Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Prodef: unificación e integración de módulos

Prodef: unification and integration of modules

Alejandro Lugo Fumero

La Laguna, 25 de mayo de 2023

Dª. **Gara Miranda Valladares**, con N.I.F. 78.563.584-T profesora Titular de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

D. **Daniel del Castillo de la Rosa**, con N.I.F. 51.154.908-X graduado en Ingeniería Informática, como cotutor.

C E R T I F I C A N

Que la presente memoria titulada:

"*Prodef: unificación e integración de módulos*"

ha sido realizada bajo su dirección por D. **Alejandro Lugo Fumero**, con N.I.F. 79.095.533-M.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos, firman la presente en La Laguna a 25 de mayo de 2023

Agradecimientos

En primer lugar, me gustaría expresar mi más sincero agradecimiento a mi tutora Gara Miranda Valladares y a mi cotutor Daniel del Castillo de la Rosa. Su constante apoyo, orientación y respaldo fueron fundamentales para llevar a cabo este trabajo con éxito, sin ellos no hubiera sido posible.

También quiero agradecer a mis compañeros de clase y profesores por su constante colaboración y ayuda durante todo este proceso académico que es la universidad. Por último me gustaría darle las gracias a mis familiares y seres queridos que me han brindado su cálido apoyo durante todos estos años.

Quiero hacer una mención especial a mi mejor amigo Joseph Francisco Gabino Rodríguez y a mi hermano Sergio Lugo Fumero, ambos me han brindado su amistad y compañía durante todos estos años y siempre han creído en mí.

Licencia

Resumen

En la actualidad existe una gran cantidad de investigaciones respecto a los algoritmos de optimización bioinspirados, además han demostrado tener un gran potencial a la hora de resolver problemas de muy diversa índole. Sin embargo, estos algoritmos se caracterizan por ser métodos complejos, ya que son difíciles de implementar y de adaptar al problema en cuestión, esto hace que su uso fuera del ámbito de la investigación esté más limitado. Partiendo de esta premisa nace Prodef. Prodef es una solución web que se caracteriza por tener una interfaz gráfica, la cual permite el modelado de problemas y algoritmos sin tener que programar ni realizar complejas configuraciones. Esta interfaz está desarrollada utilizando la biblioteca de Blockly, esta biblioteca implementa una interfaz gráfica basada en bloques que permite la construcción de problemas y algoritmos como si de unir piezas de un puzzle se tratase. No obstante, esta solución web presentaba problemas en sí misma y en la interfaz gráfica que tanto la caracteriza, estos problemas conseguían que la solución web no cumpliese con los estándares de calidad que se esperaba.

En este trabajo se ha creado una solución web desde cero, además esta solución ha recibido un cambio de marca dando lugar a Othimi. Este cambio de marca añade un plus de calidad tanto en la solución como en este TFG. Para realizar el desarrollo de Othimi se ha realizado una investigación sobre las tecnologías web que mejor encajasen con la herramienta, entre ellas se puede destacar para el Back-End el framework de GraphQL Apolo y el framework de Back-End Koa. Para el Front-End se puede destacar el uso de Vite.js, React y Blockly. Además, se ha hecho un estudio completo de la librería Blockly para comprender su funcionamiento y poder realizar implementaciones fuera de lo que la librería ofrece, consiguiendo mejorar la interfaz gráfica que tanto caracteriza a Othimi y haciendo de Othimi una herramienta única en el mundo. Y por último, se ha rediseñado por completo la creación de instancias, ahora modelar una instancia es una tarea trivial y satisfactoria de cara al usuario.

Como resultado de este Trabajo de Fin de Grado no solo se ha obtenido una nueva solución web intuitiva y agradable al usuario, sino que se ha creado una base sólida para que el futuro desarrollo de la herramienta sea una tarea sencilla. Además, se hace uso de tecnologías actuales que utilizan las grandes empresas, lo que significa que Othimi es una herramienta moderna que algún día puede llegar a ser muy utilizada, consiguiendo su objetivo de llevar los algoritmos bio-inspirados fuera del ámbito de la investigación.

Palabras clave: Herramienta para optimización combinatoria, metaheurísticas, algoritmos bioinspirados, Othimi, Prodef, Blockly, Apolo, GraphQL, React, Koa.

Abstract

There is currently a great deal of research on bio-inspired optimization algorithms, and they are widely used in academia. However, these algorithms are characterized by being complex methods, since they are difficult to implement and to adapt to the problem in question, which makes their use outside the field of research very limited. Prodef was born from this premise. Prodef is a web solution that is characterized by a graphical interface, which allows the modeling of problems and algorithms without having to program or perform complex configurations. This interface is developed using the Blockly library, this library implements a block-based graphical interface that allows the construction of problems and algorithms as if they were puzzle pieces. However, this web solution presented problems in itself and in the graphical interface that characterizes it, these problems meant that the web solution did not meet the quality standards expected.

In this work a web solution has been created from scratch, in addition this solution has received a rebranding giving rise to Othimi. This rebranding adds a plus of quality both in the solution and in this TFG. For the development of Othimi we have done a research on the web technologies that best fit with the tool, among them we can highlight for the Back-End the GraphQL Apollo framework and the Back-End Koa framework. For the Front-End we can highlight the use of Vite.js, React and Blockly. In addition, a complete study of the Blockly library has been made to understand its operation and take this library to a second level, improving the graphical interface that characterizes Othimi and making Othimi a unique tool in the world. And finally, the instance creation has been completely redesigned, now modeling an instance is a trivial and pleasant task for the user.

As a result of this Final Degree Project, not only has a new intuitive and user-friendly web solution been obtained, but also a solid base has been created so that future development of the tool will be a simple task. Furthermore, it makes use of current technologies used by large companies, which means that Othimi is a modern tool that may one day become widely used, achieving its goal of taking bio-inspired algorithms outside the realm of research.

Keywords: Tool for combinatorial optimization, metaheuristics, bioinspired algorithms, Othimi, Prodef, Blockly, Apollo, GraphQL, React, Koa.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Si bien en la actualidad existe una gran cantidad de investigación respecto a la resolución de problemas mediante metaheurísticas, su complejidad intrínseca y la dificultad para implementarlas y adaptarlas al problema en cuestión dificultan su uso. Prodef es una herramienta surgida en este contexto y que nace con el objetivo de extender el uso de estas técnicas meta-heurísticas fuera del ámbito de la investigación convencional, tratando de acercar incluso el uso de estas técnicas a pequeñas y medianas empresas. Para ello, trata de establecer una clara separación entre la definición del problema y la determinación del método de optimización a aplicar, al mismo tiempo que proporciona una interfaz gráfica para el modelado del problema, lo cual supone un salto cualitativo en lo que respecta a sencillez de uso, dejando atrás complejas implementaciones a nivel de código o configuraciones a bajo nivel.

La actual versión de Prodef es fruto de varios Trabajos de Fin de Grado desarrollados por estudiantes del Grado en Ingeniería Informática de la Universidad de La Laguna:

- *Prodef: meta-modelado de problemas de optimización combinatoria* (2020)
  Andrés Calimero García Pérez [30].
- *Prodef-GUI: Interfaz gráfica para el modelado de problemas* (2021)
  Daniel González Expósito [31].
- *Prodef: Diseño, implementación y experimentación con nuevos resolutores* (2022)
  Miguel Angel Ordoñez Morales [33].
- *Prodef-Algorithm: Interfaz para el modelado de meta-heurísticas* (2022)
  Daniel del Castillo de la Rosa [28].
- *Prodef-SaaS: Despliegue y puesta en marcha de un servicio para la resolución de problemas de optimización* (2022)
  Ángel Tornero Hernández [34].
- *Prodef-Solution: Interfaz para la representación y visualización de soluciones* (2022)
  Yeixon Reinaldo Morales Gonzalez [32].

1.1. Antecedentes

Prodef es una herramienta que permite al usuario definir un problema de forma abstracta para que luego, un componente llamado resolutor, sea capaz de recibir esta definición y así computar una solución para el problema. Inicialmente, se definió un metamodelo que permitía especificar, en términos abstractos, las características esenciales de un problema de optimización (variables, restricciones y objetivos, así como la instancia a resolver). A partir de este modelo, o especificación genérica de un problema, se ofrecía una interfaz para traducción directa a diferentes *frameworks* de optimización (como por ejemplo, `jMetal` [29] y `METCO` [27]). Este modelado del problema se realizaba directamente a través de una especificación en lenguaje `ProdefLang` (el lenguaje de dominio específico que usa Prodef) recogida a través de un fichero JSON, pero luego se dotó a Prodef de una interfaz gráfica para poder definir y gestionar de una forma mucho más sencilla diferentes problemas así como sus correspondientes instancias. De esta forma, en lugar de trabajar directamente con archivos .json para la especificación de los problemas en lenguaje `ProdefLang`, los usuarios pueden utilizar una interfaz gráfica basada en bloques.

En una segunda fase, se incluyó en Prodef una interfaz para el modelado de algoritmos. De esta forma, se permite que los usuarios puedan definir sus propios algoritmos de optimización meta-heurística de una forma sencilla. Para ello, ha sido necesario crear un nuevo resolutor. Este resolutor es fundamentalmente distinto a aquellos que ya existían, ya que permite especificar (y configurar) qué algoritmo se quiere usar para resolver el problema en cuestión. Para implementar este nuevo resolutor, ha sido necesario desarrollar un compilador de `ProdefLang` para Rust, ya que este es el lenguaje elegido para la implementación del nuevo resolutor. Para permitir el modelado de algoritmos, este resolutor hace uso de una abstracción basada en componentes genéricos de representación y definición de metaheurísticas. Uno de los mayores retos del diseño realizado es que los algoritmos que se definen son independientes de cualquier problema, aunque durante el proceso de resolución sí que será necesario elegir ciertos parámetros y componentes específicos que son dependientes del problema.

Finalmente, se ha diseñado una infraestructura para desplegar la herramienta Prodef como un servicio web empleando el modelo SaaS (software como servicio). Para ello, se ha realizado una configuración en Docker-compose capaz de poner en marcha todos los contenedores de Prodef simultáneamente. Además, se ha contratado un proveedor de servicios de nube y un plan para la automatización y el despliegue del servicio.

1.2. Objetivos

Tal y como se ha mencionado, en Prodef existen resolutores predefinidos, pero también se pueden diseñar resolutores `ad-hoc` a través de una interfaz de modelado de algoritmos. En cualquier caso, el número y tipo de componentes disponibles actualmente para diseñar algoritmos es aún limitado. Por otro lado, la herramienta puede ser muy útil y versátil, pero aún requiere de un importante trabajo en lo que se refiere a la interfaz web, la usabilidad, la disponibilidad de ejemplos y recursos de ayuda, el diseño de pruebas y la validación con múltiples problemas y/o algoritmos de ejemplo.

Por lo tanto, con el objetivo de que Prodef pueda erigirse como una herramienta única que no solo sirva para resolver problemas mediante técnicas estándar y predefinidas, sino que vaya más allá y permita a los usuarios comparar estrategias y/o seleccionar aquellas

más prometedoras o con un mejor rendimiento es necesario mejorar algunas cuestiones de diseño interno. Al mismo tiempo, y con la finalidad de que Prodef pueda convertirse en una herramienta de referencia a nivel educativo, sería necesario un diseño gráfico apropiado, así como ejemplos y documentación suficiente.

En definitiva, Prodef podría ser una plataforma idónea para aquellos usuarios que quisieran dar sus primeros pasos en el campo de la optimización con meta-heurísticas, pero antes requiere de un importante trabajo en la unificación y la integración de sus diferentes módulos e interfaces. Este será el objetivo principal de este Trabajo de Fin de Grado, el cual se pretende alcanzar mediante el desarrollo de las actividades siguientes:

Objetivo	**Descripción**
Análisis de Prodef en su estado actual.	Abordar el estudio de la herramienta Prodef, comprendiendo su estructura modular y su funcionamiento interno. Será fundamental entender los procesos de despliegue, así como identificar las fortalezas y debilidades actuales de la herramienta.
Unificación del diseño gráfico y mejora de la usabilidad.	Analizar las interfaces actuales y mejorar su diseño y/o comportamiento, prestando especial atención a criterios de usabilidad.
Validación de la herramienta.	Analizar diferentes problemas de optimización, realizar las correspondientes implementaciones en Prodef, detectar limitaciones e incorporar nuevos componentes que permitan un modelado adecuado.
Elaboración de tutoriales y otros recursos de ayuda.	Elaborar los materiales oportunos para que los futuros usuarios puedan dar, de forma sencilla y autónoma, sus primeros pasos en el campo de la optimización con meta-heurísticas.
Documentación.	Elaborar el material de soporte para el uso de la herramienta, así como la memoria del TFG y demás documentación requerida durante la realización de la asignatura de TFG.

Capítulo 2

Prodef

Prodef es una aplicación desarrollada por diferentes Trabajos de Fin de Grado, esta aplicación se encuentra alojada en la organización de GitHub llamada ULL-prodef. Prodef ha ido evolucionando y cambiando a lo largo de los años a través de diferentes TFGs y por ende hay partes que se han mejorado, partes que se han dejado de utilizar porque se han desarrollado soluciones que encajan mejor y partes que se han modificado en su plenitud.

2.1. Estado actual de la herramienta.

Como se ha mencionado anteriormente, Prodef es una herramienta fruto del trabajo de diferentes TFGs, de forma resumida se comentará el trabajo realizado en cada uno de ellos.

- *Prodef: meta-modelado de problemas de optimización combinatoria* [30]: En este TFG se desarrolló una herramienta que permite la definición de problemas de optimización combinatoria a un nivel de descripción abstracto y su posterior resolución usando librerías y *frameworks* externos, como `jMetal`.
- *Prodef-GUI: Interfaz gráfica para el modelado de problemas* [31]: En este trabajo se desarrolló la solución web que es Prodef, tanto el Front-End como el Back-End. En la Figura 2.1 se puede ver el Front-End.
- *Prodef: Diseño, implementación y experimentación con nuevos resolutores* [33]: En este TFG fue desarrollado un resolutor basado en la librería `METCO` de C++.
- *Prodef-Algorithm: Interfaz para el modelado de meta-heurísticas* [28]: En este trabajo se desarrolló el resolutor de Rust y el modelado de algoritmos.
- *Prodef-SaaS: Despliegue y puesta en marcha de un servicio para la resolución de problemas de optimización* [34]: En este TFG se realizó el despliegue de Prodef.
- *Prodef-Solution: Interfaz para la representación y visualización de soluciones* [32]: En este último trabajo se creó una interfaz gráfica para visualizar los resultados de una forma más representativa.

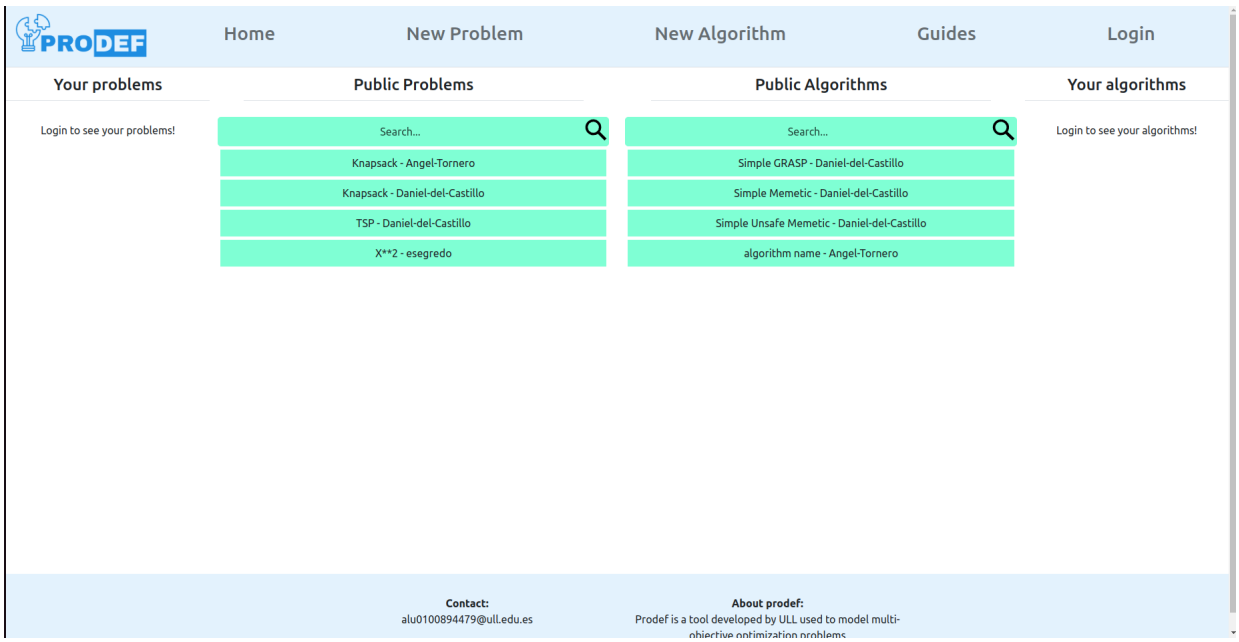

Figura 2.1: Antiguo Front-End de Prodef

2.2. Puesta en marcha de Prodef

Para poner a funcionar Prodef de forma local se ha seguido el tutorial de instalación que se encuentra en la organización ULL-prodef, hay que tener en cuenta que el objetivo final de ese tutorial es poner en funcionamiento el resolutor de Java, `JMetal`, y que el objetivo de este apartado es desplegar el Front-End, el Back-End y el resolutor que se está utilizando actualmente, el resolutor de Rust. Es por esto mencionado anteriormente que no se puede seguir el tutorial al pie de la letra y se han tenido que modificar algunos pasos. Los pasos a realizar son los siguientes:

1. Clonar el meta repositorio [23]: `https://github.com/ULL-prodef/prodef`
2. Instalar el meta repositorio, añadir en cada subrepositorio un archivo `.npmrc` con un token clásico con el permiso *read:packages*, instalar, compilar y enlazar estos subrepositorios.
3. Instalar Rust [19].
4. Realizar un `npm run start` tanto en el repositorio prodef-gui-frontend, como en el repositorio prodef-gui-backend, el Back-End se encargará mediante un `childProcess` de levantar el resolutor de Rust.

2.3. Análisis del Front-End

El análisis del Front-End ha sido dividido en tres secciones, en la primera sección se analizará la parte del Front-End que está centrada en el usuario (accesibilidad, usabilidad, redimensionamiento, etc.), en la segunda se analizará Blockly [4] y en la tercera parte se analizará el código y las herramientas utilizadas.

2.3.1. Análisis de usuario

En lo que respecta a la parte que implica al usuario de la aplicación, se encuentran diferentes problemas que hacen que la página no resulte atractiva al usuario y que pueden provocar frustración en este. Algunos de los problemas encontrados son los siguientes:

- La página presenta una paleta de colores poco atractiva y además su uso es poco intuitivo, suponiendo una barrera para los usuarios primerizos en esta.
- Solo hay un sistema de autentificación el cual es utilizando GitHub, esto obliga al usuario a tener sí o sí una cuenta de GitHub para poder utilizar la página.
- El redimensionamiento de la página no es correcto para ciertos tamaños. Además, cuando el dispositivo es muy pequeño se quedan espacios en blanco muy grandes, lo cual confunde al usuario y lo obliga a hacer *scroll*.
- A la hora de cargar las instancias se puede hacer utilizando archivos .csv, el problema es que no se cumple el formato .csv y tampoco se explica en ninguna parte el formato que se admite.
- Hay ciertos componentes que no funcionan de forma correcta, un ejemplo sería el componente de React [15] que se encarga de cambiar los datos que aparecen por pantalla según se haya escogido una instancia u otra, no cambia la información que aparece por pantalla, aun así la instancia sí cambia, es solo un fallo visual.
- En el momento de introducir datos no hay un límite, por lo tanto, un usuario puede introducir números tan grandes que cuando el problema llegue al resolutor de Rust genere *overflow* en la ejecución. Además, si se envía un problema que requiera mucho tiempo de ejecución, el resolutor se bloquea, es decir, no se puede resolver más de un problema a la vez.
- El usuario casi no recibe *feedback* por parte de la aplicación y cuando lo recibe es un *feedback* muy general, es difícil que sea capaz de saber que ha hecho mal.
- La parte de Blockly, una de las partes más importantes, ya que es la parte que permite abstraer al usuario de la implementación de problemas y algoritmos, presenta muchos problemas que son desglosados en la siguiente sección.

2.3.2. Análisis de Blockly

Blockly presenta diferentes problemas, entre algunos de ellos se encuentran los siguientes:

- Los bloques cambian de color según se van anidando, el paradigma de Blockly presenta un color para cada categoría de bloque, si los bloques van cambiando según se aniden dejan de representar la categoría de la que forman parte, haciendo que los usuarios no tengan claro el fin de cada bloque.
- Se utiliza el sistema de variables que trae por defecto Blockly, este sistema deja mucho que desear, ya que las variables se crean con botones en las diferentes categorías de bloques, pero estas variables son globales, no pertenecen a una categoría específica a diferencia de los bloques que sí pertenecen a categorías únicas. Además, para introducir el nombre de las variables aparece en la parte superior una ventana de alerta que presenta un gran problema de accesibilidad, ya que no funcionan bien con los programas que puede utilizar una persona ciega.
- Hay bloques que pertenecen a diferentes categorías, un bloque debería tener un solo fin y pertenece a una única categoría.
- No tiene sentido que una variable pertenezca a dos bloques que no tienen nada que ver, por ejemplo, una variable llamada `i` puede ser el nombre de una columna de una tabla y a su vez ser una variable local en un sumatorio.
- Hay bloques que no encajan bien con otros.
- No se puede actualizar la librería a la última versión, ya que se genera un error que deja la pantalla en blanco.

2.3.3. Análisis del código

Respecto al código del Front-End se utilizan librerías como:

- React [15].
- Axios [2] para las peticiones.
- Bootstrap [5] como *framework* de css.
- Redux [17] como librería para almacenar información siguiendo el patrón FLUX.
- Blockly [4] para el modelado de problemas y algoritmos.
- Papaparse [14] para los archivos .csv.
- Showdown [21] para convertir Markdown en código HTML.

El principal problema que presenta el Front-End a nivel de código es el caos que existe en el repositorio, los archivos tienen nombres genéricos, no existe un orden, hay archivos de cientos de líneas sin comentarios, entre otros problemas.

2.3.4. Mejoras y soluciones del Front-End

Los problemas del Front-End se pueden dividir en dos partes, la parte de Blockly y la parte del usuario, donde entraría el apartado visual y las tecnologías del Front-End.

Blockly como herramienta de modelado de problemas y algoritmos

Aprovechando que Prodef todavía es una aplicación pequeña, es momento de plantearse si Blockly es una buena herramienta para la aplicación o si es momento de cambiarla. Actualmente, Blockly se encarga de proporcionar la interfaz que permite el modelado del problema y del algoritmo de una forma que el usuario no tiene que saber programar ni tiene que entrar en configuraciones complejas, pero toda la lógica detrás de los bloques es de Prodef, por ende, cambiar de herramienta no supondría un gran problema.

A la hora de buscar un sustituto hay que tener en cuenta cuál es el papel de Blockly, su papel es el de ofrecer a los usuarios diferentes elementos los cuales van encajando para finalmente crear problemas o algoritmos, como si fuese una especie de puzzle donde solo hay que colocar cada pieza en su lugar. El planteamiento anterior es la premisa que se utilizará para buscar un sustituto.

El problema de encontrar una librería similar a Blockly es que Blockly se ha terminado convirtiendo casi en un estándar. Todas las librerías similares a Blockly parten de un *fork* de Blockly, como Scratch [20], o son librerías que implementan Blockly como parte de ellas, un ejemplo sería Node-Red [12]. Node-Red es una librería dedicada a escribir poco código para la programación de aplicaciones basadas en eventos, esta librería implementa Blockly con la librería node-red-contrib-blockly [13].

Después de realizar una búsqueda y análisis de posibles sustitutos, estas son las opciones finales:

- Rete [18], *framework* modular para la programación visual que permite crear un editor basado en nodos.
- Scratch Blocks, *fork* de Blockly que proporciona bloques con lógica.
- Dedicar este TFG a desarrollar una librería que haga el papel de Blockly.
- Seguir utilizando Blockly.

Finalmente, la última opción resultó ser la más viable, seguir utilizando Blockly, se ha tomado esta decisión por varios motivos:

- Rete no era una buena opción, ya que se basa en nodos y por ende no proporciona al usuario la sensación de completar un puzzle para construir su problema o algoritmo.
- Scratch Blocks a diferencia de Rete sí cumple con esa labor, pero no permite que los programadores desarrollen sus propios bloques.
- La idea de crear una librería supondría dedicar demasiado tiempo a este apartado, cuando no es el objetivo de este trabajo. Además, la librería tendría que tener una calidad similar o mayor a la de Blockly, algo imposible de conseguir en un solo TFG. También cabe mencionar que en un futuro habría que dedicar recursos y medios en mantener y desarrollar esa librería.

- Blockly actualmente tiene diferentes problemas que tienen que ser solucionados, pero estos problemas no son nada que no se pueda resolver con trabajo y dedicación en este apartado. Además, se consiguió arreglar el fallo que no permitía actualizar a Blockly a la última versión, este fallo era que en versiones anteriores para ejecutar código JS se hacía con `Blockly.JavaScript` y en las versiones nuevas el código se ejecuta con `Blockly.javascriptGenerator`.

Interacción del usuario

La parte visual del Front-End tiene demasiados fallos como para modificar el código que actualmente existe, es por esto que se ha decidido rehacer el Front-End desde cero. Aprovechando esta decisión se rediseñará por completo y se añadirán nuevas secciones y herramientas a la página, además de rehacer la parte de Blockly para pulir este apartado.

La herramienta que se utilizará para desarrollar el Front-End será Vite [24], ya que permite un desarrollo rápido y completo, por otro lado, se seguirá en la línea de utilizar TypeScript junto con React-Bootstrap [16], pero esta vez se utilizará en SWC [22], el cual es un compilador de JavaScrip escrito en Rust que se utiliza para compilar código JavaScript a un formato más ligero y eficiente para su uso en la web. Como es obvio, se hará uso de Blockly para el modelado de problemas y algoritmos, Redux [17] para el *localstorage*, Vitest [26] para el *testing* y por último, como en el Back-end se hará uso del *framework* de graphQL [7], Apollo [1], se utilizará Apollo-client.

2.4. Análisis del Back-End

En cuanto al Back-End existen dos problemas principales, el primero es que no se cumple una arquitectura de tres capas, ni ningún otro tipo de arquitectura, la capa de aplicación se mezcla con la capa de datos. Al tratarse todavía de un Back-End muy pequeño no representa un gran problema a simple vista, pero en el momento en que el Back-End comience a crecer y desarrollarse, este dejará de ser escalable y será muy complicado realizar un cambio en la tecnología del *framework* del Back-End o en la base de datos, ya que está todo mezclado. El segundo problema es que no utiliza ningún sistema de *token* para realizar la comunicación Front-End/Back-End, esto puede derivar en problemas de seguridad, permitiendo acceder a zonas que no debería a usuarios que no se han registrado o iniciado sesión.

2.4.1. Mejoras y soluciones del Back-End

Respecto al Back-End se implementará una arquitectura en 3 capas que permita separar Prodef en la capa de presentación, en la capa de aplicación y en la capa de datos. Aprovechando esto y que la mayoría de modelos se tendrán que rehacer debido a todos los cambios que va a sufrir el Front-End, también se rehará el Back-End desde cero. Para el Back-End se hará uso del *framework* de graphQL, Apollo, este *framework* a su vez hará uso de Koa [10], se utilizará Jest [8] para el *testing* y finalmente se hará uso de la base de datos no relacional MongoDB [11]. También cabe mencionar que se utilizarán librerías como Bcrypt [3] para el cifrado de contraseñas y JWT [9] para la comunicación segura entre Front-End y Back-End.

Capítulo 3

Front-End

Es importante mencionar que durante el desarrollo de este TFG Prodef recibió un cambio de marca y pasó a llamarse Othimi, este cambio implicó un nombre nuevo, una nueva gama de colores y un nuevo concepto. Este cambio añade un plus de profesionalidad tanto al TFG como a la propia herramienta desarrollada. Los colores y el nuevo logo de la marca se pueden ver en la Figura 3.1 y en la Figura 3.2 se puede apreciar el motivo.

Figura 3.1: Marca Othimi

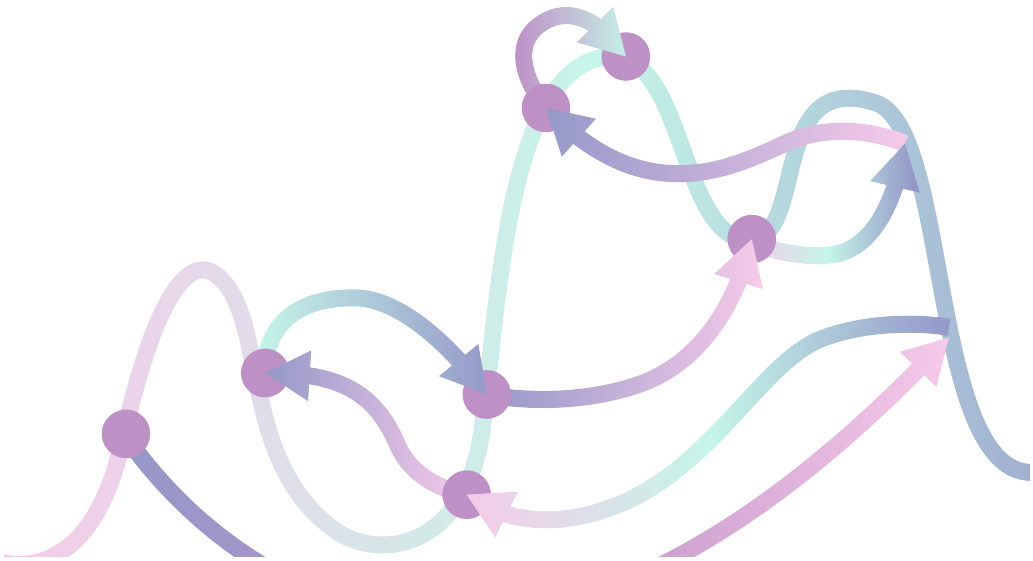

Figura 3.2: Motivo Othimi

Como se mencionó anteriormente, el Front-End ha sido desarrollado sobre el *framework* de desarrollo Vite [24], se ha escogido este *framework* frente a la herramienta utilizada en el antiguo Front-End de Prodef, create-react-app, ya que Vite tiene un rendimiento mayor durante el desarrollo, no requiere prácticamente configuración y en su versión 3.0, versión que se está utilizando, es compatible con SWC [22]. SWC es un compilador de JS/TS desarrollado en Rust por el equipo de Deno, SWC es compatible con una amplia variedad de características de JavaScript y TypeScript, incluyendo clases, decoradores, funciones asincrónicas, y mucho más. Es capaz de generar código altamente optimizado y se integra fácilmente con otros paquetes y herramientas de JavaScript, además create-react-app utiliza Webpack, solución que es más lenta que Vite. Es importante mencionar que create-react-app actualmente ya no recibe soporte por parte de los creadores de React y está en desuso.

Para evitar algunos de los problemas que tiene el antiguo Front-End de Prodef, el desarrollo se ha hecho buscando el mayor encapsulamiento posible, escalabilidad y desarrollando de la forma más ordenada posible, tanto a nivel de código como a nivel de directorios, de esta forma cuando en un futuro sea necesario desarrollar otras partes del Front-End la persona encargada lo tendrá mucho más fácil.

3.1. Componentes

Los componentes desarrollados en el Front-End se pueden clasificar en los siguientes grupos:

- **Componentes generales**: se trata de componentes que pueden aparecer en más de un componente diferente, como por ejemplo las “migas de pan”, o son de carácter general, como el *navbar*, el *footer*, la página de error cuando se accede a una ruta que no existe, etc.
- **Autentificación**: en este grupo de componentes entran todos los relacionados con esta tarea.
- **Problemas**: este grupo de componentes está formado por el propio componente encargado de los problemas, el que muestra la información de un problema, el encargado de realizar la paginación...
- **Instancias**: este grupo es igual que el de los problemas, pero para las instancias.

3.1.1. Componentes generales

En esta sección solo se hará mención a los componentes generales más interesantes.

Bread Crumb

Este componente se caracteriza por ser dinámico, en cada ruta de la web recibirá dos *arrays* de *strings* diferentes, en el primer *array* van las rutas y en el segundo van los nombres de las rutas, estos nombres serán los que vea el usuario y las rutas será a donde se le redirigirá cuando haga clic en alguno de los nombres. En la Figura 3.3 se pueden ver los parámetros que recibe el componente y en la Figura 3.4 se puede apreciar una imagen del componente en la página.

```
<BreadCrumb routes={[AppRoutes.home]} names={["Home", "My work zone"]} />
```

Figura 3.3: Declaración del componente Bread Crumb

Figura 3.4: Componente Bread Crumb

Paginador

El objetivo de este componente es mostrar de forma dinámica las diferentes páginas a las cuales puede acceder el usuario, se usa para administrar la paginación de problemas e instancias. Este componente se caracteriza por recibir el número de la página donde se encuentra, el total de páginas y un `React.Dispatch` para cambiar de página según el usuario haga clic en un número u otro, en la Figura 3.5 se puede ver una instancia de este componente. El componente como máximo mostrará 5 páginas y como mínimo 1 página, en la Figura 3.6 se puede apreciar como se vería el componente con únicamente 3 páginas; si la página donde se encuentra el usuario está a más de 3 páginas de la primera página o de la última añadirá puntos suspensivos detrás de la primera página y delante de la última respectivamente, esto último mencionado se puede ver en la Figura 3.7, en la Figura 3.8 y en la Figura 3.9.

```
<Paginator page={page} pages={info.pages} setPage={setPage} />
```

Figura 3.5: Declaración del componente Paginator

Figura 3.6: Componente Paginator con 3 páginas

Figura 3.7: Componente Paginator a menos 3 páginas de la primera página

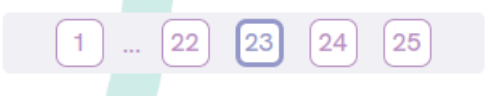

Figura 3.8: Componente Paginator a menos 3 páginas de la última página

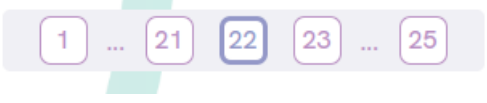

Figura 3.9: Componente Paginator a más de 3 páginas del inicio y del final

Reportador de errores

Este componente es una pieza fundamental en los problemas e instancias, ya que informa al usuario de forma dinámica de los diferentes errores o campos que no haya rellenado, además cuenta con un botón que permite eliminar el componente en cuestión, permitiendo que el usuario tenga su zona de trabajo limpia. Cabe destacar que este componente recibe un *array* de *strings* con los diferentes errores y que es responsabilidad de cada parte de Othimi generar los errores. En la Figura 3.10 se puede apreciar una instancia de este componente y en las Figuras 3.11 y 3.12 se puede ver un reporte de un problema y de una instancia respectivamente.

```
<ErrorReporter
  errors={errors}
  setErrors={setErrors}
/>
```

Figura 3.10: Declaración del componente Reportador de Errores

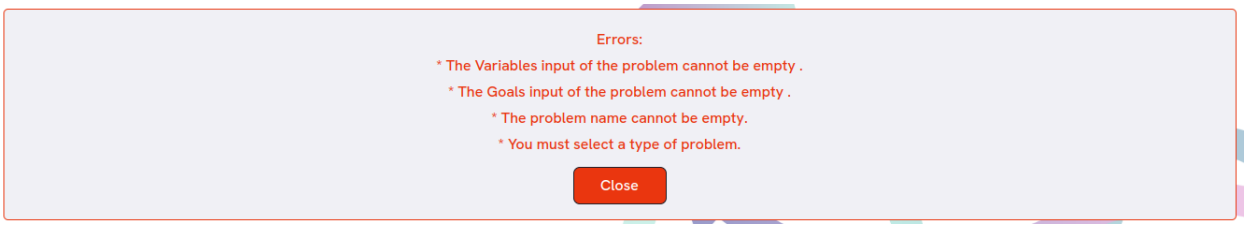

Figura 3.11: Componente Reportador de Errores en un problema básico

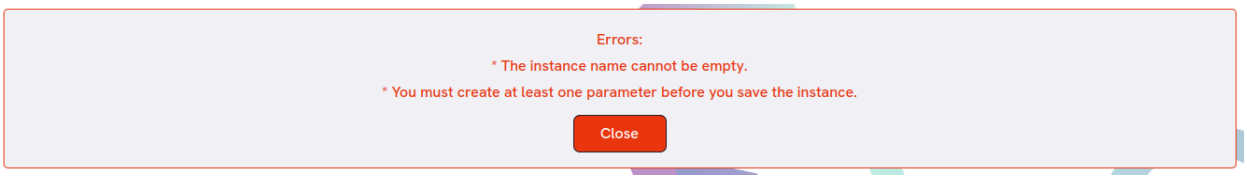

Figura 3.12: Componente Reportador de Errores en una instancia básica

3.1.2. Autentificación

Respecto a la autentificación se podrá realizar de dos formas, la primera es utilizando la cuenta de GitHub y la segunda es realizando una autentificación tradicional utilizando un correo electrónico.

Cabe destacar que es obligatorio iniciar sesión para poder acceder a la zona de trabajo, ya que esta zona se encuentra protegida, de forma que si un usuario que no está autentificado intenta acceder a esta ruta será redirigido a la ruta de autentificación:

```
1 import { Navigate, Outlet } from "react-router-dom";
2
3 type dataType = {
4   isAllowed : boolean,
5   children? : JSX.Element
6 }
```

```
export default function ProtectedRoute(data : dataType) {
  if (!data.isAllowed) {
    return <Navigate to={"/auth"}/>
  }

  return data.children ? data.children : <Outlet/>
}
```

Además, la ruta de autentificación también está protegida, si un usuario intenta acceder a esta estando autentificado será redirigido al *home*, ya que no tiene sentido que un usuario autentificado se pueda autentificar.

GitHub Auth

La autentificación mediante GitHub es la más sencilla, ya que en el Front-End el usuario solo tiene que darle permisos a la `OAuth` de Prodef y es el Back-End el encargado de manejar la información y generar el token, en capítulos posteriores se entrará en más detalle sobre el Back-End. En la Figura 3.13 se puede apreciar la *OAuth app* de Prodef en GitHub.

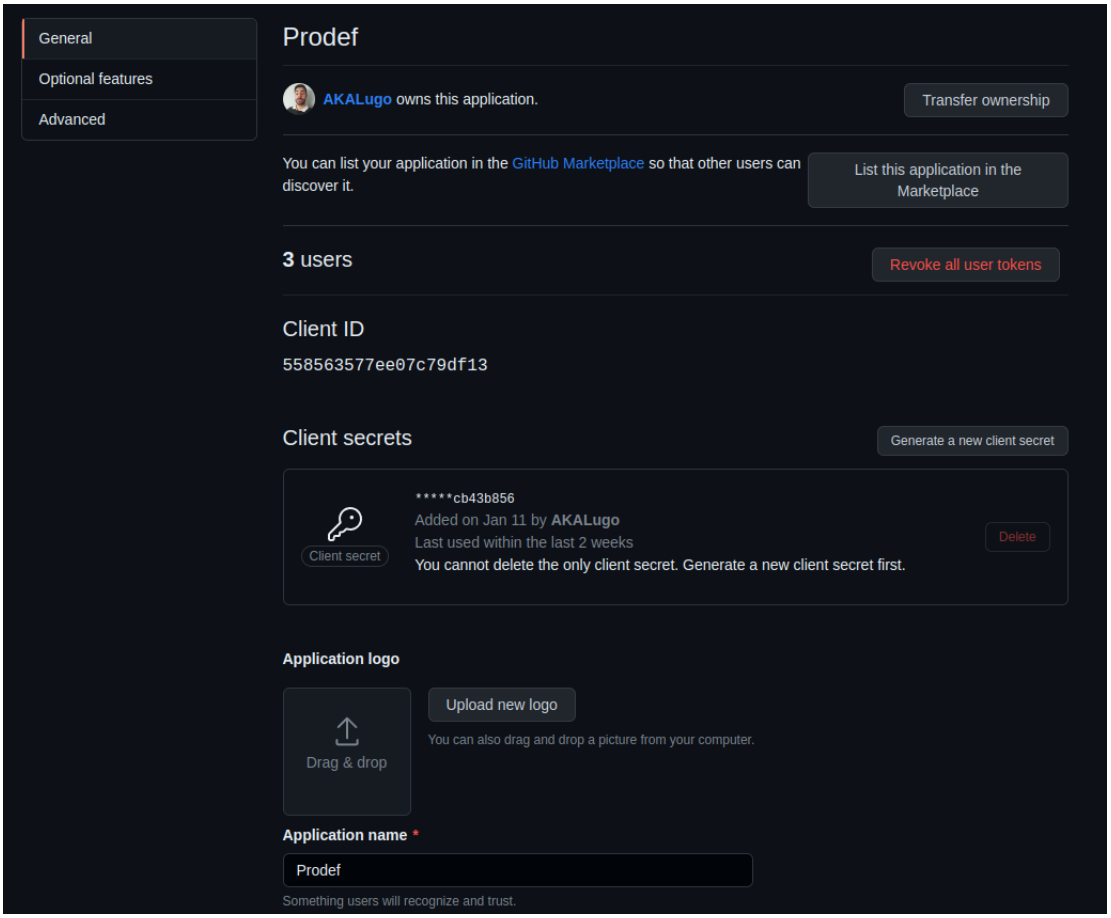

Figura 3.13: OAuth Prodef

E-Mail Auth

En lo que respecta a la autentificación utilizando un correo electrónico se puede distinguir *login* de *register*, para la implementación de ambos se ha hecho uso de expresiones regulares para comprobar los diferentes datos que se introducen en los campos y se ha utilizado la librería Formik [6] y Yup [25] para la creación del formulario, la recogida de datos, la comprobación de la información en tiempo real y para el envío de esta información al Back-End.

En primer lugar, en el *register* el usuario tendrá que introducir un nombre de usuario único, un correo electrónico único, tendrá que poner 2 veces la contraseña y tendrá que aceptar los términos y condiciones. En el primer campo contraseña se comprobará que la contraseña cumpla unos requisitos mínimos mediante el uso de expresiones regulares y en el otro campo contraseña se verificarán que las contraseñas coincidan. En la Figura 3.14 se puede ver como se le pide al usuario que introduzca los datos en los diferentes campos, cabe destacar que esta información solo aparece si se lanza el evento `onBlur` en el campo es cuestión.

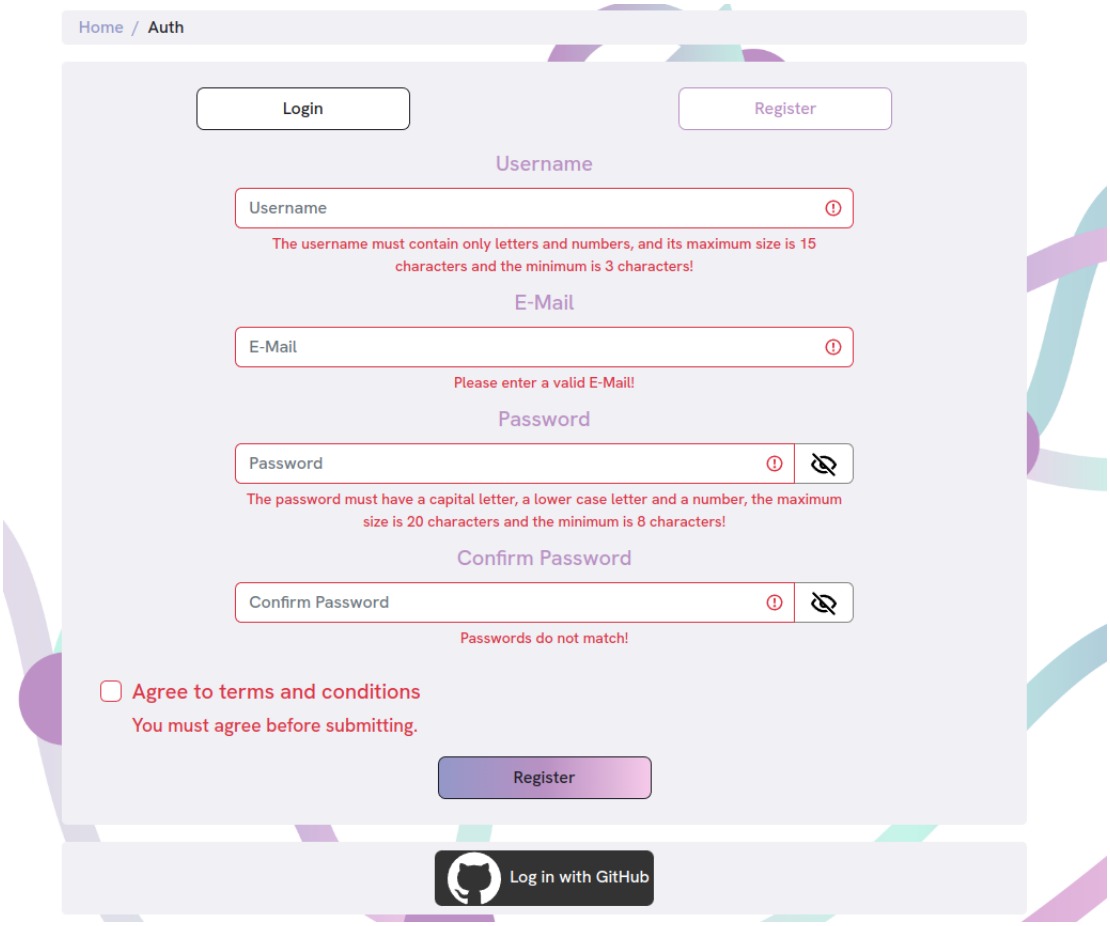

Figura 3.14: Fallo en los campos del registro de Othimi

En segundo lugar, en el *login* el usuario introducirá el correo electrónico o el nombre de usuario y una contraseña. En el propio Front-End se comprobará si el primer campo tiene una @, en el caso de que la tenga se tratará como un correo electrónico y en el caso de que no, se tratará como un nombre de usuario. En la Figura 3.15 se puede apreciar como el formulario indica qué debe introducir el usuario en cada campo, de manera similar como se hace en el *register*.

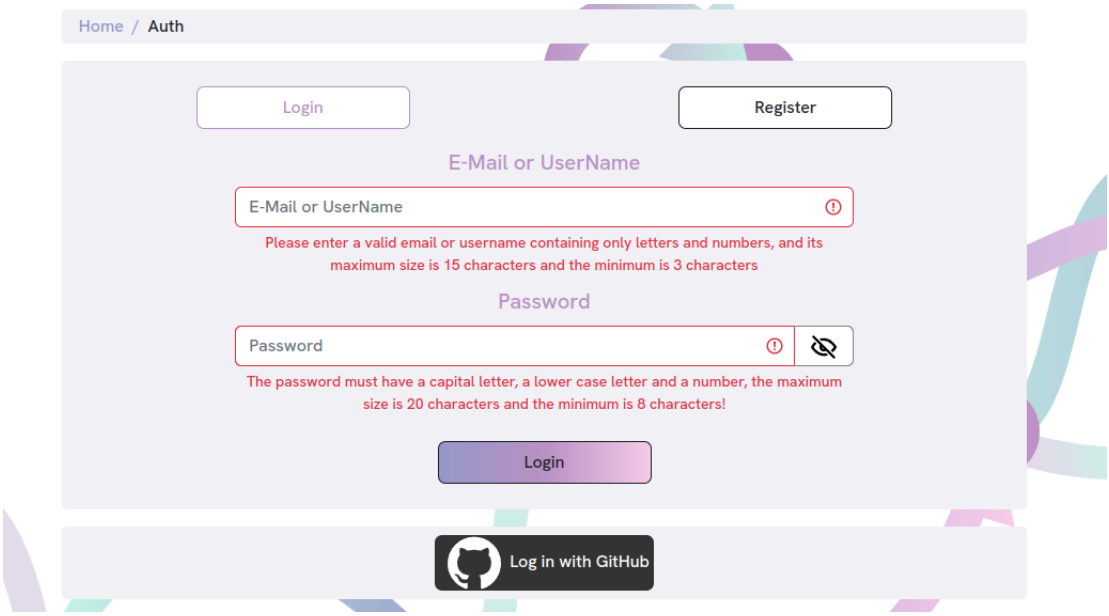

Figura 3.15: Fallo en los campos del login de Othimi

3.1.3. Problemas

Dentro del componente `WorkZone` se encuentra el componente `ProblemRoot`, este componente se encarga de mostrar los diferentes filtros, el buscador, aplicar estos filtros, listar los diferentes problemas y de manejar los diferentes modos de un problema (*copy, edit, view y delete*). En esta sección se describirán los diferentes componentes que forman `ProblemRoot`.

Buscador, filtros y listados

El componente que se encarga del buscador, aplicar los filtros y listar los diferentes problemas según cumplan los requisitos de los filtros se llama `ProblemList`. En este componente se realizan todas las peticiones al Back-End apoyándose en el cliente de Apolo para facilitar las consultas. El cliente de Apolo envuelve la App principal de React y con una configuración sencilla automáticamente se encarga colocar el *token* de JWT en cada petición, la configuración es la siguiente:

```
ReactDOM.createRoot(document.getElementById("root") as HTMLElement).render(
  <Provider store={store}>
    <PersistGate loading={null} persistor={persistor}>
      <Environment />
    </PersistGate>
  </Provider>
)


function Environment() {
  const httpLink = createHttpLink({
    uri: "http://localhost:5000/graphql",
  });

  const user = useAppSelector((state) => state.userState.userData);


```

```
  const authLink = setContext((_, { headers }) => {
    return {
      headers: {
        ...headers,
        authorization: user.token ? "Bearer ${user.token}" : "",
      }
    }
  });

  const client = new ApolloClient({
    link: authLink.concat(httpLink),
    cache: new InMemoryCache(),
  });

  return (
    <ApolloProvider client={client}>
      <App />
    </ApolloProvider>
  )
}
```

Aprovechando que Apolo es un *framework* que ofrece un sistema de cache para hacer más eficientes las consultas se ha activado esta opción, aunque cabe destacar que esta opción solo es útil si el contenido que se pide al Back-End no va a cambiar, algo que sí pasa con los problemas, instancias y algoritmos. Ya que estos cambian de forma dinámica, un ejemplo sería si un usuario borra un problema, al realizar de nuevo una *query* si la cache está activada mostrará todos los problemas, incluso el que está borrado, dando lugar a un comportamiento equivoco. Para evitar compartamientos indeseados se desactivará el sistema de cache en cada query, de forma que el Front-End queda configura para hacer uso de este sistema y si no se quiere hacer uso se desactiva, un ejemplo de una query sería el siguiente:

```
const [getProblems, get] = useLazyQuery(GET_PROBLEMS, { errorPolicy: "all",
      fetchPolicy: "no-cache" });
```

Además Apolo obliga a que el ApolloClient se encuentre en el cuerpo de un componente de React, es por este motivo que se ha creado el componente `Environment`. Gracias al ApolloClient se pueden realizar peticiones *querys* y *mutations* al Back-End de forma sencilla, las *querys* son el equivalente a un GET en una API REST (consultas de datos) y las *mutations* corresponden con un POST, DELETE o PATH, operaciones que realicen cambios en los datos. La *query* inicial que trae toda la información del Backe-End es la siguiente:

```
export const GET_PROBLEMS = gql`
query Query($owner: String!, $page: Int!, $name: String, $family: String, $tableNumbers:
    Int, $paramsNumber: Int) {
  problems(owner: $owner, page: $page, name: $name, family: $family, tableNumbers:
    $tableNumbers, paramsNumber: $paramsNumber) {
    info {
      prev
      next
      pages
      count
    }
    problem {
      filesCount
```

```
      parametersCount
      family
      name
      id
    }
    params
    tables
    familys
  }
}
‘
```

En esta *query* se pueden distinguir 5 tipos de datos principales, `info`, `problem`, `params`, `tables` y `familys`; `info` contiene la información general que permite la paginación, en `prev` se encuentra la siguiente página, en `next` la anterior, en `pages` el número total de páginas y en `count` la cantidad de elementos que coinciden con la búsqueda. En `problem` se encuentra la información general del problema, información que se le mostrará al usuario utilizando el componente `ProblemElement`, componente del cual se hablará en la siguiente sección. Por último, en `params`, `tables` y `familys` se encuentran la cantidad de parámetros que tiene cada problema, la cantidad de tablas que tiene cada problema y los tipos de familias de cada problema respectivamente. Esta información se usa en los filtros, de esta forma el usuario solo utilizará filtros que le aporten búsquedas concretas, no filtros genéricos que no sirvan para nada. En la Figura 3.16 se pueden apreciar el buscador y los diferentes filtros sin aplicar ninguno, además también aparece el botón que permite crear un problema desde cero.

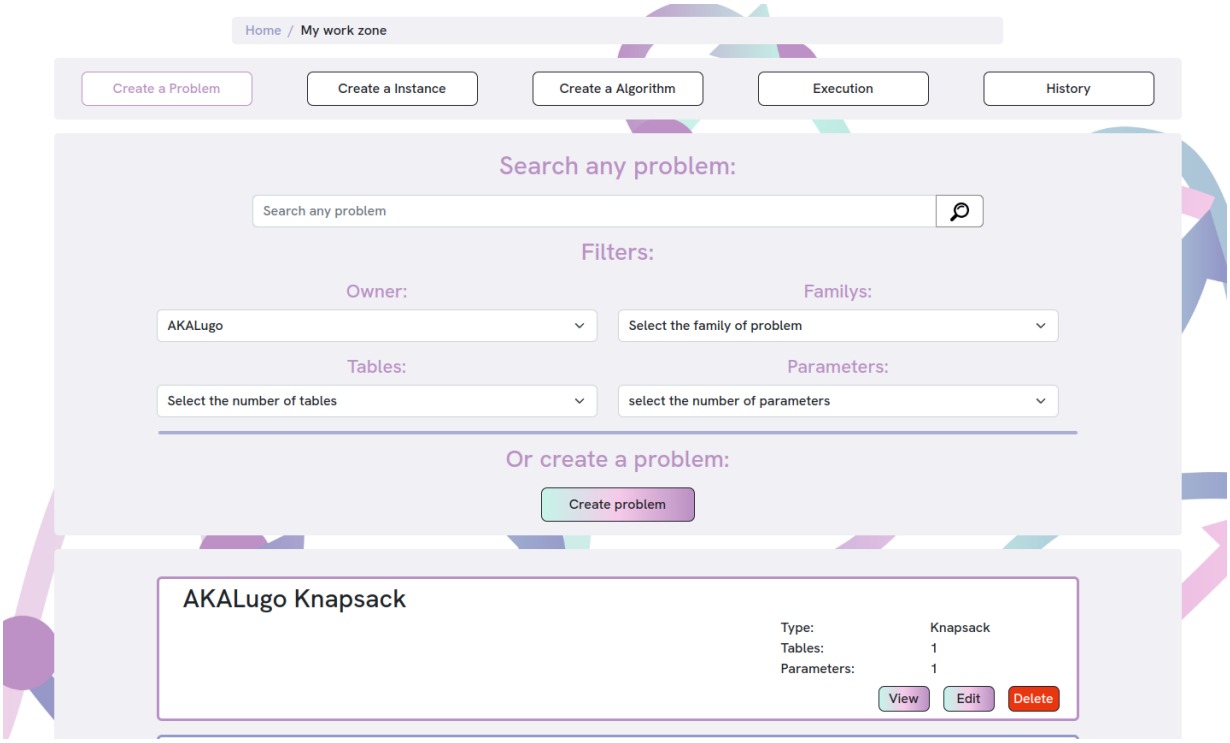

Figura 3.16: Filtros y problemas

Othimi es una web compleja que puede resultar confusa para un nuevo usuario, es por esto que en el filtro `Owner` se puede cambiar entre los problemas del usuario y los problemas de ejemplo que ofrece Othimi, esto también aplica para la sección de instancias

y algoritmos aunque es importante destacar que en este TFG el apartado de algoritmos no ha sido desarrollado, por ende no se han creado algoritmos de ejemplos ni se ha trabajo esa parte.

En la Ffigura 3.17 se puede ver como al cambiar el filtro `Owner` a los problemas de ejemplo aparecen nuevos problemas, los cuales tienen opciones diferentes a los problemas de un usuario, los problemas de un usuario tiene las opciones de *view, edit y delete* y los problemas de ejemplo tienen las opciones de *view y copy*, en la siguiente sección se habla de estas opciones.

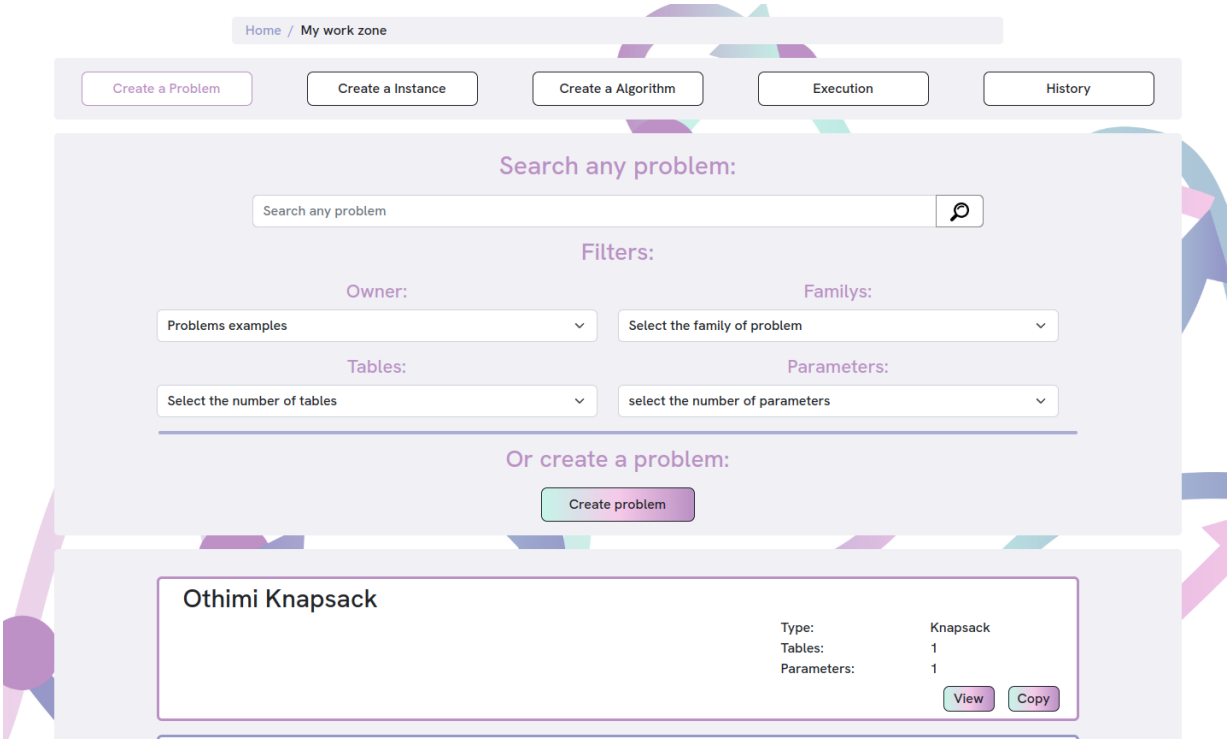

Figura 3.17: Filtros y problemas por defecto

Información de un problema

`ProblemList` se apoya en el componente `ProblemElement` para mostrar la información de los problemas y manejar las diferentes opciones de cada problema. Este componente es sencillo, ya que solo muestra el nombre del problema, la familia, la cantidad de tablas y la cantidad de parámetros. En la Figura 3.18 se pueden ver 2 problemas creados por el usuario, se sabe que esos problemas son del usuario por las opciones que ofrecen.

Dentro de las opciones de los problemas del usuario existen *view, edit y delete*, con *view* se muestra el problema, pero no se puede editar nada, un ejemplo sería el problema de la Figura 3.19, como se puede ver el problema no tiene ninguna *toolbox* con los diferentes bloques y tanto el *dropdown* de las familias como el *input* del nombre como el botón de guardar problema están desactivados.

Con la opción *edit* se carga el problema como en *view*, pero con la diferencia de que es editable y que cuando el usuario guarde el problema se realizará una *mutation* que realice un PATH en el Back-End. La query para traer la información específica de un problema sería la siguiente:

```
export const GET_PROBLEM = gql`
query Query($idProblem: String!) {
```

```
  problem(id: $idProblem) {
    name
    family
    blocklyJSON
  }
}
‘
```

Figura 3.18: Componente ProblemElement

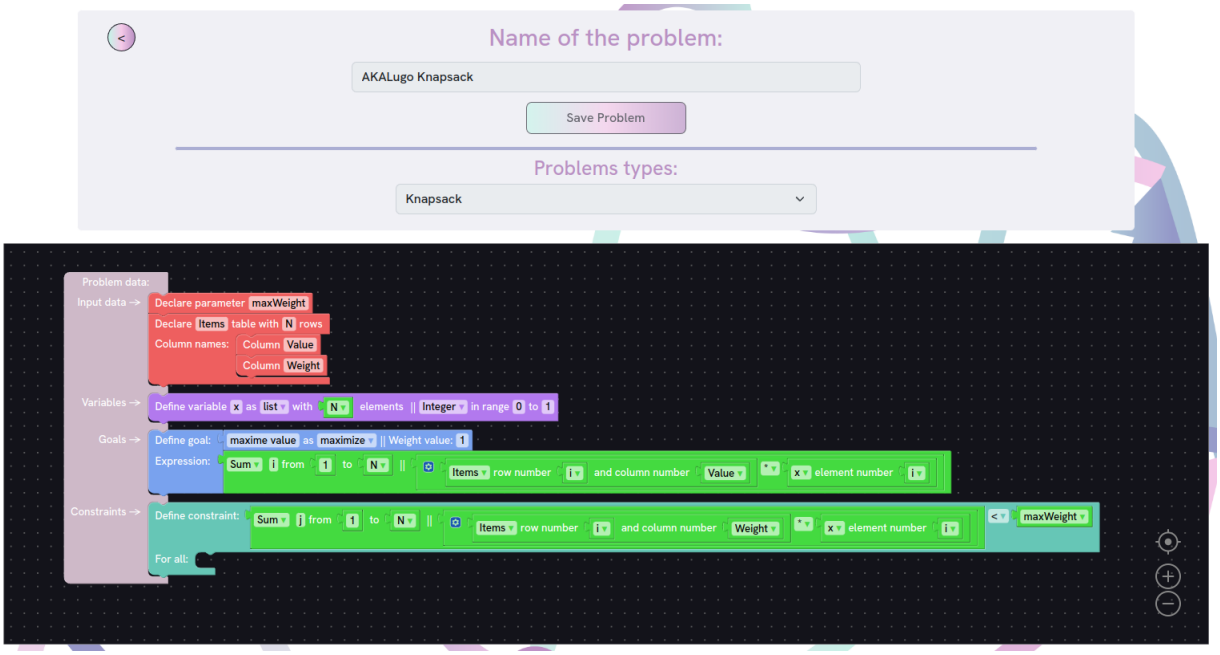

Figura 3.19: Modo View en un problema

La *mutation* que realiza el PATH en caso de guardar un problema en modo *edit* es la siguiente:

```
export const PATH_PROBLEM = gql‘
mutation Mutation($idProblem: String!, $input: ProblemInput!) {
  pathProblem(id: $idProblem, input: $input)
}
‘
```

Y por último la opción de *delete*, opción que borra un problema de la base de datos, su *mutation* es:

```
export const DELETE_PROBLEM = gql`
mutation Mutation($idProblem: String!) {
  deleteProblem(id: $idProblem)
}
`
```

Anteriormente, se ha comentado que los problemas de ejemplo tienen opciones distintas a los problemas de un usuario, sus opciones son *view y copy*; el modo *view* de un problema de ejemplo tiene el mismo comportamiento que el *view* de un problema de un usuario, pero el modo *copy* funciona similar al modo edit a diferencia de que en el título del problema la palabra Othimi es cambiada por el nombre del usuario y que cuando el usuario guarde el problema no se realizará una *mutation* que modifique el problema de ejemplo (PATH), sino se realizará una *mutation* que guardará el problema como un nuevo problema del usuario en cuestión (POST). La *mutation* que realiza el POST es:

```
export const POST_PROBLEM = gql`
mutation Mutation($input: ProblemInput!) {
  createProblem(input: $input) {
    id
  }
}
`
```

Problemas

El componente `Problem` es uno de los más importantes, ya que permite el modelado de un problema sin tener que saber programar ni entrar en configuraciones complejas, en la sección de Blockly se entra mucho más en detalle sobre este componente. En la Figura 3.20 se puede apreciar un problema vacío.

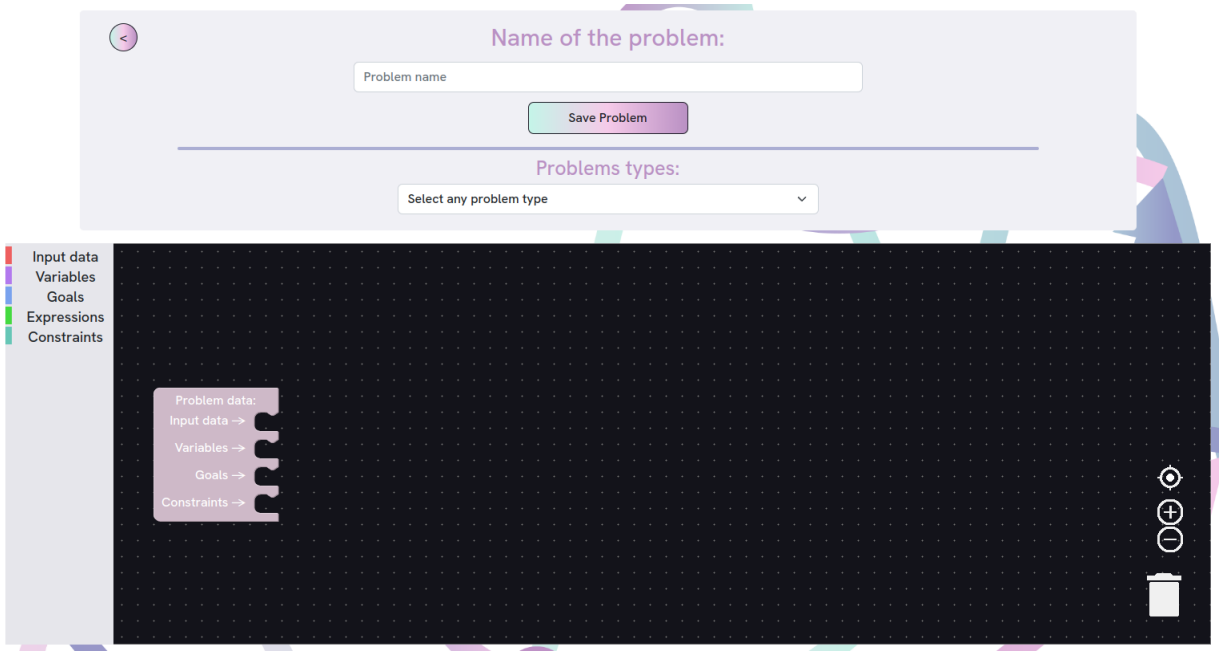

Figura 3.20: Problema vacío

Un problema consta de un nombre, de un tipo de familia y de un modelado en Blockly. Los tipos de familia de los que dispone Othimi son:

- *Bin packing*
- *Knapsack*
- *Location*
- *Portfolio optimisation*
- *Scheduling*
- *Shortest path*
- *Steiner*
- *Travelling salesman problem*
- *Cutting/packing family*
- *Vehicle routing*
- *I am not sure...*

Cabe destacar el tipo *I am not sure...* es un tipo de problema que se utilizaría cuando el usuario está empezando a modelar su problema o no tiene claro en donde encasillar este, con esta opción se pretende que el usuario se sienta cómodo utilizando la herramienta, ya que le da cierta libertad.

3.1.4. Instancias

En lo que respecta a las instancias el comportamiento es el mismo que el de los problemas, con la diferencia de que una instancia no tiene familia, simplemente tiene cantidad de tablas y cantidad de parámetros, de resto es igual, por ende se analizará directamente el componente `Instance`, sin entrar en detalles del componente `InstanceList` o `InstanceElement`. En la Figura 3.21 se puede ver el componente `InstanceList` e implícitamente el componente `InstanceElement`.

El componente `Instance` es el componente que permite crear los parámetros y tablas que luego se utilizarán en un problema. Una instancia tiene como mínimo un parámetro y puede no tener tablas, además puede tener más de una tabla y/o más de un parámetro. Como se aprecia en la Figura 3.22 una instancia tiene 3 zonas básicas:

- Administración de parámetros.
- Administración de tablas.
- Cargar tablas mediante archivos .csv.

En la zona de administración de parámetros se pueden crear y borrar parámetros, además cuando se cree un parámetro se le podrá cambiar el nombre y el valor, el nombre cambia en tiempo de ejecución en el propio *dropdown*. En la Figura 3.23 se puede apreciar como el nombre del parámetro ha cambiado a coche tanto en el *input Parameter name* como en el *dropdown*, además como hay un parámetro seleccionado, el botón *Delete Parameter* se ha activado, dando la opción de eliminar el parámetro seleccionado al usuario.

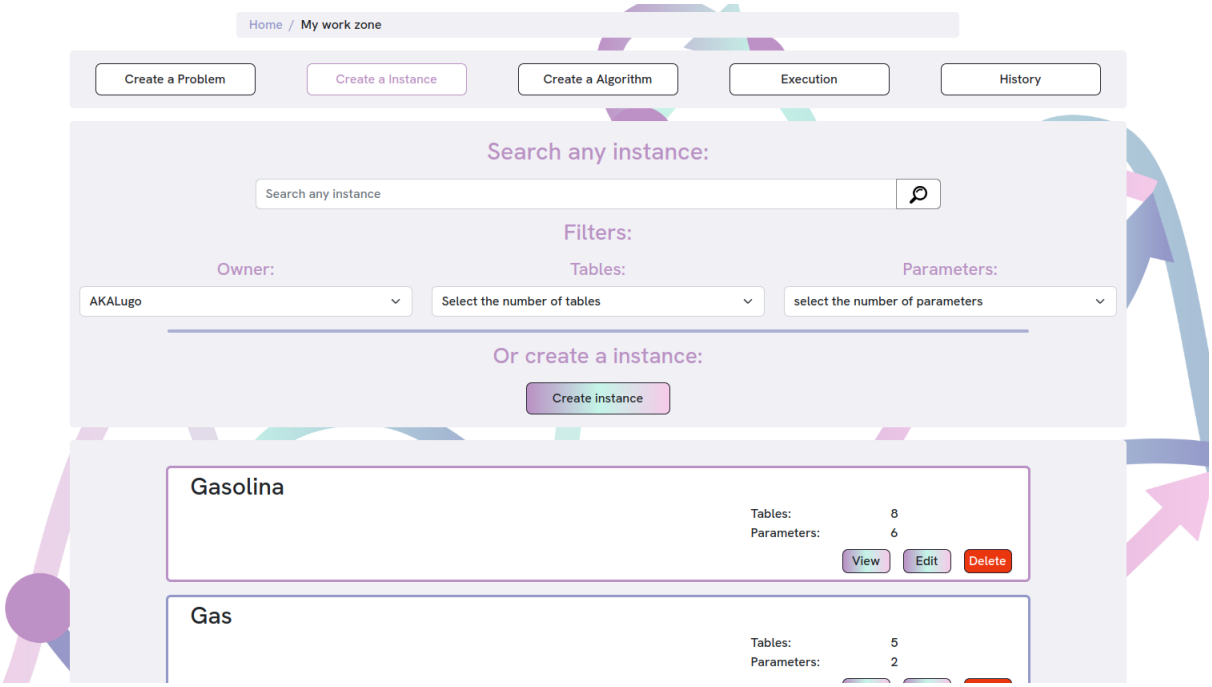

Figura 3.21: Filtros e instancias por defecto

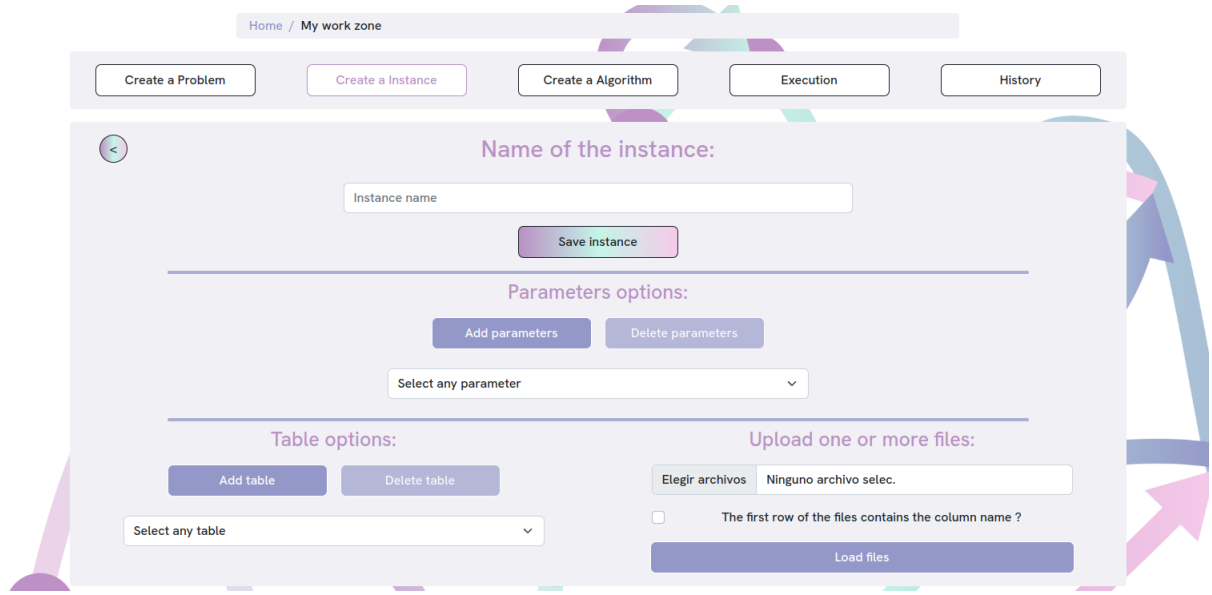

Figura 3.22: Instancia vacía

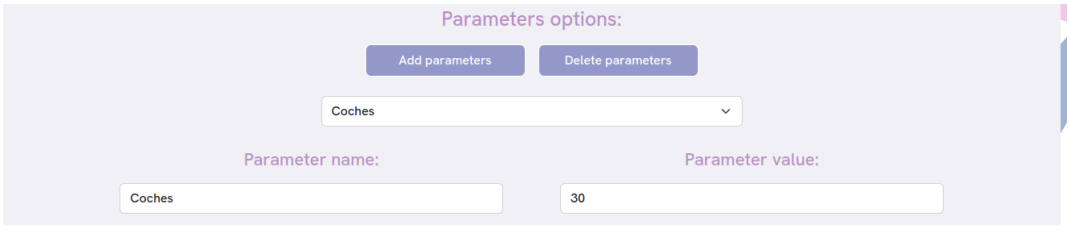

Figura 3.23: Administración de parámetros

La zona de administración de tablas sigue el mismo esquema que la zona de administración de parámetros, a diferencia de que una tabla tiene nombre, filas, columnas y los valores para cada fila y columna. En la Figura 3.24 se puede ver como la tabla

ha cambiado dinámicamente su número de filas y columnas y además se le ha añadido algunos valores a algunas filas y columnas y se ha cambiado el nombre de las columnas.

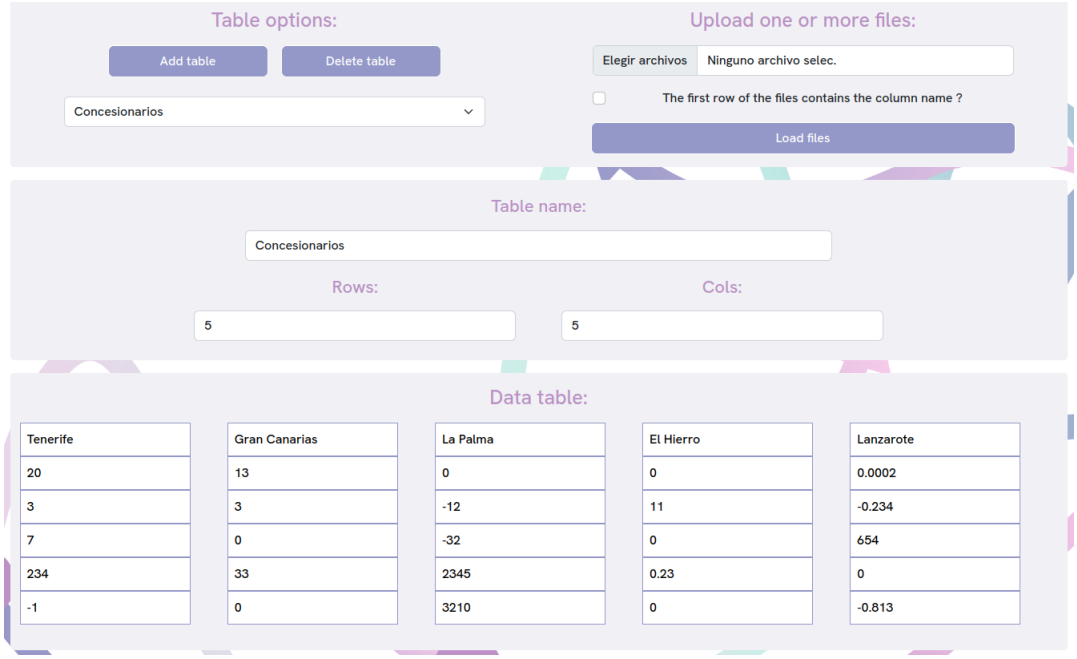

Figura 3.24: Administración de tablas

Por último mencionar la zona que se encarga de cargar tablas desde archivos .csv, esta zona notificará al usuario en caso de que haya ocurrido algún error con alguna fila. Como el formato .csv admite que el nombre de las columnas aparezca al principio del archivo o no, se le pregunta al usuario si esta condición se cumple, ya que si es el caso hay que tratar la primera fila de forma diferente. En la Figura 3.25 se puede apreciar los mensajes de error que salen al cargar un fichero .csv cuya primera fila es el nombre de las columnas y no marcar en la instancia que la primera fila contiene el nombre de las columnas, además el fichero también contiene alguna fila con más y con menos elementos que las otras.

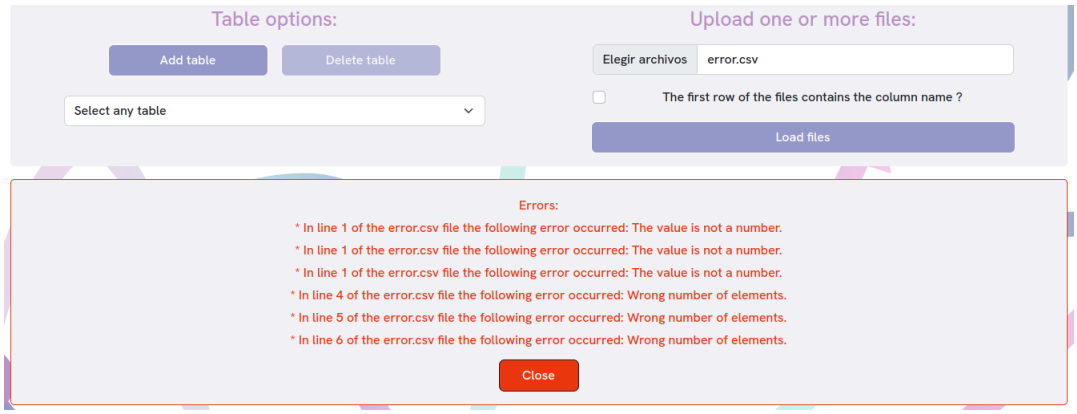

Figura 3.25: Errores al cargar un archivo .csv

3.2. Blockly

Las diferentes actualizaciones de Blockly [4], su curva de aprendizaje y todos los errores que había en el propio código de la interfaz de Blockly anterior, ha hecho que esta parte se rehaga desde cero, como se comentó en el capítulo dos. Partiendo de esta premisa, el trabajo en este apartado se ha dedicado a la mejora de Blockly y no a la creación de nuevos bloques, aunque es importante mencionar que algunos bloques han sido modificados de tal manera que se han añadido nuevos bloques de forma implícita. Esto ha sido así, ya que de nada sirve crear nuevos bloques si los que existen no funcionan bien o no están bien hechos.

3.2.1. Nueva versión de Blockly

Puesto que el apartado de Blockly se va a desarrollar en la última versión estable disponible, versión 9.3.3, hay que tener en cuenta los cambios que ha recibido la librería respecto a la versión que utilizaba el antiguo Prodef, versión 7.2.4. Los cambios relevantes son dos, el primero de ellos es que se deja de dar soporte al formato XML, el único formato que va a recibir actualizaciones es el JSON, es importante mencionar que la antigua versión de Prodef mezclaba los dos formatos, pero utilizaba principalmente el formato XML. El segundo de ellos es que para ejecutar código JS en los bloques ya no se utiliza el método `JavaScript`, sino que se utiliza el método `javascriptGenerator`, si se actualiza la versión de Blockly, pero no se cambia el método que ejecuta el código JS, la web meterá un pantallazo blanco.

3.2.2. Mutadores

Blockly es una librería muy grande, por lo tanto, crear un proyecto básico y sencillo no es muy complicado, esta librería aumenta de forma exponencial su dificultad a medida que se busca un resultado más profesional. Es aquí donde entran los mutadores, los mutadores son funciones que extienden las funcionalidades de un bloque, cada bloque solo puede tener un mutador y estos a su vez se dividen en varias partes:

- `saveExtraState`, *hook* de serialización del formato JSON encargado de crear el estado inicial del bloque.
- `loadExtraState`, *hook* de serialización del formato JSON encargado de cargar el estado del bloque y aplicarlo.
- `updateShape_`, función auxiliar encargada de modificar el bloque según los diferentes estados.
- `decompose`, función que expande el bloque principal en subbloques más pequeños que se pueden mover, agregar y borrar.
- `compose`, función que interpreta la configuración de los subbloques y modifica el bloque principal.
- `opt_helperFn`, función auxiliar que ejecuta el bloque nada más cargar el mutador.

Los mutadores combinados con las diferentes clases estáticas desarrolladas en este TFG son lo que ha permitido un desarrollo más profesional de la interfaz de creación de

problemas. En las siguientes secciones se entrará más en detalle sobre los mutadores y estas clases estáticas.

3.2.3. ExpressionBlock

El antiguo bloque que representaba una expresión, ya sea una suma, una diferencia, una potencia, una división o una exponenciación; tenía un problema y es que un bloque solo era capaz de representar una operación y, por lo tanto, se tendría que anidar tantos bloques como operaciones se quisieran realizar, esto daba lugar a un bloque gigante que resulta confuso a la vista de quien está creando un problema. En la Figura 3.26 se puede apreciar el bloque descrito anteriormente.

Figura 3.26: Anidación de expressionBlocks en el antiguo Prodef

Este bloque ha sido modificado para que un único bloque sea capaz de representar tantas operaciones como el usuario quiera, para ello se ha hecho uso de un mutador. En la Figura 3.27 se puede ver el nuevo expressionBlock.

Figura 3.27: ExpressionBlocks en Othimi

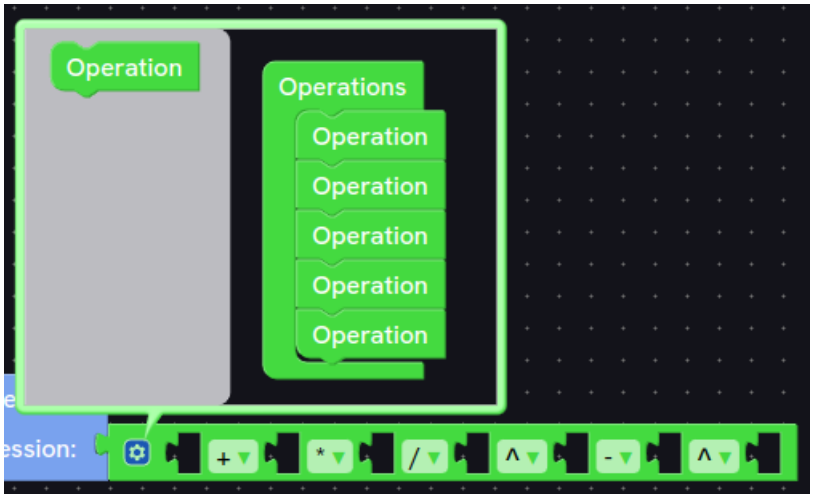

Figura 3.28: ExpressionBlocks en Othimi desplegado

Este bloque priorizaría las operaciones como se hace en el álgebra y si se coloca otro expressionBlock dentro de un *input* este tendría prioridad de operación, simulando como si estuviera en un paréntesis. En la Figura 3.28 se puede apreciar el pequeño espacio de trabajo que se desplegaría cuando un usuario hace clic en el recuadro azul, en esta zona puede añadir o eliminar las operaciones que se desee, respetando que como mínimo siempre habrá una operación.

En el Apéndice A.1 se encuentra el código del mutador de este bloque, en primer lugar están las funciones `saveExtraState` que devuelve el valor de las variables del bloque, las cuales se utilizan para llevar la cuenta de los *inputs* y para saber si el usuario ha añadido un subbloque o si lo ha eliminado y `loadExtraState` que se encarga de guardar estas variables en el bloque y de ejecutar la función auxiliar `updateShape_`.

Blockly es una librería que no explica a los desarrolladores como funciona por debajo, esto es un problema, ya que limita lo que se puede hacer con la librería, es por esto que en este TFG se ha tenido que comprender muy profundamente como funciona la librería para poder llevarla un nivel más allá, aunque en principio no esté pensada para esto.

Gracias al estudio de la librería se ha averiguado que cada vez que un bloque se mueve lo hace por copia, esto implica que el bloque que se crea como copia tiene que ser igual al bloque original, ya que si no da error. Por este motivo, un bloque que se ha modificado mediante un mutador tiene que ser capaz de recrear el estado modificado al moverse. Lo normal es que la función `updateShape_` solo se encargue de añadir un *input* si se añade un subbloque o eliminar un input si se elimina un subbloque, si esto se programara así al modificar un expressionBlock y moverlo no se podría mover y lanzaría el error que se ve en la Figura 3.29, en esta figura se puede ver que el error ocurre porque falta un *input*, y en el caso de que se esté usando un mutador aconseja que se revise la función domToMutation, esta función es la equivalente a `loadExtraState` si se usara el formato XML.

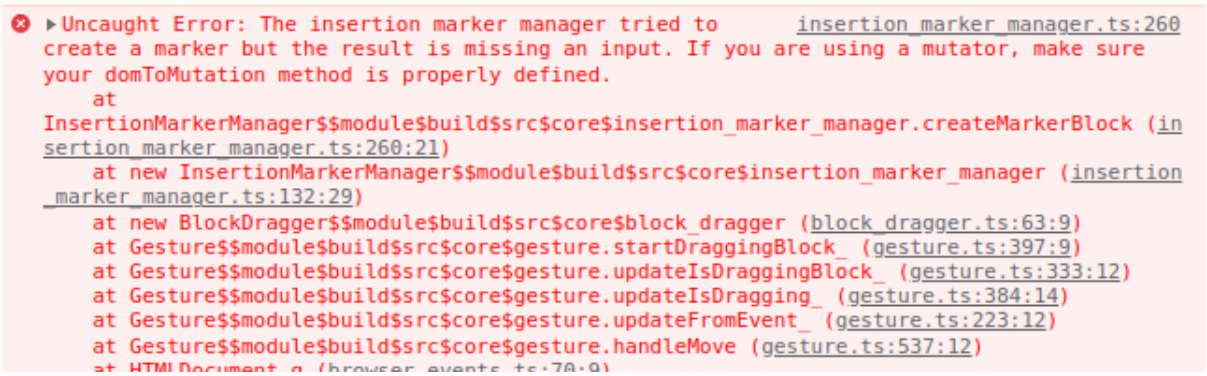

Figura 3.29: Error al mover expressionBlock

Para evitar este error, la función `updateShape_` recibe un *boolean* el cual solo será verdadero si se llama desde la función `loadExtraState` y en el caso de que sea verdadero se le añadirán al bloque tantos *inputs* como valga la variable que se encarga de llevar la cuenta de la cantidad de subbloques que hay, es decir, la cantidad de inputs que el usuario ha añadido al bloque. En el caso de que el *boolean* sea falso, quiere decir que se ha llamado al método `updateShape_` cuando el usuario ha añadido o eliminado un subbloque, por ende, se comprobará la nueva cantidad de subbloques y la antigua y según sea la diferencia se añadirán *inputs* o se eliminarán.

Continuando con el código del Apéndice A.1, en segundo lugar están las funciones `decompose` y `compose`, la función `decompose` recibe por parámetro el *workspace* que se genera al pinchar en el cuadrado azul que se mencionó anteriormente, esta función se encarga de crear en el mini *workspace* el bloque `operationBlock` y tantos bloques operationBlock como haya añadido el usuario anteriormente, si es la primera vez que se abre este mini *workspace* no habrá ningún bloque operation. Para concluir, la función `compose` se encarga de capturar si el usuario añade o elimina `operationBlocks` y procede a actualizar el bloque principal.

3.2.4. VariableBlock

Othimi tiene cuatro tipos de variables, número, lista, lista permutada y matriz. Estas variables se crean utilizando el `variableBlock`, como se puede ver en las figuras 3.30, 3.31, 3.32 y 3.30 el bloque se modifica según sea la variable que haya seleccionado el usuario, para que este comportamiento sea posible hay que hacer uso de un mutador.

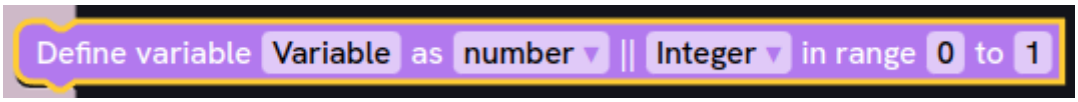

Figura 3.30: Variable number de Othimi

Figura 3.31: Variable list de Othimi

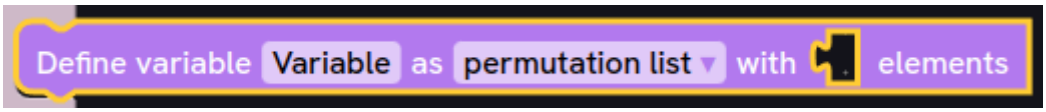

Figura 3.32: Variable permutation list de Othimi

Figura 3.33: Variable matrix de Othimi

El mutador del `variableBlock` se encuentra en el Apéndice A.2, en este apéndice, por un lado, se encuentran las funciones `saveExtraState` y `loadExtraState`; estás funciones realizan lo mismo en todos los mutadores, es por esto que no se van a volver a mencionar cuando se hable de los demás mutadores.

Por otro lado, está la función `updateShape_` que se encarga de modificar el bloque según sea el tipo de variable, por ejemplo, si actualmente el bloque tiene un *input* llamado `COLS` y otro llamado `ROWS` quiere decir que el bloque es una matriz y si el nuevo tipo de variable es una matriz no hay que realizar ninguna modificación en los *inputs*, pero si el nuevo tipo fuese un número habría que eliminar ambos *inputs* y si fuese una lista o una lista permutada solo habría que eliminar el *input* `COLS`, añadiendo que si fuese una lista permutada habría que eliminar el *input* `INTEGERREAL`.

Lo fácil sería eliminar todos los *inputs* cuando se llame a `updateShape_` y que simplemente se añadan los necesarios según el tipo de variable del bloque, el antiguo Prodef lo hacía de esta forma, pero este paradigma presenta un problema grave y es que como los *inputs* se eliminan y se crean con cada llamada a `updateShape_` los bloques que estén conectados a esos *inputs* se desconectan, dando la sensación que la herramienta no está bien programada. Y para concluir con este mutador queda la función `opt_helperFn`, esta función añade al bloque dos validadores, los validadores son funciones que se ejecutan cuando la variable que están vigilando cambia. El primer validador se encarga de guardar en el bloque las variables que controlan el tipo de bloque que se acaba de seleccionar,

esto lo hace cuando se modifique el nombre de la variable y el segundo validador hace lo mismo, pero cuando se modifique el tipo. Además, estos validadores registran el identificador del bloque, el nombre de la variable, el tipo de variable y el identificador del *workspace* en una clase llamada `RelationShipBlocks`, clase estática encargada de relacionar los `variableBlocks`, los `getterBlocks` y los `tablesBlocks`, en la siguiente sección se habla más en profundidad sobre esta clase.

Cabe mencionar que los `tablesBlocks` son como los `variableBlocks`, pero solo con el modo matriz, por esto no se entrará en detalle sobre el mutador de este bloque, su código se encuentra en el Apéndice A.4.

3.2.5. GetterBlock

Los `getterBlocks` son los bloques que se utilizan para acceder a cualquier variable que esté declarada en el *workspace* principal, aunque el tema de las variables se comentará posteriormente en otra sección, ya que en este TFG se ha creado un sistema personalizado. Es importante remarcar que una variable no tiene por qué pertenecer a un `variableBlock`, es decir, los `variableBlock` representan a las variables en el contexto de la creación y resolución del problema, pero en el *workspace* se entiende por variable cualquier texto que se encuentre en un *field input* de un bloque, por ejemplo, la variable que se utiliza como nombre de una columna, es por esto que un `getterBlock` puede tener tres formas: normal, lista o matriz. La forma normal se utiliza para representar cualquier variable que no sea una lista o una matriz, por ejemplo, la variable dentro de un sumatorio, la variable a maximizar o minimizar, el nombre de una columna...; el modo lista se utiliza para listas y listas permutadas y el modo matriz se utiliza para matrices y tablas. En las Figuras 3.34, 3.35 y 3.36 se puede apreciar como el `getterBlock` se modifica según sea el tipo de variable.

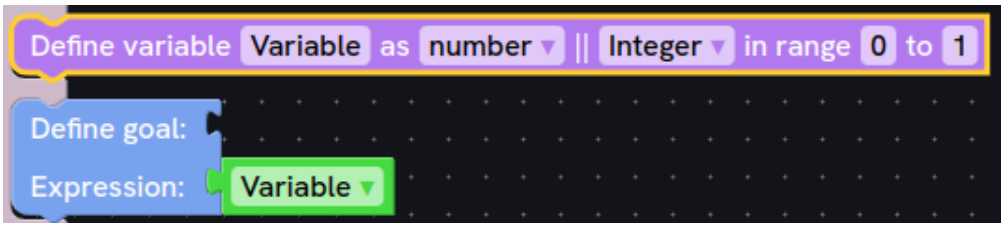

Figura 3.34: Getter number de Othimi

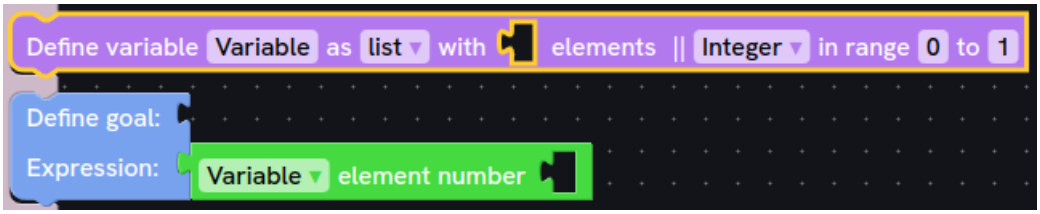

Figura 3.35: Getter list de Othimi

En el Apéndice A.3 se puede ver el código del mutador del bloque `getterBlock`, en este código caben destacar las funciones `loadExtraState` y `updateShape_`. La función `loadExtraState` cumple con su funcionamiento habitual, a excepción de que se encarga de registrar el bloque en la clase `RelationShipBlocks`, clase de la que se hablará a continuación. La función `updateShape_` recibe como parámetro un *array* de *strings*, este

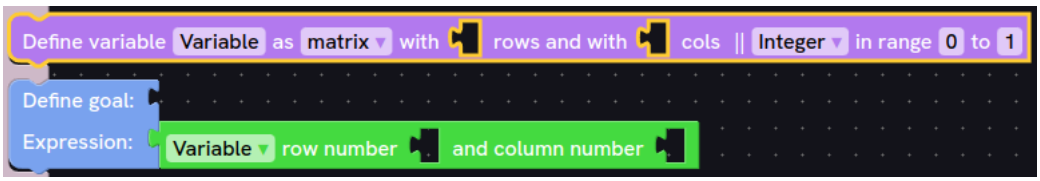

Figura 3.36: Getter matrix de Othimi

array se obtiene de la clase estática *GettersBlocks*, clase que se explicará posteriormente. En esta función se borra el *dropdown* que contiene las variables y se vuelve a añadir, esto se hace porque en el paradigma de Blockly los *dropdown* son *fields* y los *fields* pertenecen a un *input* y no son modificables, por lo tanto, cada vez que se quiera cambiar hay que borrarlo y volverlo a añadir, esta solución presenta un problema y es que al borrarse el *field* pierde todos sus validadores y hay que volverlos a añadir. Después de añadir el *field* la función `updateShape_` añade el validador que se lanza cuando se cambia de variable en el *dropdown*, este validador se encarga de suscribir el bloque en la clase `RelationShipBlocks`. En la función `updateShape_` también adapta el bloque al tipo de variable seleccionada al estilo del `variableBlock`.

La clase `RelationShipBlocks` como se mencionó anteriormente, se trata de una clase estática que permite la relación entre los `variableBlocks` y los `getterBlocks`, el código de esta clase se encuentra en el Apéndice B.1, esta clase tiene 2 métodos a destacar, `setVariable` y `setGetterVariable`. `setVariable` es el método que se encarga de almacenar y actualizar las variables de los `variableBlocks` y tableBlocks, para posteriormente buscar que `getterBlock` tiene esa variable seleccionada y modificarlo según haya cambiado el `variableBlock`, cabe recordar que `setVariable` se va a ejecutar cada vez que un `variableBlock` cambie de nombre o de tipo. El método `setGetterVariable` se encarga de almacenar y actualizar la información de los `getterBlocks` registrados, para posteriormente actualizar el bloque, es importante recordar que `setGetterVariable` se lanza cuando se cambia de variable en el *dropdown*.

3.2.6. Field Inputs

Como se ha mencionado anteriormente, el sistema de variables que implementa Blockly no cumple con los estándares de calidad de Othimi, este sistema presenta varios problemas:

- Hay que utilizar botones para añadir variables, en el antiguo Prodef estos botones se encontraban al principio de cada categoría de la *toolbox*, dándole al usuario la sensación de esos botones crean variables locales a la categoría, cuando las variables son globales. Además, para añadir el nombre de la variable se despliega un cuadro de diálogo, elemento que representa una barrera de accesibilidad. En la Figura 3.37 se puede apreciar la ventana emergente que sale al crear una variable, esta ventana también sale al cambiar el nombre de una variable.

- Las variables no son renombrables desde el propio bloque, para renombrarlas hay que seleccionarlas en un `getterBlock` y posteriormente cambiarles el nombre, este paradigma no es nada intuitivo. En la Figura 3.38 se puede ver esta opción al hacer clic en un `getterBlock`.

- Varios bloques pueden tener la misma variable, hay casos en los que esto está bien, por ejemplo, en el nombre de las columnas, ya que puede haber varias tablas que tengan una columna con el mismo nombre, pero hay otros donde esto no tiene sentido, por ejemplo, en `variableBlocks` o en los paramBlocks. En la Figura 3.38 se puede ver varios bloques compartiendo variable.

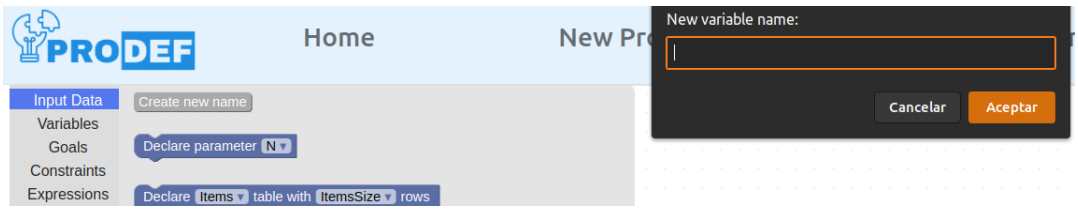

Figura 3.37: Ventana emergente al crear una variable en el antiguo Prodef

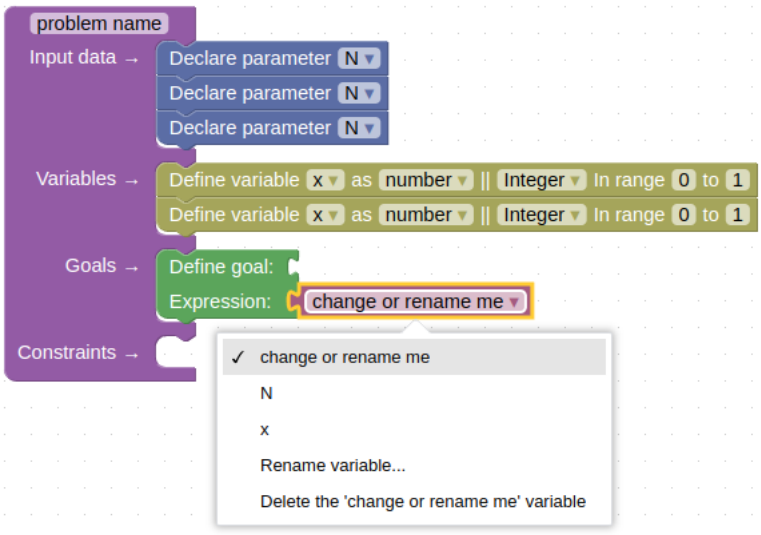

Figura 3.38: Cambio de nombre de una variable en el antiguo Prodef

Blockly no presenta ninguna alternativa a este sistema, por la consecuente en este TFG se ha desarrollado un sistema personalizado que soluciona todos estos problemas, para ello se hace uso de la clase estática `InputsBlocks` y `GettersBlocks`, clases estáticas que se explicarán en la siguiente sección. El sistema de variables creado distingue dos tipos de variables: las únicas y las comunes. Por un lado, están las variables únicas, variables que solo pueden estar en un bloque, es decir, si esta variable ya está en un *field input* que se considere único, al escribirla en otro *field input* saldrá un mensaje notificando al usuario que esa variable ya está en uso, en la Figura 3.39 se puede apreciar este mensaje. Por otro lado, están las variables comunes variables que se pueden escribir en más de un *field input* del mismo tipo, es decir, si hay una columna cuyo nombre de variable sea X, ese nombre se puede poner en otra columna, pero no se puede poner en un *field input* de otro tipo, por ejemplo el de un parámetro. Además, el nombre de las variables ahora se escribe en el propio bloque y para crear o borrar variables simplemente hay que escribirlas en un bloque o borrarlas del bloque, si una variable está escrita en un bloque, existe, si no está escrita, no existe. Por este motivo, los bloques tiene por defecto el nombre `Change me`, nombre por defecto que no es seleccionable desde un `getterBlock`. En la Figura 3.40 se puede ver que todos los bloques tienen por defecto la variable `Change me`.

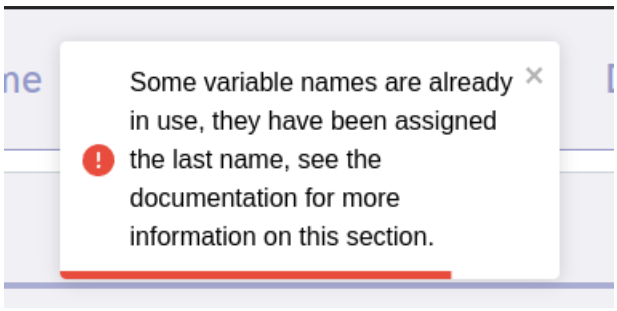

Figura 3.39: Mensaje emergente al usar el nombre de una variable que ya está en uso en Othimi

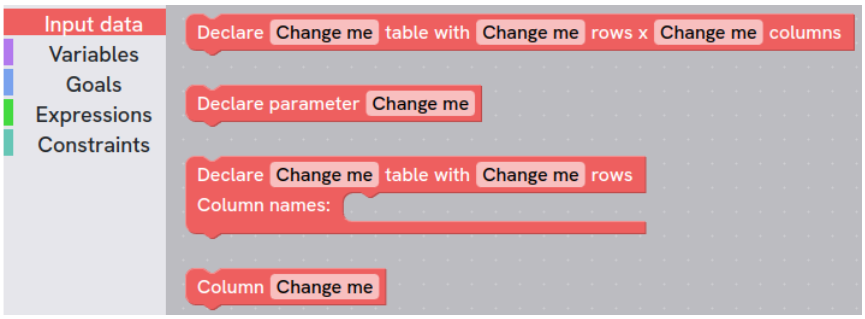

Figura 3.40: Variable por defecto en Othimi

3.2.7. WorkSpace

El *workspace* es creado y configurado en el componente `Problem`. En el Apéndice C se puede encontrar el código de este componente, en este componente se manejan los diferentes eventos que implementa el *workspace*, gracias a esta mecánica ha sido posible la creación del sistema de variables personalizado, haciendo uso de las clases estáticas `InputsBlocks` y `GettersBlocks`.

En primer lugar, el *workspace* lanza diferentes eventos, pero los más importantes y que son los que se capturan son:

- **Evento de creación**: este evento se lanza cada vez que un bloque es creado, un bloque puede ser creado cuando se saca del *toolbox* o cuando se rescata de la papelera, si un bloque es rescatado de la papelera puede ser un bloque o un conjunto de bloques. Cuando un bloque o un conjunto es sacado de la papelera se crea una copia de estos, no se saca exactamente el mismo que se eliminó, sino que se saca un bloque o un grupo de bloques iguales pero con otros identificadores. Esto último es muy importante, ya que en las clases estáticas habrá que realizar limpiezas de los bloques que se borren para que no haya conflictos.

- **Evento de modificación**: evento que es lanzado cada vez que se modifica algo dentro del *workspace*, en Othimi se usa para capturar las modificaciones de los *field inputs*.

- **Evento de borrado**: este evento se lanza cada vez que un bloque o un conjunto de bloques es borrado.

Como se mencionó anteriormente a la hora de crear o eliminar bloques, estos pueden ser simplemente un bloque o una agrupación de estos, los eventos de creación y eliminación sencillamente devuelven el JSON del bloque que se encuentra como base, no

hay forma de distinguir cada caso. Por este motivo se creó un algoritmo recursivo que se encarga de recorrer en profundidad, a nivel de inputs, para posteriormente recorrer los bloques en longitud, a nivel de conexión de bloques; este algoritmo se encarga de comprobar si el bloque que está analizando es un bloque que contenga algún *field input* o si el bloque es un `getterBlock`, en ambos casos lo registra en su respectiva clase. Este algoritmo es lanzado en la creación y en el borrado de los bloques, de esta manera no importa lo que se borre o se cree. El nombre de este método es `blocksExploration`, su código se encuentra dentro del componente `Problem`.

En segundo lugar, está la clase `InputsBlocks`, cuyo código se puede encontrar en el Apéndice B.3, esta clase es la encargada de registrar todos los bloques que tienen un *field input* donde escribir una variable, comprobar que no se rompe el paradigma de las variables comunes y únicas, mencionado en la anterior sección; y actualizar y enviar a la clase `GettersBlocks` las variables disponibles para mostrar en los `getterBlocks`. En esta clase cabe destacar los métodos siguientes:

- `setInput`, esta función es llamada cada vez que se modifica un *field input*, se encarga de llamar a `nameAdmin` en modo modificación.
- `createInput`, esta función se llama cada vez que se crea un bloque y para cada uno de sus *field input* llama a `nameAdmin` en modo creación.
- `nameAdmin`, función que se encarga de registrar las variables y comprobar que no se rompa el paradigma de las variables comunes y únicas.
- `sendInput`, función que recoge de forma única los nombres de las variables para enviárselos a la clase `GettersBlocks`.

En tercer lugar, está la clase `GettersBlock`, cuyo código se puede encontrar en el Apéndice B.2, esta clase es la encargada de registrar todos los `getterBlocks`, actualizar los *dropdown* de estos bloques y de cambiar el valor del *dropdown* si está seleccionada una variable que se acaba de eliminar. Los métodos a destacar de esta clase son 3:

- `modifyDropDown`, método que es llamado desde el método `sendInput` de la clase `InputsBlocks`, este método actualiza las variables y actualiza el dropdown de cada bloque llamando a `changeField`.
- `changeField`, método que elimina el dropdown y lo vuelve a añadir con las variables nuevas, llama al método `searchInArray` para comprobar si la variable que está seleccionada en el *dropdown* sigue existiendo.
- `searchInArray`, método que comprueba si la variable sigue existiendo, si no es así selecciona en el *dropdown* la variable por defecto *Change me* y si existe vuelve a seleccionar la que estaba seleccionada.

Capítulo 4

Back-End

El Back-End ha sido desarrollado principalmente sobre Apolo [1]. Apolo es un *framework* de GraphQL [7], por lo tanto, necesita un *framework* de Back-End por debajo para funcionar, por defecto hace uso de Express, pero en este TFG se hará uso de Koa [10]. Koa está diseñado como un núcleo mínimo y extensible, lo que significa que se basa en módulos externos para proporcionar funcionalidades adicionales, gracias a esto solo hay que instalar los paquetes que se vayan a implementar, no es necesario instalar toda la librería para luego utilizar solo algunas funcionalidades, además el servidor que utilizan los diferentes resolutores de Othimi está desarrollado en Koa, al utilizar Koa también en el Back-End se están simplificando las tecnologías que se utilizan en Othimi, facilitando el futuro desarrollo.

Como base de datos se hará uso de MongoDB[11], MongoDB es una base de datos NoSQL, que se caracteriza por su escalabilidad horizontal, lo que significa que se puede distribuir en varios servidores y por sus ricas consultas, gracias a estas consultas se puede obtener de forma sencilla la información que el Front-End requiera. El Front-End ha sido modificado a tal nivel que los modelos del antiguo Mongoose, librería de MongoDB para Node.js, ya no sirven, es por esto que los modelos se tendrán que rehacer teniendo en cuenta los nuevos requerimientos del Front-End.

El antiguo Prodef solo disponía de un sistema de autentificación, lo cual resulta poco flexible, ya que obligaba a los usuarios a hacer uso de su cuenta de GitHub para poder trabajar con Prodef, pero en Othimi se admite la autentificación utilizando el correo electrónico, es por esto que en el Back-End se tendrán que cifrar las contraseñas haciendo uso de la librería Bcrypt [3].

Para concluir es importante mencionar que el antiguo Prodef no hacía uso de JWT [9] y, por lo tanto, no se podía asegurar que la comunicación Front-End/Back-End se hiciera de forma segura, al hacer uso de JWT en el Back-End de Othimi se puede desconectar a un usuario si su token ha caducado o si su token es incorrecto, aunque esta funcionalidad es de la parte del Front-End.

4.1. Modelos de datos de Mongoose

Como se mencionó anteriormente, los modelos de Mongoose [11] se han vuelto a hacer desde cero para cubrir las necesidades del Front-End. Dentro de los modelos se encuentra los siguientes:

- `AdminUsers`, modelo de datos para guardar que usuarios son administradores de la web.

- `Algorithms`, modelo de datos de los algoritmos.
- `Examples`, modelo de datos de los ejemplos creados para la web.
- `Executions`, modelo de datos de las ejecuciones.
- `Historial`, modelo de datos del historial del usuario.
- `Instances`, modelo de datos de las instancias.
- `Problems`, modelo de datos de los problemas.
- `SharedAlgorithms`, modelo de datos de los algoritmos compartidos entre usuarios.
- `SharedProblems`, modelo de datos de los problemas compartidos entre usuarios.
- `Suggestions`, modelo de datos de las sugerencias.
- `UsersEmail`, modelo de datos de la autentificación usando el *e-mail*.
- `UsersGitHub`, modelo de datos de la autentificación usando GitHub.

De todos los modelos descritos anteriormente solo se está haciendo uso de 4, `Problems`, `Instances`, `UsersEmail` y `UsersGitHub`. El resto de modelos se dejan planteados para el futuro desarrollo de Othimi. Actualmente, se hace uso de un usuario llamado othimi para crear el material de ejemplo para `Problems` y para `Instances`, el objetivo futuro es que no haga falta utilizar un usuario especial, sino que se pueda hacer desde un panel de administración, desde este mismo panel se podrían crear tutoriales y revisar las sugerencias entre otras cosas.

En general todos los modelos tienen la misma estructura, hacen uso de una interfaz base que extiende de `Document` de mongoose, al extender de `Document` se incluyen en los modelos métodos y funcionalidades para manejar las operaciones CRUD de forma sencilla. Se utilizan la interfaz para definir el esquema y finalmente crea el modelo de datos a partir del esquema.

4.1.1. Problems

El código del modelo `Problems` es el siguiente:

```
1  import { Document, Schema, model } from "mongoose";
2
3  type familyType = "Bin packing" | "Knapsack" | "Location" | "Portfolio optimisation" | "
       Scheduling"| "Shortest path" | "Steiner" | "Travelling salesman problem" | "Cutting/
       packing family" | "Vehicle routing" | "I am not sure...";
4
5  export interface problemInterface extends Document {
6    owner: string;
7    name: string;
8    family: familyType;
9    blocklyJSON: string;
10   problemObject: any;
11   parametersCount: number
12   filesCount: number
13 }
14
```

```
const problemSchema = new Schema<problemInterface>({
  owner: {
    type: String,
    required: true
  },
  name: {
    type: String,
    required: true,
  },
  family: {
    type: String,
    required: true
  },
  blocklyJSON: {
    type: String,
    required: true
  },
  problemObject: {
    type: String,
    required: true
  },
  parametersCount: {
    type: Number,
    required: true
  },
  filesCount: {
    type: Number,
    required: true
  },
});

export const Problem = model<problemInterface>("Problem", problemSchema);
```

En este modelo cabe destacar, los campos `family`, `blocklyJSON`, `problemObject`, `parametersCount` y `filesCount`. En el campo `family` se guardará la familia a la cual pertenece el problema, en el campo `blocklyJSON` se guardará el JSON que servirá para recrear los bloques en el Front-End, en el `campoObject` se guarda el código que generaron los bloques, código que tendrá que ser traducido a `ProdefLang` en el resolutor para así poder resolver el problema y por último en los campos `parametersCount` y `filesCount` se guarda la cantidad de archivos y parámetros respectivamente que necesita el problema, de esta forma a la hora de generar la ejecución será mucho más sencillo relacionar problemas e instancias.

4.1.2. Instances

El código del modelo `Instances` es el siguiente:

```
import { Document, Schema, model } from "mongoose";

export interface instanceInterface extends Document {
  owner: string;
  name: string;
  data: {
    parameterInputs: { name: string; value: number }[];
    tablesInputs: { name: string; headers: string[]; data: number[][] }[];
  };
```

```
  parametersCount: number,
  filesCount: number,
  filesDimensions: string[]
}

const instanceSchema = new Schema<instanceInterface>({
  owner: {
    type: String,
    required: true
  },
  name: {
    type: String,
    required: true,
  },
  data: {
    parameterInputs: {
      type: [{ name: String, value: Number }]
    },
    tablesInputs: {
      type: [{ name: String, headers: [String], data: [[Number]] }]
    },
  },
  parametersCount: {
    type: Number,
    required: true
  },
  filesCount: {
    type: Number,
    required: true
  },
  filesDimensions: {
    type: [String],
    required: true
  }
});

export const Instance = model<instanceInterface>("Instance", instanceSchema);
```

En este modelo cabe destacar, los campos `data`, `parametersCount` y `filesCount`. En el campo data se guardan los archivos y parámetros creados en la instancia y en `parametersCount` y `filesCount` la cantidad de estos. Como se mencionó anteriormente, tener esta información por separado permite que la relación de problemas e instancias sea mucho más sencilla.

4.1.3. UsersEmail

El código del modelo `UsersEmail` es el siguiente:

```
import { Document, Schema, model } from "mongoose";

export interface userEmailInterface extends Document {
  userName: string;
  email: string;
  password: string;
}

const userEmailSchema = new Schema<userEmailInterface>({
```

```
  email: {
    type: String,
    unique: true,
    required: true
  },
  password: {
    type: String,
    required: true
  },
  userName: {
    type: String,
    unique: true,
    required: true
  }
});

export const userEmail = model<userEmailInterface>("UserEmail", userEmailSchema);
```

4.1.4. UsersGitHub

El código del modelo `UsersGitHub` es el siguiente:

```
import { Document, Schema, model } from "mongoose";

export interface userGitHubInterface extends Document {
  gitHubID: string,
  userName: string
}

const userGitHubSchema = new Schema<userGitHubInterface>({
  gitHubID: {
    type: String,
    unique: true,
    required: true
  },
  userName: {
    type: String,
    unique: true,
    required: true
  }
});

export const userGitHub = model<userGitHubInterface>("UserGitHub", userGitHubSchema);
```

4.2. Autentificación

Para la autentificación se han creado tres *endpoints*, `/gitHubAuth`, `/eMailAuthLogin` y `/eMailAuthRegister`. Con estos *endpoints* y una arquitectura en tres capas se cubren las necesidades para la autentificación dentro de las opciones que ofrece Othimi.

4.2.1. GitHubAuth

Como se mencionó en el capítulo 3, para la autentificación de GitHub se hace uso del protocolo OAuth. Cuando el usuario le conceda los permisos necesarios a la OAuth App

de Othimi en el Front-End, al Back-End le llegará una petición a la ruta `/gitHubAuth` con un código, con este código más el `clientID` y el `clientSecret` de Othimi se realizará una petición para conseguir un token de GitHub y finalmente con ese token se realizará otra petición para obtener el identificador y el `name` de la cuenta de GitHub del usuario que esté realizando la autentificación. Una vez obtenido el identificador y el `name` se pasaría a la capa de datos, para ello se hace uso del controlador `authController`, para ser más específicos del método `getAuthGitHub`.

```
authRouter.get("/gitHubAuth", async (ctx) => {

  const params = "?client_id=" + clientID + "&client_secret=" + clientSecret + "&code=" +
     ctx.query.code;

  try {
    const responseToken = await axios.post("https://github.com/login/oauth/access_token"
    + params);
    const gitHubToken = responseToken.data.split("&")[0].split("=")[1];

    const config = {
      headers: {
        "Authorization": "Bearer " + gitHubToken
      }
    }

    const responseUserData = await axios.get("https://api.github.com/user", config);
    const id = responseUserData.data.id;
    const name = responseUserData.data.login;

    const data = await authController.getAuthGitHub(id, name);

    ctx.status = data.res;
    data.res === 200 ? ctx.body = { id, name, serverToken: data.serverToken } : ctx.body
    = { error: data.error }

  } catch (error) {
    ctx.status = error.response.status;
    ctx.body = { error: "Error with GitHub auth." }
  }
});
```

En el controlador de la autentificación con GitHub, primero se comprueba si el usuario ya existe en la base de datos, si existe se comprueba si su nombre ha cambiado y si ha cambiado se actualiza y en el caso de que no exista se guardaría en la base de datos. Finalmente, se devuelve al Front-End el identificador del usuario junto a un JWT.

```
async getAuthGitHub(id: string, userName: string) {
    try {
      const userDataBase = await userGitHub.findOne({gitHubID: id});

      if (!userDataBase) {
        const user = new userGitHub({gitHubID: id});
        await user.save();
      } else if (userDataBase.userName !== userName) {
        const allowedUpdates = ['userName'];
        const actualUpdates = Object.keys({userName});
        const isValidUpdate =
          actualUpdates.every((update) => allowedUpdates.includes(update));
```

```
13
14        if (!isValidUpdate) {
15          return ({error: "Invalid update.", res: 400});
16        }
17
18        const userUpdate = await userGitHub.findOneAndUpdate({gitHubID: id}, {userName},
    {
19          new: true,
20          runValidators: true,
21        });
22
23        if (!userUpdate) {
24          return ({error: "Identifier not found.", res: 404});
25        }
26      }
27
28      return {serverToken: jwt.sign({id}, jwtSecret), res: 200};
29    } catch (error) {
30      return {error: error, res: 500};
31    }
32  }
```

4.2.2. EmailAuthRegister

Para realizar el registro utilizando el *E-mail,* primero se comprueba que se hayan recibido todos los datos que se necesita y se pasa a la capa de datos.

```
1  authRouter.post("/eMailAuthRegister", async (ctx) => {
2
3    try {
4      ctx.status = 404;
5
6      const userName: string | undefined = ctx.request.body.userName ? ctx.request.body.
     userName.toString() : undefined;
7      const email: string | undefined = ctx.request.body.email ? ctx.request.body.email.
     toString() : undefined;
8      const password: string | undefined= ctx.request.body.password ? ctx.request.body.
     password.toString() : undefined;
9
10     if (email && password && userName) {
11       const errors = validateData(userName, email, password);
12
13       if (errors === "") {
14         const data = await authController.postAuthEmailRegister(email, userName, password
     );
15
16         ctx.status = data.res;
17         data.res === 200 ? ctx.body = { email: data.email, userName: data.userName,
     serverToken: data.serverToken } : ctx.body = { error: data.error }
18       } else {
19         ctx.body = {error: errors};
20       }
21     } else {
22       ctx.body = { error: "Email, UserName and password are required." }
23     }
24   } catch {
25     ctx.body = { error: "Error with Email register auth." }
```

```
  }
});
```

En la capa de datos se almacena la información del nuevo usuario y se devuelve el *E-mail,* el nombre de usuario y un JWT.

```
async postAuthEmailRegister(email: string, userName: string, password: string) {
    try {
      const user = new userEmail({userName, email, password})

      user.password = await hashPassword(user.password);
      await user.save();

      return({serverToken: jwt.sign({userEmail: user.email}, jwtSecret), email: user.email, userName: user.userName, res: 200})
    } catch(error) {
      return {error: error, res: 500};
    }
  }
```

4.2.3. EmailAuthLogin

Para realizar el *login* se sigue el mismo esquema que en el registro, a diferencia de que en el *login* solo es necesario el correo o el nombre de usuario y además la contraseña.

```
authRouter.post("/eMailAuthLogin", async (ctx) => {

  try {
    ctx.status = 404;

    const userName: string | undefined = ctx.request.body.userName ? ctx.request.body.userName.toString() : undefined;
    const email: string | undefined = ctx.request.body.email ? ctx.request.body.email.toString() : undefined;
    const password: string | undefined = ctx.request.body.password ? ctx.request.body.password.toString() : undefined;

    if ( password && (email || userName)) {

      const errors = validateData(userName, email, password);

      if (errors === "") {
        const data = await authController.postAuthEmailLogin(email, userName, password);

        ctx.status = data.res;
        data.res === 200 ? ctx.body = { email: data.email, userName: data.userName, serverToken: data.serverToken } : ctx.body = { error: data.error }
      } else {
        ctx.body = {error: errors}
      }
    } else {
      ctx.body = { error: "Email and password or UserName and password are required." }
    }
  } catch {
    ctx.body = { error: "Error with Email login auth." }
  }
});
```

En la capa de datos se comprueba si el *login* se va a realizar utilizando la contraseña o el correo, se comprueba que la información sea correcta y se realiza el *login* devolviendo lo mismo que en el registro.

```
async postAuthEmailLogin(email: string | undefined, userName: string | undefined,
    password: string) {
    try {
      let user;
      if (email) user = await userEmail.findOne({email})
      else       user = await userEmail.findOne({userName})

      if (!user) return {error: "User not found.", res: 404};

      const isMatch = await comparePassword(password, user.password);
      if (!isMatch) return {error: "Invalid password.", res: 400};

      return {serverToken: jwt.sign({userEmail: user.email}, jwtSecret), email: user.
    email, userName: user.userName, res: 200}

    } catch (error) {
      return {error: error, res: 500};
    }
  }
```

4.2.4. Seguridad

La seguridad en este Back-End no depende únicamente de las expresiones regulares que se aplican en el Front-End durante la autentificación. En el Back-End también se aplican medidas de seguridad, la primera de ellas es sacar cada campo que haya enviado el usuario desde el Front-End y convertirlo a *string*, de esta forma se evita trabajar directamente con variables JSON que algún atacante podría utilizar para explotar alguna vulnerabilidad en la base de datos:

```
const userName: string | undefined = ctx.request.body.userName ? ctx.request.body.
    userName.toString() : undefined;
    const email: string | undefined = ctx.request.body.email ? ctx.request.body.email.
    toString() : undefined;
    const password: string | undefined= ctx.request.body.password ? ctx.request.body.
    password.toString() : undefined;
```

En segundo lugar, una vez comprobado que las variables que se esperan recibir han sido recibidas, se les aplica una expresión regular para comprobar que cumplen con el formato esperado, mismo formato que se ha aplicado en el Front-End. Para ello se hace uso del método `validateData`:

```
const RegularExpressions = {
  userName: /^[a-zA-Z0-9]{3,15}$/,
  eMail: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/,
  password: /^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)[A-Za-z\d]{8,20}$/
}


function validateData(userName: string | undefined, email: string | undefined, password:
    string) {
  let error: string = "";

```

```
  if (userName)
    if (!RegularExpressions.userName.test(userName)) error += "Username is invalid\n";
  if (email)
    if (!RegularExpressions.eMail.test(email)) error += "E-Mail is invalid\n";
  if (!RegularExpressions.password.test(password)) error += "Password is invalid\n"

  return error;
}
```

Y por último, si error está vacío, se procede a acceder a la capa de datos, siempre tratando las variables como *strings* y no como JSONs para evitar vulnerabilidades.

4.3. JWT

JWT [9] es un estándar RFC 7519 que define un formato compacto y autónomo para transmitir información de forma segura entre partes como un objeto JSON. Para conseguir una comunicación segura entre Front-End y Back-End se implementará como *middleware*, después de la autentificación y antes de acceder al *endpoint*, la jerarquía de *middlewares* dentro de la *app* sería la siguiente:

```
app.use(cors()).
  use(bodyParser()).
  use(loggerKoa()).
  use(authRouter.routes()).
  use(verifyToken).
  use(koaMiddleware(server));
```

Siendo `koaMiddleware(server)` el servidor de Apolo, `verifyToken` el *middleware* de JWT y `authRouter.routes()` los *endpoints* para la autentificación.

El *middleware* de JWT es el siguiente:

```
import jwt from 'jsonwebtoken';
import {jwtSecret} from '../env/config';

export const verifyToken = async (ctx, next) => {
  const header = ctx.request.header['authorization'];

  if (typeof header !== 'undefined') {
    const bearer = header.split(' ');
    const token = bearer[1];

    await jwt.verify(token, jwtSecret, async (err) => {
      if (err) {
        ctx.body = {res: 401, error: "Invalid token"};
        ctx.status = 401;
      } else {
        await next();
      }
    })

  } else {
    ctx.body = {res: 401, error: "No token provided"};
    ctx.status = 401;
  }
}
```

En este *middleware* primero se obtiene el token y se comprueba que sea válido, si no es válido o no hay token no se accede al siguiente *middleware*, además se retorna un código de error 401 que indica que las credenciales de autentificación son inválidas.

4.4. Bcrypt

La librería Bcrypt [3] se utiliza en la autentificación de *e-mail*, para cifrar contraseñas y para comprobar que una contraseña sin cifrar sea igual a una contraseña ya cifrada, las contraseñas cifradas son las que se guardan en la base de datos. Para realizar lo anteriormente mencionado se hacen uso de dos funciones:

```
// Encriptar password, para el registro.
export const hashPassword = async (password: string) => {
  const salt = await bcrypt.genSalt(10);
  return await bcrypt.hash(password, salt);
};

// Comparar password, para el login.
export const comparePassword = async (password: string, hashPassword: string) => {
    return await bcrypt.compare(password, hashPassword);
}
```

La función `hashPassword` se utiliza en el registro y se encarga de hashear la contraseña para guardarla y la función `comparePassword` compara la contraseña que se envía en el login con la contraseña hasheada que se guarda en la base de datos.

4.5. GraphQL

GraphQL [7] es un lenguaje de consulta que tiene sus propios tipos de datos básicos, entre ellos están:

- *String*, representa una cadena de texto.
- *Int*, representa un número entero.
- *Float*, representa un número de punto flotante.
- *Boolean*, representa un valor booleano.
- *ID*, representa un identificador único, para Othimi ese identificador es el identificador de la base de datos.

Además de estos tipos de datos básicos, existen otros 2. El primero es el *Input*, tipo de dato de entrada que representa un objeto con múltiples campos, lo reciben las *querys* y las *mutations*, y el segundo es el *Type*, tipo de dato de salida que representa un objeto con múltiples campos, lo devuelven las *querys* y las *mutations*.

También hay 2 tipos de consultas. La primera es la *Query*, consulta para obtener datos del Back-End, corresponde a un GET en una API REST y la segunda es la *Mutation*, consulta que modifica datos del Back-End, corresponde con un POST, PATH y DELETE de una API REST.

4.5.1. Definiciones de tipos

Para los problemas se tuvo que crear 2 *querys* y 3 *mutations*, la definición de las *querys* es la siguiente:

```
extend type Query {
    problems(owner: String!, page: Int!, name: String, family: String tableNumbers: Int,
    paramsNumber: Int): ProblemPage
    problem(id: String!): Problem
  }
```

La primera *query* busca un problema por el identificador y lo devuelve y la segunda *query* es la que se utiliza el buscador de problemas del Front-End, esta *query* recibe los siguientes parámetros:

- `owner`, dueño de los problemas.
- `page`, página que solicita el usuario.
- `name`, nombre del problema, en caso de que se busque un nombre específico.
- `family`, filtro de los tipos de familia de los problemas.
- `tableNumbers`, filtro para la cantidad de tablas.
- `paramsNumbers`, filtro para la cantidad de parámetros.

La definición de las mutaciones es la siguiente:

```
extend type Mutation {
    createProblem(input: ProblemInput!): Problem
    deleteProblem(id: String!): Boolean
    pathProblem(id: String!, input: ProblemInput!): Boolean
  }
```

La primera mutación crea un problema (POST), la segunda elimina un problema (DELETE) y la tercera modifica un problema (PATH).

Solo hay un tipo de dato *input*, este tipo de dato recibe todo lo que un problema necesita para ser creado, a excepción del identificador que se generará cuando el problema se guarde en la base de datos:

```
input ProblemInput {
    owner: String!
    name: String!
    family: String!
    blocklyJSON: String!
    problemObject: String!
    parametersCount: Int!
    filesCount: Int!
  }
```

Y por último los tipos de datos *type*:

```
type ProblemPage {
    info: InfoPage
    problem: [Problem]
    familys: [String]
    tables: [Int]
```

```
    params: [Int]
  }

type Problem {
    id: ID!
    owner: String!
    name: String!
    family: String!
    blocklyJSON: String!
    problemObject: String!
    parametersCount: Int!
    filesCount: Int!
  }
```

El primero es el que devuelve la *query* que se utiliza para la paginación en el Front-End, por lo tanto, devuelve una información básica de cada problema junto a la información de la paginación y la segunda *query* devuelve toda la información de un problema, incluyendo el identificador. El tipo `InfoPage` es el siguiente:

```
type InfoPage {
    count: Int!
    pages: Int!
    next: Int!
    prev: Int!
  }
```

Para las instancias la lógica de tipos, *querys* y mutaciones es la misma que para los problemas, por lo tanto, no se entrará en detalle, las definiciones son las siguientes:

```
extend type Query {
    instances(owner: String!, page: Int!, name: String, tableNumbers: Int, paramsNumber:
    Int): InstancePage
    instance(id: String!): Instance
  }

extend type Mutation {
    createInstance(input: InstanceInput!): Instance
    deleteInstance(id: String!): Boolean
    pathInstance(id: String!, input: InstanceInput!): Boolean
  }



input InstanceInput {
    owner: String!
    name: String!
    data: DataInput!
    parametersCount: Int!
    filesCount: Int!
    filesDimensions: [String!]!
  }

input ParameterInput {
    name: String!
    value: Float!
  }

input TablesInput {
```

```
    name: String!
    headers: [String]!
    data: [[Float!]!]!
  }

input DataInput {
    parameterInputs: [ParameterInput!]!
    tablesInputs: [TablesInput!]!
  }



type InstancePage {
    info: InfoPage
    instance: [Instance]
    tables: [Int]
    params: [Int]
  }

type Parameter {
    name: String!
    value: Float!
  }

type Tables {
    name: String!
    headers: [String]!
    data: [[Float!]!]!
  }

type Data {
    parameterInputs: [Parameter!]!
    tablesInputs: [Tables!]!
  }

type Instance {
    id: ID!
    owner: String!
    name: String!
    data: Data!
    parametersCount: Int!
    filesCount: Int!
    filesDimensions: [String!]!
  }
```

En el Apéndice D.1 y en el Apéndice D.2 se puede encontrar las implementaciones de las *querys* y de las mutaciones para los problemas y las instancias respectivamente.

4.5.2. Controladores

El código de los controladores de los problemas y de las instancias se puede encontrar en los Apéndices E.1 y E.2 respectivamente. El método a destacar de estos controladores es el que se encarga de devolver la información que se utiliza en el Front-End para realizar la paginación, utilizar filtros y utilizar el buscador. Este método es `getProblems` para los problemas y `getInstances` para las instancias, como es prácticamente igual en los dos, solo se explicará `getProblems`. Este método comienza añadiendo el nombre del `owner` a

un JSON y comprobando si hay filtros aplicándose, si hay filtros aplicándose los añade al JSON.

```
let search = {owner};
if (tableNumbers !== undefined) search["filesCount"] = tableNumbers;
if (paramsNumber !== undefined) search["parametersCount"] = paramsNumber;
if (name !== undefined) search["name"] = new RegExp(name);
if (family !== undefined) search["family"] = family;
```

Posteriormente, se obtienen la cantidad de elementos que cumplen con los requisitos de la búsqueda y se calcula cuantas páginas van a haber:

```
const totalElements: number = await Problem.countDocuments(search);
const pages: number = totalElements % 10 === 0 ? totalElements / 10 >> 0 : totalElements
    / 10 + 1 >> 0;
```

A continuación se calcula cuál es la siguiente y anterior página:

```
const next = page < pages ? page + 1 : -1;
const prev = page !== 1 ? page - 1 : -1;
```

Se obtiene la información de la página en cuestión:

```
const elements = await Problem.find(search).skip(10 * (page - 1)).limit(10);
```

Y por último se obtienen los nuevos filtros y se devuelve la información:

```
const tables = await Problem.find(search, {_id:0, filesCount: 1});
const setTables = new Set(tables.map(result => result.filesCount));
const finalTables = [...setTables];

const params = await Problem.find(search, {_id:0, parametersCount: 1});
const setParams = new Set(params.map(result => result.parametersCount));
const finalParams = [...setParams];

const familys = await Problem.find(search, {_id:0, family: 1});
const setFamilys = new Set(familys.map(result => result.family));
const finalFamily = [...setFamilys];

const info = {
  count: totalElements,
  pages,
  next,
  prev
};

return {info, problem: elements, familys: finalFamily, tables: finalTables, params:
    finalParams};
```

Capítulo 5

Modelado de problemas en Othimi

Este capítulo consta de dos partes, en la primera parte se muestra la implementación de diferentes problemas en Othimi y en la segunda parte se muestra un vídeo realizado por el alumno donde se crea un problema desde cero y una instancia para ese problema.

Los problemas de optimización que se han implementado en Othimi son el *knapsack problem*, el *travelling salesman problem*, el *vehicle routing problem*, el *bin packing problem*, el *cutting packing problem* y el *capacited facility location problem*.

5.1. Knapsack problem

El *knapsack problem*, o también conocido en español como problema de la mochila, es un problema que consiste en seleccionar a los elementos con mayor valor para colocarlos en una mochila con una capacidad limitada, de modo que se maximice el valor total de los elementos que se hayan seleccionado. Este problema recibe, por un lado, una lista de objetos que tienen asociados un peso y un valor y, por otro lado, la capacidad de la mochila donde se van a introducir los objetos. Su modelo matemático es el siguiente:

$$\begin{aligned} \text{máx} \quad & \sum_{i=1}^{N} v_i x_i \\ \text{s.a.} \quad & \sum_{i=1}^{N} w_i x_i \leq W \\ & x_i \in \{0, 1\} \end{aligned} \tag{5.1}$$

En la Figura 5.1 se puede observar la implementación de este modelo haciendo uso de Othimi. En esta implementación el campo `Input data` recibe la capacidad de la mochila, siendo esta `maxWeight` dentro del problema y W en el modelo matemático, este mismo *input* recibe también una tabla llamada `items`, esta tabla tiene `N` elementos y para cada elemento tiene un `Value` y un `Weight`, en el modelo matemático el `Value` para un elemento correspondería con la variable v_i y el `Weight` se correspondería con la variable w_i. En el campo `Variables` se crea una lista llamada `x`, esta lista tiene el mismo tamaño que filas tiene la tabla `items`, esto es así, ya que finalmente en cada posición habrá un 1 o un 0 si el `item` es guardado o no en la mochila.

En el campo `Goals` se define la función objetivo que tiene como objetivo maximizar la suma de los productos $v_i x_i$, donde v_i es un valor asociado a cada variable x_i. Esto implica que se busca encontrar los valores de x_i que maximicen la suma ponderada de las variables. Por último, en el campo `Constraints` se define una restricción donde el total de pesos que se han guardado en la mochila no deben superar la capacidad máxima de la mochila.

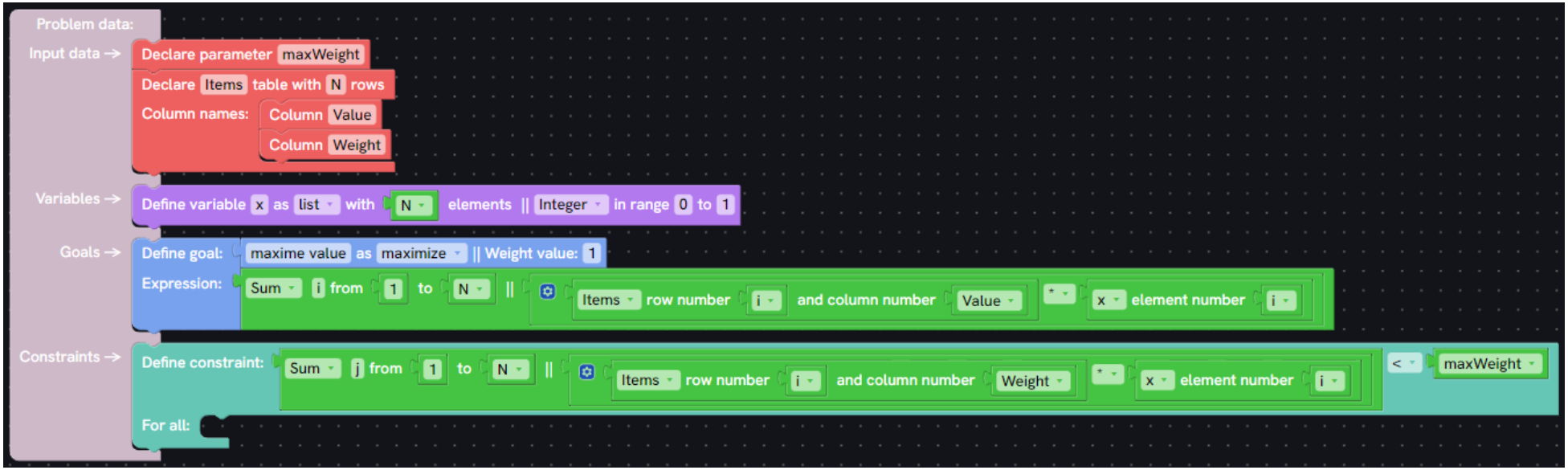

Figura 5.1: Knapsack definition

5.2. Travelling salesman problem

El *travelling salesman problem* o también conocido como TSP, es un problema cuyo objetivo es encontrar el recorrido más corto que visite todos los nodos una vez y que vuelva al origen. este problema recibe una matriz de ciudades donde cada valor es la distancia entre la ciudad *i* y la ciudad *j*. Su definición matemática es la siguiente:

$$\begin{aligned} \text{mín} \quad & d_{c_N,c_1} + \sum_{i=1}^{N-1} d_{c_{i+1},c_i} \\ & c_i \in \{0,1\} \end{aligned} \tag{5.2}$$

En las Figuras 5.2 y 5.3 se puede observar la implementación de este modelo haciendo uso de Othimi. En esta implementación el campo `Input data` recibe tabla de distancias de tamaño `N x N`, esta tabla es representada en el modelo por la variable d, y el tamaño `N` se representa mediante la variable N. En el campo `Variables` se crea una lista permutada de las ciudades que se visitará, la lista es permutada, ya que el orden de en el cual se visiten las ciudades es lo realmente importante, sí o sí hay que visitar todas las ciudades. Esta lista permutada corresponde con la variable c del modelo matemático. Por último, en el campo `Goals`, se define la función objetivo que tiene por objetivo minimizar la distancia recorrida para pasar por todas las ciudades.

Figura 5.2: Travelling salesman problem en Othimi, parte 1

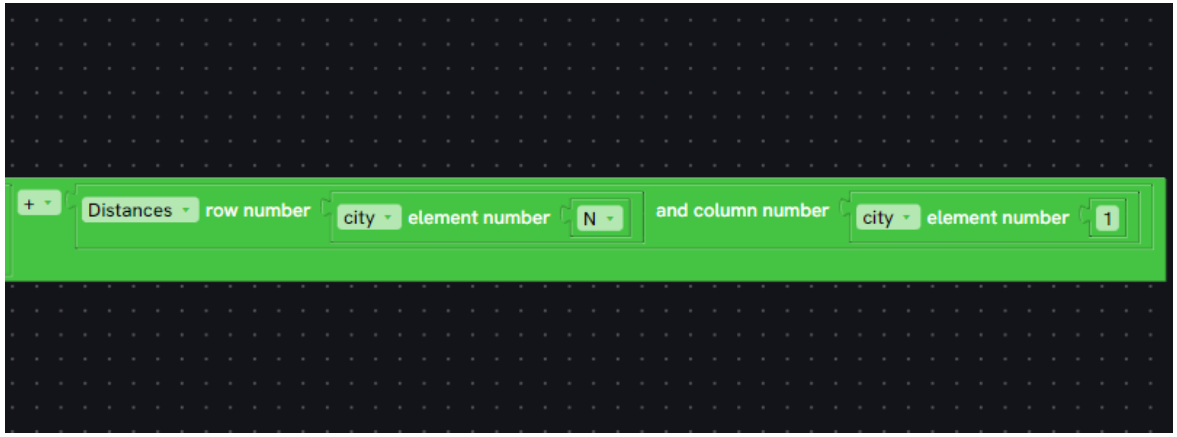

Figura 5.3: Travelling salesman problem en Othimi, parte 2

5.3. Vehicle routing problem

El VRP o problema de enrutamiento de vehículos, es un problema que buscar el conjunto de caminos óptimos para un grupo de vehículos que deben cumplir con las demandas de un grupo de clientes. Este problema recibe una matriz donde cada valor es el costo de ir de la ciudad *i* a la ciudad *j* y una cantidad de vehículos disponibles. La definición matemática del problema es la siguiente:

$$
\begin{aligned}
\text{mín} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}x_{ij} \\
\text{s.a.} \quad & \sum_{i=1}^{N} x_{i0} = m \\
& \sum_{j=1}^{N} x_{0j} = m \\
& \sum_{i=1}^{N} x_{ij} = 1, \quad j = 0, ..., N \\
& \sum_{j=1}^{N} x_{ij} = 1, \quad i = 0, ..., N \\
& x_{ij} \in \{0, 1\}, \quad i, j = 0, ..., N
\end{aligned}
\tag{5.3}
$$

En las Figuras 5.4 y 5.5 se puede observar la implementación de este modelo haciendo uso de Othimi. En esta implementación el campo `Input data` recibe una tabla llamada `citys` de `N x N`, esta tabla representa el costo de ir de una ciudad a otra, en la definición del problema correspondería con la variable c_{ij} y el tamaño `N` se representan mediante la variable N. Además, recibe una constante llamada `Vehicles` que corresponde con la cantidad de vehículos disponibles m. En el campo `Variables` se crea una matriz de `N x N`, que toma el valor 1 si se toma la ruta de la ciudad i a la ciudad j, se representa mediante la variable x_{ij}. En el campo `Goals` se crea la función objetivo que tiene como objetivo minimizar la suma de los costos de viajar entre las diferentes ciudades. Y por último, en el campo `Constraints` se definen varios sumatorios para asegurar que solo haya un 1 en cada fila y columna de x_{ij} y el resto sean 0, además la suma de x_{i0} debe ser igual al total de vehículos disponibles y lo mismo para x_{0j}.

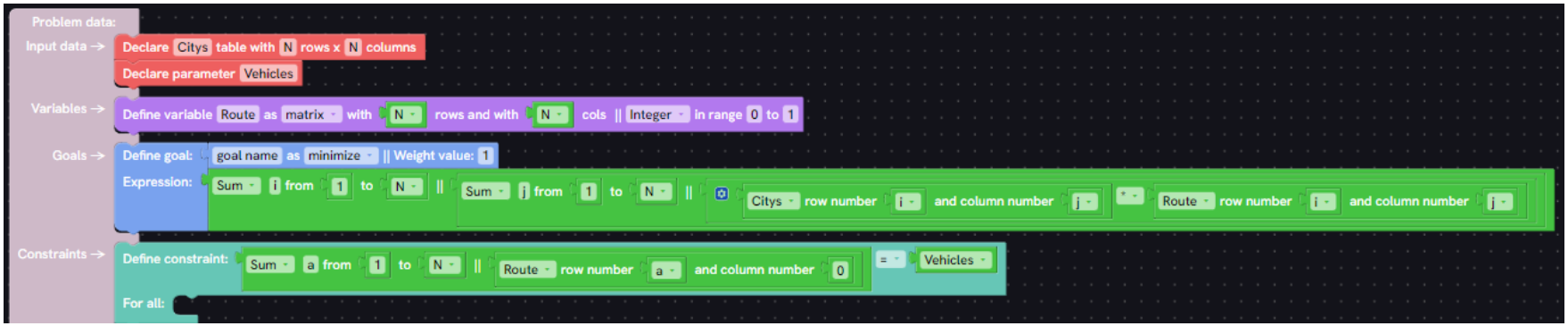

Figura 5.4: Vehicle routing problem en Othimi, parte 1

Figura 5.5: Vehicle routing problem en Othimi, parte 2

5.4. Bin packing problem

El *bin packing problem* o en español problema de la mochila, es un problema que recibe una lista de objetos junto a su peso, una lista de mochilas disponibles y la capacidad de cada mochila. El objetivo es encontrar la forma más eficiente de almacenar los objetos en las mochilas, teniendo en cuenta que se quiere utilizar la cantidad mínima de mochilas. Su definición matemática es la siguiente:

$$
\begin{aligned}
\text{mín} \quad & \sum_{i=1}^{P} y_i \\
\text{s.a.} \quad & \sum_{j=1}^{N} w_j x_{ij} \leq c_i y_i, \quad i = 1, ..., P \\
& \sum_{i=1}^{P} x_{ij} = 1, \quad j = 1, ..., N \\
& y_i = 0 \; or \; 1, \quad i = 1, ..., P \\
& x_{ij} = 0 \; or \; 1, \quad i = 1, ..., P; j = 1, ..., N
\end{aligned}
\tag{5.4}
$$

En la Figura 5.6 se puede observar la implementación de este modelo haciendo uso de Othimi. En esta implementación el campo `Input data` recibe una tabla llamada `Objects` que tiene `N` filas y una columna llamada `Size`, esta tabla representa para cada objeto su tamaño, la definición de `Size` de un `Object` es w_i y la representación de `N` es N. Además, recibe otra tabla llamada `Packages` que representa la capacidad de los paquetes, tiene `PackagesCounts` filas y tiene una columna llamada `PackagesSize` donde aparece la capacidad del paquete, la definición de la capacidad de un paquete es c_i y la definición de la cantidad de paquetes es P. En el campo `Variables` se crea una lista llamada `Selected` con tantos elementos como paquetes haya, es 1 si el paquete se usa y 0 si el paquete no se usa, su representación es y_i. Además, se crea una matriz que relaciona paquetes con objetos, esta matriz se llama `Assigned`, vale 1 si el objeto j se ha asignado al paquete i, su definición es x_{ij}. En el campo `Goals` se crea una función objetivo cuyo objetivo es minimizar la cantidad de paquetes a utilizar. Y por último, en el campo `Constraints` se definen una serie de restricciones para verificar que el peso que se almacena en cada paquete no supera el máximo del paquete en cuestión y para que un objeto solo pueda estar en un paquete.

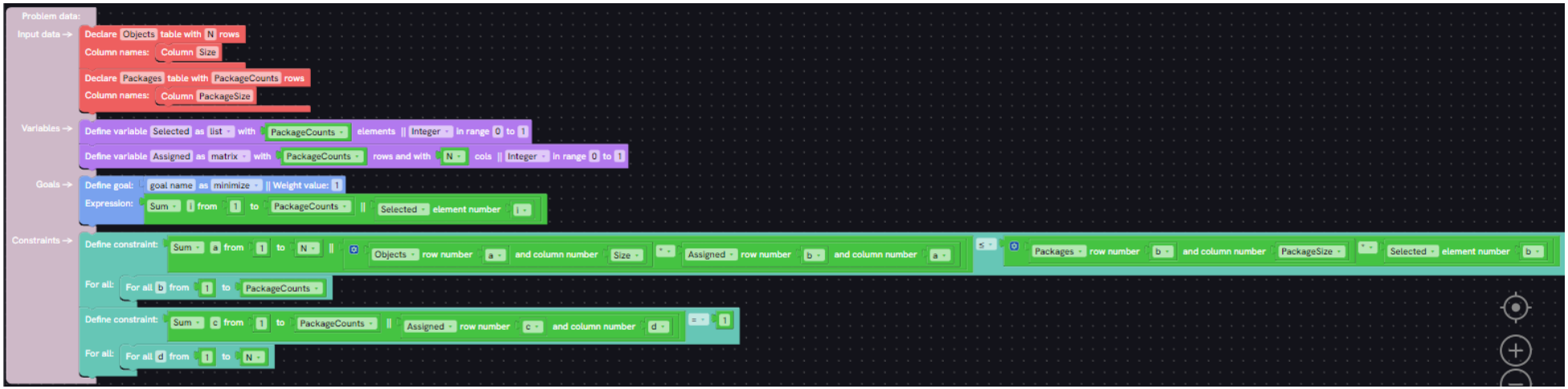

Figura 5.6: Bin packing problem en Othimi

5.5. Cutting packing problem

El *cutting packing problem* es un problema que recibe una lista de órdenes donde el valor de la orden *i* es la cantidad de piezas que requiere y una lista de patrones donde para el patrón *i* se indica cuántas veces va a ser utilizado. El objetivo de este problema es minimizar los residuos. Su definición matemática es la siguiente:

$$\begin{aligned} \text{mín} \quad & \sum_{i=1}^{N} c_i x_i \\ \text{s.a.} \quad & \sum_{i=1}^{N} a_{ij} x_i \geq q_j, \quad j = 1, ..., M \end{aligned} \tag{5.5}$$

En la Figura 5.7 se puede observar la implementación de este modelo haciendo uso de Othimi. En esta implementación el campo `Input data` recibe una tabla llamada `OrdersPieces`, que tiene `Order` filas y una columna llamada Pieces, esta tabla representa cuantas piezas requiere cada orden, su definición es q_i y la definición de `Order` es M. Además, recibe otra tabla llamada `PatternsUsed`, que tiene `Pattern` filas y una columna llamada `Used`, esta tabla representa cuantas veces se va a utilizar un patrón, su definición es x_i y la definición de `Pattern` es N. En el campo `Variables` se crea una lista llamada `Cost` que representa los desperdicios que generó el patrón en cuestión, es decir, el material que no se aprovechó, su definición es c_i. También se crea una matriz llamada `RelationPatternOrder` que indica el número de veces que la orden j aparece en el patrón i, su definición es a_{ij}. En el campo `Goals` se crea una función objetivo cuyo objetivo es minimizar el material que se desperdicia al crear cada patrón. Y por último, en el campo `Constraints` se define una restricción para que se creen más piezas o las mismas que requiera la orden en cuestión.

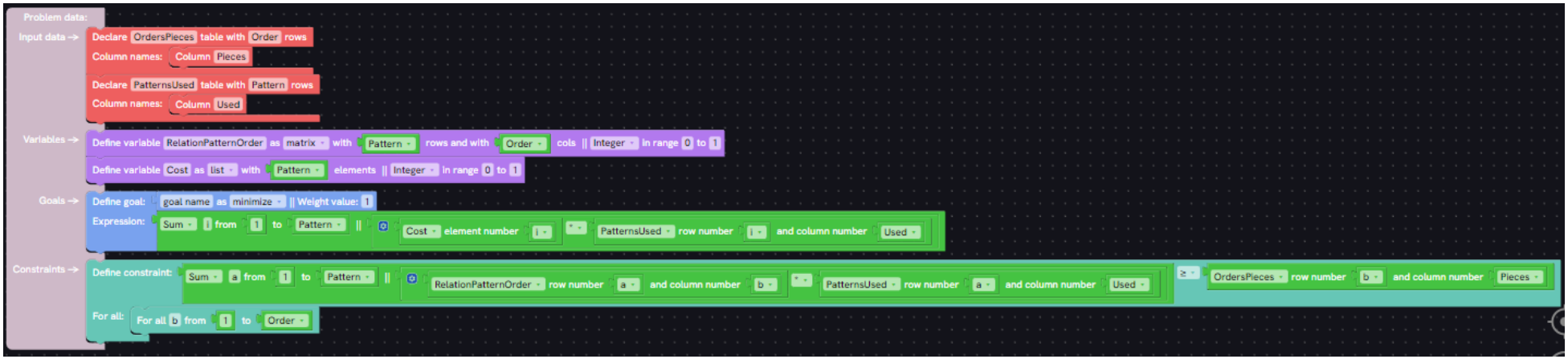

Figura 5.7: Cutting packing problem en Othimi

5.6. Capacited facility location problem

El *capacited facility location problem* es un problema cuyo objetivo es determinar la ubicación óptima de las instalaciones y asignar clientes a esas instalaciones, teniendo en cuenta las capacidades de las instalaciones y las demandas de los clientes. El problema recibe una lista de las demandas de los clientes, una matriz de costes de transporte que relaciona clientes con instalaciones, una lista con los costos anuales de mantenimiento de las instalaciones y una lista con las capacidades de las instalaciones. Su definición matemática es la siguiente:

$$
\begin{aligned}
\text{mín} \quad & \sum_{j=1}^{m} f_j y_j + \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \\
\text{s.a.} \quad & \sum_{j=1}^{m} x_{ij} = d_i, \quad i = 1, ..., n \\
& \sum_{i=1}^{n} x_{ij} \leq M_j y_j, \quad j = 1, ..., m \\
& x_{ij} \leq d_i y_j, \quad i = 1, ..., n; j = 1, ..., m \\
& x_{ij} \geq 0, \quad i = 1, ..., n; j = 1, ..., m \\
& y_j \in \{0, 1\}, \quad j = 1, ..., m
\end{aligned}
\tag{5.6}
$$

En la Figura 5.8 se puede observar la implementación de este modelo haciendo uso de Othimi. En esta implementación el campo `Input data` recibe una tabla llamada `Facility`, que tiene `M` filas y `N` columnas, esta tabla relaciona los costes de ir de la instalación i al cliente j, por ende `N` representa la cantidad de instalaciones y `M` la cantidad de clientes, la definición de `Facility` es c_{ij}, la definición de `N` es n y la de `M` es m. Además, recibe otra tabla llamada `CostVolume`, esta tabla tiene `M` filas y dos columnas `AnnualLeasing` y `VolumeServices`, en esta tabla `AnnualLeasing` representa el costo anual de mantenimiento de las instalaciones para cada instalación, su definición es f_i; y `VolumeServices` representa la capacidad de almacenamiento de las instalaciones para cada instalación, su representación es M_i. Y por último, recibe una tabla llamada `AnnualDemand` que tiene `N` filas y una columna llamada `Demand`, esta tabla establece la demanda anual de cada cliente, su defición es d_i. En el campo `Variables` se crea una lista llamada `EstablishedLocation`, que para cada valor es 1 si se va a establecer una instalación o 0 en el caso contrario, su definición es y_i, también se crea una matriz llamada `AmountServiced`, que representa la cantidad de producto que reparte la instalación i al cliente j, su definición es x_{ij}. En el campo `Goals` se crea una función objetivo cuyo objetivo es minimizar el costo total, es decir, el costo de mantenimiento de las instalaciones más el costo de llevar el producto de una instalación a un cliente. Y por último, en el campo `Constraints` se definen una serie de restricciones para que el cliente reciba todo el producto que demanda, para que la cantidad de producto que almacena una instalación para un cliente no sea mayor que su capacidad, para que una instalación no almacena más producto del que el cliente demanda y para que el producto que almacena una instalación no sea cero.

Figura 5.8: Capacited facility location problem en Othimi

5.7. Vídeo

Como se mencionó anteriormente en el siguiente vídeo se muestra como crear un problema desde cero y una instancia para ese problema utilizando Othimi: `https://youtu.be/8WPFG6zVvak`

Capítulo 6

Conclusiones y líneas futuras

Durante este Trabajo de Fin de Grado se ha realizado una puesta en marcha y análisis de Prodef, se analizó tanto el Back-End como Front-End de la herramienta, centrándose sobre todo en el apartado de la interfaz gráfica que permite el desarrollo de problemas y algoritmos sin tener que programar ni entrar en complejas configuraciones. De este análisis se concluyó que para mejorar el Front-End era mucho más sencillo comenzarlo desde cero utilizando las mejores tecnologías posibles, además se dedicaría mucho esfuerzo y tiempo a mejorar la interfaz gráfica de Blockly, ya que esta interfaz es lo que hace que Prodef sea una herramienta única, y por consiguiente también se decidió crear desde cero el Back-End, utilizando las mejores tecnologías posibles. Durante el desarrollo del TFG se creó una nueva solución web llamada Othimi, esta solución aunque no está terminada, es una base sólida para el futuro desarrollo de la herramienta, además se llevó a la librería de Blockly a un segundo nivel consiguiendo grandes mejoras en la interfaz gráfica de modelados de problemas. Para concluir con el TFG se modelaron diferentes problemas y se creó un videotutorial para ayudar a nuevos usuarios a crear problemas e instancias.

6.1. Conclusiones

Como conclusión, es destacable que la nueva herramienta desarrollada en este TFG acerca a Othimi a una versión final, ya que la apariencia de la página es mucho más profesional que en el antiguo Prodef, este aspecto es muy importante, ya que no basta con que la herramienta sea útil para que los usuarios la utilicen, también es importante que los usuarios quieran utilizarla y que no se frustren durante el proceso. Sin embargo, no ha sido posible terminarla durante este TFG, ya que el apartado de Blockly requirió mucho más tiempo del esperado, además un aspecto al que no se le ha dedicado mucho esfuerzo por falta de tiempo es a la accesibilidad, si bien el objetivo de ese TFG era mejorar la usabilidad, algo que se ha conseguido con creces, la accesibilidad es igual de importante. Además, también se podría mejorar el control de errores del Back-End, aunque estos errores se controlen, hubiera estado bien realizar un control más sólido y más específico.

6.2. Líneas de trabajo futuro

Como la solución web se ha hecho desde cero, en lo que respecta al apartado web hay muchos aspectos que se pueden trabajar en futuros TFGs:

- Terminar el apartado de algoritmos, ejecuciones e historial. Estos apartados son los que quedarían implementar para poder sacar una versión estable de Othimi de forma oficial.
- Creación de un panel de administración. Actualmente, se hace uso de un usuario para crear los problemas e instancias de ejemplos, lo ideal sería crear un panel de administración al cual solo pudiesen acceder los usuarios administradores. Desde este panel se podrían crear tutoriales, contenido de ejemplo, etc.
- Crear un tutorial dentro de la web para nuevos usuarios. Cuando un usuario se autentifique por primera vez, la web podría guiarlo por sus diferentes apartados, consiguiendo que en el tutorial cree un problema, una instancia, un algoritmo, una ejecución y que revise la ejecución en el historial.
- Crear un formulario para sugerencias. Es muy importante la retroalimentación que un usuario puede dar de la web, ya que permite el descubrimiento de bugs, errores o simplemente aportar nuevas ideas para futuros desarrollos. Sería interesante que estas sugerencias se pudiesen ver en el panel de administración.

También hay algunos aspectos fuera del apartado web que hay que trabajar en futuros TFGs:

- Modificar los resolutores para que puedan funcionar con el nuevo formato de instancias y de problemas. Los modelos del Back-End han recibido cambios al punto de que el resolutor no admite estos nuevos modelos, es por esto que es necesario modificar el resolutor para que funcione para Othimi.
- Modificar el resolutor para que se puedan resolver más de una ejecución a la vez. Cuando se estudió Prodef se descubrió que el resolutor solo puede resolver una ejecución a la vez, esto en una página web que está pensada para resolver las ejecuciones de múltiples usuarios es un error importante. Además, esto es un problema de seguridad, ya que se puede hacer un ataque de denegación de servicio (DDOS), simplemente ejecutando un script que envíe cada 20 segundos una ejecución de valores muy grandes al resolutor. Este ataque sería difícil de bloquear, ya que no es necesario un envío descomunal de peticiones.
- Modificar el resolutor para que cuando una ejecución sea resuelta, esta le llegue al usuario. En el antiguo Prodef una ejecución era enviado al resolutor y hasta que el usuario no lo volvía a solicitar no se guardaba en el Back-End. Esta solución no está bien planteada, lo lógico sería que el resolutor enviase el problema resuelto al Back-End, una vez lo resuelva y este a su vez se lo envíe al usuario en el caso de que esté esperando por la ejecución. Esto se podría conseguir con comunicación `websockets`, además Apolo implementa un sistema que permite realizar una petición cuando una variable cambie de valor, haciendo uso de `websockets`.

Capítulo 7

Summary and Conclusions

During this Final Degree Project we have carried out a start-up and analysis of Prodef, we analyzed both the Back-End and Front-End of the tool, focusing mainly on the graphical interface section that allows the development of problems and algorithms without having to program or enter into complex configurations. From this analysis it was concluded that in order to improve the Front-End it was much simpler to start from scratch using the best possible technologies, also a lot of effort and time would be devoted to improve the graphical interface of Blockly, since this interface is what makes Prodef a unique tool, and therefore it was also decided to create from scratch the Back-End, using the best possible technologies. During the development of the TFG a new web solution called Othimi was created, this solution although not finished, is a solid base for the future development of the tool, also the Blockly library was taken to a second level achieving great improvements in the graphical interface of problem modeling. To conclude with the TFG, different problems were modeled and a video tutorial was created to help new users to create problems and instances.

7.1. Conclusions

As a conclusion, it is remarkable that the new tool developed in this TFG brings Othimi closer to a final version, since the appearance of the page is much more professional than in the old Prodef, this aspect is very important, since it is not enough that the tool is useful for users to use it, it is also important that users want to use it and that they do not get frustrated during the process. However, it has not been possible to finish it during this TFG, since the Blockly section required much more time than expected, also an aspect that has not been devoted much effort due to lack of time is accessibility, although the objective of this TFG was to improve usability, something that has been achieved by far, accessibility is just as important. In addition, the Back-End error control could also be improved, although these errors are controlled, it would have been nice to make a more solid and more specific control.

7.2. Future lines of work

As the web solution has been made from scratch, as far as the web section is concerned there are many aspects that can be worked on in future TFGs:

- Finish the algorithms, executions, and history section. These sections are the ones

that remain to be implemented in order to officially release a stable version of Othimi.

- Creation of an administration panel. Currently, a user is used to create the problems and instances of examples, the ideal would be to create an administration panel to which only the administrator users could access. From this panel it would be possible to create tutorials, example content...
- Create a tutorial within the website for new users. When a user authenticates for the first time, the web could guide him through its different sections, getting him in the tutorial to create a problem, an instance, an algorithm, an execution and to review the execution in the history.
- Create a form for suggestions. It is very important the feedback that a user can give of the web, since it allows the discovery of bugs, errors or simply to contribute new ideas for future developments. It would be interesting if these suggestions could be seen in the administration panel.

There are also some aspects outside the web section that need to be worked on in future TFGs:

- Modify the solvers to work with the new instance and problem format. The Back-End models have received changes to the point that the resolver does not support these new models, this is why it is necessary to modify the resolver to work for Othimi.
- Modify the resolver so that more than one execution can be resolved at a time. When Prodef was studied it was discovered that the resolver can only resolve one execution at a time, this on a web page that is intended to resolve multiple user executions is a major bug. In addition, this is a security problem, since a denial of service (DDOS) attack can be made by simply running a script that sends every 20 seconds an execution of very large values to the resolver. This attack would be difficult to block, since it is not necessary to send a huge number of requests.
- Modify the resolver so that when an execution is resolved, it is sent to the user. In the old Prodef an execution was sent to the solver and until the user did not request it again it was not saved in the Back-End. This solution is not well-thought-out, the logical thing would be that the solver would send the solved problem to the Back-End, once it is solved, and this in turn would send it to the user in case he/she is waiting for the execution. This could be achieved with `websockets` communication, besides Apolo implements a system that allows to make a request when a variable changes value, making use of `websockets`.

Capítulo 8

Presupuesto

En este capítulo se recoge el presupuesto del trabajo realizado. El coste de este trabajo proviene del tiempo que se ha empleado es su desarrollo. Para estimar el precio de la hora de trabajo se han visitado varias plataformas de comparación de sueldos, finalmente se ha elegido un precio de 15€ la hora.

Nombre	**Horas**	**Coste (€)**
Despliegue e investigación del funcionamiento de Prodef	20	300
Análisis de Prodef	10	150
Estudio de Blockly	40	600
Estudio de las tecnologías Front-End	20	300
Desarrollo del Front-End	100	1500
Estudio de las tecnologías Back-End	20	300
Desarrollo del Back-End	60	900
Creación de tutoriales	30	450
Total	300	4500

Tabla 8.1: Presupuesto

Apéndice A

Mutadores de Blockly

A.1. Mutador del expressionBlock

```
import Blockly, { Block } from "blockly";
import operations from "./blocksOperationMutator/groupingOfOperationblock";
import operation from "./blocksOperationMutator/operationBlock";

const operation_mutator = {
  name: "operation_mutator",
  mixinObj: {
    saveExtraState: function () {
      return {
        "itemCount": (this as any).itemCount_,
        "lastItemCount": (this as any).lastItemCount_
      };
    },
    loadExtraState: function (state: { itemCount: number, lastItemCount: number }) {
      (this as any).itemCount_ = state["itemCount"];
      (this as any).lastItemCount_ = state["lastItemCount"];

      this.updateShape_(true);
    },
    updateShape_: function (move: boolean = false) {
      if (move) {
        for (let i: number = 0; i < (this as any).itemCount_; i++)
          (this as unknown as Block).appendValueInput("OPERATION" + i)
            .setCheck(
              ["expression",
                "sum_mult_definition",
                "number",
                "getter_variable"])
            .appendField(new Blockly.FieldDropdown([
              ["+", "+"],
              ["-", "-"],
              ["*", "*"],
              ["/", "/"],
              ["^", "^"],
            ]), i.toString());
      } else {
        if ((this as any).lastItemCount_ > (this as any).itemCount_) {
          for (let i: number = (this as any).lastItemCount_ - 1; i >= (this as any).
    itemCount_; i--)
            (this as unknown as Block).removeInput("OPERATION" + i, true);
```

```
      }
      else if ((this as any).lastItemCount_ < (this as any).itemCount_) {
        for (let i: number = (this as any).lastItemCount_; i < (this as any).itemCount_
; i++)
          (this as unknown as Block).appendValueInput("OPERATION" + i)
            .setCheck(
              ["expression",
                "sum_mult_definition",
                "number",
                "getter_variable"])
            .appendField(new Blockly.FieldDropdown([
              ["+", "+"],
              ["-", "-"],
              ["*", "*"],
              ["/", "/"],
              ["^", "^"],
            ]), i.toString());
      }
    }

    (this as any).lastItemCount_ = (this as any).itemCount_;
  },
  decompose: function (workspace: any) {

    const topBlock: Blockly.Block = Blockly.serialization.blocks.
      append({ "type": operations.type }, workspace);

    let connection = (topBlock as any).getInput("operations").connection;
    for (let i: number = 0; i < (this as any).itemCount_; i++) {
      let itemBlock: Blockly.Block = Blockly.serialization.blocks.
        append({ "type": operation.type }, workspace);
      connection.connect(itemBlock.previousConnection);
      connection = itemBlock.nextConnection;
    }

    return topBlock;
  },
  compose: function (topBlock: Blockly.Block) {
    let itemBlock = topBlock.getInputTargetBlock("operations");
    let connections = [];

    while (itemBlock && !itemBlock.isInsertionMarker()) {
      connections.push((itemBlock as any).valueConnection_);
      itemBlock = itemBlock.nextConnection &&
        itemBlock.nextConnection.targetBlock();
    }

    for (let i: number = 0; i < (this as any).itemCount_; i++) {
      let connection = (this as any).getInput("OPERATION" + i).
        connection.targetConnection;
      if (connection && connections.indexOf(connection) == -1) {
        connection.disconnect();
      }
    }

    (this as any).lastItemCount_ = (this as any).itemCount_;
    (this as any).itemCount_ = connections.length;
    this.updateShape_();
```

```
97
98        for (let i: number = 0; i < (this as any).itemCount_; i++) {
99          Blockly.Mutator.reconnect(connections[i], this as any, "OPERATION" + i);
100       }
101     },
102   },
103   opt_helperFn: function () {
104     (this as any).lastItemCount_ = 0;
105     (this as any).itemCount_ = 0;
106     (this as any).updateShape_();
107   },
108   opt_blockList: [operation.type]
109 };
110
111 export default operation_mutator;
```

A.2. Mutador del variableBlock

```
1  import Blockly, { Block } from "blockly";
2  import relationShipBlocks from "./utils/relationshipBlocksClass";
3
4  enum typeOfBlock {
5    TypeNumber,
6    TypeList,
7    TypeMatrix
8  }
9
10 const type_of_variable_mutator = {
11   name: "type_of_variable_mutator",
12   mixinObj: {
13     saveExtraState: function () {
14       if ((this as unknown as Block).getFieldValue("type") === "list")
15         return {
16           "typeOfBlock": typeOfBlock.TypeList,
17           "permutationList": false
18         };
19       else if ((this as unknown as Block).getFieldValue("type") ===
20         "permutation_list")
21         return {
22           "typeOfBlock": typeOfBlock.TypeList,
23           "permutationList": true
24         };
25       else if ((this as unknown as Block).getFieldValue("type") ===
26         "matrix")
27         return {
28           "typeOfBlock": typeOfBlock.TypeMatrix,
29           "permutationList": false
30         };
31       else
32         return {
33           "typeOfBlock": typeOfBlock.TypeNumber,
34           "permutationList": false
35         };
36     },
37     loadExtraState: function (state: { typeOfBlock: string,
38       permutationList: boolean }) {
```

```
    (this as any).typeOfBlock_ = state["typeOfBlock"];
    (this as any).permutationList_ = state["permutationList"];
    this.updateShape_();
  },
  updateShape_: function () {
    const ColsExists = (this as unknown as Block).getInput("COLS");
    const RowsExists = (this as unknown as Block).getInput("ROWS");
    const IntegerReal = (this as unknown as Block).getInput("INTEGERREAL");

    if (!IntegerReal && !(this as any).permutationList_) {
      (this as unknown as Block)
        .appendDummyInput("INTEGERREAL")
        .appendField("||", "||")
        .appendField(new Blockly.FieldDropdown([
          ["Integer", "integers"],
          ["Real", "reals"],
        ]), "typeintegerReal")
        .appendField("in range", "in range")
        .appendField(new Blockly.FieldNumber(0), "typelowerBound")
        .appendField("to", "to")
        .appendField(new Blockly.FieldNumber(1), "typeupperBound");
    } else if (IntegerReal && (this as any).permutationList_) {
      (this as unknown as Block).removeInput("INTEGERREAL", true);
    }

    if (((this as any).typeOfBlock_ === typeOfBlock.TypeList ||
      (this as any).typeOfBlock_ === typeOfBlock.TypeMatrix) && !RowsExists) {
      (this as unknown as Block)
        .appendValueInput("ROWS")
        .setCheck(["number", "getter_variable", "expression"])
        .appendField("with", "with");
      (this as unknown as Block)
        .appendDummyInput("FINAL_TEXT");
      (this as unknown as Block).moveInputBefore("ROWS", "dummy2");
      (this as unknown as Block).moveInputBefore("FINAL_TEXT", "dummy2");
    }

    if ((this as any).typeOfBlock_ === typeOfBlock.TypeNumber) {
      (this as unknown as Block).removeInput("COLS", true);
      (this as unknown as Block).removeInput("ROWS", true);
      (this as unknown as Block).removeInput("FINAL_TEXT", true);
    } else if ((this as any).typeOfBlock_ === typeOfBlock.TypeList) {
      if (!RowsExists) {
        (this as unknown as Block).getInput("FINAL_TEXT")?.
          appendField("elements", "elements");
      } else if (ColsExists) {
        (this as unknown as Block).removeInput("COLS");
        (this as unknown as Block).getInput("FINAL_TEXT")?.
          removeField("cols");
        (this as unknown as Block).getInput("FINAL_TEXT")?.
          appendField("elements", "elements");
      }
    } else if ((this as any).typeOfBlock_ === typeOfBlock.TypeMatrix) {
      if (!RowsExists) {
        (this as unknown as Block).getInput("FINAL_TEXT")?.
          appendField("cols", "cols");
      }
      if (!ColsExists) {
```

```
          (this as unknown as Block)
            .appendValueInput("COLS")
            .setCheck(["number", "getter_variable", "expression"])
            .appendField("rows and with", "rows and with");
          (this as unknown as Block).moveInputBefore("COLS", "FINAL_TEXT");
        }
        if (RowsExists && !ColsExists) {
          (this as unknown as Block).getInput("FINAL_TEXT")?.
            removeField("elements");
          (this as unknown as Block).getInput("FINAL_TEXT")?.
            appendField("cols", "cols");
        }
      }
    },
  },
  opt_helperFn: function () {
    (this as any).getField("CustomTextInputSingle").
      setValidator((option: string) => {
        if ((this as unknown as Block).getStyleName() !== "auto_#000000") {
          let state: string = (this as any).getFieldValue("type");
          relationShipBlocks.setVariable(
            (this as unknown as Block).id,
            option,
            state,
            (this as unknown as Block).workspace.id
          );

          if (state === "list") {
            (this as any).typeOfBlock_ = typeOfBlock.TypeList;
            (this as any).permutationList_ = false;
          }

          else if (state  === "permutation_list") {
            (this as any).typeOfBlock_ = typeOfBlock.TypeList;
            (this as any).permutationList_ = true;
          }

          else if (state === "matrix") {
            (this as any).typeOfBlock_ = typeOfBlock.TypeMatrix;
            (this as any).permutationList_ = false;
          }

          else {
            (this as any).typeOfBlock_ = typeOfBlock.TypeNumber;
            (this as any).permutationList_ = false;
          }

          (this as any).updateShape_();
        }
    });

    (this as any).getField("type").setValidator((option: string) => {
      if ((this as unknown as Block).getStyleName() !== "auto_#000000") {
        relationShipBlocks.setVariable(
          (this as unknown as Block).id,
          (this as any).getFieldValue("CustomTextInputSingle"),
          option,
          (this as unknown as Block).workspace.id
```

```
        );

        if (option === "list") {
          (this as any).typeOfBlock_ = typeOfBlock.TypeList;
          (this as any).permutationList_ = false;
        }

        else if (option === "permutation_list") {
          (this as any).typeOfBlock_ = typeOfBlock.TypeList;
          (this as any).permutationList_ = true;
        }

        else if (option === "matrix") {
          (this as any).typeOfBlock_ = typeOfBlock.TypeMatrix;
          (this as any).permutationList_ = false;
        }

        else {
          (this as any).typeOfBlock_ = typeOfBlock.TypeNumber;
          (this as any).permutationList_ = false;
        }

        (this as any).updateShape_();
      }
    });
  },
  opt_blockList: undefined
};

export default type_of_variable_mutator;
```

A.3. Mutador del getterBlock

```
import Blockly, { Block, MenuOption } from "blockly";
import GettersBlocks from "./utils/gettersBlocks";
import RelationShipBlocks from "./utils/relationshipBlocksClass";

enum typeOfBlock {
  TypeNumber,
  TypeList,
  TypeMatrix
}

const getVariable_mutator = {
  name: "getVariable_mutator",
  mixinObj: {
    saveExtraState: function () {
      if ((this as unknown as Block).getInput("COLS"))
        return {
          "typeOfBlock": typeOfBlock.TypeMatrix,
          "id": (this as unknown as Block).getFieldValue("getterVariableName"),
          "lastArray": GettersBlocks.getArray()
        };
      else if ((this as unknown as Block).getInput("ROWS"))
        return {
          "typeOfBlock": typeOfBlock.TypeList,
```

```
        "id": (this as unknown as Block).getFieldValue("getterVariableName"),
        "lastArray": GettersBlocks.getArray()
      };
    else
      return {
        "typeOfBlock": typeOfBlock.TypeNumber,
        "id": (this as unknown as Block).getFieldValue("getterVariableName"),
        "lastArray": GettersBlocks.getArray()
      };
  },
  loadExtraState: function (state: { typeOfBlock: string, id: string, lastArray: string[][] }) {
    if (state["id"] !== "Select any" && (this as unknown as Block).getStyleName() !== "auto_#000000")
      RelationShipBlocks.setGetterVariable(state["id"], this as unknown as Block, false);

    (this as any).typeOfBlock_ = state["typeOfBlock"];
    this.updateShape_(state["lastArray"]);
  },
  updateShape_: function (lastArray: string[][] = []) {

    if (lastArray.length !== 0) {
      (this as unknown as Block).inputList[0].removeField("getterVariableName");
      (this as unknown as Block).inputList[0].appendField(
        new Blockly.FieldDropdown(lastArray as MenuOption[]), "getterVariableName");
      (this as any).getField("getterVariableName").setValidator((option: string) => {
        if ((this as unknown as Block).getStyleName() !== "auto_#000000")
          RelationShipBlocks.setGetterVariable(option, (this as unknown as Block));
      });
    }

    const ColsExists = (this as unknown as Block).getInput("COLS");
    const RowsExists = (this as unknown as Block).getInput("ROWS");

    if (((this as any).typeOfBlock_ === typeOfBlock.TypeList ||
      (this as any).typeOfBlock_ === typeOfBlock.TypeMatrix) && !RowsExists) {
      (this as unknown as Block)
        .appendValueInput("ROWS")
        .setCheck(["number", "getter_variable", "expression"]);
    }

    if ((this as any).typeOfBlock_ === typeOfBlock.TypeNumber) {
      (this as unknown as Block).removeInput("COLS", true);
      (this as unknown as Block).removeInput("ROWS", true);
    } else if ((this as any).typeOfBlock_ === typeOfBlock.TypeList) {
      if (!RowsExists) {
        (this as unknown as Block).getInput("ROWS")?.
          appendField("element number", "element number");
      } else if (ColsExists) {
        (this as unknown as Block).removeInput("COLS");
        (this as unknown as Block).getInput("ROWS")?.
          removeField("row number");
        (this as unknown as Block).getInput("ROWS")?.
          appendField("element number", "element number");
      }
    } else if ((this as any).typeOfBlock_ === typeOfBlock.TypeMatrix) {
      if (!RowsExists) {
```

```
          (this as unknown as Block).getInput("ROWS")?.
            appendField("row number", "row number");
        }
        if (!ColsExists) {
          (this as unknown as Block)
            .appendValueInput("COLS")
            .setCheck(["number", "getter_variable", "expression"])
            .appendField("and column number", "and column number");
        }
        if (RowsExists && !ColsExists) {
          (this as unknown as Block).getInput("ROWS")?.
            removeField("element number");
          (this as unknown as Block).getInput("ROWS")?.
            appendField("row number", "row number");
        }
      }
    },
  },
  opt_helperFn: undefined,
  opt_blockList: undefined
};

export default getVariable_mutator;
```

A.4. Mutador del tableBlock

```
import { Block } from "blockly";
import relationShipBlocks from "./utils/relationshipBlocksClass";

enum typeOfBlock {
  TypeNumber,
  TypeList,
  TypeMatrix
}

const table_mutator = {
  name: "table_mutator",
  mixinObj: {
    saveExtraState: function () {
      return {
        "typeOfBlock": typeOfBlock.TypeMatrix
      };
    },
    loadExtraState: function (state: { typeOfBlock: string }) {
      (this as any).typeOfBlock_ = state["typeOfBlock"];
    },
  },
  opt_helperFn: function () {
    (this as any).getField("CustomTextInputTableSingle").
      setValidator((option: string) => {
        if ((this as unknown as Block).getStyleName() !== "auto_#000000") {
          relationShipBlocks.setVariable(
            (this as unknown as Block).id,
            option,
            "matrix",
            (this as unknown as Block).workspace.id
```

```
          );
        }
      });
  },
  opt_blockList: undefined
};

export default table_mutator;
```

Apéndice B

Clases estáticas

B.1. Clase estática RelationShipBlocks

```
import Blockly, { Block } from "blockly";

enum typeOfBlock {
  TypeNumber,
  TypeList,
  TypeMatrix
}

class RelationShipBlocks {
  static variables: { [key: string]: {variableID: string,
    type: string} } = {};
  static getters: {
    [key: string]: { variableID: string, block: any };
  } = {};

  static workspace: Blockly.WorkspaceSvg;



  static setWorkSpace(workspace: Blockly.WorkspaceSvg) {
    this.workspace = workspace;
  }



  static setVariable(BlockID: string, variableID: string, type:
    string, workspaceID: string) {
    if (workspaceID === this.workspace.id) {

      if (this.variables[BlockID]) {
        this.variables[BlockID].type = type;
        this.variables[BlockID].variableID = variableID;
      } else {
        this.variables[BlockID] = {
          type,
          variableID
        };
      }
      for (let blockID in this.getters) {

```

```
      if (this.getters[blockID].variableID === variableID) {
        this.getters[blockID].block.typeOfBlock_ =
          this.typeFunction(type);
        this.getters[blockID].block?.updateShape_();
      }
    }
  }
}

private static typeFunction(type: string) {
  if (type === "list" || type === "permutation_list")
    return typeOfBlock.TypeList;
  else if (type === "matrix")
    return typeOfBlock.TypeMatrix;
  else
    return typeOfBlock.TypeNumber;
}



static setGetterVariable(
  variableID: string,
  block: Block,
  callUpdateShape: boolean = true
) {
  if (block.workspace.id === this.workspace.id) {
    if (this.getters[block.id]) {
      this.getters[block.id].variableID = variableID;
    } else {
      this.getters[block.id] = {
        variableID: variableID,
        block: block,
      };
    }
    if (callUpdateShape) {
      let type = ""
      for (let blockID in this.variables)
        if (this.variables[blockID].variableID === variableID)
          type = this.variables[blockID].type;


      this.getters[block.id].block.typeOfBlock_ =
        this.typeFunction(type);
      this.getters[block.id].block?.updateShape_();
    }
  }
}

static deleletedBlocksUpdate(blockIDS: string[]) {
  for (let id of blockIDS) {
    if (this.getters[id]) {
      delete this.getters[id];
    }
  }
}

static cleanGetters(blockID: string) {
  delete this.getters[blockID];
```

```
  }

  static cleanVariables(blockID: string) {
    if (this.variables[blockID])
      delete this.variables[blockID];
  }

  static cleanAll() {
    this.variables = {};
    this.getters = {};
  }
}

export default RelationShipBlocks;
```

B.2. Clase estática GettersBlocks

```
import Blockly, { Block, MenuOption } from "blockly";
import RelationShipBlocks from "./relationshipBlocksClass";

class GettersBlocks {
  static gettersBlocks: { [key: string]: any } = {}
  static lastArray: string[][] = [["Select any", "Select any"]];



  private static changeField(key: string) {
    const value: string = this.gettersBlocks[key].
      getField("getterVariableName")?.getValue();
    this.gettersBlocks[key].inputList[0].
      removeField("getterVariableName");
    this.gettersBlocks[key].inputList[0].
      appendField(new Blockly.FieldDropdown(this.lastArray as MenuOption[]),
        "getterVariableName");
    if (this.searchInArray(value))
      this.gettersBlocks[key].getField("getterVariableName")?.
        setValue(value);
    else
      this.gettersBlocks[key].typeOfBlock_ = 0;
    this.gettersBlocks[key].getField("getterVariableName").
      setValidator((option: string) => {
        RelationShipBlocks.setGetterVariable(option, this.gettersBlocks[key]);
      });
    this.gettersBlocks[key].updateShape_();
  }



  private static searchInArray(value: string) {
    for (let i = 0; i < this.lastArray.length; i++)
      if (this.lastArray[i][0] === value && this.lastArray[i][0] !== "Select any")
        return true

    return false
  }

```

```
40
41
42    static getArray() {
43      return this.lastArray;
44    }
45
46
47
48    static modifyDropDown(array: string[][]) {
49      this.lastArray = array;
50      for (let blockID in this.gettersBlocks) {
51        if (this.gettersBlocks[blockID]) {
52          this.changeField(blockID);
53        }
54      }
55    }
56
57
58
59    static setGetterBlock(blockID: string, block: Block) {
60      this.gettersBlocks[blockID] = block;
61    }
62
63
64
65    static cleanGetterBlock(blockID: string) {
66      delete this.gettersBlocks[blockID];
67      RelationShipBlocks.cleanGetters(blockID);
68    }
69
70
71
72    static cleanAll() {
73      this.gettersBlocks = {};
74      this.lastArray = [["Select any", "Select any"]];
75    }
76  }
77
78  export default GettersBlocks
```

B.3. Clase estática InputsBlocks

```
1  import GettersBlocks from "./gettersBlocks";
2  import RelationShipBlocks from "./relationshipBlocksClass";
3  import Blockly from "blockly";
4
5  class InputsBlocks {
6    static inputBlocks: { [key: string]: { fields: string[],
7      values: string[] } } = {}
8    static workspace: Blockly.WorkspaceSvg;
9    static errorMessage: string = "";
10
11
12
13   static setWorkSpace(workspace: Blockly.WorkspaceSvg) {
14     this.workspace = workspace;
```

```
}



static sendInput() {
  this.errorMessage = "";
  let auxArray: string[] = [];
  let array: string[][] = [["Select any", "Select any"]];

  for (let blockID in this.inputBlocks) {
    for (let i = 0; i < this.inputBlocks[blockID].values.length; i++)
      if (this.inputBlocks[blockID].values[i] !== "Change me")
        auxArray.push(this.inputBlocks[blockID].values[i]);
  }

  let elemets = new Set(auxArray);
  for (let elemet of elemets) {
    array.push([elemet, elemet]);
  }

  GettersBlocks.modifyDropDown(array);
}



private static nameAdmin(blockID: string, value: string,
  field: string, mode: "CREATE" | "CHANGE") {

  if (value === "Change me") return
  const single: boolean = /Single$/.test(field);
  let errorChange: boolean = false;

  for (let blockId in this.inputBlocks) {
    for (let i = 0; i < this.inputBlocks[blockId].values.length; i++) {
      if (blockId !== blockID) {
        if (single && this.inputBlocks[blockId].values[i] === value)
          errorChange = true;
        else if (!single && this.inputBlocks[blockId].values[i] === value &&
          this.inputBlocks[blockId].fields[i] !== field)
          if ((this.inputBlocks[blockId].fields[i] !==
            "CustomTextInputRowCommon" || field !== "CustomTextColumnCommon") &&
            (this.inputBlocks[blockId].fields[i] !== "CustomTextColumnCommon" ||
            field !== "CustomTextInputRowCommon"))
            errorChange = true;

        if (errorChange) break;
      }
    }
    if (errorChange) break;
  }

  if (errorChange) {
    const block = this.workspace.getBlockById(blockID);
    for (let i = 0; i < this.inputBlocks[blockID].fields.length; i++) {
      if (this.inputBlocks[blockID].fields[i] === field) {
        if (mode === "CHANGE") {
          this.errorMessage = "Some variable names are already in use, \
          they have been assigned the last name, see the documentation \
```

```
            for more information on this section.";
            (block as Blockly.Block).getField(field)?.setValue(this.
              inputBlocks[blockID].values[i]);
          }
          else {
            this.errorMessage = "Some variable names are already in use, \
            they have been assigned the default name \"Change me\", see \
            the documentation for more information on this section.";
            (block as Blockly.Block).getField(field)?.setValue("Change me");
            this.inputBlocks[blockID].values[i] = "Change me";
          }
        }
      }
    } else {
      const index: number = this.inputBlocks[blockID].fields.indexOf(field);
      this.inputBlocks[blockID].values[index] = value;
    }
  }



  static setInput(blockID: string, value: string, field: string) {
    this.nameAdmin(blockID, value, field, "CHANGE");
  }



  static createInput(blockID: string, values: string[], fields: string[]) {
    this.inputBlocks[blockID] = { fields, values };
    for (let i = 0; i < fields.length; i++)
      this.nameAdmin(blockID, values[i], fields[i], "CREATE");
  }



  static cleanInput(blockID: string, values: string[]) {
    this.inputBlocks[blockID].values = values;
    RelationShipBlocks.cleanVariables(blockID);
  }



  static cleanAll() {
    this.inputBlocks = {};
    this.errorMessage = "";
  }
}

export default InputsBlocks
```

Apéndice C

Componente Problem

```
import { Row, Col, Button, Container, InputGroup, Form } from "react-bootstrap";
import Blockly from "blockly";
import { useEffect, useState } from "react";
import { ITheme } from "blockly/core/theme";
import toolbox from "../../../../blockly/toolBox";
import { CreateRootBlock, SaveProblemJSON } from "../../../../blockly/adminRootBlock";
import InputBlocks from "../../../../blockly/problem/mutators/utils/inputBlocksClass";
import GettersBlocks from "../../../../blockly/problem/mutators/utils/gettersBlocks";
import { ToastContainer, toast } from "react-toastify";
import ErrorReporter from "../../../common/ErrorReporter";
import StyleToast from "../../../../css/Toast.module.css";
import StyleWorkZone from "../../../../css/WorkZone.module.css";
import StyleProblem from "../../../../css/Problem.module.css";
import { PATH_PROBLEM, POST_PROBLEM } from "../../../../graphql/problemQL";
import "react-toastify/dist/ReactToastify.css";
import { javascriptGenerator } from "blockly/javascript";
import errorsReporter from "../../../../blockly/errorsReporter";
import Separator from "../../../common/Separator";
import { useMutation } from "@apollo/client";
import { useAppDispatch, useAppSelector } from "../../../../app/hooks";
import { logout } from "../../../../features/user/userSlice";
import RelationShipBlocks from "../../../../blockly/problem/mutators/utils/
    relationshipBlocksClass";

let workspace: Blockly.WorkspaceSvg;

export default function Problem({ json, name, family,
  viewMode, mode, setMode, idProblem, setIdProblem }:
  { json: string, name: string, family: string, viewMode:
    boolean, mode: string, setMode:
    React.Dispatch<React.SetStateAction<string>>,
    idProblem: string, setIdProblem:
    React.Dispatch<React.SetStateAction<string>> }) {

  const user = useAppSelector((state) => state.userState.userData);
  const dispatch = useAppDispatch();

  const notify = (message: string) => toast.error(message, {
    position: toast.POSITION.TOP_CENTER,
    className: `${StyleToast.toastMessage}`
  });

```

```
  const [errors, setErrors] = useState([""]);
  const [problemName, setProblemName] = useState(name);
  const [problemType, setProblemType] = useState(family);
  const [rootBlock, setRootBlock] = useState("");



  const [postProblem, post] = useMutation(POST_PROBLEM,
    { errorPolicy: "all" });
  const errorPost = post.error;
  const dataPost = post.data;
  const [apolloManagerPost, setApolloManagerPost] =
    useState(false);
  useEffect(() => {
    if (apolloManagerPost && errorPost) {
      if (errorPost.message === "Response not successful: \
        Received status code 401")
        dispatch(logout())
      else {
        setErrors([errorPost.message])
        setApolloManagerPost(false);
      }
    } else if (apolloManagerPost && dataPost) {
      setIdProblem("");
      setMode("LIST");
    }
  });



  const [pathProblem, path] = useMutation(PATH_PROBLEM,
    { errorPolicy: "all" });
  const errorPath = path.error;
  const dataPath = path.data;
  const [apolloManagerPath, setApolloManagerPath] =
    useState(false);
  useEffect(() => {
    if (apolloManagerPath && errorPath) {
      if (errorPath.message === "Response not successful: \
        Received status code 401")
        dispatch(logout())
      else {
        setErrors([errorPath.message])
        setApolloManagerPath(false);
      }
    } else if (apolloManagerPath && dataPath) {
      setIdProblem("");
      setMode("LIST");
    }
  });



  function recursiveCountTablesAndParameters(json: any,
      parametersCount: number, filesCount: number):
      { parametersCount: number, filesCount: number } {
    if (json.block.type === "table_definition" ||
      json.block.type === "table_definition_matrix") filesCount++;
```

```
    if (json.block.type === "parameter_definition") parametersCount++;

    if (!json.block?.next) return { parametersCount, filesCount };
    return recursiveCountTablesAndParameters(json.block.next,
      parametersCount, filesCount);
  }



  function countTablesAndParameters() {
    const block = workspace.getBlockById(rootBlock);
    const JSON = Blockly.serialization.blocks.save(block as
      Blockly.Block);

    let parametersCount: number = 0;
    let filesCount: number = 0;
    if ((JSON as any)?.inputs?.root_input_data) {
      return recursiveCountTablesAndParameters((JSON as any).
        inputs.root_input_data, parametersCount, filesCount);
    } else return { parametersCount, filesCount };
  }



  function blocksExploration(json: any, mode: "SAVE" | "CLEAN") {
    if (json.inputs) {
      const jsonInputs: [string, any][] = Object.entries(json.inputs);
      for (let i = 0; i < jsonInputs.length; i++)
        blocksExploration(jsonInputs[i][1].block, mode);
    }

    const type: string = json.type;

    if (type === "parameter_definition" || type === "table_definition_matrix" ||
      type === "column_definition" || type === "table_definition" ||
      type === "variable_definition" ||
      type === "sum_mult_definition" || type === "for_all_input") {
      const fields: [string, string][] = Object.entries(json.fields);
      let values: string[] = [];
      if (mode === "SAVE") {
        let names: string[] = [];

        for (let i = 0; i < fields.length; i++) {
          if (!/^type/.test(fields[i][0])) {
            names.push(fields[i][0]);
            values.push(fields[i][1]);
          }
        }
        InputBlocks.createInput(json.id as string, values, names);
      } else {
        for (let i = 0; i < fields.length; i++)
          values.push("Change me");

        InputBlocks.cleanInput(json.id as string, values);
      }
    }

    else if (type === "getter_variable") {
```

```
    if (mode === "SAVE") {
      const block = workspace.getBlockById(json.id);
      GettersBlocks.setGetterBlock(json.id as string,
        (block as Blockly.Block));
    } else {
      GettersBlocks.cleanGetterBlock(json.id as string);
    }
  }

  if (json.next) blocksExploration(json.next.block, mode);
}



function changeSVGZoomTrashCan() {
  const elementsZoom = document.getElementsByClassName("blocklyZoom");
  for (let item of elementsZoom) {
    item.children[1].setAttribute("href", "/sprites.png");
    item.children[1].setAttribute("xlink:href", "/sprites.png");
  }

  const elementsTrashCan = document.
    getElementsByClassName("blocklyTrash");

  elementsTrashCan[0].children[1].
    setAttribute("href", "/sprites.png");
  elementsTrashCan[0].children[1].
    setAttribute("xlink:href", "/sprites.png");

  elementsTrashCan[0].children[3].
    setAttribute("href", "/sprites.png");
  elementsTrashCan[0].children[3].
    setAttribute("xlink:href", "/sprites.png");
}



function variableParameterEvent(event: any) {
  if (event.type === Blockly.Events.BLOCK_CHANGE &&
    /^CustomText/.test(event.name as string)) {
    if ((event.newValue as string) === "Select any" ||
      (event.newValue as string) === "") {

      const block = workspace.getBlockById(event.blockId);
      (block as Blockly.Block).getField(event.name)?.
        setValue("Change me");
      InputBlocks.setInput(event.blockId as string,
        "Change me", event.name);
      InputBlocks.sendInput();
    }
    else {
      InputBlocks.setInput(event.blockId as string,
        event.newValue as string, event.name);
      if (InputBlocks.errorMessage !== "")
        notify(InputBlocks.errorMessage);
      InputBlocks.sendInput();
    }
  }
```

```
  if (event.type === Blockly.Events.BLOCK_CREATE) {
    blocksExploration(event.json, "SAVE");
    if (InputBlocks.errorMessage !== "")
      notify(InputBlocks.errorMessage);
    InputBlocks.sendInput();
  }

  if (event.type === Blockly.Events.BLOCK_DELETE) {
    blocksExploration(event.oldJson, "CLEAN");
    InputBlocks.sendInput();
  }
}



function sendProblem() {
  errorsReporter.clean();
  let code = javascriptGenerator.workspaceToCode(workspace);
  let error: string[] = [];

  const auxCode = JSON.parse(code);
  if (auxCode["variable"].length === 0)
    error.push("The Variables input of the problem cannot be empty .");
  if (auxCode["goal"].length === 0)
    error.push("The Goals input of the problem cannot be empty .");
  if (!/^(?=.*[0-9a-zA-Z])[a-zA-Z0-9\s]+$/.test(problemName))
    error.push("The problem name cannot be empty.");
  if (problemType === "-1")
    error.push("You must select a type of problem.");

  error = [...error, ...errorsReporter.report()];
  setErrors([...error]);
  if (error.length === 0) {
    const { parametersCount, filesCount } = countTablesAndParameters();

    if (mode === "CREATE" || mode === "COPY") {
      setApolloManagerPost(true);
      postProblem({
        variables: {
          input: {
            owner: user.id,
            name: problemName,
            family: problemType,
            blocklyJSON: JSON.stringify(SaveProblemJSON(rootBlock,
              workspace)),
            problemObject: code,
            parametersCount,
            filesCount
          }
        }
      });
    } else if (mode === "EDIT") {
      setApolloManagerPath(true);

      pathProblem({
        variables: {
          idProblem,
```

```
            input: {
              owner: user.id,
              name: problemName,
              family: problemType,
              blocklyJSON: JSON.stringify(SaveProblemJSON(rootBlock,
                workspace)),
              problemObject: code,
              parametersCount,
              filesCount
            }
          }
        });
    }
  }
}



function ProblemSelect() {
  return (
    <Form.Select
      disabled={mode === "VIEW"}
      className="mb-3"
      onChange={e => {
        setProblemType(e.target.value);
      }}
      value={problemType}
    >
      <option value="-1">Select any problem type</option>
      <option value="Bin packing">Bin packing</option>
      <option value="Knapsack">Knapsack</option>
      <option value="Location">Location</option>
      <option value="Portfolio optimisation">Portfolio optimisation</option>
      <option value="Scheduling">Scheduling</option>
      <option value="Shortest path">Shortest path</option>
      <option value="Steiner">Steiner</option>
      <option value="Cutting/packing family">Cutting/packing family</option>
      <option value="Travelling salesman problem">Travelling salesman problem</option>
      <option value="Vehicle routing">Vehicle routing</option>
      <option value="I am not sure...">I am not sure...</option>
    </Form.Select>
  );
}



useEffect(() => {
  setErrors([]);
  InputBlocks.cleanAll();
  RelationShipBlocks.cleanAll();
  GettersBlocks.cleanAll();


  const blocklyArea = document.getElementById("blocklyArea") as HTMLElement;
  const blocklyDiv = document.getElementById("blocklyDiv") as HTMLElement;

  workspace = Blockly.inject(blocklyDiv, {
    toolbox: toolbox,
```

```
      theme: {
        "componentStyles": {
          "flyoutBackgroundColour": "#E6E6EB",
          "toolboxBackgroundColour": "#E6E6EB",
          "workspaceBackgroundColour": "#13131A"
        },
        "fontStyle": {
          "family": "Hanken Grotesk Medium",
          "size": 12,
        }
      } as ITheme,
      zoom: {
        controls: true,
        wheel: true,
        startScale: 1.0,
        maxScale: 3,
        minScale: 0.3,
        scaleSpeed: 1.2,
        pinch: true
      },
      scrollbars: true,
      grid: {
        spacing: 20,
        length: 2,
        colour: "#777",
        snap: true,
      },
      trashcan: true,
      readOnly: viewMode,
    } as Blockly.BlocklyOptions);

    InputBlocks.setWorkSpace(workspace);
    RelationShipBlocks.setWorkSpace(workspace);
    if (json !== "") blocksExploration(JSON.parse(json), "SAVE");

    const onresize = () => {
      // Compute the absolute coordinates and dimensions of blocklyArea.
      let element = blocklyArea as HTMLElement;
      let x = 0;
      let y = 0;
      do {
        x += element.offsetLeft;
        y += element.offsetTop;
        element = element.offsetParent as HTMLElement;
      } while (element);

      // Position blocklyDiv over blocklyArea.
      blocklyDiv.style.left = x + "px";
      blocklyDiv.style.top = y + "px";
      blocklyDiv.style.width = blocklyArea.offsetWidth + "px";
      blocklyDiv.style.height = blocklyArea.offsetHeight + "px";
      Blockly.svgResize(workspace);
    };

    workspace.addChangeListener(variableParameterEvent);

    setRootBlock(CreateRootBlock(workspace, json));
    workspace.addChangeListener(Blockly.Events.disableOrphans);
```

```
  if (mode !== "VIEW") changeSVGZoomTrashCan();
  window.addEventListener("resize", onresize, false);
  onresize();
}, []);



return (
  <>
    <Container className={`${StyleWorkZone.containerWorkZone}
      col-12 col-md-10 col-lg-10 my-3`}>
      <Row className="d-flex justify-content-center">
        <Col className="col-12 col-md-1 mt-3 text-center">
          <Button onClick={() => { setMode("LIST"); setIdProblem(""); }}
            className={`${StyleProblem.goback} mb-2 p-0`}>{"<"}</Button>
        </Col>
        <Col className="text-center mt-3 col-12 col-md-10">
          <h1 className={`${StyleProblem.title}`}>Name of the problem:</h1>
        </Col>
        <Col className="col-12 col-md-1">
        </Col>
      </Row>
      <Row className="d-flex justify-content-center">
        <Col className="text-center my-1 col-10 col-sm-8 col-md-6">
          <InputGroup>
            <Form.Control
              disabled={mode === "VIEW"}
              placeholder="Problem name"
              aria-label="ProblemName"
              aria-describedby="Problem name input"
              value={problemName}
              onChange={(e) => {
                setProblemName(e.target.value);
              }}
            />
          </InputGroup>
        </Col>
      </Row>
      <Row className="d-flex justify-content-center">
        <Col className="text-center my-2 col-10 col-sm-8 col-md-6">
          <Button disabled={mode === "VIEW"} className={`${StyleProblem.
              saveProblemButton}`} onClick={() => sendProblem()}>
            Save Problem
          </Button>
        </Col>
      </Row>
      <Separator />
      <Row className="d-flex justify-content-center">
        <Col className="text-center col-10 col-sm-10 col-md-6">
          <h2 className={`${StyleProblem.title2} m-0`}>Problems types:</h2>
          <Row className="d-flex justify-content-center">
            <Col className="text-center col-10 my-1">
              <ProblemSelect />
            </Col>
          </Row>
        </Col>
      </Row>
```

```
      </Container>
      <ErrorReporter
        errors={errors}
        setErrors={setErrors}
      />
      <Row className="d-flex justify-content-center m-0">
        <Col className="text-center col-12 col-sm-10 p-0">
          <div id="blocklyArea">
            <div id="blocklyDiv" style={{ height: "500px" }} />
          </div>
        </Col>
      </Row>
      <ToastContainer />
    </>
  );
}
```

Apéndice D

Implementación de querys y mutations

D.1. Problemas

```
export const resolvers = {
  Query: {
    async problems(_, {owner, page, name, family, tableNumbers, paramsNumber}) {
      if (tableNumbers === -1) tableNumbers = undefined;
      if (paramsNumber === -1) paramsNumber = undefined;
      if (name === "") name = undefined;
      if (family=== "") family = undefined;
      try {
        const problemsPages = await problemController.getProblems(owner, page, name,
    family, tableNumbers, paramsNumber);
        return problemsPages;
      } catch(_) {
        throw new GraphQLError('An error has occurred with the problems', {
          extensions: {
            code: 'BAD_ID',
          },
        });
      }
    },
    async problem(_, {id}) {
      try {
        const problem = await problemController.getProblem(id);
        return problem;
      } catch(_) {
        throw new GraphQLError('An error has occurred with the owner', {
          extensions: {
            code: 'BAD_OWNER',
          },
        });
      }
    }
  },
  Mutation: {
    async createProblem(_, { input }) {
      try {
        const saveProblem = await problemController.postProblem(input);
        return saveProblem;

      } catch (_) {
        throw new GraphQLError('There is already a problem with that name', {
```

```
          extensions: {
            code: 'BAD_PROBLEM_NAME',
          },
        });
      }
    },
    async deleteProblem(_, {id}) {
      try {
        const deleteProblem = await problemController.deleteProblem(id);
        return deleteProblem;
      } catch (_) {
        throw new GraphQLError('An error has occurred with the delete', {
          extensions: {
            code: 'BAD_PROBLEM_DELETE',
          },
        });
      }
    },
    async pathProblem(_ , {id, input}) {
      try {
        const pathProblem = await problemController.pathProblem(id, input);
        return pathProblem;
      } catch (_) {
        throw new GraphQLError('There is already a problem with that name', {
          extensions: {
            code: 'BAD_PROBLEM_NAME',
          },
        });
      }
    }
  }
}
```

D.2. Instancias

```
export const resolvers = {
  Query: {
    async instances(_, {owner, page, name, tableNumbers, paramsNumber}) {
      if (tableNumbers === -1) tableNumbers = undefined;
      if (paramsNumber === -1) paramsNumber = undefined;
      if (name === "") name = undefined;
      try {
        const instancesPages = await instanceController.getInstances(owner, page, name,
    tableNumbers, paramsNumber);
        return instancesPages;
      } catch(_) {
        throw new GraphQLError('An error has occurred with the instances', {
          extensions: {
            code: 'BAD_ID',
          },
        });
      }
    },
    async instance(_, {id}) {
      console.log(id);
      try {
```

```
        const instance = await instanceController.getInstance(id);
        return instance;
      } catch(_) {
        throw new GraphQLError('An error has occurred with the owner', {
          extensions: {
            code: 'BAD_OWNER',
          },
        });
      }
    }
  },
  Mutation: {
    async createInstance(_, { input }) {
      try {
      const saveInstance = await instanceController.postInstance(input);
      return saveInstance;
      } catch (_) {
        throw new GraphQLError('There is already a instance with that name', {
          extensions: {
            code: 'BAD_INSTANCE_NAME',
          },
        });
      }
    },
    async deleteInstance(_, {id}) {
      try {
        const deleteInstance = await instanceController.deleteInstance(id);
        return deleteInstance;
      } catch (_) {
        throw new GraphQLError('An error has occurred with the delete', {
          extensions: {
            code: 'BAD_INSTANCE_DELETE',
          },
        });
      }
    },
    async pathInstance(_ , {id, input}) {
      try {
        const pathInstance = await instanceController.pathInstance(id, input);
        return pathInstance;
      } catch (_) {
        throw new GraphQLError('There is already a problem with that name', {
          extensions: {
            code: 'BAD_INSTANCE_NAME',
          },
        });
      }
    }
  }
}
```

Apéndice E

Controladores

E.1. Problems

```
export default class ProblemController {

  async postProblem(input: { owner: string, name: string, family: string, blocklyJSON:
    string, problemObject: string, parametersCount: number, filesCount: number }) {

    const isProblem = await Problem.findOne({name: input.name});
    if (isProblem) throw "There is already a problem with that name";

    const newProblem = new Problem(input);
    await newProblem.save();

    return newProblem;
  }

  async getProblems(owner: string, page: number, name: string, family: string,
    tableNumbers: number, paramsNumber: number) {
    let search = {owner};
    if (tableNumbers !== undefined) search["filesCount"] = tableNumbers;
    if (paramsNumber !== undefined) search["parametersCount"] = paramsNumber;
    if (name !== undefined) search["name"] = new RegExp(name);
    if (family !== undefined) search["family"] = family;

    const totalElements: number = await Problem.countDocuments(search);
    const pages: number = totalElements % 10 === 0 ? totalElements / 10 >> 0 :
    totalElements / 10 + 1 >> 0;

    const next = page < pages ? page + 1 : -1;
    const prev = page !== 1 ? page - 1 : -1;

    const elements = await Problem.find(search).skip(10 * (page - 1)).limit(10);

    const tables = await Problem.find(search, {_id:0, filesCount: 1});
    const setTables = new Set(tables.map(result => result.filesCount));
    const finalTables = [...setTables];

    const params = await Problem.find(search, {_id:0, parametersCount: 1});
    const setParams = new Set(params.map(result => result.parametersCount));
    const finalParams = [...setParams];

    const familys = await Problem.find(search, {_id:0, family: 1});
```

```
    const setFamilys = new Set(familys.map(result => result.family));
    const finalFamily = [...setFamilys];

    const info = {
      count: totalElements,
      pages,
      next,
      prev
    };

    return {info, problem: elements, familys: finalFamily, tables: finalTables, params: finalParams};
  }

  async getProblem(id: string) {
    const problem = await Problem.findOne({_id: id});

    return problem;
  }

  async deleteProblem(id: string) {
    const problem = await Problem.findOneAndDelete({_id: id});

    if (problem) return true;
    else return false;
  }

  async pathProblem(id: string, input: { owner: string, name: string, family: string, blocklyJSON: string, problemObject: string, parametersCount: number, filesCount: number }) {

    const isProblem = await Problem.findOne({name: input.name});
    console.log(id, isProblem?._id);
    if (isProblem && isProblem._id.toString() !== id) throw "There is already a problem with that name";

    const problem = await Problem.findOneAndUpdate({_id: id}, {name: input.name, family: input.family, blocklyJSON: input.blocklyJSON, problemObject: input.problemObject, parametersCount: input.parametersCount, filesCount: input.filesCount}, {new: true, runValidators: true});
    if (problem) return true;
    else return false;
  }
}
```

E.2. Instancias

```
export default class InstanceController {

  async postInstance(input: { owner: string, name: string, data: { parameterInputs: { name: string, value: number }[], tablesInputs: { name: string, headers: string[], data: number[][] }[] }, parametersCount: number, filesCount: number, filesDimensions: string[] }) {

    const isInstance = await Instance.findOne({name: input.name});
    if (isInstance) throw "There is already a instance with that name";
```

```
    const newInstance = new Instance(input);
    await newInstance.save();

    return newInstance;
  }

  async getInstances(owner: string, page: number, name: string, tableNumbers: number,
    paramsNumber: number) {
    let search = {owner};
    if (tableNumbers !== undefined) search["filesCount"] = tableNumbers;
    if (paramsNumber !== undefined) search["parametersCount"] = paramsNumber;
    if (name !== undefined) search["name"] = new RegExp(name);


    const totalElements: number = await Instance.countDocuments(search);
    const pages: number = totalElements % 10 === 0 ? totalElements / 10 >> 0 :
    totalElements / 10 + 1 >> 0;

    const next = page < pages ? page + 1 : -1;
    const prev = page !== 1 ? page - 1 : -1;

    const elements = await Instance.find(search).skip(10 * (page - 1)).limit(10);

    const tables = await Instance.find(search, {_id:0, filesCount: 1});
    const setTables = new Set(tables.map(result => result.filesCount));
    const finalTables = [...setTables];


    const params = await Instance.find(search, {_id:0, parametersCount: 1});
    const setParams = new Set(params.map(result => result.parametersCount));
    const finalParams = [...setParams];


    const info = {
      count: totalElements,
      pages,
      next,
      prev
    };

    return {info, instance: elements, tables: finalTables, params: finalParams};
  }

  async getInstance(id: string) {
    const instance = await Instance.findOne({_id: id});

    return instance;
  }

  async deleteInstance(id: string) {
    const instance = await Instance.findOneAndDelete({_id: id});

    if (instance) return true;
    else return false;
  }

  async pathInstance(id: string, input: { owner: string, name: string, data: {
```

```
    parameterInputs: { name: string, value: number }[], tablesInputs: { name: string,
    headers: string[], data: number[][] }[] }, parametersCount: number, filesCount:
    number, filesDimensions: string[] }) {
63
64      const isInstance = await Instance.findOne({name: input.name});
65      if (isInstance && isInstance._id.toString() !== id) throw "There is already a
        instance with that name";
66
67      const instance = await Instance.findOneAndUpdate({_id: id}, {name: input.name, data:
        input.data, parametersCount: input.parametersCount, filesCount: input.filesCount,
        filesDimensions: input.filesDimensions}, {new: true, runValidators: true});
68      if (instance) return true;
69      else return false;
70    }
71  }
```

Bibliografía

[1] Apollo GraphQL | Supergraph: unify APIs, microservices, databases in a composable graph — apollographql.com. https://www.apollographql.com/. [Accessed 12-May-2023].

[2] Axios — npmjs.com. https://www.npmjs.com/package/axios. [Accessed 12-May-2023].

[3] Bcrypt — npmjs.com. https://www.npmjs.com/package/bcrypt. [Accessed 12-May-2023].

[4] Blockly — npmjs.com. https://www.npmjs.com/package/blockly. [Accessed 12-May-2023].

[5] Bootstrap — npmjs.com. https://www.npmjs.com/package/bootstrap. [Accessed 12-May-2023].

[6] Formik — formik.org. https://formik.org/. [Accessed 14-May-2023].

[7] GraphQL | A query language for your API — graphql.org. https://graphql.org/. [Accessed 12-May-2023].

[8] Jest — npmjs.com. https://www.npmjs.com/package/jest. [Accessed 12-May-2023].

[9] Jwt — npmjs.com. https://www.npmjs.com/package/jwt. [Accessed 12-May-2023].

[10] Koa — npmjs.com. https://www.npmjs.com/package/koa. [Accessed 12-May-2023].

[11] MongoDB: La Plataforma De Datos Para Aplicaciones — mongodb.com. https://www.mongodb.com/es. [Accessed 12-May-2023].

[12] Node-red — npmjs.com. https://www.npmjs.com/package/node-red. [Accessed 12-May-2023].

[13] Node-red-contrib-blockly — npmjs.com. https://www.npmjs.com/package/node-red-contrib-blockly. [Accessed 12-May-2023].

[14] Papaparse — npmjs.com. https://www.npmjs.com/package/papaparse. [Accessed 12-May-2023].

[15] React — npmjs.com. https://www.npmjs.com/package/react. [Accessed 12-May-2023].

[16] React-bootstrap — npmjs.com. https://www.npmjs.com/package/react-bootstrap. [Accessed 12-May-2023].

[17] Redux — npmjs.com. https://www.npmjs.com/package/redux. [Accessed 12-May-2023].

[18] Rete — npmjs.com. https://www.npmjs.com/package/rete. [Accessed 12-May-2023].

[19] Rust, el lenguaje de programacixF3;n — rust-lang.org. https://www.rust-lang.org/es. [Accessed 22-May-2023].

[20] Scratch-blocks — npmjs.com. https://www.npmjs.com/package/scratch-blocks. [Accessed 12-May-2023].

[21] Showdown — npmjs.com. https://www.npmjs.com/package/showdown. [Accessed 12-May-2023].

[22] SWC: Speedy Web Compiler — swc.rs. https://swc.rs/. [Accessed 12-May-2023].

[23] ULL-prodef — github.com. https://github.com/ULL-prodef/prodef. [Accessed 12-May-2023].

[24] Vite — vitejs.dev. https://vitejs.dev/. [Accessed 12-May-2023].

[25] Yup — npmjs.com. https://www.npmjs.com/package/yup. [Accessed 14-May-2023].

[26] Patak Mohammad Bagher Ivan Demchuk Joaquín Sánchez Zixuan Chen Yoho Po Ari Perkkiö Anthony Fu, Vladimir and Vitest contributors. Vitest — vitest.dev. https://vitest.dev/. [Accessed 12-May-2023].

[27] Gara Miranda Coromoto León and Carlos Segura. Metco: A parallel plugin-based framework for multi-objective optimization. *International Journal on Artificial Intelligence Tools*, 18(4):569–558, 2009.

[28] Daniel del Castillo de la Rosa. Prodef-algorithm: Interfaz para el modelado de meta-heurísticas. Technical report, Universidad de La Laguna, 2022.

[29] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[30] Andrés Calimero García Pérez. Prodef: meta-modelado de problemas de optimización combinatoria. Technical report, Universidad de La Laguna, 2020.

[31] Daniel González Expósito. Prodef-gui: Interfaz gráfica para el modelado de problemas. Technical report, Universidad de La Laguna, 2021.

[32] Yeixon Reinaldo Morales Gonzalez. Prodef-solution: Interfaz para la representación y visualización de soluciones. Technical report, Universidad de La Laguna, 2022.

[33] Miguel Angel Ordoñez Morales. Prodef: Diseño, implementación y experimentación con nuevos resolutores. Technical report, Universidad de La Laguna, 2022.

[34] Ángel Tornero Hernández. Prodef-saas: Despliegue y puesta en marcha de un servicio para la resolución de problemas de optimización. Technical report, Universidad de La Laguna, 2022.