INTERFAZ WEB PARA DEMOSTRADORES DE TEOREMAS EN MAUDE

WEB INTERFACE FOR THEOREM PROVERS IMPLEMENTED IN MAUDE

TRABAJO FIN DE GRADO
CURSO 2023-2024

AUTOR
JAIME LOZANO DÍAZ (GII)
GONZALO VÍLCHEZ RODRÍGUEZ (GIS)

DIRECTOR
ADRIÁN RIESCO

GRADO EN INGENIERÍA INFORMÁTICA, GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

INTERFAZ WEB PARA DEMOSTRADORES DE TEOREMAS EN MAUDE

WEB INTERFACE FOR THEOREM PROVERS IMPLEMENTED IN MAUDE

TRABAJO DE FIN DE
GRADO EN INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

AUTOR
JAIME LOZANO DÍAZ (GII)
GONZALO VÍLCHEZ RODRÍGUEZ (GIS)

DIRECTOR
ADRIÁN RIESCO

CONVOCATORIA: SEPTIEMBRE 2024

GRADO EN INGENIERÍA INFORMÁTICA, GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

3 DE SEPTIEBRE DE 2024

RESUMEN

Interfaz web para demostradores de teoremas en Maude

Este proyecto consiste en hacer una interfaz web para poder trabajar con las herramientas CafeInMaude y CITP, las cuales son demostradores de teoremas; implementados en el lenguaje Maude, de forma más intuitiva.

CafeInMaude es una herramienta que permite trabajar con el lenguaje CafeOBJ a través de la cual se pueden demostrar teoremas haciendo uso de la sintaxis de CafeOBJ y herramientas desarrolladas explícitamente para este lenguaje.

CITP es una herramienta para demostrar que ciertos sistemas de software cumplen con propiedades específicas, utilizando grafos orientados o sistemas de transiciónes basados en constructoras.

Nuestra web permite ejecutar estas dos herramientas (CITP y CafeInMaude); y crear una conexión que posibilite al usuario enviar comandos de forma directa a través de esta, estos comandos se introducirán en Maude y las dos herramientas haciendo uso del módulo *pexpect* de Python, que permite generar aplicaciones hijo, controlarlas y responder a patrones esperados en su salida. Estos patrones serán los *prompts* correspondientes a cada una de las herramientas usadas (CafeInMaude y CITP). Una vez se haya encontrado dicho *prompt*, se capturará la salida previa si es que hubiese y se mostrará el resultado por pantalla para que el usuario pueda visualizarlo.

Palabras clave

Maude, CafeInMaude, CITP, CafeOBJ, Aplicación web, Pexpect

ABSTRACT

Web interface for theorem provers implemented in Maude

This project consists of making a web interface to be able to work with the tools CafeInMaude and CITP, which are theorem provers; implemented in the Maude language, in a more intuitive way.

CafeInMaude is a tool that allows working with the CafeOBJ language through which theorems can be proved using the CafeOBJ syntax and tools developed explicitly for this language.

CITP is a tool to demonstrate that certain software systems comply with specific properties, using oriented graphs or transition systems (arrow diagrams that show how things change from one state to another).

Our website allows us to run these two tools (CITP and CafeInMaude); and create a connection that allows the user to send commands directly through it, these commands will be introduced in Maude and the two tools using the Python pexpect module, which allows us to generate child applications, control them and respond to expected patterns in their output. These patterns will be the prompts corresponding to each of the tools used (CafeInMaude and CITP). Once this prompt has been found, the previous output, if any, will be captured and the result will be displayed on the screen so that the user can view it.

Keywords

Maude, CafeInMaude, CITP, CafeOBJ, web application, Pexpect

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

En el ámbito del desarrollo del software y la formalización de sistemas, es crucial contar con lenguajes que permitan realizar demostraciones de propiedades y especificaciones de forma automática, como es el lenguaje Maude, que sin embargo están desarrolladas con una interfaz de línea de comandos cuya interfaz no resulta accesible para todos los usuarios.

1.1 Antecedentes

Entre las diversas aplicaciones de Maude, se encuentran las herramientas de CITP y CafeInMaude, que centrarán el foco de trabajo de este Trabajo de Fin de Grado (TFG).

El *Constructor-based Interactive Theorem Prover* (CITP) es una herramienta que permite verificar propiedades de sistemas software.

CafeOBJ in Maude (CafeInMaude) es un intérprete que permite la implementación combinada de especificaciones CafeOBJ y la capacidad de realizar pruebas como ya dijimos previamente. A través de los especificadores se pueden escribir puntuaciones de prueba en CafeOBJ y realizar pruebas con ellas al ejecutarlas.

La ejecución habitual de Maude se realiza a través de una interfaz de línea de comandos (CLI), que, aunque tiene ciertas ventajas, presenta los inconvenientes de que dificulta su uso para un gran número de usuarios, con el inconveniente añadido de que está principalmente diseñada para sistemas UNIX, pudiendo solamente ser utilizada por usuarios de Windows, el sistema operativo más utilizado a nivel global (Desktop Operating System Market Share Worldwide, 2020), utilizando herramientas de virtualización de Linux.

En el contexto del proyecto desarrollado en este TFG, no resulta relevante la descripción en profundidad del funcionamiento del lenguaje de Maude y las herramientas de CITP y CafeInMaude, más allá de que son demostradores de propiedades implementados en Maude, ya que solo nos centraremos en que estas herramientas tengan las mismas funcionalidades a través de la web que en la CLI.

Se puede profundizar más sobre el lenguaje de Maude consultado la obra All About Maude - A High-Performance Logical Framework (Manuel Clavel, 2007)

1.2 Objetivos

La finalidad principal de este TFG es la de diseñar y desarrollar una herramienta que permita la ejecución de algunas herramientas como CITP y CafeInMaude a través de Maude, de forma sencilla desde cualquier navegador web.

La aplicación deberá contar con las siguientes características:

1. Compatibilidad amplia con los navegadores actuales: Deberá ser accesible desde cualquier navegador web cumpliendo con los estándares actuales en esa materia.
2. Diseño responsive: Deberá estar diseñada para poder acceder desde cualquier dispositivo independientemente del tamaño de la pantalla y se puedan leer correctamente los contenidos, o sea, una aplicación responsive.
3. Interfaz amigable y fácil de usar: La aplicación deberá contar con una interfaz intuitiva, de forma que el manejo de esta pueda hacerse sin necesidad de entrenamiento previo del usuario.
4. Seguridad y privacidad: Deberá garantizar la seguridad de cualquier dato sensible que posea, desde el diseño.
5. Capacidad de actualización y escalabilidad: La aplicación deberá diseñarse modularmente, para que sea fácil de corregir y, si hay que añadir un nuevo módulo o funcionalidad, que esto se consiga sin rediseñar toda la aplicación.

Se ha creado un repositorio de GitHub para el control de versiones y la publicación del proyecto de la aplicación, que puede encontrarse en la siguiente dirección web:

https://github.com/gonzalovlchz/WebMaude

La aplicación desarrollada se encontrará alojada en un servidor que será accesible a través del siguiente enlace:

https://maude.ucm.es/webmaude/

1.3 Plan de trabajo

Como se puede apreciar en la Ilustración 1-1 Diagrama de Gantt, las dos primeras semanas las dedicamos a la fase de planificación, en la primera etapa de esta fase, la cual duró una semana, ambos nos dedicamos a investigar el lenguaje Maude y, por individual, las herramientas correspondientes de cada uno con el objetivo de reunir toda la información posible y necesaria para entender el lenguaje y sus herramientas y así poder realizar correctamente el proyecto que se nos designó. Durante la segunda etapa de esta fase nos repartimos el trabajo para que uno de nosotros definiera los requisitos del sistema y el otro elaborara el plan de proyecto y un cronograma detallado.

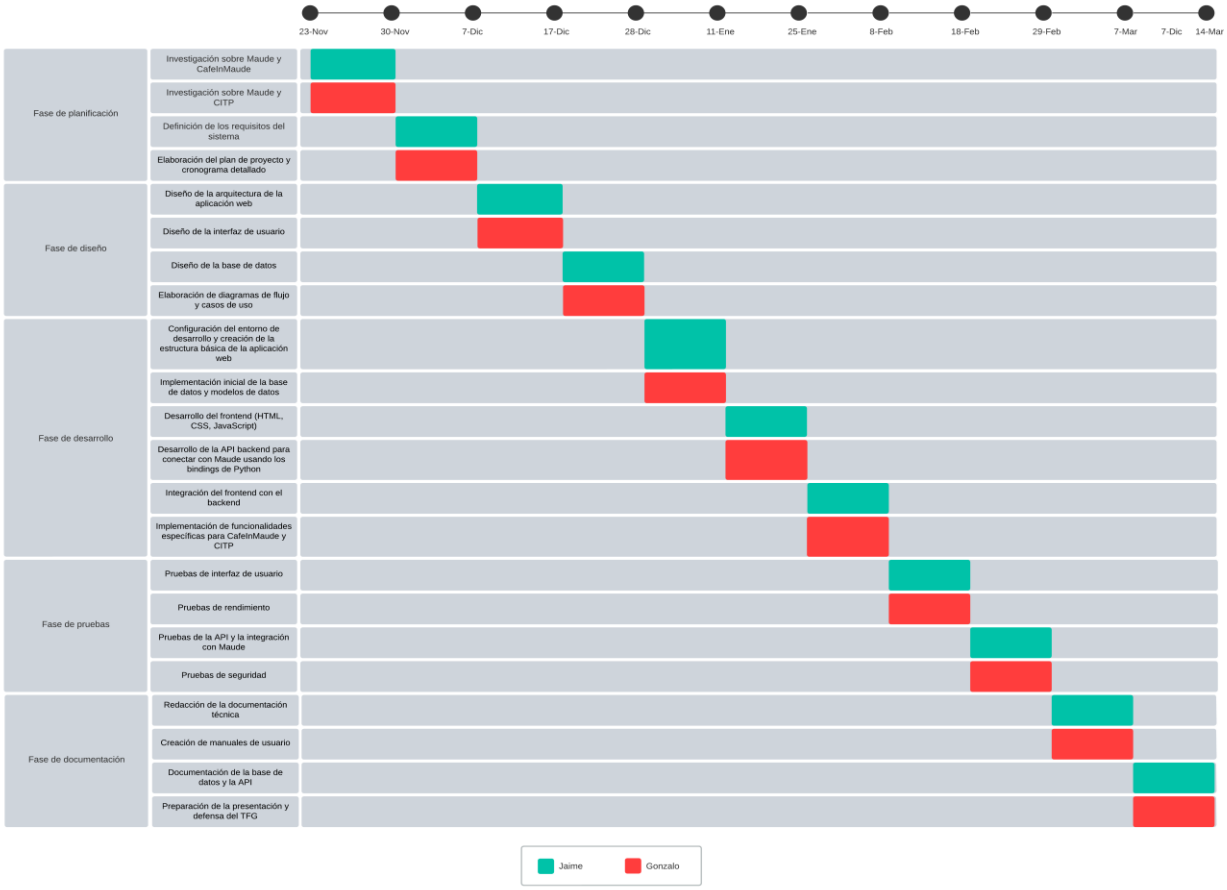

Ilustración 1-1 Diagrama de Gantt

La fase de diseño la dividimos también en dos etapas, en las que durante la primera semana y media uno de nosotros debía hacer el diseño de la arquitectura de la interfaz web y otro el diseño de la interfaz de usuario. Durante la segunda semana y media de esta etapa uno debía diseñar la base de datos considerando todas las tablas necesarias para el funcionamiento de la interfaz. En este periodo el otro debía elaborar los diagramas de flujo y casos de uso.

En la fase de desarrollo, que es la más larga con una duración total de 6 semanas, durante las dos primeras uno de nosotros se centró en configurar el entorno de desarrollo y crear la estructura básica de la interfaz web, mientras que otro dedicó ese tiempo a la implementación inicial de la base de datos, construyendo las tablas, previamente pensadas, que considerábamos necesarias, pues más adelante nos dimos cuenta de que había que añadir alguna tabla, fila o columna nueva para el correcto funcionamiento de la implementación.

Las dos semanas siguientes fueron destinadas al desarrollo del *front-end*, haciendo uso de código HTML, CSS y JavaScript y al desarrollo del *back-end* para poder conectar con Maude a través de *pexpect*. Las dos últimas semanas las utilizamos para integrar el código perteneciente al *front-end* con el del *back-end* y a implementar las funcionalidades específicas para cada una de las herramientas (CafeInMaude y CITP).

Las siguientes semanas las destinamos a la fase de pruebas en la que la primera semana y media la dedicamos a pruebas de interfaz de usuario y de rendimiento para verificar el correcto funcionamiento de todo la web y poder detectar errores para solucionarlos antes de pasar a la siguiente fase.

Durante la última fase, la de documentación, nos centramos en redactar la documentación técnica del proyecto, la creación de manuales de usuario claros y sencillos de entender para un correcto uso de la web y a documentar la base de datos para redactar esta memoria y que el tutor nos diese indicaciones de posibles cambios o consejos para poder mejorarla.

Capítulo 2 - Estado de la cuestión

En esta sección documentaremos qué proyectos similares hemos encontrado, en qué nos hemos basado de estos que nos pareciese interesante incorporar al nuestro, en qué nos hemos fundamentado para el desarrollo y qué hemos considerado oportuno mejorar y por qué en nuestra web.

Primero investigamos acerca del lenguaje Maude[1], para ver que los programas de este lenguaje se basan en la definición de funciones y datos en base a unas reglas de escritura y operaciones algebraicas. Según estas reglas se transforman los términos declarados en los distintos módulos.

También observamos que ambas herramientas (CafeInMaude y CITP) estaban destinadas a la realización de demostraciones por parte de los usuarios y mostrar los resultados obtenidos posteriormente.

En nuestra búsqueda de proyectos similares, o que guarden algún tipo de relación con el nuestro, nos topamos con ciertas interfaces web que nos sirvieron para formar una base de qué queríamos integrar en la nuestra, y observar qué pueden hacer y qué no. De este modo, pudimos pensar qué opciones o funcionalidades deseábamos incorporar a nuestro proyecto que nos resultasen interesantes y que el resto de las interfaces no consideraron o no incorporaron.

Una de las más interesantes que encontramos fue la de AGES[2]. Ésta permite importar archivos alojados en el equipo del usuario o incorporar su código por medio de un área de texto para poder así trabajar con él. Incorpora además una opción para que el usuario seleccione las interpretaciones del predicado, así como una opción para seleccionar con qué reglas quiere interactuar. Otra opción para tener en cuenta y que

---

[1] The Maude System - https://maude.cs.illinois.edu/wiki/The_Maude_System

[2] AGES - http://zenon.dsic.upv.es/ages/

incorpora esta web es la de que se puede ajustar el *timeout* de la prueba pudiendo aumentarlo y reducirlo según sea necesario. También cuenta con un área de texto en el que el usuario puede especificar las metas que quiere obtener en dicha prueba.

Otra interfaz interesante es la de Hets, the DOLiator[3]. Esta interfaz consta del mismo sistema para la selección de archivos que la anterior con la diferencia de que dota al usuario de la posibilidad de seleccionar de qué tipo es el archivo seleccionado y la posibilidad de elegir entre una gran cantidad de ejemplos predeterminados. Esta interfaz también permite exportar los resultados de las pruebas en varios formatos y, al igual que la anterior, cuenta con la opción de modificar el timeout.

En general, todas las interfaces que hemos encontrado trabajan con el mismo sistema de importación de archivos para la realización de las pruebas, con la diferencia de que las dos mencionadas anteriormente además dotan a sus usuarios de poder realizar una configuración de éstas más amplia desde la propia interfaz. En la web de The ELP Group[4] se puede consultar otras aplicaciones para realizar este tipo de pruebas.

No hemos encontrado ninguna interfaz que trabaje específicamente con las herramientas CafeInMaude y CITP, por lo que nuestro proyecto se presenta como una novedad en este ámbito. Con base a lo comentado con anterioridad, y ya que estas herramientas trabajan cargando archivos sobre los que realizar las pruebas, integramos un sistema de importación de archivos alojados en el equipo de nuestros usuarios añadiendo la opción de cargar ejemplos de prueba predeterminados. Estos ejemplos son gestionados por el administrador pudiendo añadir nuevos o eliminarlos.

Tampoco hemos visto que los proyectos existentes proporcionen una forma de guardar las sesiones de los usuarios pudiendo así recuperarlas en cualquier momento y continuar trabajando sobre ellas, lo cual nos pareció muy útil, y por ello decidimos integrarlo en nuestra aplicación. Además, incorporamos un panel derecho desde el cual el usuario pudiese seleccionar ciertas opciones útiles en la realización de sus

---

[3] Hets, the DOLiator - http://rest.hets.eu

[4] The ELP Group – Developed Software - https://elp.webs.upv.es/soft.html

pruebas, como pueden ser ver los objetos actuales de la prueba, mostrar el árbol de pruebas completo, listar las metas y pruebas con las que se está trabajando, etc.

En la Tabla 2-1 Comparación entre herramientas existentes se puede ver una comparación entre las diversas herramientas que hemos encontrado.

La opción de poder modificar el *timeout* de las pruebas por parte de los usuarios también es una opción para tener en cuenta debido a que hay algunas de éstas que tienen una duración demasiado larga. Pero hasta el momento esta configuración solo puede ser modificada por el administrador.

Tabla 2-1 Comparación entre herramientas existentes

Interfaces	*Hets, the DOLiator*	*HEMS*	*AGES*	*ACUOS^2*	*PRESTO*	***Nuestra aplicación***
Importar archivos	✓	✗	✓	✓	✓	✓
Escribir los módulos en un área de texto	✓	✓	✓	✓	✓	✗
Proporcionan ejemplos	✓	✓	✓	✓	✓	✓
Configuración de la prueba	*Interfaz comando*	*Comando*	*Interfaz comando*	*Comando*	*Comando*	*Comando*
Ayuda para la construcción de comandos	✗	✗	✓	✗	✗	✗
Posibilidad de modificar el timeout por parte del usuario	✓	✗	✓	✗	✗	✗
Autoconfiguración de pruebas de ejemplo	✓	✓	✓	✓	✓	✓
Exportar la prueba	✓	✗	✗	✗	✗	✗
Guardar el estado de las pruebas	✗	✗	✗	✗	✗	✓

Capítulo 3 - Tecnologías utilizadas

En este capítulo describiremos las tecnologías utilizadas en el proyecto y su razón de ser, así como las alternativas que se han valorado y por qué se han rechazado en cada caso. Se ha intentado además que todas las tecnologías sean de parte del proyecto de software libre y de código abierto.

3.1 Elección de Python y Django como base del proyecto

Para desarrollar la web, se ha buscado un *framework* adecuado y desarrollado en Python y se ha encontrado que el que se utiliza más ampliamente y con mayor comunidad, además de ser un proyecto de código abierto, es el Django.

Django es un *framework* de Python para el desarrollo de aplicaciones web que posee ciertas características que nos son de gran utilidad a la hora de desarrollar este proyecto (Django overview, s.f.):

- Rapidez: está diseñado para permitir a los desarrolladores diseñar sus aplicaciones de la forma más rápida posible. Esto es beneficioso en nuestro proyecto ya que nos permite centrar el foco de trabajo en la integración con Maude.
- Precargado: Django incluye varios extras que llevan a cabo las actividades más comunes de los sitios web. Estos módulos son relevantes en nuestro trabajo ya que evitan desarrollar partes de la aplicación que son comunes a muchas otras aplicaciones, como la autenticación de usuarios que utilizamos y el sitio web para administrador.
- Seguridad: ayuda a prevenir a los desarrolladores errores de seguridad como la inyección de SQL o el *Cross-site request forgery*. En el contexto de este TFG, nos releva de la tarea de encontrar las mejores técnicas de seguridad en comunicaciones web y de los sistemas.

Se puede profundizar más sobre este marco de trabajo leyendo *Learning Django Web Development: From Idea to Prototype, a Learner's Guide for Web Development with the Django Application Framework* (Sanjeev Jaiswal, 2015).

3.2 Ejecución de subprocesos

Se han buscado diversas alternativas para ejecutar subprocesos desde Python, como son *subprocess* y *pexpect*.

El módulo *subprocess* de Python es el más común para este propósito. Este es un módulo que permite crear procesos nuevos, conectarse a sus tuberías de input/output y obtener sus códigos de retorno. (subprocess — Subprocess management, s.f.)

Este módulo no resulta adecuado en este proyecto, al ser independientes las tuberías de input y output. Cuando un proceso espera la introducción de datos por parte del usuario, se bloquea la ejecución del programa.

La otra alternativa, *pexpect*, es un módulo puro de Python que permite ejecutar subprocesos y controlarlos, respondiendo a patrones conocidos en su salida, controlando la aplicación como si un humano estuviera introduciendo los comandos en una sesión de terminal. (Pexpect, 2013)

Utilizando el módulo *pexpect*, y conociendo el programa que se va a ejecutar, se puede predecir el *prompt* que el programa va a darle al usuario para introducir texto, y una vez aparece este *prompt* indicado, se introduce al programa el comando nuevo que se quiere ejecutar.

En el caso de Maude, como se puede apreciar en la Ilustración 3-1 Ejecución de ejemplo de Maude, después de la respuesta a cada comando del usuario, Maude muestra el *prompt* "Maude> ". Podemos pasarle al objecto de *pexpect* este patrón a través del método *expect()*, que espera a que el programa muestre ese *prompt* para continuar con la ejecución.

Utilizaremos también el método *before()* para extraer de la ejecución la salida del comando actual.

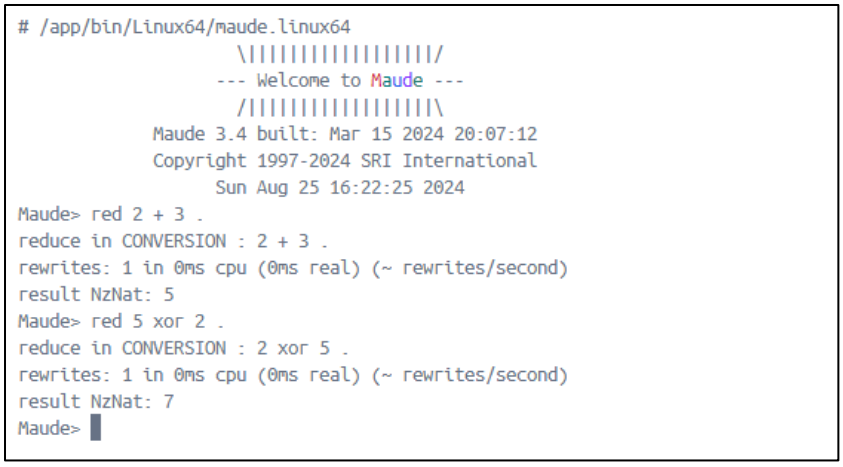

Ilustración 3-1 Ejecución de ejemplo de Maude

En el caso de CITP, el *prompt* es "CITP >". Sin embargo, una vez que ejecutas el comando "begin proof [P] of [G]", el *prompt* cambia para indicar que se está trabajando en esa prueba, a "CITP/[P] >", como se puede observar en el ejemplo de la Ilustración 3-2 Ejemplo de ejecución de CITP. Esto no es problemático ya que el método *expect()* de la biblioteca admite tanto cadenas de texto literales como patrones. En este caso se puede usar una expresión regular como "*r'CITP(?:/[^]*)? >*'".

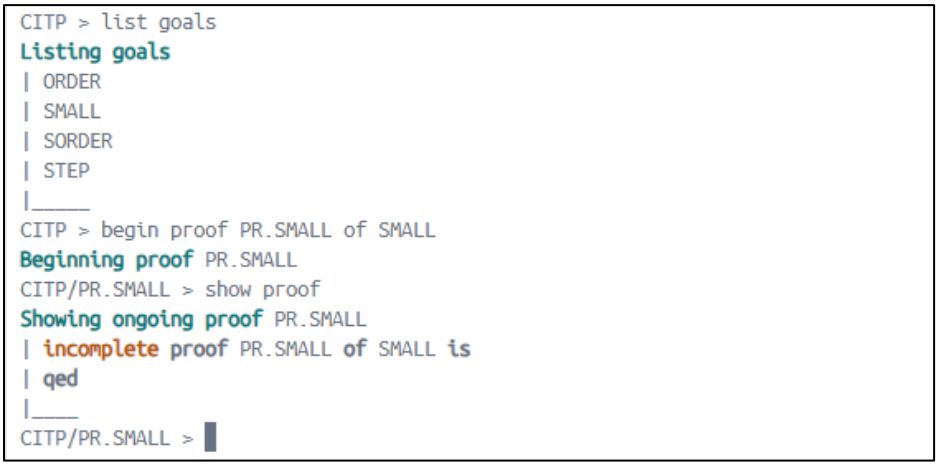

Ilustración 3-2 Ejemplo de ejecución de CITP

En el caso de CafeInMaude, el *prompt* es "CafeInMaude> " y este no varía durante la ejecución por lo que no es necesaria su modificación a la largo de la misma.

3.3 Base de datos

El marco de desarrollo Django facilita enormemente la integración con distintos tipos de bases de datos para las aplicaciones en las que son necesarias. A través del conector adecuado, y habiendo definido correctamente el modelo de datos de la aplicación, Django crea y actualiza todas las tablas necesarias (a través del comando "migrate") y abstrae para el desarrollador las instrucciones básicas (inserción, lectura, actualización y borrado) a través de objetos de datos.

En nuestro caso, es necesaria una base de datos para, al menos, las tareas de autenticación de usuarios con sus permisos, almacenamiento de las sesiones de cada usuario y del histórico de comandos de cada sesión.

En un principio, la aplicación se empezó a desarrollar utilizando una base de datos de MariaDB, que es un sistema de gestión de base de datos de código abierto derivado del comercial MySQL.

Si bien al utilizar una base de datos externa se simplificaba el acceso a los datos y la modificación a través de un cliente externo comúnmente utilizado, como puede ser PhpMyAdmin (web) o HeidiSQL (escritorio), la instalación de esta base de datos (se requiere instalar el servidor, así como las bibliotecas necesarias de Python) así como su mantenimiento es más complejo. A pesar de las ventajas de este sistema, en el contexto de este proyecto la desventaja tiene un peso mayor, por lo que se opta por cambiar el tipo de la base de datos a SQLite.

SQLite es, al igual que MariaDB, un sistema de bases de datos relacional de código abierto, que sin embargo es mucho más ligero y no requiere instalación. Este tipo de base de datos es ideal para aplicaciones pequeñas, como es el caso de este proyecto, además de que no necesita un servidor ya que está contenida en un solo archivo. Es, además, la base de datos por defecto en Django, ya que como hemos descrito anteriormente, este busca la rapidez en la configuración.

El libro *The Definitive Guide to SQLite* (Grant Allen, 2010) profundiza sobre este sistema de bases de datos, si se quiere aprender más sobre él.

3.4 Tecnologías de interfaz de usuario

En el diseño web de la aplicación, hemos intentado usar algunas de las tecnologías más utilizadas hoy en día, como son Bootstrap y Vue.js.

Hemos optado por un diseño con una sola página principal, en la que se van cargando los distintos componentes y datos a través de AJAX. Es decir, la página entera solo se carga una vez en el navegador del usuario, y solo se hacen peticiones al servidor de un fragmento de página concreto o de sus datos.

Estos datos son devueltos por el servidor en forma de respuesta JSON, que es un tipo de definición de estructuras de datos ampliamente utilizado tanto en las tecnologías de *front-end* como de *back-end* para intercambiar mensajes.

3.4.1 Bootstrap

Bootstrap es el *framework* de diseño de interfaces web más utilizado actualmente, de código abierto, desarrollado por Twitter.

Este marco es compatible con todos los navegadores actuales, con lo que se cumple el punto 1 de los objetivos.

Está compuesto de múltiples componentes (como pueden ser botones, formularios, ventanas modales, ...) y estilos predefinidos que facilitan la labor del desarrollador, tanto en el diseño de la estructura y su apariencia, como en el comportamiento dinámico de cada componente, haciendo además que la aplicación mantenga una consistencia interna para el usuario.

Al ser el *framework* de diseño más popular, los usuarios están normalmente ya familiarizados con el diseño (consistencia externa), lo que facilita el aprendizaje de la interfaz y reduce el tiempo que el usuario emplea en empezar a trabajar con la aplicación.

Bootstrap además está diseñado para ser compatible con dispositivos con cualquier tamaño o resolución de pantalla, es decir, responsive, con lo que se cumple el punto 2 de los objetivos.

3.4.2 Vue.js

Este *framework* de Javascript, de código abierto, facilita el diseño de sitios web de una sola página, a través de la declaración de los distintos componentes y utilizando una sintaxis de plantilla.

Vue.js mantiene un modelo de datos que es actualizable y que propaga los cambios automáticamente a los distintos elementos, además de un sistema de observadores de eventos.

A través de este modelo, cuando hacemos una petición de datos al servidor, la respuesta en forma de JSON se procesa y se asigna a la variable correspondiente. A continuación, Vue.js propaga los cambios en la interfaz a los elementos cuyo contenido corresponde a esa variable de forma automática.

3.5 Herramientas de control de versiones: GitHub

A la hora de desarrollar la aplicación, es interesante contar con una herramienta de control de versiones, que facilita el desarrollo colaborativo a través de la creación de diversas ramas, y ayuda a garantizar que no se pierden cambios. En caso de errores, es fácil restaurar una versión anterior.

En nuestro caso, hemos optado por la utilización de GitHub. Esta plataforma, accesible desde cualquier parte de internet, utiliza el software de control de versiones Git.

GitHub además cuenta con GitHub Actions, que es una herramienta que ayuda con las tareas de Integración Continua sin la necesidad de utilizar sitios web o plataformas de terceros.

En este proyecto, hemos utilizado GitHub Actions para la creación y compilación de contenedores Docker, que facilitan el despliegue de la aplicación en un servidor. Para esto, se ha creado un flujo de trabajo que se activa cada vez que se hace "push" sobre la rama principal (*main*).

Este flujo de trabajo utiliza la última versión de Ubuntu disponible para:

1. Hacer "checkout" de la rama *main*.

2. Instalar *Docker Buildx* y otras herramientas necesarias para la compilación y la firma
3. Compilar la imagen Docker a partir del *Dockerfile* disponible en el repositorio y publicar esta imagen en el servidor propio de GitHub (ghcr.io)
4. Firmar la imagen publicada

Se puede obtener más información sobre Git en su manual oficial, Pro Git (Chacon, 2014).

3.6 Creación de contenedores con Docker

Docker es una plataforma que permite a los desarrolladores empaquetar aplicaciones en contendores simplificando su instalación y asegurando la instalación de las dependencias que pueda tener.

Cada contenedor se ejecuta en un entorno aislado, lo que garantiza la seguridad de la máquina. Si la seguridad de un contenedor se compromete, solo se verá afectado el entorno de este contenedor y no la máquina anfitriona.

Esto es especialmente importante en nuestro proyecto, ya que al utilizar *pexpect* con el ingreso de comandos por parte del usuario a través de la web, existía una vulnerabilidad en la seguridad.

Con el uso del contenedor, si un atacante tuviera éxito y consiguiera ejecutar instrucciones distintas de las autorizadas en el terminal, solo se vería afectado el entorno del contenedor y no el servidor.

Además, el aislamiento de dependencias hace que cada contenedor pueda tener una versión distinta de una misma biblioteca o herramienta, eliminando los conflictos que puedan existir.

Docker además ayuda garantizando la consistencia entre entornos. Si la máquina es compatible con un servidor Docker, los contenedores que este contenga se ejecutarán igual que en cualquier otra máquina. Asimismo, Docker contribuye al desarrollo colaborativo ya que, al utilizar el mismo entorno, se eliminan problemas entre los desarrolladores de "no funciona en mi máquina".

Para la construcción de un contenedor Docker, se crea un archivo *Dockerfile* que contiene las instrucciones que se ejecutarán en cada virtualización. Los contenedores se suelen crear a partir de otras imágenes de contenedores.

En nuestro proyecto, al principio optamos por el uso de imágenes de contenedores de Python. Esto facilita el despliegue de aplicaciones desarrolladas en Python y la gestión de sus dependencias a través de PIP. En el contexto de nuestro trabajo, sin embargo, tuvimos que optar por otra imagen, ya que el uso de la imagen de Python no nos permitía controlar el entorno en el que se ejecutaría Maude a través de los subprocesos, si es que fuera acaso posible.

Después de una comparación de las distintas imágenes que existen para Docker de sistemas operativos tipo Linux (ya que Maude está desarrollado para ejecutarse en Unix, y MacOS no es una opción fácil de encontrar en Docker, ni sería muy adecuada, ya que se intenta utilizar herramientas dentro del proyecto de Software Libre), hemos optado por usar una versión de Debian.

En este entorno, primero se instala Python y las herramientas asociadas y luego se instalan las dependencias necesarias de la aplicación a través de "pip install -r requirements.txt". A continuación, se exporta la variable de entorno de MAUDE_LIB tal y como se describe en la guía de configuración de Maude y CITP, y a continuación se ejecuta el script de inicio del contenedor "entrypoint.bash".

Este es un script de consola que realiza las configuraciones iniciales:

1. Ejecuta el comando "migrate" que genera la estructura de tablas necesarias.
2. Crea el superusuario "root" con la contraseña "root".
3. Ejecuta el comando "runserver" que pone en marcha el servidor, en la interfaz 0.0.0.0 con el puerto 8000

Es posible conocer más sobre la tecnología de creación de contenedores con Docker en el libro *The Docker Book* (Turnbull, 2019).

Capítulo 4 - Arquitectura

A la hora de describir la arquitectura de la aplicación, diferenciaremos entre el *back-end* (la parte que se ejecuta en el servidor) y el *front-end* (la parte que se ejecuta en el navegador web del usuario).

4.1 Arquitectura del *Back-end*

Para desarrollar esta aplicación, se ha usado el *framework* web Django de Python. Este *framework* crea por defecto una estructura básica de carpetas y archivos en la raíz del directorio en el que nos encontremos a través del comando "startproject [proyecto]", en nuestro caso WebMaude, como la siguiente:

- WebMaude /
    - manage.py
    - WebMaude /
        - __init__.py
        - settings.py
        - urls.py
        - asgi.py
        - wsgi.py

El fichero *manage.py* es el punto de entrada ejecutable de la aplicación y a través del cual se ejecutan los comandos que hemos visto anteriormente.

Dentro del fichero settings.py se configuran los parámetros de forma global que utilizaremos dentro de la aplicación. En este fichero es dónde se configuran, entre otros, el conector y los datos de conexión a la base de datos, las rutas para los ficheros estáticos (*static*) y para los ficheros subidos (*media*), y otras variables definidas por nosotros, como las rutas de los ejecutables de Maude, CITP y CafeInMaude, o el tiempo límite por defecto de ejecución de cada comando.

En el archivo urls.py se guarda la configuración de rutas base del proyecto. En este archivo se incluye la ruta "/admin" al portal de administración generado por defecto, y la ruta a la aplicación MaudeApp, cuya creación se explica a continuación, en la ruta "/". Asimismo, se incluye aquí la ruta a los ficheros estáticos y los subidos por el usuario.

Este proyecto creado es una base para las distintas aplicaciones que se pueden desarrollar dentro de él. En nuestro caso, hemos desarrollado la aplicación MaudeApp.

Para crear la nueva aplicación, se utiliza el comando "startapp MaudeApp", que genera asimismo una estructura de ficheros y carpetas dentro de la carpeta raíz WebMaude, de la siguiente forma:

- MaudeApp/
    - __init__.py
    - admin.py
    - apps.py
    - migrations/
        - __init__.py
    - models.py
    - tests.py
    - views.py

En este directorio se definirán el modelo de datos de la aplicación, los métodos del controlador, que en Django se denominan vistas (*views*), y las vistas a través de plantillas (*templates*).

4.1.1 Modelo de datos

Dentro del fichero *models.py* hemos definido el modelo de datos de la aplicación como se describe en la Ilustración 4-1 Diagrama ER del modelo, en nuestro caso:

La clase *Session* contiene la información de una sesión de chat determinada. Tiene como clave foránea el usuario al que pertenece la sesión, y contiene el tipo (CITP

o CafeInMaude), así como un nombre descriptivo, la fecha y hora de creación y modificación.

Command contiene la información de cada comando que se envía, así como la salida que le da el intérprete. Cada comando pertenece a una sesión concreta (clave foránea), y tiene un atributo de *timestamp* que marca el momento en el que se escribe el comando.

En la clase *ExampleFile* se definen los archivos que el administrador del programa sube para que sean utilizados como ejemplo. Contienen el nombre que se le da al ejemplo, el tipo de sesión para el que se puede utilizar (CITP o CafeInMaude), y las rutas a los ficheros, tanto de Maude (se ejecutan directamente en el intérprete de Maude), como del programa concreto (se ejecutan en el intérprete de CITP o CafeInMaude).

Por último, se ha definido la clase *SiteSetting* que contiene los parámetros que el administrador puede definir: de momento, solo el tiempo límite de ejecución de cada comando. Esta clase tiene los atributos de *key* (nombre del parámetro) y *value* (su valor).

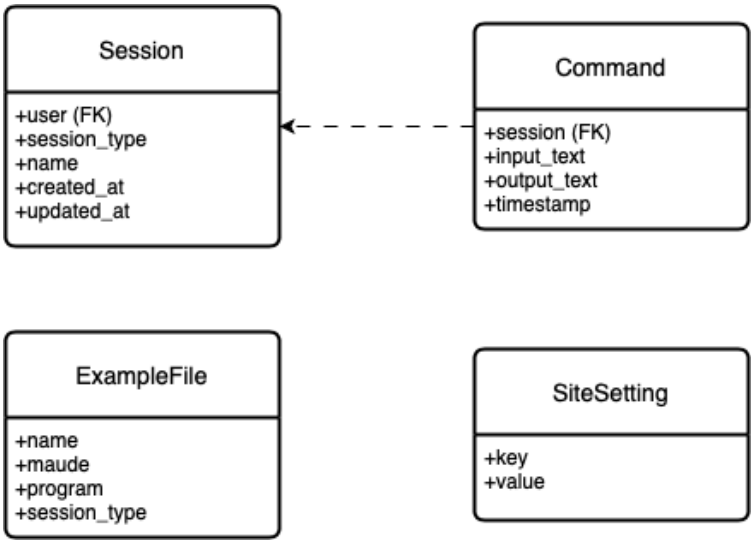

Ilustración 4-1 Diagrama ER del modelo

4.1.2 Vistas del controlador

Para intentar realizar un diseño más modular, se ha sustituido el archivo *views.py* por un directorio *views*, que contiene los ficheros de las vistas organizados por funcionalidad.

A la hora de elegir la nomenclatura de cada función en las vistas, se ha seguido la costumbre de que los nombres contengan al principio "post" o "get" según el tipo de petición que se espera del cliente.

Se ha definido un fichero *index_views* que contiene las vistas principales de la aplicación, como son el *get_index()*, y el formulario de autenticación del usuario *get_login_form()*.

En el fichero *acl_views* se han definido las operaciones relativas al proceso de autenticación de los usuarios, como son la petición de autenticación de usuarios (*post_login*), que comprueba si el usuario puede identificarse y en caso afirmativo lo redirige a la pantalla principal, y el cierre de sesión (*get_logout*), que tras realizar la operación correctamente redirige al usuario al formulario de autenticación.

Dentro del fichero *chat_views* se ha definido toda la lógica más relevante para este proyecto, relacionada con el funcionamiento principal de la aplicación, es decir todas las relacionadas con las sesiones de chat y la ejecución de comandos:

4.1.2.1 Funciones de chat_views

- *get_start_new_session():*

Esta función recibe en su petición POST un tipo de sesión, y crea en la base de datos un objeto de sesión. Asimismo, crea en el directorio *maude_files* una carpeta que tiene como nombre el identificador de la sesión. Dentro de esta carpeta, crea los archivos vacíos de "previous", en el que se añade el contenido de un fichero Maude en el caso de que el usuario lo haya añadido para sesiones de CITP, y un archivo con el nombre del id de la sesión. En este archivo se almacenarán los contenidos de los ficheros subidos por el usuario durante la ejecución, así como los comandos que el usuario introduzca, en orden cronológico, para poder retomar la sesión posteriormente.

Esta función devuelve una respuesta JSON que contiene el id de la sesión.

- *get_chat_session():*

En esta función se recibe a través de la petición GET un id de una sesión, y, tras comprobar que la sesión pertenece al usuario identificado, devuelve en JSON la lista de

comandos (con el comando ejecutado y la respuesta obtenida), y el tipo de sesión (CITP o CafeInMaude).

- *get_all_sessions():*

Esta función devuelve la lista de sesiones del usuario identificado.

- *get_files_by_session_type():*

En esta función se recibe mediante GET el tipo de sesión para el que se solicitan los archivos de ejemplo, y devuelve la lista con los nombres de estos ejemplos, con el objetivo de que el usuario pueda elegir uno si lo desea.

- *post_execute_new_command():*

Esta función recibe mediante POST el id de la sesión actual, y comprueba que pertenece al usuario identificado.

Si el usuario ha seleccionado un ejemplo del desplegable, se busca ese ejemplo en el modelo, y, si existe: se agrega el contenido del fichero Maude al fichero "previous" y el contenido del fichero del programa al fichero que tiene de nombre el id de la sesión.

Si el usuario ha agregado uno o varios ficheros a través del selector de archivos, estos también se añaden al fichero que tiene de nombre el id de la sesión.

Si el usuario ha especificado un comando, este se almacena en el modelo, y, dependiendo del tipo de sesión (CITP o CafeInMaude):

- Si es de tipo CITP, si el usuario ha especificado además un comando extra (en la plantilla se sitúan en el panel derecho), se agregan este y el comando principal como parámetros para la función *ExecuteCITPCommand()*, que es la que ejecutan en el sistema de Maude el comando. Esta devuelve la salida tanto del comando principal como del comando extra.
- Si es de tipo CafeInMaude, se ejecuta la función *ExecuteCafeInMaudeCommand(),* que de forma análoga devuelve la salida del comando.

Las salidas de los comandos se almacenan en el modelo para el comando actual, y posteriormente, se devuelve en formato de JSON la salida del comando principal, y en su caso, del comando extra.

- *executeCITPCommand():*

Esta es una función auxiliar que se encarga de ejecutar el comando de CITP y devolver el resultado. Para ello, recibe el fichero "previous", el fichero del id de la sesión, el comando nuevo, el comando extra si existe y el tiempo límite de ejecución si se especifica.

Se instancia el objeto de *pexpect* con el ejecutable de Maude, y se carga el fichero "previous".

A continuación, se carga el fichero que contiene el ejecutable de CITP (*run-citp.maude*). Se añade a CITP el fichero con el id de la sesión (que contiene el histórico de comandos y archivos cargados), y posteriormente se ejecutan el comando principal y el comando extra si se ha especificado.

Se elimina de la respuesta del comando principal, la propia petición del comando, que *pexpect* incluye por defecto.

Posteriormente, se devuelven las salidas del comando principal y el extra en su caso.

Si se hubiera alcanzado el tiempo límite de ejecución, se devolvería un texto explicativo.

- *executeCafeInMaudeCommand():*

De forma análoga a la función explicada anteriormente, esta es una función auxiliar que ejecuta el comando en sesiones de CafeInMaude.

Recibe solamente el fichero del id de la sesión (ya que no es necesario cargar ningún fichero en la ejecución principal de Maude).

Instancia el objeto *pexpect* con el ejecutable de Maude y carga el fichero con el ejecutable de CafeInMaude (*cafeInMaude.Maude*)

A continuación, carga el fichero que contiene el histórico de comandos y ficheros cargados, y ejecuta el comando que ha pedido el usuario.

Si se alcanzara el tiempo límite de ejecución, devuelve un mensaje explicativo, y en caso contrario, devuelve la salida del comando.

4.2 Arquitectura del *Front-end*

En esta sección se ha optado por una interfaz principal común realizada con HTML, CSS y JavaScript, que irá cambiando según el usuario vaya interactuando con ella.

La primera diferencia que encontramos es a la hora de iniciar una sesión, como se aprecia en el *div* con id "modal_new_session", dentro de la carpeta *templates*. Los usuarios que quieran iniciar una nueva sesión se encontrarán con un seleccionable en el cual podrán elegir entre una sesión de CITP o una de CafeInMaude, si selecciona la opción de CITP tendrá la oportunidad de elegir si quiere subir algún archivo previo que se cargue al iniciar Maude a través de un input de tipo fichero.

Una vez se ha iniciado la sesión, mostramos un contenido u otro dependiendo del tipo de sesión elegido por el usuario. VueJS es un marco de JavaScript para la creación de interfaces de usuario, se basa en HTML, CSS y JavaScript estándar y ofrece un modelo de programación declarativo basado en componentes que ayuda a desarrollar con eficacia interfaces de usuario de cualquier complejidad.

La siguiente diferencia se muestra en el tipo de la sesión en el panel izquierdo de la web seguido del nombre de la misma sesión, como se muestra en la Ilustración 4-2 Lista de sesiones.

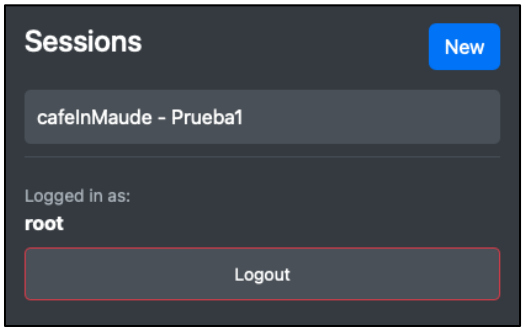

Ilustración 4-2 Lista de sesiones

Otro cambio apreciable sería en el *div* con id "div_chat_panel", a través del cual se puede seleccionar entre un número de ejemplos cargados en la web para poder trabajar sobre ellos o bien cargar archivos externos sobre los que interactuar. El usuario también escribe el comando que quiere introducir en Maude desde esta sección, viendo el resultado, si es que tuviese, en el propio chat.

Más cambios apreciables se visualizan en el panel derecho de la aplicación, en el que se ven distintas opciones en el caso de que la sesión sea de CITP (Ilustración 4-3 Panel de estado en sesión CITP) que en el caso de que sea de CafeInMaude (Ilustración 4-4 Panel de estado en sesión CafeInMaude).

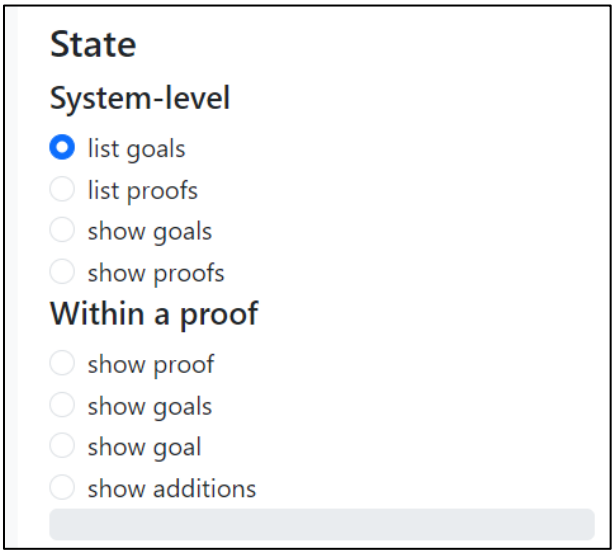

Ilustración 4-3 Panel de estado en sesión CITP

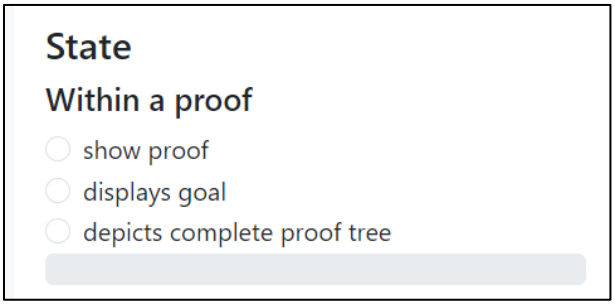

Ilustración 4-4 Panel de estado en sesión CafeInMaude

Cada una de estas opciones tiene una funcionalidad diferente dependiendo de las necesidades del usuario y de qué tipo de información quiere obtener respecto a la

prueba que está realizando y su resultado se muestra en un pequeño panel debajo de esta selección.

Toda esta información introducida en cada uno de los campos previamente mencionados se enviará posteriormente al *back-end* utilizando JaveScript para poder trabajar con ella y mostrar los resultados esperados por el usuario.

4.3 Problemas encontrados durante la implementación

Debido al conocimiento de la existencia de la biblioteca que contiene *bindings* para Python (Rubio, Maude as a Library: An Efficient All-Purpose Programming Interface, 2022), se ha optado por utilizar el lenguaje de Python para el desarrollo de la aplicación.

Tras realizar diversas pruebas utilizando la biblioteca que contiene los bindings, vemos que no es útil en este proyecto ya que solo está diseñada para ejecutar comandos de Maude más sencillos, y no permite la carga de las herramientas de CITP o CafeInMaude. Estas herramientas funcionan como una subejecución dentro de Maude a través del comando "load", cuyas salidas no son visibles para los bindings en Python.

Se han buscado alternativas para poder ejecutar las herramientas, y se ha encontrado que la forma más viable es que se ejecuten a través de la interfaz de línea de comandos de Maude utilizando un subproceso en Python.

Para llevar a cabo esto, se debe usar alguna biblioteca que permita gestionar los subprocesos, así como sus entradas y salidas. Más adelante se detallan las soluciones encontradas.

Capítulo 5 - Funcionalidad y guía de uso

En este apartado vamos a explicar la funcionalidad de la web de una forma visual para que sea lo más gráfica y fácil de entender que sea posible.

Diferenciaremos entre 2 portales, el de la aplicación denominada MaudeApp, que es el objeto principal de este proyecto, en la que los usuarios pueden interactuar con el intérprete, y otro para el administrador, en la ruta "/admin", en el que los usuarios con este privilegio pueden llevar a cabo tareas como la creación de usuarios o la subida de archivos de ejemplo para las demostraciones.

5.1 MaudeApp

Al acceder a la web, el usuario es redirigido a la pantalla de identificación, donde tendrá introducir su nombre de usuario y contraseña, como se muestra en la Ilustración 5-1 Pantalla de inicio de sesión.

En esta pantalla también se le da la opción al usuario de registrarse en el sistema haciendo clic en "Sign up". Este registro deberá ser aprobado por el administrador.

Ilustración 5-1 Pantalla de inicio de sesión

Una vez que el usuario pulsa el botón de "Sign in", si la identificación se produce de forma correcta, se le redirige a la página principal, como se muestra en lallustración 5-2 Pantalla principal.

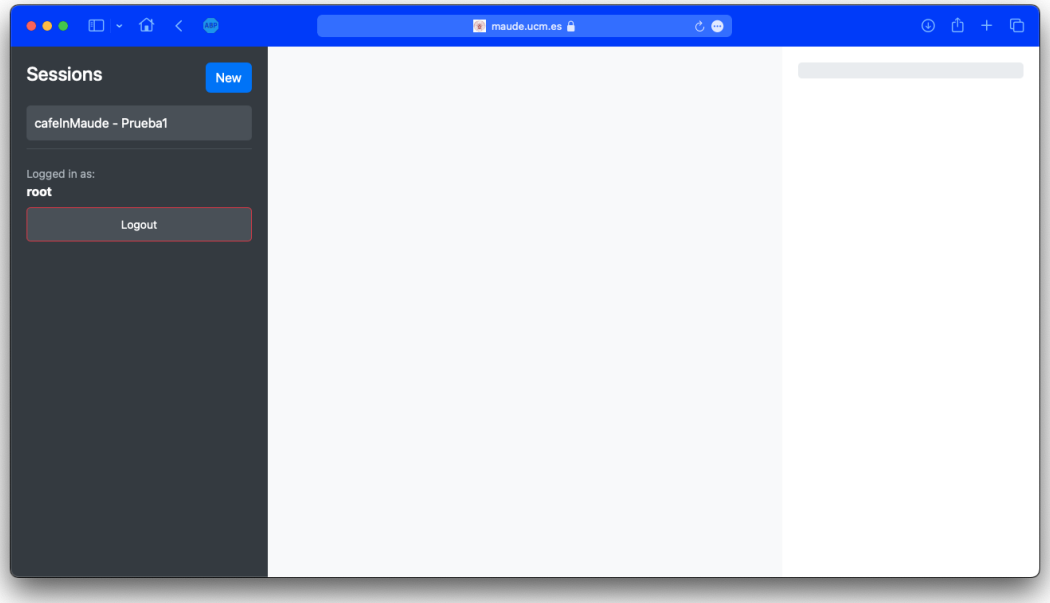

Ilustración 5-2 Pantalla principal

En esta pantalla también podrá cerrar sesión, siendo redirigido a la pantalla de identificación, a través del botón "Logout".

En la pantalla principal, el usuario puede crear una nueva sesión haciendo clic en el botón "New", con el que se despliega un diálogo modal en el que se debe elegir los parámetros de la nueva sesión (Ilustración 5-3 Diálogo modal de sesión CafeInMaude), eligiendo en el desplegable el tipo (CafeInMaude o CITP), y, en el caso de que sea CITP, podrá seleccionar archivos que se necesiten cargar previamente a iniciar la herramienta de Maude (Ilustración 5-4 Diálogo modal de sesión de CITP).

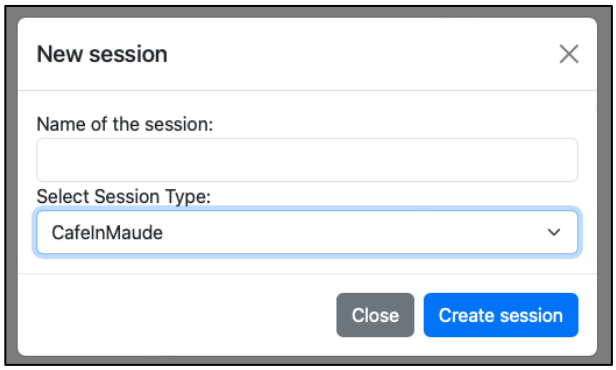

Ilustración 5-3 Diálogo modal de sesión CafeInMaude

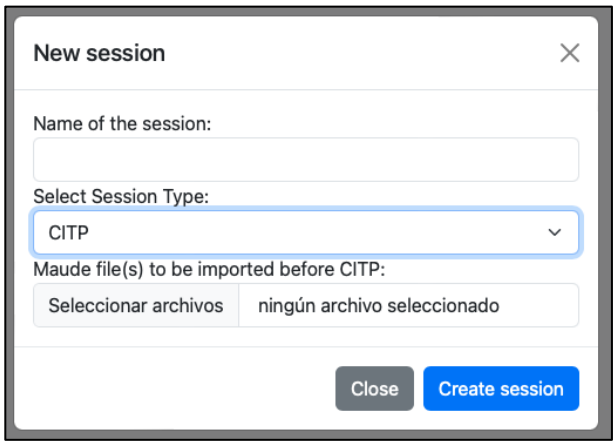

Ilustración 5-4 Diálogo modal de sesión de CITP

Se diferenciará a continuación entre los dos tipos de sesiones para las que se ha desarrollado este proyecto (CITP y CafeInMaude):

5.1.1 CafeInMaude

Comenzaremos explicando la funcionalidad de CafeInMaude. Una vez seleccionado el tipo de sesión y pulsado el botón de "Create session", aparecerá la pantalla del chat del intérprete.

En esta, en la parte inferior, se puede seleccionar un ejemplo entre una cantidad predeterminada de estos, se pueden introducir archivos externos en caso de desear realizar otro tipo de pruebas o directamente trabajar sobre las de los propios ejemplos ya mencionados como se puede ver en la Ilustración 5-5 Ventana del chat para CafeInMaude.

En este momento también se introducirá el comando que se desea ejecutar y, si fuera requerida, una de las posibles opciones del panel derecho que permiten mostrar el árbol de prueba en la que un objetivo se muestra con * si ya se ha demostrado y con > si es el actual; mostrar el objeto actual; o mostrar el árbol de pruebas completo, un objetivo se muestra con * si ya se ha probado y con > si es el actual.

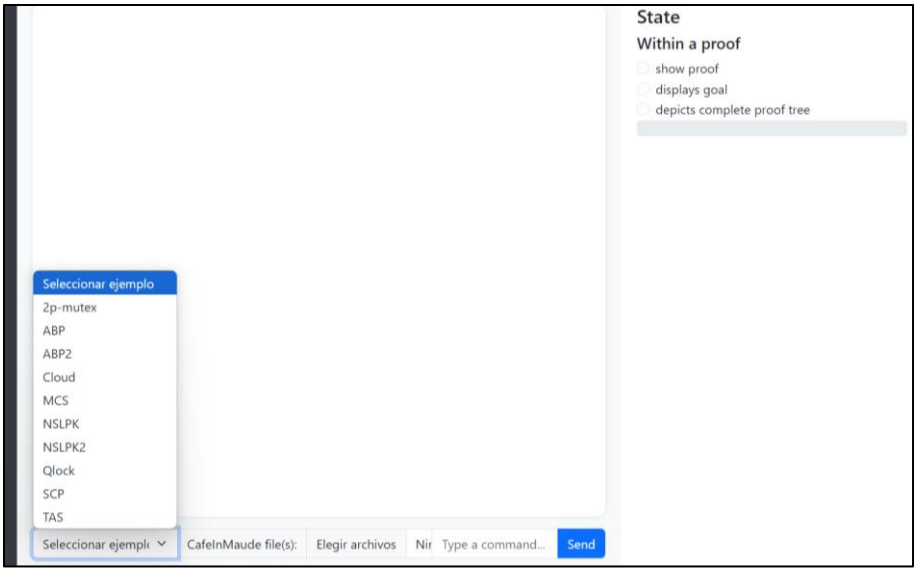

Ilustración 5-5 Ventana del chat para CafeInMaude

Para la prueba actual elegiremos el ejemplo predefinido "2p-mutex", cargaremos la prueba relacionada con este protocolo, denominada "all_proofs.cafe" inferiremos la prueba utilizando el comando ":infer-proof inv1 ." y en el panel derecho seleccionaremos la opción de "show proof" para ver tanto los resultados del comando en el chat, como las hojas del árbol de pruebas en el panel derecho (Ilustración 5-6 Selección de parámetros para prueba de CafeInMaude).

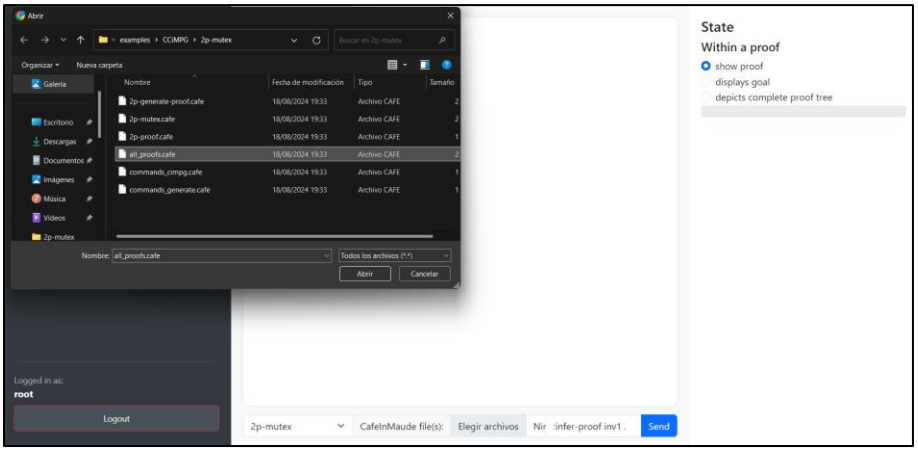

Ilustración 5-6 Selección de parámetros para prueba de CafeInMaude

Con esta selección el usuario obtendrá los resultados mostrados en la Ilustración 5-7 Resultados de prueba CafeInMaude.

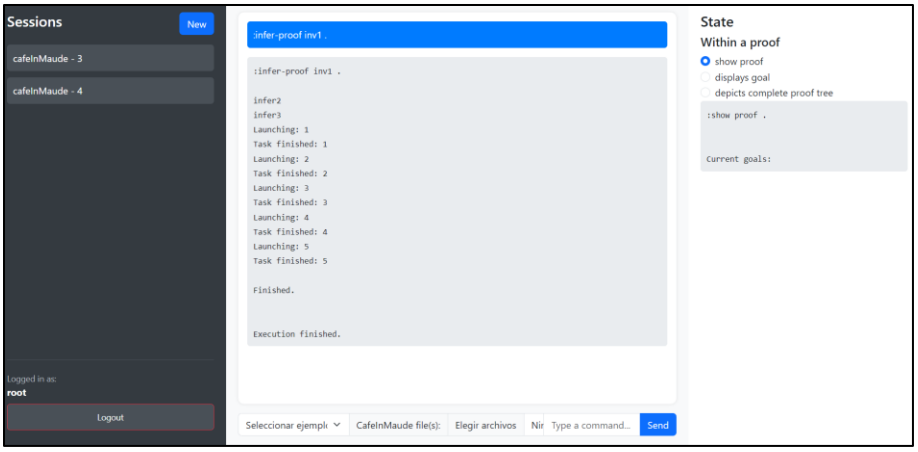

Ilustración 5-7 Resultados de prueba CafeInMaude

5.1.2 CITP

En el caso de CITP, una vez creada la sesión, aparecerá la pantalla del chat del intérprete de forma análoga a CafeInMaude, con un panel a la derecha en el cual se puede ver el estado de las pruebas actualizado después de la ejecución de cada comando.

Existen dos momentos diferenciados en la ejecución de CITP, el principal y en el que se encuentra dentro de una prueba (cuando se utiliza el comando "Begin proof"). El panel derecho muestra los comandos en 2 secciones para que el usuario elija uno u otro en función de este momento.

De la misma forma que en el caso de CafeInMaude, en la parte inferior se presenta un desplegable para seleccionar ejemplos precargados por el administrador, un selector de archivos para añadir a la prueba, y el campo de texto para la introducción de comandos, como se puede apreciar en la Ilustración 5-8 Pantalla de ejecución de CITP.

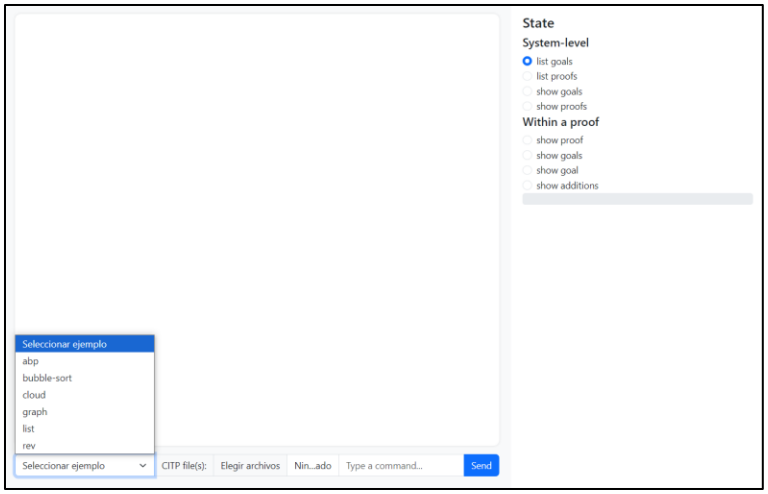

Ilustración 5-8 Pantalla de ejecución de CITP

5.2 Portal del administrador

Para las labores de administración contamos con este portal en la ruta "/admin" (Ilustración 5-9 Pantalla principal del administrador), en la cual los usuarios con este nivel de privilegio pueden, entre otros: subir archivos de ejemplo o eliminar algunos de los ya existentes, ver las sesiones activas y eliminar alguna si fuera necesario, así como modificar el tiempo límite de ejecución (predefinido inicialmente en 30 segundos). Esto último será necesario ya que hay pruebas que requieren un tiempo muy por encima de este valor.

Desde aquí el administrador también podrá gestionar los usuarios y grupos existentes y ver los comandos utilizados por sesión.

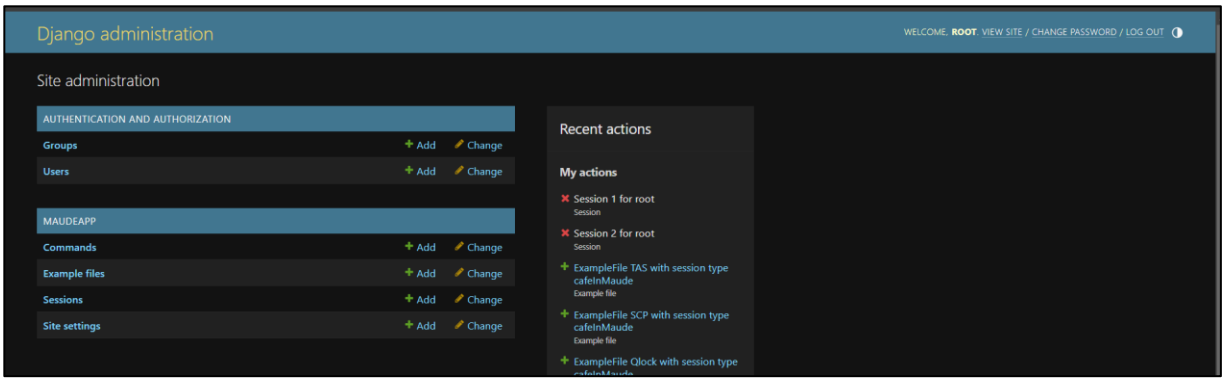

Ilustración 5-9 Pantalla principal del administrador

Desde el apartado de usuarios (*Users*), el administrador puede modificar, crear y eliminarlos. Además, si un usuario se registra a través de la web, el administrador podrá desde aquí autorizar el acceso (Ilustración 5-10 Gestión de usuarios del administrador).

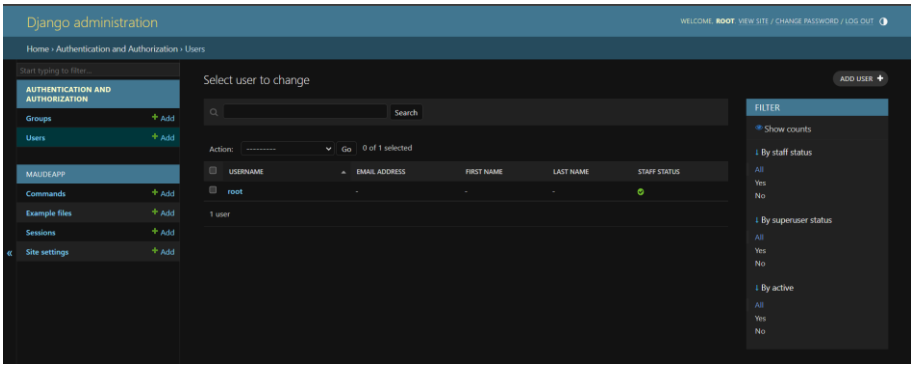

Ilustración 5-10 Gestión de usuarios del administrador

En el apartado de comandos (*Commands*), el administrador puede ver qué comandos han sido ejecutado por los usuarios, así como el momento en el que se ejecutaron (Ilustración 5-11 Histórico de comandos del administrador).

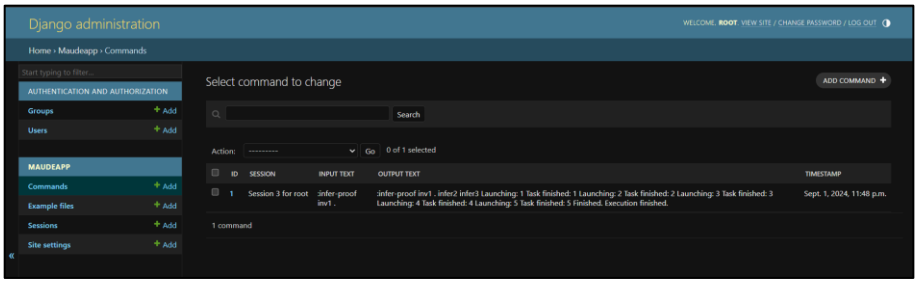

Ilustración 5-11 Histórico de comandos del administrador

Dentro del apartado de ficheros de ejemplo (*Example files*), se pueden subir ficheros de ejemplo para ambas herramientas, además de editar o eliminar los existentes. Estos podrán ser seleccionados individualmente por los usuarios en sus demostraciones.

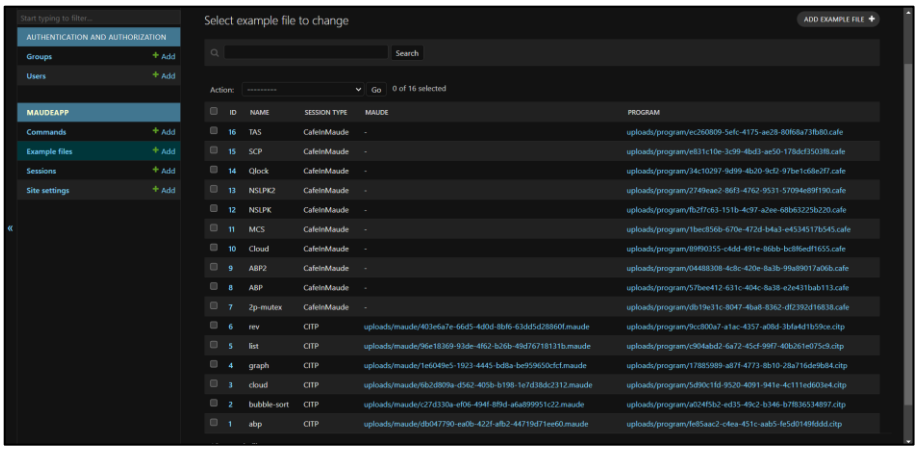

Ilustración 5-12 Gestión de ficheros de ejemplo del administrador

El apartado de sesiones (*Sessions*) muestra las sesiones creadas por los usuarios, así como su tipo y la fecha (Ilustración 5-13 Histórico de sesiones del administrador).

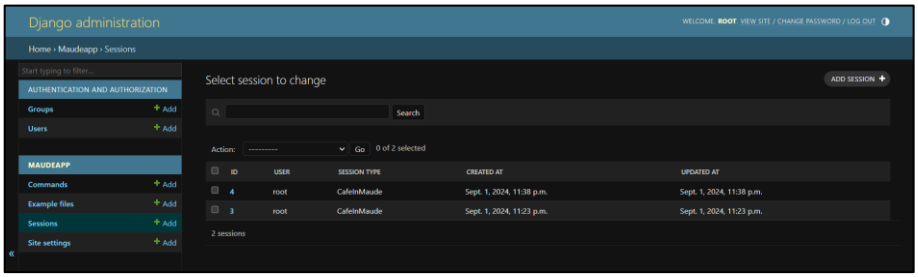

Ilustración 5-13 Histórico de sesiones del administrador

Desde el apartado de ajustes del sitio (*Site settings*), el administrador puede cambiar parámetros de funcionamiento. En este momento, solo se puede cambiar el tiempo límite de ejecución, que está definido por defecto en 30 segundos (Ilustración 5-14 Parámetro del sitio del administrador).

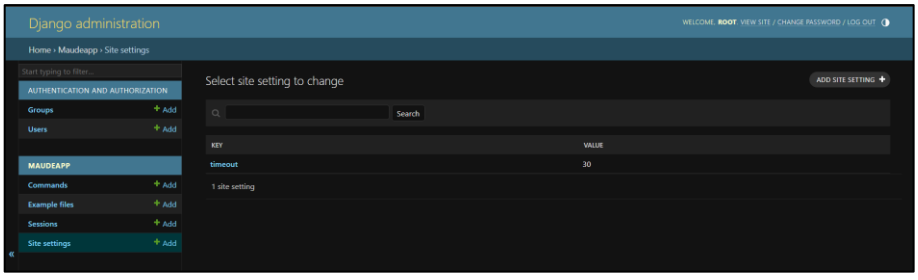

Ilustración 5-14 Parámetro del sitio del administrador

Conclusiones y trabajo futuro

En conclusión, en este proyecto se ha tenido el objetivo de diseñar una interfaz web que facilite el uso de las herramientas CITP y CafeInMaude, demostradores de teoremas implementados en Maude, haciendo que el sistema sea más accesible y fácil de usar.

Se ha propuesto el diseño y la implementación de una aplicación web que contenga una interfaz gráfica, utilizando las tecnologías más utilizadas hoy en día, que permita ejecutar las herramientas de CITP y CafeInMaude de forma intuitiva y desde cualquier navegador, a la vez que permita mantener un histórico de las sesiones para que el usuario las pueda volver a consultar o trabajar con ellas en ocasiones posteriores.

Hemos intentado que todas las tecnologías utilizadas sean parte del proyecto de Software Libre, y entre ellas:

- Python con Django en el *back-end* para gestionar la estructura del proyecto y gestionar la base de datos.
- Bootstrap y Vue.js en el *front-end* para la estructura de la interfaz de usuario y la gestión de datos en el navegador web, utilizando AJAX para la comunicación con el servidor.
- Docker para que el despliegue de la aplicación se realice de forma automática y sin apenas interacción por parte del usuario, además de garantizar la seguridad del servidor al crear un entorno contenido.
- GitHub para la gestión del repositorio, incluyendo GitHub Actions para la integración continua del proyecto.

El principal desafío que hemos encontrado a la hora del desarrollo de este proyecto ha sido encontrar la forma de integrar Maude y las herramientas de CITP y CafeInMaude dentro de un entorno web, superando las limitaciones de la biblioteca de Maude para Python, con la que contábamos al principio pero que resultó ser inadecuada para este proyecto, por lo que se tuvieron que encontrar otras alternativas.

Estas alternativas resultaron ser la emulación del proceso de consola para la ejecución de Maude desde Python, a través de módulos como *subprocess* y *pexpect*.

Además, otro desafío fue el de implementar medidas de seguridad que protegieran la integridad de los datos y el entorno del servidor. Esto último se ha conseguido al utilizar el entorno aislado de los contenedores Docker, en los que, si hubiera alguna vulneración de las medidas de seguridad, solo se vería afectado el entorno de la aplicación, y no el sistema del servidor.

Se ha conseguido desarrollar una aplicación funcional en la que los usuarios pueden interactuar con CITP y CafeInMaude, en el entorno de Maude, desde una web que es sencilla de utilizar y eficiente, garantizando la compatibilidad con todos los navegadores más comunes y todos los dispositivos que accedan a ella.

Trabajo futuro

Esta aplicación se ha diseñado para trabajar solo con CITP y CafeInMaude. En el futuro, se puede ampliar para trabajar con otras herramientas que funcionan con Maude, o con el propio entorno de Maude.

Para esto, será necesario refactorizar el código primero, de forma que se abstraiga y se separe la lógica de cada herramienta del funcionamiento de la web, facilitando la inclusión de nuevas herramientas.

Más concretamente dentro de la aplicación actual, y dado que, por ejemplo, en CafeInMaude existen pruebas que pueden durar varias horas, sería interesante que el usuario pueda cambiar el tiempo límite de ejecución para una prueba, solicitándolo primero al administrador.

Introduction

In the field of software development and system formalization, it is crucial to have languages that allow automatic property and specification demonstrations, such as the Maude language, which are, however, developed with a command line interface that is not accessible to all users.

Motivation

Among the various Maude applications are the CITP and CafeInMaude tools, which will be the focus of this Final Degree Project (TFG).

The Constructor-based Interactive Theorem Prover (CITP) is a tool for verifying properties of software systems.

CafeOBJ in Maude (CafeInMaude) is an interpreter that allows the combined implementation of CafeOBJ specifications and the ability to perform tests as previously mentioned. Through specifiers you can write test scores in CafeOBJ and test against them when you run them.

Maude is usually run through a command line interface (CLI), which, although it has certain advantages, has the disadvantage of making it difficult to use for a large number of users, with the added disadvantage that it is mainly designed for UNIX systems, and can only be used by users of Windows, the most widely used operating system globally (Desktop Operating System Market Share Worldwide, 2020), using Linux virtualisation tools.

In the context of the project developed in this TFG, it is not relevant to describe in depth the functioning of the Maude language and the CITP and CafeInMaude tools, beyond the fact that they are demonstrators of properties implemented in Maude, as we will only focus on the fact that these tools have the same functionalities through the web as in the CLI.

You can learn more about the Maude language in All About Maude - A High-Performance Logical Framework (Manuel Clavel, 2007)

Goals

The main purpose of this TFG is to design and develop a tool that allows the execution of some tools such as CITP and CafeInMaude through Maude, in a simple way from any web browser.

The application must have the following characteristics:

1. Broad compatibility with current browsers: It should be accessible from any web browser in compliance with current web browser standards.
2. Responsive design: It must be designed to be accessible from any device regardless of the size of the screen and the contents can be read correctly, in other words, a responsive application.
3. User-friendly and easy-to-use interface: The application should have an intuitive interface, so that it can be operated without any prior training of the user.
4. Security and privacy: You must ensure the security of any sensitive data you hold, right from the design.
5. Upgradability and scalability: The application should be designed modularly, so that it is easy to fix and, if a new module or functionality needs to be added, this should be achieved without redesigning the whole application.

A GitHub repository has been created for version control and publication of the application project, which can be found at the following web address:

https://github.com/gonzalovlchz/WebMaude

The developed application will be hosted on a server that will be accessible through the following link:

https://maude.ucm.es/webmaude/

Work plan

As can be seen in Illustration 5-15 Gantt diagram, the first two weeks were dedicated to the planning phase. In the first stage of this phase, which lasted one week, we both dedicated ourselves to researching the Maude language and, individually, the

corresponding tools for each of us, with the aim of gathering all the information possible and necessary to understand the language and its tools and thus be able to correctly carry out the project we were assigned. During the second stage of this phase, we divided the work so that one of us would define the system requirements and the other would draw up the project plan and a detailed schedule.

The design phase was also divided into two stages, where during the first week and a half one of us had to design the web interface architecture and the other the user interface design. During the second week and a half of this stage, one of us had to design the database, considering all the tables necessary for the interface to function. During this period, the other one had to elaborate the flowcharts and use cases.

In the development phase, which is the longest with a total duration of 6 weeks, during the first two weeks one of us focused on configuring the development environment and creating the basic structure of the web interface, while the other one dedicated that time to the initial implementation of the database, building the tables, previously thought, that we considered necessary, as later on we realized that some new table, row or column had to be added for the correct functioning of the implementation.

The next two weeks were devoted to front-end development, using HTML, CSS and JavaScript code, and back-end development to connect to Maude through pexpect. The last two weeks were used to integrate the front-end code with the back-end code and to implement the specific functionalities for each of the tools (CafeInMaude and CITP).

The following weeks are dedicated to the testing phase in which the first week and a half are dedicated to user interface and performance tests to verify the correct functioning of the entire website and to detect errors and fix them before moving on to the next phase.

During the last phase, the documentation phase, we focused on writing the technical documentation for the project, creating clear and easy-to-understand user manuals for the correct use of the website and documenting the database so that the tutor could give us indications of possible changes or advice on how to improve it.

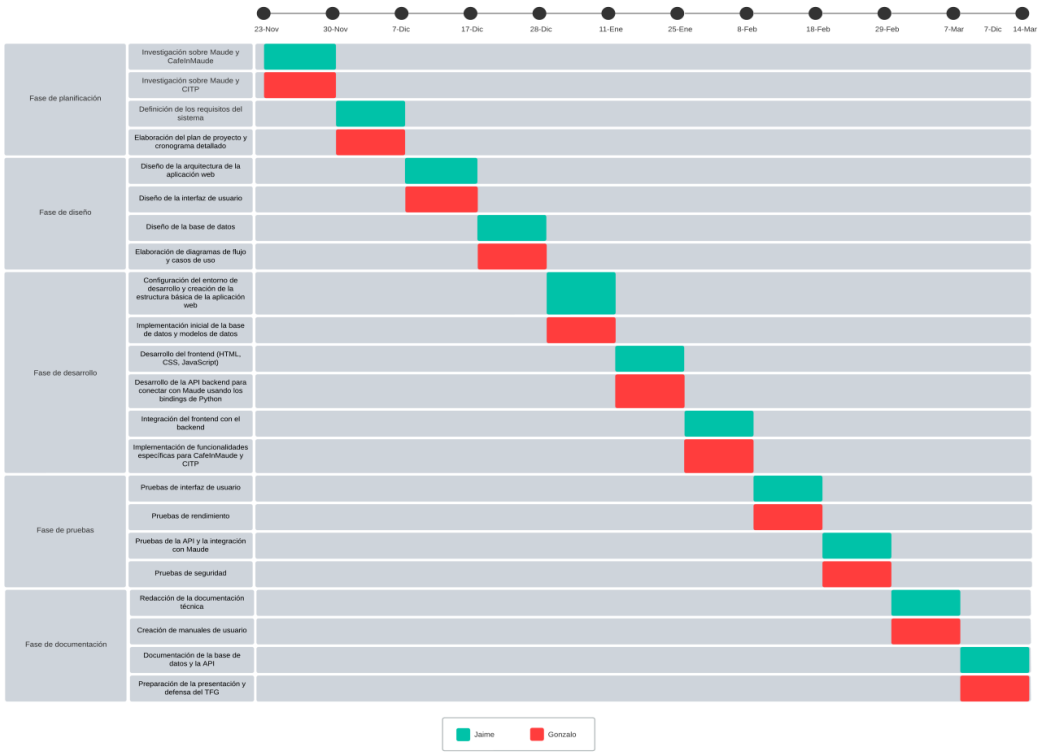

Illustration 5-15 Gantt diagram

Conclusions and future work

In conclusion, the aim of this project has been to design a web interface that facilitates the use of the CITP and CafeInMaude tools, theorem provers implemented in Maude, making the system more accessible and easier to use.

It has been proposed the design and implementation of a web application that contains a graphic interface, using the technologies most used today, which allows the CITP and CafeInMaude tools to be run intuitively and from any browser, while at the same time allowing a history of the sessions to be kept so that the user can consult or work with them again on subsequent occasions.

We have tried to make all the technologies used part of the Free Software project, and among them:

- Python with Django in the back-end to manage the project structure and manage the database.
- Bootstrap and Vue.js on the front-end for user interface structure and data management in the web browser, using AJAX for communication with the server.
- Docker to deploy the application automatically and with little user interaction, as well as ensuring the security of the server by creating a contained environment.
- GitHub para la gestión del repositorio, incluyendo GitHub Actions para la integración continua del proyecto.

The main challenge we have found when developing this project has been to find a way to integrate Maude and the CITP and CafeInMaude tools within a web environment, overcoming the limitations of the Maude library for Python, which we had at the beginning, but which turned out to be inadequate for this project, so other alternatives had to be found.

These alternatives turned out to be the emulation of the console process for running Maude from Python, through modules such as subprocess and pexpect.

In addition, another challenge was to implement security measures to protect the integrity of the data and the server environment. The latter has been achieved by using the isolated environment of Docker containers, where, if there were any breach of security measures, only the application environment would be affected, and not the server system.

We have managed to develop a functional application in which users can interact with CITP and CafeInMaude, in the Maude environment, from a website that is simple to use and efficient, guaranteeing compatibility with all the most common browsers and all the devices that access it.

Future work

This application is designed to work only with CITP and CafeInMaude. In the future, it can be extended to work with other tools that work with Maude, or with the Maude environment itself.

For this, it will be necessary to refactor the code first, so that the logic of each tool is abstracted and separated from the functioning of the website, facilitating the inclusion of new tools.

More specifically within the current application, and given that, for example, in CafeInMaude there are tests that can last several hours, it would be interesting if the user could change the execution time limit for a test, by first requesting it to the administrator.

CONTRIBUCIONES PERSONALES

Gonzalo Vílchez Rodríguez

Cuando empezamos a trabajar en este proyecto, dividimos el trabajo entre las dos herramientas para las que íbamos a realizar esta interfaz (CITP y CafeInMaude), y decidimos que yo me encargar de investigar y trabajar con CITP.

A la hora de diseñar y conceptualizar el proyecto, estuve investigando sobre los bindings de Maude para Python, y cómo podrían ser útiles con CITP. Tras diversas pruebas, llegué a la conclusión de que estos no iban a ser de utilidad en mi parte del proyecto, al menos.

Después de esto, empecé a buscar alternativas que fueran útiles para esto, y estuve haciendo pruebas con subprocess. Después de estas pruebas, viendo que la ejecución de la aplicación siempre se quedaba bloqueada, estuve investigando el por qué, y vi que era porque la entrada y la salida del objeto de subprocess van por tuberías distintas, por lo que no es posible saber cuándo un programa solicita la interacción del usuario con un prompt de forma sincronizada con la entrada.

Ya que subprocess no iba a poder ser una de las opciones, o si lo era iba a tener que ser de una forma más compleja, estuve buscando otra opción que fuera la más utilizada en estos casos, en aplicaciones en las que se espera que el usuario introduzca información. Así encontré la biblioteca de pexpect, y tras diversas pruebas con el intérprete de Maude por sí solo y con CITP, vi que podía ser útil, ya que siempre se sabe cómo el programa pide al usuario información a través de un prompt conocido.

Una vez que ese problema estuvo solucionado, empecé a desarrollar la web, leyendo sobre Django y sus posibilidades, y siguiendo un tutorial básico para la creación de una web simple, que serviría de base para la aplicación que hemos desarrollado finalmente.

Empecé diseñando la aplicación de forma local, primero focalizando en crear una estructura que fuera a facilitar la creación del proyecto de forma más modular. Empecé desarrollando la aplicación con una base de datos de MariaDB. Poco después, investigué sobre la posibilidad de creación de un contenedor, y, con esto, cambié la

base de datos a SQLite, para que la instalación fuera más sencilla y estuviera contenida dentro de la aplicación.

Una vez diseñado el contenedor y cómo se iba a construir, investigando si era mejor utilizar una base de Python o un Linux, siendo más conveniente esta última, diseñé un punto de entrada a la aplicación que ha ido cambiando según las necesidades del proyecto, ya que, al desarrollar la aplicación principalmente en un equipo de Windows, era necesario compilar el contenedor tras cada modificación cada vez que era necesario realizar una prueba del funcionamiento.

Utilicé GitHub Actions para establecer un flujo de integración continua en el que, cada vez que se hiciera "push" en la rama principal, se construyera el contenedor y se subiera al repositorio.

Posteriormente, utilizando Bootstrap, diseñé una interfaz tipo chat, inspirándome en otras aplicaciones de chats conocidas como WhatsApp o ChatGPT, también utilizando Vue.js para facilitar la constancia de los datos. Incluí en la vista de chats la integración con CITP, además de la creación de sesiones.

También me encargué de diseñar el modelo de la aplicación, creando las figuras de sesiones y comandos, archivos de ejemplo y parámetro de la aplicación.

Con respecto a la memoria, me he encargado de la introducción y los antecedentes.

Posteriormente, redacté los capítulos de tecnologías utilizadas, a excepción de la parte referente a CafeInMaude.

De la sección de Arquitectura, me he encargado de redactar lo referente al backend.

En el capítulo de funcionalidad y guía de uso, solo me he encargado de redacción y la búsqueda de figuras de la parte de CITP.

Por último, me he encargado de redactar las conclusiones del proyecto y la sección de trabajo futuro.

Jaime Lozano Díaz

Cuando hicimos el reparto del trabajo a mí se me asignó la parte relacionada con el módulo de CafeInMaude.

Las primeras semanas me dedique a investigar la funcionalidad de este y a documentarme lo máximo posible para el trabajo designado. Encontré bastante información en artículos que me suministro el tutor, este también me facilitó el enlace de un repositorio GitHub con todo lo necesario para poder trabajar con CafeInMaude.

También tuve que investigar acerca de los bindings de Python y como poder trabajar con ellos.

Al comienzo intente desarrollar mi parte basándome en ellos sin mucho éxito, solo lograba mandar el comando, pero no era capaz de obtener una respuesta de maude. Más adelante mi compañero me comento que hablando con el tutor, éste le había comentado que era mejor utilizar subprocess para poder introducir texto en la sesión interactiva pudiendo así escribir en la entrada estándar del programa de Python.

Con esta información decidí usar pexpect, recomendada por mi compañero que ya había tenido éxito en su parte. Pexpect permite que su script genere una aplicación secundaria y la controle como si un humano estuviera escribiendo comandos, pero solo funciona en Linux. Para poder trabajar en mi máquina Windows tuve que optar por usar wexpect, que es lo mismo, pero para mí sistema operativo.

Hablando con el tutor sobre la decisión de usar wexpect me comento que al trabajar él con Linux era mejor que buscase otra forma de hacerlo para poder aparte unificar código con el de mi compañero. De esta manera decidí hacer uso de la herramienta WSL que provee una interfaz que simula un kernel de Linux (sin contener código de Linux propiamente dicho), el cual puede ejecutar aplicaciones de espacio de usuario GNU, por ejemplo, una instalación base de Ubuntu, Debian y Kali Linux entre otros. Con esta herramienta ya sí que note avances significativos pudiendo arrancar Maude, pasarle ciertos comandos tal cual y recibiendo una respuesta; todo esto a través de pexpect, lo que me permitió unificar mi parte de funcionalidad con el código principal.

Más adelante nos decantamos por utilizar un Dockerfile por lo que tuve que añadir las carpetas de trabajo con las que funciona el módulo de CafeInMaude a las variables de entorno de este docker.

Incorpore a la implementación que a la hora de seleccionar una nueva sesión se pudiera elegir entre una del tipo CITP o una del tipo CafeInMaude y que dependiendo de la sesión seleccionada mostrase el contenido especifico de CITP, realizado por mi compañero, o el relacionado con mi sesión. También añadí al models.py la funcionalidad necesaria para el correcto funcionamiento de mi sesión, modifiqué el chat_views.py con todo lo necesario para poder trabajar el módulo de CafeInMaude apoyándome en el código desarrollado por mi compañero y añadí la definición de executeCafeInMaudeCommand() que es donde esta implementada la funcionalidad para por medio de pexpect conectar con Maude.

Incorporé todo el contenido del repositorio GitHub con todo lo necesario para el correcto funcionamiento del módulo de CafeInMaude, suministrado por mi profesor como comenté anteriormente y añadí a la parte del administrado los archivos necesarios para poder mostrar y cargar los ejemplos predefinidos de este módulo en nuestra web.

Con respecto a la memoria, me encargué de redactar el resumen que aparece en la primera sección de la memoria. Realice el diagrama de Gantt del apartado de Plan de trabajo y su correspondiente redacción.

También redacté el apartado de Estado de la cuestión, el cual es el apartado dos de esta memoria.

Me encargué de la redacción de Funcionalidad y guía de uso excluyendo el apartado en el que se explica la funcionalidad de CITP, que realizó mi compañero.

En lo que corresponde al apartado de Arquitectura, realice la redacción del apartado relacionado con el Front-end.

BIBLIOGRAFÍA

Chacon, S. a. (2014). *Pro Git.* Apress.

Desktop Operating System Market Share Worldwide. (27 de 12 de 2020). Obtenido de StatCounter Global Stats: https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-202208-202306

Django overview. (s.f.). Obtenido de https://www.djangoproject.com/start/overview/

Grant Allen, M. O. (2010). *The Definitive Guide to SQLite.* Apress.

Manuel Clavel, F. D.-O. (2007). *All About Maude - A High-Performance Logical Framework.* Springer Berlin, Heidelberg.

Pexpect. (2013). Obtenido de https://pexpect.readthedocs.io/en/stable/

Rubio, R. (2022). Maude as a Library: An Efficient All-Purpose Programming Interface. En *Rewriting Logic and Its Applications - 14th International Workshop* (págs. 274-294). Springer.

Sanjeev Jaiswal, R. K. (2015). *Learning Django Web Development: From Idea to Prototype, a Learner's Guide for Web Development with the Django Application Framework.* Packt Publishing.

subprocess — Subprocess management. (s.f.). Obtenido de https://docs.python.org/3/library/subprocess.html

Turnbull, J. (2019). *The Docker Book.*