Universidad de Valladolid

Escuela de Ingeniería Informática de Valladolid

Servicio de análisis y ejecución de programas

Trabajo de Fin de Grado
Grado en Ingeniería Informática
Mención: Tecnologías de la Información y comunicación

Autor: Andrés Cabero Mata
Tutor: Valentín Cardeñoso Payo

28 de Junio de 2021

Dedicado a mi familia por apoyarme y a mis amigos por aguantarme.

Agradecimientos

Quiero agradecer a mi familia por apoyarme durante todo el periodo que conlleva la carrera, a los amigos que me han ayudado a pasar cualquier problema y me han alegrado cuando me ha hecho falta, y a los compañeros y profesores que me han ayudado a lo largo de todos los estudios.

Resumen

Cuando se requiere ejecutar un programa para comprobar que este funciona correctamente puede ser muy tedioso preparar ese conjunto de pruebas necesario. Es peor aún cuando se tienen que ejecutar un gran conjunto de programas similares que cumplen la misma función y tienen distintas estructuras que no facilitan la automatización. Este trabajo tiene como objetivo implementar una solución a ese problema que pueden tener, por ejemplo, muchos profesores que imparten asignaturas que conllevan proyectos de una escala superior, que pueden tener estructuras un poco más complejas que un único fichero. Para crear este servicio se intentará hacer una base de lo que sería un ejecutor y analizador de programas, empezando por programas en lenguaje Java. Se intentará que sea lo más escalable posible, haciendo que el diseño sea seguro utilizando herramientas que proporcionen colas de trabajo que permitan ejecutar los procesos necesarios sin elevar la carga del sistema, y software de encapsulamiento para la ejecución segura del código. Se busca hacer un servicio web en el que se pueda mandar un paquete de programa que contenga el código a ejecutar con una configuración de ejecución establecida para el tipo de programa y devuelva unos resultados sobre la ejecución de unos tests preestablecidos por el administrador del servicio.

Abstract

When it's needed to test how a program works, whether it fulfills its requirements or not, it can be very tedious to make the necessary test set. It gets worse if you have to test the same program with multiple structures, it performs the same function, but its differences hinder the automatization of the process. This work has as an objective to implement a solution to this one problem that, for example, occurs to a lot of teachers of programming courses that have projects with a heavier scale with structures more complex than a single file. To implement this service, it will be tried to make a basis of a program runner and analyzer, starting with the Java language. Scalability will be one of the biggest factors to take care of, making a safe design using tools that provide job queues to run code without making a big load of jobs to the service, and using a container software to have a controlled and safe run. It's wanted to make a web service where you can send a program package with the code to run, using an established run configuration for the kind of program, and get in return the results provided by the run of a test set, presetted by the administrator of the service.

Índice general

Índice de cuadros

Parte I

Objeto, Concepto y Método

Capítulo 1

Introducción

1.1. Introducción

La propuesta para este trabajo de fin de grado es la realización de un servicio web capaz de compilar, ejecutar, y analizar un programa en lenguaje Java. Para la compilación y ejecución se tendrán que tener establecidos previamente unos parámetros para conseguir unos requisitos mínimos en la confianza de la ejecución y el tratamiento del programa. La creación de este proyecto conlleva un estudio y aprendizaje de distintas herramientas necesarias para el mismo, cuya elección se basa en la importancia y facilidad de reemplazo que pudieran tener en el propósito del proyecto.

El servicio será capaz de ejecutar un conjunto de pruebas que verifiquen el funcionamiento del programa, tras ello realizará un análisis inicial del código que pueda determinar el nivel de calidad del código pudiendo compararlo con otras versiones distintas del programa. La siguiente fase es poder hacer un análisis estructural del código, dando los puntos mejora o incluso de error si este no se estuviera ejecutando correctamente.

1.2. Motivación

Normalmente cuando se corrige un ejercicio de programación a un alumno, excepto cuando se trata de una pequeña cantidad de líneas de código, es muy complicado para el profesor el ayudar al alumno a encontrar en que punto falla su código, ya sea por un error de escritura en el lenguaje, o más difícil aun, por un fallo en la lógica. Esto hace complicado a los profesores dar una retroalimentación concreta y correcta de los ejercicios de programación a los alumnos. Viéndolo desde el punto de vista del alumno a veces se tiene cierto desconocimiento de ciertas notas y de donde encontrar el punto de mejora, por lo que una retroalimentación con ciertas pautas podría ayudar a reparar esos fallos para continuar con el aprendizaje.

1.3. Estructura

Este documento se compone de tres partes, comenzando por una breve introducción con la intención de situar al lector en un contexto más amigable. En el siguiente capítulo se explicarán los distintos objetivos y el alcance el proyecto, y tras ello se expondrá la metodología de trabajo. La siguiente parte consiste del marco conceptual y teórico, así como el contexto del proyecto, mostrando distintas soluciones existentes al problema. En la tercera y última parte se mostrará todo el proceso de análisis, diseño e implementación del proyecto.

Capítulo 2

Objetivos y Alcance

2.1. Objetivos

1. Realizar una aplicación capaz de compilar y ejecutar código Java.
2. Proporcionar seguridad a la máquina host de la aplicación en el momento de ejecutar los programas.
3. Establecer y calcular unas medidas de calidad de código Java genéricas y unas relativas a un programa concreto.
4. Establecer la base para un análisis estructural de código.
5. Aprender a utilizar herramientas de contenerización para programas.
6. Aprender y estudiar la actualidad de automatización de ejecución y testeo de programas.

2.2. Alcance

En este proyecto se intentará crear una herramienta que sirva como base para posibles futuras expansiones que proporcionen funcionalidad más completa a la hora de automatizar la ejecución y análisis de programas en distintos lenguajes. Por ello se considera imprescindible ser consciente de la escalabilidad y/o reemplazabilidad de las distintas herramientas y modelos que se utilicen a lo largo de la implementación del proyecto, así como buscar la mayor modularidad para los distintos casos de uso posibles.

Capítulo 3

Metodología

El trabajo se realizará mediante un servicio online que proporcionará su propia interfaz web donde se podrán iniciar las ejecuciones y consultar los resultados. Para realizar este servicio web se utilizará el framework Flask[8] de Python debido a su simplicidad y facilidad de aprendizaje, pudiendo centrar esfuerzos en otros puntos del proyecto. Este servidor será el encargado de preparar el programa para su ejecución y testeo. Para la parte más importante del proyecto, la ejecución, se utilizará Docker[4] como principal herramienta para la encapsulación de la ejecución añadiendo así un nivel mayor de seguridad. Pese a que en un principio para la ejecución de los tests de forma ordenada se proponía utilizar una herramienta de automatización como Jenkins[6], se ha decidido simplificar por la dificultad de aprendizaje en la coordinación entre el pipeline de Jenkins y la ejecución de los programas en Docker, por lo que se utilizará un script (en lenguaje bash) como coordinador para la compilación y ejecución de los tests.

3.1. Fases y costes

El proyecto consiste de unas ciertas fases las cuales, aunque variables, se prevén realizar en unos ciertos tiempos. La fase más variable es posiblemente la de análisis y aprendizaje, pues al no haber una forma fija de plantear el problema, la inexperiencia con las posibles herramientas a utilizar, y el tiempo requerido en el estudio de las que al final sea necesario descartar, no es posible saber con certeza que cantidad de horas serán necesarias. Las tareas se presentan en rangos de diferencia de dos semanas según la variabilidad esperada en el estudio inicial del proyecto. También se intenta presentar las distintas fases en el orden esperado, aunque por facilidad de implementación es muy fácil ver que muchas fases serán paralelas o entrelazadas.
Las fases y duración prevista de cada una (Duracion mínima y máxima previstas) son las que se detallan en el 3.1.

Nombre de actividad	Meses
Análisis y aprendizaje de las herramientas a utilizar	0.5 - 1
Creación de un servicio web capaz de compilar y ejecutar programas Java.	1 - 1.5
Implementación de la herramienta de contenerización en el proyecto	1 - 1.5
Inclusión de un primer análisis del código mediante medidas de calidad.	0.5 - 1
Análisis estructural básico del código	0.5 - 1
Inclusión de otro lenguaje como Python, C... (Opcional)	

Cuadro 3.1: Fases de desarrollo del proyecto previstas.

Parte II

Marco Conceptual y Contexto

Capítulo 4

Marco Conceptual

La forma de ejecutar un programa es una parte muy importante del proceso de desarrollo de un producto informático. Cuando se crea una nueva pieza de software siempre se hace teniendo en mente como se va a utilizar, pudiendo ser una llamada a un servicio web, un ejecutable, una aplicación móvil, una página web, etc. El método varía según los requisitos que tenga el programa a desarrollar, siendo estos requisitos los que fuerzan el diseño del modelo a descartar ciertas posibilidades, dando a elegir al desarrollador, normalmente, pocas opciones de utilidad para los casos de uso que se quieren solventar.

Aun así este problema no conlleva más que un único (o a veces varios) modelo de ejecución del programa, por ejemplo en el caso de un servicio web, diseñar las llamadas que se van a crear para los distintos casos de uso que se soporten. Por ello la ejecución final de un programa como producto no es algo que se suela automatizar, sino que se utiliza a través de un método definido, ya sea manual o hecho por un disparador. Es mucho más común intentar automatizar un proceso de ejecución cuando este pueda variar a lo largo del tiempo, como puede ser el hacer pruebas a un software aun en desarrollo.

4.1. Testing

Cuando se desarrollan herramientas software es muy común ir realizando pruebas a los distintos módulos de código y distintas estructuras, ya sea para comprobar su funcionamiento lógico, si siguen el modelo del programa, si tienen vulnerabilidades que haya que subsanar, y un largo etc. Para estos casos se suelen realizar baterías de pruebas, normalmente a través de herramientas del propio lenguaje de implementación de la herramienta, o dependiendo del software, mediante pruebas manuales de uso. En el primer caso se suelen hacer una larga lista de métodos que cubran la totalidad o una gran parte de la implementación, dando validez a la misma. Por lo general se utilizan herramientas que automaticen el proceso de creación y ejecución de estas pruebas, como por ejemplo la librería PyTest[15] de Python. Sin embargo, el problema puede seguir en aplicaciones más grandes que requieran pasar un ciclo de pruebas más complejo.

En el proceso de desarrollo de un producto en una empresa privada o un servicio público se suele seguir ciertos métodos de trabajo que aseguren un cierto nivel de confianza en la calidad del proyecto. En las distintas etapas del desarrollo y versiones del producto este se suele someter a pruebas tanto de ejecución como de análisis, debido a ello existen una gran cantidad de herramientas profesionales para facilitar la automatización y realización de estas pruebas, como por ejemplo el ya mencionado Jenkins, que proporciona una forma relativamente fácil de crear un proceso único para el producto en el que se establecen una serie de etapas a través de un pipeline. Una vez cumplidas estas etapas, si

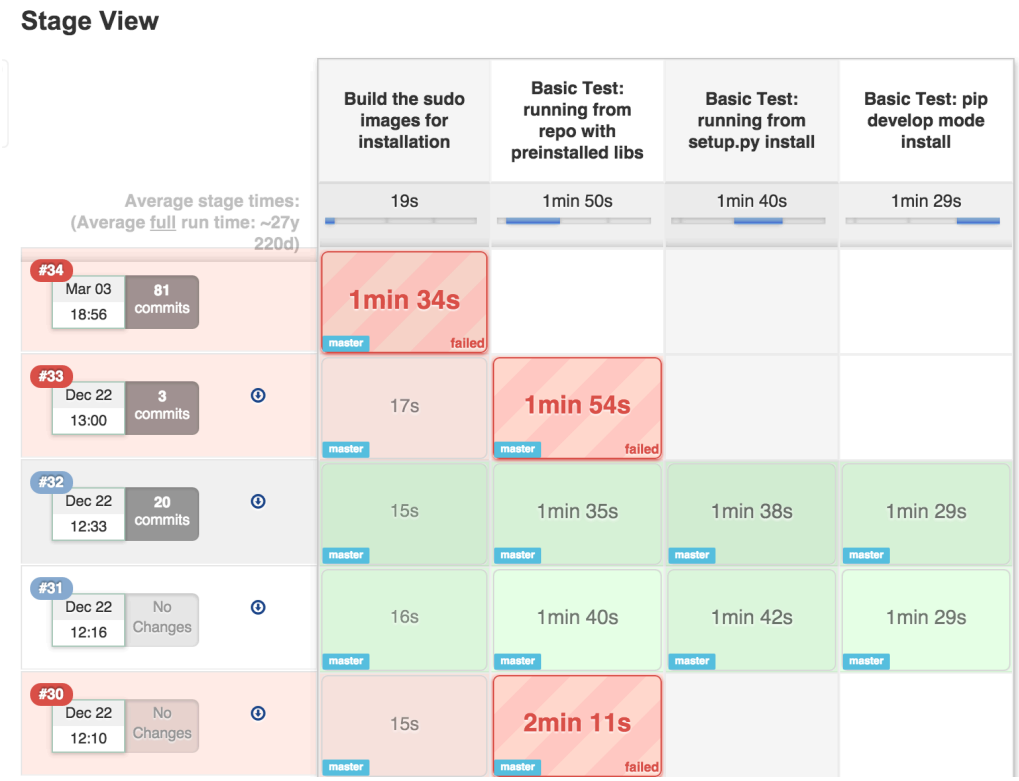

Figura 4.1: Ejemplo de pipeline para ejecución de pruebas de un programa en Jenkins[1].

están diseñadas correctamente, aumenta considerablemente la confianza en el funcionamiento del programa.

4.2. Otros casos

Entonces, ¿cuál es el problema? Si existen herramientas que facilitan tanto el testear el código, ¿para qué preocuparse por ello?

Por supuesto el tipo de herramientas anteriormente mencionadas, que buscan dar soporte al desarrollo son muy útiles, pero no cumplen todos los casos de uso. Hay veces que incluso profesionales que se dedican casi en exclusiva a desplegar software pueden llegar a tener un mal rato con estas herramientas debido a la complejidad que ofrecen, por lo que es normal que en otro tipo de casos más sencillos no sea conveniente utilizar estas herramientas, pues requieren de un trabajo mayor al soporte que dan al problema.

4.2.1. Competiciones

Las competiciones de programación es un ejemplo típico y bastante común a día de hoy. En este tipo de competiciones se suele establecer un problema de programación en el que se establecen

[1] Imagen de DevOps top 10 Best Practices for Jenkins [3]

ciertos parámetros y se busca un resultado concreto. Independientemente de la complejidad del problema, puede llegar a haber mucha participación, lo que conlleva a muchas soluciones diferentes a un mismo problema, con implementaciones que pueden ser totalmente distintas, pero que, si dan el resultado correcto, debería ser el mismo.

Figura 4.2: Code Jam, concurso competitivo de programación de Google[2].

En el caso más mínimo en el que se quisiera utilizar el tipo de herramientas que se utilizan para el desarrollo de un producto, por cada programa con distinta estructura que se quisiera probar haría falta hacer un fichero de configuración, lo cual no es nada práctico si tenemos varios programas, ya que, pese a que su propósito y su modelo pueda ser similares o idénticos, la estructura que tenga hecha la implementación (incluso en el mismo lenguaje) puede afectar a la configuración que se tengan que hacer de las distintas herramientas. Este problema se puede evitar haciendo un conjunto de pruebas independiente a la implementación y que sea resultadista, comprobando el funcionamiento macro del programa y teniendo en cuenta exclusivamente los resultados. Estas herramientas, por supuesto, son más que capaces de funcionar para este tipo de pruebas, y es posible que se utilicen en competiciones grandes como la Google Code Jam, debido a la gran cantidad de participantes (+30.000 en 2019 [2]), pero para competiciones más pequeñas requiere una labor de estructuración mayor a la necesaria.

4.2.2. Evaluación escolar

Si hay otro caso típico en el que se busca de una cierta automatización en la ejecución de varios programas con distintas implementaciones es cuando un profesor necesita evaluar muchos programas hechos por los alumnos y por tanto con estructuras que pueden ser totalmente distintas. Estos, en general, suelen ser mucho más restrictivos que los de las competiciones, pues el profesor suele buscar de un control mayor en las implementaciones, pues se busca asegurar que se han adquirido unos conocimientos. Aun así, en los casos en los que la cantidad de alumnos es demasiado grande, puede dar una gran cantidad de trabajo realizar las pruebas de forma manual, por lo que lo más común es establecer unas directrices tanto en la forma y formato de los parámetros de entrada, como

[2]Imagen de Google’s Coding Competitions [1]

en la forma en la que se presenta la salida. Esto facilita la tarea, pero aun así, no está completando la tarea de un educador, pues se busca ayudar al alumno a avanzar en los fallos que haya podido tener, y un resultado negativo en una ejecución no aporta ninguna información realmente valiosa para subsanar esos problemas.

Capítulo 5

Soluciones Existentes

Existen varios servicios que se dedican a la ejecución online de programas, teniendo en cuenta un programa como un único fichero, pese a que el diseño de este proyecto considera un programa como un conjunto de archivos en un paquete, el sistema de este tipo de herramientas realmente es similar, compilar el código y ejecutar unas pruebas en el código que demuestran que es funcional.

La solución de uso más real es la de preparar un conjunto de herramientas de las explicadas anteriormente para un único proyecto, o hacer un diseño inicial con esas herramientas que pueda ser reutilizado mediante una configuración específica para cada nuevo proyecto, dando así la posibilidad de reutilizar el trabajo realizado en la configuración de las herramientas.

Pese a esto, existen soluciones que siguen un sistema parecido al que se quiere plantear en el proyecto, si se hace una pequeña búsqueda para ejecutar de forma online código existen muchos IDE online que realizan esa función, pero de forma más específica cabe destacar un proyecto el cual sigue un diseño interno muy parecido al que se propone, con la diferencia de que da servicio a una utilidad muy concreta como son los exámenes de programación de forma online.

5.1. CodeRunner

CodeRunner es un proyecto open-source gratuito que, en forma de plugin para Moodle, puede ejecutar una gran variedad de lenguajes en un formato de preguntas (5.1)

El uso que proponen para un examen realizado desde Moodle es establecer un problema de programación, el cual se escribe directamente sobre un input de texto que da el propio plugin y poder ejecutar en tiempo real el código, dando rápidamente un resultado según los tests establecidos para el problema y dejando al alumno que realiza el examen que pueda modificar el código y ejecutar de nuevo, típicamente por una pequeña multa en la nota.

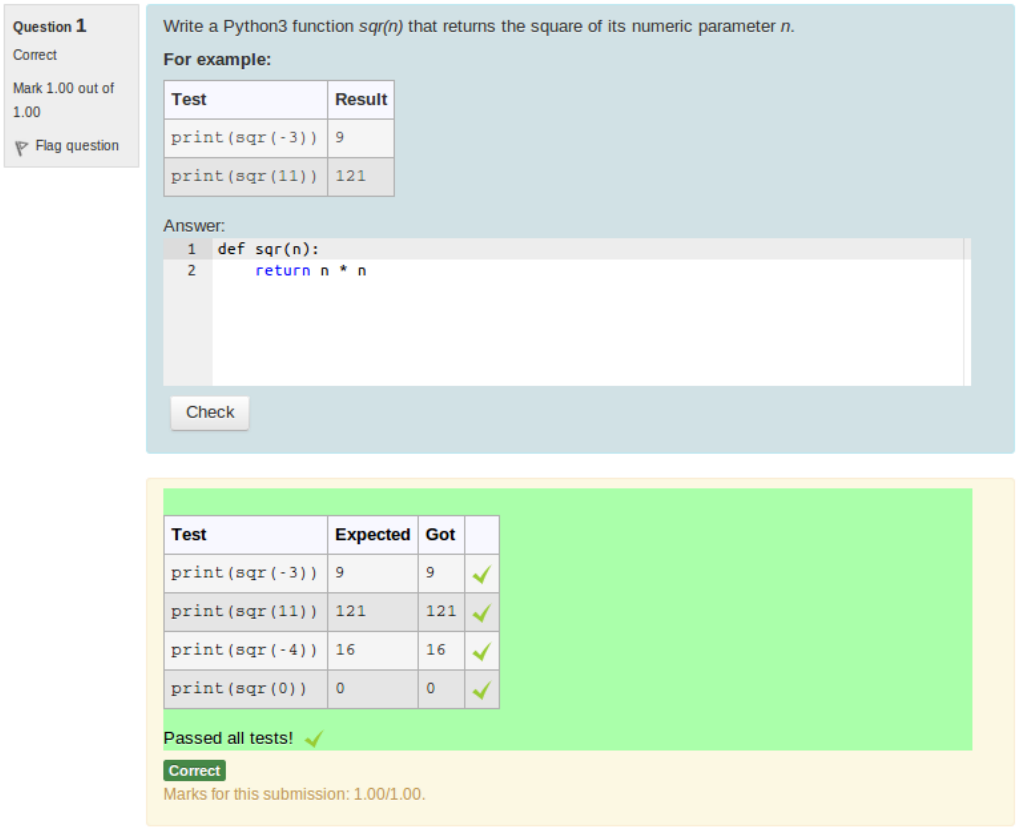

Figura 5.1: Ejemplo de una ejecución de una pregunta con el plugin de Moodle CodeRunner[1].

Dentro de su página web[10] podemos encontrar una gran cantidad de ejemplos que demuestran el funcionamiento del sistema, con distintos lenguajes de programación y distintos tipos de preguntas. Es muy adaptable a lo que requiera el profesor para la prueba y cuenta con una amplia documentación para su uso en su repositorio.

Las diferencias principales entre CodeRunner y este proyecto se basan en lo que se considera un programa. Para CodeRunner, por el propósito que intenta cumplir, un programa es un único archivo que resuelve un problema concreto, es normalmente de tamaño reducido, llegando a ser incluso una única función. En este proyecto se considera un programa como un paquete con uno o múltiples archivos que, pese a que el propósito sea el mismo, sus funciones e implementación pueden ser totalmente independientes, así como la estructura interna del programa.

[1] Imagen del sitio web de CodeRunner. [10]

Parte III

Desarrollo del Sistema

Capítulo 6

Análisis

Una vez está claro la base del problema y por ende los objetivos y tareas que componen el proyecto, se prosigue con la etapa de análisis del problema donde se intentarán establecer los requisitos que debe cumplir la aplicación que ejecuta los programas.

Estos requisitos se dividirán entre los requisitos no funcionales y los funcionales. Dentro de los funcionales se hará otra división para una mayor estructuración entre requisitos de un usuario normal, que puede acceder a las mismas funciones que el resto de usuarios del sistema, y requisitos para un usuario administrador del sistema. Los requisitos del usuario normal se aplican también a los de un usuario con permisos de administrador.

6.1. Requisitos

A continuación se mostrarán en forma de lista los distintos requisitos, que deberá cumplir la aplicación, divididos según su tipo.

6.1.1. Requisitos Funcionales del usuario

RF-1	Registrar una cuenta
[Dependencias]	Ninguna
Característica	El sistema deberá permitir al usuario registrar una cuenta propia.
Descripción	El sistema proporcionará un formulario donde el usuario podrá insertar un nombre de usuario único, un correo y una contraseña asociados a la cuenta.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Media
[Precondición]	El nombre de usuario no está ya registrado, el correo es válido y la contraseña deberá asegurarse con una doble entrada.
[Postcondición]	El usuario es guardado en la base de datos, se inicia sesión en la cuenta y se entra a la página principal
Comentarios	

Cuadro 6.1: RF-1: Registrar una cuenta

RF-2	Iniciar sesión en una cuenta
[Dependencias]	RF-1 (6.1)
Característica	El sistema deberá permitir al usuario iniciar sesión en una cuenta.
Descripción	El sistema proporcionará un formulario donde el usuario podrá insertar el nombre de usuario y contraseña de una cuenta existente.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Media
[Precondición]	La cuenta debe haberse creado en el sistema, ya sea a través de un registro o por inserción directa en la base de datos.
[Postcondición]	Se deberá iniciar sesión si los datos son correctos y llevar al usuario a la página principal o a la referida por el navegador.
Comentarios	Si el usuario no tiene la sesión iniciada cualquier intento de conexión con la página web debería llevar a la página de inicio de sesión.

Cuadro 6.2: RF-1: Iniciar sesión en una cuenta

RF-3	Salir de la sesión
[Dependencias]	RF-2 (6.2)
Característica	El sistema deberá permitir al usuario salir de la sesión de una cuenta.
Descripción	El sistema proporcionará un método sencillo para salir de la sesión en cualquier parte de la página web.
Interfaz de servicio	Sí
[Importancia]	Media
[Prioridad]	Media
[Precondición]	El usuario deberá haber iniciado sesión previamente, o mantener la sesión iniciada de una conexión anterior.
[Postcondición]	Se cerrará la sesión y se mostrará al usuario el formulario de inicio de sesión.
Comentarios	

Cuadro 6.3: RF-3: Salir de la sesión

RF-4	Mostrar la lista de las ejecuciones del usuario
[Dependencias]	RF-2 (6.2)
Característica	El sistema deberá poder mostrar al usuario la información de las ejecuciones asociadas a su cuenta.
Descripción	El sistema proporcionará una interfaz que contendrá el estado e información relevante a sus distintas ejecuciones.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Alta
[Precondición]	El usuario debe haber iniciado sesión y debe tener ejecuciones asociadas a su cuenta.
[Postcondición]	No se mostrará información de otras ejecuciones del sistema que no estén asociadas al usuario.
Comentarios	

Cuadro 6.4: RF-4: Mostrar la lista de las ejecuciones del usuario

RF-5	Mostrar la lista de las configuraciones de ejecución permitidas.
[Dependencias]	RF-2 (6.2)
Característica	El sistema deberá poder mostrar al usuario la información de las configuraciones asociadas al usuario.
Descripción	El sistema proporcionará una interfaz que contendrá la información relevante de las distintas configuraciones de ejecución que estén asociadas al usuario.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Media
[Precondición]	El usuario debe haber iniciado sesión y debe tener configuraciones asociadas a su cuenta.
[Postcondición]	No se mostrará información de otras configuraciones que no estén asociadas al usuario.
Comentarios	

Cuadro 6.5: RF-5: Mostrar la lista de las configuraciones de ejecución permitidas.

RF-6	Mostrar la lista de los scripts de ejecución.
[Dependencias]	RF-2 (6.2)
Característica	El sistema deberá poder mostrar al usuario la información de las configuraciones asociadas al usuario.
Descripción	El sistema proporcionará una interfaz que contendrá la información relevante de los distintos scripts de ejecución que existan en el sistema.
Interfaz de servicio	Sí
[Importancia]	Baja
[Prioridad]	Baja
[Precondición]	El usuario debe haber iniciado sesión, debe haber scripts de ejecución en el sistema.
[Postcondición]	No se mostrará el código del script al usuario por motivos de seguridad.
Comentarios	

Cuadro 6.6: RF-6: Mostrar la lista de los scripts de ejecución.

RF-7	Añadir una ejecución de un paquete de programa a la cola de ejecución.
[Dependencias]	RF-2 (6.2), RF-4 (6.4)
Característica	El sistema deberá permitir al usuario añadir a la cola de ejecución un paquete de programa.
Descripción	El sistema proporcionará un formulario en el cual se permitirá elegir un paquete de programa (archivo comprimido), un nombre de ejecución, una configuración y el script de ejecución asociado.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Alta
[Precondición]	El usuario debe haber iniciado sesión, debe haber una configuración asociada al usuario y un script de ejecución asociada a la configuración.
[Postcondición]	Se añadirán los datos a la cola, añadirá la ejecución a la cola y se llevará al usuario al listado de ejecuciones.
Comentarios	La ejecución recién añadida deberá ser claramente visible en el listado.

Cuadro 6.7: RF-7: Añadir una ejecución de un paquete de programa a la cola de ejecución.

RF-8	Mostrar datos de la cuenta del usuario
[Dependencias]	RF-2 (6.2)
Característica	El sistema deberá poder mostrar al usuario los datos relevantes de su cuenta.
Descripción	El sistema proporcionará una interfaz que mostrará la información relevante del usuario.
Interfaz de servicio	Sí
[Importancia]	Media
[Prioridad]	Media
[Precondición]	El usuario deberá haber iniciado sesión.
[Postcondición]	No se mostrará información de otros usuarios.
Comentarios	

Cuadro 6.8: RF-8: Mostrar datos de la cuenta del usuario

RF-9	Modificar datos de la cuenta del usuario
[Dependencias]	RF-2 (6.2), RF-8 (6.8)
Característica	El sistema deberá permitir modificar al usuario los datos de su cuenta.
Descripción	El sistema proporcionará unos formularios que permitirán al usuario modificar el correo y la contraseña de la cuenta.
Interfaz de servicio	Sí
[Importancia]	Media
[Prioridad]	Media
[Precondición]	El usuario deberá haber iniciado sesión, los datos introducidos deben ser correctos.
[Postcondición]	Se proporcionará un error si los datos introducidos no están permitidos, por ejemplo un correo no válido.
Comentarios	

Cuadro 6.9: RF-9: Modificar datos de la cuenta del usuario

RF-10	Validar un paquete de programa según la configuración de ejecución
[Dependencias]	Ninguna
Característica	El sistema deberá ser capaz de validar un paquete de programa según la configuración elegida para la ejecución.
Descripción	El sistema tendrá que tener una función en el script de ejecución que asegure que el paquete de programa sea válido para su ejecución.
Interfaz de servicio	No
[Importancia]	Alta
[Prioridad]	Alta
[Precondición]	Se debe haber añadido una ejecución con un paquete de programa a la cola.
[Postcondición]	Tras el proceso de validación se continúa con la ejecución del programa, la función de validación debe dar una salida con el resultado de la validación.
Comentarios	

Cuadro 6.10: RF-10: Validar un paquete de programa.

RF-11	Compilar un paquete de programa.
[Dependencias]	RF-10 (6.10)
Característica	El sistema deberá ser capaz de compilar un paquete de programa según la configuración elegida para la ejecución.
Descripción	El sistema tendrá que contar con una función del script de ejecución que compile el paquete de programa si es necesario para su ejecución.
Interfaz de servicio	No
[Importancia]	Alta
[Prioridad]	Alta
[Precondición]	Se debe haber añadido una ejecución con un paquete de programa a la cola.
[Postcondición]	Se sigue con el proceso de test del paquete de programa, la función de compilación debe dar una salida con el resultado de la compilación.
Comentarios	Esta función podría intentar o no intentar ejecutarse dependiendo de la validación, pero en cualquier caso debe dar una salida con el resultado de la compilación.

Cuadro 6.11: RF-11: Compilar un paquete de programa.

RF-12	Ejecutar un paquete de programa.
[Dependencias]	RF-10 (6.10), RF-11 (6.11)
Característica	El sistema deberá ser capaz de ejecutar un paquete de programa con un par entrada-salida preestablecido.
Descripción	El sistema tendrá que contar con una función del script de ejecución que ejecute el paquete de programa siguiendo los pares entrada-salida de los tests establecidos en la configuración de la ejecución.
Interfaz de servicio	No
[Importancia]	Alta
[Prioridad]	Alta
[Precondición]	Se debe haber añadido una ejecución con un paquete de programa a la cola.
[Postcondición]	Se devuelven los resultados de la ejecución completa (validación, compilación y ejecución) al sistema y se guardan los resultados.
Comentarios	Esta función podría intentar o no ejecutarse dependiendo de la validación y la compilación del paquete de programa, pero en cualquier caso debe dar una salida con el resultado de la/s ejecución/es.

Cuadro 6.12: RF12-: Ejecutar un paquete de programa.

RF-13	Analizar el código del paquete de programa
[Dependencias]	Ninguna
Característica	El sistema deberá ser capaz de realizar un análisis semántico del código de un paquete de programa de una ejecución.
Descripción	El sistema tendrá que contar con un analizador semántico que pueda proporcionar un análisis básico relevante del código subido en el paquete de programa.
Interfaz de servicio	No
[Importancia]	Alta
[Prioridad]	Media
[Precondición]	Se debe haber añadido una ejecución con un paquete de programa a la cola.
[Postcondición]	Se recogerán los datos y se añadirán a la salida de la ejecución.
Comentarios	

Cuadro 6.13: RF-13: Analizar el código del paquete de programa

6.1.2. Requisitos Funcionales del administrador

RF-14	Mostrar al administrador el listado de usuarios
[Dependencias]	Ninguna
Característica	El sistema deberá poder mostrar al administrador el listado de todos los usuarios.
Descripción	El sistema deberá proporcionar una vista con la información de todos los usuarios del sistema.
Interfaz de servicio	Sí
[Importancia]	Media
[Prioridad]	Media
[Precondición]	Debe haber usuarios en el sistema.
[Postcondición]	No se deberá mostrar la información privada del usuario.
Comentarios	

Cuadro 6.14: RF-14: Mostrar al administrador el listado de usuarios.

RF-15	Mostrar al administrador el listado de ejecuciones
[Dependencias]	Ninguna
Característica	El sistema deberá poder mostrar al administrador el listado de todas las ejecuciones del sistema.
Descripción	El sistema deberá proporcionar una vista con la información de todas las ejecuciones que haya en el sistema.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Media
[Precondición]	Debe haber ejecuciones en el sistema.
[Postcondición]	No se deberá mostrar información privada del usuario asociado.
Comentarios	

Cuadro 6.15: RF-15: Mostrar al administrador el listado de ejecuciones

RF-16	Mostrar al administrador el listado de configuraciones
[Dependencias]	Ninguna
Característica	El sistema deberá poder mostrar al administrador el listado de todas las configuraciones del sistema
Descripción	El sistema deberá proporcionar una vista con la información de todas las ejecuciones que haya en el sistema.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Media
[Precondición]	Debe haber configuraciones en el sistema.
[Postcondición]	No se deberá mostrar información privada de los usuarios asociados.
Comentarios	

Cuadro 6.16: RF-16: Mostrar al administrador el listado de configuraciones

RF-17	Definir un script de ejecución.
[Dependencias]	Ninguna
Característica	El sistema deberá permitir al administrador definir un script de ejecución
Descripción	El sistema proporcionará un formulario donde se podrá definir el archivo del script de ejecución, así como una descripción de su funcionamiento.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Alta
[Precondición]	El script debe ser verificado por el administrador para que funcione correctamente.
[Postcondición]	El script se guardará en el sistema de archivos preparado para las ejecuciones.
Comentarios	El script tiene que tener las tres funciones de validación, compilación y ejecución que deben proporcionar la salida necesaria.

Cuadro 6.17: RF-17: Definir un script de ejecución

RF-18	Definir una configuración de ejecución
[Dependencias]	RF-17 (6.17)
Característica	El sistema deberá permitir al administrador definir una configuración.
Descripción	El sistema proporcionará un formulario donde se podrá definir una configuración con un nombre único, una descripción, scripts asociados, ruta a la raíz del paquete, ruta al archivo principal y tests de la configuración.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Alta
[Precondición]	El nombre de la configuración debe ser único, debe tener una descripción y debe haber un archivo principal del paquete.
[Postcondición]	Se añadirá la configuración al sistema y se realizará la estructura necesaria de ficheros para la ejecución y tests de la configuración.
Comentarios	

Cuadro 6.18: RF-18: Definir la configuración de ejecución

RF-19	Definir los tests de la configuración
[Dependencias]	RF-18 (6.18)
Característica	El sistema deberá permitir definir los tests de ejecución de la configuración.
Descripción	El sistema proporcionará un formulario donde se podrá crear los tests según la necesidad de configuración, así como introducir los tests como un paquete ya formado.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Alta
[Precondición]	Deben crearse mientras se crea una nueva ejecución, o mientras es modificada.
[Postcondición]	Se guardarán la estructura de los tests en la configuración y se guardará el paquete de los tests en el sistema de archivos para la ejecución.
Comentarios	

Cuadro 6.19: RF-19: Definir los tests de la configuración

RF-20	Asociar configuraciones a los usuarios
[Dependencias]	RF-18 (6.18)
Característica	El sistema deberá permitir asociar configuraciones de ejecución a los usuarios del sistema.
Descripción	El sistema proporcionará de un formulario al administrador para asociar las distintas configuraciones existentes en el sistema a los usuarios que tengan que hacer uso de ellas.
Interfaz de servicio	Sí
[Importancia]	Alta
[Prioridad]	Alta
[Precondición]	Deben existir tanto usuarios como configuraciones en el sistema.
[Postcondición]	Se guardarán los pares configuración-usuario al sistema.
Comentarios	

Cuadro 6.20: RF-20: Asociar configuraciones a los usuarios

RF-21	Asociar scripts a configuraciones
[Dependencias]	RF-17 (6.17)
Característica	El sistema deberá permitir asociar scripts de ejecución a las configuraciones.
Descripción	El sistema proporcionará de un formulario al administrador para asociar las distintas configuraciones existentes en el sistema a los scripts de ejecución.
Interfaz de servicio	Sí
[Importancia]	Baja
[Prioridad]	Media
[Precondición]	Deben existir tanto configuraciones como scripts en el sistema.
[Postcondición]	Se guardarán los pares configuración-script en el sistema
Comentarios	

Cuadro 6.21: RF-21: Asociar scripts a configuraciones

RF-22	Modificar la configuración de ejecución
[Dependencias]	Ninguna
Característica	El sistema deberá permitir modificar la información de las configuraciones.
Descripción	El sistema proporcionará unos formularios que permitan modificar la descripción, rutas y tests de una configuración
Interfaz de servicio	Sí
[Importancia]	Baja
[Prioridad]	Baja
[Precondición]	Deben existir configuraciones en el sistema
[Postcondición]	Se validarán y guardarán los cambios de la configuración en el sistema
Comentarios	

Cuadro 6.22: RF-22: Modificar la configuración de ejecución

RF-23	Modificar el script de ejecución
[Dependencias]	Ninguna
Característica	El sistema deberá permitir modificar el script de ejecución y su información.
Descripción	El sistema proporcionará un formulario para la modificación del código y datos del script de ejecución.
Interfaz de servicio	Sí
[Importancia]	Baja
[Prioridad]	Media
[Precondición]	Debe existir el script de ejecución en el sistema.
[Postcondición]	Se guardarán los cambios del script en el sistema.
Comentarios	El formulario permitirá cambiar el script cambiando el archivo o modificando el código.

Cuadro 6.23: RF-23: Modificar el script de ejecución

6.1.3. Requisitos no funcionales

RNF-1	Ejecución independiente
Característica	La ejecución de un paquete de programa ha de ser independiente de la ejecución del servidor web que aloja el servicio.
Descripción	El servidor web debe seguir su funcionamiento natural sin esperas debido a la ejecución de los paquetes de programa.
[Importancia]	Alta
[Prioridad]	Alta
Comentarios	

Cuadro 6.24: RNF-1: Ejecución independiente

RNF-2	Ejecución segura de paquetes de programa
Característica	La ejecución de un paquete no debe poner en riesgo la integridad de la máquina que contiene el sistema.
Descripción	Se debe asegurar que el sistema no pueda ser puesto en riesgo por culpa de un código erróneo o malintencionado.
[Importancia]	Alta
[Prioridad]	Alta
Comentarios	Se puede encapsular la ejecución del código en un contenedor para evitar que la ejecución afecte al sistema.

Cuadro 6.25: RNF-2: Ejecución segura de paquetes de programa

RNF-3	Encapsulación de la ejecución eficiente
Característica	El método de encapsulación no puede conllevar de una gran carga al sistema.
Descripción	Se debe asegurar que el método de encapsulación no afecte al rendimiento de la máquina, haciendo que sea rápido y efectivo.
[Importancia]	Alta
[Prioridad]	Alta
Comentarios	Es importante reutilizar la misma imagen del contenedor que ejecute el código y no crearla cada vez que se ejecute.

Cuadro 6.26: RNF-3: Encapsulación de la ejecución eficiente

Capítulo 7

Diseño

A continuación se presenta el diseño de la aplicación. Se mostrará el modelo de datos utilizado, los casos de uso tanto del usuario como del administrador del sistema y la estructura del proyecto.

7.1. Modelo de datos

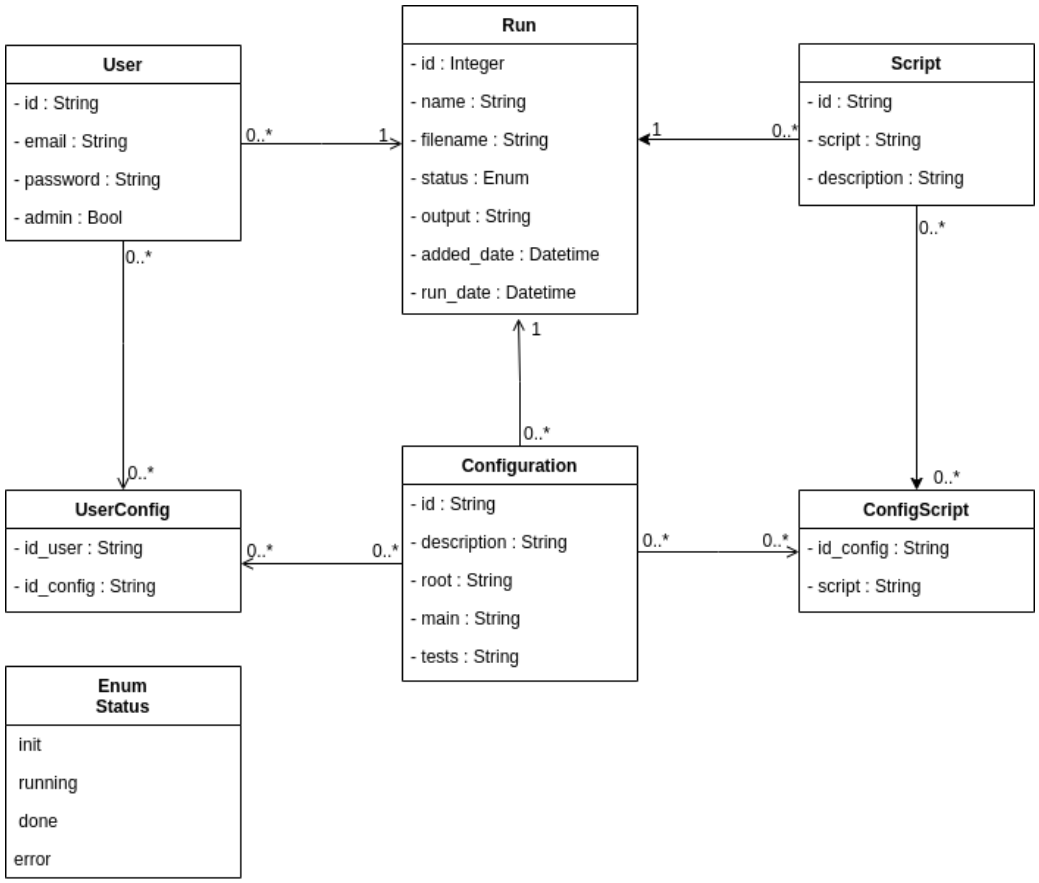

Figura 7.1: Modelo de datos del sistema de ejecución de programas.

En el modelo de datos se presentan los distintos componentes que dan la estructura al proyecto.

1. Usuario: Contiene la información personal del usuario, así como información necesaria para la gestión del servidor.

    a) id: Nombre del usuario, único para cada uno.

    b) email: Correo asociado al usuario.

    c) password: Contraseña de acceso de la cuenta.

    d) admin: Identifica si el usuario tiene poderes de administrador o no.

2. Configuración: guarda la información necesaria para la ejecución de un paquete de programa, así como la descripción del tipo de programa.

    a) id: Nombre de la configuración, único para cada uno.

    b) description: Describe como debe ser y que debe cumplir el programa que se ejecute mediante esa configuración.

    c) root: Establece la ruta de la carpeta raíz del programa dentro del paquete. Es la raíz del paquete por defecto.

    d) main: Establece la ruta del fichero de inicio del programa.

    e) tests: Contiene la estructura de los tests en formato JSON como la figura 7.3

3. Ejecución: tiene el estado, datos y resultado de una ejecución de un paquete de programa.

    a) id: Identificador de la ejecución, numérico único para cada una.

    b) name: Nombre identificativo de la ejecución.

    c) filename: Nombre del paquete subido para la ejecución.

    d) status: Estado de la ejecución, puede ser inicial, en cola, en ejecución o error de ejecución.

    e) output: La salida de la ejecución con el formato de la figura 7.2

    f) added_date: Fecha y hora de la entrada a la cola de la ejecución.

    g) run_date: Fecha y hora del inicio de la ejecución.

4. Script: guarda la información de identificación y descripción de un script para las ejecuciones.

    a) id: Nombre del script de ejecución, único para cada uno.

    b) language: Lenguaje que cubre el script.

    c) description: Descripción del funcionamiento del script.

```
{
    "test": {
        "number": 2,    //Number of tests
        "passed": 2     //Number of passed tests
    },
    "tests":
    [
        {
            "run_output": "",       //Java run command output error
            "run_result": 0,        //Java run command output code
            "test_output": "Output is identical",   //Output files comparison
            "test_result": 0        //Output comparison command code
        },
        {
            "run_output": "",
            "run_result": 0,
            "test_output": "Output is identical",
            "test_result": 0
        }
    ],
    "validation": {
        "file": {
            "output": "",       //File check command output error
            "result": 0         //File check command output code
        },
        "main": {
            "output": "",       //Main file check command output error
            "result": 0         //Main file check command output code
        },
        "unzip": {
            "output": "",       //Unzip file command output error
            "result": 0         //Unzip file command output code
        }
    },
    "compilation": {
        "output": "",       //Compilation program command output error
        "result": 0         //Compilation program command output code
    }
}
```

Figura 7.2: Ejemplo de una salida de una ejecución.

```
[
    {
        "inputs": [                     //Inputs del test
            {
                "text": "numbers_100",  //Nombre del fichero
                "type": "File"          //Tipo de input fichero
            },
            {
                "text": "10",           //Texto del input
                "type": "Text"          //Tipo de input de texto
            }
        ],
        "output": "expected_100"        //Nombre del fichero de salida
    },
    {
        "inputs": [
            {
                "text": "numbers_1000",
                "type": "File"
            },
            {
                "text": "10",
                "type": "Text"
            }
        ],
        "output": "expected_1000"
    }
]
```

Figura 7.3: Ejemplo de la estructura de los tests de una configuración.

7.2. Casos de uso

A continuación se presentan los diagramas de casos de uso divididos por los actores usuario y administrador. Los casos de uso del usuario se incluyen en los del administrador.

7.2.1. Casos de uso del usuario

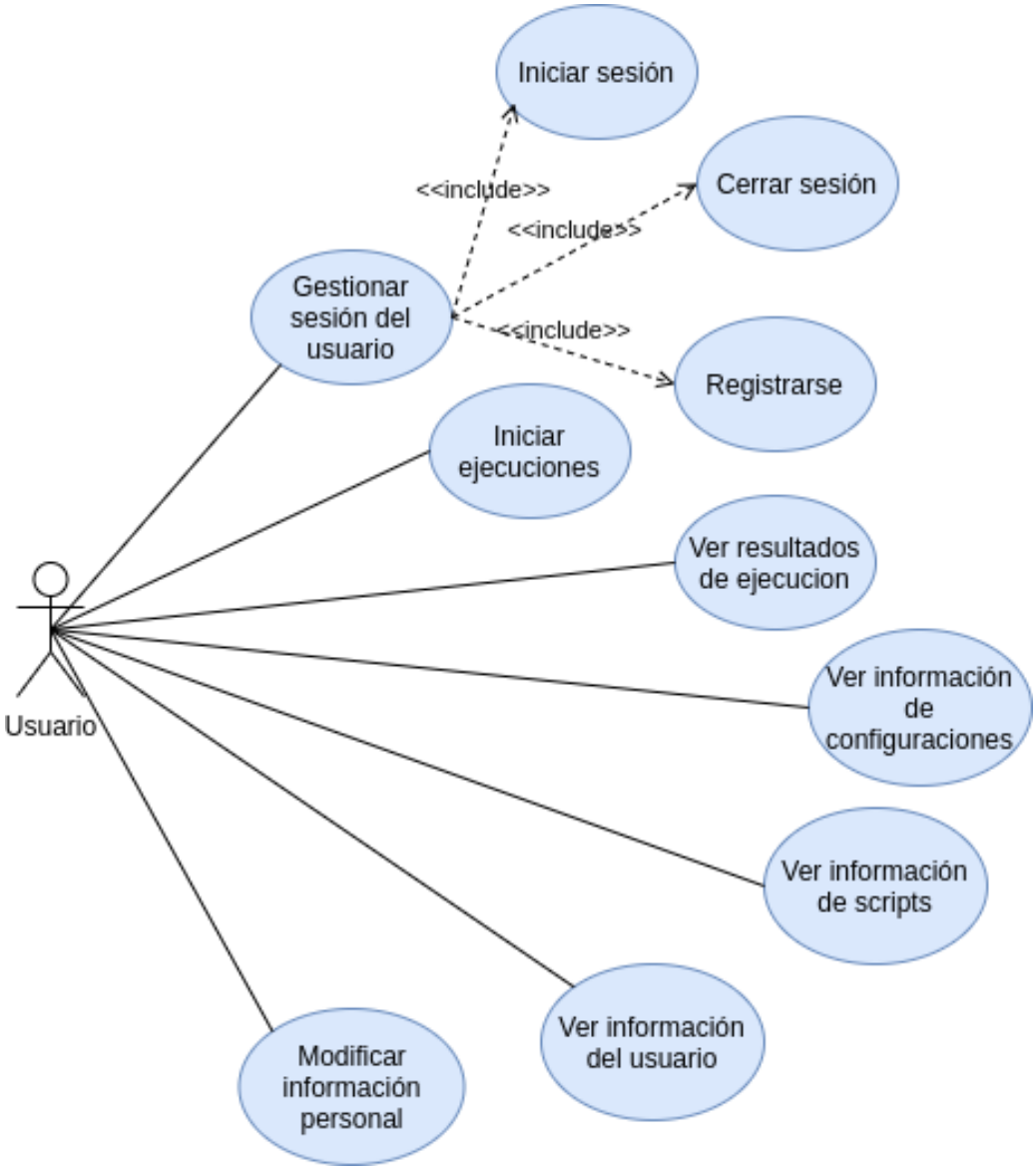

Figura 7.4: Diagrama de casos de uso del usuario.

7.2.2. Casos de uso del administrador

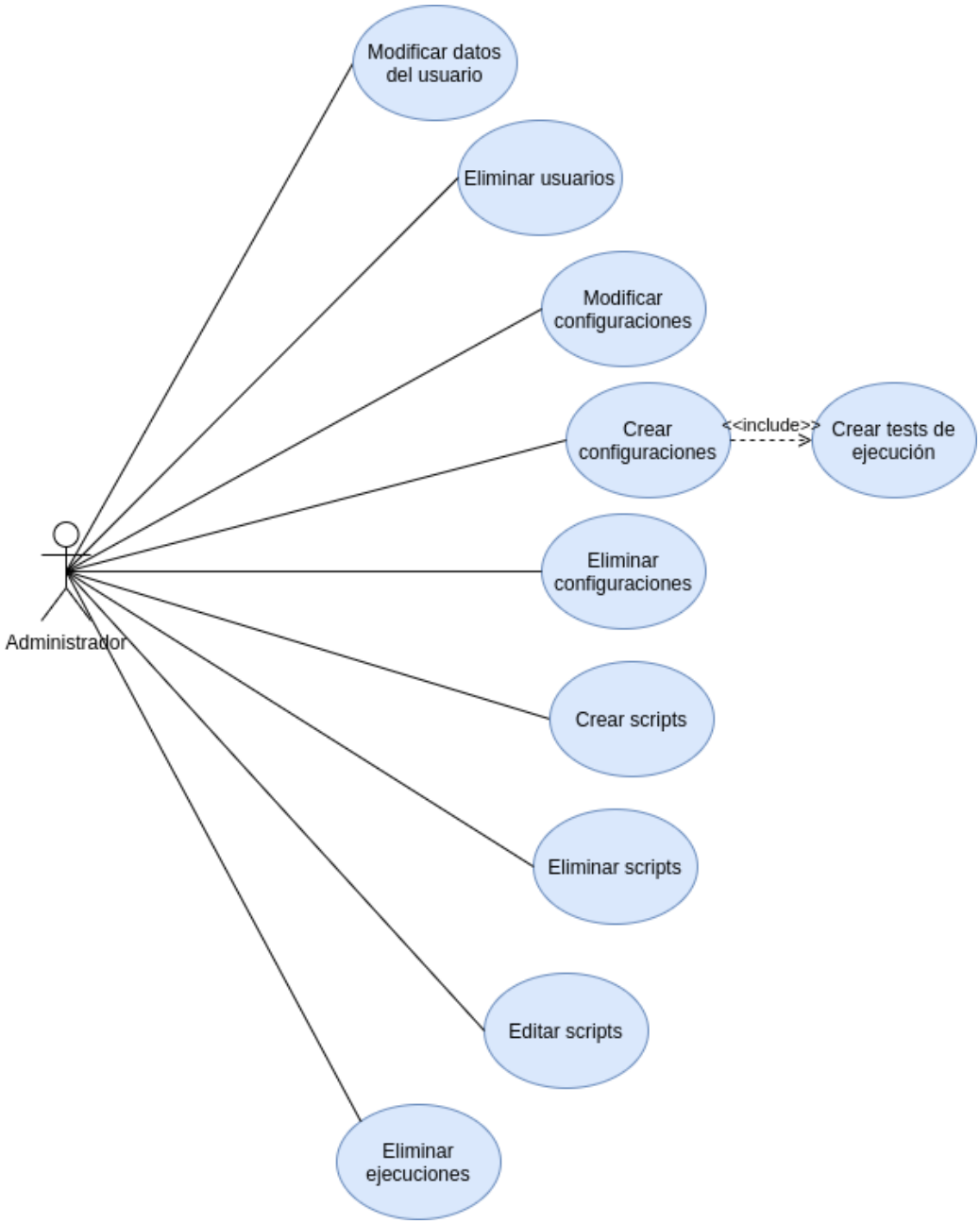

Figura 7.5: Diagrama de casos de uso del administrador.

7.3. Diagrama de secuencia

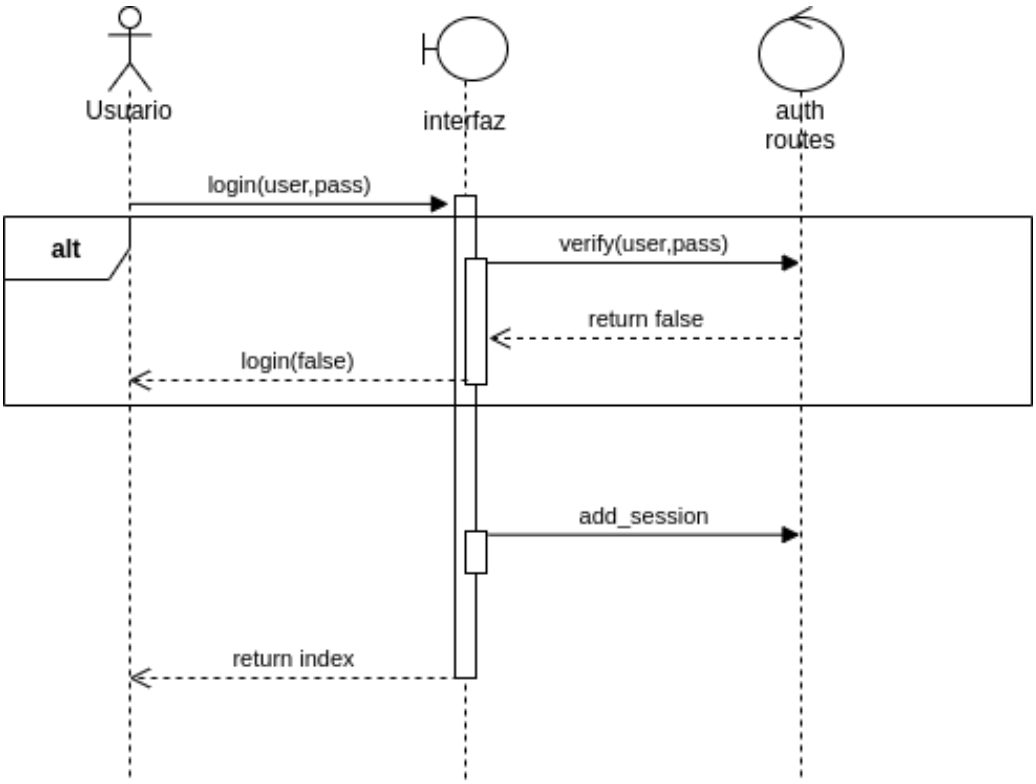

Figura 7.6: Diagrama de secuencia del login a la web.

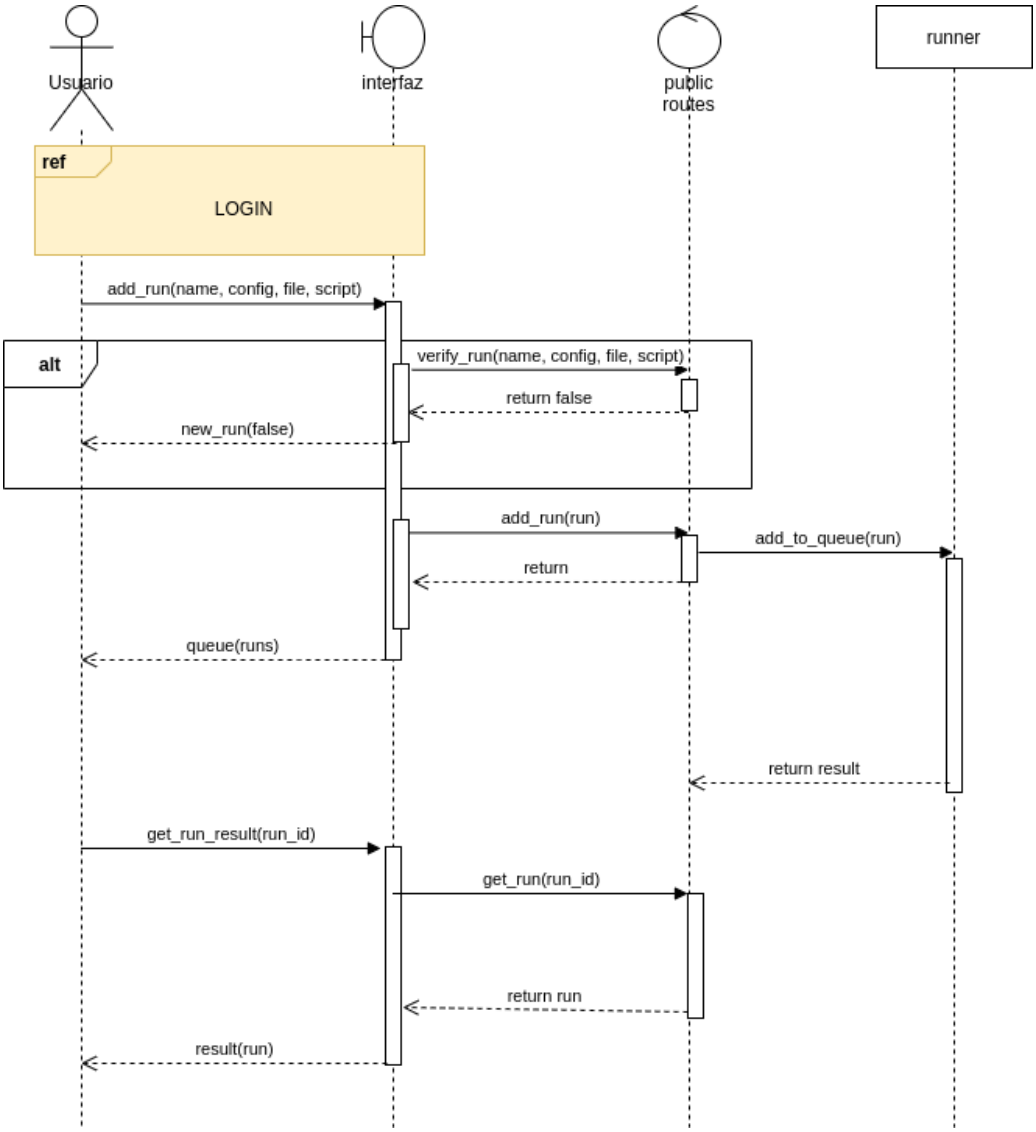

Figura 7.7: Diagrama de secuencia de ejecución de un paquete de programa.

Capítulo 8

Implementación

En este capítulo presentaremos las diferentes herramientas que hemos empleado para la realización de este proyecto: tanto para la gestión del mismo, elaboración de documentos, editores y entornos de desarrollo de programas, bibliotecas y paquetes para servidor y cliente.

El manual de administración e instalación se pueden consultar en el apéndice A. El manual de usuario se encuentra en el apéndice B y el manual de desarrollo para poder orientar futuros cambios está en el apéndice C.

8.1. Herramientas

A continuación se va a hacer un listado de las herramientas utilizadas y su fin en el proyecto:

- Lenguajes
  - Python: Se utilizará para toda la parte del servidor web, elegido por la simplicidad de aprendizaje y uso de las múltiples librerías necesarias y su fácil comunicación con el resto de componentes.
  - Bash: Utilizado en todo lo referente a la ejecución de comandos dentro del contenedor dedicado a ejecutar los programas.
  - HTML5 y CSS: Construyen el diseño de la interfaz web que implementa el servidor.
  - JavaScript: Está en la interfaz cuando se necesite hacer una llamada al servidor con el fin de obtener información de forma asíncrona (Ajax).
  - SQL: Lenguaje utilizado para la interacción directa con la base de datos.
- Librerías de Python
  - Flask: Framework web minimalista útil para servicios web pequeños. Ofrece un gran conjunto de utilidades para simplificar lo máximo posible el diseño de un servidor web.
  - TinyDB: Librería de gestión de datos en ficheros JSON de Python. Utilizada en las primeras iteraciones del sistema como bases de datos de prueba.
  - MySQL-connector: Conector de la base de datos MySQL para Python, integrado para su uso con ORM de Python.
  - SqlAlchemy: Librería de ORM para Python. Ofrece la posibilidad de usar librerías externas como conectores a bases de datos.
  - WTForms: Librería de formularios web para Python, incluye métodos de abstracción de formularios para la generación y validación directa de estos desde un servidor en Python utilizando herramientas de plantillas.

  - Docker-py: SDK de Docker para Python. Ofrece una comunicación directa entre Python y el cliente de contendores Docker.
  - Redis-queue (rq): Es una librería que, utilizando como base de las comunicaciones un servidor Redis, crea hilos de trabajo de Python.
  - PyTest: Librería de test de código para Python.

- Librerías de Flask

  - flask-sqlalchemy: Librería que integra totalmente el paquete SqlAlchemy con Flask, para obtener la interacción directa del servidor con la base de datos para elementos automáticos, como la sesión web.
  - flask-wtf: Librería que une el paquete de generación de formularios con el servidor para automatizar los movimientos de los datos de las peticiones en los formularios, entre otras cosas.
  - flask-login: Mantiene la lógica de las sesiones web en el servidor Flask.
  - flask_codemirror: Librería que implementa un editor gráfico de texto en HTML que formatea directamente el texto a distintos lenguajes.

- Frontend

  - Bootstrap 5.0: Biblioteca con herramientas de diseño para páginas web, utilizado mayormente como herramienta para dar estructurar la página.
  - jQuery: Librería del lenguaje JavaScript que simplificar el trato con documentos HTML. Utilizado para interacciones en la página así como para llamadas AJAX al servidor.
  - Jinja2: Motor de plantillas web para Python.

- MySQL: Servidor de bases de datos relacionales utilizado para el almacén del modelo y datos del servidor.

- Docker: Proporciona los contenedores que aíslan la ejecución del código del servidor. Además proporciona un entorno controlado de ejecución.

- Redis: Servidor de bases de datos en memoria, necesario para la comunicación entre el trabajador que ejecuta el programa y el servidor.

- PMD: Analizador de código estático. Es extensible y configurable y da soporte a Java, JavaScript, XML, entre otros.

- LaTeX: Lenguaje utilizado para la creación de este documento. Se utiliza la herramienta online Overleaf, que ofrece un IDE completo de código LaTeX sin necesidad de instalar ningún tipo de software.

- PyCharm: Entorno de desarrollo centrado en lenguaje Python de la compañía JetBrains que ofrece una versión comunitaria gratuita y una versión profesional la cual es también gratuita con propósitos educativos a estudiantes.

- GitLab: Servicio de control de versiones utilizado para guardar el código.

8.2. Entorno de desarrollo

Para la elaboración de este proyecto es obligatorio utilizar el sistema operativo Linux, pues la librería encargada de ejecutar los hilos necesita de la operación *fork()*.

El entorno de desarrollo se centraliza en el IDE Pycharm. Este IDE contiene el editor en el cual se da forma al código del servidor, el control de los directorios necesarios para el manejo de los programas y la creación del los contenedores Docker. Este IDE proporciona además utilidades de servicios desde las cuales podemos directamente encender, conectarnos, y controlar distintos servicios externos que utiliza la aplicación, como puede ser Docker, para ver en tiempo real la creación y eliminación de los contenedores y la información que contienen, o Redis, cuya terminal aporta el estado de los hilos de trabajo que ejecutan los programas.

Este IDE ofrece también una conexión directa a múltiples servicios de control de versiones, incluido GitLab. Se mantiene así el código al día en GitLab directamente con los controles que incluye el editor.

8.3. Implementación

La implementación del servicio web, como en casi otros casos se divide en frontend y backend. En este caso, al implementar un servicio muy concreto, se divide la explicación en esos dos, más la implementación del sistema de ejecución de los paquetes de programa.

8.3.1. Servidor

El servidor está implementado utilizando el framework Flask. Este framework gracias a ser liviano es muy útil para páginas web pequeñas o servicios pequeños con muchas llamadas. También gracias a su método de funcionar a través de extensiones, ya sean propias como la extensión para el control de sesiones, o externas como la librería SQLAlchemy que típicamente sirve de ORM para aplicaciones Flask. Esta característica convierte a Flask en un servidor muy modular y sencillo para prototipos, sencillez que ayuda a que la herramienta tenga una dificultad de entrada menor en comparación a otros fameworks web, como Spring Boot[14] de Java o Django[11], también de Python.

Base de datos

Casi siempre, la otra parte importante del software que se elige en conjunto con el servidor que almacena la web es la base de datos. En los primeros momentos del desarrollo, como la prioridad principal no es hacer una página web, sino un ejecutor de programas, se utilizó una librería simple de Python que hace la función de base de datos sin necesidad de tener un servidor web, ni un modelo real de tablas. Esto ha sido importante para poder avanzar al inicio del proyecto, en la etapa de análisis del problema y aprendizaje de las herramientas, cuando no se tienen claro ciertos elementos del diseño. La librería es TinyDB[13], utiliza un sistema de documentos JSON, los cuales guardan la información de un diccionario directamente en un documento, utiliza una interfaz de lectura y escritura tipo base de datos, por lo que se pueden hacer consultas parecidas a las de una base de datos relacional.

Una vez el diseño estaba mucho más desarrollado se pasó a una base de datos relacional. Se ha optado por utilizar MySQL, por su fácil conexión con Flask.[7]. Para realizar el tratamiento de los datos a través del servidor se ha utilizado el ORM SQLAlchemy, modelando los distintos datos del

diseño (7.1) como objetos de Python (8.1), los cual facilita el tránsito entre servidor y base de datos, sin tener que diseñar consultas complejas (ni simples) de SQL.

```python
class User(db.Model, UserMixin):
    __tablename__ = 'user'

    id = db.Column(db.String(32), primary_key=True)

    email = db.Column(db.String(256), nullable=False)

    password = db.Column(db.String(128), nullable=False)

    admin = db.Column(db.Boolean, default=False)


    def set_password(self, password):
        self.password = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password, password)

    def save(self):
        db.session.add(self)
        db.session.commit()

    def remove(self):
        db.session.delete(self)
        db.session.commit()
```

Figura 8.1: Ejemplo de la clase User con el ORM SQLAlchemy.

Estructura del servidor

La estructura ha ido cambiando a lo largo del desarrollo, en un comienzo era una estructura típica de una aplicación simple de Flask con un único archivo para las llamadas al servidor, las cuales contenían lo necesario para la ejecución de un paquete. Había además un archivo de inicio del servidor y otro con los modelos.

Una vez se concreta el diseño y se establece una estructura mucho más común de los servidores web de Flask, haciendo utilidad de los Blueprints (8.2). Los Blueprints en Flask son una forma de dividir los contenidos de una página o un servicio por módulos, haciendo que la aplicación principal los importe y sean relativamente independientes entre ellos. Como siempre ocurre en la práctica esto no es del todo así, lo más común en una página web es que dos funcionalidades totalmente distintas se necesiten entre ellas, como por ejemplo las funciones de autenticación son necesarias en la gran mayoría o todas las vistas que contenga la web, por lo que son dependientes. Aun así el módulo que necesite de esta autenticación, no le importa como sea el proceso, ni como sean las vistas que utiliza, ni de que otras cosas necesite, por lo que sigue siendo una forma muy útil de separar las partes de la web, facilitando las tareas de desarrollo, ya sea para añadir funcionalidades o encontrar errores.

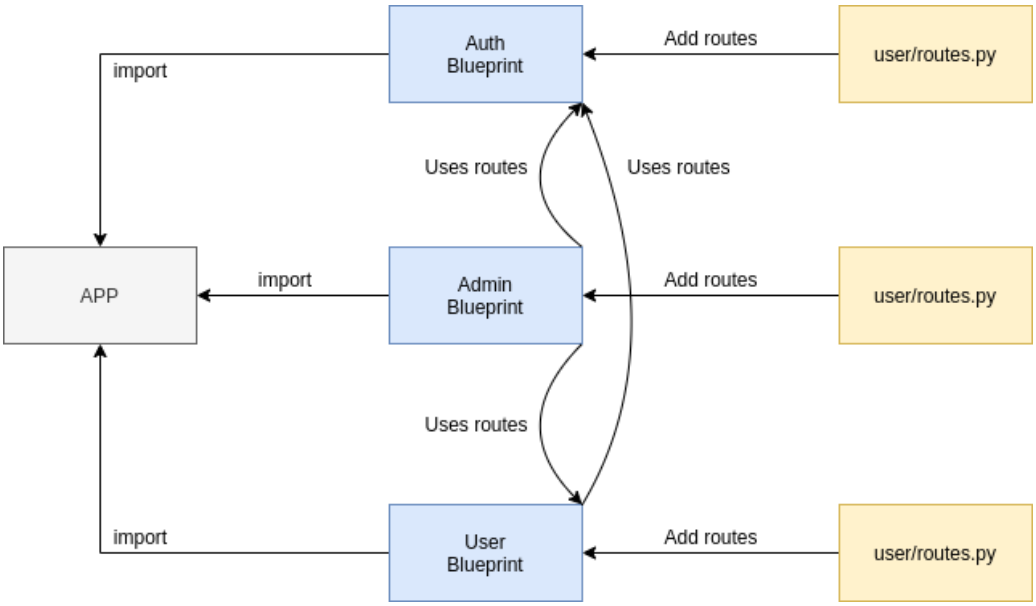

Figura 8.2: Diagrama de funcionamiento de los Blueprints.

Con la idea de utilizar los Blueprints como método de separar las funcionalidades de la web, se elige que la estructura que tenga el código en el servidor sea representando estos módulos, por lo que se agrupan las llamadas, vistas y controladores únicos de cada uno y se agrupan elementos comunes del servidor, como pueden ser los modelos, elementos base de las vistas, métodos de utilidades como el sistema de control de ficheros o incluso la parte del código que se dedica a la ejecución de programas, que no depende de ninguna vista. Con todo eso en mente la estructura del servidor web queda tal y como está representada en la figura 8.3 como una estructura basada en aplicaciones [16].

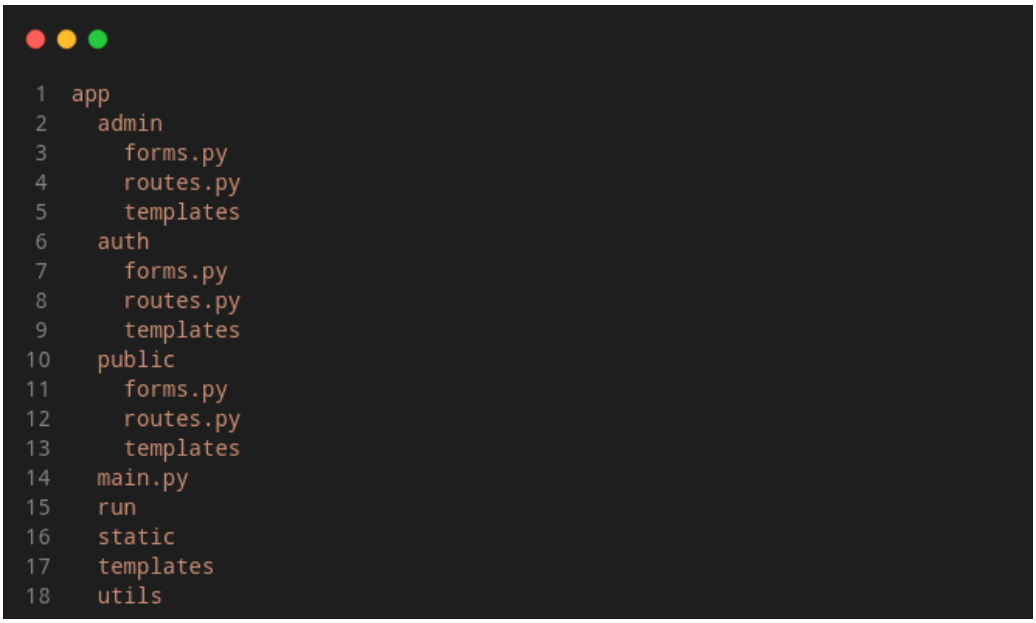

Figura 8.3: Estructura de la aplicación.

8.3.2. Interfaz

La interfaz de la web está elaborada con el motor de plantillas por defecto de un servidor Flask, Jinja2. Este motor tiene conexión directa con el servidor, por lo que puede hacer uso del sistema de sesiones y otras herramientas implementadas dentro de Flask. También permite insertar o expandir otras plantillas, lo que hace muy sencillo reutilizar diseños de formularios y visualizaciones de datos a lo largo de las distintas páginas de la web. Se utiliza jQuery y JavaScript para realizar las llamadas AJAX necesarias dentro de las vistas. El diseño de la web está hecho con la librería Bootstrap (sobre todo para estructurar los contenidos de la página) y con CSS para las modificaciones visuales necesarias.

Formularios

Una de las partes más importantes de la web son los formularios, necesarios para casi cualquier interacción con las distintas funcionalidades. Estos están diseñados con ayuda de la librería de Python WTForms, que genera los campos de entrada correspondientes a una clase Python donde se establece el tipo y las validaciones necesarias para que el campo sea aceptado. Tiene la posibilidad de introducir un formulario como campo de otro, por lo que es una forma muy útil de modular ciertos formularios que se vayan a reutilizar con distintas intenciones (por ejemplo, crear y editar). Esto hace que sea más fácil separar las vistas según el contenido de los formularios e incluso crearlos dependiendo de los permisos de la sesión del usuario.

8.3.3. Ejecución de paquetes

El sistema que se ha diseñado para la ejecución de programas ha buscado cumplir varias características. Lo primero y más importante es que la ejecución sea independiente del servidor web que almacena el servicio, ya que es imprescindible que varios usuarios puedan utilizar el sistema al mismo tiempo, por ello se elige utilizar un sistema de colas en el cual una ejecución, una vez

validada, entra a la cola de la que se alimenta el hilo trabajador que se dedica a la ejecución de programas, independientemente de la configuración o fichero de ejecución que utilicen. Otro punto de vista influyente al diseño de la ejecución es hacer que el servicio pudiera funcionar en otra máquina, de modo que solo hiciera falta compartir la información de la ejecución y el paquete, y el servicio pudiera funcionar de forma totalmente separada al servidor web. Por último se busca que puedan funcionar programas livianos, como otros que den una cierta carga al sistema.

El método de ejecución por tanto se diseña como una cola (FIFO) de ejecuciones las cuales se deben ejecutar de forma independiente al servidor y por medio de un script. El script puede estar implementado de distinta forma según el su propósito, características, lenguajes, pero grosso modo consta de las siguientes etapas:

- Validación: En esta etapa se valida el paquete (como es un archivo comprimido es necesario expandirlo), la estructura según la configuración elegida y que se encuentren los archivos que necesite el script y la configuración para su ejecución.
- Compilación: Esta es una etapa optativa según el lenguaje o lenguajes que utilice el script. En esta etapa se compilan los archivos de código necesarios.
- Ejecución de tests: Tercera y última etapa en la cual se ejecutan los tests establecidos en la configuración de la ejecución.

Sistema de colas

Con todo esto en mente, se buscó un sistema que encajara con el sistema de desarrollo del servicio web, posiblemente implementado en Python y que no diera problemas al acomodarse con el resto del servicio. Se ha optado por utilizar la librería Python-rq[5]. Esta es una librería de colas y trabajadores liviana implementada en Python que está diseñada para que su dificultad de uso no sea una barrera de entrada.

Para hacer funcionar la librería, como su nombre indica, es tener en funcionamiento un servidor Redis. Redis es un servidor de base de datos en memoria, el cual se utiliza, entre otras funciones, como bróker de mensajería. La librería, al iniciar un hilo trabajador, monta dentro del servidor Redis un buzón del cual escucha y utiliza como cola. En la otra parte, en nuestro caso el servidor web, se establece una conexión con ese buzón del servidor Redis, a esa conexión es a donde se mandan funciones, a las cuales debe tener acceso el trabajador, y los parámetros de la función como mensajes.

Con esta estructura de la librería es como se comenzó el desarrollo y se creó la función que utiliza el trabajador para ejecutar las funciones, a la cual como parámetro se le pasaba toda la información de la ejecución. Debido al método con el que funcionaba esta cola rápidamente se encontró el problema de obtener el resultado de la ejecución. Una forma con la que cuenta esta librería para obtener los resultados es asociar un ID con un trabajo de la cola, de modo que puedes obtener el estado y resultado preguntando a la conexión con Redis.

Pese a que es una forma de hacer que funcione sigue dando un problema, pues si no se quiere parar el servidor para preguntar por el estado es necesario que el usuario pida esta información y, como en todo, esta información dentro del servidor Redis tiene un tiempo límite. Para subsanar este problema se cambió la forma de iniciar la ejecución, insertando primero la información de la ejecución en la base de datos, y dando al trabajador una conexión con la misma. De este modo al trabajador solamente se le pasa el ID de la ejecución, y el mismo actualiza el estado y el resultado de la misma según termine. La estructura final del diseño de la cola es la representada en la figura 8.4.

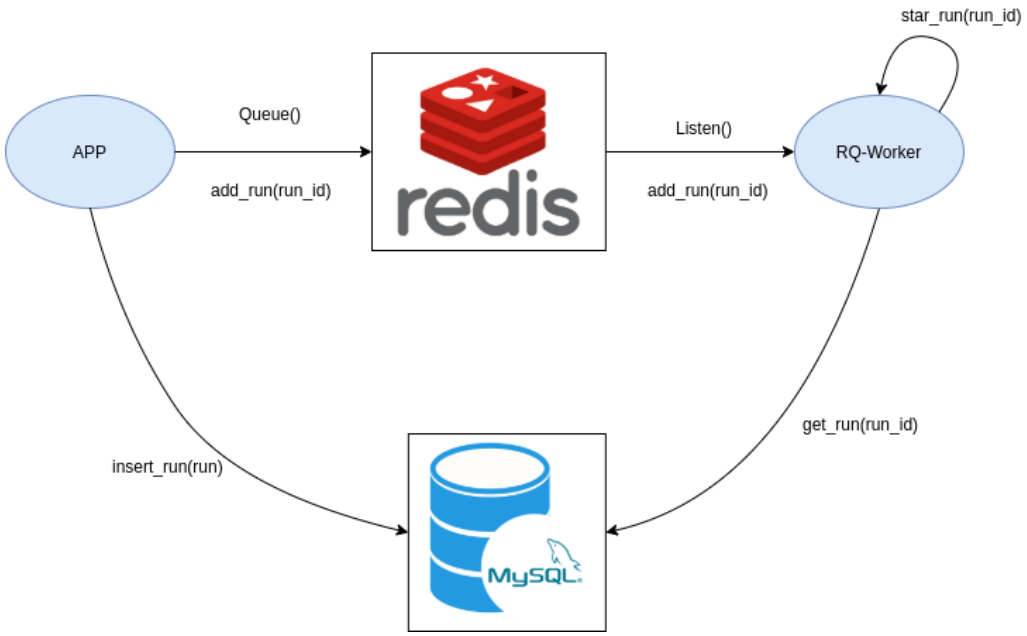

Figura 8.4: Estructura de la cola del ejecutor de programas.

Encapsulación

La otra parte del diseño es hacer que las ejecuciones sean independientes a la máquina que contiene el trabajador, para ello se utiliza la encapsulación de la ejecución del script. Posiblemente el software más utilizado para encapsular casi cualquier software es Docker[4]. Su sistema de imágenes y contenedores hace que sea muy fácil diseñar una imagen que pueda ser reutilizable para cualquier ejecución. Esto realmente es así, pero como siempre que se utiliza una herramienta por primera vez, la falta de experiencia hace que surjan problemas.

En un primer momento en el uso de Docker se diseñó una imagen que generaba un contenedor en el cual directamente desde el archivo de Docker (Dockerfile) se copiaba el paquete de programa, script, y tests necesarios para la ejecución del paquete. Esto hace que se genere una nueva imagen por cada ruta distinta a los archivos, y debido a la estructura con la que se guardan (A), esto hace que sea una imagen por ejecución.

Figura 8.5: Estructura de archivos para Docker.

Este problema con el uso de Docker se arregló en parte al encontrar que Docker cuenta con un SDK para Python, el cual facilita el trato con los contenedores. Así se simplificó el Dockerfile para generar la imagen solo con las librerías necesarias y se crea un contenedor por ejecución al cual se le copian a través de esa utilidad los archivos necesarios para la ejecución. Debido a que esto se empezó a utilizar tarde en el desarrollo no se ha llegado a explorar lo suficiente y se ha establecido una única función de inicio de la ejecución que ejecuta los tres apartados del script (validación, compilación y ejecución de tests), pero es muy posible separar los tres y actualizar el resultado de la ejecución en la base de datos de forma mucho más dinámica y posiblemente útil para ejecuciones pesadas.

8.3.4. Análisis de código

La parte final en la implementación del proyecto ha sido el análisis del código. Debido a las limitaciones de tiempo se buscó una herramienta que diera una solución fácilmente adaptable al diseño actual del proyecto. Se encontró el analizador de código estático PMD[9]. Este da soporte al lenguaje en el que se inicia el proyecto, Java, así como a otros (JavaScript, Scala, XML...). Debido a que no es una herramienta que pueda servir para cualquier lenguaje que se quiera implementar posiblemente habría que observar en un futuro como, o mezclarla con otras herramientas que den soporte a otros lenguajes, o reemplazarla totalmente.

La ejecución de la herramienta es simple: se elige un fichero de configuración (para el proyecto se ha elegido el propuesto como inicial por los autores de PMD) y se ejecuta en la raíz del programa a analizar. La salida se establece en formato JSON y se incluye con el resto de la salida generada en la ejecución, agrupando toda la salida en el mismo campo. Para el proyecto se ejecuta la herramienta desde el worker mandando la orden al contenedor que ejecuta los tests del programa, una vez han finalizado, para que esto funcione se ha modificado la imagen del contenedor instalando el programa PMD y todo lo que requiera para su funcionamiento. Se ha utilizado como base la imagen (no oficial) de Docker[12], adaptando lo necesario para la imagen de ejecución del proyecto.

La herramienta es muy configurable, tiene por defecto una gran cantidad de reglas que se pueden aplicar en el análisis y dan información del uso de estructuras de código, mal uso de llamadas, fallos en la sintaxis... Además de que da la posibilidad al usuario de añadir nuevas reglas, ya sea agrupando varias de las existentes o componiendo nuevas.

Capítulo 9

Pruebas

9.1. Test del servicio de ejecución

Debido a la naturaleza del diseño, las pruebas del funcionamiento del servicio requieren ser hechas por cada script implementado. En esta primera versión solo se cuenta con un script que ejecuta y testea programas escritos en Java. Para comprobar el correcto funcionamiento se ha preparado un conjunto de pruebas con dos configuraciones de ejecución distintas:

- NumSum: Programa que suma los números de un fichero. Hay dos entradas, la primera es la ruta a un fichero de entrada y la segunda la ruta a un fichero de salida. El fichero de entrada está compuesto por números escritos uno por línea. El fichero de salida debe ser un fichero con una única línea con el resultado de la suma del resto de ficheros. La estructura del proyecto debe ser un único archivo llamado NumSum.java.
- product: Programa que multiplica los números de un fichero. La entrada del programa son dos rutas, la primera es la ruta a un fichero de entrada y la segunda a un fichero de salida. El fichero de entrada está compuesto por números, escritos uno por línea. El fichero de salida es un fichero con una única línea con el resultado del producto de los números del fichero de entrada. El fichero main debe estar en la ruta ./src/Producto.java, nombre del fichero incluido.

La primera configuración es por tanto un programa cuyo fichero de inicio se encuentra en la raíz del paquete, se han preparado distintas variantes del mismo programa que dan distintos resultados y se encuentran en la carpeta *dev* del proyecto:

- NumSum.zip: Un programa que cumple con los dos tests con los que cuenta la configuración.
- NumSum20s.zip: Un programa que cumple con los dos tests, y tarda 20 segundos, pese a que actualmente no cubre ningún caso, es útil para comprobar el funcionamiento de la cola.
- NumSum_cifrado.zip: Un programa que cuenta con contraseña en el paquete comprimido, se espera que falle en descomprimir el paquete.
- NumSum_FirstTest.zip: Programa que solo cumple con el primer test, dando el mismo resultado en el segundo.
- NumSumadd1.zip: Programa que falla ambos tests, pues añade uno al resultado final.
- NumSumCompError.zip: Programa que falla en la compilación debido a un error en el código.

Para visualizar los resultados se pone la vista de las de ejecuciones de la página (9.1):

Figura 9.1: Resultados de las pruebas de la configuración NumSum.

La configuración *product* en cambio tiene el fichero de inicio dentro de un directorio. Se han preparado otros paquetes que cubren otros casos de código Java:

- Producto.zip: Programa que cumple con los dos tests con los que cuenta la configuración y está dividido en varios archivos Java.
- Producto_interface.zip: Programa que cumple con los dos tests y utiliza e importa una interfaz.
- Producto_no_java.zip: Un programa que no cuenta con ficheros .java.
- Producto_no_main.zip: Un programa que no tiene el archivo main donde se espera según la configuración.

Para visualizar los resultados se pone la vista de ejecuciones de la página (9.2)

Figura 9.2: Resultados de las pruebas de la configuración product.

Capítulo 10

Conclusiones

En este proyecto se ha buscado de realizar una versión inicial básica de un servicio de ejecución y análisis de programas. El proceso de realizar el servicio ha ayudado a adquirir conocimientos sobre herramientas y métodos de trabajo útiles y de uso muy común en el sector laboral de la informática y la programación, destacando elementos como Docker, el cual es un software con una gran variedad de utilidades, sobre todo en sistemas que necesitan automatización, y con una gran comunidad que no hace más que representar el gran uso que se le da en el sector. Creo que Docker es una herramienta que debería enseñarse a lo largo de los estudios, ya que la facilidad para montar, por ejemplo, un servidor de bases de datos preparado para funcionar con un solo comando y que pueda estar integrado en el IDE es una utilidad muy práctica. También quiero destacar a Flask, el cual cuenta con una gran comunidad y una enorme variedad de utilidades que hacen muy fácil el aprendizaje del funcionamiento de servicios web.

En el proceso de análisis de las herramientas he aprendido que características son las que se deben buscar en una herramienta cuando empiezas a realizar un proyecto, descartando las herramientas que tienen una curva de entrada muy grande, haciendo que para proyectos que se encuentran limitados en el tiempo no sean demasiado prácticas, o herramientas que pueden ser demasiado potentes en algunas características muy concretas, como Spring Boot que está muy centrado en páginas web y tiene una estructura bastante cerrada, propiedades que pueden señalar que no es una herramienta indicada para llevar a cabo un proyecto que requiere de algo de maniobrabilidad en las primeras etapas de análisis.

Posiblemente lo más importante que me llevo del proyecto es entender del todo lo importante de la comunicación con un compañero o un líder, pues debido a la falta de comunicación con el tutor por mi parte muchos problemas que por falta de experiencia yo no he sido capaz de encontrar podrían haberse solucionado mucho más rápido si hubiera tenido una comunicación más continua y por tanto me hubiera podido guiar más.

10.1. Trabajo futuro

Queda pendiente trabajar más en la parte del análisis del código, posiblemente utilizando rankings por configuración que puedan dar una vista en perspectiva sobre la calidad real de un programa. También queda pendiente mejorar el funcionamiento de la ejecución del script, pues se puede hacer mucho más modular, lo que da un mayor dinamismo al servicio.

En el apartado de pruebas falta hacer una prueba de carga, ver cuáles son los límites del diseño, que problemas podría dar un proyecto realmente grande, y analizar la dificultad que pueda conllevar

añadir otros scripts y más elementos a la configuración de ejecución, o incluso poder personalizar más una ejecución sobrescribiendo parámetros de la configuración.

Apéndice A

Administración e Instalación

A.1. Instalación

¡Tiene que ser ejecutado en Linux!

Clonar del repositorio GitLab:

```
https://gitlab.inf.uva.es/andcabe/tfg-andrescabero.git
```

Instalar los paquetes requeridos de Python con pip:

```
pip  install -r requirements.txt
```

Instalar:

- Docker Engine
- Redis server
- MYSQL Database

A.1.1. MYSQL

Sentencias de creación de las tablas, incluida la inserción del administrador por defecto en el archivo **dev/sql_inserts**.

```
Usuario: root
Clave: root
```

A.2. Configuración

La configuración del servidor se establece en el archivo **data/config.yml**

Ejemplo:

```
#FLASK CONFIG
DEBUG: False
MAX_CONTENT_LENGTH: 16000000 #16MB max file size
ALLOWED_EXTENSIONS: ['zip'] #Do not change
```

```
#REDIS
REDIS_URL: 'redis://localhost:6379'

#MYSQL
MYSQL_USER: 'root'
MYSQL_PASS: 'root'
MYSQL_SERVER: 'localhost'
MYSQL_PORT: '3306'
MYSQL_DB: 'programrunner'
```

A.2.1. Ejecución

Una vez iniciada la base de datos y el servidor Redis, iniciar servidor web Flask:

```
python main.py
```

Iniciar Redis-Queue worker:

```
python worker.py
```

Apéndice B

Manual de Usuario

B.1. Usuario

B.1.1. Introduccion

Nada más hacer login en la web encontramos la página con la lista de ejecuciones, tanto esta como el resto páginas siguen el mismo esquema (B.1). A la izquierda de la web está el menú lateral donde se encuentran los enlaces a las distintas vistas. El menú de arriba contiene a la derecha tanto el enlace al perfil como el botón para salir de la sesión. El resto son los contenidos de la página en la que se encuentra el usuario.

Figura B.1: Vista general de la web.

Todos los listados de elementos tienen el mismo formato que los que se ven en la imagen. La forma de interactuar con ellos para obtener más información es haciendo clic en los títulos, mostrando una ventana modal con mayor información y enlaces a la información asociada al objeto de interés (B.2). Existen tres listados disponibles para el usuario, el listado de ejecuciones del usuario, el listado de configuraciones habilitadas para el usuario, y el listado de scripts.

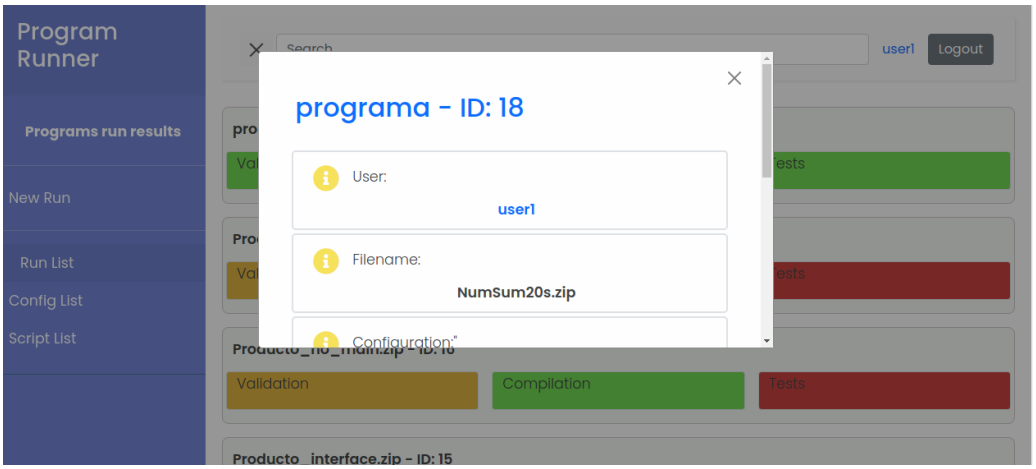

Figura B.2: Ejemplo de ventana modal de la web.

B.1.2. Modificación del perfil

Si entramos en la página de nuestro perfil del usuario podremos modificar nuestra información personal, ya sea el correo electrónico asociado a la cuenta, o la contraseña. Disponemos para ello de dos formularios independientes (B.3).

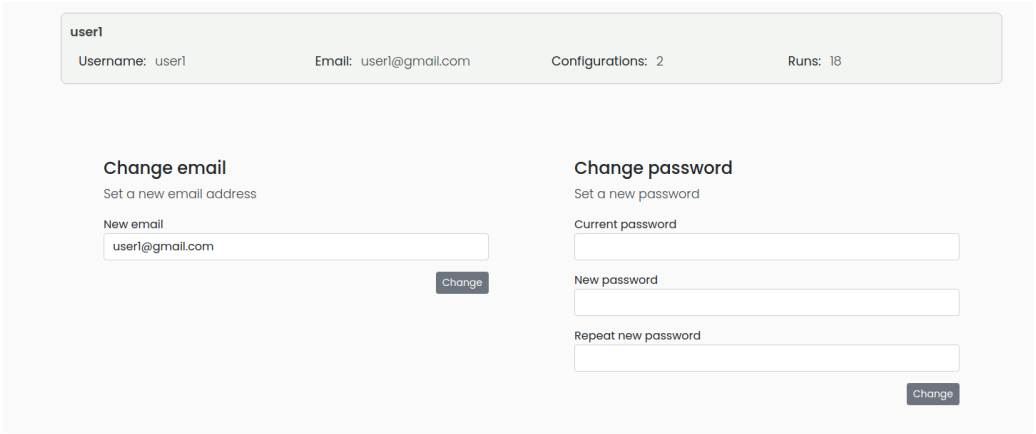

Figura B.3: Formularios del perfil de usuario.

B.1.3. Inicio de una ejecución

Utilizando el menú entramos a la página *New Run* para iniciar una nueva ejecución. Completando el formulario (B.4) que nos encontramos y eligiendo el archivo (empaquetado en formato zip) que contenga el programa, eligiendo la configuración asociada correctamente, se puede incluir en la cola una nueva ejecución.

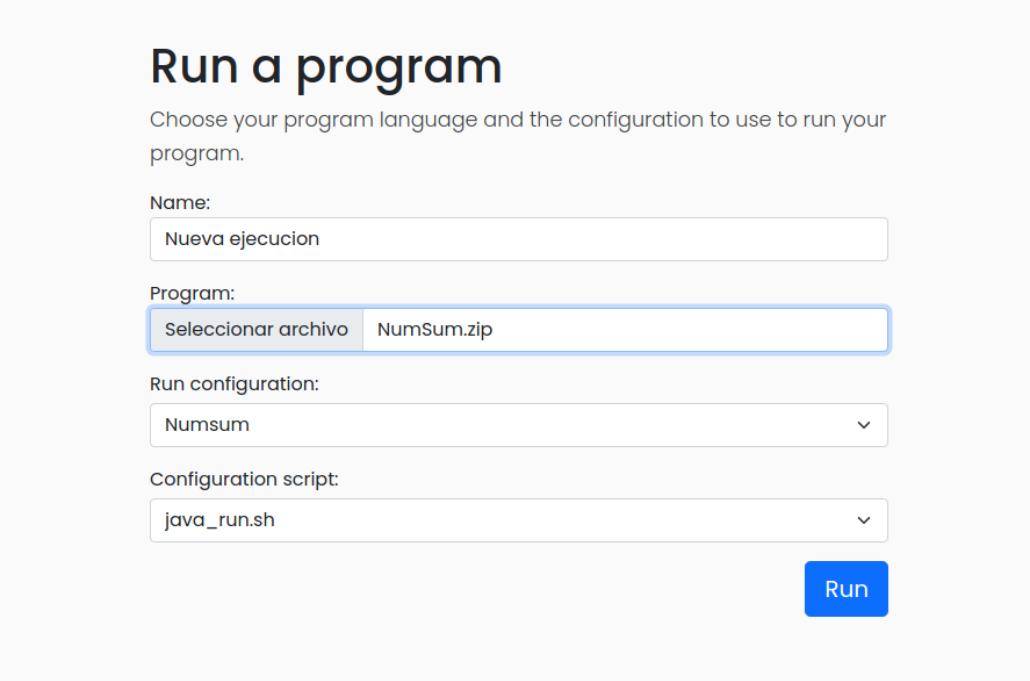

Figura B.4: Formulario de nueva ejecución.

Si el servicio está disponible se redireccionará al usuario al listado de ejecuciones, viendo la última en estado de *Queued*.

B.2. Administración

Para el administrador los listados no tienen únicamente lo asociado a su cuenta, sino todos los elementos del servicio. Además cuenta con un listado de los usuarios. En todos los elementos de los listados tiene un botón arriba a la derecha de cada elemento que le da la posibilidad de borrarlo (tras un modal de confirmación).

Para cada página de perfil de los distintos elementos el administrador cuenta con formularios que le permite modificar distintas características, ya sea la información de los usuarios, los tests de una configuración o incluso modificar el script de ejecución.

Un administrador también tiene la posibilidad de añadir nuevos scripts y configuraciones, con formularios similares a los que añaden una nueva ejecución. En el caso de la configuración incluye un formulario dinámico para configurar los tests desde la web (B.5).

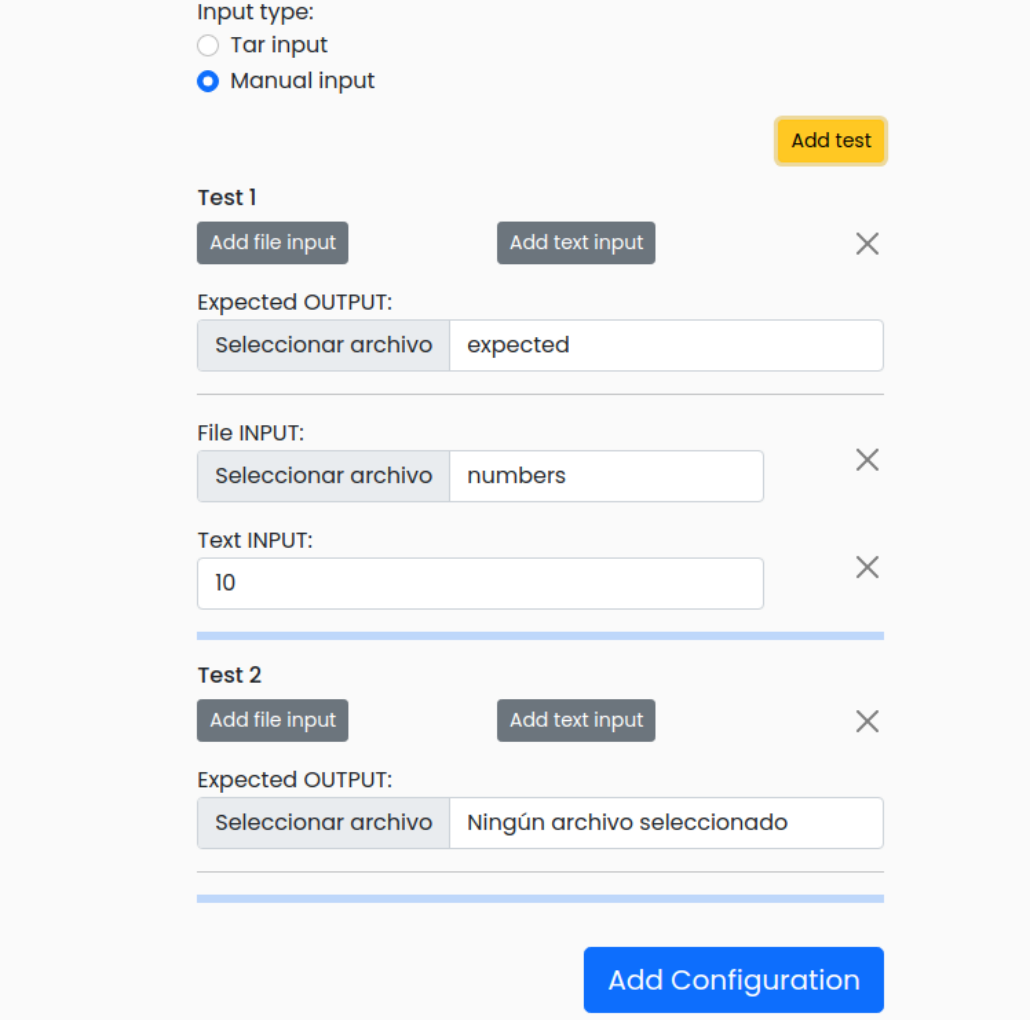

Figura B.5: Formulario de creación de tests.

Apéndice C

Manual de Desarrollo

C.1. Ampliación de la web

Para ampliar la web se propone seguir con la estructura presentada con Blueprints de la herramienta Flask. Aunque no son totalmente independientes se entiende que una vista esta incluida en un Blueprint si está incluida en la funcionalidad que proporciona ese Blueprint. Actualmente hay 3, *Auth*, que tiene las páginas y funciones que se utilizan para el manejo de la sesión; *Admin*, que tiene las páginas asociadas exclusivamente a administración; *Public*, incluye todas las páginas accesibles por cualquier usuario registrado.

C.2. Inclusión de nuevos lenguajes

Esta es la parte que más se ha intentado simplificar, solo es necesario añadir un nuevo script a la web que se comporte como se explica a lo largo de la memoria. El resultado del script se espera dentro del contenedor de ejecución en la ruta $/program/output/output$ con el esquema explicado previamente (validación, compilación, tests, análisis). Se puede utilizar como base el script de Java (java_run.sh) y adaptarlo a otros lenguajes o modificarlo como sea conveniente.

C.3. Propuestas

Las propuestas para ampliar la funcionalidad del servicio son las siguientes:

- Implementar barra de búsqueda para los listados. Se encuentra en la vista, pero no es funcional.
- Implementar la subida de tests por archivo comprimido, se encuentra en la vista, pero no es funcional.
- Crear fichero de reglas para el análisis y diseñar una vista según las reglas elegidas.
- Añadir grupos a los usuarios para mayor comodidad en gestión de permisos.
- Separar las órdenes de cada fase de ejecución cambiando los scripts a una agrupación de métodos a los que llamar uno a uno desde el mismo contenedor. Mejorar la vista para ver los distintos estados por los que pasa.
- Incluir enlaces desde los que se puedan descargar los ficheros de ejecución.

Bibliografía

[1] Google's Coding Competitions. *Code Jam Google Contest*. 2021. url: https://codingcompetitions.withgoogle.com/codejam (visitado 13-05-2021).

[2] Google's Coding Competitions. *Codejam 2019 Results*. 2021. url: https://codingcompetitions.withgoogle.com/codejam/archive/2019 (visitado 16-05-2021).

[3] DEVOPS.COM. *Top 10 Best Practices for Jenkins Pipeline Plugin*. 2019. url: https://devops.com/top-10-best-practices-for-jenkins-pipeline-plugin/ (visitado 14-06-2021).

[4] Inc. Docker. *Docker Software Documentation*. 2021. url: https://docs.docker.com/ (visitado 15-05-2021).

[5] Vincent Driessen. *Python-rq, Simple job queue library for python*. 2021. url: https://python-rq.org/ (visitado 15-03-2021).

[6] Kohsuke Kawaguchi. *Jenkins Software Documentation*. 2021. url: https://www.jenkins.io/doc/ (visitado 15-05-2021).

[7] Pallets. *Flask-SQLAlchemy configuration*. 2021. url: https://flask-sqlalchemy.palletsprojects.com/en/2.x/config/ (visitado 11-06-2021).

[8] Pallets. *Flask, web server library for python*. 2021. url: https://flask.palletsprojects.com/en/2.0.x/ (visitado 21-05-2021).

[9] PMD. *PMD source code static analyzer*. 2021. url: https://pmd.github.io/ (visitado 24-06-2021).

[10] Tim Hunt Richard Lobb. *CodeRunner plugin for moodle*. 2021. url: https://coderunner.org.nz/ (visitado 14-06-2021).

[11] Ximena Rodriguez. *Django vs Flask*. 2021. url: https://openwebinars.net/blog/django-vs-flask/ (visitado 11-06-2021).

[12] rody. *Docker PMD image*. 2021. url: https://github.com/rody/docker-pmd-image (visitado 24-06-2021).

[13] Markus Siemens. *TyniDB python DB library docs*. 2021. url: https://tinydb.readthedocs.io/en/latest/ (visitado 20-04-2021).

[14] Slant. *Spring-boot vs Flask*. 2021. url: https://www.slant.co/versus/158/1398/~spring-boot_vs_flask (visitado 11-06-2021).

[15] Pytest-dev Team. *Pytest Library Documentation*. 2021. url: https://docs.pytest.org/en/6.2.x/ (visitado 16-05-2021).

[16] Hsiaoming Yang. *Structure of a Flask Project*. 2018. url: https://lepture.com/en/2018/structure-of-a-flask-project (visitado 11-06-2021).