrado, será descifrado y pasará a ser visible el archivo `uoc.data`. Además, no se volverá a cifrar en el cierre. En este archivo se podrán consultar los eventos registrados relacionados con la actividad del alumno. En la figura 19 podemos observar un ejemplo de este archivo, en la línea 1 ha quedado registrado un evento de apertura del proyecto, en la línea 2 se ha registrado la creación de un nuevo archivo, finalmente, entre las líneas 3 y 12 se registra una acción de pegado de código, mostrando el código que ha sido pegado.

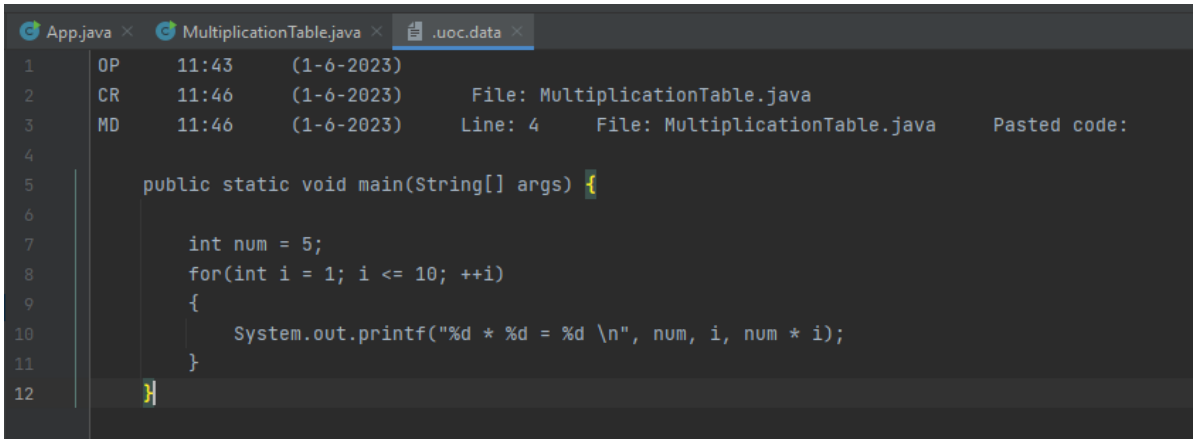

Figura 19: Captura de pantalla del contenido del archivo de registro de actividad del usuario.

- La función del menú del *plugin*, *Export to zip,* sirve para generar el proyecto para los alumnos. Existe la opción de incluir, en el archivo `uoc.data`, del proyecto generado, el nombre del pool al que se debe subir ese proyecto, así como la dirección del servidor. Los datos de estos parámetros serán tomados de los introducidos desde el menú del plugin en *Settings,* accesible desde la barra de herramientas.

4.2. Usuario estándar

Cuando el estudiante descomprime el archivo ZIP, obtiene el proyecto cifrado. Este solo puede ser abierto desde un IDE de JetBrains donde el plugin esté instalado. Una vez que se abre, el plugin descifrará el proyecto automáticamente, excepto el archivo `uoc.data`. Este archivo permanecerá cifrado en todo momento, solo será descifrado de forma silenciosa durante los intervalos en que el plugin necesite acceder a él. El archivo de registro `uoc.data` no se mostrará en el IDE y se marcará como oculto en el sistema operativo. Al cerrar el proyecto o el IDE, el proyecto se cifrará nuevamente.

Dependiendo del IDE y el tipo de proyecto, es posible que el estudiante deba realizar ciertas configuraciones iniciales al abrir un proyecto por primera vez, como seleccionar la versión del JDK que quiere usar.

Internamente, el plugin mantiene un registro de los proyectos abiertos bajo su control y puede manejar más de un proyecto de la UOC abierto a la vez sin problemas.

Para exportar un proyecto, el estudiante debe introducir, además de su nombre, el nombre de usuario de la UOC. Si no se proporcionan estos datos, la acción de exportación no será posible. Se añadirán ambos nombres al final del archivo de registro antes de realizar la exportación. Cada proyecto que se exporte siempre se cifrará durante el proceso.

Si el estudiante desea exportar un proyecto directamente al servidor, puede ingresar el pool y el servidor en los campos correspondientes en la configuración del plugin. Si no se proporciona esta información, el *plugin* intentará buscarla en el archivo de registro.

5. Conclusiones y trabajos futuros

Tras la conclusión de este proyecto, he logrado expandir significativamente mi conocimiento sobre el funcionamiento de los *plugins* para los IDEs de JetBrains. Este aprendizaje me ha llevado a comprender que, cuando se combinan con los *plugins* adecuados, estos IDEs se convierten en potentes herramientas para el aprendizaje. La amplia gama de personalizaciones posibles ofrece un marco muy versátil y, creo firmemente, que sería altamente beneficioso integrar estos IDEs en el proceso educativo de los estudiantes, su potencial es indiscutible.

Se ha iniciado el desarrollo a partir de un plugin que simplemente era mostrado en los menús del IDE y, desde ese punto, se han ido añadiendo las distintas funciones hasta que, finalmente, aunque los principales objetivos planteados al inicio del proyecto han sido alcanzados siempre es posible realizar más pruebas y mejoras.

5.1 Objetivos alcanzados

Objetivos alcanzados del *plugin*:

- Capacidad para exportar los proyectos de forma local.
- Capacidad para exportar los proyectos a un servidor remoto.
- Habilitar proyectos que solamente podrán ser abiertos si el *plugin* se encuentra instalado.
- Funciones de monitorización y registro “silencioso” de la actividad del usuario.
- Compatibilidad con las principales IDEs de JetBrains.
- Distinción y comportamiento según el tipo de usuario (administrador o usuario estándar).

Objetivos alcanzados del servidor:

- Desarrollo de un servidor funcional basado en MVC SpringBoot que permita recibir y descargar los proyectos exportados.
- Interfaz gráfica con información sobre los proyectos entregados.
- Posibilidad de establecer fechas límite para la recepción de proyectos.

Se debe mencionar que algunos de estos objetivos no estaban definidos de forma concreta inicialmente. Especialmente el tipo de información sobre la actividad del usuario registrada. Se ha dedicado buena parte del esfuerzo a implementar la capacidad de detectar acciones como el pegado de código.

5.2 Funcionalidades descartadas

- Para los alumnos (usuarios estándar), se ha descartado la posibilidad de descargar los proyectos directamente desde el plugin.
- Mecanismo para cambiar la contraseña en el *plugin* sin necesidad de compilar de nuevo.
- En el servidor, mecanismo para crear usuarios y modificar las contraseñas.

5.3 Posibles líneas de trabajo futuras

En los últimos meses, se ha logrado el desarrollo de un *plugin* eficaz. Sin embargo, como cualquier pieza de *software*, siempre existe la posibilidad de mejorarla y pulirla.

Un componente crucial para considerar en el futuro es la implementación de pruebas rigurosas y extensivas del comportamiento en la amplia gama de entornos de IDEs en los que este plugin podría ser aplicable. En teoría, se prevé que sea compatible con la mayoría de IDEs de JetBrains.

Como ya se ha resaltado en esta documentación, la característica de detección de eventos de pegado de código es altamente configurable y puede requerir ciertos ajustes para diferenciar entre la inserción de código por parte del usuario y la del propio IDE.

Un posible añadido interesante sería que el complemento notifique cuando se aproxime la fecha límite de un proyecto.

6. Glosario

- **API:** Siglas de Application Programming Interface, o Interfaz de Programación de Aplicaciones en español. Es un conjunto de reglas y protocolos establecidos por el software para la interacción con otros softwares. Esencialmente, es una manera para que diferentes programas se comuniquen entre sí.

- **AES:** Advanced Encryption Standard, o Estándar de Encriptación Avanzada en español. Es un protocolo de encriptación de datos que se utiliza a nivel mundial. Fue establecido por el Instituto Nacional de Estándares y Tecnología (NIST) de los Estados Unidos en 2001.

- **Buffer:** Es una región de memoria en la que se almacenan temporalmente datos.

- **IDE:** Integrated Development Environment, o Entorno de Desarrollo Integrado en español. Es una aplicación que proporciona servicios para facilitar el desarrollo de software. Estos servicios suelen incluir un editor de texto, herramientas de construcción automática, y un depurador.

- **Javadoc**: Es una herramienta de software incluida en el JDK de Oracle que genera automáticamente la documentación de API en formato HTML a partir de los comentarios de código fuente en Java. Los desarrolladores incluyen en sus comentarios etiquetas predefinidas de Javadoc, que la herramienta luego utiliza para generar la documentación. Esta documentación sirve como referencia y guía para entender la funcionalidad y uso de las clases, métodos y variables en un código de programa Java.

- **JDK:** Java Development Kit, o Kit de Desarrollo de Java en español. Es un conjunto de herramientas de software que se utilizan para desarrollar aplicaciones en el lenguaje de programación Java. Incluye el intérprete de Java, el compilador de Java, el archivador y otras herramientas útiles.

- **Logger:** En el contexto de la programación, un *logger* (o registrador) es una herramienta o proceso que registra automáticamente los eventos o las interacciones que ocurren en un sistema o aplicación. Esto puede ser especialmente útil para la depuración o para monitoreo de operaciones.

- **Task:** En el contexto del desarrollo de Java y Kotlin, especialmente cuando se trata del sistema de construcción de Apache Ant o Gradle, una *task* es un bloque de código que puede ser ejecutado. Las *tasks* se utilizan para definir qué se debe hacer en ciertos puntos del proceso de construcción del proyecto. Por ejemplo, puede haber

una *task* para compilar el código fuente, otra para empaquetarlo en un archivo JAR y otra para limpiar los archivos generados durante la construcción.

7. Bibliografía

[1] https://plugins.jetbrains.com/docs/intellij/welcome.html [Consultado 9 abril 2023].

[2] https://plugins.jetbrains.com/plugin/13608-pyphanon [Consultado 9 abril 2023].

[3] https://plugins.jetbrains.com/plugin/18353-showyourwork [Consultado 10 abril 2023].

[4] https://plugins.jetbrains.com/plugin/17434-cse335-course-extensions [Consultado 10 abril 2023].

[5] https://medium.com/@thiagohernandes_10323/java-cryptography-architecture-jca-by-examples-9a3a3396dd3 [Consultado 15 junio 2023].

[6] https://plugins.jetbrains.com/docs/intellij/verifying-plugin-compatibility.html#plugin-verifier [Consultado 15 junio 2023].

[7] https://github.com/JetBrains/intellij-plugin-verifier [Consultado 15 junio 2023].