TecnoCampus
Escola Superior Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

Grado en Ingeniería Informática de Gestión y Sistemas de Información

Automatización de la corrección de ejercicios de programación utilizando Github Actions

Nikola Nikolaev Hristozov

TUTOR: Alfredo Rueda Unsain

2023-2024

TecnoCampus
Mataró-Maresme

Abstract

Programming learning tools have been prevalent for more than twenty years, automating processes such as correcting exercises or giving feedback about the results. This project aims to use the Github Actions to automate JUnit tests for practices of the EDA subject at the University of Tecnocampus, it also aims to create its own Moodle server to also simulate a student's score in the case of having all the correct tests.

Resum

Les eines d'aprenentatge de programació han estat prevalents des de fa més de vint anys, automatitzant processos com la correcció d'exercicis o donant comentaris sobre el resultat. Aquest projecte pretén utilitzar Github Actions per automatitzar tests de JUnit de practiques de l'assignatura d'EDA de la universitat de Tecnocampus, a més es pretén crear un propi servidor Moodle per simular la puntuació d'un alumne en cas de tenir tots els tests correctes.

Resumen

Las herramientas de aprendizaje de programación han sido prevalentes desde hace ya más de veinte años, automatizando procesos como la corrección de ejercicios o dando comentarios acerca del resultado. Este proyecto pretende utilizar Github Actions para automatizar tests de JUnit de prácticas de la asignatura de EDA de la universidad de Tecnocampus, además se pretende crear un propio servidor Moodle para además simular la puntuación de un alumno en el caso de tener todos los tests correctos.

Índice

Tabla de contenido

Índice de Figuras

1. Introducción

Este proyecto pretende desarrollar una forma de automatizar correcciones de prácticas de EDA e integrar estas correcciones directamente en el moodle [1]del propio ecampus.

El objetivo de este proyecto es desarrollar utilizando Github Actions, una forma de automatizar la corrección de los Tests de prácticas de EDA o ejercicios Kata para ofrecer una mayor eficiencia a la hora de corregir este tipo de ejercicios.

Mediante Github Actions aprenderemos las diferentes funcionalidades que tiene para luego poder automatizar e integrar estas correcciones en el moodle.

Una vez decidido desarrollamos cumpliendo todos los objetivos tecnológicos y funcionales impuestos.

Todos los usuarios deben poder usar esta automatización una vez configurada a través de Github, dichas prácticas estarán configuradas para automatizar sus correcciones de las clases test cuando se realicen las acciones que provoquen esta automatización sea por ejemplo un push a branch [2]o un pull request[3].

Adicionalmente se quiere crear un informe PDF que informará a los usuarios de los tests que estén funcionando correctamente y los que no. Por último, se querrá integrar la automatización con el moodle de la universidad para poder modificar las notas de estas mismas prácticas una vez acabadas el plazo para entregarlas.

[1] Plataforma de aprendizaje diseñada para proporcionar a educadores y estudiantes un sistema para crear ambientes de aprendizaje personalizado

[2] Comando de git, se utiliza para calcular la diferencia de datos entre el repositorio local y el remoto para luego subir (push) la diferencia al otro repositorio

[3] A diferencia del push un pull request solicita a otro desarrollador del mismo repositorio que incorpore los datos de tu repositorio a una rama del suyo

2. Estudio Previo

2.1 Las primeras automatizaciones en corrección de ejercicios

La corrección de ejercicios es un pilar fundamental a la hora de educar a los estudiantes, siendo crucial para evaluar la comprensión y aplicación del conocimiento adquirido por su parte (S. Benford, E. Burke, E. Foxley, N. Gutteridge, A. Mohd Zin, 1993). Dentro de la programación es necesario y esencial practicar para poder aplicar los conocimientos de mejor forma por lo que cobra una importancia aún mayor la necesidad de una evaluación precisa, dada la relevancia de la experiencia práctica en el dominio de la programación. Aun así, la evaluación de tareas de programación a través de métodos tradicionales manuales se encuentra con ciertos obstáculos como por ejemplo la tediosidad y la susceptibilidad al error humano.

Las limitaciones de las metodologías de evaluación convencionales, como, por ejemplo, en el contexto de la educación en programación, donde la capacidad de discernir la funcionalidad de un programa a través de una inspección visual sigue siendo una búsqueda elusiva (H. Aldriye, A. Alkhalaf, M. Alkhalaf, 2019). Agravado por el creciente número de estudiantes, especialmente en cursos fundamentales de programación, la carga administrativa y las limitaciones de la evaluación tradicional agudizan la problemática a la que se enfrentan los educadores.

En 1960 aparece la primera herramienta de autocorrección en programación por Hollingsworth, que decidía con un “wrong answer” o “program complete” su funcionalidad (M. Huzaifah Ismail, M. Modi Lakulu, 2020).

En 1988 aparece Ceilidh, un sistema automatizado de evaluación diseñado específicamente para evaluar la competencia de los estudiantes en programación. Ceilidh marca un hito en la evaluación educativa al emplear un mecanismo automático de evaluación que analiza programas desde perspectivas multifacéticas. A través de la evaluación dinámica de la corrección, el análisis del estilo y complejidad de programación y la facilitación de diversas

funciones administrativas, Ceilidh se esfuerza por revolucionar el panorama de la evaluación dentro de los entornos educativos (Z. Shukur, 1999).

El origen de Ceilidh se remonta a su inicio en 1988, donde inicialmente respaldaba cursos de programación en C, marcando el comienzo de su trayectoria evolutiva. Fases posteriores de desarrollo, incluida su adopción en diversos cursos de programación en todo el mundo, subrayan su adaptabilidad, relevancia y potencial transformador dentro de los marcos educativos contemporáneos (S. Benford, E. Burke, E. Foxley, N. Gutteridge, A. Mohd Zin, 1993).

La 2a versión de Ceilidh tuvo inversión por parte de la universidad de Nottingham, que al final fue usada para un curso de C++ en una clase de 160 alumnos durante los años 1991/92.

Hoy en día estas herramientas se siguen utilizando e innovando con el paso del tiempo, hoy en día herramientas como Automata (2014), SAUCE (2015), Aristotle (2017), Paprika (2019) son las más recientes e utilizadas en los entornos de correcciones de programación automática (M. Huzaifah Ismail, M. Modi Lakulu, 2020).

2.2 Ventajas y desventajas de la Automatización

La automatización de testeo es la ejecución de un programa independiente al que queremos realizar las pruebas para controlar la ejecución de las pruebas y comparar los resultados obtenidos por la automatización con los esperados (Z. Shukur, 1999). Muchas de estas herramientas agilizan aspectos específicos de las pruebas manuales, pero no todo se puede llegar a automatizar.

La principal ventaja de las pruebas automatizadas es el tiempo que se gana en ejecutar varias pruebas que en otro caso requeriría de mucho más tiempo si se fueran a ejecutar manualmente además de evitar el esfuerzo y coste si se fueran a hacer de esta forma.

Aun así, el éxito de que estas herramientas de automatización funcionen dependen también de que una persona sea capaz tanto con las herramientas de automatización

como con el software al que se está intentando automatizar para poder establecer tests efectivos (E. Dustin, J. Rashka, J. Paul, 1999).

Mubarak Albarka y Chen Zhanfang (2019) explican resumidamente las ventajas y desventajas que conlleva la automatización de los tests.

Ventajas:

- Mayor rapidez y precisión a la hora de encontrar errores comparado si se fuera hacer de forma manual
- Ahorra tiempo y esfuerzo haciendo la comprobación de tests más eficiente
- Mejora el test coverage por el hecho de poder utilizar varias herramientas de testeo al mismo tiempo y automatizar todas y cada una de ellas
- La automatización de los tests es repetible tantas veces como sea necesario

Desventajas:

- Utilizar la herramienta adecuada para realizar el testeo y automatizarla requiere de tiempo y esfuerzo
- Conocimientos previos sobre la herramienta de testeo que se va a utilizar
- El coste de comprar dicha herramienta de testeo y su mantenimiento
- Proficiencia a la hora de emplear dichas herramientas de automatización

2.3 Categorías de herramientas de Automatización

Dentro de la automatización de tests podemos encontrar herramientas divididas en en las siguientes categorías (M. Albarka, C. Zhanfang, (2019).

Unit Testing

Unit testing se emplea para evaluar los componentes más básicos del código, utilizando herramientas conocidas como herramientas de Unit Testing para agilizar el proceso. Se

utilizan e integran en entornos como NetBeans. La principal función de Unit Testing es verificar la funcionalidad de una pequeña parte del código por lo que se requieren de muchos Unit Testing en el caso de tener un código con muchas funcionalidades.

Adicionalmente se examina la estructura del código asegurando el cumplimiento de una programación correcta además de encontrar errores durante las primeras etapas del desarrollo. Algunos marcos de Unit Testing incluyen JUnit, HTMLUnit, Jasmine.

Functional Testing

Functional testing a diferencia de unit testing se centra más en verificar que el software funcione correctamente acorde a los requerimientos acordados y las expectativas del usuario. Herramientas de testeo funcional son aquellas que dadas una información inicial se compara con el resultado obtenido para verificar el testeo. Algunas herramientas de Functional Testing incluyen Cypress, Selenium, TestComplete.

Code coverage

Las Herramientas de Code Coverage son esenciales a la hora de automatizar nuestros tests, con ellas podemos determinar que porciones de nuestro código, funciones y líneas han sido ejecutadas durante los tests para determinar la calidad de nuestro código.

Con herramientas de Code Coverage podemos llegar a determinar partes de nuestro código que no han sido testeadas lo suficiente por lo que podrían a llegar a ser susceptibles a errores o vulnerabilidades en el futuro. Normalmente Code Coverage se expresa como un porcentaje en el que un mayor número nos da a entender que hemos testeado casi todas las áreas de nuestro código y por lo tanto los errores serán menos comunes. Algunas herramientas de Code Coverage incluyen JaCoCo, Cobertura, NCover.

Test Management

Utilizadas y diseñadas para facilitar los diversos procesos del testeo de software en una organización. Estas herramientas sirven para organizar y gestionar los diferentes testeos a lo largo del desarrollo del software. Dentro de las Test Management tools podemos encontrar muchas, cada una ofreciendo diferentes funcionalidades para la gestión de las

actividades de testeo, pero en general ayudan a agilizar el testeo, su eficiencia y a la calidad general del software final. Algunas herramientas de Test Management incluyen JQAComplete, Junoone, TestLink.

Performance Testing

Estas herramientas son utilizadas para evaluar cómo se comporta el software, su estabilidad y rendimiento cuanto se les somete a diferentes configuraciones y carga de trabajo.

Dependiendo de la herramienta que usemos esta se especializa en un apartado de los muchos que hay a la hora de hacer Performance Testing para nuestro software. Algunas herramientas de Performance Testing incluyen Silk Performer, LoadRunner, Taurus.

2.4 La Automatización de ejercicios en Catalunya

Las prácticas de programación son esenciales para ayudar a los alumnos a mejorar su experiencia de aprendizaje, muchos profesores aplican varios ejercicios a lo largo de un curso para ayudar a sus alumnos a la hora de enseñarles a como programar. (O. Gimenez, J. Petit, S. Roura, 2011).

Gran parte de estos ejercicios requieren que los alumnos piensen en un algoritmo que dados unos datos iniciales produzcan el resultado deseado para resolver el ejercicio, muchos de estos problemas se llegan a resolver entendiendo las estructuras básicas de datos combinadas con instrucciones básicas para llegar a resolver el problema.

La corrección de estos ejercicios de programación llega a ser tediosos y susceptibles al error humano cuando son corregidos varias veces por el instructor llegando a desear y crear herramientas de automatización para corregir estos ejercicios para así poder evitar tanto los posibles errores que se cometan como también evitar la tediosa tarea de corregir uno por uno los ejercicios (O. Gimenez, J. Petit, S. Roura, 2011).

Ya por 2011 existían (y siguen existiendo) varias plataformas online que proponen ejercicios de programación que automáticamente tienen una corrección al acabar. Estos

jueces en línea son aplicaciones web que ofrecen repositorios de programación y soluciones a sus diferentes ejercicios. De entre todos los jueces online podemos destacar:

- UVa
- Timus
- Sphere
- CodeChef
- TopCoder

El mayor problema de todos estos Jueces es su poca accesibilidad y alta dificultad para estudiantes que empiezan a programar. Muchos de estos jueces se utilizan en entornos altamente competitivos por lo que no son muy adecuados para alumnos que empiezan su aprendizaje. Es por ello por lo que en 2011 aparece Jutge, un juez online diseñado para enseñar a programar a principiantes.

A diferencia de otras plataformas similares, Jutge está diseñado tanto para estudiantes como profesores, integrando ideas tanto de otros Jueces online como de plataformas como Moodle, pudiendo los profesores crear ejercicios, como monitorizar a los alumnos o añadir nuevos cursos.

Jutge es una de las primeras plataformas en Catalunya con ejercicios que automatizan sus correcciones, siendo una herramienta muy apreciada en profesores de programación.

2.5 Github Actions

En 2018 aparece Github Actions para ayudar a los desarrolladores automatizar sus flujos de trabajo, a diferencia de otras herramientas de automatización, Github Actions va un

paso más allá dando la oportunidad de automatizar webhooks [4]además de integrar el flujo CI/CD en sus repositorios.

La automatización en Github Actions se integra en el software a través de eventos, dichos eventos se pueden especificar siendo un evento un nuevo pull request o un nuevo miembro en el repositorio. Todas las automatizaciones se manejan a través de un archivo YAML que se guarda en la carpeta. github/workflows, dicha carpeta es la que se encarga de controlar las automatizaciones de Github Actions.

Dentro de un archivo YAML encontramos varios conceptos, estos incluyen:

- Events: Son los que desencadenan los workflow en los archivos YAML. Estos pueden ser configurados para mirar una o más condiciones a la hora de ejecutarse, pueden también ejecutarse en branches de repositorios de Github
- Jobs: Conjunto de acciones que se ejecutan en paralelo o secuencialmente.
- Steps: Las tareas individuales que forman parte de un Job y ejecutan sus comandos.
- Actions: Es la orden que se ejecuta en un Runner y el elemento clave de Github
- Runners: Un Runner es un servidor de Github. Se queda escuchando por Jobs libres y ejecuta en paralelo para luego reportar el progreso y resultados. Cada Runner puede ser alojado tanto en Github o en un servidor local, los alojados en Github utilizan Ubuntu Linux, Windows y macOS.

Estos conceptos ayudan a los desarrolladores novatos como profesionales a automatizar sus procesos y flujos de trabajo, desde simples órdenes a más complejas dependiendo de la necesidad de cada usuario.

[4] Mensaje automatizado que se envía a una aplicación externa al ocurrir un evento

3. Objetivos y Alcance

El objetivo final de este proyecto es automatizar la corrección de prácticas de EDA utilizando Github Actions.

Nuestra automatización debe identificar y mostrar las clases tests que se ejecutan correctamente y las que no se ejecutan como deberían, generar un archivo PDF que indique que tests han pasado y que tests no, además de subir este mismo archivo a la plataforma moodle en el apartado de correcciones que corresponda a cada alumno que entregue la práctica.

Adicionalmente se quiere conocer el tiempo que ha tardado en corregir estos tests para añadir a la calidad del producto, su uso como su reporte tiene que ser lo más claro posible para facilitar su uso y entender su valor.

El público objetivo de este proyecto serían escuelas y universidades en las asignaturas de programación. La gran parte de este público no dispone de una forma para automatizar sus correcciones de prácticas. No por eso, hay que dejar al resto del público fuera del foco: Debe ser accesible para toda clase de públicos.

Habiendo dejado claro que se debe lograr, vamos a diseccionar los grandes bloques que se van a tener que trabajar para que el proyecto tenga un futuro.

- Aprender y practicar las herramientas encontradas en Github Actions.
- Entender el entorno/plataforma Moodle.
- Diseñar la Automatización.
- Ejecutar la automatización con Github Actions e integración de la corrección en el MoodlE.
- Testear y Validar.

4. Metodología

Este proyecto dispone de la dirección académica por parte de Alfredo Rueda Unsain.

Con Alfredo Rueda Unsain se realizarán reuniones semanales o cada dos semanas para conocer el estado del proyecto, consultar dudas o conocer los siguientes pasos a realizar, estas reuniones se realizan en Zoom y constan de acta que podemos encontrar en la carpeta compartida de Google, también se pueden encontrar en esta carpeta compartida, diferentes versiones del anteproyecto para ver el progreso que ha seguido este proyecto.

Para poder realizar este proyecto es necesario organizar tanto el tiempo como los recursos que tenemos disponibles. Se necesita una planificación con diferentes fases para monitorizar y llevar el proyecto con buen ritmo. Se ha decidido en este proyecto planificar acorde a cuatro objetivos clave.

El primer objetivo en este proyecto es la automatización de la ejecución de clases test en las prácticas de EDA utilizando Github Actions. Para poder conseguir este primer paso será necesario estudiar y aprender a utilizar esta tecnología utilizando la documentación oficial de Github además de varios ejercicios de prueba y error para conseguir este primer objetivo

Una vez conseguido este primer objetivo procederemos al siguiente que es generar un fichero cada vez que se ejecute está automatización. Dicho fichero debe indicar si se ha ejecutado correctamente cada clase test además de indicar el nombre de la persona que lo ha puesto en marcha.

Una vez tengamos este 2o objetivo cumplido se proseguirá con la implementación de este mismo fichero que hemos generado directamente a un moodle propio para añadir valor a nuestro producto. Para conseguir este objetivo necesitaremos una plataforma donde alojar un Moodle para poder tener acceso a las entregas y así poder automatizar las correcciones. Para este paso deberemos nuevamente volver a estudiar la documentación de Github para poder implementarlo con Moodle.

Por último, se ha planteado añadir el tiempo que se ha tardado en ejecutar los tests en el fichero para determinar si se han usado metodologías correctas a la hora de realizar los ejercicios de EDA. Este último paso no es esencial, pero se ha decidido añadir como una forma de reto en el caso de que se disponga tanto tiempo como recursos.

Cada fase tendrá una duración específica debido al tiempo limitado que disponemos para acabar este proyecto, no obstante, los ciclos pueden tener una duración indeterminada a medida que entendamos mejor las herramientas y tecnologías.

5. Requerimientos Funcionales y Tecnológicos

Requerimientos Funcionales:

- La automatización debe funcionar y ejecutarse según como lo hemos diseñado.
- Mostrar los tests fallidos y los tests correctos que debe de ejecutar.
- Recopilar en un archivo los resultados de la corrección de una forma sencilla.

Requerimientos Tecnológicos:

- Poder implementar la corrección de los tests en el moodle del ecampus
- Ofrecer una herramienta

6. Análisis de Referentes

La corrección de ejercicios de programación es prevalente en cualquier ámbito de aprendizaje siendo las universidades y escuelas donde mas sucede.

Como ya se ha visto en el Marco Teórico se han desarrollado herramientas para ayudar a los profesores y alumnos a facilitar la tarea de aprendizaje y corrección como es el caso de CodeChef, TopCoder, Sphere o Jutge aquí en Cataluña

La mayoría de estas herramientas exceptuando Jutge eran poco accesibles para los estudiantes que empezaban a programar, Jutge apareció en Cataluña siendo de las primeras herramientas que incorporaban ejercicios de otras programas similar sino que además ayudaba con la tarea de corrección de los profesores puesto que según Gimenez, Petit y Roura es una tarea tediosa (2011).

Y en 2018 aparece Github Actions ayudando a los desarrolladores a automatizar sus flujos de trabajo en sus repositorios.

Conociendo la tediosidad de la corrección de ejercicios de programación y utilizando Github Actions se decide crear una automatización de las correcciones de ejercicios de programación para solucionar un problema que se demuestra con las varias herramientas de aprendizaje y corrección que se han creado e iterado con el paso de los años

7. Desarrollo

7.1 Automatización de Tests con Github Actions

Inicialmente los ejercicios de EDA fueron creados en Eclipse por lo que la ejecución de las correcciones en Github Actions a través de los archivos *.yml* resultaron imposibles debido a la inexperiencia del desarrollador con estas herramientas.

Para solucionar este primer problema los ejercicios fueron reimportados a IntelliJ y añadiendo Maven para que en las ejecuciones de los test se pudieran utilizar los comandos *"mvn test --batch-mode"* . Inicialmente la corrección de los ejercicios se realizaba en una sola ejecución en la clase *"main"*, se tuvo que separar las correcciones en varios tests de JUnit que se explicara en otro apartado.

Fue necesario añadir Maven Surefire Plugin para poder ejecutar estos tests, sin este plugin los resultados no aparecían.

Figura 1: Añadiendo Maven Surefire. Fuente: Elaboración Propia

Una vez estos cambios fueron implementados las Github Actions se ejecutaban perfectamente con cada push o pull requests que se le pedía al repositorio, pero aún faltaba un test reporter con el que se pudieran ver el resultado de estas correcciones.

Al principio el desarrollador utilizó la herramienta dorny/test-reporter pero esta herramienta requería de un archivo *.xml* en la que normalmente se encuentra el resultado de los tests. Para generar dicho archivo era necesario implementar el Surefire Report de Maven pero debido a problemas con la implementación y fallos constantes a la hora de importar las herramientas dentro del archivo pom.xml se descartó tanto implementar el test reporter de dorny.

El siguiente test reporter que se decidió usar fue ScaCap, tanto su implementación como uso fue mucho más sencillo debido a que no requería de ningún archivo previo para poder generar los reportes.

```yaml
name: Test Reporter

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:

    runs-on: ubuntu-latest

    permissions:
      checks: write
      contents: read

    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 21
      uses: actions/setup-java@v3
      with:
        java-version: '21'
        distribution: 'temurin'
        cache: maven

    - name: Build and Run Tests
      run: mvn test --batch-mode --fail-at-end
    - name: Publish Test Report
      if: success() || failure()
      uses: ScaCap/action-surefire-report@v1.7.3
```

Figura 2: Añadiendo ScaCap a Github Actions. Fuente: Elaboración Propia

Dentro de la ejecución de la Github Action podemos dividir su ejecución en diferentes partes

- Name: Como bien indica es el nombre que se le pone al “Workflow” o flujo de trabajo, en este caso se ha decidido ponerle de nombre Test Reporter.

- Triggers: Indica cuando se ejecuta esta Github Action, en nuestro caso ocurren (“on”) cuando se decide hacer un push o pull request en la main Branch.

- Jobs: Todo el trabajo que se tiene que realizar para completar la Action.
- Runs-on: Indica que esta Action se deberá ejecutar en la última versión de Ubuntu que pueda proveer Github.
- Permissions: Especifica los permisos de la Action, en nuestro caso dejamos que pueda leer y escribir los contenidos necesarios.
- Steps: Los pasos en orden para realizar la Action.

En nuestro caso tenemos dos steps para acabar la acción, nuestro primer Step no es mas que un paso previo para acabar de configurar la maquina seteando un JDK 21 para poder automatizar los tests. El segundo paso es la ejecución de los tests invocando el comando *mvn test --batch-mode --fail-at-en.*

Una vez invocado y completado este comando se publica el resultado de los tests usando la herramienta de ScaCap que se encarga de mostrarnos un reporte sencillo de los tests que funcionan como los que no.

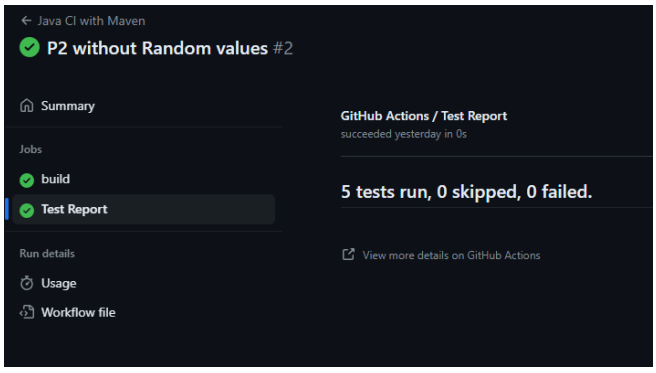


Figura 3: Test Reporter. Fuente: Elaboración Propia

Tanto la automatización de los tests como un reporte sencillo fueron logrados durante esta primera etapa de desarrollo.

7.2 Implementación de JUnit en las prácticas de EDA

Las correcciones de los ejercicios al principio se realizaban en una única ejecución de la clase "*main*" por lo que fue necesario separar todas las correcciones en diferentes tests de JUnit para poder integrarlo mejor con Maven y Github Actions.

A la hora de crear los tests de JUnit se utilizaba como código fuente las correcciones que normalmente se utilizarían en "*main*" pero adaptadas para los test de JUnit, en total se han desarrollado e implementado los tests para un total de 4 practicas.

7.2.1 VehiclePark

Esta primera práctica consistía en implementar un sistema de gestión de vehículos utilizando una colección para guardarlos. Dentro de esta implementación encontramos funcionalidades como:

- Recuento del número de vehículos privados o públicos
- Confirmación de si un vehículo existe dentro de la colección
- Añadir vehículo si se cumplen las condiciones
- Eliminar vehículo si se cumplen las condiciones
- Confirmación de si hay vehículos con peso peligroso

En total se han desarrollado 8 test de JUnit de los cuales 6 pertenecen a la clase que implementa la gestión (VehicleParkImplementation) y 2 a la clase Plate para saber si los métodos compareTo y equalsTo funcionan como se especifican.

```java
@Test
void enter() {

    NullPointerException exceptionNull = assertThrows(NullPointerException.class, () -> {
        vp.enter((Vehicle) null);
    });
    AlreadyStoredException exceptionStored = assertThrows(AlreadyStoredException.class, () -> {
        vp.enter(new PrivateVehicle(plates[1], weight: 10, owners[2], seats: 4, wheels: 4));
    });

    assertEquals( expected: "Null parameter", exceptionNull.getMessage(), message: "Exception message should match");
    assertEquals( expected: "Already there", exceptionStored.getMessage(), message: "Exception message should match");


}
```

Figura 4: Test enter() de VehicleParkImplementationTest. Fuente: Elaboración Propia

7.2.2 AirRegisterImp

La segunda práctica es muy similar a la primera pero esta vez se usa un Map para guardar los aviones en vez de una colección, los aviones se guardan dentro de una compañía de aviones en orden ascendente del año en el que fueron creados.

Dentro de esta práctica encontramos funcionalidades como:

- Encontrar compañia
- Añadir compañía si se cumplen las condiciones
- Añadir avión a la compañía si se cumplen las condiciones
- Eliminar avión de la compañía si se cumplen las condiciones
- Encontrar compañía

En esta práctica se han desarrollado un total de 5 test de JUnit, todos ellos perteneciendo a las funcionalidades de AirRegisterImp.

```
nnikolaev97
@Test
@Order(1)
void addCompany() {
    for (int i=0; i<companies.length; i++)
        assertTrue(registre.addCompany(companies[i]), message: "These companies should be added");

    for (int i=0; i<companies.length; i++)
        assertFalse(registre.addCompany(companies[i]), message: "addCompany. We should not add companies that are already in the register");
}
```

Figura 5: Test addCompany() de AirRegisterImpTest. Fuente: Elaboración Propia

7.2.3 SocialNetworkServiceImp

Esta tercera práctica se dedica a gestionar una red social, al igual que la anterior practica esta también utiliza un Map para guardar las amistades entre usuarios. Esta practica en especial tiene un algoritmo sencillo que indica las amistades entre usuarios para llegar a cierto usuario (Ej. Usuario 1 es amigo con Usuario 2 y este con Usuario 99 por lo que hay un salto mínimo de 1)

Dentro de esta práctica encontramos funcionalidades como:

- Añadir una nueva persona a la red social si se cumplen las condiciones
- Conocer el numero de personas en la red social
- Añadir amistad entre dos usuarios existentes
- Conocer las amistades de una persona en particular
- Camino mínimo de amistades entre usuarios

En esta práctica se han desarrollado un total de 5 test de JUnit, todos ellos perteneciendo a las funcionalidades de SocialNetworkServiceImp.

```java
@Test
void addPerson() throws PersonAlreadyExistsException {

    PersonAlreadyExistsException exceptionPersonExists = assertThrows(PersonAlreadyExistsException.class, () ->
            socialNetwork.addPerson(new Person( name: "Person 1"))
    );

    Assertions.assertEquals( expected: "Person Person 1 already exists", exceptionPersonExists.getMessage(),  message: "Exception message should match");

    socialNetwork.addPerson(new Person( name: "Person 1001"));
    Assertions.assertEquals( expected: 101, socialNetwork.getPeople().size(),  message: "Add Person should've added Person 1001");
}
```

Figura 6: Test addPerson() de SocialNetworkServiceImpTest. Fuente: Elaboración Propia

7.2.4 SpotifyImpl

La cuarta y última práctica es un sistema de gestión de canciones, al igual que las anteriores se utiliza un Map para guardar esta información siendo la clave principal la lista de reproducción donde se guarda la canción. Esta practica dispone de muchas mas funcionalidades que las anteriores además de tener dos versiones de ejecución utilizando un HashMap y un TreeMap comparando por el año ascendente de la creación de la canción.

Dentro de esta práctica encontramos funcionalidades como:

- Añadir una nueva lista de reproducción
- Añadir canción a una lista de reproducción
- Devolver todas las canciones de una lista de reproducción
- Sortear por género, artista, canción, duración, mejores artistas, …etc.

En esta práctica se han desarrollado un total de 30 test de JUnit, donde la mitad pertenecen a la clase SpotifyImplTreeMap y la otra mitad a SpotifyImpl.

```
nnikolaev97
@Test
void addPlaylist() {
    assertTrue(spotify.addPlaylist(playlist1));
    assertFalse(spotify.addPlaylist(playlist1));
}
```

Figura 7: Test addPlaylist() de SpotifyImplTest. Fuente: Elaboración Propia

7.3 Automatización de las entregas con Github Actions

Una vez desarrollados los JUnit test y la Github Action dedicada a ejecutar y mostrar los reportes se pasa a la creación de un Moodle para poder realizar un reporte automático de las entregas y ejercicios del Moodle con las github Actions.

Similar a las entregas que encontramos en el aula virtual del tecnocampus el objetivo de esta fase es conseguir que además de mostrar el reporte de los tests se pueda modificar la nota del estudiante dependiendo de si ha conseguido ejecutar correctamente todos los tests de JUnit o no.

7.3.1 Instalación de Moodle 4 con WampServer

Para lograr este objetivo se necesitaba de un aula virtual donde alojar entregas del desarrollador y poder experimentar con el entorno. Se ha decido utilizar WampServer no por el conocimiento previo que tuviera el desarrollador con dicha herramienta sino por la vasta cantidad de información que se ha encontrado de cómo utilizarla (Google, Foros, Youtube). WampServer es una herramienta de desarrollo web compatible con Windows que utiliza Apache para procesar las peticiones Web, MySQL como sistema de base de datos y PHP para crear el contenido de la propia web.

Una vez instalado WampServer se procedió a la instalación de Moodle, una herramienta de aprendizaje para educadores, escuelas y universidades perfecta para realizar nuestras pruebas.

7.3.2 Port Forwarding

Teniendo ya Moodle4 en nuestra maquina aun nos queda el paso de hacer publico nuestro servidor para poder conectar con Github a través de una petición web. Para ello se ha hecho Port Forwarding [5]con el ordenador personal del desarrollador para conseguir este objetivo, utilizando el puerto 80, nuestra IP privada además de crear reglas adicionales de entrada en el firewall para poder permitir el acceso externo.

Port Forwarding puede ser peligroso ya que se abren puertos que se pueden llegar a comprometer, pero se ha decidido utilizar este método porque era el mas sencillo y gratuito a la hora de implementar y desarrollar la web de moodle.

Durante este último se ha tenido que cambiar el archivo config.php de Moodle4 para que los usuarios no locales tuvieran acceso a la página web.

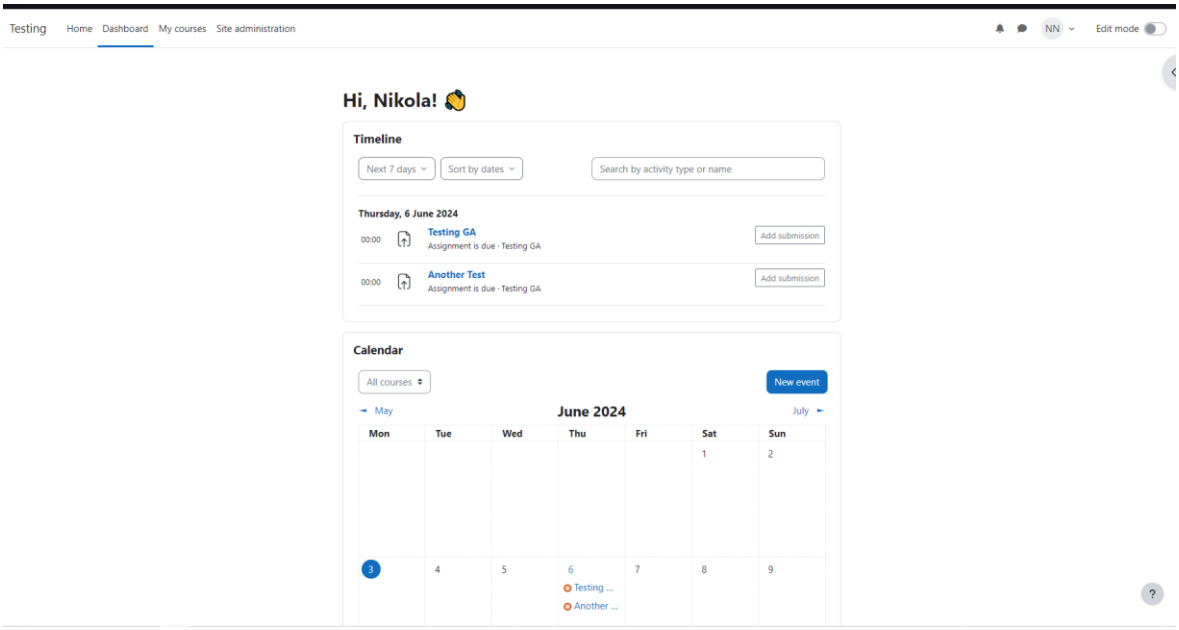


Figura 8: Moodle4 propio usando Port Forwarding. Fuente: Elaboración Propia

[5] Redirección de puertos para permitir el acceso de una persona o equipo externo a una dirección privada dentro de una LAN

7.3.3 Configuración del Moodle

Antes de implementar el Workflow que permite puntuar notas de estudiantes era necesario crear un curso con una entrega dentro de la página del Moodle además de añadir estudiantes a los que podamos puntuar.

La adición de estos elementos fue sencillo e intuitivo en el modo edición que Moodle4 proporciona a todos los administradores.

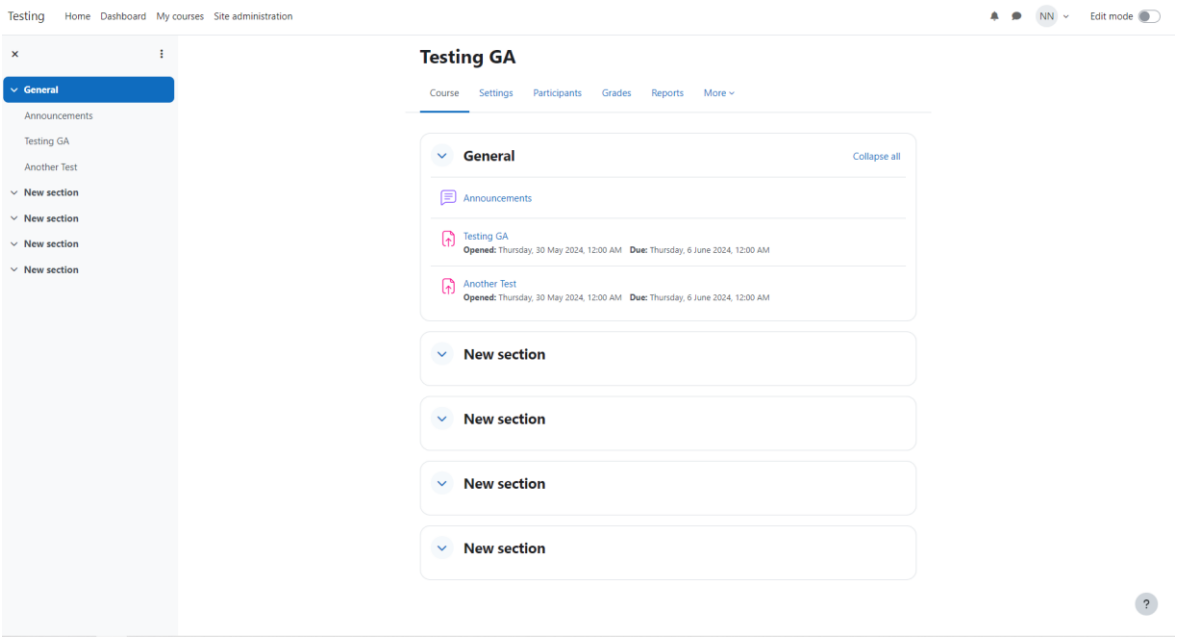


Figura 9: Curso con entregas en Moodle4. Fuente: Elaboración Propia

El ultimo paso era permitir las peticiones Web, empezando por habilitar el REST protocol.

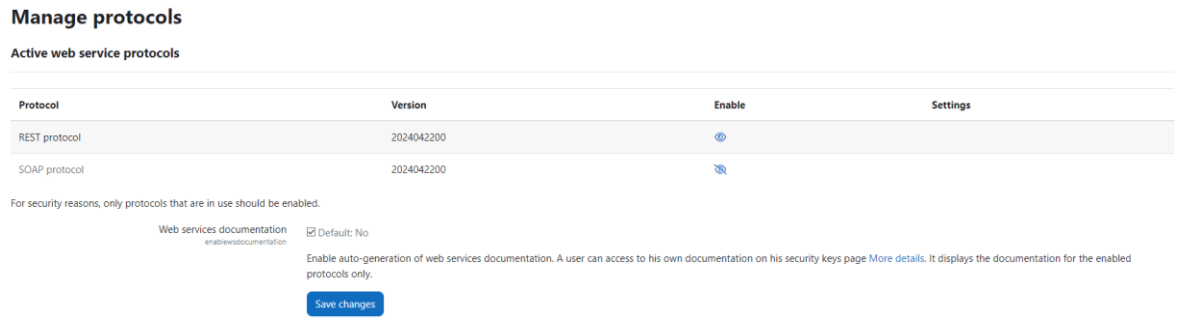
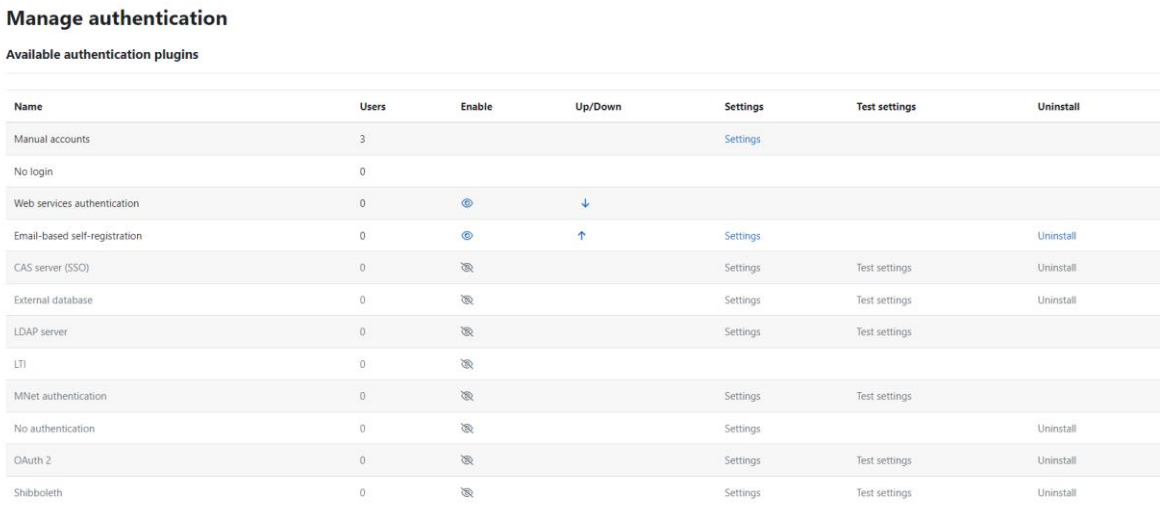


Figura 10: Habilitando REST Protocol en Moodle4. Fuente: Elaboración Propia

A la vez que Web Services Authentication.



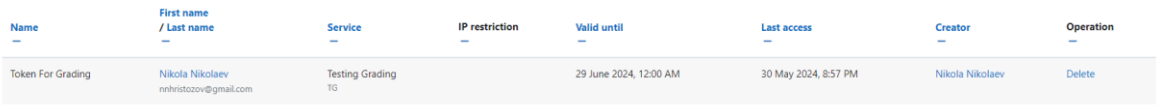
Manage authentication

Available authentication plugins

Name	Users	Enable	Up/Down	Settings	Test settings	Uninstall
Manual accounts	3			Settings		
No login	0					
Web services authentication	0		↓			
Email-based self-registration	0		↑	Settings		Uninstall
CAS server (SSO)	0			Settings	Test settings	Uninstall
External database	0			Settings	Test settings	Uninstall
LDAP server	0			Settings	Test settings	
LTI	0					
MNet authentication	0			Settings	Test settings	
No authentication	0			Settings		Uninstall
OAuth 2	0			Settings	Test settings	Uninstall
Shibboleth	0			Settings	Test settings	Uninstall

Figura 11: Habilitando Web Services Authentication en Moodle4. Fuente: Elaboración Propia

Y por último la creación de un Token Web para el uso de las peticiones web, este token tenía la funcionalidad de guardar las notas de los estudiantes, este token se genera para el profesor que debería puntuar a los alumnos.



Name	First name / Last name	Service	IP restriction	Valid until	Last access	Creator	Operation
Token For Grading	Nikola Nikolaev nnhristozov@gmail.com	Testing Grading TG		29 June 2024, 12:00 AM	30 May 2024, 8:57 PM	Nikola Nikolaev	Delete

Figura 12: Token Web en Moodle4. Fuente: Elaboración Propia

Una vez acabados todos estos pasos solamente quedaba crear la Github Action que se encargara de asignar una nota a la entrega de cada usuario.

7.4 Automatización de calificaciones con Github Actions

Ya con todos los pasos previos resueltos y seteados tan solo quedaba desarrollar el workflow que permita la corrección de las notas de los usuarios.

```yaml
grade-assignment:
  runs-on: ubuntu-latest
  needs: build

  steps:
  - name: Checkout repository
    uses: actions/checkout@v2

  - name: Grade Assignment
    if: ${{ needs.build.result == 'success' }}
    env:
      MOODLE_URL: ${{ secrets.MOODLE_URL }}
      MOODLE_TOKEN: ${{ secrets.MOODLE_TOKEN }}
      ASSIGNMENT_ID: 2
      USER_ID: 4
      GRADE: 85
      APPLY_TO_ALL: 0
    run: |
      curl -X POST "${MOODLE_URL}/webservice/rest/server.php" \
        -d "wstoken=${MOODLE_TOKEN}" \
        -d "wsfunction=mod_assign_save_grade" \
        -d "moodlewsrestformat=json" \
        -d "assignmentid=${ASSIGNMENT_ID}" \
        -d "userid=${USER_ID}" \
        -d "grade=${GRADE}" \
        -d "attemptnumber=-1" \
        -d "addattempt=0" \
        -d "workflowstate=graded" \
        -d "applytoall=${APPLY_TO_ALL}"
```

Figura 13: Workflow de asignación de nota. Fuente: Elaboración Propia

Este workflow extiende del que ya teníamos previamente porque necesita saber si el resultado de los Tests es correcto para que se pueda realizar. En el caso de que todos los tests sean correctos se hace una petición web a nuestro servidor de Moodle.

En el caso de querer asignar una nota a un estudiante según la documentación de Moodle4 es necesario proveer de un token que se la haya generado dicho permiso, la ID del estudiante a cuál queremos poner la nota, la ID de la tarea a la cual pondremos nota, una nota y una confirmación de si es un trabajo grupal o no.

En el caso del token se proporciona a los estudiantes el token que se ha generado para el profesor por lo que para evitar uso indebido se utiliza la encriptación de Github Secrets para evitar problemas.

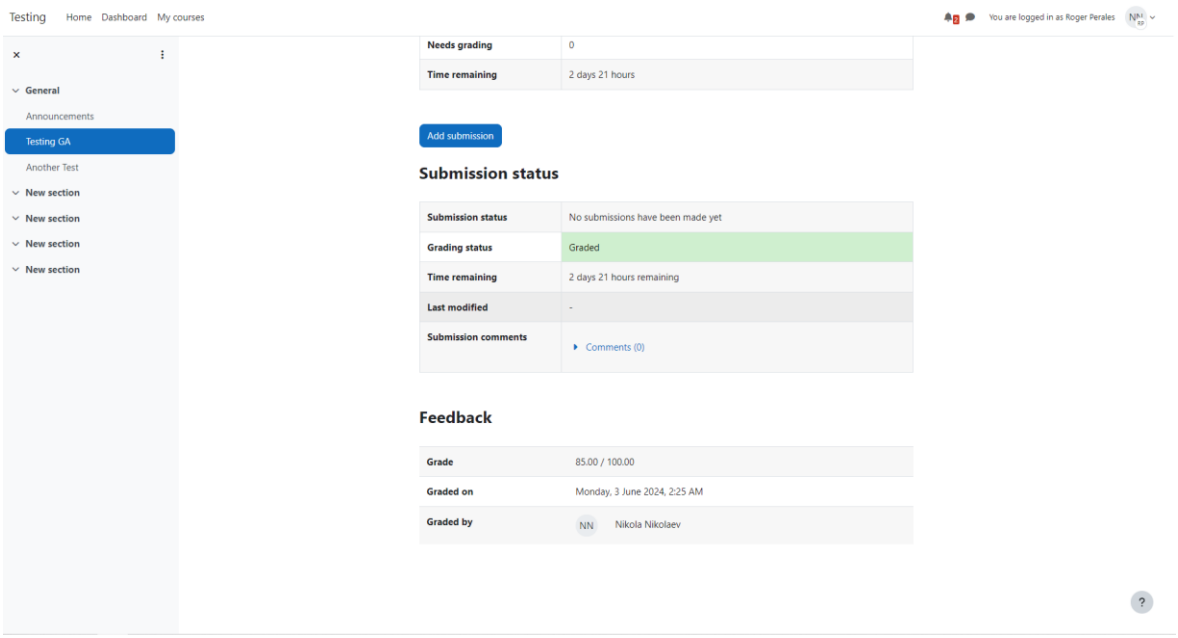


Figura 14: Usuario con nota. Fuente: Elaboración Propia

Una vez proporcionada la información necesaria se comprueba si la petición ha sido realizada correctamente.

8. Conclusiones

El objetivo principal de este proyecto era proveer de una herramienta de automatización de correcciones de prácticas de EDA.

Empezando por el primer objetivo planeado se ha conseguido automatizar las correcciones de los tests de JUnit a cada práctica que hemos decidido aplicar el workflow de Github, no obstante, la parte en la que se generaba el archivo mostrando todos los tests correctos e incorrectos además de mostrar el tiempo de ejecución no se ha desarrollado, en su lugar se ha decidido utilizar una herramienta de reporte de tests que indica en el propio repositorio el resultado de estas pruebas.

Continuando con el objetivo de puntuación de las notas en Moodle se ha conseguido automatizar que cada usuario tenga una nota en el caso de que los tests se ejecuten correctamente. Aun así, hay insatisfacción con esta parte debido a que cada usuario dispone de una ID diferente por lo que seria necesario editar el workflow para cada usuario.

Se puede concluir que el objetivo inicial se ha logrado satisfactoriamente, aun así, se puede lograr mejoras en partes que no se han podido llegar a desarrollar y que se explican mejor en el apartado de posibles ampliaciones.

9. Futuras Ampliaciones

En el apartado de conclusiones se muestra insatisfacción a la hora de automatizar las notas de los estudiantes ya que se necesita cambiar manualmente la ID de cada usuario para poder corregir a todos.

Debido a esta inconveniencia se plantea enlazar la ID de cada usuario de Github con la ID que se tiene con Moodle. Para ello sería necesario implementar Github Classroom una herramienta educacional de Github.

10. Bibliografía

Benford, S., Burke, E., Foxley, E., Gutteridge, N., & Zin, A. M. (1993). Early experiences of computer-aided assessment and administration when teaching computer programming. *Research in Learning Technology*, *1*(2).

Umar, Mubarak Albarka, and Chen Zhanfang. "A study of automated software testing: Automation tools and frameworks." *International Journal of Computer Science Engineering (IJCSE)* 6 (2019): 217-225.

What-is-GitHub.Actions_.Benefits-and-examples.pdf

Ismail, M. H., & Lakulu, M. M. (2020). A critical review on recent proposed automated programming assessment tool. *Psychology and Education*, 1049-1060.

Shukur, Z. (1999). *The automatic assessment of Z specifications* (Doctoral dissertation, University of Nottingham).

Aldriye, H., Alkhalaf, A., & Alkhalaf, M. (2019). Automated grading systems for programming assignments: A literature review. *International Journal of Advanced Computer Science and Applications*, *10*(3).

Petit, J., Roura, S., Carmona, J., Cortadella, J., Duch, J., Gimnez, O., ... & Venkataramani, D. (2017). Jutge. org: Characteristics and experiences. *IEEE Transactions on Learning Technologies*, *11*(3), 321-333.