UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

Análisis e implementación de un sistema de generación automática de horarios mediante inteligencia artificial

Diseño e implementación de una aplicación web interactiva capaz de atender a preferencias y restricciones para la posterior generación automática de los horarios mediante heurísticas.

Autor
Diego Barreiro Pérez

Tutores
Alberto José Bugarín Diz
Juan Carlos Vidal Aguiar

Grado en Ingeniería Informática

Julio 2023

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática.

Agradecimientos

Me gustaría agradecer a mis tutores, Alberto Bugarín y Juan Vidal, la oportunidad brindada para realizar este Trabajo con ellos. Desde que les mencioné la posibilidad de hacer un programa para la gestión de horarios la han estado apoyando dando ideas y modelándolo hasta lo que es ahora (ya que ha habido bastantes cambios desde la idea original y el trabajo final). La libertad y confianza que han estado aportando ha sido muy apreciada, lo que me ha permitido desarrollar toda la aplicación de la mejor manera posible.

También me gustaría apreciar el apoyo dado por parte de mi familia durante toda la carrera. Siempre me han amparado a la hora de elegir la carrera que siempre me gustó desde hace años, y me han permitido vivir experiencias estudiando en el extranjero.

Además, agradecer la oportunidad que tuve al realizar las prácticas en Amazon Dublín, las cuales me permitieron aprender muchas de las tecnologías utilizadas en este proyecto. El poder tener una experiencia laboral, y además en el extranjero, me abrió muchas puertas para mi formación y carrera laboral.

También quiero agradecer a la organización de Software Freedom Conservancy por colaborar en el proyecto Git y permitir ejecutar el comando `git commit --amend -m "Quick Fixes"`, el cual ha sido utilizado en múltiples ocasiones durante el desarrollo de este Trabajo de Fin de Grado.

Y como mención especial, quisiera comentar que gracias a la realización de este Trabajo me ha sido posible contribuir a otros proyectos mediante solicitudes de cambio que se mencionan en el apéndice D.

Resumen

La planificación horaria siempre ha sido un problema complejo, y más todavía a la hora de programar horarios en las universidades. A diferencia de los institutos u otros centros más pequeños, en las universidades existen diferentes centros (escuelas y facultades) gestionados independientemente con numerosos tipos de restricciones que se deben atender. En consecuencia, los horarios de la mayoría de facultades son realizados manualmente, por lo que el resultado es, en general, mejorable de cara a estudiantes, docentes u ocupación de espacios. Véase el ejemplo de la *Escuela Técnica Superior de Ingeniería*, donde en algún caso se tiene clase mañanas y tardes todos los días de la semana, mientras que en otras facultades tan sólo se tiene o por las mañanas o por las tardes. De aquí surge la principal motivación del trabajo: **mejorar la planificación horaria mediante algoritmos**. Sin embargo, la mayoría de herramientas que permiten resolver estos problemas de optimización no son intuitivas y requieren de formación matemática y/o informática, lo cual dificulta su uso por parte del personal de gestión de los centros. Ergo, es necesario desarrollar una herramienta que no solo genere los horarios, sino que también sea intuitiva de cara a los usuarios finales.

Por ello, el principal objetivo de este Trabajo de Fin de Grado es la **creación de una plataforma web interactiva para la generación automática de horarios** en base a ciertas preferencias y restricciones aplicadas al conjunto de "entidades" (facultades, estudios, espacios, etc.) que participan en este problema. La generación de estos horarios se realizará a nivel de centro (facultad o escuela), de tal forma que como centro independiente sea capaz de definir tales entidades bajo su alcance. El resultado ha sido el despliegue satisfactorio de la plataforma, cuya instancia pública es accesible a través de la URL `https://horarios.barreiro.xyz/`, la cual permite comprobar el correcto funcionamiento de la aplicación.

Índice general

1. Introducción **1**
- 1.1. Motivación . . . 1
- 1.2. Objetivos . . . 2
  - 1.2.1. Estado de la tecnología . . . 3
- 1.3. Estructura de la memoria . . . 4
- 1.4. Descripción del sistema . . . 4
  - 1.4.1. Métodos y tecnologías utilizados . . . 5
  - 1.4.2. Metodología de desarrollo . . . 5
  - 1.4.3. Limitación del alcance . . . 6
- 1.5. Modelos y resultados de ejemplo . . . 7

2. Especificación de requisitos **8**
- 2.1. Glosario . . . 8
- 2.2. Requisitos de la plataforma . . . 9
  - 2.2.1. Requisitos de información . . . 10
  - 2.2.2. Requisitos funcionales . . . 10
  - 2.2.3. Requisitos no funcionales . . . 11
- 2.3. Requisitos del planificador . . . 12
  - 2.3.1. Requisitos funcionales . . . 12
  - 2.3.2. Requisitos no funcionales . . . 13
- 2.4. Objetivos del proyecto . . . 13
  - 2.4.1. Restricciones . . . 16

3. Diseño e implementación **17**
- 3.1. Arquitectura de la infraestructura . . . 17
  - 3.1.1. Autómata de estados del planificador . . . 20
- 3.2. Modelado de la plataforma . . . 21
  - 3.2.1. Modelos de datos . . . 21
  - 3.2.2. Diseño de la interfaz . . . 25
- 3.3. Ejecuciones del planificador . . . 28
  - 3.3.1. Creación de la ejecución . . . 29
  - 3.3.2. Invocación del planificador . . . 30
- 3.4. Modelado lógico del problema . . . 30

3.4.1. Responsabilidad código-algoritmo . . . 31
3.4.2. Declaración de *hechos* . . . 32
3.4.3. Declaración de *generadores* . . . 34
3.4.4. Declaración de *normales* . . . 35
3.4.5. Declaración de *restricciones* . . . 35
3.4.6. Declaración de *optimizaciones* . . . 36
3.4.7. Declaración de *directivas* . . . 37

4. Pruebas **38**
4.1. Pruebas de la plataforma . . . 38
4.1.1. Pruebas funcionales . . . 39
4.1.2. Validación de la interfaz . . . 39
4.1.3. Pruebas de la configuración . . . 41
4.2. Pruebas del planificador . . . 41
4.2.1. Validaciones de la generación de horarios . . . 42
4.3. Pruebas de integración . . . 43
4.3.1. Caso de uso real . . . 43
4.4. Cumplimiento con los objetivos . . . 45

5. Conclusiones **48**
5.1. Posibles ampliaciones . . . 48
5.2. Lecciones aprendidas . . . 50

A. Manuales técnicos **51**
A.1. Código fuente del proyecto . . . 51
A.2. Desarrollo en entorno local . . . 51
A.3. Despliegue de la aplicación . . . 52

B. Manuales de usuario **54**
B.1. Uso de la plataforma web . . . 54
B.1.1. Configuración de preferencias y restricciones . . . 54
B.1.2. Visualización de resultados . . . 59
B.2. Uso del planificador . . . 59
B.2.1. Línea de comandos `cli` . . . 59
B.2.2. Línea de comandos `manual_execution` . . . 60

C. Licencia **63**

D. Otras contribuciones **64**

E. Glosario de AWS **65**

F. Requisitos y pruebas detallados **67**
F.1. Requisitos de la plataforma 67
F.1.1. Requisitos de información 67
F.1.2. Requisitos funcionales de la interfaz 68
F.1.3. Requisitos no funcionales 69
F.2. Requisitos del planificador 70
F.2.1. Requisitos funcionales 70
F.2.2. Restricciones y preferencias 71
F.2.3. Requisitos no funcionales 72
F.3. Pruebas de la plataforma 73
F.3.1. Pruebas funcionales 73
F.3.2. Resultados del cuestionario SUS 75
F.3.3. Pruebas de la configuración 78
F.4. Pruebas del planificador 81
F.4.1. Pruebas funcionales 81
F.4.2. Validaciones de la generación de horarios 82

G. Formatos de archivos **87**
G.1. Archivos usados por la plataforma 87
G.2. Archivos usados por el planificador 91
G.2.1. Archivos de datos adicionales 93

Bibliografía **97**

Índice de figuras

1.1. Horario de un grupo del Grado de Ingeniería Informática 2
1.2. Ejemplo de visualización de la salida del planificador 7

2.1. Relación entre las entidades involucradas 9

3.1. Diagrama de la infraestructura *serverless* 17
3.2. Autómata de estados del planificador 20
3.3. Interfaz de configuración del planificador 25
3.4. Interfaz de resultado del planificador 27
3.5. Diálogo de detalles de una sesión, y horario de ocupación por aulas 28
3.6. Proceso de invocación del planificador desde la plataforma 28

4.1. Resultado de una ejecución para el *Grado de Ingeniería Informática* 45
4.2. Resultado de una ejecución para el *Grado de Inteligencia Artificial* 46

B.1. Opciones para "Ajustes del Entorno de Ejecución" 56
B.2. Opciones para "Ajustes Básicos" 56
B.3. Opciones para "Preferencia global de Intervalos" 56
B.4. Opciones para "Tipos de Sesión permitidos por Aula" 57
B.5. Opciones para "Modo de planificación para Programas" 57
B.6. Opciones para "Intervalos preferidos para Tipos de Sesión por Programa" 58
B.7. Opciones para "Preferencias de Aula por Programa" 58

F.1. Distribución de los resultados del cuestionario SUS 78

Índice de cuadros

2.1. Requisitos funcionales del planificador en la plataforma 11
2.2. Objetivo `OBJ-001`: interfaz gráfica para las ejecuciones 14
2.3. Objetivo `OBJ-002`: interfaz gráfica para las preferencias 14
2.4. Objetivo `OBJ-003`: ejecución del planificador 15
2.5. Objetivo `OBJ-004`: visualización de resultados 15
2.6. Objetivo `OBJ-005`: escalabilidad y disponibilidad 16

3.1. Modelo de ejecuciones del planificador 24

F.1. Requisitos de información de la plataforma 67
F.2. Requisitos funcionales de la plataforma 68
F.3. Requisitos no funcionales de la plataforma 69
F.4. Requisitos funcionales del planificador 70
F.5. Requisitos funcionales de restricciones y preferencias del planificador 71
F.6. Requisitos no funcionales del planificador 72
F.7. Pruebas funcionales de la plataforma 73
F.8. Resultados del cuestionario SUS 76
F.9. Pruebas funcionales del planificador 81
F.10. Validación de restricciones y preferencias del planificador 82

Índice de bloques de código

3.1. Ejemplo de sintaxis de Clingo 30
3.2. Lista de hechos definidos en Clingo 32
3.3. Lista de generadores definidos en Clingo 34
3.4. Lista de normales definidas en Clingo 35
3.5. Lista de restricciones definidas en Clingo 35
3.6. Lista de optimizaciones definidas en Clingo 36
3.7. Lista de directivas definidas en Clingo 37

A.1. Compilación de `models` 52
A.2. Compilación de `backend` 52
A.3. Compilación de `frontend` 52
A.4. Compilación de `solvers-asp` 53
A.5. Compilación de `cdk` 53

B.1. Ejemplos de comandos invocables mediante `cli` 60
B.2. Ejemplos de comandos invocables mediante `manual_execution` . . 61

G.1. Ejemplo de archivo `input_config.json` 87
G.2. Ejemplo de archivo `input_courses.json` 88
G.3. Ejemplo de archivo `input_rooms.json` 89
G.4. Ejemplo de archivo `output_week.json` 90
G.5. Ejemplo de archivo `input.json` 91
G.6. Ejemplo de archivo `output.json` 93
G.7. Ejemplo de archivo `asp_optimization.json` 94
G.8. Ejemplo de archivo `asp_problem.json` 94
G.9. Ejemplo de archivo `asp_solution.json` 95
G.10. Ejemplo de archivo `asp_statistics.json` 95
G.11. Ejemplo de archivo `asp_status.json` 96

Capítulo 1

Introducción

1.1. Motivación

El ser humano y el tiempo siempre han tenido una relación complicada a lo largo de su historia. El "querer hacer cosas" siempre ha estado ligado a la dependencia de la disponibilidad del tiempo necesario y de una ubicación espacial concreta, lo que lleva a la fórmula perfecta para complicar el "querer hacer cosas": **eventos** que han de ser realizados en cierto **tiempo** en un **espacio** concreto.

Pero estas dos restricciones no son por lo general inamovibles: suele haber relación de *dependencia condicional* en la que se han de tener en cuenta otras múltiples restricciones. Y es ahí donde surge el problema de la **programación horaria**: el ser capaz de programar en una franja temporal una actividad en una ubicación concreta de acuerdo a las restricciones (o preferencias) dadas.

El ejemplo más claro se encuentra en el ámbito académico: los **horarios de las clases**. Las instituciones académicas (ya sean colegios, institutos u otros) suelen tener que programar las clases de acuerdo a una gran cantidad de restricciones, tanto temporales como espaciales. Por ejemplo, el no solapar clases de un mismo curso, o el no programar dos sesiones en la misma clase al mismo tiempo.

Esto *a priori* puede parecer una tarea sencilla, pero cuanto más compleja es la institución, más restricciones aparecen. Y en el caso de las universidades estas restricciones son de una enorme complejidad al tener, por ejemplo, aulas dedicadas a tipos de clase concretos, o incluso edificios enteros asignados a ciertos grados. Esto hace que, a veces, los horarios creados manualmente no puedan tener en cuenta la solución más eficiente o, idealmente, la óptima y, por tanto, los horarios generados puedan ser mejorables. Véase el ejemplo en la figura 1.1, en la que se muestra el horario real del Grupo 2 de 3er curso durante el 2^{o} cuatrimestre [BP21]. Tal como se aprecia, existen huecos sin docencia entre clases, y los horarios de cada día son muy irregulares, lo que dificulta la compatibilidad con otras actividades que tengan un horario más uniforme.

El ser capaces de tener una **herramienta que**, dadas unas restricciones o

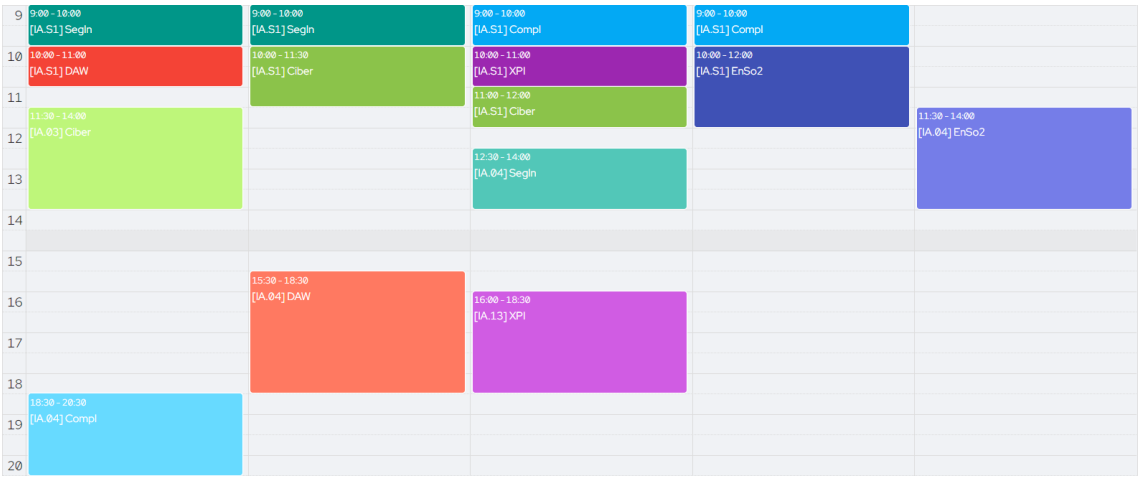

Figura 1.1: Horario de un grupo del Grado de Ingeniería Informática

preferencias, **genere los horarios más óptimos solucionaría este problema**, y sería capaz de, tan solo cambiando unos parámetros, **generar otro horario más óptimo ante la menor leve modificación**. Se ha de tener en cuenta la posibilidad de restricciones externas a respetar, por lo que es posible que no todas las "irregularidades" se eliminen.

1.2. Objetivos

El principal objetivo de este trabajo es el analizar, diseñar, implementar, probar y desplegar una **plataforma que permita realizar la generación de horarios mediante inteligencia artificial**. El caso de uso a resolver será, además, la generación de horarios de la propia *Escuela Técnica Superior de Ingeniería* y tratar de resolver algunos de los problemas mencionados anteriormente. Por ello, será necesario que la plataforma cuente con la mayor versatilidad posible, para atender a las diversas restricciones y preferencias que puedan existir para este (y otros) casos de uso.

Esto todo se podría resumir en los siguientes objetivos secundarios:

- Identificar y modelar las entidades asociadas al problema de generación de horarios (asignaturas y aulas, junto con sus dependencias a facultades, edificios, grados, etc.).
- Identificar y modelar las restricciones del problema. Estas son restricciones que pueden ser tanto duras (por ejemplo, imposibilidad de programar dos clases en la misma aula al mismo tiempo), como blandas (por ejemplo, preferir clases expositivas en cierta franja horaria, o evitar programar clases en ciertas franjas horarias).

- Definir el problema mediante algún lenguaje para su resolución programática. El lenguaje deberá ser suficientemente flexible como para permitir la introducción de nuevos tipos de restricciones de manera sencilla. Además, se permitirá ejecutar en diferentes plataformas, ya que previsiblemente requerirá de bastante tiempo de ejecución.

- Visualizar la salida del programa a través de una interfaz gráfica, dando opción a visualizar tanto un "horario clásico" (semanal) como un "horario de ocupación" (uso de los espacios).

1.2.1. Estado de la tecnología

Existen múltiples soluciones para la generación automática de horarios, pero analizando varias de ellas no existe ninguna que se adecue a las necesidades de la *Escuela Técnica Superior de Ingeniería*. Las existentes son soluciones genéricas poco configurables que se enmarcan en la resolución de problemas sencillos.

Se pueden encontrar desde soluciones comerciales [OfC23] hasta soluciones de código abierto con investigaciones publicadas [Mü09], pero ninguna de ellas se adecúa a la idiosincrasia y funcionamiento de la *Escuela Técnica Superior de Ingeniería*. Por ejemplo no permiten crear lo que se denomina como **grupos de sesiones**[1], algo muy utilizado en la *Escuela Técnica Superior de Ingeniería* (y en la *Universidad de Santiago de Compostela* en general) que otras soluciones ya existentes no contemplan (ya que suelen estar enfocados a las universidades de Estados Unidos, y no es frecuente requerir la separación por grupos al tender a tener aularios de mayor tamaño).

Desde el punto de vista formal, **el problema a resolver es el "Curriculum-Based Course Timetabling"** (o *CB-CTT* por sus siglas) [LHG11, MEJ13, BCRT15]. Y para tal problema, debido al amplio espectro de soluciones y su complejidad, se puede resolver tanto de manera heurística como con metaheurísticas [LPRD07, Her17]. El problema a diseñar es una adaptación de este mismo pero aplicado al modo de funcionamiento de la *Escuela Técnica Superior de Ingeniería*, la cual involucra una mayor complejidad y entidades que el predefinido.

El problema de la optimización en la generación del horario se resuelve a través de un modelo de satisfacción de restricciones, utilizando el paradigma de programación declarativa de "**Answer Set Programming**" (*ASP* en adelante) [BET11] ya que permite la formulación de problemas formales como el CB-CTT. En ASP, un problema se define como un conjunto de reglas y restricciones llamado programa lógico, el cual consta de átomos (definidos por *predicados*) y reglas que

[1] Un grupo de sesiones es una agrupación de sesiones dentro de una asignatura que han de ser tomadas por todos los estudiantes, al menos tantas veces por semana como se indique. Por ejemplo, en un grupo de sesiones de tipo laboratorio con 2 repeticiones por semana y 4 grupos, los estudiantes deberán asistir a 2 sesiones, pero se programarán un total de 8 sesiones aplicando las restricciones de solapamiento entre ellas.

definen las relaciones entre ellos. El objetivo es encontrar un conjunto de átomos que satisfaga todas las reglas y restricciones del programa ("respuesta").

En concreto, se utilizará Gringo y Clasp [GKKS19], los cuales forman parte de **Clingo**. Existe una implementación a la solución del problema denominada *teaspoon* [BIK+19], la cual implementa varias restricciones, pero sigue sin satisfacer al 100 % las necesidades de la *Escuela Técnica Superior de Ingeniería*. Por ello, es necesario realizar un desarrollo específico del problema para tener en cuenta de las restricciones de uso habituales en la *Escuela Técnica Superior de Ingeniería* (las cuales son más estrictas que las de problemas ya existentes [BIK+19]).

1.3. Estructura de la memoria

La memoria se estructurará en las siguientes partes principales:

- **Especificación de Requisitos**: se elicitan los requisitos de la aplicación junto con las funcionalidades que se han de implementar para satisfacer las necesidades de la programación de horarios. Se definirán requisitos a nivel del "problema a resolver" (configuraciones a soportar por la aplicación) y los requisitos de la propia aplicación (visualización, interacción, etc.).
- **Diseño**: se describe de la arquitectura de la aplicación, la interacción de los componentes y el modelado de los datos. Se incluye además la formalización del problema con la definición de las sentencias lógicas.
- **Pruebas**: se muestran los resultados obtenidos, junto con la comprobación de la satisfacción de los requisitos y objetivos esperados inicialmente. Es decir, que la aplicación sea capaz de generar los horarios con las restricciones indicadas de manera correcta.

La memoria finalizará con un apartado de **Conclusiones**, donde se analiza el grado de éxito del Trabajo y posibles mejoras a futuro.

1.4. Descripción del sistema

El sistema será una aplicación web accesible en línea. Sin embargo, para evitar la creación de un monolito y optar por la separación entre definición del problema y resolución del mismo, se crearán dos aplicaciones diferentes:

- **Plataforma web**: esta aplicación web será la encargada de gestionar la base de datos, renderizar la interfaz web e interactuar con los componentes necesarios para invocar el planificador.
- **Planificador**: esta aplicación será un "ejecutable" invocado desde la aplicación web (o de manera local), capaz de interpetar una entrada con restricciones y devolver los horarios planificados mediante ASP con Clingo.

1.4.1. Métodos y tecnologías utilizados

Con el fin de hacer toda la aplicación escalable y acelerar el desarrollo, se opta por el uso de **tecnologías *serverless***. Esto añade un nivel de complejidad adicional al requerir de una gestión más granular de la infraestructura, pero permite el despliegue de manera automatizada en la nube de Amazon Web Services[2] gracias a su Cloud Development Kit [AWS23b].

Además, se opta por la separación de cliente-servidor, generando dos proyectos diferentes: `backend` (el cual consiste en **código Python puro** invocado por funciones *AWS Lambda* [AWS23c] mapeadas a respuestas en el protocolo HTTP) y `frontend` (una **aplicación *React*** utilizando *Ant Design* [Ant23] como librería de diseño). La comunicación entre ambos se realiza mediante una **API definida con *AWS Smithy*** [Smi23] en un repositorio `models`, un lenguaje el cual permite definir APIs y generar un fichero *OpenAPI* para posteriormente ser automáticamente mapeado tanto por `frontend` (genera el cliente) como por `backend` (genera los modelos de transferencia de datos), además de contar con especificaciones concretas para servicios de AWS. Todo esto es desplegado automáticamente mediante un proyecto de *AWS Cloud Development Kit* definido en el repositorio `cdk`, el cual genera unas definiciones de *AWS CloudFormation* para gestionar los recursos en AWS de manera automática.

En cuanto al planificador, se define el repositorio `solvers-asp` para la implementación de su código en Python como una **capa de más alto nivel sobre Clingo**. Permite ser invocado desde una función *AWS Lambda*, desde un contenedor *Docker* (*AWS Batch* [AWS23a]) o directamente desde un cliente local.

Todos los repositorios son enlazados mediante submódulos de *git* en un único repositorio, referenciando las versiones concretas y siendo desplegadas automáticamente mediante un proceso de CI/CD[3] en *Github Actions* [Git].

1.4.2. Metodología de desarrollo

El proyecto ha sido desarrollado una **metodología ágil incremental**. Las principales fases o iteraciones del desarrollo fueron:

1. **Desarrollo del planificador.** Esta fase consistió en crear una primera versión ejecutable del planificador, para poder resolver el problema de manera "manual" invocándolo sin ninguna interfaz.

2. **Diseño de la plataforma.** Posteriormente se procedió a definir la base de datos y definir la plataforma para gestionar las entidades involucradas.

[2]El despliegue ha sido probado en la región `eu-south-2` (Spain) debido a su proximidad geográfica y la disponibilidad de todos los servicios requeridos.

[3]Integración y Entrega Continuas o, por sus siglas en inglés, *Continuous Integration and Delivery*.

3. **Despliegue de la infraestructura.** A continuación, se hizo un desarrollo y despliegue de la infraestructura *serverless*. Permitió comenzar a acceder a la plataforma sin necesidad de una ejecución local.

4. **Integración entre plataforma y planificador.** Se integró el planificador dentro de la plataforma, para poder realizar invocaciones programáticas utilizando la propia interfaz.

5. **Definición de restricciones y preferencias.** Una vez finalizada la integración, se comenzaron a definir con exactitud todas las posibles configuraciones. Se fue desarrollando la plataforma para poder realizar tales configuraciones y definir el problema en el planificador.

6. **Pruebas de caso real.** Finalmente se realizaron varias pruebas del planificador con los casos de uso de la *Escuela Técnica Superior de Ingeniería* para programar los distintos estudios ofertados. En base a estas pruebas se fueron realizando pequeños ajustes de mejora tanto en la aplicación como en el planificador.

Se fueron definiendo pequeñas tareas durante todo el desarrollo en vez de grandes objetivos, ya que la gran ambigüedad existente al inicio hacía imposible seguir una metodología más clásica como cascada. Esto además reitera la necesidad de una tecnología *serverless*, la cual permite aplicar estos cambios de manera más sencilla.

1.4.3. Limitación del alcance

A pesar de la necesidad de crear y definir todos los repositorios anteriormente citados para el correcto funcionamiento de la aplicación, el proyecto se centrará en la parte de la aplicación relativa a la **invocación del planificador**. Esto incluye desde la identificación de las restricciones del problema y el diseño de la interfaz gráfica para gestionarlas, el modelado de las entidades y el almacenamiento de las mismas, hasta la definición del problema y la invocación de Clingo.

La aplicación anteriormente mencionada fue creada y es completamente funcional, pero debido a su elevado alcance y complejidad, se restringe la explicación del trabajo a la parte citada relativa al planificador. Por ejemplo, se omite el diseño de la API entre `backend` y `frontend`, de la propia infraestructura no relativa al planificador o operaciones disponibles ante los tipos de datos.

Por ello, para validar correctamente el alcance del proyecto, se definen posteriormente una serie de objetivos en el apartado 2.4. Estos serán validados para comprobar el grado de cumplimiento con esta limitación extraordinaria.

1.5. Modelos y resultados de ejemplo

Durante las pruebas, los modelos de entrada serán los **escenarios reales de los horarios de la Escuela Técnica Superior de Ingeniería**. Para ello, se ha realizado un *scrapping* y volcado de los mismos para ser insertados en la aplicación. En concreto, los más utilizados serán los del Grado de Ingeniería Informática y Grado de Inteligencia Artificial. Para restringir el archivo histórico, se utilizarán los horarios del curso académico 2022-2023 y, en concreto, del primer cuatrimestre (o primer periodo).

En cuanto a los resultados, se podrán visualizar en una interfaz tal y como se ve en la figura 1.2, la cual permite distintos tipos de visualización.

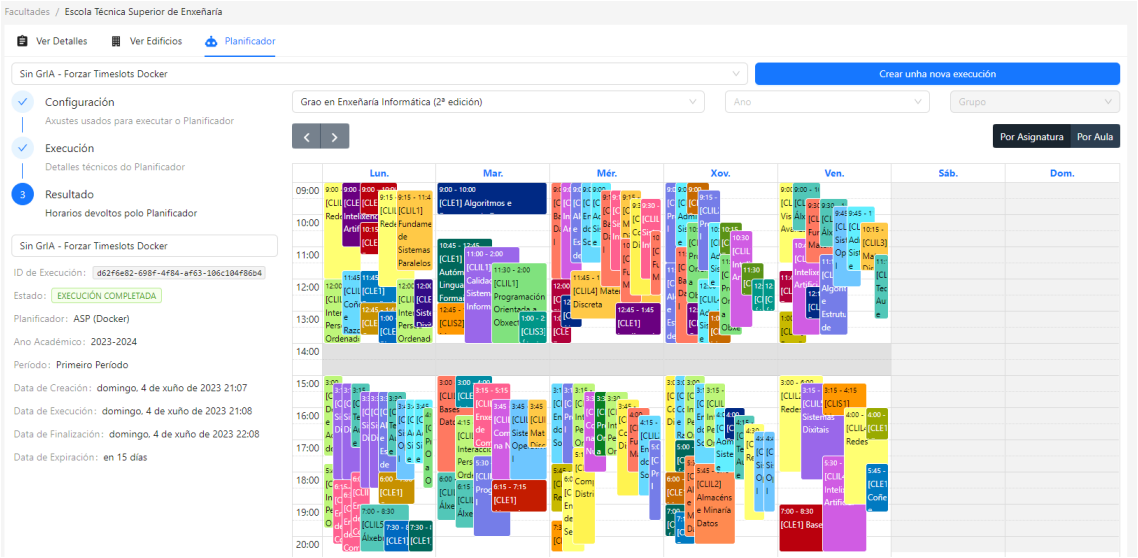

Figura 1.2: Ejemplo de visualización de la salida del planificador

La instancia desplegada es accesible a través de la URL `https://horarios.barreiro.xyz/`, la cual permite acceder al planificador para comprobar su toda funcionalidad.

Capítulo 2

Especificación de requisitos

2.1. Glosario

El objetivo de este glosario es definir ciertos términos para evitar ambigüedades a lo largo de la memoria, ya que en algunos casos la terminología podría llegar a dar lugar a confusión. Los principales términos en lo referente al sistema global son:

- **Aplicación**: sistema general que engloba la plataforma web junto con el planificador, y cuya interacción entre ambos permite definir los horarios y visualizar los resultados.
- **Plataforma (web)**: sistema accesible en línea para realizar la invocación del planificador, junto con la definición de las preferencias y restricciones. También gestiona la base de datos y la interfaz gráfica.
- **Planificador**: subsistema encargado de invocar Clingo para generar los horarios en base a los ajustes indicadas.
- **Clingo** [GKKS19]: sistema ASP para fundamentar y resolver programas lógicos. Incorpora *Gringo* (el "grounder") y *Clasp* (el "solver").

Además, conviene definir ciertos términos en lo relativo a las "entidades" a utilizar por parte del sistema:

- **Año académico**: periodo temporal con ciclo anual cuya duración equivale al curso académico (*se evita usar el término "curso académico" ya que en inglés siempre se habla de año, y "curso" puede causar ambigüedad con "asignatura"*) y que se puede dividir en distintos periodos (trimestre, cuatrimestre, semestre, etc.).
- **Facultad**: centro independiente dentro de la universidad capaz de "tener" programas, edificios y asignaturas.

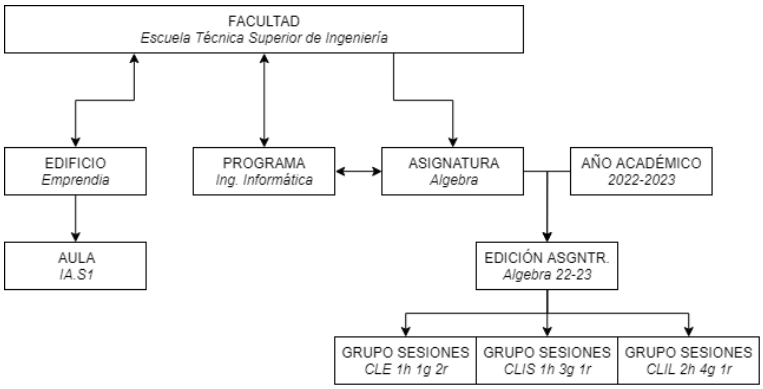

Figura 2.1: Relación entre las entidades involucradas

- **Edificio**: recurso físico dependiente de una o varias facultades.
- **Aula**: recurso físico dentro de un edificio usado para impartir docencia.
- **Programa**: agrupación de varias asignaturas en cierto año en la que los estudiantes pueden matricularse que pertenece a una o varias facultades.
- **Asignatura**: materia incorporada en uno o varios programas, cuya responsable es una única facultad y puede estar asociado a ciertas ediciones.
- **Edición de la asignatura**: para cierto año académico, indica si la asignatura tuvo docencia o no, y contiene grupos de sesiones con docencia.
- **Grupo de sesiones**: definición mínima de unidad de docencia dentro de una asignatura.
- **Sesión**: unidad programada en un espacio y tiempo de un grupo de sesiones.

La relación entre las entidades se puede apreciar en la figura 2.1.

2.2. Requisitos de la plataforma

Los requisitos de la plataforma son los requisitos que "afectan" la plataforma web en lo estrictamente relativo a la interacción con el planificador. Es decir, definen el almacenamiento de las ejecuciones o los resultados, las restricciones que debe poder soportar o cualquier otro requisito que sea necesario para que la interacción entre la plataforma y el planificador pueda funcionar.

2.2.1. Requisitos de información

En los requisitos de información de la plataforma web se define la información requerida para interactuar con el planificador (y, por tanto, se asume que las entidades ya son almacenadas de manera correcta). Por ello, se definen los requisitos listados a continuación, cuyos detalles vienen en el cuadro F.1.

- `IR-001` Almacenar los datos relativos a cada ejecución del planificador.
- `IR-002` Mantener una copia de los datos usados en la ejecución.
- `IR-003` Almacenar la configuración de la ejecución.
- `IR-004` Almacenar los resultados del planificador.

2.2.2. Requisitos funcionales

Los requisitos funcionales de la plataforma web atienden a las funcionalidades que ha de tener la plataforma, y se pueden clasificar en dos grandes grupos dependiendo del área al que afectan: **funcionalidades de la interfaz**, y **funcionalidades de restricciones y preferencias**.

Funcionalidad de la interfaz

Estos requisitos definen que aspectos y funciones ha de tener la interfaz gráfica para interactuar tanto con el usuario como con el planificador, pero no definen las configuraciones que el usuario puede utilizar. Se definen los siguientes requisitos funcionales sobre la interfaz, los cuales están detallados en el cuadro F.2.

- `FR-A001` Creación y visualización de una ejecución.
- `FR-A002` Lanzamiento de una ejecución.
- `FR-A003` Transferencia de configuraciones entre ejecuciones.
- `FR-A004` Modificación de preferencias y restricciones.

Restricciones y preferencias del planificador

Estos otros requisitos sí que definen que configuraciones ha de poder soportar la invocación del planificador. Definen las **restricciones y preferencias** que son ajustables, los parámetros de personalización, en el cuadro 2.1.

FR-B001 **Horas de inicio y fin.** Para el día natural, se permite especificar un intervalo disponible para la programación de sesiones.
FR-B002 **Duración de cada intervalo.** Se permite especificar la precisión con la que se programan las sesiones. Equivale a la duración mínima que puede tener una sesión.
FR-B003 **Semana lectiva.** Se permite especificar los días de la semana natural hábiles para programar las sesiones.
FR-B004 **Intervalos bloqueados globalmente.** Se permite especificar ciertas franjas horarias que no pueden usarse para programar sesiones.
FR-B005 **Intervalos no deseados globalmente.** Se permite especificar ciertas franjas horarias que se deben evitar para programar sesiones, con distintas prioridades.
FR-B006 **Tipos de sesión permitidos por aula.** Se permite indicar las aulas permitidas para cada tipo de sesión.
FR-B007 **Modo de planificación por programa.** Se permite indicar qué tipo de planificación se ha de aplicar a cada programa ("no planificar", "no solapar asignaturas del mismo año", etc.).
FR-B008 **Máximo de solapamientos por programa.** Se permite indicar cuantos solapamientos máximos puede haber en cada grupo de sesiones para cada programa.
FR-B009 **Preferencias de intervalos.** Se permite indicar, a nivel de programa (y año) por tipo de sesión, ciertas preferencias de intervalos: "evitar", "preferir" o "no permitido".
FR-B010 **Preferencias de aulas.** Se permite indicar, a nivel de programa (y año), ciertas preferencias de edificios y/o aulas: "evitar", "preferir" o "no permitido".

Cuadro 2.1: Requisitos funcionales del planificador en la plataforma

2.2.3. Requisitos no funcionales

Estos requisitos de la plataforma vienen definidos por la naturaleza de la misma, y son asumidos durante el desarrollo y, por tanto, no dan cabida a posibles ajustes. Son, por ejemplo, el tipo de despliegue requerido o los recursos disponibles a usar por el planificador. Sus detalles están listados en el cuadro F.3.

- NFR-001 Los componentes de la aplicación serán de tipo *serverless* en la nube de *Amazon Web Services*.
- NFR-002 Las entidades disponibles estarán limitadas al alcance de la facultad.

2.3. Requisitos del planificador

Los requisitos del planificador son diferentes a los de la plataforma al ser sistemas totalmente independientes. El planificador estará diseñado de tal forma que pueda ser conectado con otra plataforma que gestione otro tipo de restricciones y que, por tanto, será abstraído de su implementación.

2.3.1. Requisitos funcionales

Los requisitos funcionales del planificador en el cuadro F.4 definirán el comportamiento del mismo junto con sus funcionalidades. Las restricciones y preferencias vendrán heredadadas de la plataforma, pero se definen de manera diferente para abstraerse del significado que conllevan.

- FRS-001 Ejecución en distintos entornos.
- FRS-002 Ejecuciones con tiempo límite.
- FRS-003 Almacenamiento de estadísticas.

Restricciones y preferencias

Se muestran a continuación los requisitos que ha de soportar el planificador en cuanto a restricciones y preferencias. En el cuadro F.5 aparece la equivalencia con el requisito respectivo en la plataforma, ya que todo requisito FR-BXXX de la plataforma tiene al menos un requisito RP-XXX del planificador asociado.

- RP-001 Hora de inicio.
- RP-002 Hora de fin.
- RP-003 Días de la semana.
- RP-004 Duración del intervalo.
- RP-005 Intervalos modificados globalmente.
- RP-006 Tipos de sesión permitidos por aula.
- RP-007 Sesiones que no se pueden solapar.

- `RP-008` Sesiones que no deberían coincidir en el tiempo.
- `RP-009` Sesiones que deben estar en el mismo aula en caso de ser contiguas.
- `RP-010` Sesiones que deben tener en cuenta las distancias.
- `RP-011` Preferencias de aulas para sesiones.
- `RP-012` Preferencias de intervalos para sesiones.

2.3.2. Requisitos no funcionales

Estos requisitos vienen dados por la naturaleza del planificador, ya sea por su necesidad de independencia o por restricciones en el despliegue. Están mostrados sus enunciados a continuación, con los detalles en el cuadro F.6.

- `NFRS-001` Abstracción sobre el significado de los recursos.
- `NFRS-002` Conexión con las APIs de *AWS S3*.

2.4. Objetivos del proyecto

Con el fin de simplificar la meta a alcanzar en el proyecto en base a todos los requisitos, se definen los siguientes objetivos globales que agrupan otros requisitos. El fin de este proyecto es el **satisfacer estos objetivos** de manera correcta y dentro de los valores indicados en el grado de satisfacción.

Estos objetivos están definidos con la nomenclatura `OBJ-XXX` y serán los que se utilizarán para comprobar el éxito del proyecto. Vienen limitados tal y como se definió el alcance en el apartado 1.4.3.

OBJ-001	**Disponibilidad de una interfaz gráfica para la gestión de las ejecuciones del planificador.**
Descripción	Deberá existir una interfaz gráfica accesible mediante navegadores web para poder gestionar las ejecuciones del planificador. Se podrá visualizar una lista de todas las ejecuciones junto con su estado y todos los parámetros usados para su creación. Además, se podrán visualizar los estados por los que el planificador ha pasado junto con datos asociados a los mismos.
Aceptación	La interfaz gráfica existe y la experiencia del usuario es fluida y "agradable".
Grados	**Bajo**: la interfaz sólo permite seleccionar y crear las ejecuciones. **Medio**: se permite visualizar los parámetros asociados a cada ejecución y se puede navegar dentro de los estados de cada ejecución. **Alto**: se permiten exportar las configuraciones usadas entre ejecuciones.
Requisitos	IR-001, IR-002, FR-A001, FR-A002, FR-A003, NFR-002

Cuadro 2.2: Objetivo OBJ-001: interfaz gráfica para las ejecuciones

OBJ-002	**Disponibilidad de una interfaz gráfica para la visualización y modificación de las restricciones y preferencias del planificador.**
Descripción	Se deberá crear una interfaz gráfica en la plataforma web para poder realizar las modificaciones a las restricciones y preferencias de la configuración del planificador en cada ejecución. Los parámetros disponibles vienen definidos en el cuadro 2.1.
Aceptación	Se permiten configurar las preferencias mediante componentes web.
Grados	**Bajo**: sólo se permite la modificación de algunas restricciones mediante la plataforma (otros son fijos). **Medio**: se permite configurar todas las restricciones mediante la plataforma de manera intuitiva. **Alto**: existen agrupaciones para ciertas restricciones y componentes que interactúan entre sí para facilitar las configuraciones.
Requisitos	IR-003, FR-A004, FR-BXXX

Cuadro 2.3: Objetivo OBJ-002: interfaz gráfica para las preferencias

OBJ-003	**Ejecución del planificador que devuelva horarios de acuerdo a las restricciones.**
Descripción	El planificador es capaz de programar horarios de acuerdo a las entidades y configuraciones dadas en el cuadro F.5. Se produce una salida entendible por la plataforma para poder visualizarse *a posteriori*.
Aceptación	Se devuelven horarios de las sesiones y aulas solicitadas.
Grados	**Bajo**: los horarios programados no respetan las prohibiciones de solapamientos temporales especificadas. **Medio**: los horarios respetan las prohibiciones de solapamiento temporales pero algunas restricciones espaciales no son respetadas. **Alto**: todas las restricciones se cumplen de manera correcta.
Requisitos	FRS-002, FRS-003, RP-XXX

Cuadro 2.4: Objetivo OBJ-003: ejecución del planificador

OBJ-004	**Visualización de los resultados del planificador.**
Descripción	La plataforma es capaz de mostrar los resultados obtenidos del planificador de manera visual y entendible en formato "horario". Además, se deberá poder acceder a otro tipo de datos como estadísticas o posibles causas de error.
Aceptación	Se pueden visualizar los horarios semanalmente.
Grados	**Bajo**: tan sólo se puede visualizar los horarios de manera semanal. **Medio**: existe además una visualización por recurso físico ("horario de ocupación por aula"). **Alto**: se puede visualizar estadísticas y posibles causas de error recibidas del planificador.
Requisitos	IR-004

Cuadro 2.5: Objetivo OBJ-004: visualización de resultados

OBJ-005	**Escalabilidad, alta disponibilidad y versatilidad de la plataforma.**
Descripción	El desarrollo de la aplicación ha de tener en cuenta la posible escalabilidad horizontal para garantizar una alta disponibilidad. Además, dado que el planificador deberá ser un componente independiente, deberá poder ejecutarse en múltiples plataformas.
Aceptación	La aplicación cuenta con componentes *serverless*.
Grados	**Bajo**: la plataforma web cuenta con ciertos componentes *serverless* pero depende de servidores clásicos. **Medio**: toda la aplicación hace uso de componentes *serverless* y se garantiza la disponibilidad con redundancia entre zonas geográficas. **Alto**: el sistema está distribuido conectando distintos servicios y subsistemas (como el planificador), y el planificador permite ejecutarse en entornos totalmente externos conectándose a la plataforma de manera segura.
Requisitos	NFR-001, NFR-002, FRS-001, FRS-002, NFRS-002

Cuadro 2.6: Objetivo OBJ-005: escalabilidad y disponibilidad

2.4.1. Restricciones

El desarrollo de este proyecto ha venido limitado por el calendario académico de la Universidad de Santiago de Compostela y la normativa del Trabajo de Fin de Grado del Grado de Ingeniería Informática:

- Las horas de trabajo están limitadas a la extensión de la materia del Trabajo Fin de Grado. Esta viene definida con una extensión de 12 créditos ECTS [TFG23], por lo que el proyecto deberá ajustarse, en lo posible, a 300 horas.
- La fecha límite para la finalización del proyecto y la entrega de la documentación será el lunes 3 de julio del 2023.

Sin embargo, gracias al servicio de WakaTime [Wak], ha sido posible contabilizar las horas de trabajo de, estrictamente, sólo la programación del proyecto. La programación de toda la aplicación (incluyendo lo que está fuera del alcance del apartado 1.4.3) han consumido un tiempo de **502 horas y 33 minutos** hasta el 2 de julio del 2023.

Capítulo 3

Diseño e implementación

3.1. Arquitectura de la infraestructura

Tal y como se indica en el apartado 1.4.1, el desarrollo de la aplicación se ha realizado utilizando un planteamiento multi-repositorio: se separan las diferentes "aplicaciones" en sus respectivos repositorios. Esto permite desacoplar totalmente `frontend` del `backend`, pero además permite separar la definición de la comunicación entre ambos mediante la definición de la API en el repositorio de `models`. Y estos repositorios son gobernados por el repositorio `cdk`, el cual define la infraestructura a desplegar.

Esta infraestructura es clave para entender el flujo de información, la cual está **basada en componentes totalmente *serverless* de *Amazon Web Services***. Una visión de alto nivel de la infraestructura se puede observar en la figura figura 3.1. En el apéndice E se incluye un glosario de servicios de AWS utilizados para indicar las principales funcionalidades.

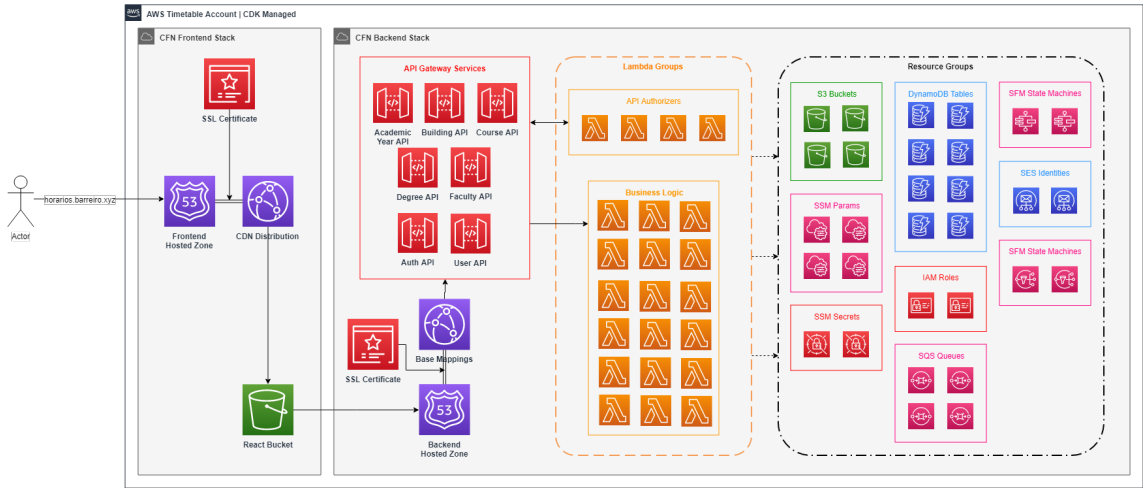

Figura 3.1: Diagrama de la infraestructura *serverless*

El flujo de trabajo a la hora de acceder a la aplicación interactiva por parte de un usuario es el siguiente:

1. El usuario accede a la URL de la plataforma (en este caso, `https://horarios.barreiro.xyz/`). Este dominio está alojado en una "zona" del componente *AWS Route 53*, el cual es un servicio DNS (una "zona" hace referencia a un dominio con sus sub-dominios).

2. Gracias a la existencia de un certificado SSL, este dominio se redirige internamente de forma segura a una distribución de *AWS CloudFront*, un servicio CDN[1] para servir la aplicación `frontend`.

3. Esta distribución de contenido está sirviendo a su vez contenido en un contenedor de *AWS S3*, cuyos archivos almacenados son en realidad la aplicación `frontend` ya compilada y lista para ser servida.

Una vez se realiza esta transferencia, el usuario ya está accediendo a la aplicación de `frontend` desde su navegador local para acceder a la plataforma. A partir de este momento, el resto de comunicaciones se realizan mediante la *API Fetch* de JavaScript [Fet] para servir todo el contenido de manera dinámica y garantizar una experiencia más fluida sin recargas de páginas y reduciendo la transferencia de datos.

Cuando la aplicación `frontend` requiere de algún dato del servidor, se realiza entonces una **comunicación con la API de** `backend` definida mediante `models`. El proceso de comunicación es el siguiente:

1. `frontend` conoce el dominio de la API de `backend` durante el proceso de compilación, tal como viene definido en el apartado 1.4.1. Por tanto, para cada comunicación requerida, se emite una solicitud a la URL con los datos necesarios.

2. Este dominio de la API de `backend` también está alojado en la misma zona de *AWS Route 53* que el dominio de la aplicación `frontend`.

3. Una vez más, existe un certificado SSL que garantiza que la transferencia de datos es segura desde `frontend` hasta las APIs de `backend`. Sin embargo, en este caso el dominio de la API está apuntando a una agrupación de *AWS API Gateway Base Mappings*, que son una agrupación de varias APIs con un dominio personalizado. Este servicio se encarga de redirigir la solicitud a la API apropiada dependiendo de la ruta solicitada.

4. *AWS API Gateway* redirige la solicitud a la API apropiada (ya que existe una API por gran servicio: "facultades", "autentificación" ,"edificios", etc.). La solicitud pasa a procesarse dentro de la API correspondiente en *AWS API Gateway*.

[1]Red de Distribución de Contenido o, por sus siglas en inglés, *Content Delivery Network*.

5. Algunas solicitudes requieren de autenticación. Para las operaciones que la requieren, *AWS API Gateway* redirige antes la solicitud a un *AWS API Gateway Authorizer* que invoca una función *AWS Lambda* (una función de computación), la cual indica si, dependiendo de la información recibida en la solicitud, tiene acceso o no a la operación. En caso de no tenerla, rechaza la solicitud indicando falta de permisos.

6. En caso de tener permisos, entonces redirige la solicitud a la función *AWS Lambda* apropiada para procesar la operación. Esta función no necesita comprobar ningún permiso, ya que ya han sido comprobados previamente, reduciendo así la complejidad de la lógica a únicamente la "lógica de negocio" necesaria para la operación.

7. Una vez dentro de la función *AWS Lambda*, se realizan las operaciones requeridas y se invocan otros servicios de AWS necesarios (por ejemplo, para la base de datos se realizarían solicitudes a *AWS DynamoDB*, para invocar el planificador a *AWS Step Functions*, o para obtener claves secretas como la de sesión a *AWS Secrets Manager*).

8. Una vez finalizado, se construye la respuesta y se devuelve desde la función *AWS Lambda*, la cual es procesada por *AWS API Gateway* para ser traducida al protocolo HTTP y devuelta a `frontend` para renderizar la interfaz.

Las diseño de una arquitectura como esta supone una serie de ventajas:

- **Escalabilidad**: los componentes involucrados tienen funciones muy específicas que pueden escalar horizontalmente. No es necesaria la reserva de recursos por adelantado.

- **Seguridad**: las funciones *AWS Lambda* tienen permisos granulares. Es decir, una función asociada al servicio de "años académicos" no va a poder acceder a la tabla de "facultades" ni invocar otros servicios que no le sean requeridos.

- **Mantenibilidad**: cada componente tiene una responsabilidad muy limitada y su lógica está totalmente separada de la infraestructura. Detectar y resolver fallos se hace (casi) de manera automática gracias a los registros de *AWS CloudWatch* y a los mapas de trazas de *AWS X-Ray*.

De esta forma, se satisfacen los requisitos relativos a escalabilidad y modularidad. En cuanto a la infraestructura del planificador, hace uso de estos pasos al requerir comunicación entre `frontend` y `backend` para interactuar con el usuario.

3.1.1. Autómata de estados del planificador

Además de las operaciones mediante la API, la ejecución del planificador consiste la ejecución de un autómata de estados o, como se denomina en AWS, una **máquina de estados**. Esta máquina de estados es la encargada de gestionar y resolver posibles errores durante la ejecución del planificador, para evitar estados inconsistentes en toda la aplicación. Su diagrama viene definido en la figura 3.2.

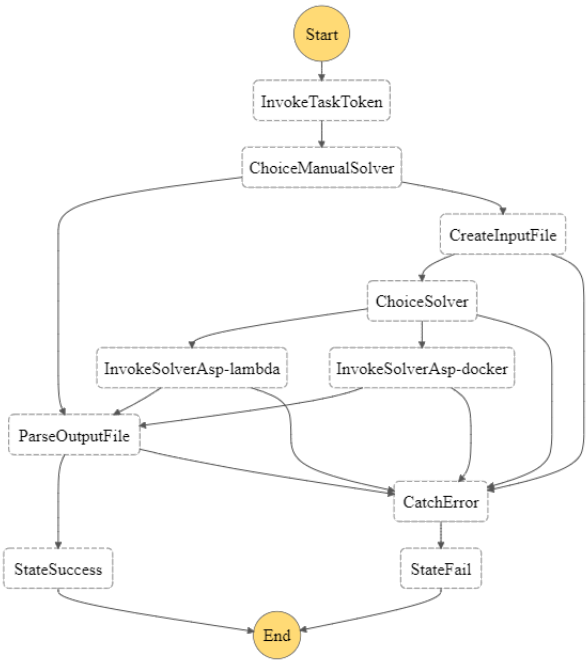

Figura 3.2: Autómata de estados del planificador

Las responsabilidades de los estados en el autómata son las siguientes:

1. `InvokeTaskToken`: este estado "espera" a que el usuario decida comenzar la ejecución del planificador. Esperará hasta 3 meses desde su creación. Invoca una función *AWS Lambda*.

2. `ChoiceManualSolver`: en caso de que la ejecución sea en un entorno externo, se salta directamente a generar el archivo del horario.

3. `CreateInputFile`: recibidos los archivos de sesiones, aulas y configuración, se genera el archivo de entrada unificado para el planificador. Invoca una función *AWS Lambda*.

4. `ChoiceSolver`: se selecciona un planificador u otro en función de la configuración.

5. `InvokeSolverName-type`: siendo `Name` el nombre del planificador y `type` el tipo (ejecución en *AWS Lambda* o *AWS Batch* para Docker), este paso es la ejecución del planificador como tal.

6. `ParseOutputFile`: dado el archivo bruto de salida del planificador, genera el archivo del horario entendible por `frontend`. Invoca una función *AWS Lambda*.

7. `CatchError`: en caso de cualquier error, este estado lo gestiona y resuelve inconsistencias dentro de la aplicación. Invoca una función *AWS Lambda*.

El alcanzar el estado final de esta máquina implica, o bien la generación de un archivo de horario estandarizado, o bien marcar como ejecución fallida la ejecución.

3.2. Modelado de la plataforma

El usuario **interactúa con el planificador utilizando la plataforma web**. Por ello, la responsabilidad de esta plataforma es la gestión del almacenamiento de datos y la invocación del propio planificador, dejando a este último con la mera responsabilidad de calcular los horarios adecuados.

3.2.1. Modelos de datos

Tal y como se avanzó en el glosario del apartado 2.1, hay una serie de entidades involucradas en el proceso de planificación horaria. Las entidades se almacenan en *AWS DynamoDB* [Ama23], una base de datos de tipo clave-valor. La clave primaria está compuesta por una **clave de partición y una clave de ordenado opcional**. Para realizar consultas, es necesario realizar la búsqueda sólo contra esos dos atributos, ya que el resto de atributos no están indexados (una consulta sobre los otros atributos resultaría en una lectura de toda la base de datos, siendo una operación muy costosa). Sin embargo, dado que en algunos casos es necesario filtrar por otros atributos, es posible crear **índices secundarios** para utilizar como clave de partición alternativa.

En cuanto a las relaciones entre los modelos, en el atributo utilizado para realizar la relación se indica la clave primaria foránea de manera manual (no existen "relaciones" como en SQL en *AWS DynamoDB*).

Con el fin de simplificar la explicación del modelo de datos, se muestran a continuación los principales atributos de las mismas junto con posibles valores a tomar:

- **Años académicos**
  - **Clave de partición**: `code` (*'2022-2023'*).

- **Facultades**
  - **Clave de partición**: `code` (*'etse'*).
  - `courses` (*'G4012101', 'G4012102'*). Relación con la tabla de *Asignaturas* de tipo uno-a-muchos.
  - `degrees` (*'G4012P01', 'G4021P01'*). Relación con la tabla de *Programas* de tipo muchos-a-muchos.
  - `edificios` (mapa de atributos):
    - `codes` (*'316e42cf-f6bc-4742-a980-e9d10b7628f1', '0adfb5b9-5cbb-472f-82a5-94c7a3b3a404'*). Relación con la tabla de *Edificios* de tipo muchos-a-muchos.
    - `distances` ({}). Es un documento JSON que almacena un diccionario de códigos de edificios con otros edificios e indica las distancias[2].
- **Edificios**
  - **Clave de partición**: `code` (*'316e42cf-f6bc-4742-a980-e9d10b7628f1'*).
  - `floors` (*-1, 0, 1, 2*).
  - `sectors` (*'Enxeñaría Química', 'Docencia'*).
  - `distances` (mapa de atributos):
    - `floors` ({}). Es un documento JSON que almacena un diccionario de dos niveles, donde la primera clave es un piso y la segunda clave es otro piso contiguo, cuyo valor es la distancia en minutos.
    - `sectors` ({}). Es un documento JSON que almacena un diccionario de dos niveles, donde la primera clave es un sector y la segunda clave es otro sector, cuyo valor es la distancia en minutos.
  - `faculties` (*'etse'*). Relación con la tabla de *Facultades* de tipo muchos-a-muchos.
- **Aulas**
  - **Clave de partición**: `building` (*'316e42cf-f6bc-4742-a980-e9d10b7628f1'*). Heredada de la tabla *Edificios*.
  - **Clave de ordenamiento**: `code` (*'A3'*).
  - `capacity` (*33*).
  - `floor` (*1*).

[2]Las distancias se almacenan a nivel de facultad ya que es posible que distintos centros con los mismos edificios midan de distinta manera las distancias (por ejemplo, debido a distintas puertas de acceso o zonas disponibles).

  - `sector` (*'Docencia'*).

- **Programas**
  - **Clave de partición**: `code` (*'G4012P01'*)
  - `faculties`: (*'etse', 'mates'*). Relación con la tabla de *Facultades* de tipo muchos-a-muchos.
  - `courses`: (*'G4012101', 'G4012102'*). Relación con la tabla de *Asignaturas* de tipo muchos-a-muchos.
- **Asignaturas**
  - **Clave de partición**: `code` (*'G4012101'*).
  - `faculty` (*etse*). Relación con la tabla de *Facultades* de tipo muchos-a-uno.
  - `degrees` (*'G4012P01', 'G4021P01'*). Relación con la tabla de *Programas* de tipo muchos-a-muchos.
- **Ediciones de asignaturas**
  - **Clave de partición**: `course` (*'G4012101'*). Heredada de la tabla *Asignaturas*.
  - **Clave de ordenamiento**: `academic_year` (*'2022-2023'*). Heredada de la tabla *Años académicos*.
  - `session_groups` (lista de mapas de atributos):
    - `discriminator` (*'2ddc'*). Es el identificador del grupo de sesiones.
    - `type` (*LECTURE*, *SEMINAR* o *LABORATORY*).
    - `num_per_week` (*2*).
    - `num_groups` (*4*).
    - `duration` (*120*).
    - `period` (*'2'*).

Para trabajar con los objetos en `backend`, se han definido unos modelos utilizando la librería *Pydantic* [Pyd23], la cual simplifica el uso y validación de los objetos. Para acceder y modificar los datos en la base de datos, se han creado funciones para realizar los cambios de manera atómica utilizando la librería *PynamoDB* [Pyn23].

Ejecución del planificador.		
PK	`execution_id`	ID único de la ejecución
	`solver`	Algoritmo a utilizar en el planificador
	`ready_checks`	Comprobación de disponibilidad de la ejecución
	`faculty`	Código de la facultad propietaria de la ejecución
	`academic_year`	Año académico relativo a la ejecución
	`periodo`	Periodo de las asignaturas a programar
	`alias`	Nombre personalizado para identificar la ejecución
	`date_created`	Fecha de creación de la ejecución
	`date_started`	Fecha de inicio del planificador
	`date_finished`	Fecha de fin (éxito o fracaso) del planificador
	`status`	Estado de la ejecución
	`expire`	Fecha de expiración (3 meses después de la creación)

Cuadro 3.1: Modelo de ejecuciones del planificador

Modelo de ejecución

Además de estos modelos de datos de las entidades, existe otro objeto adicional: las **ejecuciones del planificador**. Este modelo existe con el fin de simplificar el almacenamiento y gestión de estado de las propias ejecuciones, para evitar inconsistencias. Sus atributos están definidos en el cuadro 3.1.

- `solver`: puede tomar los valores `asp-lambda` (para ejecuciones en *AWS Lambda*), `asp-docker` (para ejecuciones en *AWS Batch*) o `asp-manual` (para ejecuciones externas).
- `ready_checks` (mapa de atributos):
  - `task_token`: almacena el código secreto para lanzar la máquina de *AWS Step Functions* una vez se quiera iniciar el planificador.
  - `courses_file`: booleano que indica que el archivo de asignaturas ha sido creado.
  - `rooms_file`: booleano que indica que el archivo de aulas ha sido creado.
- `status`:
  - *PENDING_CREATION*: todavía faltan por generar archivos de asignaturas o aulas, o la máquina no ha sido creada todavía. No se puede modificar la configuración.
  - *NOT_STARTED*: todavía no se ha iniciado el planificador, pero se pueden modificar las restricciones y preferencias.

- *RUNNING*: la configuración pasa a ser inmutable, se genera el archivo de entrada y se lanza el planificador.
- *SUCCESS*: el planificador ha devuelto algún horario con éxito y se ha generado un archivo para su visualización.
- *FAILED*: ha ocurrido algún fallo durante la ejecución.

- `expire`: cuando se cumple la fecha de expiración, AWS eliminará automáticamente el registro de la base de datos y borrará los archivos asociados de manera permanente.

3.2.2. Diseño de la interfaz

Como ya se mencionó anteriormente, la interfaz ha sido diseñada utilizando *React* junto con la librería de diseño *Ant Design* [Ant23]. Para la visualización de los horarios, se utiliza la librería *FullCalendar* [Ful23], la cual renderiza las distintas visualizaciones.

La interfaz gráfica ha sido divida en distintas secciones, tal como se aprecia en la figura 3.3. En el apéndice B existe una guía detallada sobre el uso de las preferencias y restricciones. Los elementos relevantes de la interfaz en el paso de *Configuración* son (enumerados en la imagen):

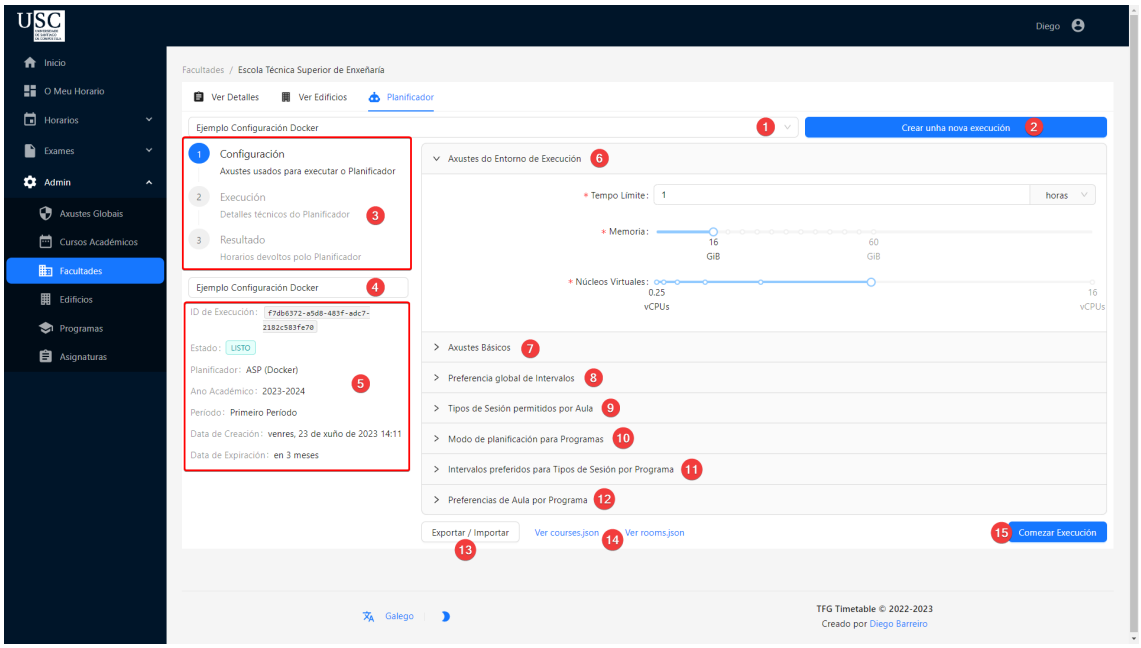

Figura 3.3: Interfaz de configuración del planificador

1. **Selector de ejecución.** Permite filtrar y seleccionar ejecuciones concretas.

2. **Creación de una nueva ejecución.** Abre un diálogo preguntando el algoritmo, año académico y periodo sobre el que crear la nueva ejecución.

3. **Pasos de la ejecución.** Con el fin de simplificar la gestión del estado, de cara al usuario tan solo se muestran 3 estados: edición de la configuración, detalles de la ejecución y resultado. Seleccionando cada uno de ellos cambia la interfaz.

4. **Alias de la ejecución.** Permite modificar el nombre de la ejecución para su mejor identificación.

5. **Datos de la ejecución.** Se muestran los datos inmutables de la ejecución (fechas, algoritmos, etc.).

6. **Ajustes del entorno.** Esta opción solo está visible para las ejecuciones sobre Docker, y permite configurar los recursos disponibles para el planificador.

7. **Ajustes básicos.** Restricciones `FR-B001`, `FR-B002` y `FR-B003`.

8. **Preferencias globales de intervalos.** Restricciones `FR-B004` y `FR-B005`.

9. **Tipos de sesión permitidos por aula.** Restricción `FR-B006`.

10. **Modo de planificación para programas.** Restricciones `FR-B007` y `FR-B008`.

11. **Intervalos preferidos para tipos de sesión por programa.** Restricción `FR-B009`.

12. **Preferencias de aula por programa.** Restricción `FR-B010`.

13. **Exportación o importación de la configuración.** Abre un diálogo que permite exportar la configuración o importarla de otra ejecución.

14. **Visualización de archivos brutos.** Permite visualizar los archivos de asignaturas y aulas.

15. **Lanzar la ejecución.** Invoca al planificador con las preferencias y restricciones seleccionadas.

Los elementos 3 y siguientes tan solo están visibles cuando se selecciona una ejecución, y los elementos 6 y siguientes cuando el paso seleccionado es *Configuración*. Los campos están bloqueados en caso de que la ejecución ya se haya lanzado al planificador. Para el paso de *Ejecución*, se muestran detalles técnicos estadísticos de la propia ejecución en el planificador.

Visualización de resultados

En el paso de *Resultado*, se muestra un mensaje de error en caso de que el planificador no haya encontrado ninguna solución válida. Pero en caso de existir algún horario posible, se muestra tal y como se ve en la figura 3.4.

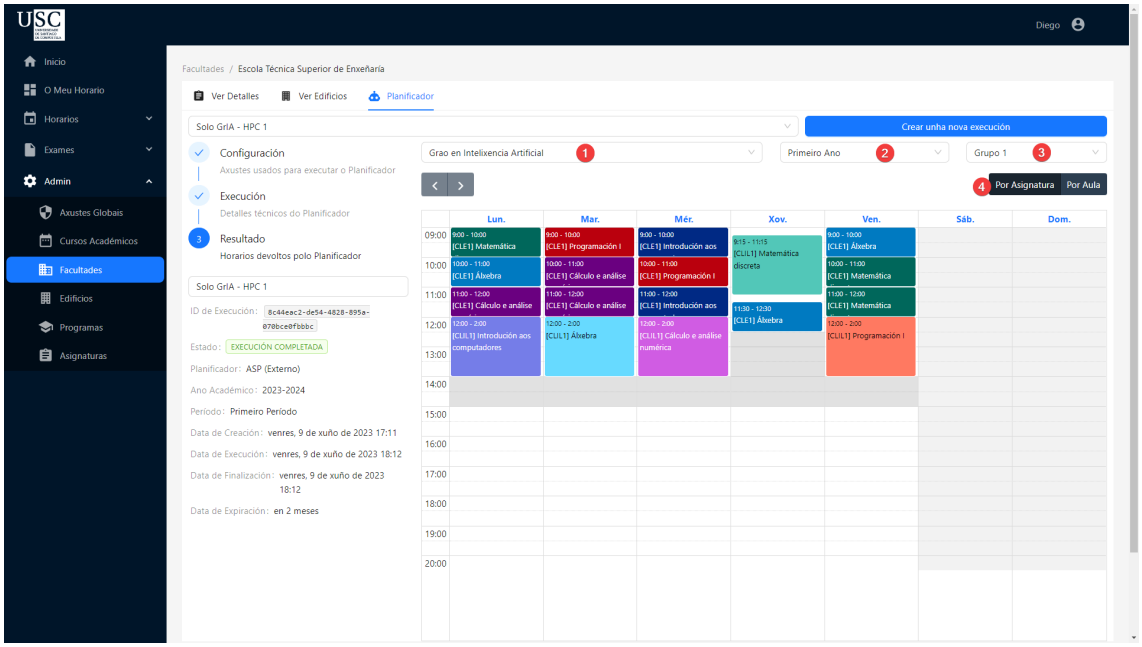

Figura 3.4: Interfaz de resultado del planificador

Los principales elementos de esta interfaz son:

1. **Filtro de programa.** Para planificaciones con varios programas, se permite "filtrar" para visualizar los horarios de un programa concreto.

2. **Filtro de año.** Si se filtra por algún programa, se puede entonces "filtrar" los horarios de las asignaturas de cierto año.

3. **Filtro de grupo.** Si se filtra por algún año, se puede entonces "filtrar" los horarios de un grupo concreto, siendo este resultado un posible horario de algún alumno.

4. **Cambio de tipo de visualización.** Permite alternar entre la visualización horaria y la visualización por recursos (horario de ocupación de aulas, figura 3.5).

Además, al hacer clic en alguna de las sesiones programadas, es posible ver todos los detalles asociados a esa sesión, tal como se ve en la figura 3.5. Esto incluye desde la duración hasta los programas en los que está incluida y el aula en la que está programada, con vínculos a las páginas para gestionar estas otras entidades.

(a) Diálogo (b) Ocupación

Figura 3.5: Diálogo de detalles de una sesión, y horario de ocupación por aulas

3.3. Ejecuciones del planificador

El planificador se ejecuta de manera externa a la plataforma (aunque puede ser o dentro del propio AWS o de manera totalmente externa). Sin embargo, existe cierta comunicación entre ambos, y es la plataforma la que gestiona el "estado" de cada ejecución del planificador. En la figura 3.6 se puede visualizar un diagrama de alto nivel (simplificando la máquina de estados, figura 3.2) de todas las interacciones requeridas para su ejecución.

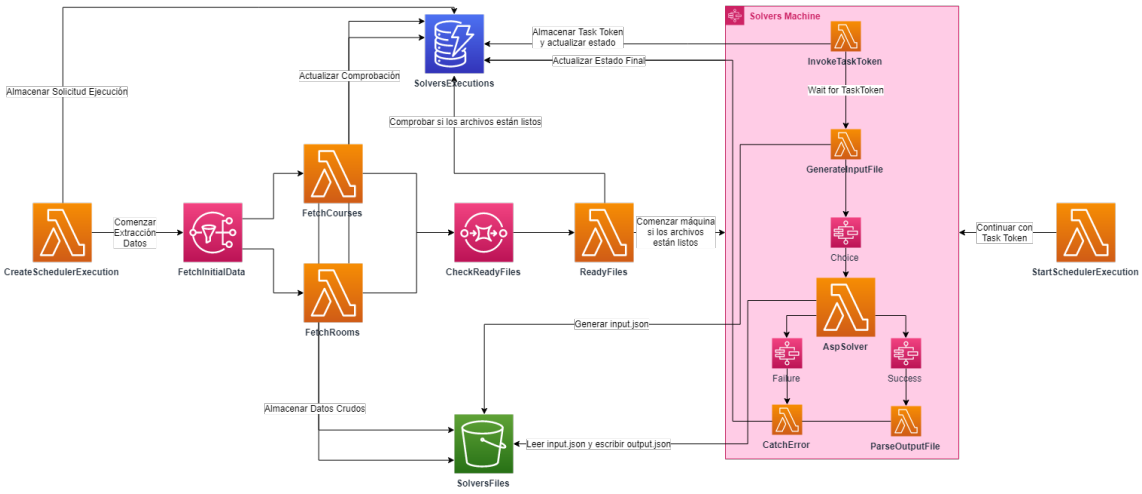

Figura 3.6: Proceso de invocación del planificador desde la plataforma

Todo el proceso fue diseñado de manera asíncrona para evitar bloqueos y consumos de recursos innecesarios. Las respuestas de la API tardan unos milisegundos, mientras por detrás se obtienen todos los datos y se actualiza la interfaz una vez está todo listo. En caso de que se hubiese realizado de manera síncrona, el

usuario quedaría bloqueado durante varios segundos, y la cantidad de permisos que tendrían que ser asignados a la función sería excesivo. Con esta arquitectura, se permite recuperar de posibles fallos y detectar errores a nivel de cada componente, evitando que un fallo catastrófico cree un estado inconsistente.

3.3.1. Creación de la ejecución

Explicando la figura 3.6, cuando el usuario solicita la creación de la nueva ejecución, desde la API se está invocando la función *AWS Lambda* llamada `CreateSchedulerExecution`. Esta función crea un registro en la tabla de ejecuciones con un ID único en el estado *PENDING_CREATION*, y a continuación publica un mensaje a un "tema" de *AWS SNS* con el ID de esta ejecución.

Otras dos funciones *AWS Lambda*, `FetchCourses` y `FetchRooms`, están escuchando este tema, y al recibir el mensaje resuelven la facultad que solicita la creación de la ejecución y proceden a crear los archivos `input_courses.json` y `input_rooms.json` respectivamente, los cuales son una "instantánea" de la base de datos de la facultad en ese momento (de tal forma que se garantiza la integridad en cada ejecución, aún habiendo cambios de por el medio). Estos archivos serán los que se utilizarán para generar la entrada del planificador y visualizar los resultados. Una vez se almacenan estos archivos en un contenedor *AWS S3* y se actualiza el `ready_check` del archivo correspondiente, se publica un mensaje en una cola de *AWS SQS* indicando que esa función ya ha acabado.

Otra función *AWS Lambda*, `ReadyFiles`, está escuchando y recibiendo mensajes de esta cola. Para cada ejecución, realmente recibirá dos mensajes iguales (ya que se publica un mensaje por cada función), pero dado que se ha configurado la integración para que espere un margen de tiempo, es capaz de des-duplicar los mensajes y tan solo procesará un único mensaje. Aún así, por seguridad, comprueba previamente que no se haya realizado ya este proceso. La responsabilidad de esta función es garantizar la existencia de ambos archivos y, en caso de ser así, iniciará una ejecución de la *AWS Step Function Machine* que invoca al planificador. Esta ejecución de la máquina tiene un ID que coincide con el ID de la ejecución del planificador.

Una vez la máquina está iniciada, el siguiente paso es "esperar" a que el usuario indique que la configuración ya está finalizada y se puede ejecutar el planificador. Para lograrlo, se invoca una función *AWS Lambda* denominada `InvokeTaskToken`. Esta función almacenará una clave secreta en `ready_checks` en la ejecución que permitirá continuar la ejecución de la máquina *a posteriori*. Una vez realizado esto, se cambia el estado de la ejecución a *NOT_STARTED*, de tal forma que el usuario ya puede editar la configuración para ajustar las preferencias y restricciones.

3.3.2. Invocación del planificador

Una vez se tiene la configuración deseada, se realiza una llamada a la API que invoca la función *AWS Lambda* `StartSchedulerExecution`. Esta función toma la clave secreta almacenada en `ready_checks` para poder continuar la ejecución de la máquina de estado. Pasa entonces la ejecución al estado *RUNNING*.

Los siguientes pasos son la creación del archivo de entrada y la propia invocación, pero ya ha sido debidamente descrita en el apartado 3.1.1 con todas las responsabilidades de cada paso.

3.4. Modelado lógico del problema

Con la plataforma y su interfaz lista, queda definir el modelado del problema. Como se comentó al inicio, se utilizará un algoritmo heurístico para resolver este problema. En concreto, se utilizará "*answer set programming*", o por su nombre en español, **programación de conjuntos de respuestas**. ASP es una forma declarativa de programación orientada a la resolución de problemas mediante espacios de búsqueda aplicando restricciones, la cual se acomoda a las necesidades de este problema.

En concreto, se utilizará **Clingo** [GKKS19], una colección de programas para resolver problemas mediante ASP. Hace uso de *Gringo* y *Clasp* (un "grounder" y un "solver") para ejecutarse, los cuales en las últimas versiones ya van integrados y distribuidos de manera conjunta.

El lenguaje utilizado se denomina *AnsProlog*, y se puede encontrar una breve introducción a su sintaxis en el bloque de código 3.1. Los comentarios comienzan con el carácter `%`. Las sentencias acaban con un `.` y tienen dos partes: cabeza (*head*) y cuerpo (*body*), tal como se ve en la línea 1. Una sentencia que solo tiene cabeza se denomina **hecho** (*head*, "*la parte izquierda siempre es cierta*", línea 3), ya que los cuerpos indican condiciones. En caso contrario, de solo tener cuerpo sin cabeza, la sentencia es una **restricción** (*constraint*, "*la parte derecha nunca es cierta*", línea 4). Si la sentencia tiene ambos elementos, se trata de una **regla** (*rule*, "*la parte izquierda sólo es cierta si la parte derecha se cumple*", línea 5).

```
1 <head> :- <body> .

3 fact(x,y,z).
4 :- constraint(x).
5 rule(X) :- fact(X,_,_).
```

Bloque de código 3.1: Ejemplo de sintaxis de Clingo

Las reglas pueden contener "generadores" (para, dada alguna condición, seleccionar de un registro de opciones), y existen otras sentencias declarativas como `#show` para realizar operaciones más concretas. Los parámetros en minúscula indican constantes, y los parámetros en mayúscula variables (de ahí que solo

aparezcan en las reglas para referenciar cabeza y cuerpo). Los términos `fact` o `constraint` usados en el ejemplo son conocidos como predicados, y pueden ser simples (sin parámetros) o compuestos (con parámetros, se usarán todos de este tipo).

Las restricciones y preferencias recibidas en el planificador no coinciden en la misma estructura con las definidas en la plataforma. Semánticamente son equivalentes, pero la forma está cambiada para simplificar y quitar significado a los datos recibidos en el planificador. Por ejemplo, el modo de planificación de los programas en la plataforma es totalmente desconocido al planificador, pero las sesiones recibidas tienen indicado con qué otras sesiones no pueden coincidir dependiendo del modo seleccionado.

Para este problema, se definen los siguientes tipos de reglas:

1. **Hechos**: definen las entidades y restricciones configuradas por el usuario en el apartado 3.4.2. *Estas son las únicas reglas que cambian de una ejecución a otra, las otras son constantes.*

2. **Generadores**: generan las posibilidades de selección en el árbol de búsqueda en el apartado 3.4.3.

3. **Normales**: generan hechos para la solución actual en base a los generadores en el apartado 3.4.4.

4. **Restricciones**: descartan soluciones en base a restricciones no satisfechas en el apartado 3.4.5.

5. **Optimizaciones**: generan los predicados para optimizar el problema en el apartado 3.4.6.

6. **Directivas**: especifican los predicados a optimizar y solución a mostrar en el apartado 3.4.7.

3.4.1. Responsabilidad código-algoritmo

A lo largo del desarrollo, se fue pasando desde un enfoque delegando toda la responsabilidad en el algoritmo, a resolver todo cuanto se pueda en código. Para plantear el problema, se realizan dos traducciones:

1. **Traducción de plataforma a planificador**: se unifican los dos archivos `input_courses.json` y `input_rooms.json` junto con las restricciones y preferencias indicadas en el archivo `input.json`.

2. **Traducción de `input.json` a Clingo**: se genera el propio problema dados los datos de entrada.

El primer paso es necesario para abstraer el planificador de todo significado: se pasa de trabajar con edificios y programas a tan solo sesiones y aulas. Por tanto, restricciones como solapamientos entre y dentro de programas se simplifican y ya se calculan en las propias sesiones.

El segundo paso supone la traducción al problema. Originalmente, este paso era simplemente una traducción a varios hechos y reglas, pero por temas de rendimiento se procedió a que el algoritmo evitase evaluar muchas de las restricciones, y que "la mayoría de cosas ya se les diese hechas". Por ejemplo, en vez de haber una restricción con respecto a qué aula está disponible para un tipo de sesión, directamente se indicaba que aulas podía utilizar la sesión (generando de esta forma solo soluciones válidas con respecto a esa restricción).

Principalmente por este segundo motivo (y la abstracción del significado), a la hora de describir el problema, habrá muchas **restricciones que ya estén implícitas en los hechos** por temas de rendimiento, y no exista una relación 1:1 entre las preferencias o restricciones nombrados en los requisitos y las reglas definidas.

3.4.2. Declaración de *hechos*

La declaración de los hechos consiste en definir en Clingo las "entidades". Se definen como hechos ya que siempre son ciertas al existir. Para identificar los hechos en Clingo se utilizan los UUIDs recibidos en el archivo de entrada (que no son los mismos que los de la plataforma), los cuales se convierten a formato hexadecimal para ser trabajados como cadenas de texto.

Para cada ejecución, se definen los predicados listados en el bloque de código 3.2 para las entidades recibidas.

```
1  timeslot(1..230).
2  undesirableTimeslot(1, 10).
3  room(room_a836, 35).
4  roomDistance(room_a836, room_c7b3, 10).
5  session(session_e65b, 4).
6  eligibleTimeslotForSession(session_e65b, 1..10).
7  eligibleRoomForSession(session_e65b, room_a836).
8  noTimeslotOverlapInSessions(session_e65b, session_c849).
9  avoidTimeslotOverlapInSessions(session_e65b,
       ↪ session_8e3d).
10 sameRoomIfContiguousSessions(session_e65b, session_c849)
       ↪ .
11 applyRoomDistancesToSessions(session_e65b, session_c849)
       ↪ .
12 preferredRoomForSession(session_e65b, room_a836).
13 penalizedRoomForSession(session_e65b, room_c7b3).
14 preferredTimeslotForSession(session_e65b, 1).
15 penalizedTimeslotForSession(session_e65b, 5).
```

Bloque de código 3.2: Lista de hechos definidos en Clingo

1. `timeslot`: **huecos horarios disponibles**; el único parámetro indica el ID incremental del hueco.
   En base a las restricciones `RP-001`, `RP-002` y `RP-003`, se generan la duración de la semana. Usando la restricción `RP-005` se genera el número de huecos horarios en la semana. La semana se considera como "un único día contiguo", de tal forma que se trata de una única línea temporal, independientemente del día de la semana.

2. `undesirableTimeslot`: **huecos horarios no deseados**; el primer parámetro indica el ID del hueco, y el segundo el valor de la penalización a aplicar.
   En base a la restricción `RP-005`, se generan los huecos que han sido modificados de manera global.

3. `room`: **aulas disponibles**; el primer parámetro indica el ID del aula, y el segundo la capacidad de la misma.

4. `roomDistance`: **distancia entre dos aulas**; los dos primeros parámetros indican los IDs de las aulas, y el tercero la distancia en número de huecos horarios.

5. `session`: **sesiones a programar**; el primer parámetro indica el ID de la sesión, y el segundo indica la duración en huecos horarios.

6. `eligibleTimeslotForSession`: **huecos horarios disponibles para programar la sesión**; el primer parámetro indica el ID de la sesión, y el segundo el hueco horario disponible.
   Teniendo en cuenta que los huecos horarios son contiguos, para una sesión con H huecos de duración, en cada día se descartan los últimos $H - 1$ huecos. Para los intervalos deshabilitados en `RP-005` y `RP-012`, se eliminan también y se aplica la misma fórmula para descartar los "anteriores".

7. `eligibleRoomForSession`: **aulas disponibles para utilizar en la sesión**; el primer parámetro indica el ID de la sesión, y el segundo el ID del aula.
   Se genera, para cada sesión, eliminando las aulas que no cumplen las restricciones `RP-006` y `RP-011`.

8. `noTimeslotOverlapInSessions`: **sesiones que no pueden solaparse en el tiempo**; ambos parámetros son IDs de sesiones.
   Se genera aplicando la restricción `RP-007`, la cual viene generada desde la plataforma.

9. `avoidTimeslotOverlapInSessions`: **sesiones que no deben solaparse en el tiempo**; ambos parámetros son IDs de sesiones.
   Se genera aplicando la restricción `RP-008`, la cual viene generada desde la plataforma.

10. `sameRoomIfContiguousSessions`: **sesiones que deben estar programadas en el mismo aula en caso de ser contiguas**; ambos parámetros son IDs de sesiones.
    Se genera aplicando la restricción `RP-009`, la cual viene generada desde la plataforma.

11. `applyRoomDistancesToSessions`: **sesiones que deben tener en cuenta las distancias al ser programadas**; ambos parámetros son IDs de sesiones.
    Se genera aplicando la restricción `RP-010`, la cual viene generada desde la plataforma.

12. `preferredRoomForSession`: **aulas preferidas para la sesión**; el primer parámetro es el ID de la sesión, y el segundo el ID del aula.
    Se genera aplicando la restricción `RP-011`.

13. `penalizedRoomForSession`: **aulas a evitar para la sesión**; el primer parámetro es el ID de la sesión, y el segundo el ID del aula.
    Se genera aplicando la restricción `RP-011`.

14. `preferredTimeslotForSession`: **huecos horarios preferidos para la sesión**; el primer parámetro es el ID de la sesión, y el segundo el ID del hueco horario.
    Se genera aplicando la restricción `RP-012`.

15. `penalizedTimeslotForSession`: **huecos horarios a evitar para la sesión**; el primer parámetro es el ID de la sesión, y el segundo el ID del hueco horario.
    Se genera aplicando la restricción `RP-012`.

3.4.3. Declaración de *generadores*

Estas reglas "generan" posibles soluciones alternativas, y son las encargadas de abrir el espacio de búsqueda. Aparecen en el bloque de código 3.3.

```
1 1 { assignedTimeslot(T, S) : eligibleTimeslotForSession
    ↪ (S, T) } 1 :- session(S, _).
2 1 { assignedRoom(R, S) : eligibleRoomForSession (S, R) }
    ↪  1 :- session(S, _).
```

Bloque de código 3.3: Lista de generadores definidos en Clingo

1. "Para cada `session`, selecciona entre 1 y 1 asignaciones de huecos horarios en el predicado `assignedTimeslot` siempre y cuando exista un hueco válido definido en `eligibleTimeslotForSession`."

2. "Para cada `session`, selecciona entre 1 y 1 asignaciones de aula en el predicado `assignedRoom` siempre y cuando exista un aula válida definida en `eligibleRoomForSession`."

3.4.4. Declaración de *normales*

Estas reglas se usan para que, en cada solución generada, se generen otros predicados utilizados más adelante. Estas reglas están especificadas en el bloque de código 3.4.

```
1 scheduledSession(T..T+H-1, S, R) :- session(S, H),
    ↪ assignedTimeslot(T, S), assignedRoom(R, S).
```

Bloque de código 3.4: Lista de normales definidas en Clingo

1. "Para cada `session` con duración H, `assignedTimeslot` y `assignedRoom`, genera H predicados del tipo `scheduledSession` con los huecos horarios yendo desde el definido en `assignedTimeslot` T hasta $T + H - 1$, ambos incluídos."

3.4.5. Declaración de *restricciones*

Estas reglas sirven para descartar posibles soluciones generadas por los generadores que no cumplen con las restricciones establecidas. En el bloque de código 3.5 se listan todas las restricciones aplicadas en Clingo.

```
1 :- not { scheduledSession(T, _, R) } 1, room(R, _),
    ↪ timeslot(T).
2 :- not { scheduledSession(T, S1, _); scheduledSession(T,
    ↪  S2, _) } 1, noTimeslotOverlapInSessions(S1, S2),
    ↪ timeslot(T).
3 :- not { scheduledSession(T-1, S2, R2); scheduledSession
    ↪ (T, S1, R1); scheduledSession(T+1, S2, R2) } 1,
    ↪ sameRoomIfContiguousSessions(S1, S2), assignedRoom
    ↪ (R1, S1), assignedRoom(R2, S2), R1 != R2.
```

Bloque de código 3.5: Lista de restricciones definidas en Clingo

1. "Descarta las soluciones que contengan más de 1 `scheduledSession` asignada en el mismo `room` y `timeslot`."

2. "Descarta las soluciones que contengan más de 1 `scheduledSession` asignada en el mismo `timeslot`, siempre y cuando esté definido el predicado `noTimeslotOverlapInSessions` ."

3. "Descarta las soluciones que contengan más de 1 `scheduledSession` asignadas en huecos horarios contiguos para dos sesiones diferentes en dos aulas diferentes, estando definido el predicado `sameRoomIfContiguousSessions`."

3.4.6. Declaración de *optimizaciones*

Una vez se tiene una solución, es necesario calcular un valor para poder compararla con otras soluciones. Esto se hace mediante un sistema de bonificación y penalización con costes y prioridades. En el bloque de código 3.6 se muestran las reglas encargadas de definir estas optimizaciones, para que en el apartado 3.4.6 se indice a Clingo como optimizar el problema.

```
1 penalty("UndesirableTimeslot",PC,S,4) :-
    ↪ undesirableTimeslot(T,PC), scheduledSession(T,S,_)
    ↪ , PC == 10.
2 penalty("UndesirableTimeslot",PC,S,5) :-
    ↪ undesirableTimeslot(T,PC), scheduledSession(T,S,_)
    ↪ , PC == 20.
3 penalty("UndesirableTimeslot",PC,S,6) :-
    ↪ undesirableTimeslot(T,PC), scheduledSession(T,S,_)
    ↪ , PC == 50.
4 penalty("AvoidRoomForSession",15,S,3) :-
    ↪ penalizedRoomForSession(S,R), assignedRoom(R,S).
5 bonus("PreferRoomForSession",15,S,1) :-
    ↪ preferredRoomForSession(S,R), assignedRoom(R,S).
6 penalty("AvoidSessionOverlap",15,S1,2) :- not {
    ↪ scheduledSession(T,S1,_); scheduledSession(T,S2,_)
    ↪  } 1, avoidTimeslotOverlapInSessions(S1,S2),
    ↪ timeslot(T).
7 penalty("AvoidTimeslotForSession",20,S,3) :-
    ↪ penalizedTimeslotForSession(S,T), scheduledSession
    ↪ (T,S,_).
8 bonus("PreferTimeslotForSession",20,S,1) :-
    ↪ preferredTimeslotForSession(S,T), scheduledSession
    ↪ (T,S,_).
```

Bloque de código 3.6: Lista de optimizaciones definidas en Clingo

1. “Aplicar penalización para sesiones programadas en huecos globalmente no deseados con coste 10 (menor prioridad).”
2. “Aplicar penalización para sesiones programadas en huecos globalmente no deseados con coste 20.”
3. “Aplicar penalización para sesiones programadas en huecos globalmente no deseados con coste 50 (mayor prioridad).”
4. “Aplicar penalización para sesiones programadas en aulas no deseadas.”
5. “Aplicar bonificación para sesiones programadas en aulas preferidas.”
6. “Aplicar penalización para sesiones programadas al mismo tiempo si no deberían estarlo.”

7. “Aplicar penalización para sesiones programadas en huecos horarios no deseados.”

8. “Aplicar bonificación para sesiones programadas en huecos horarios preferidos.”

3.4.7. Declaración de *directivas*

Estas reglas sirven para indicar a Clingo que función de optimización aplicar (a mayor valor de prioridad, antes trata de resolverla Clingo), y que solución debe devolver como resultado de la ejecución. El bloque de código 3.7 muestra todas las sentencias utilizadas.

```
1 #minimize { PC@PP,PN,PV : penalty(PN, PC, PV, PP) }.
2 #maximize { BC@BP,BN,BV : bonus(BN, BC, BV, BP) }.
3 #show scheduledSession/3.
4 #show penalty/4.
5 #show bonus/4.
```

Bloque de código 3.7: Lista de directivas definidas en Clingo

1. “Minimiza los predicados de tipo `penalty` utilizando como valor de coste el segundo parámetro, y como valor de prioridad el último parámetro.”

2. “Maximiza los predicados de tipo `bonus` utilizando como valor de coste el segundo parámetro, y como valor de prioridad el último parámetro.”

3. “Muestra los predicados `scheduledSession` (sesiones asignadas, el ‘horario’).”

4. “Muestra los predicados `penalty` (las penalizaciones aplicadas).”

5. “Muestra los predicados `bonus` (las bonificaciones aplicadas).”

Capítulo 4

Pruebas

Para comprobar el correcto funcionamiento de la aplicación, se definen una serie de pruebas a realizar. Debido a la complejidad de la aplicación, no es posible definir un "proyecto de prueba global", ya que, por ejemplo, ciertas restricciones debe analizarse por separado al ser varias de ellas incompatibles. Como resumen general, los objetivos de esta sección son:

a. **validar la usabilidad de la interfaz de la plataforma web**,

b. **validar las funcionalidades disponibles en la plataforma web**,

c. **garantizar la correcta comunicación entre plataforma y planificador**,

d. **validar la correcta aplicación de las restricciones en el planificador**,

e. y **realizar pruebas con datos de escenarios reales**.

El garantizar estos 5 elementos es sinónimo de cumplimiento de los requisitos del proyecto, tal y como estaban indicados en el capítulo 2. Por ello, para algunos de los elementos de definirán casos de prueba específicos, mientras que para otro se realizará una prueba de integración general.

Para la comprobación de los requisitos, se ha definido **al menos una prueba** por cada requisito a comprobar. Es posible que para ciertos requisitos exista más de una prueba, al requerir probar distintos casos de uso.

4.1. Pruebas de la plataforma

Estas pruebas están diseñadas para verificar el cumplimiento de los requisitos relacionados con la plataforma (descritos en el apartado 2.2). Se dividen en dos grandes subgrupos:

- **Pruebas funcionales**: garantizan el correcto comportamiento de la plataforma ante la interacción mediante las funcionalidades previstas.
- **Pruebas de la configuración**: comprueban que los cambios en las restricciones y preferencias de la ejecución del planificador se adecúan a lo esperado.

4.1.1. Pruebas funcionales

Se definen las siguientes pruebas (detalladas en el cuadro F.7) para realizar las comprobaciones sobre los requisitos funcionales:

- `FT-A001` Creación de una ejecución.
- `FT-A002` Visualización de una ejecución
- `FT-A003` Navegación entre pasos de la ejecución
- `FT-A004` Lanzamiento de una ejecución al planificador.
- `FT-A005` Exportación de la configuración de una ejecución.
- `FT-A006` Importación de la configuración a una ejecución.
- `FT-A007` Edición del *alias* de una ejecución.
- `FT-A008` Edición de la configuración de una ejecución.

Todas las pruebas han sido satisfactorias sin errores. En cuanto a los requisitos no funcionales `NFR-001` y `NFR-002`, ambos se han validado mediante las pruebas de caso real.

4.1.2. Validación de la interfaz

Con el fin de valorar la usabilidad de la interfaz de la plataforma de manera objetiva, se utilizarán dos métodos para su evaluación: un **cuestionario SUS**[1] [Bro13], y los **10 principios heurísticos de Nielsen** [Nie94, Nie05]. Mediante estos métodos, se validarán aspectos de usabilidad sobre el uso de la interfaz para el fin al que está enfocada: la planificación de horarios.

[1]Escala de Usabilidad del Sistema o, por sus siglas en inglés, *System Usability Scale*.

Cuestionario SUS

Se ha solicitado la prueba de la interfaz a un total de 15 individuos con distinto perfil: tanto estudiantes del *Grado de Ingeniería Informática* como ingenieros informáticos profesionales con varios años de experiencia. Se han incluido los resultados desglosados en el apéndice F.3.2.

La puntuación final del cuestionario es de 78,50 **sobre** 100, lo cual está por encima de la media general (68 puntos) e indica un buen diseño (aunque mejorable) de la interfaz de cara a la usabilidad. Además, se ha de tener en cuenta que el cuestionario ha sido cubierto por individuos sin experiencia previa en la programación de horarios, por lo que cabe valorar muy positivamente el hecho de obtener una puntuación relativamente alta (debido a que no se les indicó ninguna información previa sobre su funcionamiento, se quería evaluar la capacidad de deducción y entendimiento de la interfaz sin un manual).

En general, los resultados del cuestionario reflejan una buena integración y consistencia en toda la interfaz. Los usuarios fueron capaces de encontrar todas las configuraciones necesarias, aunque cabe mencionar algunas puntuaciones que reflejan una posible necesidad de aportar algunas aclaraciones o un manual de introducción básico para el uso del planificador por parte de usuarios sin conocimientos previos (lo cual era de esperar ya que es una tarea bastante específica).

Heurísticos de Nielsen

1. **Visibilidad del estado del sistema.** Cada acción que involucra una llamada al `backend` genera un estado de "carga" que desaparece cuando se finaliza la solicitud. Al finalizar, la interfaz se actualiza o muestra algún mensaje sobre la acción resultante.

2. **Relación entre el sistema y el mundo real.** La plataforma está traducida a 3 idiomas (gallego, español e inglés) utilizando un vocabulario específico y consistente para evitar ambigüedades.

3. **Libertad y control por parte del usuario.** Los usuarios tienen la opción de exportar y realizar copias de seguridad de las configuraciones con el fin de deshacer posibles errores o reutilizar configuraciones óptimas.

4. **Consistencia y estándares.** Toda la interfaz mantiene el mismo posicionamiento y mismo número de confirmaciones para realizar los mismos tipos de operaciones. Se mantiene un uso consistente en los componentes utilizados.

5. **Prevención de errores.** Se limita la entrada de valores mediante validaciones en los propios componentes, de tal forma que no puedan ocurrir errores mediante interacciones normales.

6. **Reconocimiento antes que recuerdo.** En el proceso de configuración de la planificación, las diferentes secciones comparten información con el fin de, por ejemplo, prevenir seleccionar dos intervalos ya bloqueados.

7. **Flexibilidad y eficiencia de uso.** Se permite el uso de atajos de teclados para realizar ciertas interacciones, y otras interacciones naturales como ocultar diálogos haciendo clic en otro área diferente.

8. **Estética y diseño minimalista.** Se evita mostrar información irrelevante y duplicada, simplificando la interfaz y los diálogos.

9. **Ayudar a los usuarios a reconocer.** Los mensajes de error por entradas incorrectas muestran un error descriptivo sobre la causa del mismo.

10. **Ayuda y documentación.** Existen ciertos mensajes de ayuda en los campos que pueden resultar ambiguos o desconocidos para el usuario.

El cumplimiento de estos 10 principios sobre la usabilidad de la interfaz demuestra que la interfaz es sencilla de usar e intuitiva, una organización clara de la información y una retroalimentación constante sobre el estado interno.

4.1.3. Pruebas de la configuración

Esta sección está relacionada con la prueba funcional `FT-A008`, la cual validó la persistencia de los cambios. Sin embargo, es necesario comprobar que cada cambio en la interfaz se refleja correctamente en el archivo de configuración `input_config.json`, usado posteriormente para generar la entrada del planificador.

Se han creado un total de 23 pruebas funcionales `FT-BXXX` para validar los requisitos `FR-BXXX`. Dado que todas las pruebas han sido satisfactorias, se desplaza el listado detallado de las mismas al apéndice F.3.3, con su respectiva equivalencia de los requisitos.

4.2. Pruebas del planificador

Se han realizado una serie de comprobaciones mediante pruebas funcionales para validar los requisitos funcionales del planificador, tal y como vienen detalladas en el cuadro F.9. En cuanto a la comprobación de la aplicación de las restricciones y preferencias indicadas, se ha creado el apartado 4.2.1 para analizar cada tipo de restricción.

- `FTS-001` Ejecución en *AWS Lambda*.
- `FTS-002` Ejecución en *AWS Batch*.

- FTS-003 Ejecución en entornos externos.
- FTS-004 Ejecución con tiempo límite.
- FTS-005 Creación de archivos de depuración.

Todas las pruebas han tenido un resultado satisfactorio sin errores. Los requisitos no funcionales NFRS-001 y NFRS-002 están validados mediante la prueba de integración, en la que se comprueba la ocultación de todo significado y la integración con AWS.

4.2.1. Validaciones de la generación de horarios

Para validar la correcta generación de horarios por parte del planificador, se procedió a aplicar ciertas funcionalidades FR-BXXX de tal forma que afecten a las restricciones RP-XXX para ser probadas. Se muestran a continuación los nombres de las pruebas realizadas, junto con anotaciones para las que no han sido satisfactorias.

- RPT-001 Cumplimiento de la hora de inicio.
- RPT-002 Cumplimiento de la hora de fin.
- RPT-003 Cumplimiento de la semana lectiva.
- RPT-004 Cumplimiento de la duración del intervalo.
- RPT-005 Cumplimiento de la duración de las sesiones.
- RPT-006 Cumplimiento de los intervalos bloqueados.
- RPT-007 Cumplimiento de los intervalos no deseables.
- RPT-008 Cumplimiento de las aulas permitidas.
- RPT-009 Cumplimiento de sesiones que no pueden solaparse.
- RPT-010 Cumplimiento de sesiones que no deberían solaparse.
- RPT-011 Cumplimiento de sesiones que han de estar en el mismo aula.
  Parcialmente satisfactorio. Algunas sesiones contiguas que han de estar en el mismo aula no respetan la restricción.
- RPT-012 Cumplimiento de sesiones que han de tener en cuenta las distancias para programarse.
  No satisfactorio. No se tienen en cuenta las distancias para programar las sesiones.
- RPT-013 Cumplimiento de las preferencias de las aulas en las sesiones.
- RPT-014 Cumplimiento de las preferencias de los intervalos en las sesiones.

4.3. Pruebas de integración

Finalmente, una vez todas las funcionalidades están validadas, se procede a realizar una prueba de integración global. Esta prueba general consiste en el uso continuado de la plataforma de todas las funcionalidades, en la que se comprueba que todos los componentes interactúan de manera correcta.

- La comunicación entre `frontend` y `backend` es correcta.
- Las interacciones entre componentes de AWS en `backend` son correctas.
- La comunicación entre `backend` y `solvers-asp` es correcta.

Para el primer elemento, la definición de la API y de los clientes para interactuar con las mismas se realiza de manera automática con *AWS Smithy*. Gracias a esto, las solicitudes ya se formatean en el formato adecuado, y no hay posibilidad a fallos internos por solicitudes mal formadas (*AWS API Gateway* rechaza cualquier solicitud cuyos parámetros no coincidan con los especificados).

El segundo elemento se comprueba mediante la monitorización en *AWS CloudWatch*. Se han definido unos paneles de control para obtener métricas de latencia y fallos en las interacciones, y durante todas las pruebas realizadas no ha surgido ningún valor fuera de lo esperado. Además, mediante *AWS X-Ray* se ha comprobado para ciertas operaciones API complejas que las llamadas a los componentes AWS son las correctas. Como dato general, la latencia P90 de respuesta de las operaciones API es de menos de 150 ms, lo cual excede con las expectativas de rendimiento con respecto a una aplicación convencional.

Y el tercer elemento es comprobado mediante las ejecuciones de los distintos algoritmos de planificador. Para las ejecuciones en *AWS Lambda* y *AWS Batch*, la ejecución no llega a salir de AWS, por lo que todas las comunicaciones se hacen en la propia red interna. Para las ejecuciones en entorno externo, la comunicación se realiza mediante la API privada, y no se ha percibido ningún error entre el clúster HPC del CiTIUS - *Centro Singular de Investigación en Tecnologías Inteligentes* y AWS (se han ejecutado un total de 84 programaciones sin fallos).

4.3.1. Caso de uso real

Para estos casos de prueba de integración general, se han estado utilizando datos reales de la *Escuela Técnica Superior de Ingeniería*. Estos datos se obtuvieron mediante una técnica de *scrapping*, lo cual permitió crear una réplica exacta de las asignaturas, programas y edificios en el año académico 2022-2023. Además, se definieron exactamente los mismos grupos de sesiones a los existentes, permitiendo intentar generar horarios para este escenario real.

Para poner en contexto la magnitud del problema, la *Escuela Técnica Superior de Ingeniería* tiene un total de 222 asignaturas a programar en el año

académico 2022-2023. Esas asignaturas tienen un total de 182 grupos de sesiones (algunas asignaturas no tienen docencia en el primer cuatrimestre), los cuales generan un total de 482 **sesiones a programar**. Se disponen de un total de 36 **aulas** para programar estas sesiones. Dado que se tiene un intervalo de 15 minutos para programar las sesiones y las horas de inicio y fin son las 9:00 y 20:30 respectivamente, se sabe que en un día hay 46 intervalos disponibles, o 230 **en una semana** de lunes a viernes. Con lo cual, se pueden aplicar una fórmula rápida para obtener una estimación de cuantas posibilidades de programación hay disponibles (exponenciación de las sesiones programar a los intervalos disponibles y a las aulas disponibles):

$$452^{36^{230}} \simeq \text{ número con } 2{,}4 \cdot 10^{358} \text{ dígitos}$$

Este número está calculado a la alza ya que algunas posibilidades ya se obvian de generar restringiendo huecos horarios y otras posibilidades no son válidas por solapamientos. Sin embargo, es una estimación que permite visualizar la magnitud del problema, al comprobar que es intratable calcular todas las soluciones por *fuerza bruta*.

Se probó a ejecutar el programa en el clúster HPC del CiTIUS - *Centro Singular de Investigación en Tecnologías Inteligentes*[2] reservando un nodo de 48 núcleos, pero tras 24 horas de ejecución no fue capaz de obtener una solución válida para la *Escuela Técnica Superior de Ingeniería*. Esto era de esperar debido a la complejidad del problema, ya que se trata de un elevado número de posibilidades. Por tanto, se procedió a reducir el alcance del problema y analizar los programas individualmente.

- **Grado en Ingeniería Informática**
  Para este programa, se requiere al menos 2 horas de ejecución para obtener horarios similares a los actuales.
- **Grado en Ingeniería Química**
  Este caso es muy similar al anterior, ya que el número de asignaturas es similar en ambos grados.
- **Grado en Inteligencia Artificial**
  Este grado era más sencillo que los anteriores, ya que solo tenía un año disponible y las restricciones "'encuadraban' y restringían de forma más dura los horarios, y en tan solo 5 minutos se obtenían prácticamente los mismos horarios a los existentes.

En estas condiciones, se encontraron bastantes soluciones válidas dentro de tiempos razonables. A pesar de que una ejecución de varias horas pueda parecer

[2]Las pruebas han sido realizadas en los nodos $3 \cdots 9$, los cuales cuentan con 2 × Intel Xeon Gold 5220R @2,2 GHz (24 núcleos) y 192 GB de memoria RAM [CiT]. Las pruebas reservaban todo un nodo para ser ejecutadas.

un tiempo "excesivo", cabe mencionar que la creación de los horarios de forma manual es una tarea tediosa que lleva varios días y que requieren ser modificados aún una vez ha empezado la docencia para el periodo en cuestión. Los horarios generados cumplen estrictamente con todas las restricciones dadas, por lo que no hay lugar a posibles solapamientos innecesarios. En el apartado 5.1 se mencionan posibilidades para mejorar los horarios generados, ya que todavía hay cierto margen de mejora adicional para esta generación automatizada.

Se adjunta en la figura 4.1 el resultado de una ejecución para el *Grado de Ingeniería Informática*, y en la figura 4.2 el resultado de una ejecución para el *Grado de Inteligencia Artificial*. Además, para el *Grado de Ingeniería Informática*, se ha filtrado para mostrar el horario para el 3er año para evitar una saturación de sesiones en la imagen.

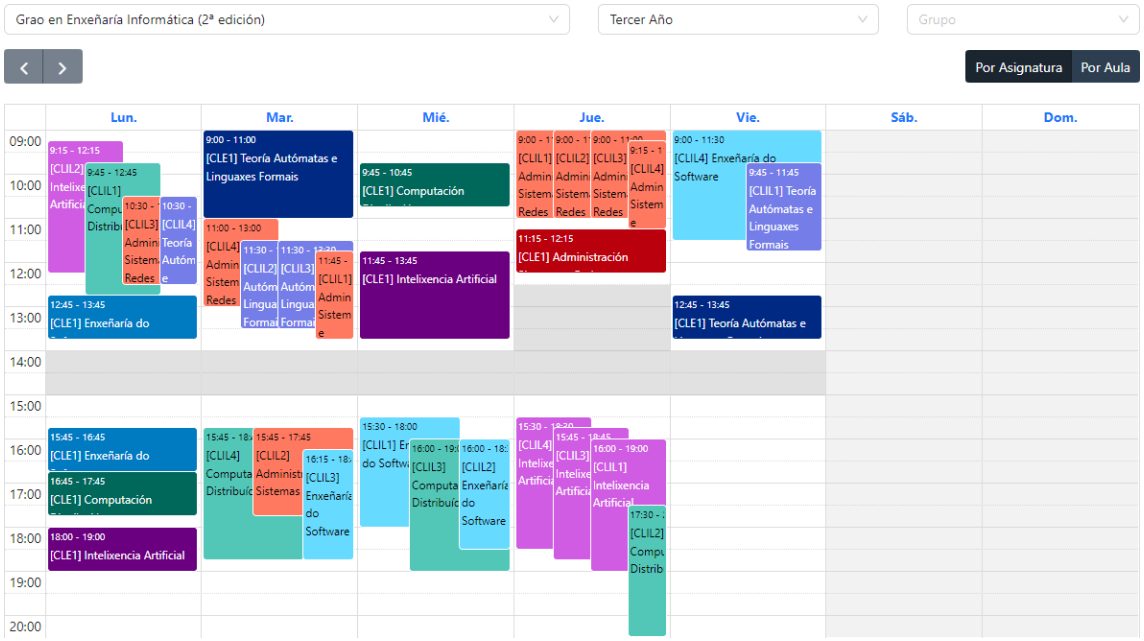

Figura 4.1: Resultado de una ejecución para el *Grado de Ingeniería Informática*

4.4. Cumplimiento con los objetivos

En el apartado 2.4 se definían 5 **objetivos principales** del proyecto. Estos objetivos engloban los requisitos y se definen con el fin de evaluar el desarrollo general del Trabajo. Por ello, se muestra a continuación el grado de cumplimiento para cada uno de los objetivos propuestos.

- `OBJ-001`: **Disponibilidad de una interfaz gráfica para la gestión de ejecuciones del planificador.** Cuadro 2.2

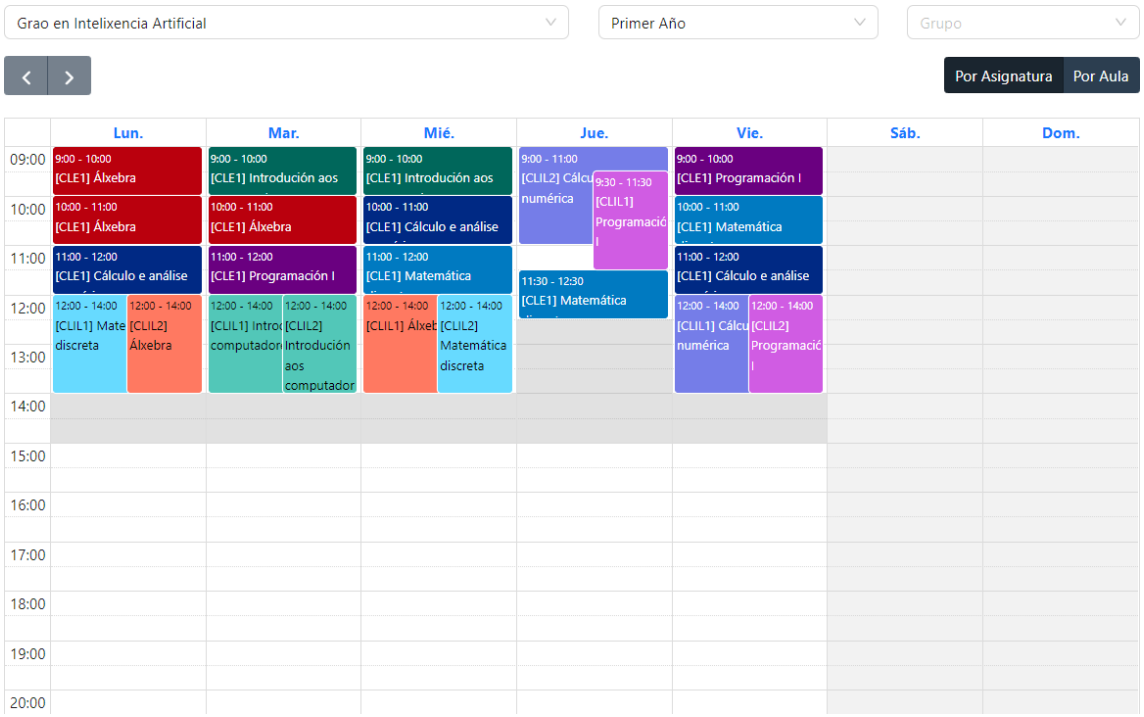

Figura 4.2: Resultado de una ejecución para el *Grado de Inteligencia Artificial*

Grado de cumplimiento: Alto (*se permiten exportar las configuraciones usadas entre ejecuciones*).
La interfaz de la plataforma cuenta con todas las funcionalidades esperables para una aplicación como está. Además, esta interfaz no requiere de ninguna recarga de la página, reduciendo la transferencia de datos y los tiempos de espera para visualizar los datos.

- `OBJ-002`: **Disponibilidad de una interfaz gráfica para la visualización y modificación de las restricciones y preferencias del planificador.** Cuadro 2.3
  Grado de cumplimiento: Alto (*existen agrupaciones para ciertas restricciones y componentes que interactúan entre sí para facilitar las configuraciones*).
  La interfaz para gestionar las restricciones y preferencias cumple con las expectativas. Además, con el fin de facilitar su uso y gestión, se han utilizado componentes de las librerías *Ant Design* y *FullCalendar*, los cuales hacen el proceso de definir los parámetros mucho más intuitivos (por ejemplo, para indicar intervalos simplemente hay que seleccionarlos en el propio calendario) al no tener que escribir nada directamente.

- `OBJ-003`: **Ejecución del planificador que devuelva horarios de acuerdo a las restricciones.** Cuadro 2.4

Grado de cumplimiento: Medio (*los horarios respetan las prohibiciones de solapamiento temporales pero algunas restricciones espaciales no son respetadas*).
Los horarios devueltos por el planificador son consistentes temporalmente de acuerdo a las restricciones y preferencias. Tampoco hay varias sesiones programadas al mismo tiempo en un mismo aula. Sin embargo, ciertas restricciones espaciales (mismo aula para sesiones contiguas, tener en cuenta las distancias) no son aplicadas correctamente debido al coste computacional que supone el tener en cuenta estas restricciones.

- `OBJ-004`: **Visualización de los resultados del planificador.** Cuadro 2.5
  Grado de cumplimiento: Alto (*se puede visualizar estadísticas y posibles causas de error recibidas del planificador*).
  Existe una interfaz que permite visualizar los resultados de manera cómoda. Además, se permite visualizar un horario de ocupación de recursos, junto con estadísticas y otros datos de depuración recibidos por parte del planificador.

- `OBJ-005`: **Escalabilidad, alta disponibilidad y versatilidad de la plataforma.** Cuadro 2.6
  Grado de cumplimiento: Alto (*el sistema está distribuido conectando distintos servicios y subsistemas, y el planificador permite ejecutarse en entornos totalmente externos conectándose a la plataforma de manera segura*).
  Se ha comprobado que la latencia P90 de la plataforma no supera los 150 ms. El sistema es capaz de escalar horizontalmente cada uno de los componentes gracias al uso de la tecnología *serverless*, y además es posible desacoplar totalmente el planificador de la nube al poder ejecutarse en entornos externos y comunicarse mediante una API. Cabe mencionar que parte del trabajo realizado para este objetivo ha quedado excluido del alcance del proyecto, tal y como se explicó en el apartado 1.4.3.

Capítulo 5

Conclusiones

Finalmente, una vez se ha comprobado la implementación de los requisitos y el correcto funcionamiento de la plataforma, es posible sacar una serie de conclusiones de todo el proyecto en general.

El desarrollo de la aplicación culmina con el **despliegue satisfactorio de la plataforma** en la nube de *Amazon Web Services*. Es posible utilizar al completo todas las funcionalidades descritas (incluyendo la gestión de las entidades). La interfaz mediante la cual se pueden programar estos horarios ha obtenido una muy buena puntuación por parte de usuarios totalmente desconocedores de esta tarea, garantizando una gran usabilidad de la aplicación. Además, la integración de la plataforma como subsistemas separados es excelente, y se permite ejecutar el propio planificador (el *alma* del Trabajo) en múltiples entornos, haciéndolo completamente modular.

Como conclusión general, en base al grado de éxito de los objetivos en apartado 4.4 se puede cerrar el proyecto como **completado con éxito**.

5.1. Posibles ampliaciones

A pesar de tener una aplicación completamente funcional, a lo largo del desarrollo se fueron dejando "pequeños detalles" que podrían ser puntos de mejora en un futuro. El propio alcance del trabajo ya está limitado y no muestra todo el desarrollo de la aplicación final, por lo que en general hay muchas opciones de expansión posibles.

En lo referente a la plataforma, los principales puntos de mejora o posibles ampliaciones son:

- **Finalizar la aplicación para brindar un sistema de gestión de horarios completo.**
  La aplicación puede gestionar todas las entidades, pero dado que el trabajo se centraba en la generación de horarios, ciertos aspectos como la gestión del expediente o visualización pública de horarios de dejaron sin desarrollar.

- **Permitir cancelar ejecuciones.**
  Se podría permitir cancelar ejecuciones en *AWS Batch* para que, en caso de detectarse algún error o querer editar la configuración, se pudiese sin necesidad de consumir recursos innecesarios.

- **Creación de pruebas en el flujo de despliegue.**
  Para garantizar una mayor excelencia operacional [IBM], se podrían definir una serie de pruebas más avanzadas. En el proceso de compilación se podrían definir una serie de pruebas de integración entre plataforma y planificador, para automáticamente detectar posibles fallos de comunicación. Además, en el proceso de despliegue, se deberían definir pruebas de carga automatizadas con el fin de garantizar la escalabilidad, y pruebas de "canarios" para detectar posibles caídas del servicio antes que los propios usuarios.

- **Validaciones de la configuración.**
  Ahora mismo no se hace ninguna comprobación con respecto a la configuración sin enviarse al planificador. Sería interesante que la propia plataforma fuese capaz de indicar posibles contradicciones (aulas preferidas y no deseadas al mismo tiempo) o que cierta configuración generará un problema sin solución (los intervalos disponibles son demasiado pequeños como para alojar la sesión más larga).

Y en lo que respecta a la generación de horarios, también hay una serie de aspectos a mejorar. Sin embargo, algunos de ellos ya fueron implementados pero posteriormente descartados por su elevado coste temporal de ejecución, pero son puntos de posible mejora de cara a futuro.

- **Compactación de horarios.**
  En la línea temporal, se debería conseguir que las sesiones se programasen de la manera más compacta posible, reduciendo la jornada (o incluso los días) que requiera docencia.

- **Optimización de espacios.**
  Teniendo los datos de ocupación y división por pisos y sectores de los edificios, se podría tratar de optimizar la generación para programar las sesiones ocupando el mayor porcentaje de cada piso o sector. De esta forma, sería posible ahorrar energía cerrando regiones no necesarias.

- **Generación con vistas al profesorado.**
  El planteamiento de este trabajo se basa en generar horarios de cara al alumnado (programas, años, etc.). Sin embargo, podría ser posible tener en cuenta qué docentes imparten cada sesión, para tener también en cuenta su posible dispersión temporal o las distancias entre aulas.

- **Configuración de los parámetros de Clingo.**
  Clingo tiene la opción de configurar parámetros para definir perfiles de ejecución en *clasp*. Se podría investigar si el uso de otros perfiles mejora los tiempos de ejecución.

A pesar de estos "grandes" puntos de mejora, cabe mencionar de nuevo el funcionamiento general de toda la plataforma, y la generación de horarios en tiempos razonables. La complejidad del problema crece de manera exponencial, por lo que el aplicar algunas de estas mejoras implicaría un mayor coste que requeriría un estudio detallado, puesto que quizás no llegue a compensar.

5.2. Lecciones aprendidas

Este trabajo ha ido evolucionando mucho desde la propuesta inicial hace ya casi un año. Originalmente se presentó como una idea para implementar una plataforma para, simplemente, gestionar los horarios y brindar esa aplicación para visualizar y acceder de manera cómoda a los horarios. Sin embargo, se redirigió hacia algo más experimental: la generación automática de estos horarios. *A priori* esta tarea parecía sencilla: generar los horarios en base a ciertas preferencias. Sin embargo, analizando las herramientas posibles y estudios ya existentes, se empezó a divisar la complejidad real del problema, y se comprobó la gran cantidad de posibles soluciones que habría que evaluar.

Durante la experimentación, se divisó otro problema: la gestión de los propios recursos. Iba a ser muy tedioso el definir de manera constante las sesiones a programar junto con los espacios disponibles. Por ello, juntado con la idea original, se procedió a realizar una idea mixta: **una aplicación que sea capaz de generar los horarios de manera intuitiva**. Y es cuando comenzó el desarrollo de la plataforma con arquitectura *serverless*, con el fin de experimentar con esta tecnología y realizar un proyecto completamente funcional y escalable.

Sin duda, la realización de este trabajo ha supuesto mucho más esfuerzo del requerido sobre el papel, pero no ha sido en vano. A través de las más de 94,300 líneas de código del proyecto, se ha podido experimentar y aprender muchas nuevas tecnologías, y sobre todo entender las verdaderas complicaciones de traducir un problema que a simple vista parece sencillo (como la programación de horarios) a algún paradigma de programación declarativa.

Como futuro proyecto personal, se continuará el desarrollo tanto de la plataforma como del planificador, con el fin de poder completar e implementar las mejoras mencionadas. Quizás algún día se pueda poner en producción en alguna universidad esta aplicación...

Apéndice A

Manuales técnicos

A.1. Código fuente del proyecto

Tal y como se describe en apéndice C, el código desarrollado se distribuye mediante una licencia propietaria con todos los derechos reservados. Una copia del código fuente se puede encontrar en la siguiente ubicación:

- **URL**: `https://diego.gal/tfg`
- **Contraseña**: `etse2023`

Tras introducir la contraseña, ese enlace redirige a un archivo `.zip` con el código fuente, alojado en la nube de *Microsoft OneDrive* de la *Universidad de Santiago de Compostela*. Para acceder al archivo se requiere una cuenta institucional de la *Universidad de Santiago de Compostela*.

A.2. Desarrollo en entorno local

Algunos repositorios no pueden ser desarrollados en entorno local y los cambios requieren ser desplegados a *Amazon Web Services*. Se muestran las opciones disponibles a continuación:

- `backend`: se requiere realizar las pruebas en la cuenta de AWS.
- `frontend`: se pueden probar la ejecución en entorno local simulando un servidor `backend` local.
- `models`: se pueden probar los modelos generados con el código local.
- `solvers-asp`: se pueden probar los cambios localmente utilizando la línea de comandos `cli`.
- `cdk`: se requiere realizar las pruebas en la cuenta de AWS.

A.3. Despliegue de la aplicación

Para desplegar la totalidad de la aplicación, es necesario contar con una cuenta de *Amazon Web Services*. El despliegue tendrá un coste mínimo mensual de 0,50€ asociado al alojamiento de la zona en *AWS Route 53* y de 0,40€ asociado a la clave secreta de las sesiones en *AWS Secrets Manager*.

El primer paso es generar los paquetes de todos los repositorios. Es necesario contar con *Java 8*, *Python 3.10* y *Node.js 18* instalados. Se muestran los comandos requeridos a continuación para cada repositorio (se han de ejecutar en su respectiva carpeta).

```
1 # Compilar los modelos
2 ./gradlew build
```

Bloque de código A.1: Compilación de `models`

```
1  # Instalar las dependencias
2  mkdir -p build/deps/python/
3  pip install --target=build/deps/python/ -r
     ↪ requirements.txt
4  pip install --platform=manylinux2010_x86_64 --only-
     ↪ binary=:all: --target=build/deps/python/ --
     ↪ upgrade cryptography

6  # Preparar el codigo
7  mkdir -p build/code/python/
8  cp -R src/* build/code/python/

10 # Empaquetar
11 mkdir dist/
12 cd src/         && zip     ../dist/handler.zip
     ↪          lambda_handlers.py && cd ../
13 cd build/code/ && zip -r ../../dist/code.zip
     ↪          .                    && cd ../../
14 cd build/deps/ && zip -r ../../dist/dependencies.zip
     ↪  .                   && cd ../../
```

Bloque de código A.2: Compilación de `backend`

```
1 # Instalar dependencias
2 npm install

4 # Compilar aplicacion
5 npm run build
```

Bloque de código A.3: Compilación de `frontend`

```
# Instalar dependencias
mkdir -p build/python/
pip install --target=build/python/ -r requirements.
    ↪ txt

# Empaquetar
mkdir dist
cd src/   && zip -r ../dist/code.zip          . && cd
    ↪  ..
cd build/ && zip -r ../dist/dependencies.zip . && cd
    ↪  ..

# Preparar Docker
mkdir docker
cp    Dockerfile       docker/
cp    requirements.txt docker/
cp -R src/             docker/
```

Bloque de código A.4: Compilación de `solvers-asp`

El siguiente paso es compilar la infraestructura con *AWS Cloud Development Kit*. Este paso requiere *Node.js 16*, y tener configurada la sesión con las credenciales de la cuenta de *Amazon Web Services*. El primer despliegue tardará en torno a 20 o 25 minutos, pero los siguientes serán menos de 5 minutos.

```
# Instalar dependencias
npm install

# (Solo para el primer despliegue) Preparar la
    ↪ cuenta para despliegues CDK
cdk bootstrap

# Desplegar infraestructura
npm run deploy
```

Bloque de código A.5: Compilación de `cdk`

Apéndice B

Manuales de usuario

B.1. Uso de la plataforma web

Para probar la plataforma web, se puede o bien desplegar una instancia personal tal y como se describe en las instrucciones del apéndice A.3, o utilizar la instancia de prueba alojada en `https://horarios.barreiro.xyz/`.

Accediendo a `https://horarios.barreiro.xyz/` con una cuenta institucional de la *Universidad de Santiago de Compostela* creará automáticamente una cuenta en la plataforma. Esta cuenta tendrá los permisos de *gestión* para la facultad de la *Escuela Técnica Superior de Ingeniería*. Se podrá acceder a la URL `https://horarios.barreiro.xyz/admin/faculty/etse#scheduler`, en la cual se pueden realizar las pruebas que se deseen para invocar al planificador.

B.1.1. Configuración de preferencias y restricciones

Navegación entre ejecuciones

Existe un selector en la parte superior de la página con el que se pueden seleccionar las ejecuciones del planificador disponibles asociadas a la facultad en cuestión. Las planificaciones están ordenadas de más recientes a menos recientes, y su nombre es el *alias* indicado (y en caso de no haberlo, un nombre con la fecha de creación).

Para crear nuevas ejecuciones, existe un botón al lado del selector de ejecuciones. Se abrirá un diálogo donde habrá que seleccionar el año académico, el periodo y el algoritmo. Estos 3 ajustes son permanentes y no se pueden modificar una vez creada la ejecución. El *alias* sí puede ser alterado (un nombre personalizado para identificar la ejecución) mediante un campo de texto visible al margen izquierdo de la pantalla una vez seleccionada la ejecución.

Dentro de cada ejecución, existe un selector de "paso" para navegar entre los distintos estados de la ejecución. La configuración puede ser gestionada en el paso *Configuración*, y cuando ya se ha lanzado la ejecución se pueden ver detalles en

el paso *Ejecución*. Finalmente se pueden ver los resultados en *Resultado*, el cual informará del error producido en caso de no ser un resultado de éxito.

Gestión de la configuración

La configuración de las restricciones y preferencias a enviar al planificador ha sido agrupada en distintas secciones "comprimibles" para facilitar su uso y evitar renderizar una página excesivamente grande. Estos grupos son:

- **Ajustes del Entorno de Ejecución** (sólo para ejecuciones en *AWS Batch*): figura B.1.
  Permite gestionar los recursos disponibles para la ejecución: tiempo de ejecución, núcleos y memoria.

- **Ajustes Básicos**: figura B.2.
  Regula las horas de inicio y fin, las duraciones de los intervalos y los días de la semana. Permite gestionar los recursos disponibles para la ejecución: tiempo de ejecución, núcleos y memoria.

- **Preferencia global de Intervalos**: figura B.3.
  Regula las preferencias globales de los intervalos (no deseados o bloqueados). Seleccionando y deslizando el ratón se pueden seleccionar los nuevos intervalos (no se permiten solapamientos), y hacer clic en intervalos ya modificados permite editarlos.

- **Tipos de Sesión permitidos por Aula**: figura B.4.
  Gestiona que aulas permiten ciertos tipos de sesiones. Seleccionando las aulas se pueden transferir de un bloque a otro para activarlas o desactivarlas.

- **Modo de planificación para Programas**: figura B.5.
  Permite especificar que modo de planificación para cada programa se aplicará (solapamientos entre años principalmente). También permite ajustar cuantos solapamientos para un mismo año podrá haber como máximo.

- **Intervalos preferidos para Tipos de Sesión por Programa**: figura B.6.
  Regula las preferencias de los intervalos por cada programa y año (no deseados, bloqueados o preferidos). Seleccionando y deslizando el ratón se pueden seleccionar los nuevos intervalos (no se permiten solapamientos), y hacer clic en intervalos ya modificados permite editarlos.

- **Preferencias de Aula por Programa**: figura B.7.
  Gestiona las preferencias de aulas a nivel de programa y año. Permite indicar que aulas son preferidas, no deseadas o que están desactivadas.

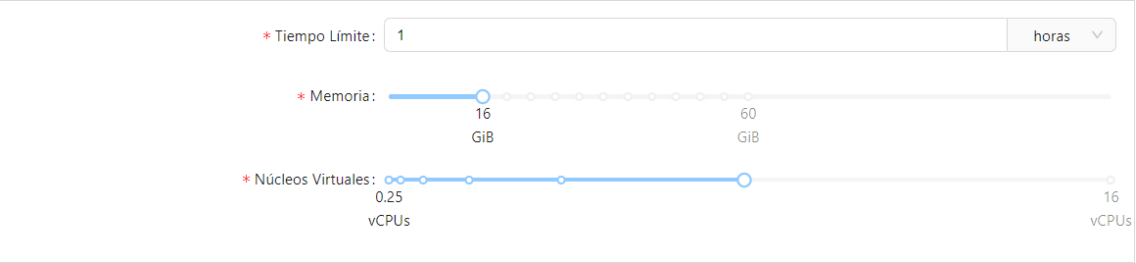

Figura B.1: Opciones para "Ajustes del Entorno de Ejecución"

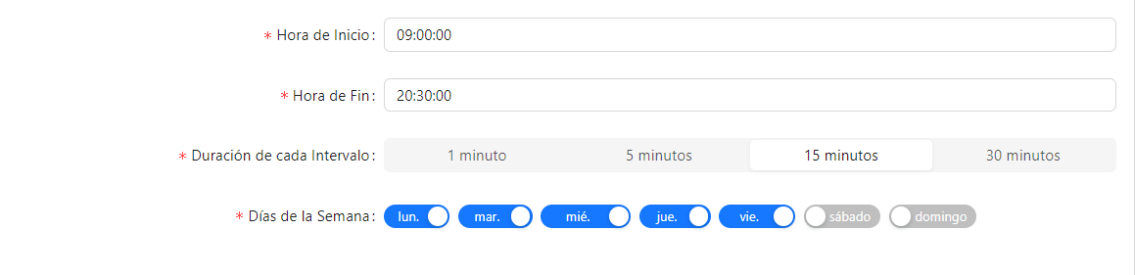

Figura B.2: Opciones para "Ajustes Básicos"

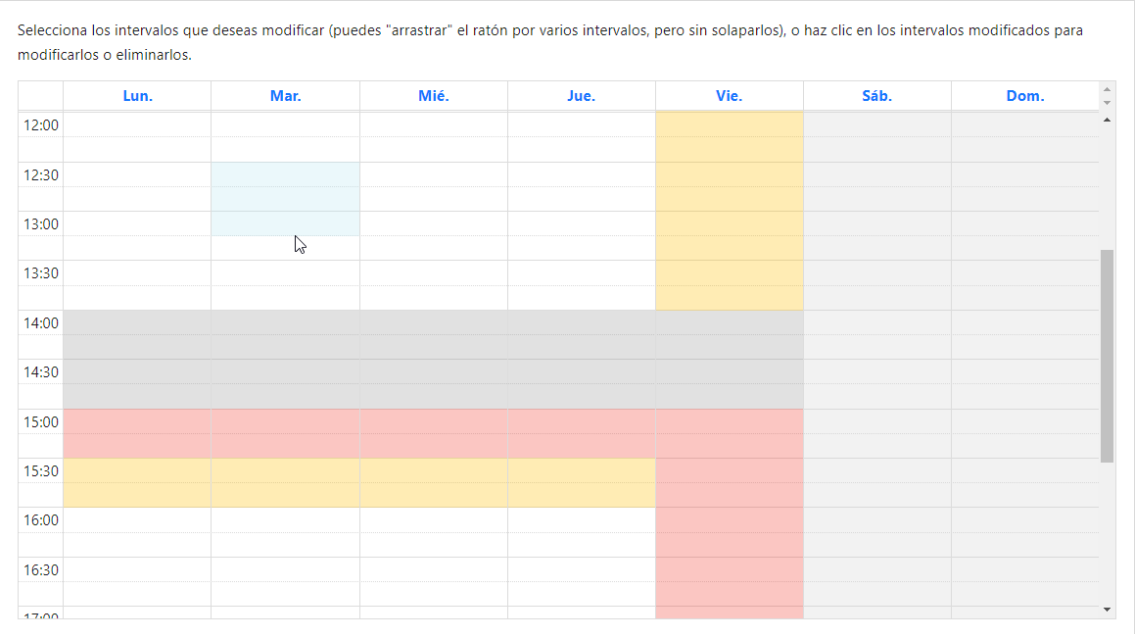

Figura B.3: Opciones para "Preferencia global de Intervalos"

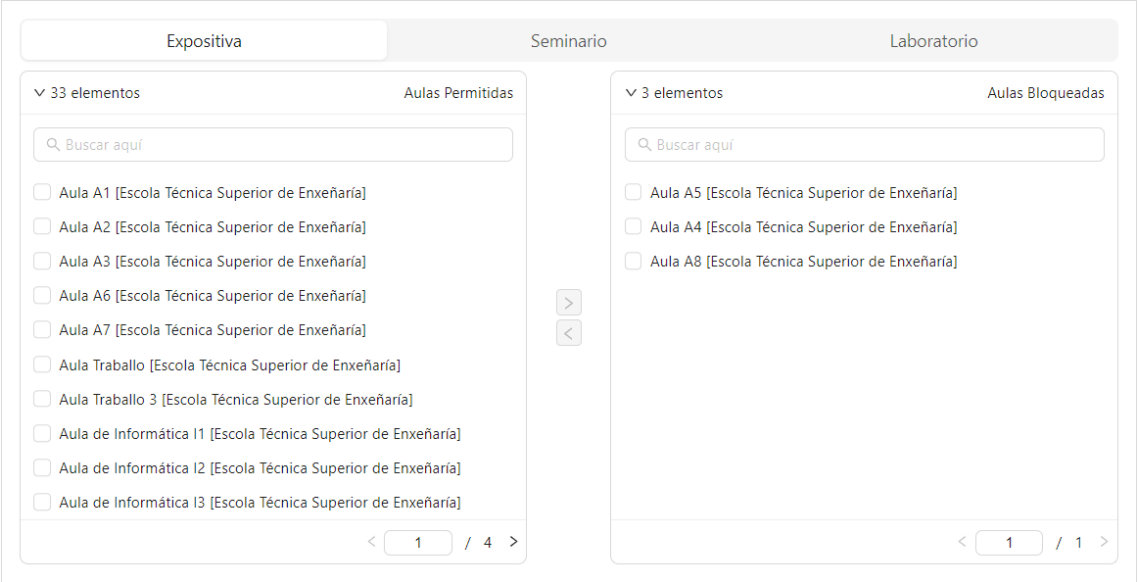

Figura B.4: Opciones para "Tipos de Sesión permitidos por Aula"

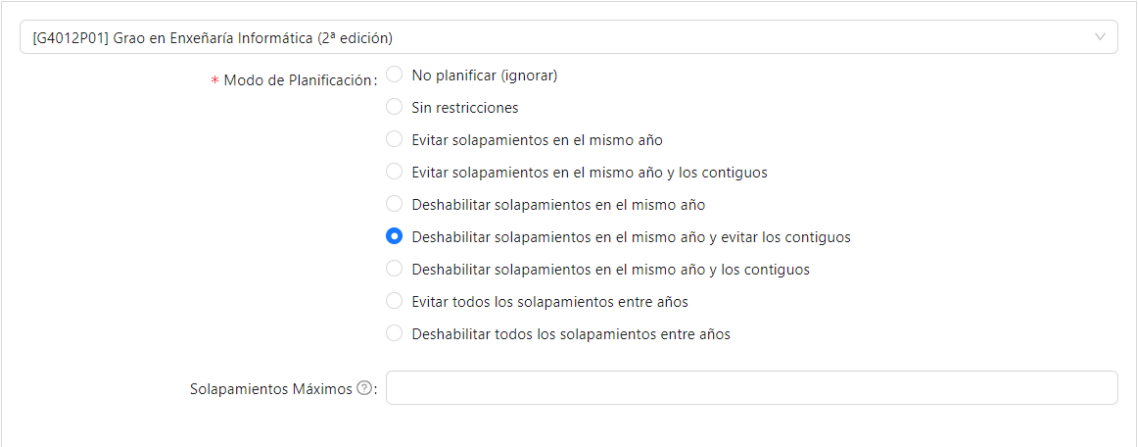

Figura B.5: Opciones para "Modo de planificación para Programas"

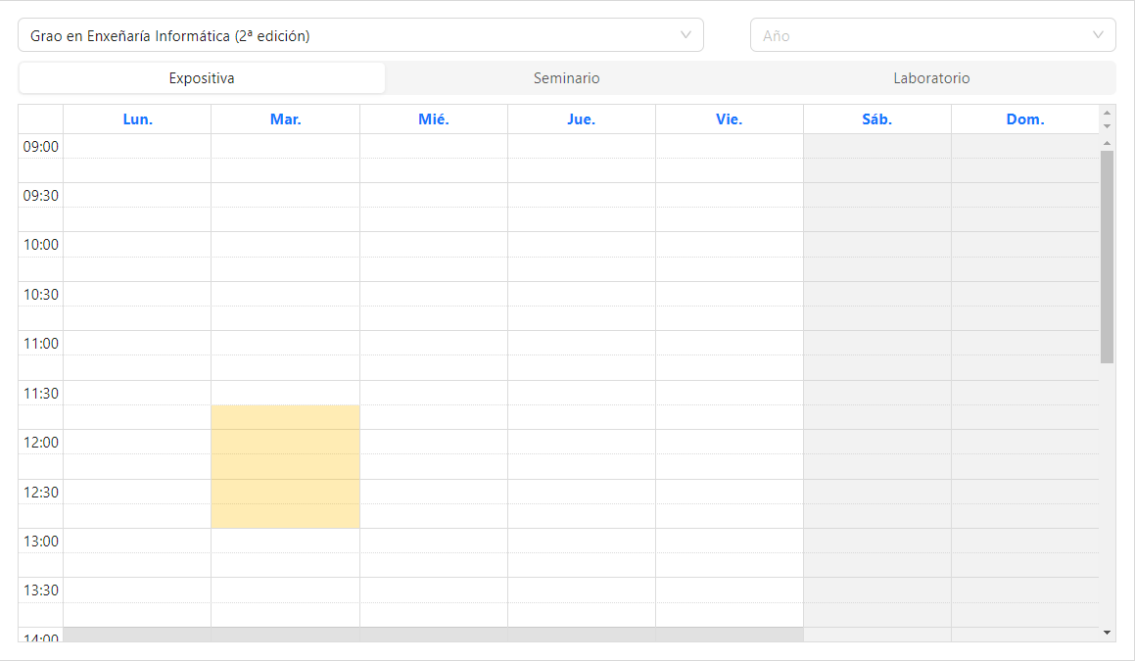

Figura B.6: Opciones para "Intervalos preferidos para Tipos de Sesión por Programa"

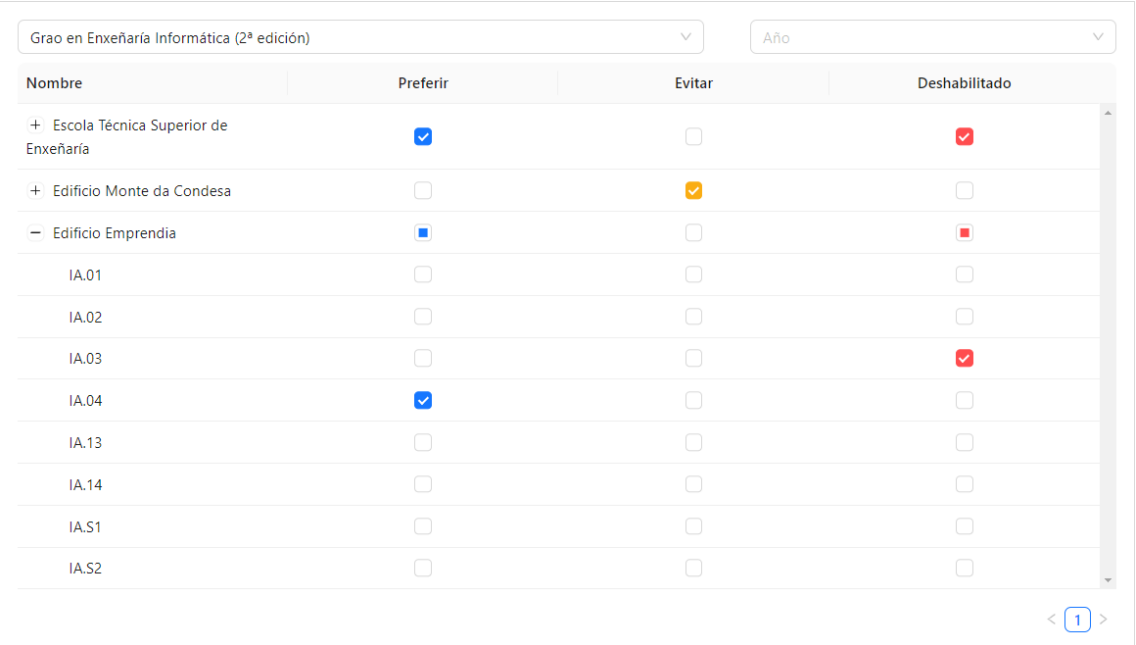

Figura B.7: Opciones para "Preferencias de Aula por Programa"

B.1.2. Visualización de resultados

Una vez el planificador ha acabado, se permiten visualizar los resultados en el paso *Resultado*. En esta pantalla, en caso de haber algún error, se informa del error concreto junto con sugerencias sobre la posible causa.

En caso de ser satisfactorio, por defecto se visualiza un horario semanal con las sesiones programadas. Se puede filtrar por programa, año y grupo programado, para simular exactamente un grupo de docencia. Además, en la esquina superior derecha se puede cambiar a la visualización por ocupación de aula, en la que se puede visualizar que sesiones hay en cada aula en cada momento. Haciendo clic en alguna sesión programada se abre un diálogo con detalles sobre la sesión, su asignación horaria y el aula designada.

B.2. Uso del planificador

B.2.1. Línea de comandos `cli`

```
uso: Planificador ASP [-h] (-e EXECUTIONARN | -f WORKDIR)
     [-t TIMEOUT]

Programa horarios utilizando Clingo ASP

opciones:
  -h, --help            muestra este mensaje y sale
  -e EXECUTIONARN, --executionArn EXECUTIONARN
                        ARN de la Ejecución de la AWS State
                        Machine
  -f WORKDIR, --workDir WORKDIR
                        Directorio de trabajo local con el archivo
                        input.json
  -t TIMEOUT, --timeout TIMEOUT
                        Clingo se detendrá cuando se cumplan estos
                        segundos
```

En el bloque de código B.1 se muestran distintos ejemplos de comandos invocables mediante la herramienta de línea de comandos `cli`:

1. Ejecución en *AWS Batch*.
2. Ejecución en entorno local.

```
1 python cli.py --executionArn arn:aws:states:eu-south
    ↪ -2:500264414701:execution:
    ↪ SfmStateMachineD5EEC6CA-a6aVknixwaGV:2752901c-
    ↪ e499-4316-98b3-a41d7019cd34 --timeout 3600
2 python cli.py --workDir=../data/
```

Bloque de código B.1: Ejemplos de comandos invocables mediante `cli`

B.2.2. Línea de comandos `manual_execution`

```
uso: TFG Timetable Asistente de Ejecuciones Manuales del
     Planificador [-h] {start,retarget,upload,finish} ...

Herramienta de línea de comandos para interactuar con la API
interna y gestionar las ejecuciones manuales del planificador.

opciones:
  -h, --help            muestra este mensaje y sale

modo:
  {start,retarget,upload,finish}
    start               Crea una Ejecución del Planificador Manual
                        y obtiene el input.json
    retarget            Cambia una Ejecución del Planificador
                        normal a una Manual y obtiene el
                        input.json
    upload              Sube los archivos generados (output.json
                        si existe y los asp.txt)
    finish              Sube los archivos generados y marca la
                        Ejecución del Planificador como completada
```

En el bloque de código B.2 se muestran distintos ejemplos de comandos invocables mediante la herramienta de línea de comandos `cli`:

1. Crea una nueva ejecución clonando los datos de otra ya finalizada, y clona el archivo de entrada al directorio de trabajo.

2. Envía los archivos generados hasta el momento.

3. Notifica a la plataforma que la ejecución del propio planificador ha finalizado.

```
1 python manual_execution.py start -s 9d057308-7464-43
    ↪ c3-b35c-877b9dcc4bc7 -a "Prueba API"
2 python manual_execution.py upload
3 python manual_execution.py finish
```

Bloque de código B.2: Ejemplos de comandos invocables mediante `manual_execution`

Modo `start`

En base a un ID de una ejecución ya finalizada dado, genera una copia de la misma y obtiene el archivo de entrada respectivo. Creará un archivo oculto para identificar el ID de ejecución y no necesitar especificarlo en otros comandos.

```
uso: TFG Timetable Asistente de Ejecuciones Manuales del
     Planificador start [-h] -s SOURCE_EXECUTION_ID [-a ALIAS]

opciones:
  -h, --help            muestra este mensaje y sale
  -s SOURCE_EXECUTION_ID, --source-execution-id SOURCEEXECUTIONID
                        Source Scheduler Execution ID from which
                        to retrieve the input files
  -a ALIAS, --alias ALIAS
                        Alias for this Scheduler Execution
```

Modo `retarget`

En base a un ID de una ejecución no iniciada dado, cambia el algoritmo por el de ejecución manual y obtiene el archivo de entrada respectivo. Creará un archivo oculto para identificar el ID de ejecución y no necesitar especificarlo en otros comandos.

```
uso: TFG Timetable Asistente de Ejecuciones Manuales del
     Planificador retarget [-h] -s SOURCE_EXECUTION_ID

opciones:
  -h, --help            muestra este mensaje y sale
  -s SOURCE_EXECUTION_ID, --source-execution-id SOURCEEXECUTIONID
                        Source Scheduler Execution ID to retarget
                        and retrieve the input files
```

Modo `upload`

Habiendo una ejecución manual en el directorio actual, envía a la plataforma los archivos que coincidan con los patrones `output.json` y `asp_*.txt`.

```
uso: TFG Timetable Asistente de Ejecuciones Manuales del
     Planificador upload [-h]

opciones:
  -h, --help  muestra este mensaje y sale
```

Modo `finish`

Habiendo una ejecución manual en el directorio actual, envía los archivos tal y como se indica en el modo `upload` y marca la ejecución del planificador como completada (para que la plataforma verifique el éxito o fracaso de la misma).

```
uso: TFG Timetable Asistente de Ejecuciones Manuales del
     Planificador finish [-h]

opciones:
  -h, --help  muestra este mensaje y sale
```

Apéndice C

Licencia

El código fuente del planificador ASP se distribuye como código abierto bajo licencia GNU General Public License versión 3, a 29 de junio del 2007.

- **TFG-Timetable-Solvers-ASP/**`LICENSE`
- GNU GPL 3.0

El resto del código fuente (los proyectos `frontend`, `backend`, `models` y `cdk`) son código cerrado propietario con todos los derechos reservados por parte de Diego Barreiro Pérez.

Apéndice D

Otras contribuciones

Durante el desarrollo del trabajo, fue necesario realizar diversas contribuciones a otros proyectos para satisfacer las necesidades de las dependencias. A pesar de la posibilidad de poder simplemente "pedir" el cambio, se optó por directamente implementarlo, ya que algún proyecto tiene bajo nivel de desarrollo y era posible que los cambios tardasen en aplicarse.

Los principales cambios realizados fueron:

- `Aluriak/clyngor`: librería usada para invocar Clingo desde Python.
  - *Yield stats on UNSATISFIABLE* (`PR #32`): cambio para poder recibir las estadísticas cuando un problema no fuese satisfacible lógicamente.
  - *Improve API for non-satisfiable problems* (`PR #33`): cambio que permite diferenciar si no se ha encontrado solución por falta de tiempo o por no satisfacible.
- `aws-cloudformation/cloudformation-coverage-roadmap`: solicitudes de cambio para AWS Cloudformation.
  - *Support for provisioning AWS Batch resources in eu-south-2 (Spain) region via CFN* (`Issue #1600`): petición para que se incorporase a la definición JSON de AWS Cloudformation el despliegue de recursos AWS Batch en la región de España.

Apéndice E

Glosario de AWS

Se adjunta este glosario de servicios de *Amazon Web Services* para indicar su funcionalidad.

- ***AWS Smithy***: lenguaje de programación para definir APIs de servicios.
- ***AWS IAM*** *(Identity and Access Management)*: servicio para gestionar roles, políticas y usuarios en lo referente a permisos dentro de la nube.
- ***AWS CloudFormation***: servicio para gestionar el despliegue de recursos en AWS mediante plantillas.
- ***AWS CDK*** *(Cloud Development Kit)*: librería para definir infraestructura de AWS mediante código. Se traduce a *AWS CloudFormation*.
- ***AWS CloudWatch***: servicio para monitorización de registros de AWS.
  - ***AWS X-Ray***: servicio para monitorizar ejecuciones internas de código (funciones concretas, medir latencias, etc.) y ver trazas de llamadas entre servicios de AWS.
- ***AWS VPC*** *(Virtual Private Cloud)*: servicio para definir redes privadas virtuales, aisladas del exterior o para situar componentes con acceso restringido a la red.
- ***AWS Lambda***: servicio de computación *serverless*. Máximo de 15 minutos de ejecución.
- ***AWS Step Functions***: servicio para definir máquinas de estado, integrándose con otros servicios de AWS.
  - ***AWS SFM*** *(State Function Machine)*: maquina de estados de *AWS Step Functions*.

- ***AWS Batch***: servicio de computación en contenedores. Se requieren definiciones de trabajos, definiciones de entornos de computación y colas de trabajo.
    - ***AWS Fargate***: modalidad de *AWS Batch* para ejecutar contenedores *serverless*.
- ***AWS DynamoDB***: servicio de bases de datos no-sql con estructura clave-valor.
- ***AWS S3*** *(Simple Storage Service)*: servicio de almacenamiento estático de archivos.
- ***AWS SQS***: servicio de colas de mensajes (los mensajes se "consumen", copias únicas).
- ***AWS SNS***: servicio de notificación de mensajes (una copia del mensaje por receptor).
- ***AWS Systems Manager*** *(anteriormente AWS SSM o AWS Simple Systems Manager)*: servicio para gestionar parámetros y configuraciones de aplicaciones o sistemas.
- ***AWS Secrets Manager***: servicio para gestionar automáticamente valores secretos (y evitar usar variables de entorno o valores codificados).
- ***AWS API Gateway***: servicio para construir definiciones de APIs en AWS y conectar otros servicios de solicitudes HTTP a solicitudes de APIs de AWS.
- ***AWS CloudFront***: servicio de distribución de contenidos.
- ***AWS Route 53***: servicio para la gestión de nombres de dominio.
- ***AWS SES*** *(Simple Email Service)*: servicio para la distribución de correo masivo.

Apéndice F

Requisitos y pruebas detallados

Se crea este apéndice adicional para detallar los requisitos y pruebas con todos los campos requeridos, con el fin de aligerar la lectura de la memoria.

F.1. Requisitos de la plataforma

F.1.1. Requisitos de información

IR-001	**Almacenar los datos relativos a cada ejecución del planificador.**
Cuando la plataforma invoca el planificador, la plataforma es responsable de almacenar correctamente ciertos datos relativos a la ejecución. Esto implica, por ejemplo, ser consciente de qué periodo se está solicitando o qué año académico es el deseado para la visualización del resultado.	
Datos	ID de ejecución, Algoritmo, Año académico, Periodo, Estado
Estabilidad	Alta
IR-002	**Mantener una copia de los datos usados en la ejecución.**
Debido a que una ejecución puede llevar hasta días, es necesario mantener una copia de los datos originales usados en cada ejecución. Se ha de realizar una "instantánea" de las entidades a utilizar en el momento de crear la ejecución, para evitar que cualquier cambio pueda afectar a la visualización de los resultados y que se creen estados inconsistentes.	
Datos	Edificios y aulas, Programas y asignaturas
Estabilidad	Alta

Continúa en la página siguiente...

Cuadro F.1: Requisitos de información de la plataforma

IR-003	**Almacenar la configuración de la ejecución.**
Se ha de almacenar correctamente la configuración (preferencias y restricciones) de cada ejecución.	
Datos	Ajustes técnicos, Datos básicos, Preferencias, Restricciones
Estabilidad	Alta
IR-004	**Almacenar los resultados del planificador.**
Dado que el planificador sólo tiene la funcionalidad de ser ejecutado, la responsabilidad de almacenar los resultados recae sobre la plataforma web. Es necesario almacenar el resultado recibido para poder ser visualizado posteriormente.	
Datos	Resultado del planificador en bruto, Resultado del planificador procesado, Otros datos recibidos del planificador
Estabilidad	Alta

Cuadro F.1: Requisitos de información de la plataforma

F.1.2. Requisitos funcionales de la interfaz

FR-A001	**Creación y visualización de una ejecución.**
La plataforma deberá permitir crear ejecuciones asociadas a un año académico y periodo especificado, junto con el algoritmo deseado. Además, se deberá permitir seleccionar y visualizar una ejecución que haya sido creada previamente.	
Aceptación	Se permite la creación de ejecuciones y los parámetros indicados son consistentes con los utilizados. Se permite visualizar los datos de la ejecución, junto con la configuración y el estado asociado a la misma.
Precondición	*Ninguna* para creación; la ejecución debe existir para su visualización.
Importancia	Vital
FR-A002	**Lanzamiento de una ejecución.**
Cuando se acabe de configurar las restricciones y preferencias, se permitirá "ejecutar" la ejecución en el planificador.	
Aceptación	Se recibe la ejecución en el planificador y éste la ejecuta.
Precondición	La ejecución existe y no ha sido iniciada en ningún momento.

Continúa en la página siguiente...

Cuadro F.2: Requisitos funcionales de la plataforma

Importancia	Vital
FR-A003	**Transferencia de configuraciones entre ejecuciones.**
Debido a que ciertas preferencias y restricciones pueden ser tediosas de aplicar, se permitirá exportar e importar las configuraciones entre diferentes ejecuciones.	
Aceptación	Se permite exportar la configuración de una ejecución, e importarse en otra siendo el resultado satisfactorio.
Precondición	La ejecución existe (y, para importar la configuración, no se ha iniciado en ningún momento).
Importancia	Media
FR-A004	**Modificación de preferencias y restricciones.**
La plataforma deberá ser capaz de permitir, mediante una interfaz, gestionar las preferencias y restricciones de cada ejecución del planificador.	
Aceptación	Se permite modificar tales preferencias y restricciones, y son guardadas de manera correcta.
Precondición	La ejecución existe y no ha sido iniciada en ningún momento.
Importancia	Vital

Cuadro F.2: Requisitos funcionales de la plataforma

F.1.3. Requisitos no funcionales

NFR-001	**Los componentes de la aplicación serán de tipo *serverless* en la nube de *Amazon Web Services*.**
En vez de depender de un servidor puro y desplegar una aplicación clásica, se utilizarán tecnologías *serverless* con el objetivo de abaratar costes y facilitar el desarrollo. Se opta por la nube de *Amazon Web Services* debido a su facilidad de interconexión en los servicios y simpleza en el despliegue mediante *AWS CloudFormation*.	
Aceptación	La aplicación no requiere de ningún servidor "puro" para ser desplegada, sino que hace uso de componentes en servicios de *Amazon Web Services* sin necesidad de configurar servidores ni conexiones externas.
Importancia	Alta
NFR-002	**Las entidades disponibles estarán limitadas al alcance de la facultad.**

Continúa en la página siguiente...

Cuadro F.3: Requisitos no funcionales de la plataforma

Dado que las ejecuciones son lanzadas a nivel de facultad, las entidades disponibles para las mismas se han de limitar a los recursos que estén disponibles para la facultad. Es decir, que no se podrá programar ninguna sesión en un edificio o de un programa que no tenga vinculación con la facultad.	
Aceptación	Las ejecuciones no utilizan ninguna entidad que no esté previamente asociada a la facultad.
Importancia	Alta

Cuadro F.3: Requisitos no funcionales de la plataforma

F.2. Requisitos del planificador

F.2.1. Requisitos funcionales

`FRS-001`	**Ejecución en distintos entornos.**
El planificador podrá ser ejecutado en diferentes entornos para ganar versatilidad y portabilidad. Los entornos a soportar son: **entorno *serverless***, **entorno de contenedores** y **entorno "local"**.	
Aceptación	El planificador puede ser ejecutado en *AWS Lambda*, *AWS Batch* y el clúster HPC del CiTIUS - *Centro Singular de Investigación en Tecnologías Inteligentes*.
Precondición	*Ninguna*
Importancia	Alta
`FRS-002`	**Ejecuciones con tiempo límite.**
Todas las ejecuciones deberán indicar un tiempo límite para resolver la planificación.	
Aceptación	Las ejecuciones acaban dentro del tiempo establecido.
Precondición	*Ninguna*
Importancia	Alta
`FRS-003`	**Almacenamiento de estadísticas.**
Dado que las ejecuciones son complejas, se deberán almacenar y transferir a la plataforma datos estadísticos sobre Clingo.	
Aceptación	Se escribe un archivo con las estadísticas de Clingo, que posteriormente es mostrado a través de la aplicación.

Continúa en la página siguiente...

Cuadro F.4: Requisitos funcionales del planificador

Precondición	*Ninguna*
Importancia	Baja

Cuadro F.4: Requisitos funcionales del planificador

F.2.2. Restricciones y preferencias

RP-001	**Hora de inicio.** Para el día natural, se permite especificar el primer instante disponible para programar sesiones.	Equivalencia FR-B001
RP-002	**Hora de fin.** Para el día natural, se permite especificar el último instante disponible para programar sesiones.	Equivalencia FR-B001
RP-003	**Días de semana.** Para la semana natural, se permite especificar los días disponibles para programar sesiones.	Equivalencia FR-B003
RP-004	**Duración del intervalo.** Se permite especificar la precisión con la que se programan las sesiones (y, por tanto, cuantos "intervalos" tiene definidos cada día).	Equivalencia FR-B002
RP-005	**Intervalos modificados globalmente.** Se permite especificar ciertos intervalos que o bien están bloqueados o que no son deseables para ser programados.	Equivalencia FR-B004 FR-B005
RP-006	**Tipos de sesión permitidos por aula.** Se permite indicar los tipos de sesión permitidos por cada aula.	Equivalencia FR-B006
RP-007	**Sesiones que no se pueden solapar.** Se permite indicar ciertas parejas de sesiones que no deben colisionar en el tiempo (estar programadas con algún solapamiento temporal).	Equivalencia FR-B007 FR-B008
RP-008	**Sesiones que no deberían coincidir en el tiempo.**	Equivalencia

Continúa en la página siguiente...

Cuadro F.5: Requisitos funcionales de restricciones y preferencias del planificador

	Se permite indicar ciertas parejas de sesiones que no deberían colisionar en el tiempo (estar programadas con algún solapamiento temporal).	FR-B007
RP-009	**Sesiones que deben estar en el mismo aula en caso de ser contiguas.**	Equivalencia
	Se permite indicar ciertas parejas de sesiones que deberán estar en la misma aula en caso de ser contiguas en el tiempo.	FR-B007
RP-010	**Sesiones que deben deben tener en cuenta las distancias.**	Equivalencia
	Se permite indicar ciertas parejas de sesiones que deberán tener en cuenta las distancias entre aulas para ser programadas.	FR-B007
RP-011	**Preferencias de aulas para sesiones.**	Equivalencia
	Se permite indicar ciertas preferencias ("evitar", "preferir" o "no permitir") para las aulas en cada sesión.	FR-B010
RP-012	**Preferencias de intervalos para sesiones.**	Equivalencia
	Se permite indicar ciertas preferencias ("evitar", "preferir" o "no permitir") para intervalos en cada sesión.	FR-B009

Cuadro F.5: Requisitos funcionales de restricciones y preferencias del planificador

F.2.3. Requisitos no funcionales

NFRS-001	**Abstracción sobre el significado de los recursos.**
Ya que el planificador está diseñado para poder ejecutarse de manera independiente a la plataforma, éste se abstraerá de todo significado que puedan tener "facultades", "programas" o incluso "edificios". Se trabajará al más bajo nivel: con "sesiones" y "aulas", y las respectivas preferencias y restricciones.	
Aceptación	La información recibida carece de significado, a excepción de ciertos metadatos entendibles por la plataforma para re-asociar las entidades.
Precondición	*Ninguna*

Continúa en la página siguiente...

Cuadro F.6: Requisitos no funcionales del planificador

Importancia	Alta
NFRS-002	**Conexión con las APIs de *AWS S3*.**
Debido al elevado tamaño de los archivos de entrada y salida, se creará una conexión nativa con las APIs de *AWS S3* para la transferencia de archivos.	
Aceptación	El planificador puede recibir el archivo de entrada de *AWS S3*, escribir el archivo de salida y transferir archivos de datos adicionales para la ejecución indicada, teniendo los permisos adecuados.
Precondición	*Ninguna*
Importancia	Vital

Cuadro F.6: Requisitos no funcionales del planificador

F.3. Pruebas de la plataforma

F.3.1. Pruebas funcionales

FT-A001	**Creación de una ejecución.**
Se comprueba que es posible crear nuevas ejecuciones del planificador.	
Requisito	FR-A001
Validación	Se procede a clicar en el botón `Crear una nueva ejecución`, cubrir los datos requeridos por el diálogo y clicar en el botón `Aceptar`. Al acabar la pantalla de carga, se comprueba como la ejecución ha sido creada con el estado *PENDIENTE DE CREACIÓN* y a los pocos segundos pasa al estado *LISTO*.
Resultado	*Satisfactorio*
FT-A002	**Visualización de una ejecución.**
Se comprueba que es posible ver ejecuciones previamente creadas.	
Requisito	FR-A001
Validación	Se procede a seleccionar alguna ejecución en el selector. Se comprueba como aparecen nuevos elementos en la pantalla con los detalles de la ejecución en cuestión.
Resultado	*Satisfactorio*
FT-A003	**Navegación entre pasos de la ejecución.**

Continúa en la página siguiente...

Cuadro F.7: Pruebas funcionales de la plataforma

Se comprueba que es posible cambiar y visualizar los detalles de los pasos de las ejecuciones.	
Requisito	`FR-A001`
Validación	Se procede a seleccionar alguna ejecución ya completada en el selector. Se comprueba como es posible cambiar entre los pasos de *Configuración*, *Ejecución* y *Resultado*, mostrándose unos elementos distintos relativos a cada uno de los pasos.
Resultado	*Satisfactorio*
`FT-A004`	**Lanzamiento de una ejecución al planificador.**
Se comprueba que es posible iniciar una ejecución con el planificador.	
Requisito	`FR-A002`
Validación	Se procede a seleccionar alguna ejecución ya creada pero sin iniciar en el selector. Se hace clic en el botón `Comenzar ejecución`. A continuación se observa como la ejecución pasa al estado *EJECUTANDO*.
Resultado	*Satisfactorio*
`FT-A005`	**Exportación de la configuración de una ejecución.**
Se comprueba que es posible exportar a un archivo la configuración de alguna ejecución.	
Requisito	`FR-A003`
Validación	Se procede a seleccionar alguna ejecución ya creada en el selector. Se selecciona el paso *Configuración*, y se hace clic en el botón `Exportar / Importar`. En el diálogo abierto, se hace clic en el botón `Exportar esta Configuración`, el cual descarga al ordenador un archivo con extensión `.tcnf`.
Resultado	*Satisfactorio*
`FT-A006`	**Importación de la configuración a una ejecución.**
Se comprueba que es posible importar de un archivo la configuración de otra ejecución.	
Requisito	`FR-A003`
Validación	Se procede a seleccionar alguna ejecución ya creada sin iniciar en el selector. Se hace clic en el botón `Exportar / Importar` y, en el diálogo abierto, se selecciona el archivo previamente exportado. Se comprueba como, tras hacer clic en *Importar*, la configuración ha sido reemplazada a la descargada previamente.

Continúa en la página siguiente...

Cuadro F.7: Pruebas funcionales de la plataforma (continuación)

Resultado	*Satisfactorio*
FT-A007	**Edición del *alias* de una ejecución.**
Se comprueba que es posible cambiar el nombre personalizado de las ejecuciones.	
Requisito	FR-A004
Validación	Se procede a seleccionar alguna ejecución ya creada que no esté en ejecución en el selector. Se prueba a editar el campo de texto que hay visible con el alias, y se comprueba que tras la recarga de la página el *alias* mantiene su último valor.
Resultado	*Satisfactorio*
FT-A008	**Edición de la configuración de una ejecución.**
Se comprueba la persistencia de los cambios en la configuración de las ejecuciones.	
Requisito	FR-A004
Validación	Se procede a seleccionar alguna ejecución ya creada sin iniciar en el selector. Se prueba a editar alguna configuración, y tras la recarga de la página se verifica la persistencia de los cambios.
Resultado	*Satisfactorio*

Cuadro F.7: Pruebas funcionales de la plataforma

F.3.2. Resultados del cuestionario SUS

Preguntas del cuestionario

Las siguientes afirmaciones se planteaban para valorar en una escala de puntuación numérica de 1 a 5:

1. Creo que me gustaría utilizar este sistema con frecuencia.
2. Encontré el sistema innecesariamente complejo.
3. Pensé que el sistema era fácil de usar.
4. Creo que necesitaría el apoyo de un técnico para poder utilizar este sistema.
5. Encontré que las diversas funciones de este sistema estaban bien integradas.
6. Pensé que había demasiada inconsistencia en este sistema.
7. Me imagino que la mayoría de la gente aprendería a utilizar este sistema muy rápidamente.
8. Encontré el sistema muy complicado de usar.

9. Me sentí muy seguro usando el sistema.
10. Necesitaba aprender muchas cosas antes de empezar con este sistema.

Los valores de las respuestas equivalían a los siguientes grados de conformidad con las afirmaciones:

1. Totalmente en desacuerdo
2. En desacuerdo
3. Neutro
4. De acuerdo
5. Totalmente de acuerdo

Distribución de resultados

Se obtuvieron un total de 15 respuestas, desglosadas por pregunta tal y como se muestra en la siguiente tabla:

	1	**2**	**3**	**4**	**5**
Creo que me gustaría utilizar este sistema con frecuencia.	1 (7 %)	0	0	7 (47 %)	7 (47 %)
Encontré el sistema innecesariamente complejo.	5 (33 %)	5 (33 %)	3 (20 %)	1 (7 %)	1 (7 %)
Pensé que el sistema era fácil de usar.	1 (7 %)	0	2 (13 %)	8 (53 %)	4 (27 %)
Creo que necesitaría el apoyo de un técnico para poder utilizar este sistema.	5 (33 %)	8 (53 %)	1 (7 %)	0	1 (7 %)

Continúa en la página siguiente...

Cuadro F.8: Resultados del cuestionario SUS

	1	**2**	**3**	**4**	**5**
Encontré que las diversas funciones de este sistema estaban bien integradas.	0	0	1 (7 %)	4 (27 %)	10 (67 %)
Pensé que había demasiada inconsistencia en este sistema.	11 (73 %)	2 (13 %)	2 (13 %)	0	0
Me imagino que la mayoría de la gente aprendería a utilizar este sistema muy rápidamente.	0	2 (13 %)	2 (13 %)	6 (40 %)	5 (33 %)
Encontré el sistema muy complicado de usar.	5 (33 %)	8 (53 %)	1 (7 %)	0	1 (7 %)
Me sentí muy seguro usando el sistema.	1 (7 %)	0	1 (7 %)	8 (53 %)	5 (33 %)
Necesitaba aprender muchas cosas antes de empezar con este sistema.	5 (33 %)	8 (53 %)	1 (7 %)	0	1 (7 %)

Cuadro F.8: Resultados del cuestionario SUS

El cálculo de la puntuación final del cuestionario se realiza aplicando la siguiente fórmula:

$$X = \sum \text{resultados de las preguntas impares} - 5$$

$$Y = 25 - \sum \text{resultados de las preguntas pares}$$

$$\text{Puntuación SUS} = (X + Y) \times 2{,}5$$

Para los resultados de las preguntas, se aplica la media de las puntuaciones en cada pregunta. En este caso, se tomaban los valores $X = 15{,}8$ y $Y = 18{,}6$, lo que genera la puntuación de 78,50. Esta puntuación se encuentra dentro del percentil 80 de la distribución SUS, siendo este un grado *bueno* de usabilidad.

En la figura F.1 se aprecia una distribución por cuartiles de las respuestas, pudiendo visualizar de mejor forma los posibles aspectos a mejorar.

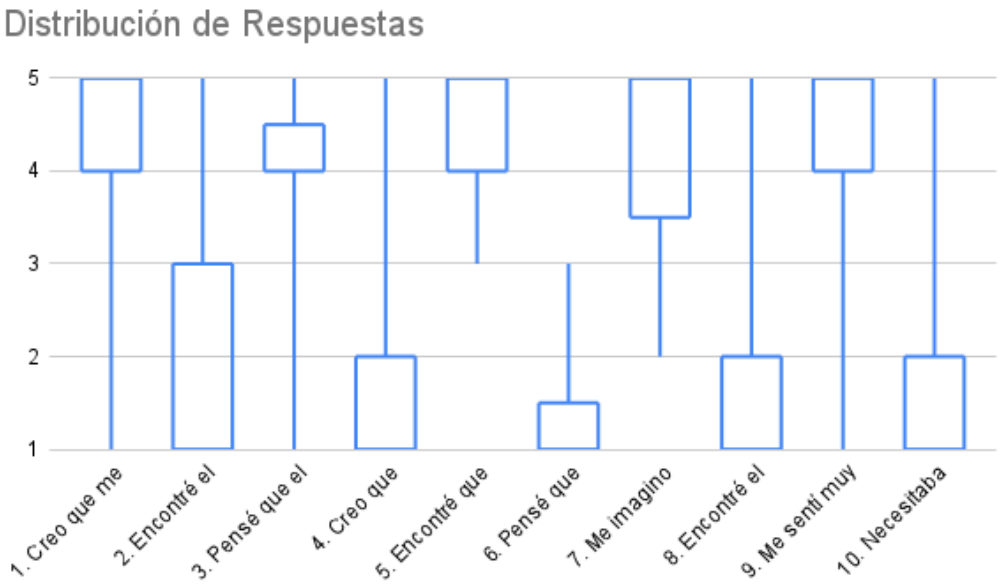

Figura F.1: Distribución de los resultados del cuestionario SUS

F.3.3. Pruebas de la configuración

- `FT-B001` **Cambio en la hora de inicio.**
  Se procede a cambiar el primer momento en el que las sesiones pueden ser programadas.
  Requisitos: `FR-B001`.

- `FT-B002` **Cambio en la hora de fin.**
  Se procede a cambiar el último momento en el que las sesiones pueden ser programadas.
  Requisitos: `FR-B001`.

- `FT-B003` **Cambio de la duración de los intervalos.**
  Se procede a cambiar mediante el selector proporcionado la duración de cada intervalo de sesión.
  Requisitos: `FR-B002`.

- `FT-B004` **Habilitación de un día lectivo.**
  Se procede a activar un día lectivo en la semana, el cual anteriormente estaba sin docencia.
  Requisitos: `FR-B003`.

- `FT-B005` **Deshabilitación de un día lectivo.**
  Se procede a desactivar un día lectivo en la semana, el cual anteriormente estaba con docencia.
  Requisitos: `FR-B003`.

- `FT-B006` **Creación de un intervalo deshabilitado globalmente.**
  Se procede a crear un nuevo intervalo que no permite la programación de sesiones a nivel global.
  Requisitos: `FR-B004`.

- `FT-B007` **Creación de un intervalo no deseado globalmente.**
  Se procede a crear un nuevo intervalo que se debe evitar en la programación de sesiones a nivel global.
  Requisitos: `FR-B005`.

- `FT-B008` **Modificación de un intervalo modificado globalmente.**
  Se procede a cambiar el tipo (distintos niveles de "no deseado", "no deseado" a "bloqueado" y viceversa) de un intervalo modificado globalmente.
  Requisitos: `FR-B004` y `FR-B005`.

- `FT-B009` **Deshabilitación de un aula para un tipo de sesión.**
  Se procede a deshabilitar la programación de sesiones con un cierto tipo de sesión en un aula.
  Requisitos: `FR-B006`.

- `FT-B010` **Habilitación de un aula para un tipo de sesión.**
  Se procede a habilitar la programación de sesiones con un cierto tipo de sesión en un aula.
  Requisitos: `FR-B006`.

- `FT-B011` **Cambio en el modo de planificación de un programa.**
  Se procede a cambiar el modo de planificación de las sesiones de un programa.
  Requisitos: `FR-B007`.

- `FT-B012` **Cambio en el máximo de solapamientos de un programa.**
  Se procede a cambiar el número máximo de sesiones solapadas permitidas para un mismo grupo de un programa.
  Requisitos: `FR-B008`.

- `FT-B013` **Creación de un intervalo modificado de un programa.**
  Se procede a crear un nuevo intervalo modificado para un programa, sin seleccionar el año.
  Requisitos: `FR-B009`.

- FT-B014 **Modificación de un intervalo modificado de un programa.**
  Se procede a cambiar el tipo (entre "evitado", "preferido" y 'no permitido'') de un intervalo modificado para un programa seleccionando el año.
  Requisitos: FR-B009.

- FT-B015 **Creación de un intervalo modificado de un programa en un año.**
  Se procede a crear un nuevo intervalo modificado para un programa, sin seleccionar el año.
  Requisitos: FR-B009.

- FT-B016 **Modificación de un intervalo modificado de un programa en un año.**
  Se procede a cambiar el tipo (entre "evitado", "preferido" y 'no permitido'') de un intervalo modificado para un programa seleccionando el año.
  Requisitos: FR-B009.

- FT-B017 **Añadir una preferencia de aula a un programa.**
  Se procede a alterar un aula para marcarla como "evitable", "preferible" o "no permitida", sin seleccionar el año del programa.
  Requisitos: FR-B010.

- FT-B018 **Eliminación de una preferencia de aula a un programa.**
  Se procede a alterar un aula para desmarcarla de su preferencia seleccionada, sin seleccionar el año del programa.
  Requisitos: FR-B010.

- FT-B019 **Añadir una preferencia de aula a un programa en un año.**
  Se procede a alterar un aula para marcarla como "evitable", "preferible" o "no permitida", seleccionando el año del programa.
  Requisitos: FR-B010.

- FT-B020 **Eliminación de una preferencia de aula a un programa en un año.**
  Se procede a alterar un aula para desmarcarla de su preferencia seleccionada, seleccionando el año del programa.
  Requisitos: FR-B010.

- FT-B021 **Cambio del tiempo límite.** Solo para ejecuciones en *AWS Batch*.
  Se procede a modificar la duración máxima de la ejecución tanto en el valor como en la unidad.

- FT-B022 **Cambio de la memoria disponible.** Solo para ejecuciones en *AWS Batch*.

Se procede a modificar la memoria máxima disponible para el planificador, reflejando los cambios en las opciones de núcleos disponibles.

- `FT-B023` **Cambio de número de núcleos virtuales disponible.** Solo para ejecuciones en *AWS Batch*.
  Se procede a modificar el número de núcleos disponibles para el planificador.

F.4. Pruebas del planificador

F.4.1. Pruebas funcionales

`FTS-001`	**Ejecución en *AWS Lambda*.**
Se comprueba que es posible invocar el planificador en un entorno de *AWS Lambda* dentro de una *AWS Step Function Machine*.	
Requisito	`FRS-001`
Validación	Se crea una ejecución con el algoritmo `ASP-Lambda` y se comprueba como es posible ejecutarla y obtener un resultado.
Resultado	*Satisfactorio*
`FTS-002`	**Ejecución en *AWS Batch*.**
Se comprueba que es posible invocar el planificador en un entorno de *AWS Batch* dentro de una *AWS Step Function Machine*.	
Requisito	`FRS-001`
Validación	Se crea una ejecución con el algoritmo `ASP-Docker` y se comprueba como es posible ejecutarla y obtener un resultado.
Resultado	*Satisfactorio*
`FTS-003`	**Ejecución en entornos externos.**
Se comprueba que es posible invocar el planificador en un entorno externo a AWS.	
Requisito	`FRS-001`

Continúa en la página siguiente...

Cuadro F.9: Pruebas funcionales del planificador

Validación	Se comprueba la existencia de una herramienta de línea de comandos denominada `manual_executions`. Se verifica que etsa herramienta interactúa con una API "privada" (la cual requiere una clave) para recibir y enviar archivos a la plataforma. Se lanza la ejecución mediante la herramienta de línea de comandos `cli`, y se comprueba la correcta interacción entre plataforma web y ejecución local.
Resultado	*Satisfactorio*
`FTS-004`	**Ejecución con tiempo límite.**
Se comprueba que es posible invocar el planificador indicando un tiempo límite de ejecución.	
Requisito	`FRS-002`
Validación	Se comprueba que, mediante la herramienta de línea de comandos `cli`, se permite indicar un parámetro indicando los segundos máximos permitidos para ejecutarse. Se verifica que unos segundos antes de cumplirse, la ejecución ha terminado.
Resultado	*Satisfactorio*
`FTS-005`	**Creación de archivos de depuración.**
Se comprueba que el planificador genera archivos de depuración.	
Requisito	`FRS-003`
Validación	Se comprueba que, tras acabar las ejecuciones, el planificador emite una serie de archivos de depuración ("estadísticas", "optimización", "problema en bruto", etc.).
Resultado	*Satisfactorio*

Cuadro F.9: Pruebas funcionales del planificador

F.4.2. Validaciones de la generación de horarios

`RPT-001`	**Cumplimiento de la hora de inicio.**
Comprobar que los horarios generados no se programan antes de la hora de inicio.	
Requisito	`RP-001` mediante `FR-B001`

Continúa en la página siguiente...

Cuadro F.10: Validación de restricciones y preferencias del planificador

Validación	Se genera una ejecución del planificador de prueba especificando una hora de inicio diferente a la de fin. Se lanza la ejecución y se comprueba que ninguna sesión programada tiene una hora de inicio anterior a la indicada.
Resultado	*Satisfactorio*
`RPT-002`	**Cumplimiento de la hora de fin.**
Comprobar que los horarios generados no se programan después de la hora de fin.	
Requisito	`RP-002` mediante `FR-B001`
Validación	Se genera una ejecución del planificador de prueba especificando una hora de fin diferente a la de inicio. Se lanza la ejecución y se comprueba que ninguna sesión programada tiene una hora de fin posterior a la indicada.
Resultado	*Satisfactorio*
`RPT-003`	**Cumplimiento de la semana lectiva.**
Comprobar que los horarios generados se programan sólo en los días habilitados de la semana.	
Requisito	`RP-003` mediante `FR-B003`
Validación	Se genera una ejecución del planificador de prueba especificando unos días lectivos de la semana aleatorios. Se comprueba que todas las sesiones programadas están dentro de esos días, y ninguna en días no habilitados.
Resultado	*Satisfactorio*
`RPT-004`	**Cumplimiento de la duración del intervalo.**
Comprobar que las sesiones se programan dentro de los intervalos establecidos.	
Requisito	`RP-004` mediante `FR-B002`
Validación	Se genera una ejecución del planificador de prueba indicando un cierto valor de intervalo. Se comprueba que todas las horas de inicio y fin de las sesiones programadas se adecúan al intervalo (por ejemplo, en intervalos de 15 minutos, las sesiones han de empezar y acabar en los minutos :00, :15, :30 o :45).
Resultado	*Satisfactorio*
`RPT-005`	**Cumplimiento de la duración de las sesiones.**

Continúa en la página siguiente...

Cuadro F.10: Validación de restricciones y preferencias del planificador (continuación)

Comprobar que las sesiones se programan usando tantos intervalos como son necesarios.	
Requisito	RP-004
Validación	Se genera una ejecución del planificador de prueba con sesiones de distinta duración. Se comprueba que las sesiones son programadas un número de intervalos contiguos que coinciden con los especificados.
Resultado	*Satisfactorio*
RPT-006	**Cumplimiento de los intervalos bloqueados.**
Comprobar que las sesiones no se programan en intervalos bloqueados globalmente.	
Requisito	RP-005 mediante FR-B004
Validación	Se genera una ejecución del planificador de prueba con ciertos intervalos bloqueados globalmente. Se comprueba que ninguna sesión es programada en tales intervalos.
Resultado	*Satisfactorio*
RPT-007	**Cumplimiento de los intervalos no deseables.**
Comprobar que las sesiones evitan ser programadas en intervalos no deseables globalmente.	
Requisito	RP-005 mediante FR-B005
Validación	Se genera una ejecución del planificador de prueba con ciertos intervalos no deseados globalmente. Se comprueba que, en caso de ser posible, las sesiones son programadas en otros intervalos.
Resultado	*Satisfactorio*
RPT-008	**Cumplimiento de las aulas permitidas.**
Comprobar que las sesiones respetan las aulas que se adecúan a su tipo de sesión.	
Requisito	RP-006 mediante FR-B006
Validación	Se genera una ejecución del planificador de prueba con ciertas aulas que no admiten algunos tipos de sesión. Se comprueba que las sesiones son programadas en aulas que sí que soportan su tipo correctamente.
Resultado	*Satisfactorio*

Continúa en la página siguiente...

Cuadro F.10: Validación de restricciones y preferencias del planificador (continuación)

RPT-009	**Cumplimiento de sesiones que no pueden solaparse.**
Comprobar que las sesiones no son programadas en el mismo instante que otras sesiones.	
Requisito	RP-007 mediante FR-B007
Validación	Se genera una ejecución del planificador de prueba con ciertas sesiones que no pueden solaparse en el tiempo al deshabilitar solapamientos dentro de un mismo programa en el mismo año. Se verifica que esta restricción se respeta, y las sesiones de un mismo año (para este caso) se programan en momentos diferentes.
Resultado	*Satisfactorio*
RPT-010	**Cumplimiento de sesiones que no deberían solaparse.**
Comprobar que las sesiones evitan ser programadas en el mismo instante que otras sesiones.	
Requisito	RP-008 mediante FR-B007
Validación	Se genera una ejecución del planificador de prueba con ciertas sesiones que no deberían solaparse en el tiempo al deshabilitar solapamientos dentro de un mismo programa en años contiguos. Se verifica que esta preferencia se respeta en caso de ser posible, y las sesiones son programadas en momentos diferentes.
Resultado	*Satisfactorio*
RPT-011	**Cumplimiento de sesiones que han de estar en el mismo aula.**
Comprobar que las sesiones contiguas son programadas en el mismo aula.	
Requisito	RP-009 mediante FR-B007
Validación	Se genera una ejecución del planificador de prueba con ciertas sesiones que en caso de ser programadas de manera contigua, han de estar en el mismo aula. Se comprueba el cumplimiento parcial de esta restricción.
Resultado	*Parcialmente satisfactorio*
RPT-012	**Cumplimiento de sesiones que han de tener en cuenta las distancias para programarse.**
Comprobar que las sesiones evitan ser programadas en el mismo instante que otras sesiones.	
Requisito	RP-010 mediante FR-B007

Continúa en la página siguiente...

Cuadro F.10: Validación de restricciones y preferencias del planificador (continuación)

Validación	Se genera una ejecución del planificador de prueba con ciertas sesiones que en caso de ser programadas de manera contigua, han de tener en cuenta la distancia para desplazarse de un aula a otra. Se comprueba que esta restricción no es respetada.
Resultado	*No satisfactorio*
`RPT-013`	**Cumplimiento de las preferencias de las aulas en las sesiones.**
Comprobar que las sesiones respetan las preferencias indicadas para las aulas.	
Requisito	`RP-011` mediante `FR-B010`
Validación	Se genera una ejecución del planificador de prueba con ciertas sesiones indicando aulas deshabilitadas, aulas preferidas y aulas a evitar. Se comprueba que los tres tipos de preferencia para las aulas son respetados.
Resultado	*Satisfactorio*
`RPT-014`	**Cumplimiento de las preferencias de los intervalos en las sesiones.**
Comprobar que las sesiones respetan las preferencias indicadas para los intervalos.	
Requisito	`RP-012` mediante `FR-B009`
Validación	Se genera una ejecución del planificador de prueba con ciertas sesiones indicando intervalos deshabilitados, intervalos preferidos e intervalos a evitar. Se comprueba que los tres tipos de preferencia son respetados.
Resultado	*Satisfactorio*

Cuadro F.10: Validación de restricciones y preferencias del planificador

Apéndice G

Formatos de archivos

Se adjunta este anexo adicional para mostrar ejemplos de archivos utilizados tanto por la plataforma como por el planificador, con el fin de incluír y visibilizar los formatos de almacenamiento y transferencia de datos.

G.1. Archivos usados por la plataforma

1. **Bloque de código G.1**: almacenamiento de la configuración de preferencias y restricciones en la plataforma.
2. **Bloque de código G.2**: almacenamiento de la "instantánea" de los datos de las asignaturas a usar en la ejecución del planificador.
3. **Bloque de código G.3**: almacenamiento de la "instantánea" de los datos de las aulas a usar en la ejecución del planificador.
4. **Bloque de código G.4**: archivo con todas las sesiones unificadas del planificador (similar a `output.json`, pero con una única ocurrencia por sesión, en vez de tantas ocurrencias como huecos horarios utilizados).

```
{
  "dayStart": "09:00:00",
  "dayEnd": "20:30:00",
  "weekDays": [1, 2, 3, 4, 5],
  "slotDuration": 900.0,
  "modifiedSlots": [
    {
      "weekDay": 1,
      "timeframe": {
        "start": "14:00:00",
        "end": "15:00:00"
      },
```

```
      "slotType": "blocked"
    }
  ],
  "disallowedRoomsPerSessionType": {
    "CLE": [
      [
        "2ba481e6-8ddb-4842-b40d-e993024d3298",
        "AT"
      ]
    ]
  },
  "degreesPreferences": {
    "G4021P01": {
      "maxGroupOverlaps": null,
      "yearPreferences": {
        "0": {
          "roomsPreference": {
            "0": [
              [
                "0adfb5b9-5cbb-472f-82a5-94c7a3b3a404",
                "I5"
              ]
            ]
          },
          "preferredTimeslotsPerSessionType": {
            "CLE": [
              {
                "weekDay": 1,
                "timeframe": {
                  "start": "15:00:00",
                  "end": "20:30:00"
                },
                "preferenceType": "0"
              }
            ]
          }
        }
      },
      "schedulingMode": "DSAC"
    }
  },
  "coursesPreferences": {}
}
```


Bloque de código G.1: Ejemplo de archivo `input_config.json`


```
{
  "courses": [
```

```
    {
      "code": "G4012443",
      "name": "Calidade dos Sistemas Informaci\u00f3n",
      "shortName": "CSI",
      "period": "1P",
      "status": "A",
      "degrees": {
        "G4012P01": 4
      },
      "sessionGroups": [
        {
          "discriminator": "d76154bdf39e4817a5a118c9a04d4d4d",
          "sessionType": "CLIL",
          "numPerWeek": 1,
          "numGroups": 1,
          "duration": 10800
        }
      ]
    }
  ],
  "degrees": [
    {
      "code": "G4012P01",
      "name": "Grao en Enxe\u00f1ar\u00eda Inform\u00e1tica (2\u00aa
        ↪ edici\u00f3n)",
      "publicUrl": "https://www.usc.gal/gl/estudos/graos/enxenaria-
        ↪ arquitectura/grao-enxenaria-informatica-2aedicion",
      "level": "B",
      "status": "A",
      "courses": ["G4012101"]
    }
  ]
}
```

Bloque de código G.2: Ejemplo de archivo `input_courses.json`

```
{
  "rooms": [
    {
      "code": "A1",
      "name": "Aula A1",
      "capacity": 135,
      "floor": 3,
      "sector": "Docencia",
      "status": "U",
      "building": "2ba481e6-8ddb-4842-b40d-e993024d3298"
    }
  ],
```

```
  "buildings": [
    {
      "code": "2ba481e6-8ddb-4842-b40d-e993024d3298",
      "name": "Escola T\u00e9cnica Superior de Enxe\u00f1ar\u00eda",
      "publicUrl": "https://www.usc.gal/gl/edificio/escola-tecnica-
          ↪ superior-enxenaria",
      "address": "",
      "status": "H",
      "areas": {
        "floors": [3],
        "sectors": ["Docencia"]
      },
      "distances": {
        "buildings": {
          "0adfb5b9-5cbb-472f-82a5-94c7a3b3a404": 3.0
        },
        "floors": {},
        "sectors": {}
      }
    }
  ]
}
```

Bloque de código G.3: Ejemplo de archivo `input_rooms.json`

```
[
  {
    "slot": {
      "weekDay": 1,
      "timeframe": {
        "start": "10:00:00",
        "end": "11:00:00"
      }
    },
    "session": {
      "id": "c57181bd-ab72-4a83-8234-b16dce7ae641",
      "constraints": {},
      "metadata": {
        "sessionGroup": "d5d382e1e754454690ec45d1064a3371",
        "course": "G4121101",
        "nGroup": 0,
        "nWeek": 2,
        "numGroups": 1,
        "numWeeks": 3
      }
    },
    "room": {
      "id": "fbf93176-3a59-49fa-8142-7bde553e337f",
```

```
      "constraints": {},
      "metadata": {
        "building": "316e42cf-f6bc-4742-a980-e9d10b7628f1",
        "room": "IAS2"
      }
    }
  }
]
```

Bloque de código G.4: Ejemplo de archivo `output_week.json`

G.2. Archivos usados por el planificador

1. **Bloque de código G.5**: archivo de entrada unificado generado en la plataforma y usado por el planificador.
2. **Bloque de código G.6**: archivo de salida generado por el planificador y recibido por la plataforma. *Existe un registro por cada hueco horario usado por cada sesión.*

```
{
  "settings": {
    "dayStart": "09:00:00",
    "dayEnd": "20:30:00",
    "weekDays": [1, 2, 3, 4, 5],
    "slotDuration": 900.0,
    "modifiedSlots": [
      {
        "weekDay": 1,
        "timeframe": {
          "start": "14:00:00",
          "end": "15:00:00"
        },
        "slotType": "blocked"
      }
    ]
  },
  "sessions": [
    {
      "id": "e11a5554-8d43-4c0d-8bd3-d4da5baaa68f",
      "constraints": {
        "sessionType": "CLIL",
        "duration": 7200.0,
        "cannotConflictInTime": ["05013fd2-cbdf-4ee3-906f-016840
            ↪ e26104"],
        "avoidConflictInTime": [],
```

```
        "roomsPreferences": {
          "disallowedRooms": ["224b289b-7bad-4794-a482-104f2fedb4cc"]
              ↪ ,
          "penalizedRooms": [],
          "preferredRooms": []
        },
        "timeslotsPreferences": {
          "disallowedSlots": [
            {
              "weekDay": 1,
              "timeframe": {
                "start": "09:00:00",
                "end": "12:00:00"
              }
            }
          ],
          "penalizedSlots": [],
          "preferredSlots": []
        }
      },
      "metadata": {
        "sessionGroup": "193badd56e5a4f4c84ff1ed84f88b279",
        "course": "G4121103",
        "nGroup": 1,
        "nWeek": 0,
        "numGroups": 2,
        "numWeeks": 1
      }
    }
  ],
  "rooms": [
    {
      "id": "224b289b-7bad-4794-a482-104f2fedb4cc",
      "constraints": {
        "capacity": 135,
        "sessionTypes": ["CLE", "CLIS"],
        "preferredSessionTypes": null,
        "distancesInMinutes": {}
      },
      "metadata": {
        "building": "2ba481e6-8ddb-4842-b40d-e993024d3298",
        "room": "A1"
      }
    }
  ]
}
```


Bloque de código G.5: Ejemplo de archivo `input.json`

```
{
  "timetable": [
    {
      "slot": {
        "weekDay": 1,
        "timeframe": {
          "start": "10:15:00",
          "end": "10:30:00"
        }
      },
      "session": {
        "id": "c57181bd-ab72-4a83-8234-b16dce7ae641",
        "constraints": {},
        "metadata": {
          "sessionGroup": "d5d382e1e754454690ec45d1064a3371",
          "course": "G4121101",
          "nGroup": 0,
          "nWeek": 2,
          "numGroups": 1,
          "numWeeks": 3
        }
      },
      "room": {
        "id": "fbf93176-3a59-49fa-8142-7bde553e337f",
        "constraints": {},
        "metadata": {
          "building": "316e42cf-f6bc-4742-a980-e9d10b7628f1",
          "room": "IAS2"
        }
      }
    }
  ]
}
```


Bloque de código G.6: Ejemplo de archivo `output.json`

G.2.1. Archivos de datos adicionales

1. **Bloque de código G.7**: contiene los datos sobre la optimización del problema.
2. **Bloque de código G.8**: definición del problema para resolver con Clingo.
3. **Bloque de código G.9**: parámetros de los predicados de la solución devueltos por Clingo.
4. **Bloque de código G.10**: datos técnicos de la ejecución.

5. **Bloque de código G.11**: estado de la ejecución de Clingo.

```
bonus           "PreferRoomForSession"  15
    ↪ session_6b9f2d0e76c8463081a315f5a4f9bf2e
bonus           "PreferRoomForSession"  15
    ↪ session_a6c97767ba8f496bba7b44b42ba64bef
bonus           "PreferRoomForSession"  15
    ↪ session_776663b517c64955b6650bceb3a991af
penalty         "UndesirableTimeslot"   10
    ↪ session_b941249aac2e4f84ab7458416668e9af
...
```

Bloque de código G.7: Ejemplo de archivo `asp_optimization.json`

```
timeslot(1..20;25..66;71..112;117..152;163..204;209..230).
...
room(room_224b289b7bad4794a482104f2fedb4cc,135). % A1 | 2ba481e6-8ddb
    ↪ -4842-b40d-e993024d3298
...
session(session_05013fd2cbdf4ee3906f016840e26104,st_CLE,4).% CLE |
    ↪ G4121103 | 5fc3faa3a04548b787fae7b5b2e75f21 | 1-1
...

1 { assignedTimeslot(T,S) : eligibleTimeslotForSession(S,T) } 1 :-
    ↪ session(S,_).
1 { assignedRoom(R,S) : eligibleRoomForSession(S,R) } 1 :- session(S,
    ↪ _).

scheduledSession(T..T+H-1,S,R) :- session(S,H), assignedTimeslot(T,S)
    ↪ , assignedRoom(R,S).

:- not { scheduledSession(T,_,R) } 1, room(R,_), timeslot(T).
:- not { scheduledSession(T,S1,_); scheduledSession(T,S2,_) } 1,
    ↪ noTimeslotOverlapInSessions(S1,S2), timeslot(T).

penalty("UndesirableTimeslot",PC,S,4) :- undesirableTimeslot(T,PC),
    ↪ scheduledSession(T,S,_), PC == 10.
...

#minimize { PC@PP,PN,PV : penalty(PN,PC,PV,PP) }.
#maximize { BC@BP,BN,BV : bonus(BN,BC,BV,BP) }.
#show scheduledSession/3.
#show penalty/4.
#show bonus/4.
```

Bloque de código G.8: Ejemplo de archivo `asp_problem.json`

```
6       session_c57181bdab724a838234b16dce7ae641
   ↪ room_fbf931763a5949fa81427bde553e337f
110     session_4b57094c206d4c58944e05f5afcf9875
   ↪ room_f78687605a6f4ca196a6349984ec1f18
202     session_e11a55548d434c0d8bd3d4da5baaa68f
   ↪ room_612d222a66f34853b12f84fafba0a560
53      session_c28188dd0c7a4833a2b286150990382a
   ↪ room_4dae3e3d536e4e4da9d9ee62ff4d08a2
7       session_c57181bdab724a838234b16dce7ae641
   ↪ room_fbf931763a5949fa81427bde553e337f
...
```

Bloque de código G.9: Ejemplo de archivo `asp_solution.json`

```
TIME LIMIT      1
Models  11+
Optimum unknown
Optimization    50 -150
Calls   1
Time    870.010s (Solving: 869.65s 1st Model: 0.02s Unsat: 0.00s)
CPU Time        869.936s

Choices 24427568
Conflicts       12845330 (Analyzed: 12845330)
Restarts        21306    (Average: 602.90 Last: 12842119)
Model-Level     442.1
Problems        1        (Average Length: 0.00 Splits: 0)
Lemmas  12845330 (Deleted: 12606839)
Binary  21120    (Ratio:   0.16%)
Ternary 1538     (Ratio:   0.01%)
Conflict        12845330 (Average Length:  106.6 Ratio: 100.00%)
Loop    0        (Average Length:    0.0 Ratio:   0.00%)
Other   0        (Average Length:    0.0 Ratio:   0.00%)
Backjumps       12845330 (Average:  1.77 Max: 414 Sum: 22740676)
Executed        12845330 (Average:  1.77 Max: 414 Sum: 22740676 Ratio
   ↪ : 100.00%)
Bounded 0        (Average:  0.00 Max:   0 Sum:      0 Ratio:   0.00%)
Rules   162819   (Original: 65368)
Choice  46
Minimize        2
Atoms   42104
Bodies  48228    (Original: 33833)
Count   1219     (Original: 10791)
Equivalences    67317    (Atom=Atom: 19275 Body=Body: 25139 Other:
   ↪ 22903)
Tight   Yes
Variables       4714     (Eliminated:    0 Frozen: 2698)
```

```
Constraints      34498    (Binary:  86.8% Ternary:   6.9% Other:
    ↪ 6.2%)
```

Bloque de código G.10: Ejemplo de archivo `asp_statistics.json`

```
SATISFIABLE
```

Bloque de código G.11: Ejemplo de archivo `asp_status.json`

Bibliografía

[Ama23] ¿Qué es Amazon DynamoDB? - Amazon DynamoDB. `https://docs.aws.amazon.com/es_es/amazondynamodb/latest/developerguide/Introduction.html`, 2023. (Última vez consultado el 27 de junio del 2023).

[Ant23] Ant Design — The world's second most popular React UI framework. `https://ant.design/`, 2023. (Última vez consultado el 24 de junio del 2023).

[AWS23a] ¿Qué es AWS Batch? - AWS Batch. `https://docs.aws.amazon.com/es_es/batch/latest/userguide/what-is-batch.html`, 2023. (Última vez consultado el 27 de junio del 2023).

[AWS23b] ¿Qué es la AWS CDK? - AWS Cloud Development Kit (AWS CDK) V2. `https://docs.aws.amazon.com/es_es/cdk/v2/guide/home.html`, 2023. (Última vez consultado el 27 de junio del 2023).

[AWS23c] ¿Qué es AWS Lambda? - AWS Lambda. `https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html`, 2023. (Última vez consultado el 27 de junio del 2023).

[BCRT15] Andrea Bettinelli, Valentina Cacchiani, Roberto Roberti, and Paolo Toth. An overview of curriculum-based course timetabling. *Top*, 23:313–349, 2015.

[BET11] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[BIK+19] Mutsunori Banbara, Katsumi Inoue, Benjamin Kaufmann, Tenda Okimoto, Torsten Schaub, Takehide Soh, Naoyuki Tamura, and Philipp Wanko. teaspoon: solving the curriculum-based course timetabling problems with answer set programming. *Annals of Operations Research*, 275(1):3–37, 2019.

[BP21] Diego Barreiro Pérez. Horarios ETSE. `https://horarios.lawebdepepe.com/`, 2021. (Última vez consultado el 20 de junio del 2023).

[Bro13] John Brooke. SUS: a retrospective. *Journal of usability studies*, 8(2):29–40, 2013.

[CiT] CiTIUS. CiTIUS - Cluster de Computación de Altas Prestacións (HPC). `https://wiki.citius.usc.es/es:centro:servizos:hpc`. (Última vez consultado el 29 de junio del 2023).

[Fet] Fetch API - Web APIs — MDN. `https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API`. (Última vez consultado el 30 de junio del 2023).

[Ful23] FullCalendar. `https://fullcalendar.io/`, 2023. (Última vez consultado el 24 de junio del 2023).

[Git] Understanding GitHub Actions - GitHub Docs. `https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions`. (Última vez consultado el 30 de junio del 2023).

[GKKS19] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Potssdam Answer Set Solving Collection. `https://potassco.org/clingo/`, 2019. (Última vez consultado el 19 de junio del 2023).

[Her17] Achini Kumari Herath. Genetic Algorithm For University Course Timetabling Problem. *eGrove*, 2017.

[IBM] What is Operational Excellence? - IBM. `https://www.ibm.com/cloud/blog/delivering-value-through-operational-excellence`. (Última vez consultado el 30 de junio del 2023).

[LHG11] Zhipeng Lü, Jin-Kao Hao, and Fred Glover. Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics*, 17:97–118, 2011.

[LPRD07] Rhydian Lewis, Ben Paechter, and Olivia Rossi-Doria. *Metaheuristics for university course timetabling*. Springer, 2007.

[MEJ13] Nashat Mansour and Hanaa El-Jazzar. Curriculum based course timetabling. *ICNC*, pages 787–792, 2013.

[Mü09] Tomáš Müller. ITC2007 solver description: a hybrid approach. *Annals of Operations Research*, 172(1):429–446, 2009.

[Nie94] Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 152–158, 1994.

[Nie05] Jakob Nielsen. Ten usability heuristics, 2005.

[OfC23] OfCourse Scheduling. `https://ofcourse.com/`, 2023. (Última vez consultado el 21 de junio del 2023).

[Pyd23] Overview — Pydantic. `https://docs.pydantic.dev/latest/`, 2023. (Última vez consultado el 27 de junio del 2023).

[Pyn23] Welcome to PynamoDB's documentation! `https://pynamodb.readthedocs.io/en/stable/`, 2023. (Última vez consultado el 27 de junio del 2023).

[Smi23] Smithy 2.0. `https://smithy.io/2.0/index.html`, 2023. (Última vez consultado el 23 de junio del 2023).

[TFG23] Trabajo Fin de Grado — Universidade de Santiago de Compostela. `https://www.usc.gal/es/estudios/grados/ingenieria-arquitectura/grado-ingenieria-informatica-2aedicion/20222023/grado-18352-17520-2-98085`, 2023. (Última vez consultado el 23 de junio del 2023).

[Wak] WakaTime. WakaTime - Dashboards for developers. `https://wakatime.com/`. (Última vez consultado el 28 de junio del 2023).