ULPGC Universidad de Las Palmas de Gran Canaria | ESCUELA DE INGENIERÍA INFORMÁTICA

Optimización de Turnos mediante modelos matemáticos con el Diseño de una Herramienta Visual para Configuración y Análisis. (ResQPlan/GUI)

TITULACIÓN: Grado en Ciencia e Ingeniería de Datos

AUTORA: Andrea Hernando González

TUTORIZADO POR:

José Miguel Santos Espino

Izzat Sabbagh Rodríguez

Fecha: [6/25]

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/Dª Andrea Hernando González, autor/a del Trabajo de Fin de Título Optimización de Turnos mediante modelos matemáticos con el Diseño de una Herramienta Visual para Configuración y Análisis. (ResQPlan/GUI),
correspondiente a la titulación Grado en Ciencia e Ingeniería de datos, plan 40,
en colaboración con la empresa/proyecto (indicar en su caso) Singular Factory

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida, haciendo constar que

[X] se autoriza / [ ] no se autoriza, la grabación en audio de la exposición y turno de preguntas.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[X] No está previsto.

Y para que así conste firma la presente. (fecha en firma electrónica)

El/la estudiante

Fdo.: _____

A rellenar y firmar **obligatoriamente** por el/la/los/las tutor/a/es/as.
En relación a la presente solicitud, se informa: (firmar donde corresponda)

Positivamente (en caso de detección de copia, esta firma quedará invalidada)	Negativamente (la justificación en caso de informe negativo deberá incluirse en el TFT05)
Fdo.: _____	Fdo.: _____

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

A mi tutor por su dedicación, orientación y por acompañarme en cada etapa de este trabajo.

A mis padres, por estar siempre ahí, por animarme en todo momento y por enseñarme con su ejemplo el valor del esfuerzo. Gracias por ser mi mayor apoyo en cada paso del camino.

A mi hermana por acompañarme en largas sesiones de trabajo y por escuchar, una y otra vez, cada fragmento que necesitaba releer en voz alta.

A mi novio, por acompañarme, aconsejarme y motivarme en cada paso. Gracias por aguantar mis momentos de agobio y por ayudarme a desconectar cuando más lo necesitaba.

Y a Cynthia, mi amiga y compañera de TFG, por todos los momentos que hemos compartido a lo largo de la carrera y por haber llegado juntas hasta el final.

Resumen

ResQPlan es una herramienta de planificación de turnos que surgió a raíz de un caso real: la organización de retenes del cuerpo de Bomberos de La Palma. A partir de ese escenario inicial, se ha construido un sistema capaz de generar planificaciones optimizadas a partir de descripciones en lenguaje natural, que son procesadas por IA generativa y resueltas mediante el *solver* Gurobi. Para facilitar la interacción con el sistema, se ha creado una interfaz web sencilla e intuitiva que permite introducir la descripción del contexto del problema, configurar restricciones y visualizar los resultados de forma clara y dinámica. Su diseño flexible permite adaptarse a diferentes contextos, tales como la gestión de turnos de hospitales, retenes y cualquier otro entorno que gestiona turnos rotativos. Con ello se agiliza la creación de horarios y se reduce la probabilidad de errores en la planificación.

Abstract

ResQPlan is a shift-planning solution born from a real-world challenge: coordinating firefighter rosters for the La Palma Fire Department. From that initial use case, we have developed a system that generates optimized schedules from natural-language descriptions, leveraging generative AI and the Gurobi optimizer. A clean, intuitive web interface lets users specify the problem context, set constraints, and visualize results in a clear, interactive dashboard. Its modular architecture easily adapts to diverse scenarios, whether hospital staffing, fire-crew deployments, or any other rotating-shift environment, streamlining schedule creation while minimizing planning errors.

Índice de contenido

Índice de figuras

1. Introducción

La optimización de turnos en el área de emergencias representa un desafío complejo que va más allá de la simple planificación. Lo que se busca es asignar de manera eficiente tanto los recursos humanos como materiales en contextos caracterizados por su imprevisibilidad, urgencia y alta carga emocional y operativa. Situaciones como desastres naturales, emergencias sanitarias o crisis de gran escala requieren respuestas rápidas y bien coordinadas, donde cada decisión puede tener un impacto directo en la efectividad de la intervención. En muchos casos, una correcta asignación del personal puede marcar la diferencia entre una atención eficaz y una respuesta tardía, e incluso entre salvar o no una vida.

Antiguamente, cuando no se disponían de medios para realizarlo, la planificación de turnos se realizaba manualmente, basándose en la propia experiencia personal o en la práctica diaria. Este método, aunque era útil en entornos reducidos, mostraba ciertas dificultades a la hora de llevarlos a organizaciones más grandes con necesidades operativas más complejas y cambiantes.

Con el avance de la tecnología, la gestión de turnos dio un giro importante. Se introdujeron sistemas informáticos en las organizaciones que permitió dejar atrás esa planificación manual y abrió paso a herramientas que agilizaban y facilitaban dicho proceso. Estas herramientas permitían introducir reglas básicas y gestionar bases de datos con información sobre el personal, horarios y necesidades operativas. Sin embargo, su funcionalidad era todavía limitada, pues no tenían la flexibilidad necesaria para adaptarse a cambios inesperados, o situaciones cambiantes y tampoco permitían la definición de restricciones legales u operativas de forma simultánea.

En la actualidad, el desarrollo de modelos matemáticos de optimización ha transformado por completo la forma en la que se maneja la gestión de turnos. Herramientas como Google OR-Tools, Gurobi o PuLP permiten manejar escenarios complejos, usando múltiples variables y restricciones, desde la cobertura mínima requerida por turno hasta la normativa laboral vigente, para generar soluciones óptimas.

1.1. Motivación

Este proyecto se desarrolla en colaboración con la empresa Singular Factory, que busca dar una solución innovadora para la gestión de turnos, inicialmente orientada al cuerpo de Bomberos de La Palma, pero con la intención de extender su aplicación a otros servicios de emergencias. La motivación inicial consiste en la necesidad de contar con una herramienta de planificación que permita una asignación eficiente de los recursos humanos disponibles.

Una adecuada gestión de turnos no solo mejora la eficiencia operativa, sino que también garantiza la disponibilidad del personal sin sobrecargar a los profesionales, reduciendo así los riesgos tanto para ellos como para las personas asistidas.

Sin embargo, y a pesar de los avances tecnológicos en este ámbito, aún presentan desafíos importantes:

- Se deben respetar las normativas laborales, en cuanto a tiempos mínimos de descanso, límite de horas trabajadas y resto de condiciones establecidas.
- La planificación debe asegurar que todos los turnos estén cubiertos sin sobrecargar a los trabajadores.
- Distribuir los trabajos de forma justa para evitar desequilibrios que puedan causar malestar o conflictos entre trabajadores.

Un estudio publicado por el *National Center for Biotechnology Information (NCBI)* titulado *Optimization of Service Process in Emergency Department Using Simulation* (2022) [5] demuestra cómo la optimización y la simulación de procesos pueden reducir el tiempo de espera en departamentos de emergencia. La aplicación de técnicas de optimización en la planificación de recursos permitió mejorar la eficiencia operativa y reducir los tiempos de respuesta en emergencias.

Además, los problemas derivados de una mala gestión de turnos no son exclusivos en el ámbito de emergencias. Sectores como el sanitario, el transporte o los servicios públicos también se ven afectados. Analizar algunas de estas situaciones reales, permitirá comprender mejor los riesgos que implican una planificación inadecuada.

A continuación, se presentan varios casos reales que muestran las consecuencias de una mala planificación de turnos de trabajo.

1. Hospital de Salamanca (España)[1]: En noviembre de 2024, el personal del Hospital de Salamanca anunció su intención de manifestarse debido a la escasez de personal y las malas condiciones laborales derivadas de una planificación inadecuada de los turnos. Entre los principales problemas se encontraban las guardias excesivas, la falta de respeto por los días de descanso y una insuficiente dotación de personal en turnos de tarde y noche. Estas deficiencias comprometían la calidad asistencial y generaban una sobrecarga laboral en el personal sanitario.

2. Falta de personal en el servicio de bomberos en Burgos[2]: La falta de personal en el servicio de bomberos de Burgos ha provocado un incremento en las horas extra, afectando la calidad de vida de los bomberos y potencialmente su rendimiento en intervenciones críticas. Esta carencia de personal ha resultado en una sobrecarga de trabajo, lo que ha obligado a los bomberos a realizar turnos adicionales para cubrir la demanda. Esta mala gestión de los turnos no solo aumenta la fatiga, sino que también pone en riesgo la eficiencia y seguridad del servicio en momentos de emergencia.

3. Escasez de personal en Valencia[3]: El Sindicato Profesional de Policías y Bomberos (SPPLB) denunció que, debido a la escasez de personal de enfermería en la Unidad Sanitaria de Bomberos, la ambulancia de bomberos no pudo ser activada en una ocasión. Esto obligó a los bomberos y a la víctima de un incendio a esperar una hora hasta la llegada de una ambulancia del SAMU. Este incidente refleja una falta de planificación en la asignación de recursos y turnos, lo que afectó la capacidad de respuesta ante emergencias y puso en evidencia la sobrecarga del personal disponible.

4. Mala planificación en la Empresa Malagueña de Transportes (EMT)[4]: En Málaga, la Empresa Malagueña de Transportes (EMT) ha sido objeto de críticas debido a la deficiente planificación de los turnos de sus conductores, lo que ha resultado en un servicio ineficiente y con retrasos constantes. Los horarios mal estructurados y la falta de personal suficiente han generado estrés entre los trabajadores, afectando tanto su bienestar como la calidad del servicio que se ofrece a los usuarios. Este problema ha sido evidenciado por las quejas de los pasajeros, quienes han señalado la falta de puntualidad y la deficiencia en la cobertura de rutas. La mala organización de los turnos pone de manifiesto la necesidad urgente de optimizar la planificación laboral y aumentar el número de empleados para mejorar la eficacia del servicio y satisfacer las expectativas de los usuarios.

1.2. Herramientas de optimización

Para optimizar la asignación de turnos y la gestión de recursos, existen diversas soluciones basadas en modelos matemáticos que permiten generar planificaciones óptimas. Dichas soluciones incluyen tanto herramientas libres, basadas en librerías de código abierto (Google OR-Tools, PuLP), como paquetes comerciales (Gurobi, CPLEX) que integran algoritmos avanzados de resolución. Todas ellas comparten el objetivo de formalizar restricciones operativas y normativas en un modelo computacional, para luego obtener de forma automática propuestas de planificación que maximicen la eficiencia y garanticen el cumplimiento de requisitos. Estas plataformas se aplican habitualmente en sectores como emergencias, sanidad, transporte y logística, donde la precisión y la rapidez en la toma de decisiones son fundamentales.

Por un lado, las librerías de código abierto ofrecen una mayor flexibilidad de configuración y son accesibles sin coste de licencia, lo que favorece su adopción en proyectos académicos o de pequeña y media escala. Por otro, los solucionadores comerciales aportan un rendimiento superior y capacidades avanzadas de escalabilidad, lo que resulta determinante en escenarios industriales de gran tamaño o con requisitos de tiempo real.

En los apartados siguientes se profundizará en cada una de estas plataformas, valorando su flexibilidad de modelado, facilidad de integración, rendimiento según la escala del problema y coste de licencia o soporte. A partir de dicha comparativa, se justificará la elección de la tecnología más idónea para el proyecto de gestión de turnos en el cuerpo de Bomberos de La Palma.

2. Objetivos

2.1. Objetivos del proyecto

El objetivo principal de este Trabajo de Fin de Grado es desarrollar un sistema de gestión de turnos orientado al sector de emergencias, que permita optimizar la planificación del personal cumpliendo tanto restricciones legales (periodos de descanso, jornada máxima) como operativas (cobertura mínima/máxima, disponibilidad de los trabajadores).

Para ello se plantean los siguientes objetivos:

- Diseñar un modelo de optimización robusto y flexible, capaz de representar un problema de asignación de turnos con múltiples restricciones. Se estudiarán herramientas como Google OR-Tools o Gurobi.
- Implementar un sistema funcional basado en dicho modelo, que permita generar de forma automática el calendario de turnos óptimo respetando los requisitos definidos.
- Desarrollar una interfaz gráfica de usuario que permita interactuar con el sistema de manera intuitiva. La interfaz deberá permitir la configuración de restricciones sin necesidad de conocimientos técnicos, así como la visualización los resultados.
- Validar el prototipo desarrollado con casos de uso reales, empleando datos proporcionados por la empresa colaboradora, en concreto, la planificación de turnos del cuerpo de Bomberos de La Palma. Esta validación servirá para comprobar la aplicabilidad del sistema y su capacidad para adaptarse a escenarios reales.

La fase de desarrollo del sistema básico de asignación se llevará a cabo de manera conjunta con la compañera Cynthia Quintana Reyes. Mi contribución individual se centra en la arquitectura y el diseño de la GUI, así como en la integración final entre el motor de optimización y la capa de presentación.

2.2. Objetivos de aprendizaje

Este proyecto sigue también objetivos personales de aprendizaje relacionados con mi crecimiento profesional.

1. Motor de optimización:
    - Profundizar en técnicas de optimización (programación lineal entera mixta y programación con restricciones) y aplicarlas a un caso de uso real.
    - Evaluar y comparar *solvers* (Google OR-Tools y Gurobi) mediante *benchmarks* de problemas de distintos tamaños, analizando rendimiento, escalabilidad y facilidad de uso.
    - Aprender a interpretar soluciones subóptimas y a ajustar parámetros de *solver* (heurísticas, tolerancias y límites de tiempo) para mejorar la calidad y rapidez de las asignaciones.

2. Desarrollo web
    - Consolidar y ampliar conocimientos en tecnologías web (Flask, HTML, CSS y JavaScript) para construir interfaces de usuario funcionales y atractivas.
    - Integrar de manera eficiente el *back-end* en Python con el *front-end* en Flask, diseñando puntos de acceso y flujos de datos que garanticen una comunicación fluida entre la lógica de negocio y la capa de presentación.

En base a la competencia ED4. Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software en ámbitos de aplicación de la Inteligencia Artificial en Ciencia e Ingeniería de Datos, establecida por la Escuela de Ingeniería Informática (EII), el proyecto aborda:

- Identificación y análisis de problemas en IA y Ciencia de Datos: Justificado mediante el estudio de necesidades y limitaciones del sector de emergencias y el análisis del estado del arte en herramientas de optimización.
- Diseño de soluciones software complejas: Reflejado en la formulación del modelo matemático de programación lineal entera mixta, así como en la arquitectura del sistema de optimización y la GUI.
- Desarrollo e implementación de tecnologías avanzadas: Evidenciado en la implementación del motor de optimización con Gurobi en Python y en la construcción de la interfaz web con Flask, asegurando la integración *back-end–front-end*.
- Verificación y validación de sistemas: Demostrado durante la fase de pruebas funcionales y de rendimiento con datos reales del Cuerpo de Bomberos de La Palma, ajustando y refinando el prototipo.
- Documentación y comunicación efectiva: A través de manuales técnicos y de usuario, reportes de *benchmarks* y presentaciones profesionales que recogen decisiones de diseño y resultados obtenidos.

2.3. Propuesta de valor

El sistema de gestión de turnos desarrollado en este Trabajo de Fin de Grado ofrece una solución completa y adaptable a cualquier organización que necesite planificar recursos humanos con altos requisitos de complejidad. Partiendo de un prototipo validado con datos reales del Cuerpo de Bomberos de La Palma, la herramienta se adapta igualmente a centros educativos, unidades sanitarias (enfermería, retenes médicos), operaciones logísticas o equipos industriales.

En su núcleo, un motor de optimización basado en programación lineal entera mixta genera asignaciones que respetan tanto restricciones legales (períodos de descanso, jornada máxima), como operativas (cobertura mínima y máxima, turnos de alta especialización y preferencias individuales).

La interfaz de usuario, independiente del motor de optimización, permite definir el contexto de planificación y sus restricciones, visualizar los resultados y generar la programación, sin requerir conocimientos técnicos en optimización.

Esta herramienta no solo optimiza la asignación de turnos, sino que contribuye a mejorar el bienestar del personal, a reducir costes operativos y a aumentar la fiabilidad de los servicios.

En definitiva, este sistema de gestión de turnos plantea una propuesta de valor única: un motor optimizador potente y una interfaz amigable que, en conjunto, automatizan procesos, incrementan la eficiencia y garantizan la adaptabilidad a cualquier contexto organizativo.

3. Estructura de la memoria

La memoria se estructura en diferentes bloques temáticos que permiten recorrer de forma ordenada todo el proceso de desarrollo del proyecto, desde su justificación inicial hasta las conclusiones.

En primer lugar, se presenta una justificación del trabajo, donde se expone la motivación que ha llevado a realizarlo y se presentan las herramientas de optimización que se han considerado como punto de partida.

A continuación, se definen los objetivos del proyecto, tanto a nivel técnico como de aprendizaje personal, y se expone la metodología y planificación seguida, detallando las herramientas empleadas, el plan de trabajo inicialmente propuesto y el cronograma realmente ejecutado, incluyendo las adaptaciones realizadas a lo largo del proceso.

El bloque central corresponde al desarrollo del proyecto, que se divide en dos partes diferenciadas:

- El trabajo conjunto, desarrollado en colaboración con Cynthia Quintana Reyes, donde se presentan y comparan las principales herramientas de optimización existentes, además de otras soluciones existentes en el mercado. También se describe la selección y comparativa de herramientas de optimización, el diseño del modelo de planificación, su aplicación al caso de gestión de retenes en La Palma, los resultados preliminares obtenidos y su posterior generalización a otros escenarios de turnos rotativos.
- El trabajo individual, centrado en la definición de los requisitos funcionales y no funcionales del producto, el diseño del prototipo, la arquitectura del sistema, el desarrollo de la interfaz web y la integración del motor de optimización.

En las siguientes secciones, se presenta el producto final, acompañado de validaciones y estudio de resultados obtenidos. También se describe el proceso de despliegue y explotación del sistema, orientado a trasladar la solución desde un entorno local a uno de producción, incluyendo aspectos técnicos como la configuración de la infraestructura, la base de datos, la licencia de Gurobi, la seguridad y el mantenimiento.

Además, se incorpora una sección dedicada a la aportación individual de Cynthia, donde se describe su trabajo dentro del proyecto.

Finalmente, se exponen las conclusiones generales del trabajo y se proponen posibles líneas de mejora y evolución futura.

Por último, en los anexos se incluye una guía específica para la instalación y configuración de la licencia de Gurobi.

4. Metodología y planificación

4.1. Metodología utilizada

El desarrollo del proyecto sigue una metodología iterativa e incremental. Esta metodología permite construir el sistema de forma progresiva, incluyendo nuevas funcionalidades por fases y validando de forma continua el funcionamiento del sistema. Este enfoque hace más fácil la adaptación a posibles cambios y la detección de errores para su corrección.

El proceso se estructura en las siguientes etapas principales:

1. Investigación previa:
    - Estudio de soluciones ya existentes en el mercado relacionadas con la planificación de turnos/horarios
    - Análisis de herramientas y librerías de optimización como Google OR-Tools y Gurobi
2. Desarrollo del modelo de optimización: retenes de La Palma
    - Recopilación de datos y restricciones reales del caso de retenes
    - Implementación del modelo en Python usando Gurobi
    - Validación del modelo con datos reales
3. Generalización a otros contextos
    - Parametrización del modelo para permitir otros usos en diferentes contextos que no sean solo retenes
4. Pruebas y validación del modelo
    - Comprobación de la calidad de las soluciones obtenidas
    - Evaluación de tiempos de ejecución para distintos volúmenes de datos
5. Definición de requisitos
    - Definición y documentación de requisitos funcionales y no funcionales del sistema
    - Identificación de necesidades del usuario y especificaciones del producto
6. Diseño de la interfaz
    - Elaboración de prototipo de la interfaz usando la herramienta Figma, con el objetivo de que la interfaz sea lo más clara e intuitiva posible
7. Implementación
    - Desarrollo de la aplicación web Flask y conexión con el motor de optimización
    - Control de versiones con Github y despliegue local para pruebas de integración
8. Pruebas finales
    - Verificación del correcto funcionamiento del sistema completo (modelo + interfaz)
    - Revisión del cumplimiento de requisitos y evaluación de la experiencia de usuario

Finalmente, gracias al enfoque iterativo e incremental elegido, el proyecto ha avanzado de manera estructurada y adaptable desde un producto mínimo, centrado en la generación

de turnos para los retenes de La Palma, hasta una solución plenamente parametrizada y generalizable. Tras validar el modelo de optimización con datos reales y refinar sus tiempos de ejecución, se procedió al diseño de una interfaz clara e intuitiva en Figma y a la implementación de la aplicación web en Flask, conectando el motor de Gurobi con el *front-end*. El ciclo continuo de pruebas ha permitido corregir errores y garantizar el cumplimiento de los requisitos funcionales y no funcionales definidos. Como resultado, se dispone de un sistema completo, desplegado localmente para pruebas finales, capaz de generar horarios óptimos en distintos contextos y con una experiencia de usuario satisfactoria. Este método facilita, además, la incorporación futura de nuevas reglas de negocio, mejoras en el rendimiento y evolución de la interfaz para atender a nuevos ámbitos de aplicación.

4.2. Herramientas utilizadas

1. Lenguajes de programación
    - Python: lenguaje principal para la implementación del modelo de optimización y *back-end* de la aplicación web
    - HTML, CSS, JavaScript: utilizados para el desarrollo del *front-end*, para estructurar y dar estilo a la interfaz web
2. Frameworks y bibliotecas
    - Flask: Microframework de Python que sustenta la arquitectura monolítica de la aplicación, gestionando tanto la lógica de negocio como la presentación de la interfaz de usuario.
    - Gurobi: un *solver* de optimización utilizado para la resolución del modelo de turnos
    - Google OR-Tools: estudiado y evaluado en la fase de investigación como alternativa para la implementación del modelo
3. Herramientas de desarrollo y prototipado
    - Pycharm: editor de código usado a lo largo del proyecto
    - Figma: herramienta de diseño y prototipado de la interfaz
4. Base de datos
    - MongoDB: almacenar los datos de la configuración y restricciones
5. Control de versiones
    - Github: plataforma para la gestión de versiones, colaboración y mantenimiento del repositorio
6. Documentación
    - Microsoft Word: redacción de la memoria

4.3. Plan de trabajo planteado

A continuación, se detalla el plan de trabajo planteado inicialmente para este proyecto, estructurado por meses. El desarrollo se divide en una primera fase centrada en el modelado y la optimización matemática, que constituye la base común del sistema, y una segunda fase centrada en el diseño y desarrollo de una interfaz de usuario que permita su

uso por parte de perfiles no técnicos, siendo esta una parte especialmente relevante y diferenciadora del proyecto.

Mes 1

- Reunión con empresa para concretar y ver proyecto
- Reunión con tutor
- Investigación del estado del arte, identificando soluciones similares ya existentes en el mercado, así como herramientas y metodologías utilizadas en el ámbito de la optimización de horarios y turnos. Investigación de herramientas y librerías de optimización
- Estudio de herramientas y librerías de optimización (como Google OR-Tools, Gurobi o similares) con el fin de identificar cuál se adapta mejor a las necesidades del proyecto.

Mes 2

- Evaluación práctica de las herramientas de optimización estudiadas previamente, a través de pruebas exploratorias, valorando criterios como flexibilidad, eficiencia y facilidad de integración.
- Análisis detallado de restricciones, formalizando condiciones como el límite de horas por jornada, los descansos obligatorios, las rotaciones equitativas, y otros criterios operativos.
- Inicio del desarrollo del modelo matemático adaptado específicamente al caso de los retenes, con especial atención a la representación formal de restricciones y objetivos.
- Validación del modelo con datos reales y pruebas de ajuste

Mes 3

- Elaboración de prototipos, definición de requisitos funcionales y no funcionales
- Desarrollo de un primer prototipo de interfaz que permita introducir restricciones de manera intuitiva y visualizar soluciones
- Pruebas iniciales de la interfaz de usuario con escenarios simples, para detectar errores, mejorar la usabilidad y recoger feedback.
- Implementación de la conexión entre el modelo matemático y la interfaz, garantizando una comunicación fluida:
    - Entrada de datos desde la interfaz al motor de optimización.
    - Visualización de las soluciones generadas, con representación clara de los horarios resultantes.

Mes 4

- Incorporación de funcionalidades adicionales:
    - Edición de restricciones ya introducidas
    - Exportación de horarios generados

- Refinamiento del diseño visual y de la experiencia de usuario, aplicando mejoras derivadas del feedback recibido por parte de la empresa o usuarios de prueba.
- Pruebas integradas del sistema completo (modelo + interfaz), asegurando la coherencia entre los datos introducidos y las soluciones obtenidas.
- Documentación técnica y funcional del sistema desarrollado.

4.4. Plan de trabajo ejecutado

La ejecución del Trabajo de Fin de Grado se ha estructurado siguiendo un plan dividido en varios meses, con el objetivo de desarrollar un sistema de gestión de turnos optimizado, adaptado inicialmente al caso real de los retenes de emergencias en La Palma.

Mes 1

El proyecto comenzó con reuniones con el tutor académico y con la empresa colaboradora, con el fin de concretar los objetivos y definir los primeros pasos. Se investigaron soluciones existentes en el mercado y se realizó un estudio del estado del arte, especialmente centrado en metodologías de optimización aplicadas a la planificación de horarios. Paralelamente, se analizaron distintas librerías de optimización (como Google OR-Tools y Gurobi), valorando sus capacidades y limitaciones de cara a su posible uso en el desarrollo del modelo.

Mes 2

Al inicio del segundo mes aún no disponíamos de la información específica sobre los retenes de emergencias en La Palma, por lo que para poner a prueba las herramientas de optimización se empleó como caso de uso provisional la planificación de horarios en un centro escolar. Con este escenario de colegio se evaluaron en profundidad Google OR-Tools y Gurobi, realizando pruebas exploratorias de rendimiento, flexibilidad y facilidad de integración.

Una vez recibidos los datos detallados de la empresa sobre el caso de los retenes, se procedió a un nuevo análisis del problema para la identificación de las restricciones principales: límites de horas por jornada, descansos obligatorios, equidad en las rotaciones y prioridades operativas. A continuación, se adaptó el modelo inicialmente planteado para el entorno escolar al caso de estudio real de los retenes, ajustando su formulación y validando su correcto funcionamiento con los datos proporcionados. Una vez desarrollado para este caso concreto, se parametrizó de forma genérica para que pudiera configurarse y utilizarse por cualquier grupo de retenes con distintas estructuras organizativas.

Mes 3

La empresa solicitó que el sistema pudiera generalizarse a otros casos de uso además de los retenes de emergencias. Para ello, estudiamos varios escenarios (horarios escolares, servicios sanitarios, turnos policiales, etc.) y analizamos los puntos en común, tipos de restricciones, estructuras de turnos, criterios de equidad, que nos permitieran diseñar una solución parametrizable y adaptable a distintas organizaciones.

Durante este mes se desarrolló un primer prototipo de la interfaz gráfica, centrado en facilitar la introducción de restricciones y la visualización de soluciones. Se definieron los requisitos funcionales y no funcionales, y se llevaron a cabo pruebas de usabilidad con datos simulados para validar el diseño inicial. A continuación, se implementó la conexión entre la interfaz y el motor de optimización, de modo que el usuario pudiera introducir datos en la UI y obtener una representación visual instantánea de los horarios generados por el modelo.

Mes 4

En la cuarta fase se potenció la interfaz con un conjunto de funcionalidades avanzadas. Se añadió la edición dinámica de restricciones, que permite al usuario modificar parámetros sobre la marcha y ver inmediatamente su impacto; validaciones en tiempo real para asegurar la coherencia de las condiciones introducidas; y opciones de exportación de los horarios generados en varios formatos. Asimismo, se incorporó la gestión de proyectos o escenarios personalizados, de modo que distintos grupos de trabajo pudieran guardar y recuperar configuraciones específicas. Para mejorar la calidad de la entrada de datos, se integró un sistema de control basado en ingeniería de prompts, que realiza una validación de contexto (comprobando que el texto hace referencia efectivamente a un problema de planificación de turnos y que las restricciones son coherentes) y una extracción automática de posibles restricciones presentes en el enunciado. Estas novedades se fueron depurando a lo largo de pruebas integradas, donde también se refinó el diseño visual y la experiencia de usuario atendiendo al feedback recibido. Finalmente, se llevaron a cabo ensayos exhaustivos del sistema completo (modelo generalizado + interfaz), garantizando la adecuada comunicación entre ambos componentes y validando el flujo de trabajo de principio a fin.

Mes 5

En el quinto mes se añadieron las últimas mejoras orientadas a la usabilidad y al soporte al usuario. Se implementaron botones de ayuda contextual junto a cada sección de la interfaz, que ofrecen información sobre el propósito de cada parámetro y ejemplos de uso. Se reforzó el sistema de feedback, mostrando mensajes y sugerencias en tiempo real cuando las restricciones o los datos introducidos presentaban inconsistencias o podían optimizarse. Finalmente, se completó la documentación funcional y técnica del proyecto, incluyendo guías paso a paso para la instalación, configuración y explotación de la herramienta, de modo que tanto usuarios finales como futuros desarrolladores dispongan de toda la información necesaria para el mantenimiento y la evolución del sistema.

5. Trabajo conjunto

5.1. Estado del arte

En el campo de la optimización, existen diferentes herramientas y librerías, tanto comerciales como de código abierto, diseñadas para resolver desafíos en áreas como la planificación de recursos, la logística, la gestión de horarios y la asignación eficiente de turnos. En este apartado se describen algunas de las herramientas estudiadas.

Google OR-Tools

OR-Tools[5] es una biblioteca de código abierto desarrollada por Google que está pensada para resolver problemas combinatorios y de optimización. Este tipo de problemas aparecen cuando hay que encontrar una solución óptima teniendo en cuenta una serie de restricciones. Algunos ejemplos típicos son: planificación de turnos, reparto de rutas, asignación de recursos, organización de horarios…

Una de las grandes ventajas de OR-Tools es su flexibilidad: permite definir el problema en diversos lenguajes de programación (como Python, C++, Java o C#) y utilizar distintos solucionadores (solvers), tanto comerciales como de código abierto. Entre estos solucionadores se incluyen:

- CP-SAT (*Constraint Programming - Satisfiability*): solucionador propio de Google, especialmente eficaz en problemas complejos con muchas restricciones.
- GLOP: solucionador de programación lineal desarrollado por Google.
- SCIP y GLPK: solucionadores de código abierto muy utilizados en el ámbito académico e industrial.
- Gurobi y CPLEX: solucionadores comerciales de alto rendimiento que se pueden integrar con OR-Tools si se dispone de licencia.

Gurobi

Gurobi[7] es uno de los solucionadores comerciales más potentes y populares en la actualidad para resolver problemas de optimización matemática. Es capaz de resolver una gran variedad de problemas como la programación lineal (LP), la programación entera y mixta (ILP/MILP), la programación cuadrática (QP y MIQP) y problemas con múltiples restricciones.

Su alto rendimiento lo ha convertido en una herramienta muy popular tanto en el mundo académico como en el industrial, sobre todo en situaciones donde se requiere encontrar soluciones óptimas de forma rápida y eficiente para problemas de gran tamaño y complejidad.

Una de sus características más destacadas es su facilidad de integración con diferentes lenguajes de programación como Python, C++, Java, .NET o MATLAB. Esto permite que pueda adaptarse a distintos entornos de trabajo y lo hace accesible para un amplio perfil de usuarios.

Gurobi requiere una licencia comercial para su uso en entornos profesionales. Sin embargo, ofrece licencias gratuitas para uso académico, lo que permite a estudiantes e investigadores utilizar todas sus funcionalidades sin coste, facilitando su uso en proyectos de investigación y docencia.

Actualmente, el 70 % de las principales empresas tecnológicas del mundo utilizan Gurobi para integrar capacidades de optimización en sus soluciones, y el 85 % de las compañías de la lista Fortuna 500 aplican optimización matemática en sus operaciones[16].

Estos son algunos ejemplos de su uso:

- Suzano, el mayor productor mundial de pulpa de papel, utiliza Gurobi para optimizar su cadena de suministro mediante modelos desarrollados en Python. Gracias a esta implementación, ha conseguido reducir un 60 % el tiempo dedicado al manejo de datos, disminuir el consumo de combustible y detectar oportunidades de ahorro por unos 76 millones de reales[14].
- Querques Sustainable Packing, una empresa española dedicada al reciclaje y reparación de palés ha mejorado la eficiencia de sus rutas de transporte gracias a un sistema basado en Gurobi. Esta solución, desarrollada por Belerofonte, tiene en cuenta factores como la capacidad de los vehículos, los tiempos de entrega o el volumen de carga, permitiendo optimizar recursos, llegar a más clientes y reducir costes logísticos[15].

PuLP

PuLP[8] es una biblioteca de optimización escrita en Python que permite modelar y resolver problemas de programación lineal (LP) y programación entera (ILP) de forma sencilla e intuitiva. Está pensada para facilitar la formulación de modelos matemáticos sin necesidad de utilizar una sintaxis compleja o software especializado.

Una de sus principales ventajas es su integración con distintos solucionadores externos, tanto de código abierto como comerciales. Entre los más comunes están:

- CBC (Cuino branch and cut): solucionador de código abierto que se utiliza por defecto en PuLP.
- GLPK: otro solucionador open source, útil para problemas lineales y enteros.
- CPLEX y Gurobi: solucionadores comerciales de alto rendimiento, que pueden integrarse si se tiene licencia.

Aunque no incluye su propio solucionador, PuLP actúa como un puente entre el usuario y estos motores de optimización, facilitando la construcción del modelo, la exportación a formato LP estándar y la interpretación de resultados.

Gracias a su sintaxis clara y a su integración con Python, PuLP es una herramienta útil en educación, investigación y proyectos donde la simplicidad y la transparencia del código son prioritarias. También puede utilizarse en proyectos reales de pequeña o mediana

escala, siempre que el rendimiento del solucionador externo sea lo bastante eficiente para el tamaño del problema.

Características	OR-Tools	Gurobi	PuLP
Tipo	Biblioteca de optimización (open source)	Solucionador comercial de alto rendimiento	Interfaz de modelado (open source)
Lenguajes compatibles	Python, C++, Java, C#	Python, C++, Java, .NET, MATLAB	Python
Modelado de restricciones	Muy flexible y completo	Muy potente	Intuitivo y fácil
Facilidad de uso	Requiere curva de aprendizaje	Alta si se conoce el entorno	Fácil de aprender
Licencia	Gratuita	Comercial (gratuita para uso académico)	Gratuita
Casos de uso típicos	planificación de turnos, rutas de reparto, programación de tareas, asignación de recursos	optimización logística, planificación de producción, diseño de redes, asignación de personal, gestión de inventarios, trading financiero	proyectos educativos, problemas académicos, prototipos, análisis exploratorios, modelos de pequeña y mediana escala
Escalabilidad	Alta (según solucionador escogido)	Muy alta (ideal para grandes volúmenes de datos)	Media (depende del solucionador)

Tabla 1: Comparativa herramientas de optimización

En vista de esta comparativa, Google OR-Tools y Gurobi se presentan como las herramientas más adecuadas para enfrentar la complejidad del problema abordado en este proyecto, especialmente en escenarios con múltiples restricciones y variables críticas.

Aplicaciones existentes

Actualmente, existen diversas aplicaciones orientadas a la gestión de turnos y recursos humanos. Muchas de ellas están enfocadas en sectores específicos como el de emergencias, seguridad o administración pública. A continuación, se mencionan algunas de las soluciones más relevantes:

- **aTurnos** [9]:
  Es una plataforma especializada en la planificación y gestión de turnos de trabajo, orientada principalmente a medianas y grandes empresas de sectores como sanidad, retail, hostelería y restauración, fabricación, servicios y logística y transporte.

Entre sus funcionalidades principales destacan:

- Planificación automática de turnos: permite crear cuadrantes de trabajo adaptados a las necesidades de cada organización, teniendo en cuenta tanto las preferencias como las normativas laborales vigentes.
- Gestión de ausencias y vacaciones: facilita a los empleados la solicitud de permisos y vacaciones directamente a través de la aplicación.
- Control de fichaje: ofrece múltiples opciones para el registro de la jornada laboral, incluyendo sistemas biométricos, geolocalización y aplicaciones móviles.
- Aplicación móvil: proporciona acceso en tiempo real a los cuadrantes de turnos, notificaciones y funcionalidades de autoservicio para los empleados.

Empresas reconocidas como Quirón Salud, IKEA, Vithas, Correos y MediaMarkt utilizan aTurnos para optimizar la gestión de sus recursos humanos.

- **PGPlanning** [10]:
  Es una solución diseñada para la planificación y gestión eficiente de turnos de trabajo, especialmente útil en organizaciones con gran volumen de personal o turnos rotativos. Entre sus funcionalidades destacan:
  - Generador automático de cuadrantes: permite la creación de horarios de trabajo considerando reglas, restricciones legales y necesidades operativas específicas.
  - Simulación de escenarios: hace posible la planificación anticipada mediante la simulación de distintos escenarios, facilitando la toma de decisiones en situaciones imprevistas.
  - Control horario: ofrece control de presencia tanto presencial como móvil.
  - Integración con otras aplicaciones: mediante API Restful, permite la exportación e importación de datos.

  Empresas como Iberdrola, AirEuropa, Ilunion, Multiópticas y Policía local de Ingenio utilizan PGPPlanning.

- **Eurocop SIGEM** [11]**:**
  Eurocop SIGEM es un sistema desarrollado por la empresa Eurocop, orientado a la gestión integral de cuerpos de seguridad y emergencias. Su diseño está enfocado en facilitar la operatividad y coordinación entre distintos equipos, especialmente en situaciones críticas donde la toma de decisiones rápida y basada en datos es fundamental. Entre sus funcionalidades se incluyen:
  - Coordinación multiagencia: permite gestionar actuaciones de servicios como Policía Local, Bomberos, Ambulancias y Protección Civil.
  - Gestión de incidencias: dispone de un catálogo de hechos para la gestión eficiente de incidencias.
  - Seguimiento en tiempo real: ofrece posicionamiento y seguimiento GPS de alta precisión.

- Integración con sistemas de videovigilancia: se integra con plataformas centralizadas de cámaras y sistemas de gestión de video.

Eurocop SIGEM se presenta como una herramienta integral para la gestión de emergencias y seguridad pública.

- **Sesame HR** [12]:
  Sesame HR es una plataforma integral de recursos humanos, dirigida a empresas que desean centralizar y digitalizar todos los procesos relacionados con la gestión de personal en un único entorno digital.
  Sus funcionalidades principales incluyen:
  - Control de la jornada laboral: registro y seguimiento del horario de trabajo de los empleados.
  - Planificación de turnos: configuración y asignación de turnos de trabajo.
  - Gestión documental: almacenamiento y organización de documentos relacionados con la gestión de recursos humanos.

  Además, incluye funcionalidades como gestión de vacaciones y ausencias, bolsa de horas, onboarding, comunicación interna y encuestas de clima laboral, lo que contribuye a mejorar la transparencia, la eficiencia y el bienestar del equipo humano.

5.2. Comparativa y selección de herramienta de optimización

En el contexto del Trabajo de Final de Grado (TFG) sobre la gestión de turnos para situaciones de emergencia, se realizaron una serie de pruebas exploratorias para evaluar distintas herramientas de optimización y determinar cuál se adapta mejor a los requisitos del problema.

Dado que en una primera etapa no disponía de información detallada sobre las restricciones y características específicas del ámbito de emergencias, se optó por utilizar un caso de estudio similar: la generación de turnos escolares. Este escenario permitió trabajar con un problema de optimización ya conocido, facilitando la definición de restricciones, la implementación del modelo y la evaluación de las distintas herramientas.

El objetivo de estas pruebas fue analizar cada herramienta, según los siguientes criterios:

- Facilidad de implementación
- Flexibilidad para incorporar restricciones
- Eficiencia en la resolución de modelos
- Escalabilidad para problemas más complejos

Se evaluaron dos herramientas principales: Google OR-Tools y Gurobi.

Ambas herramientas fueron evaluadas con un conjunto inicial de datos ficticios y restricciones básicas representativas del problema, con el fin de analizar su comportamiento en un entorno controlado.

Definición de Datos y Restricciones

Se definieron unos datos de entrada a través de un fichero Excel, en el que se definía la relación entre el curso, la asignatura, el profesor que la imparte y las horas asignadas a esa materia, como se muestra en la Figura 1:

Curso	Asignatura	Profesor	Horas
1ºA	Lengua	Nuria	5
1ºA	Matemáticas	Nuria	4
1ºA	Conocimiento del medio	Nuria	4
1ºA	Arts	Nuria	3
1ºA	Inglés	Irene	3
1ºA	Educación física	Carlos	3
1ºA	Emocrea	María	2
1ºA	Religión	Carlos	1
1ºB	Lengua	Marta	5
1ºB	Matemáticas	Marta	4
1ºB	Conocimiento del medio	Marta	4
1ºB	Arts	Marta	3
1ºB	Inglés	Marta	3
1ºB	Educación física	Carlos	3

Figura 1: Fragmento del archivo de entrada

Algunas de las restricciones que se definieron fueron las siguientes:

- Un profesor no puede impartir más de una clase simultáneamente
- Solo se permite una asignatura por hora y curso
- Una asignatura no puede repetirse en el mismo día
- No se puede superar el máximo de horas de trabajo del profesor ni el máximo de horas asignadas por materia a la semana

Además, el modelo busca optimizar la continuidad en el horario de los profesores, evitando horas libres y asegurando que sus clases sean consecutivas en un mismo día.

Implementación y resultados

Para abordar la generación de turnos, se implementó un modelo de optimización utilizando dos herramientas principales ya mencionadas: Google OR-Tools y Gurobi. El objetivo fue desarrollar una solución que asignara clases a horarios específicos cumpliendo con las restricciones establecidas y maximizando la continuidad en los horarios de los profesores.

Implementación con Google OR-Tools

Para esta herramienta se utilizó el solver CP-SAT, adecuado para problemas de programación con restricciones. El modelo se construyó a partir de variables binarias que representan la asignación de una asignatura concreta a un curso y una franja horaria. Las restricciones se expresaron directamente en Python mediante las funciones que ofrece la API de OR-Tools, lo que permitió una codificación clara y estructurada del problema.

A continuación, se muestra un fragmento de código donde se establecen algunas de las restricciones definidas:

```python
model = cp_model.CpModel()

# Crear variables: Clase x Día x Hora
clases = {}
for (curso, asignatura) in asignaturas.keys():
    for dia in dias:
        for hora in range(horas_por_dia):
            clases[(curso, asignatura, dia, hora)] = model.NewBoolVar(f'{curso}_{asignatura}_{dia}_{hora}')

# Restricción: Total de clases por asignatura
for (curso, asignatura), (_, total_horas) in asignaturas.items():
    model.Add(
        sum(clases[(curso, asignatura, dia, hora)] for dia in dias for hora in range(horas_por_dia)) == total_horas
    )

# Restricción: Máximo una clase por hora por curso
for dia in dias:
    for hora in range(horas_por_dia):
        for curso in cursos:
            model.Add(
                sum(clases[(curso, asignatura, dia, hora)] for (curso_key, asignatura) in asignaturas.keys() if curso_key == curso) <= 1
            )
```

Figura 2: Ejemplo código Google OR-Tools

Implementación con Gurobi

En el caso de Gurobi, el problema se planteó como un modelo de Programación Lineal Entera (MILP), utilizando variables binarias para representar la asignación de asignaturas a cursos en determinadas franjas horarias. Las restricciones se codificaron directamente en Python mediante la API de Gurobi, como se muestra en la Figura 3.

```python
model = Model("Horario Escolar")

# Crear variables de decisión
clases = {}
for (curso, asignatura) in asignaturas.keys():
    for dia in dias:
        for hora in range(horas_por_dia):
            clases[(curso, asignatura, dia, hora)] = model.addVar(vtype=GRB.BINARY,
                                                                  name=f'{curso}_{asignatura}_{dia}_{hora}')

# Restricción: Total de clases por asignatura
for (curso, asignatura), (_, total_horas) in asignaturas.items():
    model.addConstr(
        sum(clases[(curso, asignatura, dia, hora)] for dia in dias for hora in range(horas_por_dia)) == total_horas
    )

# Restricción: Máximo una clase por hora por curso
for dia in dias:
    for hora in range(horas_por_dia):
        for curso in cursos:
            model.addConstr(
                sum(clases[(curso, asignatura, dia, hora)] for (curso_key, asignatura) in asignaturas.keys() if curso_key == curso) <= 1
            )
```

Figura 3: Ejemplo código Gurobi

Se realizaron pruebas para conjuntos de datos de diferentes tamaños, con el propósito de evaluar tanto el tiempo de ejecución, así como la eficiencia en la distribución de los horarios, medida a través de la continuidad de estos. Este indicador refleja el porcentaje de profesores que tienen sus clases asignadas de manera continua, sin huecos intermedios en su jornada.

Cabe destacar que en el caso de Google OR-Tools, se ha establecido un límite máximo de tiempo de ejecución de 250 segundos para cada prueba, para evitar que el proceso se prolongara excesivamente. Si la herramienta no consigue resolver el problema en ese tiempo, se detiene el proceso y entrega la mejor solución encontrada hasta ese momento.

Se presentan a continuación las evaluaciones de rendimiento realizadas, junto con su análisis en la Tabla 2:

Tamaño del problema	Tiempo de ejecución OR-Tools	Tiempo de ejecución Gurobi	Continuidad OR-Tools	Continuidad Gurobi
6 cursos, 9 profesores	250.38s (límite)	33s	97.78%	100%
12 cursos, 15 profesores	250.7s (límite)	57.73s	99.09%	99.73%
24 cursos, 30 profesores	251.11s (límite)	43.93s	97.97%	100%

Tabla 2: Comparación de rendimiento entre OR-Tools y Gurobi

Como se observa en la Tabla 2, Gurobi ofrece un rendimiento superior en cuanto a tiempo de ejecución. Gracias a su motor de optimización avanzado, diseñado específicamente para resolver problemas de Programación Lineal Entera (MILP), Gurobi logró obtener soluciones óptimas en tiempos notablemente reducidos. Por ejemplo, en el caso más complejo (24 cursos y 30 profesores), resolvió el problema en apenas 43.93 segundos, mientras que Google OR-Tools alcanzó el límite de tiempo establecido (250 segundos) sin garantizar la optimalidad.

En este sentido, Gurobi destaca como la herramienta más eficiente para problemas de gran escala, especialmente cuando se requiere rapidez en la obtención de resultados.

Respecto a la calidad de la solución, evaluada mediante la continuidad en los horarios de los profesores, ambas herramientas ofrecieron buenos resultados. Gurobi logró un 100% de continuidad en dos de las tres pruebas y un 99,73% en la tercera. OR-Tools también mostró valores altos (entre 97,78% y 99,09%), pero en todos los casos ligeramente inferiores a Gurobi, probablemente debido a la finalización anticipada del proceso por el límite de tiempo.

Esto sugiere que, aunque OR-Tools puede generar soluciones de buena calidad, su rendimiento se ve penalizado en problemas de mayor tamaño.

En cuanto a la formulación y resolución del problema, cada herramienta presenta ventajas y desventajas que deben tenerse en cuenta:

Google OR-Tools destaca principalmente por ser una herramienta gratuita y de código abierto, lo que facilita su uso en proyectos académicos y experimentales. Además, su solver CP-SAT permite modelar problemas con restricciones complejas de forma intuitiva, brindando una gran flexibilidad en la formulación del modelo. Sin embargo, su rendimiento puede verse limitado cuando el tamaño del problema aumenta considerablemente, ya que el tiempo de ejecución se incrementa notablemente.

Por otro lado, Gurobi se presenta como una opción muy eficiente tanto en términos de tiempo de ejecución como en la calidad de la solución. Gracias a su motor avanzado para resolver problemas de Programación Lineal Entera, es capaz de alcanzar resultados óptimos en tiempos considerablemente reducidos, incluso en problemas de gran escala. Además, Gurobi ofrece funcionalidades avanzadas que facilitan el análisis del progreso de la optimización y la detección de posibles errores en la formulación del modelo. Por ejemplo, proporciona un registro detallado de la ejecución y permite generar archivos de diagnóstico con información útil sobre el modelo, lo que simplifica la depuración y mejora del código. No obstante, su principal desventaja radica en que se trata de un software comercial que requiere una licencia para su uso fuera del ámbito académico.

Conclusiones

Tras el análisis comparativo, se concluye que Gurobi es la herramienta más adecuada para el desarrollo del modelo de generación de turnos en este TFG. Su capacidad para encontrar soluciones óptimas en tiempos reducidos, incluso ante escenarios complejos, lo convierte en la opción preferente. Además, sus herramientas de diagnóstico y depuración facilitan la mejora continua del modelo y garantizan un rendimiento estable al escalar a problemas de mayor tamaño.

Aunque Google OR-Tools representa una alternativa válida, especialmente en contextos donde el coste y la apertura del software son prioritarios, su rendimiento en términos de tiempo y optimalidad queda por debajo del mostrado por Gurobi.

5.3. Adaptación del modelo al caso real: gestión de retenes en La Palma

En situaciones de emergencia como incendios forestales, la correcta planificación de los turnos del personal de intervención es fundamental para garantizar una respuesta eficaz y evitar el agotamiento del equipo. En este contexto, el Cabildo de La Palma cuenta con retenes contra incendios que deben estar disponibles de forma permanente, tanto en condiciones normales como durante emergencias prolongadas. La planificación manual de estos turnos puede ser ineficiente y propensa a errores, especialmente cuando las necesidades cambian rápidamente en función del avance del incendio.

El esquema de trabajo del personal se organiza según dos situaciones diferenciadas:

Turnos habituales (sin incendio)

En condiciones normales, el personal del Cabildo de La Palma trabaja en turnos fijos con las siguientes características:

- **Horario:** de 12:30 a 21:30.
- **Patrón de trabajo:** dos días consecutivos de trabajo seguidos de dos días de descanso (ciclo 2x2).
- **Cobertura adicional:** durante la jornada diurna, el personal habitual y el de refuerzo coinciden hasta aproximadamente las 17:30, mientras que los huecos restantes, especialmente durante la noche, son cubiertos exclusivamente por el personal de refuerzo.

Turnos en situación de incendio

Cuando se produce un incendio, el esquema de turnos cambia con el objetivo de garantizar una cobertura continua y eficiente. En estos casos:

- **Activación inicial:** en primera instancia, interviene el personal que se encuentra de turno habitual. Si la emergencia se prolonga o escala en intensidad, se prepara un sistema de relevos estructurado para asegurar la cobertura durante los días siguientes.
- **Turnos operativos:** se establecen dos turnos diarios de 12 horas de duración efectiva en el lugar del incendio, con 1 hora adicional antes y después del turno para desplazamientos:
    - **Turno diurno:** de 08:00 a 20:00 (salida desde la base a las 07:00 y regreso a las 21:00).
    - **Turno nocturno:** de 20:00 a 08:00 (salida a las 19:00 y regreso a las 09:00).
- **Justificación del horario:** los horarios de entrada y salida están diseñados para evitar relevos en plena noche, ya que es imprescindible que los equipos reconozcan el entorno de trabajo con luz natural por motivos de seguridad.

Ciclo rotativo ideal

Uno de los principales desafíos en la planificación de estos turnos es evitar que un mismo equipo tenga que asumir de forma consecutiva todas las noches. Para mitigar esta carga desigual, se intenta introducir un patrón rotativo que rompa la continuidad de los turnos nocturnos. El esquema ideal, aunque no siempre se logra cumplir en la práctica, es el siguiente:

Noche, noche, descanso, mañana, mañana, descanso, noche, noche, descanso…

Este patrón permite alternar entre turnos nocturnos y diurnos, incorporando descansos de 24 horas para facilitar la recuperación del personal tras dos jornadas consecutivas de trabajo intensivo. Aunque se considera un modelo óptimo desde el punto de vista operativo y humano, en la práctica es difícil de mantener de forma sistemática, lo que genera ciertas quejas justificadas por parte del personal.

Requisitos generales

- Los equipos trabajan en ciclos rotativos, procurando una distribución equitativa entre turnos.
- Se intenta garantizar un mínimo de 12 horas de descanso entre turnos consecutivos, siendo preferible disponer de 24 horas en los cambios entre franjas horarias.
- La planificación debe asegurar que no se asignen más de dos turnos nocturnos consecutivos sin descanso intermedio.

Adaptación al caso de retenes

Partiendo de un modelo inicial diseñado para la asignación de horarios escolares, centrado en la distribución equilibrada de sesiones, el cumplimiento de restricciones horarias y la continuidad en la jornada del profesorado, se procedió a su adaptación al problema de planificación de turnos en situaciones de emergencia, concretamente en la gestión de retenes contra incendios del Cabildo de La Palma.

Aunque ambos contextos presentan diferencias importantes, existen también similitudes estructurales que han permitido reutilizar parte del enfoque original. En ambos casos, el objetivo principal es asignar recursos humanos (profesores o retenes) a franjas horarias (clases o turnos) respetando un conjunto de restricciones. Sin embargo, la planificación de turnos introduce condicionantes adicionales:

- **Turnos más largos** (12 horas frente a bloques de 1-2 horas en entornos escolares).
- **Ciclos rotativos con alternancia diurna/nocturna**, en lugar de una planificación semanal fija.
- **Restricciones de descanso obligatorio**, especialmente tras turnos nocturnos.

A continuación, se detallan las modificaciones principales realizadas para ajustar el modelo al escenario de retenes:

1. Redefinición del tipo de recurso

En el modelo educativo, los recursos asignables eran las clases, representadas como tuplas del tipo *(curso, asignatura)*, que debían distribuirse a lo largo de la semana escolar. En el nuevo modelo, el recurso a asignar pasa a ser el retén, identificado por un código único *(r)*. El objetivo es determinar qué retenes deben cubrir qué turno en cada día, garantizando la cobertura operativa requerida y el cumplimiento de las restricciones laborales y de seguridad.

2. Rediseño de la variable de decisión

En el contexto escolar, la variable binaria de decisión tenía la forma:

`x [(curso, asignatura, día, hora)]` donde el valor 1 indicaba que la asignatura estaba programada en esa franja horaria para ese curso.

Para el nuevo problema, esta variable se redefine como:

`x [(r, día, turno)]` donde el valor 1 indica que el retén *r* está asignado al turno (diurno o nocturno) del día correspondiente.

Esta nueva formulación permite representar de manera compacta la planificación operativa de los retenes a lo largo del tiempo.

3. Modificación de las restricciones del modelo

Las restricciones cambian por completo ya que el escenario no es el mismo

Algunos ejemplos de restricciones:

- Garantizar que cada retén solo realice un turno por día.
- Asegurar un número mínimo y máximo de retenes activos por turno.
- Respetar un descanso mínimo de 12 horas entre un turno nocturno y el siguiente turno diurno.
- Fomentar un patrón rotativo equilibrado (noche-noche-descanso-mañana-mañana-descanso) para evitar la acumulación de turnos nocturnos.

Estas restricciones buscan tanto la viabilidad operativa como el bienestar del personal, asegurando tiempos de descanso adecuados y una distribución equitativa del esfuerzo entre los equipos disponibles.

4. Reformulación de la función objetivo

También fue necesario redefinir el criterio de optimización. En el modelo educativo, se buscaba maximizar la continuidad del profesorado, es decir, minimizar los huecos entre clases en un mismo día.

En cambio, en el contexto de emergencias, la función objetivo se orienta a minimizar el número total de turnos asignados, lo que permite distribuir de manera más equitativa la carga de trabajo entre los retenes disponibles.

A continuación, se describen algunas funcionalidades clave, como la forma en que se codifican las variables, restricciones y función objetivo. Y se mostrará los resultados obtenidos en esta fase.

En primer lugar, en el constructor de ShiftOptimizer se inicializan los parámetros básicos y las variables de decisión.

```
class ShiftOptimizer:
    def __init__(self, num_retenes=22, dias=7):
        """
        Se asume que 'dias' es múltiplo de 6 para garantizar que cada ciclo completo
        (2 días en un turno, 1 descanso, 2 días en el otro turno, 1 descanso) se repita sin errores.
        """
        self.num_retenes = num_retenes
        self.dias = dias
        self.num_turnos = 2
        self.model = Model("Optimización de Turnos de Retenes")

        # Variables de decisión:
        self.y = {r: self.model.addVar(vtype=GRB.BINARY, name=f"y_{r}") for r in range(self.num_retenes)}
        self.p = {r: self.model.addVar(vtype=GRB.BINARY, name=f"p_{r}") for r in range(self.num_retenes)}
        self.d = {(r, d, t): self.model.addVar(vtype=GRB.BINARY, name=f"d_{r}_{d}_{t}")
                  for r in range(self.num_retenes) for d in range(self.dias) for t in range(self.num_turnos)}

        self.model.update()
```

Figura 4: Inicialización de parámetros y variables

Para modelar el ciclo de trabajo de cada retén de forma flexible y a la vez compacta, en nuestra implementación definimos dos tipos de variables binarias: las variables de ciclo (y o p) y las variables de asignación (d). Cada una cumple un propósito distinto:

1. Variables de ciclo
    - y[r] toma valor 0 o 1 según el turno en el que comience el ciclo de trabajo del retén *r*.
        - Si y[r] = 0, interpretamos que el retén *r* inicia su primera fase trabajando en el turno diurno (Turno 0).
        - Si y[r] = 1, comenzará en el turno nocturno (Turno 1).
    - p[r] también es binaria y controla la forma en que ese retén alterna de un bloque de dos días al siguiente.
        - Cuando p[r] = 0, el cambio tras los primeros dos días se hace “al turno contrario”: si empezó en diurno pasa a nocturno, y viceversa.
        - Si p[r] = 1, invertimos ese patrón, de modo que el retén arranca en el turno opuesto y luego vuelve al inicial.

Gracias a combinar y[r] y p[r] podemos representar con muy pocas variables un ciclo rotativo complejo —dos días de turno, un día de descanso, dos días de turno contrario y un día de descanso— sin tener que escribir una restricción para cada día y cada retén de forma manual.

2. Variables de asignación
    - d [r, d, t] codifica la presencia (1) o ausencia (0) del retén *r* en el día *d* y el turno *t*.
        - d [r, d, 0] = 1 significa que el equipo *r* cubre el turno diurno del día *d*.

- d [r, d, 1] = 1 indica que cubre el turno nocturno.
- Si d [r, d, t] = 0, el retén no está asignado en esa franja.

Esta matriz tridimensional (retén × día × turno) es la pieza clave que nos permite, en un segundo paso, imponer de forma genérica todas las restricciones de cobertura mínima, máxima y descanso. Basta con recorrerla mediante bucles para crear cientos o miles de condiciones automáticas, en lugar de tener que llamar a addConstr una a una.

Una de las condiciones más básicas para garantizar la operatividad de los retenes es asegurarse de que, en cada día y en cada turno, exista un número suficiente de equipos activos, pero sin exceder la capacidad disponible. En nuestro modelo, imponemos que haya al menos 6 retenes y como máximo 8 retenes cubriendo cada turno de cada día. Esto se codifica con dos bucles anidados:

```
# Restricción de cobertura mínima y máxima por turno
for d in range(self.dias):
    for t in range(self.num_turnos):
        expr = quicksum(self.d[r, d, t] for r in range(self.num_retenes))
        self.model.addConstr(expr >= 6, name=f"min_turno{t}_dia_{d}")  # Min 6 retenes por turno
        self.model.addConstr(expr <= 8, name=f"max_turno{t}_dia_{d}")  # Max 8 retenes por turno
```

Figura 5: Restricción cobertura máxima y mínima por turno

Para asegurar que en cada día y en cada turno haya siempre el número adecuado de retenes, se recorre de forma automática cada combinación de jornada (d) y turno (t). Con quicksum sumamos todas las variables binarias d [r, d, t]—cada una indicando si un retén r está activo en esa franja—obteniendo así el total de equipos asignados. A continuación, imponemos dos límites mediante addConstr: uno que exige que esa suma sea al menos 6 retenes, y otro que obliga a que no supere 8 retenes. Gracias a estos dos bucles anidados se generan 2 × días restricciones de cobertura sin necesidad de escribirlas una por una, garantizando siempre la operatividad mínima requerida y evitando la asignación excesiva de personal.

Por otro lado, se muestra en la siguiente imagen una de las restricciones más complicadas, el ciclo rotativo de turnos. Para reflejar el patrón ideal de 2 días de turno ➔ 1 día de descanso ➔ 2 días de turno contrario ➔ 1 día de descanso, definimos una restricción que, para cada retén r y cada día d, calcula en qué fase del ciclo se encuentra y ajusta d[r, d,t] en consecuencia.

```python
for r in range(self.num_retenes):
    for d in range(self.dias):
        j = (d - (r % 6)) % 6

        # Trabaja 2 días seguidos en el mismo turno
        if j in [0, 1]:
            for t in range(self.num_turnos):
                self.model.addConstr(
                    self.d[r, d, t] == self.y[r] * (1 - self.p[r] if t == 1 else self.p[r]),
                    name=f"trabajo_inicio_{r}_{d}_{t}"
                )
        elif j in [3, 4]:
            for t in range(self.num_turnos):
                self.model.addConstr(
                    self.d[r, d, t] == self.y[r] * (self.p[r] if t == 1 else 1 - self.p[r]),
                    name=f"trabajo_opuesto_{r}_{d}_{t}"
                )

        # Descansa en los días 2 y 5 (después de trabajar 2 días seguidos)
        if j in [2, 5]:
            for t in range(self.num_turnos):
                self.model.addConstr(self.d[r, d, t] == 0, name=f"descanso_{r}_{d}_{t}")
```

Figura 6: Restricción ciclo rotativo turnos

Para reproducir de forma automática el ciclo de 2 días de trabajo – 1 día de descanso – 2 días de trabajo en turno contrario – 1 día de descanso, calculamos para cada retén r y día d una fase j tal que:

$$j = (d - (r \bmod 6)) \bmod 6$$

De este modo, en un bloque de seis jornadas j recorre los valores 0–5, y el desplazamiento r mod 6 hace que cada retén comience su ciclo en un punto distinto, equilibrando la rotación entre todos.

1. **Fases de trabajo en turno inicial (j = 0, 1).**

   Durante los dos primeros días del ciclo (j igual a 0 o 1), el retén desempeña su labor en el turno inicial. Utilizamos las variables y[r] y p[r] para determinar si ese turno es diurno o nocturno y en qué orden:

   - Para el turno diurno (t = 0), fijamos $d[r, d,0]=y[r]\times p[r]$
   - Para el turno nocturno (t = 1), aplicamos $d[r, d,1]=y[r]\times(1-p[r])$

2. **Fases de trabajo en turno contrario (j = 3, 4).**

   En los dos días siguientes (j igual a 3 o 4), invertimos la asignación anterior, de modo que el retén cubra el turno opuesto. La expresión cambia de forma análoga, intercambiando el papel de p[r] para reflejar el turno contrario.

3. **Días de descanso (j = 2, 5).**

   En las fases intermedias de descanso (j igual a 2 o 5), simplemente forzamos $d[r, d,t]=0$ para ambos turnos, garantizando que el equipo permanezca inactivo y reciba su jornada de reposo.

Con este único bloque de código, que recorre todos los retenes y todos los días, definimos exactamente cuándo cada retén trabaja de día, trabaja de noche o descansa. De esta forma, el patrón 2-1-2-1 se propaga de manera uniforme y escalable sin necesidad de escribir manualmente varias restricciones por jornada y por equipo.

En lugar de limitarse a minimizar el número de retenes activos o la desviación en cobertura por turno, el modelo incorpora una función objetivo orientada a maximizar la equidad en la distribución del trabajo entre todos los equipos.

```
# Calcular la carga de trabajo total de cada retén (cuántos turnos ha trabajado)
for r in range(self.num_retenes):
    self.model.addConstr(
        carga_trabajo[r] == quicksum(
            self.d[r, d, t] for d in range(self.dias) for t in range(self.num_turnos)),
        name=f"calculo_carga_trabajo_{r}"
    )

# **Nueva Función Objetivo:** Minimizar la diferencia entre los retenes más cargados y menos cargados
max_carga = self.model.addVar(vtype=GRB.CONTINUOUS, name="max_carga")
min_carga = self.model.addVar(vtype=GRB.CONTINUOUS, name="min_carga")

for r in range(self.num_retenes):
    self.model.addConstr(max_carga >= carga_trabajo[r], name=f"max_carga_reten_{r}")
    self.model.addConstr(min_carga <= carga_trabajo[r], name=f"min_carga_reten_{r}")

# Función objetivo: minimizar la diferencia entre la carga de trabajo máxima y mínima
self.model.setObjective(
    max_carga - min_carga,
    GRB.MINIMIZE
)
```

Figura 7: Función objetivo

Variables de holgura

Para cada día *d* y turno *t* definimos dos variables continuas, exceso[d, t] y defecto[d, t], que miden cuánto nos apartamos de los límites de cobertura establecidos (6–8 retenes). De esta forma, las restricciones

$$\sum_r d[r,d,t] \;\geq\; 6 - \text{defecto}[d,t], \quad \sum_r d[r,d,t] \;\leq\; 8 + \text{exceso}[d,t]$$

pueden cumplirse con cierta flexibilidad, permitiendo al modelo adaptarse a casos extremos sin hacerlo inviable.

Carga de trabajo por retén

Para cada equipo r se introduce la variable continua

$$\text{carga\_trabajo}[r] = \sum_{d,t} d[r,d,t],$$

que acumula el número total de turnos asignados a lo largo del periodo de planificación.

Variables de máximo y mínimo

Se crean dos variables adicionales, max_carga y min_carga, cuya función es capturar, respectivamente, la carga de trabajo más alta y la más baja entre todos los retenes. Mediante las restricciones

$$max\_carga \geq carga\_trabajo[r], min\_carga \leq carga\_trabajo[r] \forall\, r,$$

aseguramos que estos dos valores representen efectivamente los extremos de la distribución de carga.

Reducción de la brecha

Finalmente, el objetivo del modelo es

$$\min (max\_carga - min\_carga),$$

de modo que la diferencia entre el retén con más turnos y el que menos trabaja sea tan pequeña como sea posible. Con ello, se logra una distribución homogénea y equilibrada del esfuerzo entre todos los equipos.

A continuación, se presentan los resultados obtenidos:

Semana	Día	Retén 0	Retén 1	Retén 2	Retén 3	Retén 4	Retén 5	Retén 6	Retén 7	Retén 8	Retén 9	Retén 10	Retén 11	Retén 12	Retén 13	Retén 14	Retén 15	Retén 16	Retén 17	Retén 18	Retén 19	Retén 20	Retén 21
	Lunes	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 1
	Martes	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1
	Miércoles	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso
	Jueves	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 0
	Viernes	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0
	Sábado	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso
Semana 1	Domingo	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 1
	Lunes	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1
	Martes	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso
	Miércoles	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 0
	Jueves	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0
	Viernes	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso
	Sábado	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 1
Semana 2	Domingo	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1
	Lunes	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso
	Martes	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 0
	Miércoles	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0
	Jueves	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso
	Viernes	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 1
	Sábado	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 1	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1
Semana 3	Domingo	Descanso	Turno 0	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 1	Turno 0	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 1	Descanso	Turno 0	Turno 1	Descanso	Turno 1	Turno 0	Descanso

Leyenda:	
Turno 0	8:00 - 20:00
Turno 1	0:00 - 08:00

Figura 8: Resultados de la optimización

En la Figura 8 se aprecia una organización clara de los turnos, con todas las restricciones principales cumplidas al 100 %: cada franja queda cubierta por entre 6 y 8 retenes, se garantizan los descansos obligatorios y se cumple perfectamente el ciclo rotativo 2-1-2-1.

En resumen, la transformación del modelo ha sido completamente exitosa, adaptando la estructura de recursos y variables de decisión al entorno de emergencias. La combinación de variables de ciclo y asignación ha permitido codificar de forma compacta toda la lógica de turnos y descansos, mientras que la función objetivo equilibra la carga de trabajo de manera automática. Estos resultados confirman que el sistema es plenamente aplicable a situaciones reales de emergencia.

5.4. Presentación resultados a la empresa y nuevos requisitos funcionales

Una vez, presentado estos resultados a la empresa, ellos plantearon que querían una herramienta que fuera capaz de gestionar no solo los retenes contra incendios, sino cualquier otro recurso que fuera necesario en la planificación, por ejemplo, retenes y conductores. Y además adaptarse a distintos dominios de planificación de turnos.

El objetivo era convertir el prototipo planteado originalmente para emergencias en un solucionador genérico que, además de organizar los ciclos de trabajo de los retenes de La Palma, pudiera aplicarse a escenarios tan variados como la planificación de guardias médicas, asignación de conductores de reparto o, incluso, la generación de horarios escolares.

Para ello, definimos varios de casos de uso representativos y estudiamos sus puntos comunes y sus diferencias, con el fin de poder extraer una arquitectura de modelado y parametrización que sirviera de base a cualquier escenario de planificación de turnos.

Con este objetivo en mente, seleccionamos una serie de casos de uso con distintas características, tanto en su naturaleza como en el tipo de recursos implicados.

Analizar estos escenarios nos permitió identificar patrones comunes y diferencias clave que se debían tener en cuenta en el diseño del modelo general.

A continuación, se describen cuatro de esos casos representativos:

1. Planificación de retenes contra incendios

Descripción: Se trata de organizar turnos de trabajo para retenes durante un periodo determinado, garantizando descansos, rotaciones y cobertura diaria.

Recursos implicados: retenes

Relaciones entre recursos: Independientes entre sí, cada retén puede ser asignado sin depender de otro recurso.

Restricciones comunes:

- Cobertura diaria mínima
- Descanso entre turnos
- Rotación y equidad
- Limitación de días consecutivos trabajados

2. Generación de horarios escolares

Descripción: Se trata de asignar sesiones a lo largo de la semana respetando disponibilidad de los profesores y estructura de los cursos. Se debe asegurar que no haya solapamientos y que cada asignatura reciba el número de clases establecido.

Recursos implicados: profesores, asignaturas, cursos, aulas

Relaciones entre recursos: Alta dependencia, una clase debe estar definida por la combinación de un profesor, una asignatura y un curso. Además, algunos recursos pueden

tener exclusividad temporal, por ejemplo, un profesor no puede estar en dos sitios a la vez.

Restricciones comunes:

- No solapamiento de horarios por recurso.
- Distribución equilibrada de sesiones a lo largo de la semana.
- Restricciones individuales de disponibilidad horaria.
- Vinculación entre asignaturas y profesores específicos.
- Número fijo de sesiones por asignatura y grupo.

3. Planificación de guardias médicas

Descripción: Asignar turnos de guardia a médicos y enfermeros en un hospital o centro de salud, cubriendo servicios 24/7 y respetando competencias y preferencias personales.

Recursos implicados: médicos, enfermeros, especialidades, unidades (urgencias, cuidados intensivos, etc.).

Relaciones entre recursos: Cada guardia requiere un equipo multidisciplinar (varias especialidades). La asignación de un médico está condicionada a su especialidad y a la unidad asignada.

Restricciones comunes:

- Cobertura continua de 24 horas.
- Descansos mínimos entre guardias.
- Distribución equitativa de turnos nocturnos y festivos.
- Compatibilidad de especialidad con la unidad.
- Preferencias y disponibilidad individual.

4. Planificación de vigilantes de seguridad

Descripción: Organizar turnos de trabajo para vigilantes de seguridad en distintos puestos o zonas, garantizando vigilancia continua y descansos adecuados.

Recursos implicados: vigilantes, puestos de vigilancia.

Relaciones entre recursos: Independientes; cada vigilante se asigna a un puesto sin depender de otros vigilantes.

Restricciones comunes:

- Cobertura mínima por puesto y turno.
- Descanso mínimo entre turnos.
- Rotación y equidad en la asignación de puestos.
- Limitación de días consecutivos trabajados.
- Consideración de habilitaciones o certificaciones específicas para ciertos puestos.

A partir del análisis de estos escenarios, se identificaron dos casos base que permiten estructurar el modelo de forma flexible y reutilizable para distintos dominios:

1. Asignaciones individuales

Este caso se aplicaría cuando cada recurso puede planificarse de forma independiente, sin necesidad de combinarlo con otros elementos. Por ejemplo, los retenes contra incendios o los conductores de reparto, cada unidad es autónoma y basta decidir en qué franja y día trabaja.

Necesitaríamos una sola variable de decisión por recurso, día y franja. Por ejemplo, en el caso de retenes y conductores quedaría así:

```
# Para cada retén r, cada día d y cada franja f:
self.x_reten = {
    (r, d, f): model.addVar(
        vtype=GRB.BINARY,
        name=f"x_reten_{r}_{d}_{f}"
    )
    for r in variables['lista_retenes']
    for d in range(variables['dias'])
    for f in range(variables['franjas'])
}

# Para cada conductor c, cada día d y cada franja f:
self.x_conductor = {
    (c, d, f): model.addVar(
        vtype=GRB.BINARY,
        name=f"x_conductor_{c}_{d}_{f}"
    )
    for c in variables['lista_conductores']
    for d in range(variables['dias'])
    for f in range(variables['franjas'])
}
```

Figura 9: Caso asignaciones individuales

2. Asignaciones compuestas (multientidad): recursos que deben combinarse para definir una asignación, por ejemplo, profesor + asignatura + curso.

Este otro caso se aplicaría cuando la definición de un turno requiere combinar dos o más tipos de entidad, por ejemplo, profesor + asignatura + curso en un horario escolar, o medico + especialidad + consulta en caso de consultas de hospital. Para estos casos necesitamos una variable de decisión que represente cada tupla de entidades y su asignación temporal. Por ejemplo, para el caso de horario escolar quedaría de esta manera:

```
# Para cada profesor p, curso c y asignatura a, cada día d y cada franja f:
self.x_profesor_curso_asignatura = {
    (p, c, a, d, f): model.addVar(
        vtype=GRB.BINARY,
        name=f"x_{p}_{c}_{a}_{d}_{f}"
    )
    for p in variables['lista_profesores']
    for c in variables['lista_cursos']
    for a in variables['lista_asignaturas']
    for d in range(variables['dias'])
    for f in range(variables['franjas'])
}
```

Figura 10: Caso asignaciones compuestas

5.5. Estrategia para soluciones inviables *(infeasible solutions)*

En un caso real de planificación de retenes, o cualquier otro tipo de planificación, el usuario podría especificar restricciones muy estrictas o incluso contradictorias. Lo que provocaría que el modelo devuelva un estado INFEASIBLE, un mensaje que el usuario no entendería y no sabría lo que está ocurriendo. Para evitar esta situación nuestra clase ShiftOptimizer implementa:

1. Cálculo del ISS

Si tras la llamada a model.optimize(), el atributo model.status indica INFEASIBLE o INF_OR_UNBD, ejecutamos:

```
if status in (GRB.INFEASIBLE, GRB.INF_OR_UNBD):
    print("❌ Modelo inviable. IIS:")
    self.model.computeIIS()
    for c in self.model.getConstrs():
        if c.IISConstr:
            desc = self.constraint_descriptions.get(c.constrName, "(sin descripción)")
            print(f"   ↯ {c.constrName} – {desc}")
```

Figura 11: Extracción de restricciones en conflicto

Esto extrae el conjunto mínimo de restricciones en conflicto y nos permite identificar exactamente qué condiciones no son posibles de satisfacer, ya que las muestra con su descripción en lenguaje natural.

2. Relajación automática con FeasRelax

A continuación, al detectar inviabilidad, invocamos FeasRelax para permitir únicamente la relajación de las restricciones lineales, sin tocar los límites de las variables, de la siguiente forma:

```
orig = self.model.NumVars
self.model.feasRelaxS(relaxobjtype=0, minrelax=False, vrelax=False, crelax=True)
self.model.optimize()
```

Figura 12: Relajar restricciones lineales

Esto introduce variables de slack (artificiales) en cada restricción conflictiva, y busca minimizar la suma total de estas violaciones sin modificar las variables originales.

A continuación, extraemos todos los slacks con valor positivo y, para cada uno, recuperamos la descripción en lenguaje natural de la restricción afectada:

```python
if self.model.status == GRB.OPTIMAL:
    print("✅ Modelo relajado resuelto. Objetivo:", self.model.ObjVal)
    slacks = self.model.getVars()[orig:]
    relaxed_nls = []
    for sv in slacks:
        if sv.X > 1e-6:
            cname = sv.VarName
            if cname.startswith("ArtP_") or cname.startswith("ArtN_"):
                cname = cname.split( sep: "_", maxsplit: 1)[1]
            phrase = self.constraint_descriptions.get(cname, f"(sin mapping para {cname})")
            relaxed_nls.append(phrase)
            relaxed_nls = list(dict.fromkeys(relaxed_nls))
            print(f"   · {phrase} (relajada: {sv.X:g})")
```

Figura 13: Recuperación de la descripción en Nl de la restricción afectada

Con este proceso, el usuario recibe:

- Un listado de las restricciones que hicieron el modelo infeasible (IIS).
- Una propuesta automática de relajación, con la cantidad exacta de slack aplicado a cada una.

Así, no solo alertamos de “modelo inviable”, sino que proporcionamos un diagnóstico claro y una solución de compromiso inmediata, acelerando enormemente el ajuste de las reglas operativas.

6. Trabajo individual

Durante la fase conjunta del proyecto, se definieron y validaron los pilares fundamentales de la arquitectura del solucionador genérico de turnos. Esta etapa incluyó la identificación de casos de uso representativos y la parametrización común que permite adaptar la herramienta a distintos dominios (retenes, guardias médicas, horarios escolares, reparto logístico, entre otros).

A partir de ese trabajo inicial, mi aportación individual se ha centrado en el diseño e implementación de una experiencia de usuario completa, construyendo la capa de presentación e integración con el motor de optimización Gurobi. Esta tarea abarca desde la definición del recorrido del usuario (*user journey*), hasta la generación de prototipos, el desarrollo *front-end* y *back-end*, y la documentación técnica de todo el proceso de integración, incluyendo aspectos clave como la ingeniería de prompts, la validación del contexto y la detección automática de restricciones.

A continuación, se resume de forma esquemática el conjunto de herramientas y tecnologías empleadas en el desarrollo:

- Lenguaje principal & *back-end*
    - Python: lógica de orquestación y comunicación con Gurobi
    - Flask: definición de rutas REST para gestionar peticiones y parametrizar el modelo de optimización
- *Front-end*
    - HTML, CSS y JavaScript: desarrollo de la interfaz responsive y accesible
    - Uso de bibliotecas de componentes ligeros para una experiencia interactiva y fluida
- Diseño de experiencia de usuario
    - Figma: diseño iterativo de prototipos e interacciones antes del desarrollo
- Persistencia de datos
    - MongoDB: almacenamiento de configuraciones, restricciones y resultados con esquemas dinámicos adaptados al dominio

Este conjunto de tecnologías ha permitido construir una solución flexible y extensible, capaz de ofrecer una interacción intuitiva con el sistema de planificación y facilitar la adaptación del modelo a distintos contextos de aplicación.

6.1. Recorridos de usuario en ResQPlan

Para la elaboración de estos escenarios se ha aplicado la técnica de *User Journey Mapping,* un enfoque estructurado que permite construir una representación tanto visual como cronológica de cada punto de interacción que el usuario lleva a cabo al interactuar con la plataforma. De este modo, podemos desglosar cada fase de uso, identificar los canales de contacto y comprender las expectativas y motivaciones que guían al usuario en su recorrido.

Este análisis detallado no solo nos ayuda a entender cómo y por qué los usuarios llegan a ResQPlan, sino también a optimizar cada fase de su experiencia, reforzar los mensajes clave y garantizar que la plataforma aporte valor real en función de sus objetivos y circunstancias.

A continuación, se describen tres recorridos de usuario representativos que ilustran diferentes perfiles y escenarios reales de uso, destacando los beneficios que aporta ResQPlan en cada caso:

Carlos Morales

JEFE DE OPERACIONES

Profesión: jefe de operaciones de los retenes forestales en La Palma

Edad: 46 años

PERFIL

Carlos lleva más de 20 años trabajando en la gestión de emergencias. Aunque no tiene formación técnica en informática, es una persona organizada, resolutiva y familiarizada con hojas de cálculo y herramientas de planificación básicas. Es responsable de diseñar los horarios de los trabajadores de su equipo, cumpliendo con los descansos legales y asegurando una cobertura adecuada. A veces le resulta complicado equilibrar las necesidades del operativo con el bienestar del personal y mantener a todos satisfechos.

OBJETIVO

Se ha declarado un incendio en el norte de La Palma. Carlos necesita planificar de forma urgente los turnos de los 22 retenes activos en la isla, garantizando cobertura diaria y el descanso del personal. Busca una herramienta que le permita ahorrar tiempo, reducir errores y generar una planificación justa y equilibrada.

EXPERIENCIA EN USO

Carlos accede a ResQPlan desde su portátil. Selecciona “Nuevo proyecto” y lo nombra como: Planificación Retenes – Incendio.
Carlos escribe en lenguaje natural:
“Tenemos 22 retenes. Cada día se divide en dos turnos, el turno diurno de 8:00 a 20:00 y el nocturno de 20:00 a 8:00. Cada retén puede cubrir solo un turno por día.”
El sistema detecta automáticamente las variables clave: retén, turno, día. Carlos las revisa y confirma.
Además, el sistema identifica una posible restricción implícita: "Cada retén puede cubrir solo un turno por día", y le sugiere añadirla como condición formal en la sección de restricciones.
Carlos introduce varias condiciones operativas:

- “Ningún retén puede trabajar más de 2 noches por semana.”
- “Después de una noche, el retén debe descansar al menos un turno.”
- “Repartir el trabajo de forma equilibrada entre todos los equipos.”

El sistema traduce automáticamente estas frases en restricciones matemáticas, que Carlos revisa, edita si es necesario y activa.
Carlos pulsa el botón “Optimizar”. En pocos segundos, el sistema genera una tabla con los turnos asignados para toda la semana. Puede visualizar la planificación por retén, por día, o por turno, y comprobar que todas las restricciones se cumplen.

VALOR PARA CARLOS

Gracias a ResQPlan, Carlos ahorra varias horas de trabajo administrativo, evita errores manuales y logra una planificación más justa y adaptada al operativo. La posibilidad de introducir reglas en lenguaje natural le permite centrarse en lo importante sin necesidad de conocimientos técnicos.

Laura Benítez

SUPERVISORA DE ENFERMERÍA

Profesión: Supervisora de Enfermería en el Hospital General de Tenerife

Edad: 34 años

PERFIL

Laura coordina el equipo de enfermería de planta en uno de los hospitales más grandes de Canarias. Tiene formación en gestión sanitaria y una gran experiencia organizando turnos de trabajo, pero suele verse desbordada por las solicitudes de cambio, las bajas inesperadas y las exigencias del personal. Suele usar Excel para cuadrar los horarios, pero el proceso es lento, propenso a errores y requiere mucho tiempo de revisión y ajustes manuales.

OBJETIVO

Laura debe preparar la planificación de turnos para el mes de julio para su equipo de 18 enfermeros. Necesita garantizar cobertura las 24 horas del día (mañana, tarde y noche), cumplir con los turnos de descanso obligatorios, respetar las vacaciones ya aprobadas y, si es posible, adaptarse a algunas preferencias personales del equipo. Busca una solución más eficiente, menos propensa a errores y que le permita responder de forma ágil a imprevistos.

EXPERIENCIA EN USO

Laura entra en ResQPlan desde su ordenador en la sala de coordinación. Crea un nuevo proyecto: "Turnos Enfermería – Julio 2025"

Escribe en lenguaje natural: "Hay 18 enfermeros. Cada día se divide en tres turnos: mañana (7:00–15:00), tarde (15:00–23:00), y noche (23:00–7:00). Cada enfermero puede trabajar como máximo 5 días seguidos y debe descansar al menos un día tras una guardia nocturna."
El sistema reconoce automáticamente las variables clave: personal, días, turnos. Laura confirma que están bien detectadas.
Laura introduce varias reglas adicionales:

- "Cada enfermero debe tener al menos 8 turnos de descanso al mes."
- "No repetir el mismo turno más de 3 días consecutivos."
- "Respetar vacaciones del 10 al 20 de julio para Ana y del 15 al 30 para Marcos."

El sistema traduce estas condiciones en restricciones formales. Laura las revisa y activa.
Pulsa "Optimizar". El sistema genera automáticamente la planificación de todo el mes.
Laura puede revisar la distribución de turnos por persona o por día, y comprobar fácilmente si se han respetado los descansos y preferencias.

VALOR PARA LAURA

Con ResQPlan, Laura reduce drásticamente el tiempo que dedicaba a cuadrar los turnos, evita conflictos derivados de errores humanos y consigue una distribución más justa para su equipo. La herramienta le permite introducir condiciones específicas del convenio sanitario.

Marta Rodríguez

COORDINADORA ACADÉMICA

Profesión: Coordinadora académica en un instituto de secundaria en Gran Canaria

Edad: 41 años

PERFIL

Marta es responsable de coordinar el horario semanal de todas las asignaturas, profesores y grupos del centro. Aunque cuenta con experiencia y conoce bien las necesidades del instituto, cada curso se enfrenta al mismo reto: cuadrar todas las materias, evitar solapamientos y respetar tanto las horas asignadas por materia como las disponibilidades del profesorado. Actualmente usa una combinación de plantillas Excel y pruebas a mano, lo que le consume muchas horas y le obliga a rehacer el horario varias veces ante cualquier cambio.

OBJETIVO

Estamos en septiembre y Marta necesita organizar el horario completo del nuevo curso para 12 grupos de la ESO y Bachillerato, 35 docentes y más de 20 asignaturas diferentes. Debe garantizar que no haya conflictos, que cada profesor tenga sus horas bien distribuidas y que se respeten sus ventanas de disponibilidad. Busca una herramienta que agilice este proceso y le permita generar un horario equilibrado, claro y ajustado a las necesidades del centro.

EXPERIENCIA EN USO

Marta entra en ResQPlan desde el despacho de jefatura. Crea un nuevo proyecto: “Horario Escolar – Curso 2025/2026”

Escribe en lenguaje natural: “Tenemos 12 grupos y 35 profesores. El horario lectivo va de lunes a viernes, de 8:00 a 14:00, dividido en 6 bloques de 1 hora. Cada asignatura tiene un número fijo de sesiones semanales. Cada profesor puede impartir clase en un máximo de 5 bloques por día y tiene ciertas horas bloqueadas por reuniones u otras funciones.”
El sistema detecta las variables clave: grupos, días, horas, docentes, asignaturas, y Marta las revisa y confirma.
Introduce restricciones como:

- “Ningún grupo debe tener más de 3 horas seguidas sin recreo.”
- “No repetir una misma asignatura más de 2 veces en el mismo día.”
- “Respetar la no disponibilidad de cada profesor.”
- “Las materias troncales deben estar preferentemente en las primeras horas.”

El sistema convierte estas frases en restricciones formales y Marta las revisa antes de continuar.
Pulsa “Optimizar”. En pocos segundos, el sistema genera un horario completo, evitando solapamientos, respetando las restricciones y distribuyendo de forma lógica las materias y profesorado.
Después, exporta los horarios por grupo, por docente y por aula en formato PDF y Excel, listos para su publicación.

VALOR PARA MARTA

Gracias a ResQPlan, Marta ahorra semanas de trabajo, evita errores difíciles de detectar en hojas de cálculo y obtiene un horario justo, coherente y listo para adaptarse ante imprevistos. El uso de lenguaje natural para expresar restricciones facilita enormemente el proceso, incluso sin conocimientos técnicos.

6.2. Análisis de requisitos

Esta sección recoge de forma estructurada los requisitos del sistema que se va a desarrollar, tanto los aspectos funcionales como técnicos. El objetivo es definir con claridad qué debe hacer la herramienta, cómo debe comportarse y en qué condiciones debe operar, siguiendo un enfoque orientado a la ingeniería de software.

El producto es una herramienta visual e interactiva para la planificación de turnos. A través de ella el usuario podrá:

- Configurar distintos tipos de escenarios para la planificación (retenes, enfermeros, transporte…)
- Establecer restricciones operativas, como número máximo de horas, descansos obligatorios…
- Generar soluciones óptimas

A continuación, se clasifican los requisitos según su naturaleza

- **Requisitos funcionales**: describen los servicios y funciones que la aplicación debe proporcionar al usuario, describiendo las interacciones directas con la interfaz y los procesos que esta ejecuta.
- **Requisitos no funcionales**: agrupan aspectos relacionados con el rendimiento, la tecnología, la usabilidad, la interoperabilidad y otros factores técnicos no ligados a una función específica.

Requisitos funcionales

- El sistema debe permitir al usuario crear nuevos proyectos
- El sistema debe permitir listar los proyectos existentes y permitir al usuario seleccionar uno para cargarlo
- El sistema debe autoguardar los cambios de cada proyecto
- El sistema debe permitir al usuario duplicar proyectos existentes, generando una copia del estado
- El sistema debe permitir al usuario editar el nombre del proyecto
- El sistema debe permitir añadir automáticamente al modelo las restricciones detectadas en el contexto
- El sistema debe detectar si el contexto introducido no corresponde a un problema de planificación de turnos
- El sistema debe permitir al usuario eliminar proyectos existentes
- El sistema debe permitir al usuario introducir y editar un contexto en lenguaje natural
- El sistema debe extraer automáticamente las variables clave del contexto introducido.
- El sistema debe permitir al usuario ver un resumen del contexto que ha creado
- El sistema debe permitir al usuario definir restricciones en lenguaje natural
- El sistema debe convertir las restricciones definidas al formato requerido por Gurobi.

- El sistema debe detectar si la restricción introducida no concuerda con el problema planteado en el contexto
- El sistema debe mostrar al usuario la lista de restricciones configuradas
- El sistema debe permitir la edición de restricciones previamente añadidas.
- El sistema debe permitir al usuario borrar una restricción añadida
- El sistema debe permitir al usuario activar o desactivar restricciones para evaluar diferentes escenarios de planificación
- El sistema debe permitir al usuario ver el código que se ha creado para cada restricción
- El sistema debe resaltar en el contexto las frases detectadas como restricciones
- El sistema debe validar que el contexto introducido corresponde a un problema de planificación de turnos
- El sistema debe validar que cada restricción definida esté relacionada con el dominio de planificación de turnos
- El sistema debe generar una propuesta de planificación optimizada en función de las variables y restricciones definidas
- El sistema debe mostrar los resultados de forma clara, estructurada y visual.
- El sistema debe permitir al usuario filtrar la visualización por trabajador
- El sistema debe permitir la exportación de los resultados en formato Excel (.xlsx)
- El sistema debe garantizar la persistencia del estado de cada proyecto (contexto, variables y restricciones) para permitir su restauración tras recargas o cierres del navegador
- El sistema debe tener un autoguardado automático al cerrar o recargar la página que sincronice el estado actual del proyecto sin intervención del usuario

Requisitos no funcionales:

- La interfaz debe ser intuitiva y accesible para usuarios sin conocimientos técnicos.
- El sistema debe incluir notificaciones claras y retroalimentación visual para guiar al usuario
- El sistema debe ser escalable a distintos tipos de planificación sin necesidad de rediseño completo.
- La visualización de resultados debe ser clara, dinámica y permitir filtros (por ejemplo, por trabajador).
- El sistema debe manejar errores de forma robusta, mostrando mensajes comprensibles para el usuario final.
- El sistema debe estar desarrollado para entorno web.
- El sistema debe ser compatible con los principales navegadores web (Chrome, Firefox, Edge...).
- Deberá garantizar la persistencia y consistencia de los datos ante fallos o recargas inesperadas.

6.3. Historias de usuario

Para asegurar que cada requisito del sistema se refleja en una función con verdadero valor para el usuario, se han convertido los requisitos funcionales y no funcionales en historias de usuario. Estas historias describen, desde la perspectiva del usuario, qué quiere hacer y por qué le resulta útil. De esta manera, cada funcionalidad parte de una necesidad concreta, lo que facilita:

- Priorizar aquellas características más relevantes para el usuario
- Comunicar de forma clara los objetivos de desarrollo

A continuación, se presentan las historias de usuario que siguen este patrón:
COMO *rol*
QUIERO *acción*
PARA *beneficio*

Historia de usuario: Introducir contexto en lenguaje natural	
ID	HU-01
Nombre	Introducir contexto
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** poder introducir un problema de planificación en lenguaje natural **PARA** que el sistema lo interprete correctamente
Criterios de validación	El sistema permite introducir un texto libre El sistema verifica que el contexto hace referencia a una planificación de turnos

Historia de usuario: Extraer variables automáticamente	
ID	HU-02
Nombre	Extracción de variables clave
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** que el sistema detecte automáticamente las variables clave del contexto **PARA** no tener que introducirlas manualmente.
Criterios de validación	El sistema identifica entidades como retenes, días, turnos, horarios, etc. Muestra las variables extraídas

Historia de usuario: Ver resumen contexto	
ID	HU-03
Nombre	Ver resumen contexto
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** ver las variables generadas a partir de mi contexto **PARA** comprobar que está completo y correcto
Criterios de validación	Muestra un resumen con las variables extraídas (entidades, días, horarios…) Permite volver al contexto para editar

Historia de usuario: Definir restricciones en lenguaje natural	
ID	HU-04
Nombre	Definir restricción
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** definir restricciones utilizando lenguaje natural **PARA** que no sea necesario usar notación técnica o matemática.
Criterios de validación	El sistema interpreta correctamente las restricciones introducidas. El sistema valida que la restricción pertenece al dominio del problema Se notifica si no se puede interpretar

Historia de usuario: Convertir restricciones al formato Gurobi	
ID	HU-05
Nombre	Conversión a formato Gurobi
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** que el sistema convierta mis restricciones a un formato requerido por Gurobi **PARA** que se pueda realizar la optimización automáticamente.
Criterios de validación	La restricción se convierte sin errores. Se valida que Gurobi puede procesar las restricciones generadas.

Historia de usuario: Visualizar restricciones	
ID	HU-06
Nombre	Ver restricciones
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** poder ver todas las restricciones configuradas **PARA** poder revisarlas fácilmente
Criterios de validación	Se muestra una lista de restricciones Se indica su estado (activa/inactiva).

Historia de usuario: Editar restricciones	
ID	HU-07
Nombre	Editar restricciones
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** editar las restricciones que ya he definido **PARA** corregir o ajustar
Criterios de validación	El sistema permite modificar el texto original La nueva versión se interpreta y valida de nuevo

Historia de usuario: Activar o desactivar restricciones	
ID	HU-08
Nombre	Activar/desactivar restricción
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** activar o desactivar restricciones específicas **PARA** poder evaluar diferentes escenarios de planificación.
Criterios de validación	Se puede cambiar el estado de una restricción El cambio afecta al resultado de la planificación

Historia de usuario: Ver código generado por restricción	
ID	HU-09
Nombre	Ver código generado por restricción
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** ver el código que el sistema ha generado para cada restricción **PARA** entender su traducción
Criterios de validación	Se muestra el fragmento de código en formato legible

Historia de usuario: Resaltar restricciones en contexto	
ID	HU-10
Nombre	Resaltar restricciones en contexto
Prioridad	Baja
Descripción	**COMO** usuario **QUIERO** que el sistema me marque las restricciones identificadas en el contexto **PARA** identificarlas e introducirlas en la parte de restricciones
Criterios de validación	En el contexto original se marcan las frases identificadas como posibles restricciones

Historia de usuario: Generar planificación optimizada	
ID	HU-11
Nombre	Generar planificación
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** generar una propuesta de planificación optimizada **PARA** obtener una solución eficiente según las restricciones definidas.
Criterios de validación	El sistema devuelve una solución factible La solución respeta las restricciones activas Se notifica si hay conflictos o problemas de factibilidad

Historia de usuario: Visualizar resultados	
ID	HU-12
Nombre	Visualizar resultados
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** visualizar los resultados de forma clara, estructurada y visual **PARA** entender fácilmente la planificación generada.
Criterios de validación	Los resultados se muestran en tablas, calendarios u otros formatos visuales

Historia de usuario: Exportar resultados a Excel	
ID	HU-13
Nombre	Exportar resultados
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** exportar los resultados en formato Excel (.xlsx) **PARA** analizarlos o compartirlos con otros.
Criterios de validación	El sistema genera un archivo .xlsx correctamente estructurado. El archivo contiene toda la información de la planificación.

Historia de usuario: Crear nuevo proyecto	
ID	HU-14
Nombre	Crear proyecto
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** poder crear un nuevo proyecto **PARA** iniciar un espacio de trabajo
Criterios de validación	Solicita nombre de proyecto Se inicializa contexto vacío y lista de restricciones El nuevo proyecto aparece en la lista de proyectos

Historia de usuario: Guardar y cargar proyectos	
ID	HU-15
Nombre	Gestionar proyectos
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** guardar y recuperar proyectos **PARA** continuar mi trabajo en diferentes sesiones
Criterios de validación	Se puede crear, guardar, cargar y eliminar un proyecto Los datos del contexto, variables y restricciones se conservan

Historia de usuario: Eliminar proyecto	
ID	HU-16
Nombre	Eliminar proyecto
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** eliminar proyectos que ya no uso **PARA** mantener la lista ordenada y actualizada
Criterios de validación	Pide confirmación antes de borrar Desaparece de la lista de proyectos tras confirmar

Historia de usuario: Persistencia y recuperación tras recarga	
ID	HU-17
Nombre	Persistencia automática
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** que el sistema guarde automáticamente mi progreso **PARA** evitar perder mi trabajo si se recarga o cierra la página
Criterios de validación	Los datos se recuperan tras una recarga del navegador Se guarda automáticamente al cerrar la página

Historia de usuario: Validar contexto del problema	
ID	HU-18
Nombre	Validar contexto
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** que el sistema valide que el contexto pertenece a un problema de planificación de turnos **PARA** asegurarme de que pueda procesarlo bien
Criterios de validación	Se muestran errores si el contexto no se ajusta al dominio esperado

Historia de usuario: Validar restricción según el dominio	
ID	HU-19
Nombre	Validar restricciones
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** que el sistema detecte si una restricción no tiene sentido en la planificación según mi contexto **PARA** evitar errores
Criterios de validación	Se muestran errores si la restricción no se ajusta al problema de planificación

Historia de usuario: Filtrar visualización por trabajador	
ID	HU-20
Nombre	Filtrar visualización por trabajador
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** filtrar los resultados por trabajador **PARA** poder ver la planificación individual de forma más clara y ordenada
Criterios de validación	Lista desplegable para seleccionar trabajador Se actualiza la vista para mostrar solo los resultados filtrados

Historia de usuario: Editar nombre de proyecto	
ID	HU-21
Nombre	Editar nombre de proyecto
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** poder renombrar el proyecto **PARA** mantener mis proyectos organizados
Criterios de validación	Hay un botón de editar junto al título del proyecto Se abre un campo para cambiar el texto El nuevo nombre se guarda y se muestra al instante El nuevo nombre se actualiza en la lista de proyectos

Historia de usuario: Duplicar proyecto	
ID	HU-22
Nombre	Duplicar proyecto
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** duplicar un proyecto existente **PARA** crear variantes sin perder el original
Criterios de validación	Hay un botón para duplicar proyecto en la lista de proyectos Se crea un proyecto idéntico Aparece inmediatamente en la lista Tiene toda la información que tenía el proyecto anterior

Historia de usuario: Guardar manual y estado	
ID	HU-23
Nombre	Guardar manual y estado
Prioridad	Alta
Descripción	**COMO** usuario **QUIERO** poder guardar un proyecto y ver el estado **PARA** ver si tengo cambios sin guardar
Criterios de validación	Al pulsar el botón de guardar se persisten todos los datos

Historia de usuario: Botón de información/ayuda	
ID	HU-24
Nombre	Botón de información/ayuda
Prioridad	Baja
Descripción	**COMO** usuario **QUIERO** un "i" explicativo en contexto y restricciones **PARA** entender qué debo escribir
Criterios de validación	Al pasar el cursor por el icono "i", aparece un popup con especificación

Historia de usuario: Añadir restricciones detectadas	
ID	HU-25
Nombre	Añadir restricciones detectadas
Prioridad	Media
Descripción	**COMO** usuario **QUIERO** poder añadir las restricciones que detecta en el contexto **PARA** no perderlas y no tener que definirlas de nuevo
Criterios de validación	Tras analizar el contexto, muestra las restricciones detectadas El usuario puede decidir si quiere o no añadirlas al modelo

6.4. Prototipo de la interfaz

Con el objetivo de facilitar la interacción del usuario con la herramienta desarrollada, se ha diseñado un prototipo de la interfaz gráfica que permite visualizar de forma intuitiva las funcionalidades principales. Esta interfaz servirá como guía para el desarrollo posterior y como apoyo en la validación de requisitos y de la experiencia de usuario.

Para ello, se ha utilizado Figma, una herramienta de diseño de interfaces y prototipado colaborativo que funciona en la nube. Figma permite crear maquetas visuales interactivas sin necesidad de escribir código, facilitando la representación de flujos de navegación, la disposición de elementos y la lógica visual de la aplicación.

A continuación, se muestran las capturas de las principales pantallas del prototipo, acompañadas de una breve descripción de su funcionalidad.

En la primera pantalla, el usuario se encuentra con dos áreas diferenciadas. En la parte superior aparece un recuadro donde escribir el contexto del problema en lenguaje natural y un botón Subir para procesarlo. Justo debajo, se encuentran dos paneles. A la izquierda un área para redactar nuevas restricciones y un botón Agregar restricción para añadirlas, mientras que a la derecha se muestra la lista de Restricciones actuales con iconos de editar o eliminar. De este modo, desde la definición del problema hasta el envío de las restricciones, se concentra en una única vista.

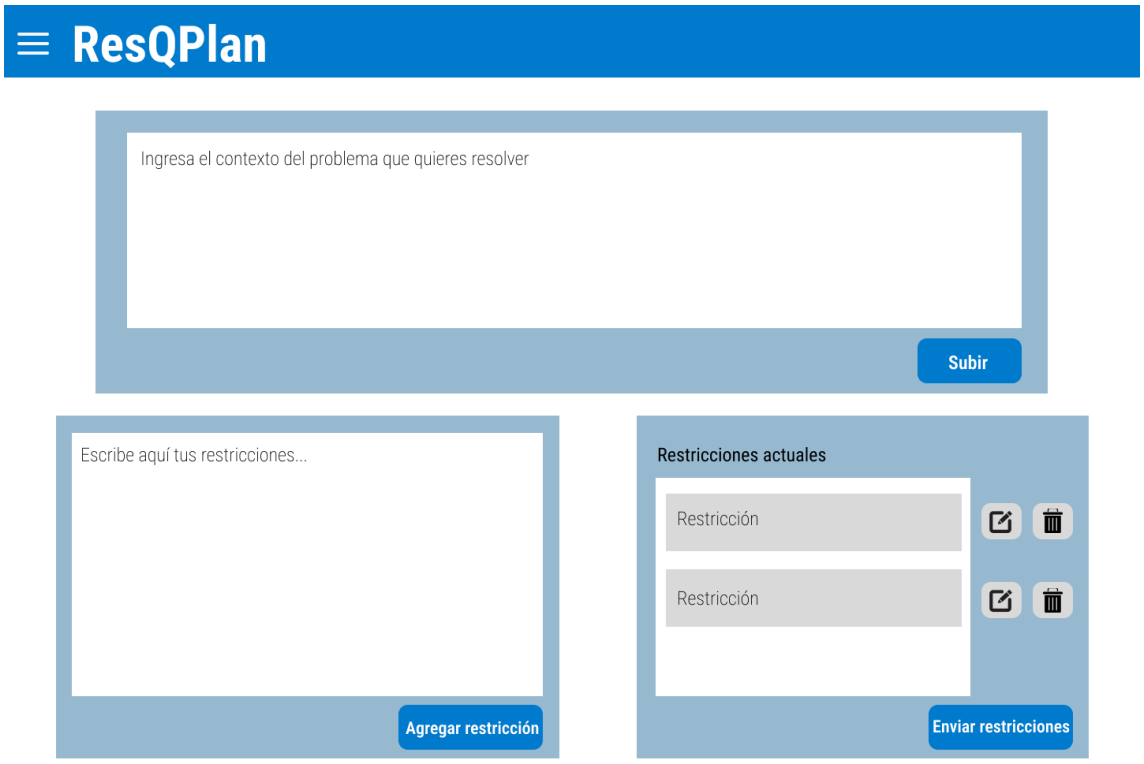


Figura 14: Prototipo página principal de la interfaz

Al hacer clic en el icono de menú situado en la esquina superior izquierda, aparece la barra lateral que permite gestionar los distintos proyectos de planificación. Un botón Nuevo proyecto abre un pequeño modal donde el usuario indica el nombre deseado y confirma o cancela la operación. Justo debajo aparece la lista de proyectos existentes, cada uno se representa con su nombre y dos botones discretos para duplicar o borrar el escenario. Al seleccionar un proyecto, su estado (contexto, variables y restricciones) se carga inmediatamente en la interfaz principal. Este diseño facilita el trabajo con múltiples escenarios sin perder el hilo del proceso.

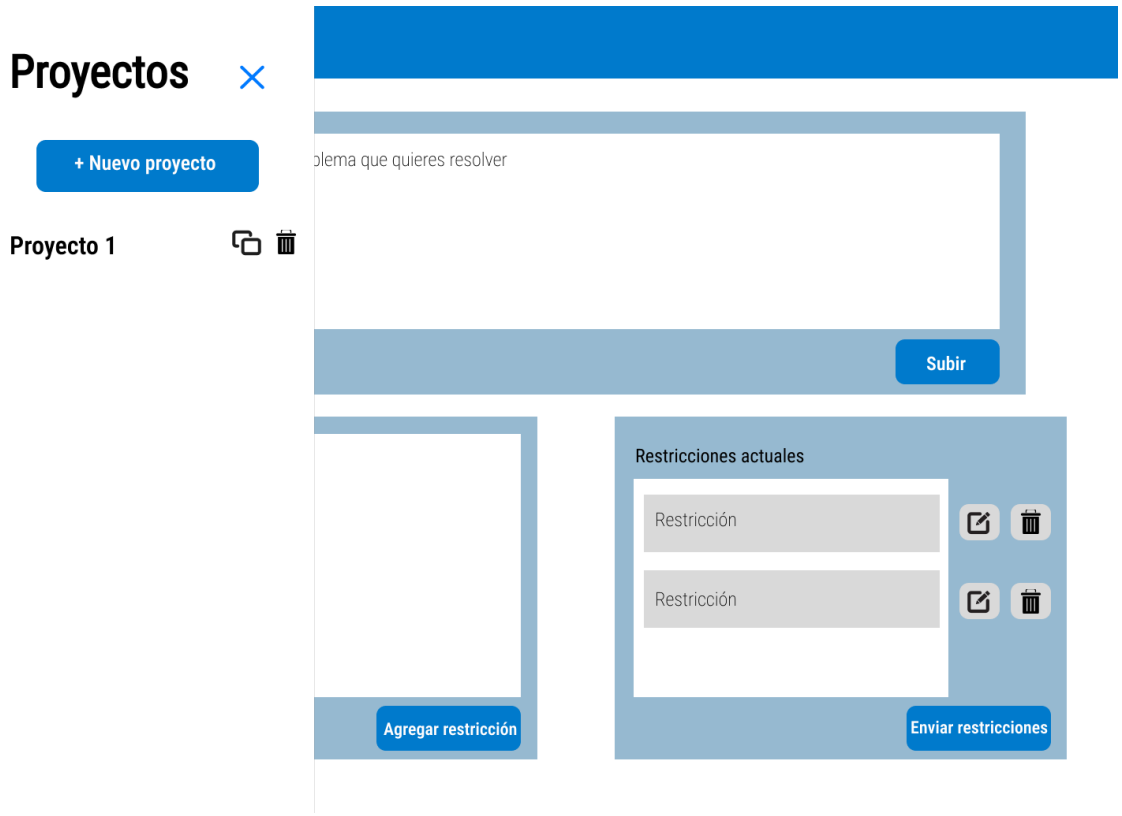


Figura 15: Prototipo sidebar para gestionar proyectos

Una vez completada la optimización, el usuario accede a la pantalla de resultados. En ella, una tabla dinámica muestra la planificación en un formato “Turno × Día”, con celdas que indican descansos o asignaciones concretas. Finalmente, un botón “Descargar Excel” facilita la exportación de los datos. Este diseño concentra toda la información relevante de manera clara y lista para publicar.

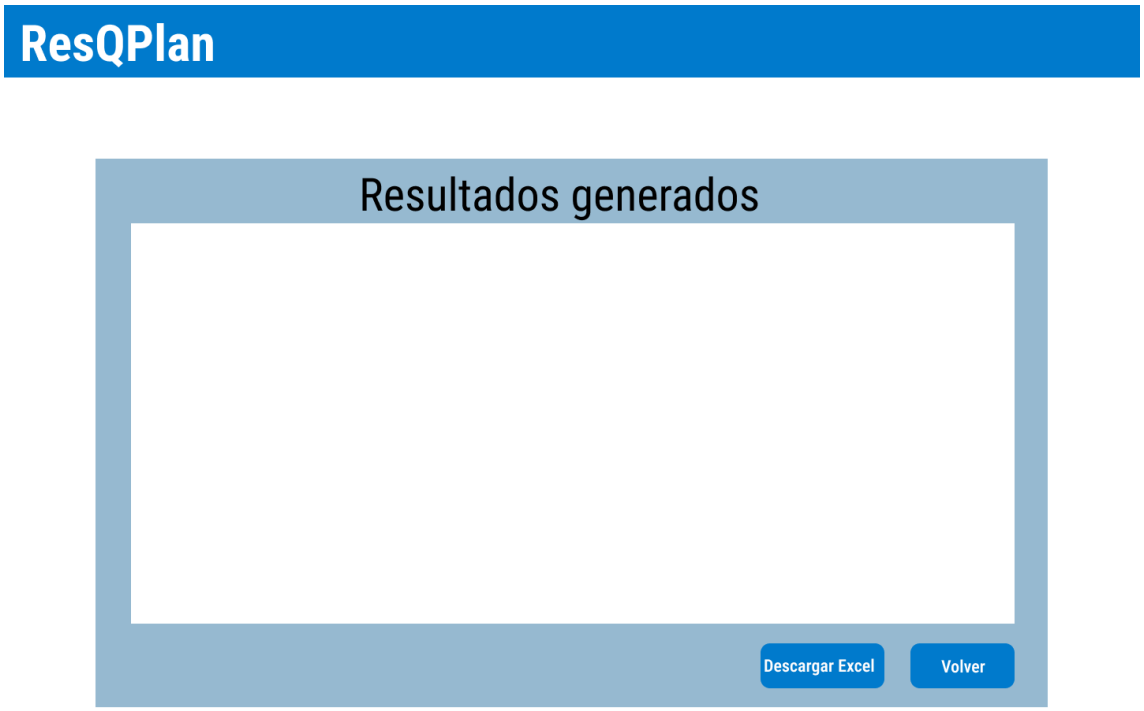


Figura 16: Prototipo pantalla de resultados

El prototipado en Figma me permitió tener una primera versión de la interfaz antes de escribir una sola línea de código. Con la maqueta interactiva vi al instante cómo encajaban los paneles, dónde resultaba más natural situar los botones de edición o eliminación y ajusté su posición hasta sentirme cómoda con el flujo de trabajo. Al mismo tiempo definí la tipografía, la paleta de colores y los iconos de Font Awesome, de modo que al arrancar el desarrollo contara con unas pautas visuales claras.

6.5. Estructura de BD

Para el almacenamiento de los proyectos generados en ResQPlan, se ha optado por utilizar MongoDB, una base de datos NoSQL orientada a documentos. Esta decisión se basa en varias razones clave relacionadas con la naturaleza del proyecto:

- Esquema flexible
  A diferencia de las bases de datos relacionales, MongoDB almacena documentos en formato JSON (BSON internamente), lo que facilita el manejo de estructuras de datos dinámicas y heterogéneas. Podemos añadir, eliminar o anidar campos sin migraciones complejas, agilizando las iteraciones y permitiendo que proyectos con configuraciones y estados intermedios variables crezcan sin limitaciones de esquema.

- Modelo orientado a documentos
  Cada proyecto se almacena como una unidad lógica que agrupa contexto, variables, restricciones… De esta manera, evitamos JOIN costosos y aseguramos que todas las modificaciones se apliquen a un documento de manera completa y fiable, sin interferir con otros documentos.

- Facilidad de integración con Flask y PyMongo
  MongoDB se integra de forma sencilla con el framework Flask a través de extensiones como PyMongo, permitiendo operaciones CRUD intuitivas y eficientes desde el backend.

- Escalabilidad horizontal
  Aunque no es un requisito en este proyecto, MongoDB permite escalar fácilmente en proyectos futuros si la herramienta se despliega en entornos con mayor carga o necesidad de almacenamiento distribuido.

La base de datos contiene una colección principal llamada *projects*, donde se almacena un documento por cada proyecto creado por el usuario. Cada documento tiene la siguiente estructura:

```
{
  "id": "uuid",
  "name": "Nombre del proyecto",
  "context": "Texto del contexto cargado",
  "detectedConstraints": [  ],
  "manualConstraints": [  ],
  "variables": {
    "decision_variables": [  ]
  },
  "validatedConstraints": [
    {
      "texto": "Texto en lenguaje natural",
      "code": "Código Gurobi generado",
      "activa": true
    }
  ],
  "gurobiState": {
    "vars": [  ],
    "cons": [  ],
    "objective": "0",
    "sense": 1
```

Figura 17: Estructura de la base de datos

Como se muestra en la imagen anterior, cada proyecto se almacena como un único documento JSON que agrupa:

- id (String): UUID público generado con uuid4()
- name (String): nombre del proyecto, tal como lo introduce el usuario
- context (String): texto en lenguaje natural que introduce el usuario, que describe la situación de planificación
- detectedConstraints (Array [String]): lista de frases que se han identificado automáticamente como posibles restricciones al procesar el context. No están aún validadas ni traducidas a código
- manualConstraints (Array [Object]): objecto por cada restricción que el usuario añade o edita manualmente antes de validarla. Cada uno incluye:
  - texto (String): frase en lenguaje natural que introdujo el usuario
  - activa (Boolean); indica si, al crearse, debería considerarse activa
- validatedConstraints (Array [Object]): tras traducirlas y validarlas, cada restricción se incluye con:
  - texto (String): la misma frase en lenguaje natural
  - code (String): el fragmento de código generado
  - activa (Boolean): si se aplica o no en la optimización
- variables (Object): parámetros extraídos del contexto, necesarios para crear y nombrar las variables de decisión en Gurobi. Contiene al menos:
  - días (Int): número de días a planificar
  - franjas (Int): número de franjas o turnos por día
  - lista_retenes, lista_conductores, lista_profesores, lista_asignaturas… (Array [String]): nombres de cada entidad del problema
  - decisión_variables (Array [String]): tras inicializar el ShiftOptimizer, este campo almacena la lista de nombres de todas las variables de decisión generadas en Gurobi
- gurobiState (Object): información mínima para reconstruir el modelo en una nueva sesión:
  - vars y cons (Array [String]): listan variables y restricciones añadidas
  - objective (String): describe la función objetivo
  - sense (Int): indica sentido (1 = minimizar, -1 = maximizar)

6.6. Arquitectura de la aplicación

La arquitectura de ResQPlan-UI garantiza un flujo eficiente y una interacción fluida entre sus componentes, lo que permite al usuario y al sistema ocuparse de las distintas tareas de forma organizada. Gracias a esta separación en módulos, el proyecto resulta escalable, fácil de mantener y abierto a la incorporación de nuevas funcionalidades.

Front-end

El módulo *front-end* es responsable de la interfaz con la que el usuario interactúa. Está construido con plantillas Jinja servidas por Flask y lógica dinámica en JavaScript puro. Su organización es la siguiente:

- Plantillas (web/templates/)
    - index.html: página principal, donde el usuario edita restricciones y controla el modelo.
    - results.html: vista de resultados tras la optimización, con tabla interactiva y controles de filtrado.
- Recursos estáticos (web/static/)
    - script.js y results.js: gestionan la edición inline de restricciones, la reconstrucción del DOM tras el análisis del modelo y el filtrado de turnos por trabajador.
    - styles.css y results.css: definen la apariencia de formularios, tablas y pop-ups, asegurando consistencia visual y usabilidad.

Tecnologías empleadas

- Flask + Jinja2 para renderizado de vistas y gestión de sesiones.
- Vanilla JavaScript con Fetch API para llamadas AJAX a la API REST.
- CSS3 con clases modulares para controles de estado (spinner, resaltado, pop-ups).

Back-end

El *back-end* expone una API RESTful que orquesta la lógica de optimización, el almacenamiento de proyectos y la traducción entre texto natural y modelo Gurobi. Sus componentes principales son:

- Punto de entrada
    - main.py: inicializa la aplicación Flask, carga la configuración y registra los blueprints de las rutas.
- Rutas y controladores (web/routes.py)
    - /api/projects: CRUD de proyectos (GET, POST, PUT, DELETE).
    - /api/edit_constraint, /api/delete_constraint, /api/view_constraint: gestionan la edición inline, eliminación y visualización de código.

    - /api/optimize, /api/download_excel: disparan la optimización con Gurobi y entregan la descarga de resultados.

- Módulo de optimización (models/shift_optimizer.py)

  Define la construcción y solución del modelo Gurobi, incluyendo la relajación de restricciones y la extracción de variables y traducción de restricciones a partir de constraint_translator.py.

- Utilidades (utils/)

    - constraint_translator.py: convierte cadenas en lenguaje natural a objetos Gurobi (variables, restricciones).

    - result_visualizer.py: formatea el resultado de Gurobi en estructuras JSON que el *front-end* renderiza como tablas y gráficos.

Tecnologías empleadas

- Flask para el servidor HTTP.
- Gurobi API para Python como solver de optimización.
- PyMongo para la conexión y operaciones con MongoDB.

6.7. Código y funcionalidades

En este apartado se presentan las funcionalidades principales de la herramienta, haciendo hincapié en aquellas que aportan un mayor valor diferencial y han supuesto retos técnicos significativos.

Reconstrucción del modelo Gurobi desde JSON

Este mecanismo permite restaurar en el cliente un proyecto de optimización completo, variables, restricciones y función objetivo, a partir de su representación en JSON, sin necesidad de intervención manual. Cuando el *front-end* recibe un objeto gurobiState que contiene: variables(vars), restricciones (cons), función objetivo (objective) y sentido(sense). Se invoca a rebuildGurobiModel(state).

```
function rebuildGurobiModel(state) {  Show usages  andreahernando
  const model :Gurobi.Model = new Gurobi.Model();
  const varsMap :{} = {};

  // 1) Variables
  state.vars.forEach(vs :T => {
    const v = model.addVar(vs.lb, vs.ub, 0, vs.type, vs.name);
    varsMap[vs.name] = v;
  });

  // 2) Restricciones
  state.cons.forEach(cs :T => {
    const lin :Gurobi.LinExpr = parseLinExpr(cs.expr, varsMap);
    model.addConstr(lin, cs.sense, cs.rhs);
  });

  // 3) Objetivo
  const obj :Gurobi.LinExpr = parseLinExpr(state.objective, varsMap);
  model.setObjective(obj, state.sense);

  model.update();
  return model;
}
```

```
function parseLinExpr(exprStr, varsMap) {  Show usages  andreahernando
  const expr :Gurobi.LinExpr = new Gurobi.LinExpr();
  exprStr.replace(/\s+/g, '').split('+').forEach(term => {
    const [coefStr, varName] = term.split('*');
    const coef :number = parseFloat(coefStr);
    if (varsMap[varName]) expr.addTerm(coef, varsMap[varName]);
  });
  return expr;
}
```

Figura 18: Reconstrucción del modelo Gurobi desde JSON

Al completar estas operaciones, el objeto window.currentGurobiModel contiene una réplica exacta del modelo original. Gracias a esto, cualquier proyecto puede exportarse a JSON y, al recargarlo. ya sea al cambiar de proyecto o al volver más tarde, se restablece al mismo estado en que se encontraba, sin necesidad de rehacer nada. De este modo, se facilita la continuidad del trabajo, la colaboración y la reproducibilidad de los experimentos de optimización.

Detección de restricciones en el texto y “click to add”

Esta funcionalidad permite, tras enviar al servidor un el texto del contexto, identificar automáticamente las restricciones presentes en el mismo, resaltarlas visualmente en el cliente y ofrecer al usuario la posibilidad de incorporarlas al modelo Gurobi con un simple clic, sin tener que volver a reescribirlas manualmente.

- Recepción de las restricciones

```
fetch( input: "/api/translate", init: {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify( value: { input_data: ctx }),
}) Promise<Response>
.then(async res : Response => {
  if (loadingOverlay) loadingOverlay.style.display = "none";

  let data;
  try {
    data = await res.json();
  } catch (_) {
    throw new Error(`Error al parsear JSON de respuesta`);
  }

  if (!res.ok) {
    const errMsg : any | string = data.message ?? data.error ?? `Error HTTP ${res.status}`;
    throw new Error(errMsg);
  }

  return data;
}) Promise<any>
```

Figura 19: Recepción de restricciones

En la respuesta JSON que recibe el cliente, se incluye un campo detectedConstraints que contiene un listado de todas las restricciones identificadas automáticamente en el fragmento de texto enviado.

- Resaltado de las restricciones en el editor

```
if (p.result.detected_constraints && p.result.detected_constraints.length) {
  contextWarning.style.visibility = 'visible';

  let html : string = contextInput.innerText;

  p.result.detected_constraints.forEach(nl => {
    const esc = nl.replace(/[-\/\\^$*+?.()|[\]{}]/g, '\\$&');
    const regex : RegExp = new RegExp( pattern: `(${esc})`, flags: 'g');
    html = html.replace(
      regex,
      replaceValue: `<mark class="highlight clickable" data-nl="${nl}">$1</mark>`
    );
  });
```

Figura 20: Resaltado de restricciones

Este fragmento comprueba si detectedConstraints contiene elementos y, de ser así, hace visible un aviso junto al texto. A continuación, captura el texto original, itera sobre cada

restricción detectada, escapa sus caracteres especiales para crear una expresión regular global y envuelve todas sus ocurrencias en una etiqueta <mark> con la clase highlight clickable y un atributo data-nl, de modo que queden resaltadas y sean clicables. Por último, sustituye el contenido del contenedor de texto por este HTML enriquecido.

- "Clic to add"

```
contextInput.querySelectorAll( selectors: 'mark.highlight.clickable').forEach( callbackFn: mark :Element => {
  mark.style.cursor = 'pointer';
  mark.title = 'Haz clic para agregar esta restricción';

  mark.addEventListener( type: 'click', listener: async () => {
    const nl :string = mark.dataset.nl;
    if (mark.classList.contains('added')) return;  // ya agregado

    const confirmar :boolean = confirm(`¿Quieres agregar la restricción:\n\n"${nl}"?`);
    if (!confirmar) return;

    const progressContainer :HTMLElement = document.getElementById( elementId: 'progress-container');
    const progressBar :HTMLElement       = document.getElementById( elementId: 'progress-bar');
    const progressLabel :HTMLElement     = document.getElementById( elementId: 'progress-label');
    progressBar.max   = 1;
    progressBar.value = 0;
    progressLabel.textContent = `Procesando 0 de 1…`;
    progressContainer.style.display = 'block';

    mark.classList.add('adding');
    mark.textContent = '';

    try {
      await intentarConvertir(nl);

      const textoPlano :string = contextInput.innerText;
      contextInput.innerHTML = textoPlano;
      sessionStorage.setItem('savedContext', contextInput.innerText);
      await autoSaveProject();
      showToast( type: 'success', message: `"${nl}" agregada correctamente.`);

    } catch (err) {
      console.error(err);
      mark.classList.remove( tokens: 'adding');
      mark.textContent = nl;
      alert('Error al agregar la restricción. Por favor, inténtalo de nuevo.');

    } finally {
      progressContainer.style.display = 'none';
    }
  });
});



} else {
  contextWarning.style.visibility = 'hidden';
}
```

Figura 21: Añadir restricciones detectadas en el contexto

Cuando el usuario hace clic en un fragmento resaltado, se solicita confirmación para añadir la restricción. Si confirma, se muestra la barra de progreso y se invoca la función intentarConvertir(), que se encarga de inyectar la nueva restricción en el modelo Gurobi.

Gestión de proyectos (CRUD)

La aplicación ofrece un sistema completo de gestión de proyectos, que incluye creación, listado, edición, duplicado y eliminación.

El usuario puede:

- **Crear** un nuevo proyecto pulsando "Nuevo proyecto" en el sidebar, introducir un nombre obligatorio y enviar POST /api/projects. Tras la respuesta, se limpia el contexto y se inicializan variables y restricciones asociadas.

```
async function createProject(name) {  Show usages  andreahernando
  const res : Response  = await fetch( input: "/api/projects",  init: {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify( value: {
      name,
      context: "",
      detectedConstraints: [],
      manualConstraints: [],
      variables: {},
      gurobiState: { vars: [], cons: [], objective: "0", sense: 1 }
    })
  });
  return res.json();
}
```

```
createProjectBtn.addEventListener( type: "click",  listener: async () => {
  const name : string  = newNameInput.value.trim();
  if (!name) return showToast( type: "warning",  message: "Ponle un nombre al proyecto");
  const proj = await createProject(name);

  contextReady = false;
  if (constraintInput) {
    constraintInput.readOnly = true;
    constraintInput.value = "";
  }
  renderContextControls();


  currentProjectId = proj.id;
  currentProjectName = proj.name;
  newPrompt.style.display = "none";
  // Limpiar estado previo
  contextInput.innerText = "";
  renderContextControls();
  detectedList.innerHTML = "";
  document.querySelector( selectors: ".restricciones-list").innerHTML = "";
  detectedPanel.style.display = "none";
  sessionStorage.setItem("restricciones", JSON.stringify( value: []));
  sessionStorage.setItem("variables", JSON.stringify( value: {}));

  await refreshProjectOptions();
  showToast( type: "success",  message: `Proyecto "${proj.name}" creado`);
});
```

Figura 22: Crear nuevo proyecto

- **Listar** proyectos, invocando GET /api/projects en refreshProjectOptions(), que actualiza dinámicamente el panel lateral con cada nombre y opciones de acción.

```
projectList.innerHTML = "";
const projects = await listProjects();
projects.forEach(p => {
  const li :HTMLLIElement = document.createElement( tagName: "li");
  li.textContent = p.name;
  li.dataset.id = p.id;
```

```
async function listProjects() {  Show usages  andreahernando
  const res :Response = await fetch( input: "/api/projects");
  const { projects } = await res.json();
  return projects;
}
```

Figura 23: Listar proyectos

- **Editar nombre**, haciendo clic en el icono de lápiz junto al título: se reemplaza el texto por un input inline, y al guardar se envía PUT /api/projects/{id}, actualizando la sesión y el DOM.

```
saveBtn.onclick = async () => {
  const newName :string = input.value.trim();
  if (!newName || newName === currentProjectName) return cancelBtn.click();

  try {
    await fetch( input: `/api/projects/${currentProjectId}`, init: {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify( value: { name: newName })
    });
    currentProjectName = newName;
    sessionStorage.setItem('currentProjectName', newName);
    titleSpan.textContent = currentProjectName;

    document.querySelectorAll( selectors: "#project-list li").forEach( callbackfn: li :Element => {
      if (li.dataset.id === currentProjectId) {
        li.firstChild.textContent = currentProjectName;
      }
    });
```

Figura 24: Editar nombre de proyecto

- **Duplicar** un proyecto con el botón de clonar, que emplea loadProject para cargar todos los datos, genera un nuevo nombre con sufijo si hace falta y lanza POST /api/projects con el payload completo.

```
duplicateBtn.addEventListener( type: "click", listener: async (e : MouseEvent ) => {
  e.stopPropagation();

  const orig = await loadProject(p.id);
  if (orig.error) return showToast( type: "error", message: "No se pudo cargar el proyecto original.");

  let newName : string = `${orig.name}1`;

  const allNames = (await listProjects()).map(x => x.name);
  let suffix : number = 1;
  while (allNames.includes(newName)) {
    newName = `${orig.name} ${++suffix}`;
  }

  const clonePayload : {…} = {
    name: newName,
    context: orig.context,
    detectedConstraints: orig.detectedConstraints,
    manualConstraints: orig.manualConstraints,
    variables: orig.variables,
    gurobiState: orig.gurobiState
  };
```

```
    try {
      const res : Response = await fetch( input: "/api/projects", init: {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(clonePayload)
      });
      const result = await res.json();
      if (!res.ok) throw new Error(result.error || `HTTP ${res.status}`);

      showToast( type: "success", message: `Proyecto duplicado como "${newName}"`);
      await refreshProjectOptions();
    } catch (err) {
      console.error(err);
      showToast( type: "error", message: "Error al duplicar el proyecto.");
    }
  });

  li.appendChild(duplicateBtn);
```

Figura 25: Duplicar proyecto

- **Eliminar** mediante DELETE /api/projects/{id}, tras confirmación, y automantiene la lista fresca.

```
deleteBtn.addEventListener( type: "click", listener: async (e : MouseEvent ) => {
  e.stopPropagation();
  if (!confirm(`¿Borrar el proyecto "${p.name}"?`)) return;
  await deleteProject(p.id);
  showToast( type: "warning", message: `Proyecto "${p.name}" eliminado`);
  await refreshProjectOptions();
});
```

Figura 26: Eliminar proyecto

Edición *inline* de restricciones manuales

Cuando el usuario pulsa el icono del lápiz junto a una restricción, el texto se convierte en un <input> editable. Al guardar, se muestra un spinner mientras se envía la petición al backend, y si todo va bien se actualiza el texto y se marca el proyecto como “modificado”. Si el usuario presiona Escape o el botón de cancelar, se restaura el texto original.

```
editButton.addEventListener( type: "click",  listener: () => {
  const oldText : string  = label.textContent;
  const inputEdit : HTMLInputElement  = document.createElement( tagName: "input");
  inputEdit.type = "text";
  inputEdit.value = oldText;
  inputEdit.classList.add("edit-input");

  inputEdit.addEventListener( type: "keydown",  listener: (e : KeyboardEvent ) => {
   if (e.key === "Enter") {
     e.preventDefault();
     saveBtn.click();
   }
   if (e.key === "Escape") {
     e.preventDefault();
     cancelBtn.click();
   }
  });
```

```
saveBtn.addEventListener( type: "click",  listener: async () => {
  const newText : string  = inputEdit.value.trim();
  if (!newText || newText === oldText) return cancelBtn.click();
  const originalContent : string  = saveBtn.textContent;
  saveBtn.textContent = "";
    cancelBtn.style.display = "none";
  saveBtn.disabled = true;
  cancelBtn.disabled = true;

  const spinner : HTMLSpanElement  = document.createElement( tagName: "span");
  spinner.classList.add("save-spinner");
  saveBtn.appendChild(spinner);

  try {
    const res : Response  = await fetch( input: "/api/edit_constraint",  init: {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify( value: { old_nl: oldText, new_nl: newText })
    });
    const result = await res.json();

    if (res.ok && result.success) {
      label.textContent = newText;
      inputEdit.replaceWith(label);
      inlineControls.replaceWith(controlsWrapper);
      guardarRestricciones();
      markDirty();
      showToast("success", "Restricción editada correctamente.");
```

Figura 27: Edición de restricciones manuales

Al pulsar el icono de eliminar, se muestra un confirm(). Si el usuario acepta, se envía la petición de borrado. Al obtener confirmación del servidor, se elimina el <li> correspondiente, se dispara el guardado automático y, en su caso, se oculta la nota de "restricciones relajadas"

```
deleteButton.addEventListener( type: "click",  listener: async () => {
  const confirmDelete :boolean  = confirm("¿Estás seguro de que quieres eliminar esta restricción?");
  if (!confirmDelete) return;

  try {
    const res :Response  = await fetch( input: "/api/delete_constraint",  init: {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify( value: { nl: label.textContent })
    });
    const result = await res.json();

    if (res.ok && result.success) {
      li.remove();
      markDirty()
      guardarRestricciones();
      updateManualConstraintsInfo()
      showToast("success", "Restricción eliminada correctamente.");

      await autoSaveProject();
      const anyRelaxed :boolean  = !!document.querySelector( selectors: ".restriccion-item.relaxed-highlight");

      if (!anyRelaxed) {
        const note :HTMLElement  = document.getElementById( elementId: "relaxed-note");
        if (note) note.remove();
      }
    } else {
```

Figura 28: Eliminar restricción

Cuando el usuario pulsa el icono </>, se solicita al servidor el fragmento de código generado para esa restricción. Si la respuesta es correcta, se crea un pequeño popup junto al botón que contiene el <pre> con el código; al hacer clic fuera o sobre el "×", el popup desaparece.

```
viewButton.addEventListener( type: "click",  listener: async (e :MouseEvent ) => {
  const constraintText :string  = label.textContent;

  try {
    const res :Response  = await fetch( input: "/api/view_constraint",  init: {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify( value: { nl: constraintText })
    });

    const result = await res.json();

    if (res.ok && result.code) {
      document.querySelectorAll( selectors: ".code-popup").forEach( callbackFn: p :Element  => p.remove());

      const popup :HTMLDivElement  = document.createElement( tagName: "div");
      popup.classList.add("code-popup");
      popup.innerHTML = `
        <button class="close-popup">×</button>
        <pre>${result.code}</pre>
      `;
      document.body.appendChild(popup);

      const buttonRect :DOMRect  = e.target.getBoundingClientRect();
      const popupWidth :number  = popup.offsetWidth;
      const spacing :number  = 8;

      const panel = e.target.closest( selector: ".panel");
      const panelRect :DOMRect  = panel.getBoundingClientRect();
```

Figura 29: Ver código generado para una restricción

Visualización de resultados con filtro por trabajador

Una vez finalizada la optimización, la interfaz construye internamente un resumen de la solución a partir de las claves devueltas—cada una descompuesta en trabajador, día y franja horaria—y despliega una tabla donde cada celda muestra los turnos asignados o indica “Descanso” si no hay coincidencias. Encima de la tabla aparece un desplegable con la lista de todos los trabajadores, que permite filtrar la vista para mostrar únicamente las filas en las que aparece el seleccionado. Además, si se detectaron restricciones relajadas, se muestra un panel de advertencia. Para completar el flujo, un botón habilita la descarga inmediata de un informe en Excel con todos los resultados. Esta funcionalidad facilita la interpretación de grandes volúmenes de datos, permite centrarse en la información relevante de cada trabajador y agiliza la exportación de los resultados.

6.8. Ingeniería de prompts

Con el objetivo de mejorar la robustez y precisión del sistema de extracción de variables y traducción de restricciones, añadí tres modificaciones clave en el prompt original que dieron como resultado una mejora en la coherencia de la salida y la reducción de errores en escenarios no válidos.

Primero, añadí un mecanismo de validación de contexto que obliga al modelo a confirmar si el texto de entrada describe efectivamente un problema de planificación de turnos. Gracias a la instrucción:

> ⚠️ IMPORTANTE: Si el texto no describe un problema de turnos, responde únicamente con: {"error": "El texto no describe un problema de turnos válido."}.

De esta manera se evitan salidas erróneas o incoherentes cuando el usuario introduce un texto arbitrario o no relacionado con horarios.

A continuación, implementé la validación de restricciones. Ahora el modelo verifica que cada restricción en lenguaje natural esté relacionada con las variables o recursos definidos en el problema. Para ello, se le indica:

> ⚠️ IMPORTANTE: Si la restricción en lenguaje natural no se corresponde con ninguna variable o recurso del problema, responde SÓLO con: {"error": "La restricción no aplica al contexto proporcionado."}

De esta manera también evitamos que se meta texto arbitrario o restricciones que no tengan que ver con el problema actual, y se generen errores internos al intentar traducir y no encontrar la variable correspondiente.

Por último, activamos la extracción automática de restricciones. El prompt solicita explícitamente que el modelo identifique y liste en un array todas las oraciones del texto original que expresen restricciones válidas (p. ej., “cada profesor solo puede impartir un máximo de dos franjas diarias”). Si no se detecta ninguna, la clave detected_constraints permanecerá vacía o desaparecerá. Esta funcionalidad no solo simplifica el análisis previo, sino que también mejora la trazabilidad, ya que permite revisar y ajustar manualmente las oraciones originales antes de generar el código correspondiente.

"En el caso que detectes alguna restricción en el contexto (Oraciones meramente descriptivas (horarios, cantidades, desplazamientos) no se incluirán.),añade la siguiente clave (SOLO EN EL CASO QUE LO DETECTES):"
"4) \"detected_constraints\" ⇒ lista de cadenas, **sin** convertirlas en código, "
" que contenga todas las oraciones del texto de entrada que parezcan "
" restricciones en lenguaje natural.\n\n"
"Si no se detecta ninguna restricción válida, devuelve 'detected_constraints': []` o no incluyas esa clave."

7. Producto final

En esta sección se presenta la versión final de la herramienta desarrollada. A través de una serie de capturas de pantalla, mostraremos el flujo de trabajo completo que debe seguir un usuario: desde la creación de un proyecto hasta la generación y exportación de la planificación optimizada. Cada paso irá acompañado de una breve explicación de la interfaz y de las acciones a realizar, de modo que quede claro cómo interactuar con la aplicación en un entorno real de uso.

Al entrar en ResQPlan, el usuario se topa con una pantalla limpia: en la parte superior, un encabezado azul muestra el nombre de la aplicación y un botón de menú para gestionar proyectos. Justo debajo, aparece un cuadro de texto donde se escribe el contexto en lenguaje natural, por ejemplo: “Organizar el horario de los 22 retenes de La Palma, con dos turnos de 08:00–16:00 y 16:00–00:00, para los próximos 15 días”. Y, al pulsar Subir, envía esa información al motor de planificación.

En la parte inferior, hay dos áreas que permiten gestionar las restricciones: a la izquierda se escribe cada regla (p. ej., “ningún reten puede trabajar más de dos días seguidos”) y se añade con Añadir; a la derecha se muestra la lista de restricciones ya introducidas. Tanto junto al cuadro de contexto como en la sección de restricciones, hay un pequeño icono de información que, al pasar el cursor sobre él, explica qué tipo de texto iría en cada campo.

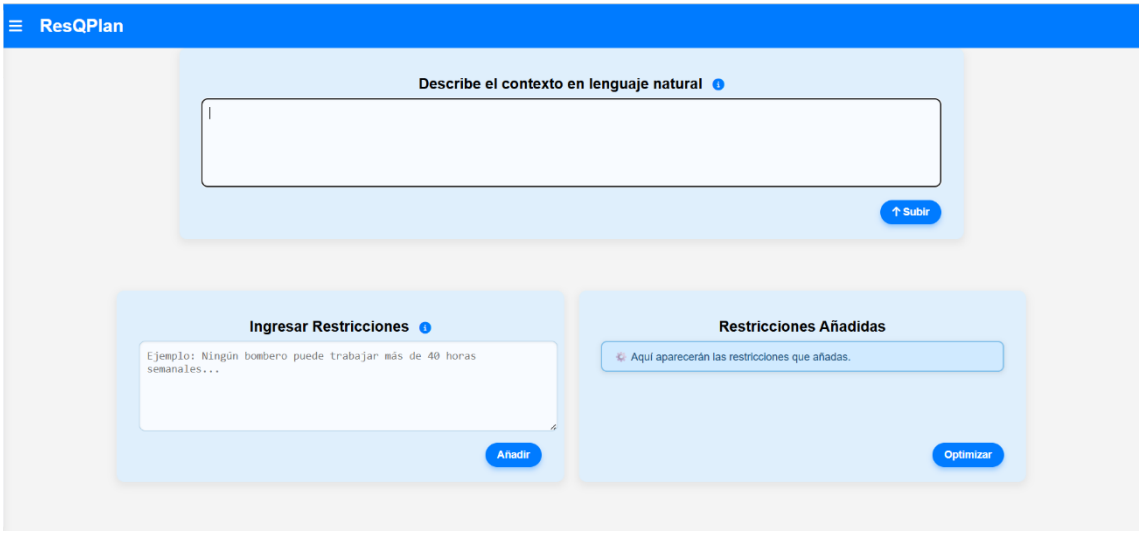


Figura 30: Página principal ResQPlan

Si el usuario ejecuta una planificación sin pertenecer a un proyecto, toda la configuración se pierde al cerrar la sesión. Para guardar los ajustes y retomarlos más tarde, debe crear o seleccionar un proyecto. Al hacer clic en el botón de menú, se despliega una barra lateral donde aparecen:

- Listado de proyectos existentes, cada uno con opciones para duplicar o eliminar.
- Botón Nuevo proyecto, que abre un cuadro de diálogo con un campo para introducir el nombre y dos botones: Crear o Cancelar.

Una vez creado un proyecto, la vista vuelve al panel principal (idéntica a la imagen anterior) pero con el nombre del proyecto visible en el encabezado, acompañado de un

ícono de editar para renombrarlo y un indicador de estado que confirma el guardado automático de los cambios.

Figura 31: Sidebar para gestionar proyectos

Por ejemplo, al cargar el proyecto **"retenes la palma"**, la pantalla principal se actualizará mostrando el nombre del proyecto en el encabezado y sus datos asociados, tal y como se ve en la siguiente captura:

Figura 32: Carga de un proyecto ya guardado

Con el contexto ya cargado, aparecen ahora dos botones adicionales:

- Editar contexto: al pulsarlo, el cuadro de texto vuelve a estar activo para modificar la descripción original. Al subir el nuevo contexto, todas las restricciones previas se borrarán automáticamente para evitar conflictos con la nueva definición del problema.
- Ver resumen: al hacer clic, se abre una ventana con las variables que el motor ha identificado (por ejemplo, número de retenes, turnos definidos, duración de los turnos, etc.), tal y como se muestra en la siguiente imagen:

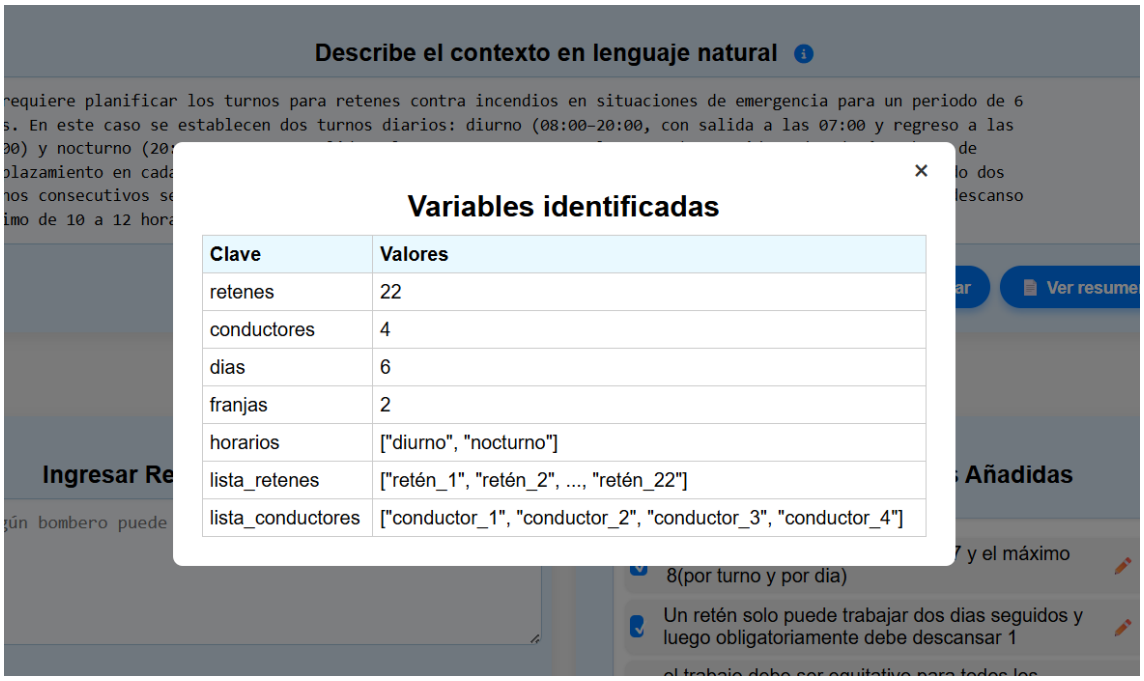


Figura 33: Variables creadas para el problema

Al pulsar Subir tras escribir el contexto, el sistema analiza el texto y, si detecta posibles restricciones en él, muestra un mensaje informativo y subraya esas frases directamente en el recuadro. Al hacer clic sobre cualquiera de los fragmentos resaltados, aparece un diálogo de confirmación: si el usuario acepta, esa frase se envía al motor para traducirse automáticamente a código de restricción y añadirse al listado.

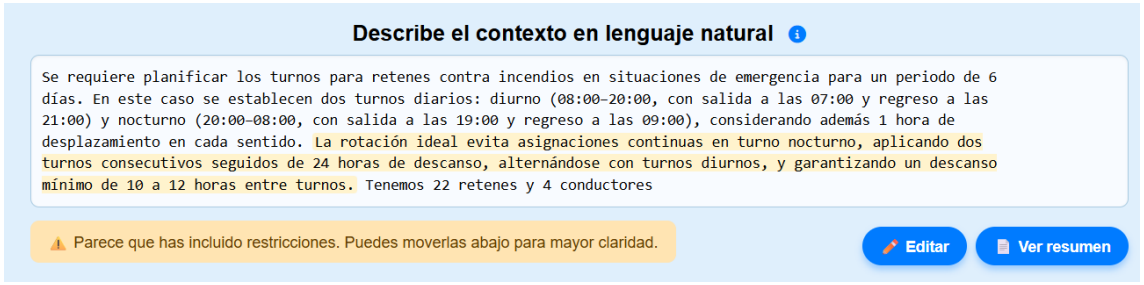


Figura 34: Resaltado de restricciones en contexto

En el panel de Restricciones añadidas se listan todas las reglas que el usuario ha incorporado al modelo. Junto a cada una encontrarás:

- Un *checkbox* que permite activarla o desactivarla, de modo que puedas probar la optimización con distintos conjuntos de restricciones.
- Un botón de editar que abre un editor de texto *inline*. Al modificar el enunciado y pulsar Guardar, esa frase se envía al motor para traducirse a código de restricción; mientras tanto, aparece un *spinner* indicando el proceso. Cuando termina, la restricción se actualiza con el nuevo texto. Si en lugar de guardar eliges Cancelar, volverá a mostrarse la versión anterior sin cambios.
- Un botón de Eliminar que borra inmediatamente la restricción del listado.
- Un botón de Mostrar código que despliega, en un cuadro de texto o modal, el fragmento de código generado por el motor para esa restricción.

Figura 35: Panel de restricciones añadidas

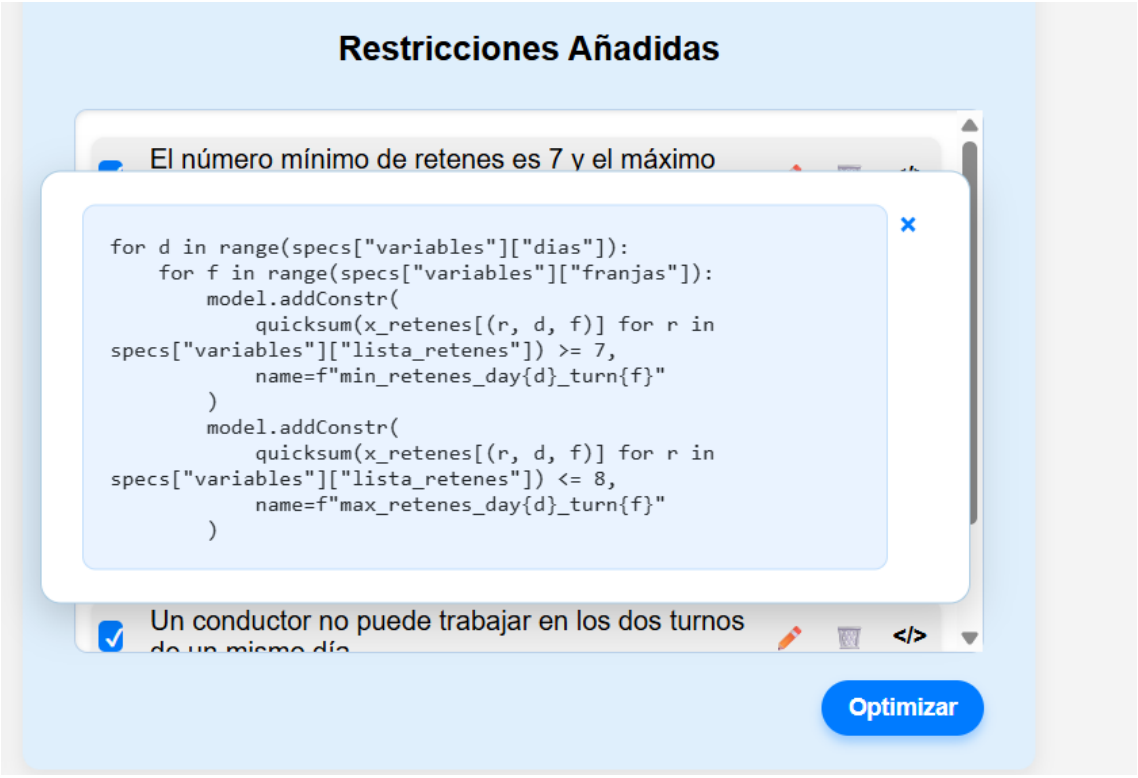


Figura 36: Código generado para restricción en ventana emergente

Una vez definidas la descripción y las restricciones, al pulsar Optimizar la aplicación muestra los resultados en una tabla "Turno × Día". Cada celda detalla las asignaciones para ese turno y día (por ejemplo, qué retenes y conductores están de servicio). En la parte inferior, los botones Descargar Excel y Volver facilitan, respectivamente, exportar el informe completo y regresar a la vista de contexto y restricciones.

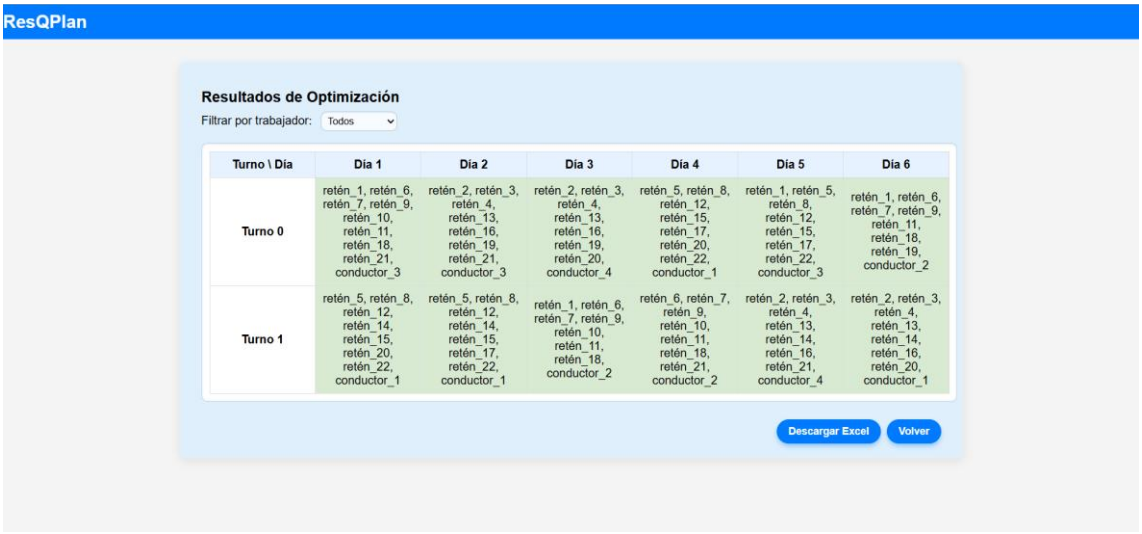
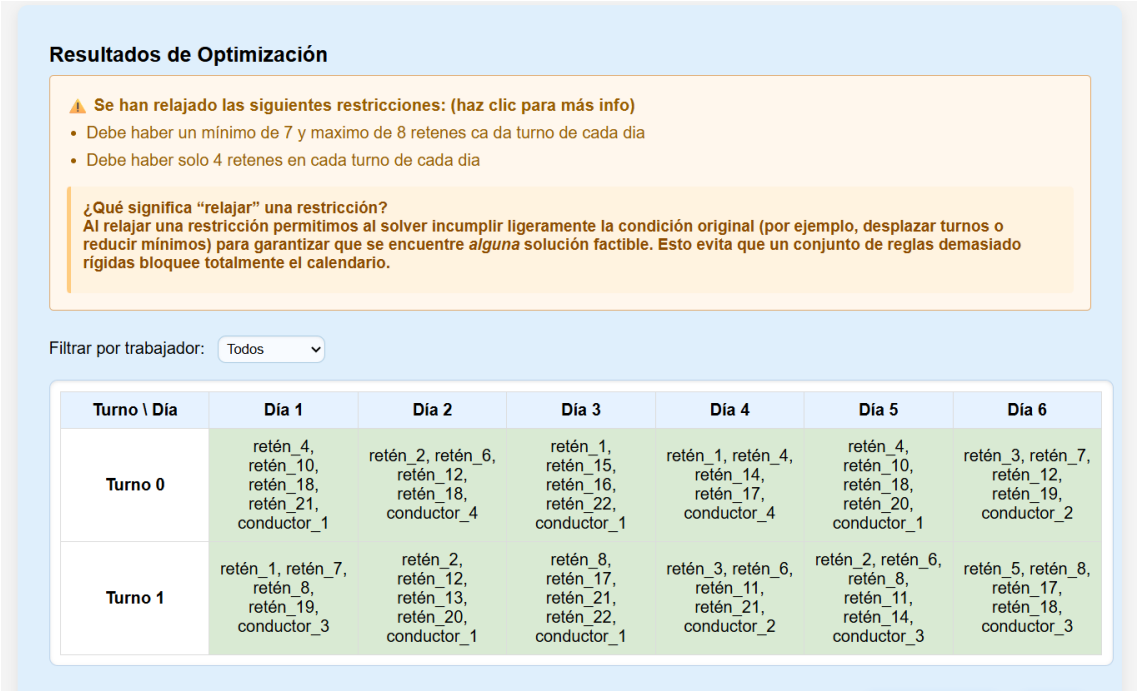


Figura 37: Resultado de la optimización

Un desplegable "Filtrar por trabajador" permite ver únicamente la planificación de un retén o conductor concreto.

Figura 38: Resultado filtrado para retén_2

Además, si las restricciones definidas resultan demasiado estrictas o contradictorias y el sistema necesita relajar alguna para generar un plan viable, aparece un cuadro de aviso justo encima de la tabla indicando qué restricciones se han suavizado. Al hacer clic sobre ese aviso, se muestra una breve explicación de qué implica "relajar" una restricción.



Turno \ Día	Día 1	Día 2	Día 3	Día 4	Día 5	Día 6
Turno 0	retén_4, retén_10, retén_18, retén_21, conductor_1	retén_2, retén_6, retén_12, retén_18, conductor_4	retén_1, retén_15, retén_16, retén_22, conductor_1	retén_1, retén_4, retén_14, retén_17, conductor_4	retén_4, retén_10, retén_18, retén_20, conductor_1	retén_3, retén_7, retén_12, retén_19, conductor_2
Turno 1	retén_1, retén_7, retén_8, retén_19, conductor_3	retén_2, retén_12, retén_13, retén_20, conductor_1	retén_8, retén_17, retén_21, retén_22, conductor_1	retén_3, retén_6, retén_11, retén_21, conductor_2	retén_2, retén_6, retén_8, retén_11, retén_14, conductor_3	retén_5, retén_8, retén_17, retén_18, conductor_3

Figura 39: Resultado de la optimización, con solución relajada

Al pulsar Volver, la aplicación regresa a la pantalla principal y, en el panel de Restricciones añadidas, las reglas que fueron relajadas aparecen sombreadas en naranja para facilitar su identificación.

Figura 40: Resaltado de restricciones relajadas

7.1. Resultados y validaciones

En esta sección se presentan los resultados obtenidos al aplicar la herramienta de generación de turnos en distintos escenarios, con el fin de evaluar su eficacia, robustez y escalabilidad.

Caso retenes:

En el caso de la planificación de turnos para retenes contra incendios, el sistema ha demostrado una alta capacidad para resolver problemas complejos en contextos operativos reales. Este escenario incluye diferentes tipos de personal (retenes y conductores), franjas horarias diferenciadas con condiciones específicas de entrada y salida, y criterios estrictos de rotación y equidad.

El modelo ha gestionado eficazmente un volumen considerable de combinaciones posibles, al coordinar 22 retenes y 4 conductores durante 6 días con dos turnos diarios, generando asignaciones óptimas que respetan múltiples dependencias lógicas y operativas.

La presencia de condiciones de alternancia, descansos obligatorios, equilibrio de carga laboral y compatibilidades entre roles y turnos convierte este caso en un problema de planificación altamente no trivial. A pesar de ello, el sistema ha sido capaz de generar soluciones válidas en cuestión de segundos, evidenciando su robustez y escalabilidad.

Este ejemplo pone de relieve la potencia del optimizador desarrollado, capaz de abordar problemas reales con múltiples dimensiones, restricciones complejas y necesidades logísticas críticas.

Resultados de Optimización

Filtrar por trabajador: Todos

Turno \ Día	Día 1	Día 2	Día 3	Día 4	Día 5	Día 6
Turno 0	retén_1, retén_6, retén_7, retén_9, retén_10, retén_11, retén_18, retén_21, conductor_3	retén_2, retén_3, retén_4, retén_13, retén_16, retén_19, retén_21, conductor_3	retén_2, retén_3, retén_4, retén_13, retén_16, retén_19, retén_20, conductor_4	retén_5, retén_8, retén_12, retén_15, retén_17, retén_20, retén_22, conductor_1	retén_1, retén_5, retén_8, retén_12, retén_15, retén_17, retén_22, conductor_3	retén_1, retén_6, retén_7, retén_9, retén_11, retén_18, retén_19, conductor_2
Turno 1	retén_5, retén_8, retén_12, retén_14, retén_15, retén_20, retén_22, conductor_1	retén_5, retén_8, retén_12, retén_14, retén_15, retén_17, retén_22, conductor_1	retén_1, retén_6, retén_7, retén_9, retén_10, retén_11, retén_18, conductor_2	retén_6, retén_7, retén_9, retén_10, retén_11, retén_18, retén_21, conductor_2	retén_2, retén_3, retén_4, retén_13, retén_14, retén_16, retén_21, conductor_4	retén_2, retén_3, retén_4, retén_13, retén_14, retén_16, retén_20, conductor_1

Figura 41: Resultado de optimización para el caso de retenes de La Palma

Caso colegio:

En el caso del centro educativo, el sistema ha resuelto con éxito un problema de planificación horaria que implica una elevada complejidad combinatoria. La tarea consistía en generar un horario completo para 12 grupos de clase (desde 1.ºA hasta 6.ºB), distribuidos en 5 días lectivos con 5 franjas horarias diarias, a partir de un claustro de 15 profesores que imparten 9 asignaturas distintas.

La generación del horario debía contemplar numerosas restricciones lógicas y pedagógicas, como la exclusividad de asignación por franja para cada curso y profesor, la compatibilidad entre horarios y asignaturas, y el respeto a las cargas lectivas individuales. Además, se requería que cada asignatura no se repitiera más de una vez al día por grupo, garantizando una planificación equilibrada y funcional.

El sistema ha demostrado su capacidad para manejar un gran volumen de datos estructurados y restricciones interrelacionadas, proporcionando una solución válida y coherente en pocos segundos.

Este caso pone en valor la potencia y adaptabilidad del optimizador, que puede aplicarse con éxito a contextos educativos reales, automatizando una tarea tradicionalmente compleja y propensa a errores, y liberando a los equipos directivos de una carga de trabajo significativa.

Filtrar por trabajador: 1ºA

Turno \ Día	Día 1	Día 2	Día 3	Día 4	Día 5
Turno 0	Nuria / 1ºA / Conocimiento del medio	Nuria / 1ºA / Matemáticas	Nuria / 1ºA / Matemáticas	Nuria / 1ºA / Conocimiento del medio	Carlos / 1ºA / Educación física
Turno 1	Nuria / 1ºA / Lengua	María / 1ºA / Emocrea	Nuria / 1ºA / Lengua	María / 1ºA / Emocrea	Nuria / 1ºA / Lengua
Turno 2	Irene / 1ºA / Inglés	Carlos / 1ºA / Educación física	Nuria / 1ºA / Conocimiento del medio	Carlos / 1ºA / Religión	Nuria / 1ºA / Arts
Turno 3	Nuria / 1ºA / Matemáticas	Irene / 1ºA / Inglés	Irene / 1ºA / Inglés	Nuria / 1ºA / Lengua	Nuria / 1ºA / Matemáticas
Turno 4	Nuria / 1ºA / Arts	Nuria / 1ºA / Lengua	Nuria / 1ºA / Arts	Carlos / 1ºA / Educación física	Nuria / 1ºA / Conocimiento del medio

Figura 43: Resultado caso colegio curso 1ºA

Filtrar por trabajador: 1ºB

Turno \ Día	Día 1	Día 2	Día 3	Día 4	Día 5
Turno 0	Marta / 1ºB / Conocimiento del medio	Marta / 1ºB / Inglés	Marta / 1ºB / Arts	Carlos / 1ºB / Educación física	Marta / 1ºB / Conocimiento del medio
Turno 1	Marta / 1ºB / Arts	Marta / 1ºB / Lengua	Marta / 1ºB / Lengua	Marta / 1ºB / Lengua	Irene / 1ºB / Religión
Turno 2	Marta / 1ºB / Inglés	Marta / 1ºB / Conocimiento del medio	María / 1ºB / Emocrea	Marta / 1ºB / Arts	Marta / 1ºB / Inglés
Turno 3	Marta / 1ºB / Lengua	Marta / 1ºB / Matemáticas	Carlos / 1ºB / Educación física	Marta / 1ºB / Conocimiento del medio	Marta / 1ºB / Lengua
Turno 4	Marta / 1ºB / Matemáticas	Carlos / 1ºB / Educación física	Marta / 1ºB / Matemáticas	María / 1ºB / Emocrea	Marta / 1ºB / Matemáticas

Figura 42: Resultado caso colegio curso 1ºB

Además, se incorporó una restricción adicional para evitar que los docentes tuvieran huecos entre clases dentro de una misma jornada laboral. Esta condición, habitual en los centros educativos reales, incrementa notablemente la complejidad del problema al reducir aún más las combinaciones viables. A pesar de ello, el sistema fue capaz de generar un horario compacto, minimizando los periodos de inactividad del profesorado y optimizando su tiempo efectivo en el centro. Cabe señalar que esta restricción solo se incumplió en cuatro de los quince profesores, lo cual resulta aceptable teniendo en cuenta que, en la práctica, muchos centros contemplan horas de apoyo, coordinación o reuniones

que no han sido modeladas explícitamente en esta planificación. Esto refuerza la utilidad del sistema como herramienta práctica, eficiente y adaptable para la gestión horaria educativa.

Para poner a prueba la flexibilidad del sistema y explorar su capacidad de adaptación a diferentes escenarios, también se experimentó con condiciones poco habituales en la definición de las variables. En particular, se plantearon distintos escenarios de planificación de turnos para retenes contra incendios a lo largo de un periodo de 6 días, con dos turnos diarios: uno diurno (08:00–20:00, con salida a las 07:00 y regreso a las 21:00) y otro nocturno (20:00–08:00, con salida a las 19:00 y regreso a las 09:00).

En un primer caso, se definieron 22 retenes identificados mediante un código personalizado compuesto por tres dígitos precedidos por el prefijo RT (RT001, RT002, ..., RT022).

Variables identificadas

Clave	Valores
retenes	22
dias	6
franjas	2
horarios	["08:00-20:00", "20:00-08:00"]
lista_retenes	["RT001", "RT002", "RT003", "RT004", "RT005", "RT006", "RT007", "RT008", "RT009", "RT010", "RT011", "RT012", "RT013", "RT014", "RT015", "RT016", "RT017", "RT018", "RT019", "RT020", "RT021", "RT022"]

Figura 44: Ejemplo 1 definición de variables específico

En un segundo escenario, se trabajó con 23 retenes organizados en tres zonas geográficas (G1, G2 y G3), asignando a cada una hasta ocho retenes numerados de R1 a R8. Esto dio lugar a códigos como G1-R1, G2-R5 o G3-R8.

Variables identificadas

Clave	Valores
retenes	23
dias	6
franjas	2
horarios	["08:00–20:00", "20:00–08:00"]
lista_retenes	["G1-R1", "G1-R2", "G1-R3", "G1-R4", "G1-R5", "G1-R6", "G1-R7", "G1-R8", "G2-R1", "G2-R2", "G2-R3", "G2-R4", "G2-R5", "G2-R6", "G2-R7", "G2-R8", "G3-R1", "G3-R2", "G3-R3", "G3-R4", "G3-R5", "G3-R6", "G3-R7"]

Figura 45: Ejemplo 2 definición de variables específico

Finalmente, se incluyó también un conjunto de conductores, definidos mediante identificadores específicos del tipo C001, C002, ..., lo que refuerza aún más la capacidad del sistema para manejar formatos de codificación personalizados y ajustados a las necesidades operativas del entorno.

Variables identificadas

Clave	Valores
retenes	22
conductores	4
dias	6
franjas	2
horarios	["diurno", "nocturno"]
lista_retenes	["G1-R1", "G1-R2", "G1-R3", "G1-R4", "G1-R5", "G1-R6", "G1-R7", "G1-R8", "G2-R1", "G2-R2", "G2-R3", "G2-R4", "G2-R5", "G2-R6", "G2-R7", "G2-R8", "G3-R1", "G3-R2", "G3-R3", "G3-R4", "G3-R5", "G3-R6", "G3-R7"]
lista_conductores	["C001", "C002", "C003", "C004"]

Figura 46: Ejemplo 3 definición de variables específico

Estos ejemplos evidencian cómo la herramienta puede adaptarse no solo a restricciones funcionales complejas, sino también a estructuras internas particulares, como esquemas de nomenclatura o segmentación de recursos, sin comprometer la coherencia ni la eficiencia de las soluciones generadas.

8. Despliegue y explotación

Para llevar ResQPlan de un entorno local de desarrollo a un escenario de uso comercial, conviene planificar varios elementos clave: desde la infraestructura técnica, la gestión de licencias, la configuración de bases de datos y la seguridad del sistema.

1. Preparación del entorno

Se necesita un servidor o máquina en continuo funcionamiento, que puede ser tanto un equipo físico dedicado como una instancia en la nube (VPS) o un servicio de hosting. En este entorno debe instalarse Python (versión 3.8 o superior) y asegurarse de tener conexión a Internet para la descarga de paquetes y librerías.

2. Licencia de Gurobi

ResQPlan utiliza Gurobi como motor de optimización, por lo que es obligatorio tener una licencia de producción adecuada. Las licencias académicas, al estar restringidas exclusivamente a fines educativos, no son aptas para explotación comercial.

Una vez adquirida la licencia, se debe colocar el fichero gurobi.lic en la ruta indicada por la documentación de Gurobi y definir la variable de entorno. Para información adicional sobre la instalación de la licencia, consultar el apartado de Anexos.

3. Configurar la base de datos

El sistema utiliza MongoDB para almacenar proyectos y configuraciones. Existen dos formas principales de desplegar la base de datos:

- Instalación local o en servidor propio: ejecutar MongoDB en el mismo servidor que la aplicación.
- Servicio gestionado: utilizar MongoDB Atlas u otro proveedor.

En ambos casos, será necesario actualizar la URI de conexión en la variable MONGO_URI del fichero de configuración de la aplicación:

```
mongodb://usuario:clave@servidor:27017/resqplan
```

4. Arranque de la aplicación

Con todo configurado, la aplicación Flask se inicia mediante:

```
python main.py
```

Para garantizar su funcionamiento ininterrumpido, se recomienda gestionar el proceso con herramientas como systemd, supervisor o pm2, configurándolas para reiniciar la aplicación en caso de fallo. Asimismo, debe abrirse en el firewall el puerto escogido

5. Puesta en producción

Para facilitar el acceso a ResQPlan desde cualquier lugar y ofrecer una conexión segura, se recomienda completar los siguientes pasos:

1. Dominio y DNS: adquirir un nombre de dominio (por ejemplo, app.resqplan.com) y apuntar sus registros A o CNAME a la IP del servidor.
2. Proxy inverso y SSL: instalar un servidor web ligero (NGINX o Traefik) que actúe de proxy inverso frente a Flask, y configurar certificados TLS (Let's Encrypt o de pago) para ofrecer conexión HTTPS segura.

9. Relación con el Módulo de Inteligencia Artificial

Uno de los objetivos principales del desarrollo del sistema ResQPlan ha sido lograr que la herramienta pueda ser utilizada no solo por perfiles técnicos con conocimientos en programación o en optimización matemática, sino también por usuarios sin experiencia previa en este tipo de tecnologías. Para ello, se decidió incorporar un componente de Inteligencia Artificial que sirviera de puente entre el lenguaje natural y el lenguaje técnico necesario para definir un problema de planificación en un entorno como Gurobi.

Este componente basado en IA, desarrollado por mi compañera Cynthia Quintana Reyes, actúa en dos fases diferenciadas pero complementarias. En una primera fase, el usuario introduce en lenguaje natural su caso: días a planificar, franjas horarias, equipos o recursos disponibles, etc. A partir de esa descripción, la IA de OpenAI analiza el texto, detecta las variables clave, conjuntos implicados, índices (días, turnos, recursos) y parámetros numéricos, y genera un objeto JSON estructurado. Este JSON puede emplearse directamente para inicializar el modelo de optimización en Gurobi, evitando al usuario la codificación manual de las definiciones.

En la segunda fase, el usuario describe sus restricciones, por ejemplo, “cada trabajador debe tener al menos un día libre a la semana” o “no se pueden asignar más de dos turnos seguidos a un mismo conductor”. Estas se mandan a OpenAI que traduce automáticamente estas frases a expresiones en código Python compatibles con Gurobi. Estas restricciones se generan utilizando las variables extraídas previamente y se estructuran en forma de sentencias model.addConstr(...) que pueden integrarse directamente en el modelo.

Esta aproximación ha permitido que el sistema sea mucho más accesible. En lugar de requerir conocimientos técnicos para definir el modelo, el usuario simplemente debe describir su situación y sus reglas en lenguaje cotidiano. De esta forma, se elimina una de las principales barreras de entrada al uso de herramientas de optimización avanzadas, haciendo posible su utilización en contextos reales por parte de responsables de planificación o logística sin formación especializada.

10. Conclusión

El sistema desarrollado cumple con los objetivos planteados en cuanto a usabilidad, funcionalidad y aplicabilidad en entornos reales de planificación de turnos. La aplicación web proporciona una interfaz visual e intuitiva, diseñada para que cualquier usuario pueda configurar rápidamente los parámetros clave de un problema de planificación como, número de días, franjas horarias, recursos disponibles (equipos, vehículos, personal), sin necesidad de conocimientos previos en programación o técnicas de optimización matemática. Esta accesibilidad permite acercar herramientas avanzadas de planificación a perfiles no técnicos, mejorando la eficiencia operativa en sectores críticos.

La validación realizada junto al cuerpo de bomberos de La Palma confirmó que el sistema es capaz de respetar el 100 % de las restricciones operativas impuestas, generando soluciones óptimas en escenarios reales. Esta capacidad no solo mejora la calidad de la planificación, sino que reduce significativamente el riesgo de errores humanos y optimiza la eficiencia del proceso.

Además de resolver con éxito escenarios de emergencia y retenes, el sistema ha demostrado su versatilidad al aplicarse a otros ámbitos, como planificación de horarios escolares, turnos hospitalarios o asignación de recursos logísticos, sin necesidad de modificaciones en el núcleo de optimización. Esta capacidad de generalización convierte al prototipo en una solución flexible, capaz de adaptarse a diferentes sectores con requisitos específicos. La interfaz permite incorporar nuevas reglas o recursos de forma directa, ofreciendo un feedback inmediato sobre la viabilidad de cada ajuste. En conjunto, esta solución no solo agiliza el proceso de creación de turnos, sino que contribuye a mejorar el bienestar del personal, a reducir costes operativos mediante un uso más eficiente de los recursos y a aumentar la fiabilidad global de los servicios en múltiples contextos.

10.1. Resultados personales

A nivel personal, este proyecto ha supuesto una experiencia de aprendizaje profunda y enriquecedora. Por un lado, he podido consolidar y ampliar mis conocimientos en el desarrollo de aplicaciones web dinámicas, perfeccionando la gestión de interfaces reactivas, ventanas modales, validación de formularios y diseño de flujos de interacción intuitivos. Al mismo tiempo, mi inmersión en la optimización matemática, trabajando con Gurobi y Google OR-Tools, me ha brindado una comprensión práctica del modelado de restricciones, la interpretación de resultados y las estrategias para optimizar el rendimiento computacional.

La colaboración con la empresa me permitió trabajar sobre datos reales, en concreto el caso de los retenes, lo que me llevó a abordar un caso de uso real: identificar las necesidades operativas y traducirlas en requisitos técnicos precisos, ajustar el modelo a escenarios concretos y validar las soluciones propuestas.

10.2. Trabajo futuro

ResQPlan ofrece muchas posibilidades para evolucionar hacia una plataforma más completa y escalable. Por ejemplo, podríamos introducir un sistema de usuarios registrados con permisos y proyectos independientes, de modo que varios planificadores trabajen a la vez sin interferencias. También sería clave permitir la importación de restricciones y recursos desde ficheros estructurados (Excel, CSV…), para que los usuarios puedan cargar datos precargados y manejar grandes volúmenes de información, como en hospitales o colegios, sin depender únicamente del lenguaje natural. Con estas mejoras, ResQPlan ganaría en usabilidad, interoperabilidad y robustez, adaptándose mejor a entornos organizados y exigentes.

11. Referencias

[1] Valdés, S., & Prieto, S. S. (2024, 18 noviembre). Cadena SER. *Cadena SER*. https://cadenaser.com/castillayleon/2024/11/18/el-personal-del-hospital-de-salamanca-plantea-manifestarse-si-sacyl-no-hace-caso-a-sus-reivindicaciones-radio-salamanca/

[2] F.L.D. (2023, 1 febrero). La falta de personal en Bomberos dispara las horas extra. *Diario de Burgos*. https://www.diariodeburgos.es/Noticia/Z8EE0867E-CE22-F4B4-61741216881F4A7E/202302/La-falta-de-personal-en-Bomberos-dispara-las-horas-extra

[3] Razón, L. (2020, 30 julio). SPPLB denuncia que la ambulancia de bomberos no está operativa. *La Razón*. https://www.larazon.es/comunidad-valenciana/20200730/3dhtki4aarbppkiocdml2w6z4a.html

[4] Orellana, J. S. (2024, 24 octubre). Cadena SER. *Cadena SER*. https://cadenaser.com/andalucia/2024/10/24/antonio-banderas-miembro-del-comite-laboral-de-los-autobuses-de-la-emt-hay-una-mala-planificacion-lo-que-provoca-un-servicio-deficiente-ser-malaga/

[5] Shokouh, S. M. H., Mohammadi, K., & Yaghoubi, M. (2022). Optimization of Service Process in Emergency Department Using Discrete Event Simulation and Machine Learning Algorithm. *DOAJ (DOAJ: Directory Of Open Access Journals)*, *10*(1), e44. https://doi.org/10.22037/aaem.v10i1.1545

[6] *Acerca de las herramientas O*. (s. f.). Google For Developers. https://developers.google.com/optimization/introduction?hl=es-419

[7] *The Leader in Decision Intelligence Technology - Gurobi Optimization*. (2025, 15 mayo). Gurobi Optimization. https://www.gurobi.com/

[8] *PuLP*. (2025, 29 mayo). PyPI. https://pypi.org/project/PuLP/

[9] *Gestión del sector de emergencias y control de la seguridad*. (s. f.). https://www.aturnos.com/gestion-emergencias-seguridad

[10] *Gestión del sector de emergencias y control de la seguridad*. (s. f.). https://www.aturnos.com/gestion-emergencias-seguridad

[11] *Gestión de Emergencias EuroCoP SIGEM - EurocoP | Software de gestión policial*. (s. f.). Eurocop | Software de Gestión Policial. https://www.eurocop.com/sistemas-de-eurocop/gestion-de-emergencias-eurocop-sigem/

[12] Sesame HR. (2025, 22 mayo). *El software de Recursos Humanos que simplifica tus procesos | Sesame HR*. https://www.sesamehr.es/

[13] Butera, G. (2025, 31 marzo). *Getting Started with Gurobi Optimizer*. Gurobi Help Center. https://support.gurobi.com/hc/en-us/articles/14799677517585-Getting-Started-with-Gurobi-Optimizer

[14] Web, E. N. (2024, 13 diciembre). *Gurobi ayuda a transformar la cadena de suministro logística de Suzano*. EL NACIONAL. https://www.elnacional.com/empresas-productos/gurobi-ayuda-a-transformar-la-cadena-de-suministro-logistica-de-suzano/

[15] Belerofontech. (s. f.). Casos de éxito. Belerofontech. https://www.belerofontech.com/casos-de-exito/

[16] businesswire. (2022, 8 septiembre). Gurobi ayuda a las empresas de software a integrar correctamente la optimización en sus soluciones. *ChannelBiz*. https://www.channelbiz.es/press-release/gurobi-ayuda-a-las-empresas-de-software-a-integrar-correctamente-la-optimizacion-en-sus-soluciones/

Anexos

Guía de instalación de Gurobi Optimizer[13]:

Para utilizar Gurobi en un entorno de desarrollo, es necesario seguir una serie de pasos para instalar correctamente tanto el software como la licencia. En este proyecto se ha utilizado el lenguaje de programación Python y el entorno de desarrollo Pycharm. A continuación, se detallan los pasos que se han seguido para la instalación y configuración de Gurobi Optimizer, de forma que cualquier usuario pueda replicarlo fácilmente en su propio equipo.

1. Registro en página de Gurobi
   El primer paso consiste en crear una cuenta en el portal oficial de Gurobi. Desde esta plataforma se gestionan las licencias de usuario, se accede a la descarga del software y a la documentación oficial.

2. Solicitud de licencia académica
   Es necesario tener una licencia para poder usar Gurobi. Existen diferentes formas de obtener una licencia, que dependerá de las necesidades. En este caso, se ha usado una licencia académica gratuita, pero existen otros tipos de licencias como se detallan en el apartado Licencias Gurobi Optimizer.
   Una vez se obtengo dicha licencia esta será visible en el apartado de Licencias del Portal de usuario en el sitio web.
   Una vez aprobada, se genera una clave de activación que se utilizará más adelante

3. Descarga e instalación de software
   A continuación, se accede a la sección de descargas para obtener el instalador correspondiente al sistema operativo utilizado (Windows, macOS o Linux).

4. Activación de la licencia
   Con el software correctamente instalado, se debe activar la licencia desde la terminal ejecutando el siguiente comando:
   grbgetkey CLAVE-DE-LICENCIA
   Sustituyendo CLAVE-DE-LICENCIA por la clave recibida. Esto genera el archivo gurobi.lic en la carpeta correspondiente, habilitando así el uso del software.
   Para el uso de una cuenta académica, deberá estar conectado a su red académica para poder ejecutar este comando.

5. Instalación del paquete gurobipy en Python
   Para utilizar Gurobi desde Python, es necesario instalar el paquete oficial de la biblioteca. Esto puede hacerse desde la terminal del sistema o directamente desde PyCharm

6. Verificar
   Finalmente, se recomienda verificar que la instalación y configuración han sido exitosas, probando la ejecución de un pequeño script en Python que importe gurobipy y resuelva un modelo sencillo de optimización.