Facultad
de
Ciencias

Desarrollo de un servicio web para la gestión de catálogos de Earth System Grid Federation.

Development of a web service for the management of Earth System Grid Federation catalogues.

Trabajo de Fin de Grado
para acceder al
GRADO EN INGENIERÍA INFORMÁTICA

Autor: Juan Vélez Velasco
Director: Antonio S. Cofiño
Codirector: Ezequiel Cimadevilla Álvarez
Julio - 2022

Agradecimientos

Tras 4 años de trabajo y esfuerzo doy por finalizada otra etapa de mi vida con este trabajo. Sin duda es de las mejores etapas, sino la mejor gracias a la gente a la que he conocido y los momentos que he vivido durante este experiencia, que me han ayudado a crecer tanto personalmente como académicamente.
Creo que no la habría disfrutado tanto de no ser por los amigos que tenía y he mantenido, y por los nuevos amigos que espero que duren para siempre.
Especialmente me gustaría agradecer esta aventura a mis colegas de la uni, "Los Chemas", y a Ángela.
Tampoco me quiero olvidar de todos los demás, que ya saben quienes son.
A mi familia, que me han educado lo mejor que han podido y me han hecho ser las persona que hoy en día soy.
A mis profesores, que me han ayudado con todo lo que han podido.
A mis dos tutores de este proyecto, Antonio Cofiño y Ezequiel Cimadevilla, sin ellos este proyecto no hubiera sido posible, han sido una gran ayuda en todos los aspectos. Sus enormes conocimientos en este ámbito y su experiencia me han hecho aprender mucho. Les tengo mucho cariño y espero que siga siendo así durante mucho tiempo.

Resumen

Earth System Grid Federation (ESGF) está formado por grupos de colaboración y agencias e instituciones internacionales dedicadas al desarrollo y gestión de un sistema *Peer-to-Peer* (P2P) para gestionar, difundir y analizar datos climáticos de gran volumen. Esta infraestructura de software tiene como finalidad facilitar los avances en la investigación del Earth System Science, entre ellos la investigación sobre el cambio climático. El Grupo de Meteorología de Santander (GMS), participa activamente en ESGF.

Las instituciones principales forman parte de ESGF como nodos primarios (*tier 1*), mientras que instituciones de menor tamaño pueden formar parte de la federación como nodos secundarios, ofreciendo solo servicio de acceso a datos (*tier 2*). El GMS participa en la federación ESGF como un nodo de datos de *tier 2*.

ESGF ofrece distintos servicios de acceso a datos, entre los que se encuentran descarga de ficheros, acceso remoto mediante DAP y descarga mediante GridFTP. La descarga de ficheros se encuentra limitada a la descarga de un *script shell* que descarga los ficheros usando Wget, o la descarga directamente usando el navegador del usuario.

Este proyecto tiene como objetivo el desarrollo de un servicio web que permita la generación, acceso y visualización de catálogos usando los servicios web de ESGF, además de ofrecer medios de descarga más sencillos y funcionales que mejoren la experiencia de usuario que desean descargar datos de modelos climáticos.

Palabras clave

ESGF, Servicio Web, datos climáticos.

Abstract

The Earth System Grid Federation (ESGF) consists of collaborative groups and international agencies and institutions dedicated to the development and management of a 'peer-to-peer' (P2P) system for managing, disseminating and analysing high-volume climate data. This software infrastructure is intended to facilitate advances in Earth System Science research, including climate change research. The Santander Meteorology Group (GMS) actively participates in ESGF.

Major institutions are part of ESGF as primary nodes (*tier 1*), while smaller institutions can be part of the federation as secondary nodes, offering only data access service (*tier 2*). The GMS participates in the ESGF federation as a data node of *tier 2*.
ESGF offers different data access services, including file download, remote access via DAP and download via GridFTP. File downloading is limited to downloading from a *script shell* that downloads files using Wget, or downloading directly using the user's browser.
This project aims to develop a web service that allows the generation, access and visualisation of catalogues using ESGF web services, as well as offering simpler and more functional means of downloading to improve the experience of users wishing to download climate model data.

Keywords

ESGF, Web Service, climate data.

Índice

1. Introducción **8**
- 1.1. Contexto . . . 8
- 1.2. Motivación . . . 10
- 1.3. Objetivo . . . 10

2. Herramientas, tecnologías utilizadas **11**
- 2.1. Tecnologías y Herramientas . . . 11
- 2.2. Lenguajes . . . 12
- 2.3. Autorización y Autenticación . . . 13
- 2.4. Conceptos previos . . . 15

3. Metodología y planificación **16**
- 3.1. Metodología . . . 16
- 3.2. Planificación . . . 16
- 3.3. Diagrama de Gantt . . . 18

4. Análisis y especificación de requisitos **19**
- 4.1. Análisis de requisitos . . . 19
  - 4.1.1. Actores del sistema . . . 19
- 4.2. Especificación de requisitos . . . 20
  - 4.2.1. Requisitos funcionales . . . 20
  - 4.2.2. Requisitos no funcionales . . . 20

5. Especificación de casos de uso **21**
- 5.1. Casos de uso de la vista de búsqueda . . . 21
- 5.2. Casos de uso de la vista de carros guardados . . . 25
- 5.3. Casos de uso de la vista de búsquedas guardadas . . . 26
- 5.4. Casos de uso de la gestión de usuarios . . . 27

6. Entornos de trabajo **28**
- 6.1. Configuración de la aplicación . . . 29
- 6.2. Despliegue . . . 31

7. Diseño e implementación **32**
- 7.1. Diseño arquitectónico . . . 33
- 7.2. Capa de presentación . . . 34
  - 7.2.1. Vista de elección de filtros . . . 35
  - 7.2.2. Vista del resultado de la búsqueda . . . 36
  - 7.2.3. Vista de los *files* de un *dataset* . . . 37
  - 7.2.4. Vista de los metadatos de un *dataset* . . . 38
  - 7.2.5. Vista del carro . . . 39
  - 7.2.6. Vista de la lista de búsquedas guardadas . . . 40
  - 7.2.7. Vista de la lista de carros guardados . . . 41

7.3. Capa de negocio 42
7.4. Capa de datos 44

8. Pruebas y resultados **48**
8.1. Pruebas unitarias 48
8.2. Pruebas de Sistema 48
8.2.1. Pruebas de usabilidad 48
8.2.2. Pruebas de rendimiento 48
8.3. Pruebas de aceptación 48

9. Conclusiones y trabajos futuros **50**

Índice de figuras

1. Representación esquemática de la arquitectura de ESGF. Fuente: [1] 9
2. Flujo del protocolo OAuth2. Fuente: [2] 13
3. Tareas a realizar. 18
4. Diagrama de Gantt. 18
5. Vista de búsqueda para el rol User 21
6. Vista de búsqueda para los usuarios registrados. 23
7. Lista de carros guardados para los usuarios registrados. 25
8. Lista de carros guardados para los usuarios registrados. 26
9. Gestión de usuarios por parte del administrador 27
10. Configuración de Derby. 29
11. Configuración de OAuth2 usando Github. 30
12. Configuración de proyectos junto a sus *facets*. 31
13. Estructura del proyecto 32
14. Diseño arquitectónico 33
15. Vista principal 34
16. Vista de elección de filtros 35
17. Vista de lista de resultados. 36
18. Vista de lista de ficheros de un *dataset*. 37
19. Vista de los metadatos de un *dataset*. 38
20. Vista del carro. 39
21. Vista de las búsquedas guardadas. 40
22. Vista de los carros guardados. 41
23. Diagrama de clases de los controladores y servicios. 43
24. ESGF Search RESTful API 44
25. Embebbed Derby 45
26. Diagrama de entidades de la base de datos relacional Derby generado con DbSchema. 46
27. Diagrama de clases de la capa DAO generado con ObjectAid. 47

Índice de tablas

1. Actores del sistema. 19
2. Requisitos funcionales del sistema. 20
3. Requisitos no funcionales del sistema. 20
4. Caso de uso: Inicio de sesión. 22
5. Caso de uso: Actualizar carro. 22
6. Caso de uso: *Check-out* del carro. 23
7. Caso de uso: Listar carros *checkeados*. 24
8. Caso de uso: Listar búsquedas guardadas. 24
9. Caso de uso: Descarga de carro checkeado. 25
10. Caso de uso: Renombrar búsqueda. 26
11. Caso de uso: Listar usuarios. 27
12. URLs de las aplicaciones. 28

1. Introducción

1.1. Contexto

Earth System Grid Federation (ESGF) proporciona una infraestructura desde 2004 para la publicación y mantenimiento de datos climáticos a nivel mundial. Sin embargo, las herramientas desde el lado del cliente para operar con estos datos son escasas, limitando la accesibilidad y sistematización de los mismos lo que reduce las posibilidades de estudios y predicciones por parte de los investigadores.

Este Trabajo Fin de Grado (TFG) presenta un servicio web para la generación, acceso y visualización de catálogos usando los servicios web de *ESGF*. Además, este servicio web permite interactuar con una interfaz de usuario en la que se puede visualizar y explorar los distintos datos resultantes de la búsqueda realizada por el usuario. Este servicio software debe englobar el flujo que va desde la búsqueda del dato hasta la descarga de un *Metalink* (formato estándar de descarga de metadatos) [3] que permitirá la posterior descarga por parte del usuario (en este caso un Investigador o Científico del Clima) usando cualquier cliente compatible con Metalink.

Esta **búsqueda facetada**, es una técnica para acceder a la información organizada con un sistema de clasificación de propiedades de los datos climáticos. El objetivo de este tipo de búsqueda es la obtención listas reducidas de resultados que cumplen unas propiedades especificas.

ESGF es una colaboración de grupos, agencias e instituciones de todo el mundo que se dedican al desarrollo y funcionamiento de un sistema a largo plazo para la gestión, acceso y análisis de datos climáticos [1]. Algunos de los retos que el ESGF se ha comprometido a abordar son:

- La enorme escala de los datos, pasando de petabytes a exabytes.
- El soporte tanto de los resultados de los modelos como de una gran variedad de datos de observación.
- La naturaleza distribuida de los archivos de datos, que están distribuidos geográficamente y funcionan de forma autónoma. La necesidad de permitir a los usuarios acceder a los datos y analizarlos con una amplia variedad de herramientas cliente, no sólo navegadores web, sino también clientes de escritorio enriquecidos, bibliotecas y conjuntos de herramientas.

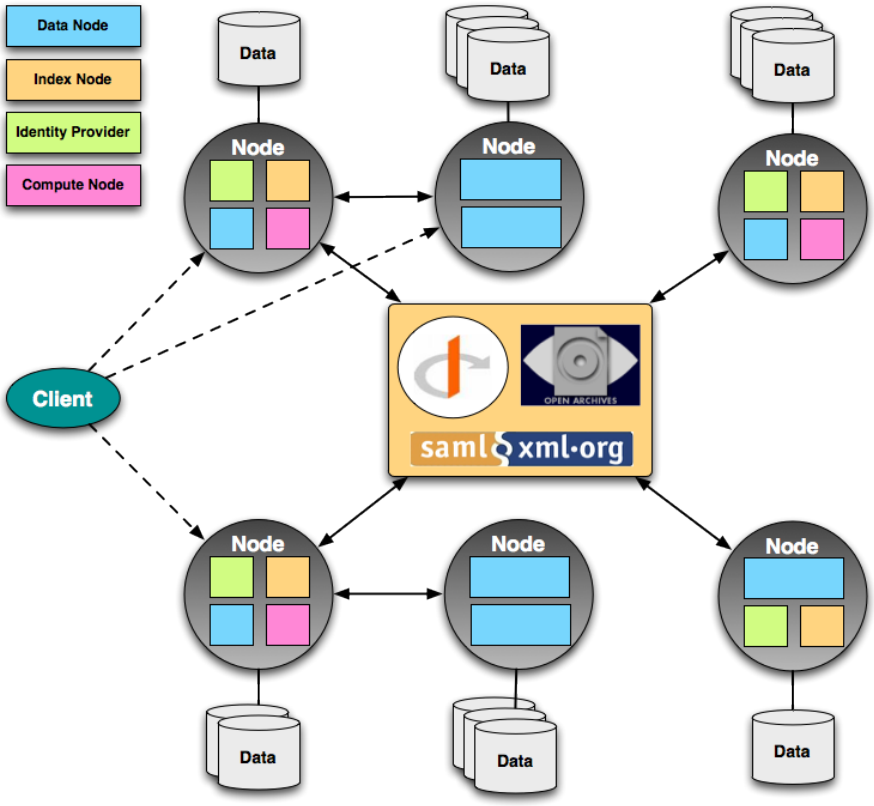

Figura 1: Representación esquemática de la arquitectura de ESGF. Fuente: [1]

La **arquitectura de ESGF** se basa en un sistema de nodos autónomos y distribuidos, que interoperan mediante la aceptación común de protocolos de federación y acuerdos de confianza. Los datos se almacenan en varios nodos y se ofrecen a través de servicios locales de datos y metadatos. Los nodos intercambian información sobre sus existencias de datos y servicios, y confían unos en otros para registrar a los usuarios y establecer decisiones de control de acceso. El resultado neto es que un usuario puede utilizar un navegador web o una aplicación de escritorio, conectarse a cualquier nodo y encontrar y acceder sin problemas a los datos de toda la federación.
En este proyecto a menudo se hablará de *facet*, que corresponde a faceta, o propiedad de un dato climático. Cada *facet* puede incluir uno o más parámetros.

Por **facet**, se hace referencia a parámetros que sirven como categorías de búsqueda usados para aplicar restricciones sobre el conjunto de datos que proporciona ESGF. Los datos devueltos por una búsqueda son aquellos que cumplen con las restricciones que hemos indicado, es decir, que los valores de los metadatos de ese dato satisfagan los filtros aplicados [1].

1.2. Motivación

Los datos climáticos en *ESGF* se publican a nivel de dataset, el cual es una colección de otros datos que generalmente están físicamente organizados en ficheros y agregaciones. Las publicaciones de estos datos se hacen en un nodo en concreto, generando así una réplica del dato, por lo que puede haber réplicas en nodos diferentes y cada réplica ofrece diferentes servicios de acceso y metadatos. Por otro lado, los servicios ofrecidos a nivel de dataset y fichero son diferentes. Hoy en día, no hay un servicio que proporcionado un dato, devuelva un catálogo con todas las réplicas y sus servicios. Esto justifica que se haga necesario implementar una herramienta que facilite la obtención de resultados a través de la arquitectura P2P que ESGF nos proporciona.
El grupo Grupo de Meteorología de Santander (GMS), trabaja haciendo uso de los datos que ofrece ESGF, en concreto acceden a *datasets* y a su posterior descarga diariamente, requiriendo la gestión de ingentes cantidades de datos y ficheros de ESGF. Es por ello que se considera adecuado o justificado presuponer que se precisa del desarrollo de una herramienta capaz de coadyuvar en el proceso de obtención de resultados.

1.3. Objetivo

Este proyecto tiene como objetivo el desarrollo de un servicio web que permita la generación, acceso y visualización de catálogos utilizando los propios servicios que *ESGF* nos proporciona. Por otro lado, deberá permitir la descarga de los datos seleccionados previamente por el usuario, pudiendo ser *datasets*(colecciones de ficheros) o ficheros. Asismismo el proyeco tiene como sub-objetivos:

- Permitir la búsqueda facetada agregando a la misma la opción de palabras clave.
- Facilitar la posibilidad de guardar un historial de búsquedas.
- Permitir la posibilidad de guardar *carros*.
- Ofrecer la opción de descargar un archivo de descarga (.metalink) partiendo de un fichero, un dataset o un carro.

2. Herramientas, tecnologías utilizadas

2.1. Tecnologías y Herramientas

A continuación se ofrece una breve explicación de las diferentes tecnologías y herramientas empleadas a lo largo del desarrollo de este proyecto.

- Git
  Software de control de versiones de código abierto y gratuito pensado en la eficiencia y la confiabilidad del mantenimiento de versiones de proyectos tanto con muchos archivos de código fuente como proyectos pequeños.

- Maven
  Maven es una herramienta de gestión y compresión de proyectos software. Está basado en un modelo de objetos de proyecto (POM).

- Spring
  Spring [4] es un framework de código abierto que proporciona soporte de infraestructura para el desarrollo de aplicaciones JAVA. Se ha usado la version de Spring 2.6.7.

- JSP
  JavaServer Pages [5] es una tecnología que proporciona una forma simplificada y rápida de crear contenido web dinámico. JSP permite el desarrollo de aplicaciones basadas en la web que son independientes del servidor y la plataforma.

- Eclipse
  Es una plataforma de desarrollo gratuita basada en Java.

- Derby
  Apache Derby [6] es una base de datos relacional de código abierto implementada completamente en Java y disponible bajo la licencia de Apache.

- aria2
  aria2 [7] es una herramienta de descarga multiprotocolo y multifuente. Soporta HTTP/HTTPS, FTP, SFTP, BitTorrent y Metalink. aria2 puede ser manipulado a través de interfaces JSON-RPC y XML-RPC incorporadas. En este sistema usaremos el tipo de archivo .metalink en su versión 4.

- Metalink
  Es un formato de archivo de metadatos extensible [3] que describe uno o varios archivos informáticos disponibles para su descarga. Especifica los archivos apropiados para el idioma y el sistema operativo del usuario; facilita la verificación de los archivos y la recuperación en caso de corrupción de datos; y enumera fuentes de descarga alternativas (mirror URIs).

- OAuth2
  OAuth [8] es un estándar abierto para la delegación de acceso, comúnmente utilizado como una forma para que los usuarios de Internet concedan a los sitios web o aplicaciones el acceso a su información en otros sitios web, pero sin facilitarles las contraseñas.

- MagicDraw
  MagicDraw UML es una herramienta CASE desarrollada por No Magic. La herramienta es compatible con el estándar UML 2.3, desarrollo de código para diversos lenguajes de programación así como para modelar datos.

- Draw.io
  Se ha usado para crear algunas de las figuras.

- DbSchema
  Se ha usado DbSchema para general el diagrama de clases de la base de datos en Derby usando el schema generado por Spring Data JPA. [9]

2.2. Lenguajes

- Java
  Java es un lenguaje de programación de alto nivel, basado en clases y orientado a objetos, diseñado para tener el menor número posible de dependencias de implementación. Es un lenguaje de programación de propósito general que permite a los programadores escribir una vez y ejecutar en cualquier lugar, lo que significa que el código Java compilado puede ejecutarse en todas las plataformas que soportan Java sin necesidad de recompilar. [10]
  Se ha usado Java 8.

- CSS
  Las hojas de estilo en cascada (CSS) son un lenguaje de hojas de estilo que se utiliza para describir la presentación de un documento escrito en un lenguaje de marcado como HTML o XML (incluidos los dialectos de XML como SVG, MathML o XHTML). CSS es una tecnología fundamental de la World Wide Web, junto con HTML y JavaScript. [11]

- HTML
  El Lenguaje de Marcas de Hipertexto o HTML es el lenguaje de marcas estándar para los documentos diseñados para ser mostrados en un navegador web. Puede ser asistido por tecnologías como las Hojas de Estilo en Cascada (CSS) y lenguajes de scripting como JavaScript. [12]

- JSON
  JSON (acrónimo de JavaScript Object Notation, 'notación de objeto de JavaScript') es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje. [13]

- XML
  XML, siglas en inglés de eXtensible Markup Language, traducido como 'Lenguaje de Marcado Extensible' o 'Lenguaje de Marcas Extensible', es un metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes. A diferencia de otros lenguajes, XML da

soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información. [14]

- YAML
  YAML es un lenguaje de serialización de datos que suele utilizarse en el diseño de archivos de configuración. [15]

2.3. Autorización y Autenticación

Para ofrecer al usuario la posiblidad de almacenar datos, es necesario que exista una forma de iniciar sesión en la aplicación. Se ha optado por usar el protocolo de **OAuth2** [16] con la ayuda que Spring nos ofrece, por lo que no ha sido necesario manejar *tokens*, ya que Spring lo hace por detrás. A continuación se expone el proceso del protocolo de OAuth2 a modo explicativo: [17]
El marco de autorización OAuth 2.0 permite a una aplicación de terceros obtener un acceso limitado a un servicio HTTP, ya sea en nombre del propietario de un recurso, orquestando una interacción de aprobación entre el propietario del recurso y el servicio HTTP, o permitiendo que la aplicación aplicación de terceros obtenga acceso en su propio nombre. Esta especificación especificación sustituye y deja obsoleto el protocolo OAuth 1.0. [18]

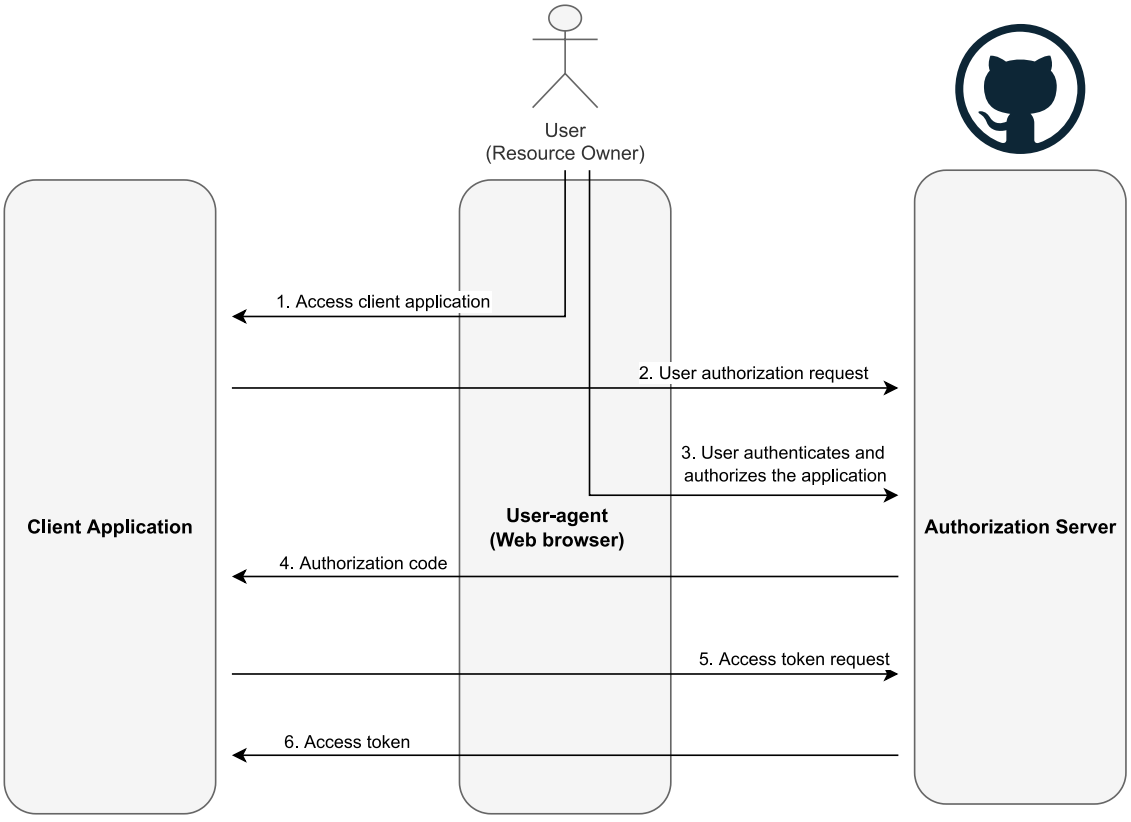

Figura 2: Flujo del protocolo OAuth2. Fuente: [2]

Roles:

- Propietario del recurso: Entidad capaz de conceder acceso a un recurso protegido. Cuando el propietario del recurso es una persona, se denomina usuario final.
- Servidor de recursos: El servidor que aloja los recursos protegidos, capaz de aceptar y responder a las solicitudes de recursos protegidos utilizando tokens de acceso.
- Cliente: Una aplicación que realiza solicitudes de recursos protegidos en nombre del propietario del recurso y con su autorización.
- Servidor de autorización: El servidor que emite tokens de acceso al cliente después de autenticar al propietario del recurso y obtener la autorización. [18]

Pasos:

- El cliente solicita autorización al propietario del recurso. La solicitud de autorización puede hacerse directamente al propietario del recurso (como se muestra), o preferiblemente de forma indirecta a través del servidor de como intermediario.
- El cliente recibe una concesión de autorización, que es una credencial que representa la autorización del propietario del recurso.
- El cliente solicita un token de acceso autenticándose con el servidor de autorización y presentando la concesión de autorización.
- El servidor de autorización autentifica al cliente y valida la concesión de la autorización y, si es válida, emite un token de acceso.
- El cliente solicita el recurso protegido al servidor de recursos y se autentifica presentando el token de acceso.
- El servidor de recursos valida el token de acceso, y si es válido sirve la solicitud. [18]

2.4. Conceptos previos

A continuación se detallan algunos de los conceptos empleados durante este proyecto y que son usados con frecuencia en esta memoria. Todo ello con el ánimo de que ayuden a entender el funcionamiento de la aplicación.

Elemento	Descripción
ESGF	Earth System Grid Federation [19] es una colaboración que desarrolla, despliega y mantiene una insfraestuctura de sofware para la gestión, difusión y análisis de los resultados de los modelos y de los datos de observación. El objetivo principal del ESGF es facilitar los avances en la ciencia del sistema terrestre. Se trata de un esfuerzo interinstitucional e internacional dirigido por el Departamento de Energía (DOE) y cofinanciado por la Administración Nacional de Aeronáutica y del Espacio (NASA), la Administración Nacional Oceánica y Atmosférica (NOAA), la Fundación Nacional de la Ciencia (NSF) y laboratorios internacionales. La misión del ESGF es apoyar las actividades actuales de la CMIP6 y preparar las evaluaciones futuras. ESGF P2P es una arquitectura de componentes diseñada expresamente para gestionar datos a gran escala para su distribución mundial. El equipo de informáticos y climatólogos ha desarrollado un sistema operativo para servir datos climáticos procedentes de múltiples lugares y fuentes. Las simulaciones de modelos, las observaciones por satélite y los productos de reanálisis se sirven desde el archivo de datos distribuidos ESGF P2P.
Dataset	Es la unidad básica de ESGF. Un dataset es una colección de ficheros individuales. Algunos datasets contienen archivos que representan diferentes variables, en cambio otros se limitan a una variable. Otros como el CMIP5 contienen muchos datasets con múltiples variables.
Facet	Faceta, es una propiedad de los datos que permite filtrar o realizar una búsqueda. Esta faceta tiene distintos parámetros como por ejemplo, la faceta Frecuencia que contiene los parámetros: horario, diario, mensual, etc.
Carro	Por *carro* se entiende el conjunto de ficheros que el usuario ha ido añadiendo. Las acciones que se pueden hacer con el *carro* son: añadir, eliminar, descargar o hacer un *check-out*.
Check-out	Es la acción que consigue almacenar un *carro* en la base de datos para su futura descarga. Al hacer un *check-out*, se almacenan en la base de datos algunas de las propiedades de los ficheros que componen el *carro*, junto a las réplicas de cada fichero para su futura descarga sin tener que atacar a la API de ESGF SOLR, puesto que los datos ya están persistidos. Con esto se consigue que la acción de descarga sea más rápida, puesto que lo que mas tiempo consume es conseguir las réplicas de cada fichero.

3. Metodología y planificación

A continuación se expone la metodología y la organización llevada a cabo durante el desarrollo del proyecto.

3.1. Metodología

Para la realización de este proyecto se ha utilizado la metodología de desarrollo de software iterativa e incremental [20]. Este modo de desarrollo consiste en la idea de diseñar una implementación inicial, exponerla y posteriormente desarrollar diferentes versiones hasta llegar al sistema adecuado.
Esta metodología permite ir entendiendo, pòr parte del desarrollador, cada uno de los elementos del problema a tratar de una forma sencilla y progresiva.
Se ha usado Gitlab como herramienta de control de versiones y así poder llevar una correcta comprobación del proceso. En cada versión, se encuentran las actividades de especificación, desarrollo y validación. Algún beneficio que nos proporciona el uso del desarrollo incremental son el bajo coste al adaptar los nuevos cambios del cliente y la sencilla obtención de retroalimentación a cerca del trabajo realizado.
Al trabajar de esta forma, realizar los cambios que el cliente solicite tiene un menor coste y es más sencillo. Esta metodología ha podido ser usada gracias a que la comunicación con el Director y el Co-Director ha sido buena y frecuente.

3.2. Planificación

Como se ha explicado en el apartado anterior, se ha seguido la metodología de desarrollo incremental para la realización de este proyecto. Las fases en las que se ha dividido el trabajo fueron: en una primera instancia se realizó un análisis de requisitos y una investigación en la que se eligieron las distintas tecnologías a utilizar.
Posteriormente se desarrolló la aplicación, dividiéndose en 5 grandes versiones, la primera versión corresponde a la obtención de los *facets* para que el usuario pueda filtrar y a partir de estos conseguir datos climáticos atacando el servicio de búsqueda en *SOLR* que *ESGF* nos proporciona, la segunda versión corresponde a la creación de la página principal, es decir, el *front-end*, la tercera versión corresponde al diseño y desarrollo de un *carro* en el que el usuario pueda añadir o eliminar *datasets* y ficheros, además de la descarga de este *carro* o de *datasets* y *files* individualmente, la cuarta versión corresponde con la funcionalidad de iniciar sesión en el sistema usando el oauth2 de Github. En la quinta versión se trata la funcionalidad de guardar un historial de búsqueda, para ello hubo que trabajar con alguna forma de persistir información, en la última versión se trabajó con la opción de persistir *carros*.

Especificación de cada versión en profundidad:

- **Versión 1**: El objetivo de esta versión es conseguir recuperar todos los *facets* con sus parámetros que ESGF SOLR nos proporciona para que el usuario sea capaz de filtrar y obtener los distintos *datasets* que cumplen con los filtros.
- **Versión 2**: En la segunda versión, se crea la página principal usando únicamente html, css y bootstrap. Se muestran los distintos *facets* en forma de menú para que el usuario sea capaz

de elegir uno o varios parámetros de cada *facet* y luego realizar la búsqueda.

- **Versión 3**: En esta versión, se busca que el usuario sea capaz de añadir y eliminar *datasets* o *files* de su *carro*. Cada vez que el usuario añade un elemento a su *carro*, este es almacenado en el servidor y permanece mientras la sesión esté activa. Más adelante se añadió la funcionalidad de descarga, para ello se crea un archivo *Metalink* compuesto por los *files* que tenga el *carro*, cada *file* esta compuesto por su título, un hash, y la URL de descarga de este y la de sus réplicas.

- **Versión 4**: En esta versión se realizó la gestión de usuarios, en la que el usuario es capaz de iniciar sesión usando *OAuth2* que Github proporciona. Una vez iniciada la sesión, su usuario es almacenado en una base de datos embebida en la máquina de java, en este caso el cliente ha optado por usar Derby en su modo embebido, por su cómodo uso con Maven.

- **Versión 5**: En esta versión se trabajó en la funcionalidad de persistir búsquedas. Para poder persistir una búsqueda el usuario ha de estar identificado. Cuando el usuario desea almacenar una búsqueda, se añaden en la base de datos sus filtros y algunos parámetros de la búsqueda. Estos datos se relacionan con la tabla de usuarios de forma que un usuario tiene 0 o más búsquedas, y una búsqueda tiene 1 usuario.

- **Versión 6**: Esta versión fue mas sencilla, puesto que ya se había trabajado con la forma de persistir datos. Se consiguió almecenar los *files* de cada *carro* junto a sus réplicas y algunos metadatos cuando un usuario hace un *check-out* del *carro*. Un usuario tiene 0 o más *carros* descargados y un *carro* tiene 1 un usuario. Un *carro* tiene 1 o más *files* y un *file* tiene 1 o más *carros*, es decir, los *carros* comparten *files*. Gracias a esto se consigue que en la base de datos no haya más *files* que los que ESGF nos proporciona.

3.3. Diagrama de Gantt

A continuación se presenta la planificación definida al principio del proyecto. En esta planificación se listan las distintas tareas a realizar junto con el tiempo que se estima que se va a tardar en completar. Planificar un proyecto es primordial para cumplir con los plazos acordados. Aunque a veces algunas tareas se alarguen, es necesario tener una planificación global del proyecto. Se ha diseñado el diagrama de Gantt de la figura 4 y las tareas de la figura 3.
Como se puede ver en la figura 4, las pruebas unitarias estaban previstas para hacerlas al finalizar el desarrollo de la aplicación, a pesar de que comúnmente se suelen hacer paralelamente al desarrollo del sistema, se decidió que se iban a hacer después de desarrollar toda la aplicación debido a su dificultad.

Planificación	**17,5 days**	**1/02/22 8:00**	**24/02/22 13:00**
Análisis de requisitos	3 days	1/02/22 8:00	3/02/22 17:00
Investigación y aprendizaje de las tecnologías	14,5 days	4/02/22 8:00	24/02/22 13:00
Desarrollo e implementación	**62 days**	**24/02/22 13:00**	**23/05/22 13:00**
Versión 1	10 days	24/02/22 13:00	10/03/22 13:00
Versión 2	7 days	10/03/22 13:00	21/03/22 13:00
Versión 3	10 days	21/03/22 13:00	4/04/22 13:00
Versión 4	10 days	4/04/22 13:00	18/04/22 13:00
Versión 5	15 days	18/04/22 13:00	9/05/22 13:00
Versión 6	10 days	9/05/22 13:00	23/05/22 13:00
Desarrollo de las pruebas	**10 days**	**23/05/22 13:00**	**6/06/22 13:00**
Test unitarios	10 days	23/05/22 13:00	6/06/22 13:00
Pruebas de sistema	3 days	23/05/22 13:00	26/05/22 13:00

Figura 3: Tareas a realizar.

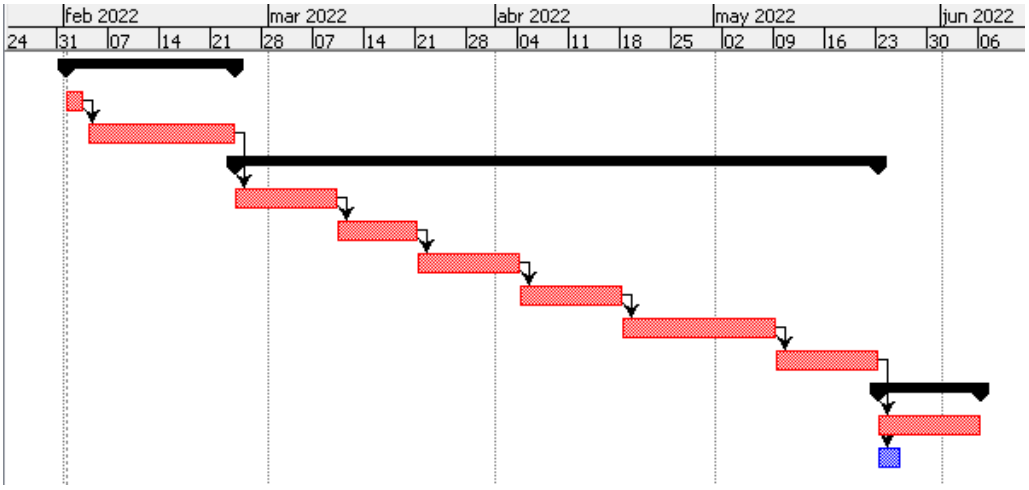

Figura 4: Diagrama de Gantt.

4. Análisis y especificación de requisitos

El problema que se pretende solucionar con este proyecto es reducir el esfuerzo que los investigadores hacen para obtener *files* y *datasets* con sus réplicas para así recuperar los datos climáticos que necesiten.

4.1. Análisis de requisitos

Los requisitos del sistema son las descripciones de lo que este debe hacer, es decir, los servicios que ofrece el sistema. Se han obtenido los siguientes requisitos a partir de la descripción del problema y los casos de uso que desea el cliente.

4.1.1. Actores del sistema

Los actores del sistema son los tipos de usuario que harán uso de alguna manera del sistema. Es importante identificarles de cara a hacer un análisis de requisitos.

Actor	Descripción
User	Tiene permiso para ejecutar todas las acciones posibles sobre el contenido del sistema, excepto la gestión de usuarios y las acciones que conllevan el uso de la base de datos, como persistir búsquedas o carros.
Registered User	Tiene permiso para ejecutar todas las acciones posibles sobre el contenido del sistema, excepto la gestión de usuarios
Admin	Funciona como un usuario registrado pero además tiene la posibilidad de gestionar usuarios.

Tabla 1: Actores del sistema.

4.2. Especificación de requisitos

Para la obtención de los requisitos se ha usado la técnica de la **entrevista** [21], en la cual se mantuvo una conversación informal para recuperar todos los requisitos principales de la aplicación. Al ser los clientes (en este caso los dos tutores del proyecto) expertos en el tema a tratar, la obtención de requisitos fue sencilla. Aun así, al usar la metodología de desarrollo incremental, a medida que se iba avanzando en la aplicación surgían más requisitos.

4.2.1. Requisitos funcionales

Los requisitos funcionales (RF) son enunciados o funciones acerca de servicios que el sistema debe proveer. Una función está compuesta por un conjunto de entradas, procesos y salidas. Los requisitos funcionales establecen el comportamiento de un sistema software. En al figura 4.2.1 se pueden observar los distintos requisitos funcionales de esta aplicación.

ID	Descripción
RF01	Todos los usuarios podrán buscar *datasets*.
RF02	Todos los usuarios podrán visualizar los *datasets* resultantes del filtro aplicado.
RF03	Todos los usuarios podrán añadir al carro tanto *datasets* como *files*.
RF04	Todos los usuarios podrán descargar *datasets*, *files*, o el contenido del carro.
RF05	Todos los usuarios podrán hacer un *check-out* del carro, para más adelante descargarlo.
RF06	El sistema no debe usar JavaScript que ejecute operaciones en el lado del cliente.
RF07	Todos los usuarios podrán guardar sus búsquedas.
RF08	Todos los usuarios podrán inicar sesión en el sistema usando la autenticación de OAuth2 (Github u ORCID).
RF09	Todos los usuarios podrán ver los metadatos de los *datasets* y *files*.

Tabla 2: Requisitos funcionales del sistema.

4.2.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) son limitaciones sobre funciones que ofrece el sistema. Estos requisitos especifican criterios que son usados para juzgar la operación de una aplicación. En al figura 4.2.2 se pueden observar los distintos requisitos no funcionales de esta aplicación.

ID	Descripción	Categoría	Importancia
RNF01	El tiempo de aprendizaje de uso del sistema por el usuario debe ser menor a 1 hora.	Usabilidad	Media
RNF02	La interfaz de usuario del sistema debe ser intuitiva para el usuario.	Usabilidad	Media
RNF03	Se debe garantizar que no se sobrecargue la memoria del sistema.	Eficiencia	Media
RNF04	Se debe refactorizar el código de manera que su mantenibilidad sea lo más sencilla posible.	Mantenibilidad	Alta

Tabla 3: Requisitos no funcionales del sistema.

5. Especificación de casos de uso

En este capítulo se muestra el funcionamiento de la aplicación. Se hace uso de diagramas de casos de uso en el que se muestran las funciones que los diferentes usuarios pueden realizar en el sistema en función de la vista en la que se encuentren. Además se describen los principales casos de uso del sistema.

5.1. Casos de uso de la vista de búsqueda

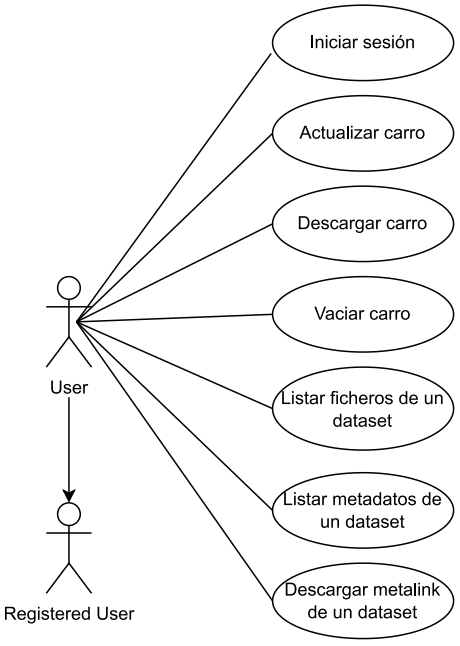

Figura 5: Vista de búsqueda para el rol User

ID	P01
Caso de uso	Iniciar sesión. RF08
Descripción	El usuario inicia sesión en el sistema.
Actores	Todos.
Precondiciones	El usuario tiene que tener una cuenta en Github u ORCID.
Flujo principal	1. El usuario pulsa el botón de inicio de sesión. 2. El usuario introduce su correo electrónico o nombre de usuario y su contraseña de Github u ORCID. 3. El sistema redirige al usuario a la página principal.
Flujo alternativo	3.1 El sistema muestra un error indicando que ha habido un error con el inicio de sesión.

Tabla 4: Caso de uso: Inicio de sesión.

ID	P02
Caso de uso	Actualizar carro. RF03
Descripción	El usuario añade o elimina elementos de su carro.
Actores	Todos.
Precondiciones	El usuario tiene que haber filtrado los *datasets* mediante los *facets*.
Flujo principal	1. El usuario selecciona uno o varios *datasets* o *files*. 2. El usuario pulsa el botón de *Update cart*. 3. El sistema añade los elementos seleccionados al carro.

Tabla 5: Caso de uso: Actualizar carro.

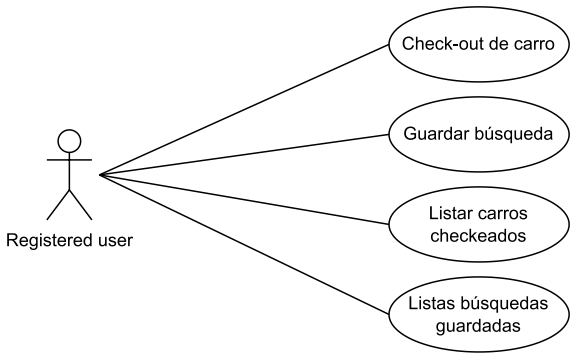

Figura 6: Vista de búsqueda para los usuarios registrados.

ID	P03
Caso de uso	*Check-out* del carro. RF05
Descripción	El usuario hace un *check-out* del carro actual.
Actores	Registered user y Admin.
Precondiciones	El usuario tiene que haber iniciado sesión en el sistema.
Flujo principal	1. El usuario usa los *facets* para filtrar los *datasets*. 2. El usuario añade *datasets* o *files* a su carro. 3. El usuario pulsa el botón de *Check-out cart*. 4. Se guarda el carro en la base de datos.
Flujo alternativo	4.1 El sistema muestra un error indicando que el usuario no ha añadido ningún elemento al carro.

Tabla 6: Caso de uso: *Check-out* del carro.

ID	P04
Caso de uso	Listar carros *checkeados*. RF05
Descripción	El usuario visualiza la lista de los carros que ha checkeado.
Actores	Registered user y Admin.
Precondiciones	1. El usuario tiene que haber iniciado sesión en el sistema. 2. El usuario tiene que haber hecho *check-out* de al menos un carro.
Flujo principal	1. El usuario pulsa el botón de *Saved carts*. 2. El sistema muestra una lista con los carros *checkeados* por ese usuario.
Flujo alternativo	2.1 El sistema muestra un error indicando que el usuario no tiene carros *checkeados*.

Tabla 7: Caso de uso: Listar carros *checkeados*.

ID	P05
Caso de uso	Listar búsquedas guardadas. RF07
Descripción	El usuario visualiza la lista de las búsquedas que ha guardado.
Actores	Registered user y Admin.
Precondiciones	1. El usuario tiene que haber iniciado sesión en el sistema. 2. El usuario tiene que haber guardado una búsqueda.
Flujo principal	1. El usuario pulsa el botón de *Saved searches*. 2. El sistema muestra una lista con las listas guardadas por ese usuario.
Flujo alternativo	2.1 El sistema muestra un error indicando que el usuario no tiene búsquedas guardadas.

Tabla 8: Caso de uso: Listar búsquedas guardadas.

5.2. Casos de uso de la vista de carros guardados

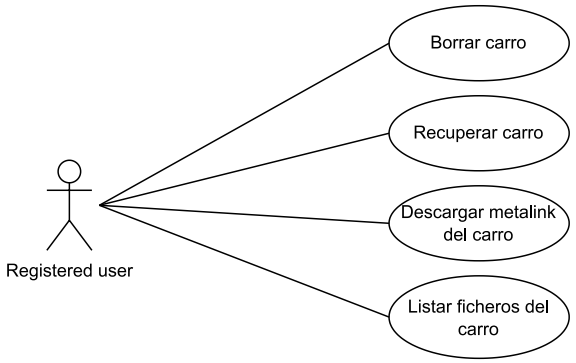

Figura 7: Lista de carros guardados para los usuarios registrados.

ID	P06
Caso de uso	Descargar *Metalink* del carro . RF04
Descripción	El usuario se descarga el archivo de descarga de un carro chequeado.
Actores	Registered user y Admin.
Precondiciones	1. El usuario tiene que haber iniciado sesión en el sistema. 2. El usuario tiene que haber hecho *check-out* de al menos un carro.
Flujo principal	1. El usuario pulsa el botón de *Saved carts*. 2. El usuario elige el carro y pulsa el botón de descarga. 3. Se descarga el archivo *Metalink* de descarga del carro.
Flujo alternativo	3.1 El sistema muestra un error indicando que la descarga ha fallado.

Tabla 9: Caso de uso: Descarga de carro checkeado.

5.3. Casos de uso de la vista de búsquedas guardadas

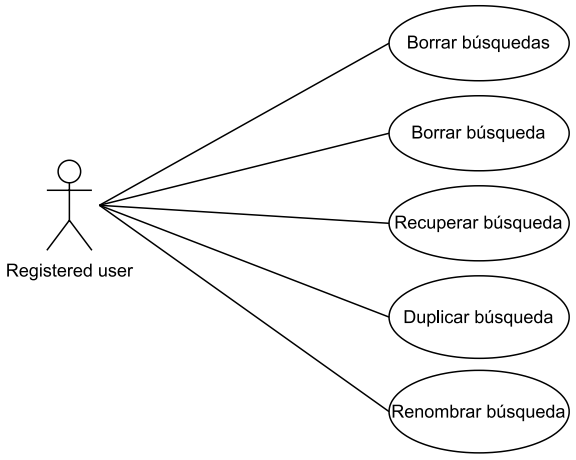

Figura 8: Lista de carros guardados para los usuarios registrados.

ID	P07
Caso de uso	Renombrar búsqueda. RF07
Descripción	El usuario renombra una búsqueda.
Actores	Registered user y Admin.
Precondiciones	1. El usuario tiene que haber guardado 1 o más búsquedas.
Flujo principal	1. El usuario pulsa el botón de *Saved searches*. 2. El usuario escribe un nuevo nombre. 3. El usuario pulsa en el botón de *Renombrar* 4. El sistema recupera la búsqueda persistida, la cambia el nombre y la vuelve a persistir.

Tabla 10: Caso de uso: Renombrar búsqueda.

5.4. Casos de uso de la gestión de usuarios

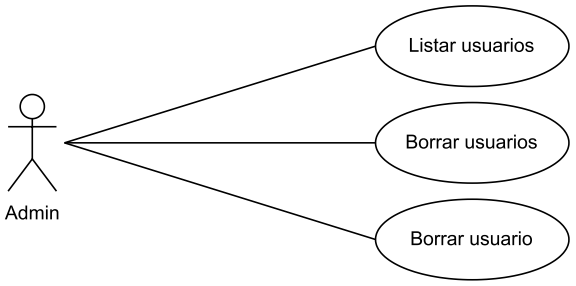

Figura 9: Gestión de usuarios por parte del administrador

ID	P08
Caso de uso	Listas usuarios.
Descripción	El administrador lista los usuarios.
Actores	Admin.
Flujo principal	1. El administrador selecciona la opción de listar usuarios. 2. El sistema recupera los usuarios persistidos y los muestra.

Tabla 11: Caso de uso: Listar usuarios.

6. Entornos de trabajo

En esta sección se va a explicar como se han creado los entornos de trabajo.

Para desarrollar la aplicación se ha usado Spring Boot junto con la base de datos Derby.
En el entorno de **desarrollo** la aplicación se ejecuta en el contexto *esgf* a través de localhost y en el puerto 8080. La base de datos se usa en modo embebido por comodidad, pero puede ser usada en modo cliente/servidor en el modo de desarrollo. La aplicación de Github usada como proveedor es la de modo desarrollo.
En el entorno de **producción** la aplicación usa siempre Derby en su modo embebido y la aplicación de Github está enlazada con la dirección del entorno de producción.

Entorno	**URL**
Desarrollo	http://localhost:8080/esgf/search
Producción	https://data.meteo.unican.es/esgf/search

Tabla 12: URLs de las aplicaciones.

6.1. Configuración de la aplicación

Los archivos de configuración que se han usado son:

- **application.properties**

  Se define la configuración de la base de datos embebida o en modo cliente/servidor. Dependiendo del perfil que esté activo se usará una configuración u otra.
  Al trabajar en una aplicación empresarial, se hace necesario un entorno de pruebas o desarrollo y otro entorno de producción. Cada entorno tiene una configuración de bases de datos diferente, URLs diferentes y podría usar parámetros distintos.
  Spring nos ofrece una manera de gestionar este problema mediante el uso de perfiles.
  Hay distintas maneras de usar perfiles con Spring, pero en este caso se ha optado por la opción de utilizar un archivo de configuración que define las propiedades de cada perfil y establece el perfil activo en cada momento. [22]
  En la figura 12 se puede ver el archivo *application.properties*, en la parte superior se definen los posibles perfiles, y a continuación se establecen las propiedades para cada perfil.

```
# ---- Activate production or dev ----- #
spring.profiles.active=production
#spring.profiles.active=dev
#spring.profiles.active=remoteDerby
# ---- -------------------------- ----- #
#--- Configuration for production profile
spring.config.activate.on-profile=production
spring.datasource.url=jdbc:derby:directory:mydbdirectory;create=true
spring.jpa.properties.hibernate.dialect=
org.hibernate.dialect.DerbyTenSevenDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.open-in-view=false
#--- Configuration for production profile ---#

#--- Configuration for remoteDerby profile ---#
spring.config.activate.on-profile=remoteDerby
spring.datasource.url=jdbc:derby://localhost:1527/userdb
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driver-class-name=
org.apache.derby.jdbc.ClientDriver
#--- Configuration for development profile ---#
```

Figura 10: Configuración de Derby.

- **application.yaml**

  En este archivo se configuran las claves de la aplicación que permiten usar Github como proveedor de OAuth2. En el entorno de desarrollo se usa una aplicación que se despliega en local, mientras que el entorno producción, la aplicación se despliega en otra URL (URLs definidas en la tabla 12). Entonces son necesarias dos aplicaciones en Github, ya que es necesario saber la dirección de retorno y es diferente en los dos modos. Como se puede ver en la figura 11, en este caso se hace uso de la configuración en yaml, en la cual usando la dependencia de Spring Security, solo es necesario incluir el **client ID** y el **client secret** de la aplicación de Github para obtener el acceso.

```yaml
spring.profiles: dev
spring:
  security:
   oauth2:
     client:
       registration:
         github:
           clientId: XXXXX
           clientSecret: XXXXX

---

spring.profiles: production
spring:
  security:
   oauth2:
     client:
       registration:
         github:
           clientId: XXXXX
           clientSecret: XXXXX
```

Figura 11: Configuración de OAuth2 usando Github.

- **roles.json**

  Este archivo es usado para almacenar los diferentes proyectos junto a sus *facets* que podrán ser elegidos en la aplicación.

```
[
   {
      "rol":{
         "name":"CMIP6",
         "facets":[
            {"activity_id": "Activity id"},
            {"data_node" : "Data node"},
            {"source_id":"Source id"},
            {"institution_id":"Institution id"},
            {"source_type":"Source type"},
            {"experiment_id":"Experiment id"},
            {"sub_experiment_id":"Sub Experiment id"},
            {"nominal_resolution":"Nominal resolution"},
            {"variant_label":"Variant label"},
            {"grid_label":"Grid label"},
            {"table_id":"Table id"},
            {"variable_id":"Variable id"},
            {"frequency":"Frequency"},
            {"realm":"Realm"}
         ]
      }
   },
```

Figura 12: Configuración de proyectos junto a sus *facets*.

6.2. Despliegue

Para desplegar la aplicación es necesario crear el .war del proyecto, en este caso, esgf.war, y introducirlo en el tomcat del servidor de data.meteo.unican.es.
Se requiere tener tomcat instalado y Java2EE en el entorno de despliegue. En caso de querer usar la aplicación con la base de datos Derby en modo cliente/servidor, también sería necesario tener Derby instalado en la máquina, puesto que en modo embebido se ejecuta en la máquina de JAVA, con añadir la dependencia de Derby al archivo pom.xml de Maven es suficiente.

7. Diseño e implementación

En el siguiente apartado se describe el diseño utilizado en diferentes niveles y la implementación de cada capa que constituye el sistema. En la figura 13 se muestra la estructura del proyecto.
En color naranja se presentan los paquetes y clases que componen la aplicación, en concreto la clase **EsgfSearchApplication.java** es la encargada de lanzar la aplicación y se sitúa en un nivel superior para ser capaz de encontrar el resto de clases. En color amarillo se destaca la carpeta **resources**, que es la encargada de almacenar los archivos de configuración de la aplicación, y por último, en color verde se encuentra la carpeta **webapp/WEB-INF**, que es la indicada para almacenar las librerías externas y lo necesario con la capa de presentación, es decir, las vistas y el estilo.

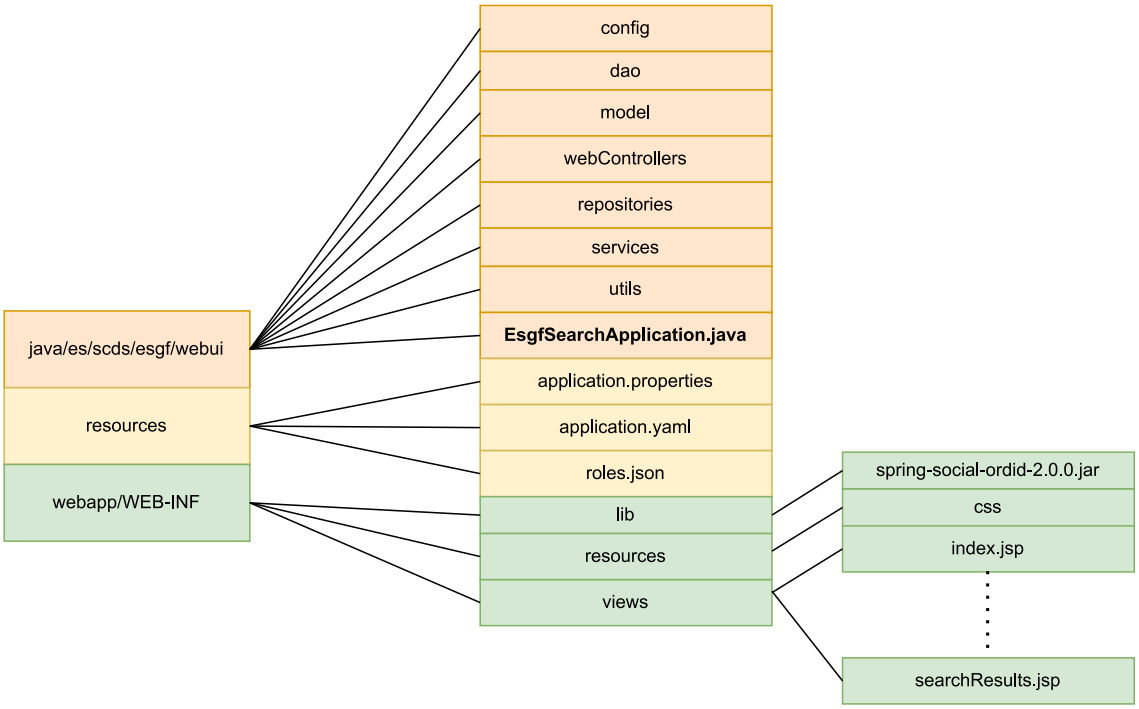

Figura 13: Estructura del proyecto

7.1. Diseño arquitectónico

El diseño arquitectónico describe la estructura y organización de alto nivel, es decir, los subsistemas o componentes y sus relaciones [23]. Representa el enlace entre la especificación de requisitos y el diseño.

Para este sistema se ha optado por la arquitectura en tres capas, es una arquitectura de software que separa las aplicaciones en tres niveles: presentación, es decir, la interfaz de usuario, aplicación o negocio, dónde se procesan los datos y la capa de datos, donde se almacenan y gestionan los datos de la aplicación.

La capa de presentación corresponde con una interfaz de usuario desarrollada con JSP, CSS y Bootstrap, la capa de negocio corresponde a los controladores y servicios, y la capa de datos corresponde a la API de SOLR y a una base de datos Derby embebida en la máquina virtual de Java.

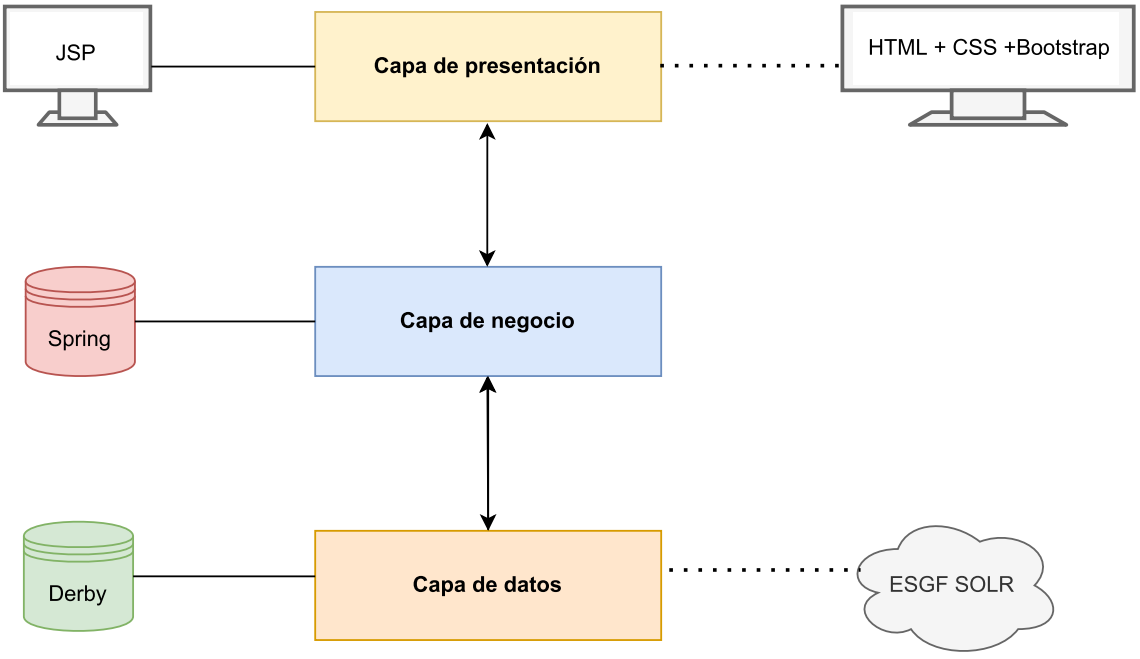

Figura 14: Diseño arquitectónico

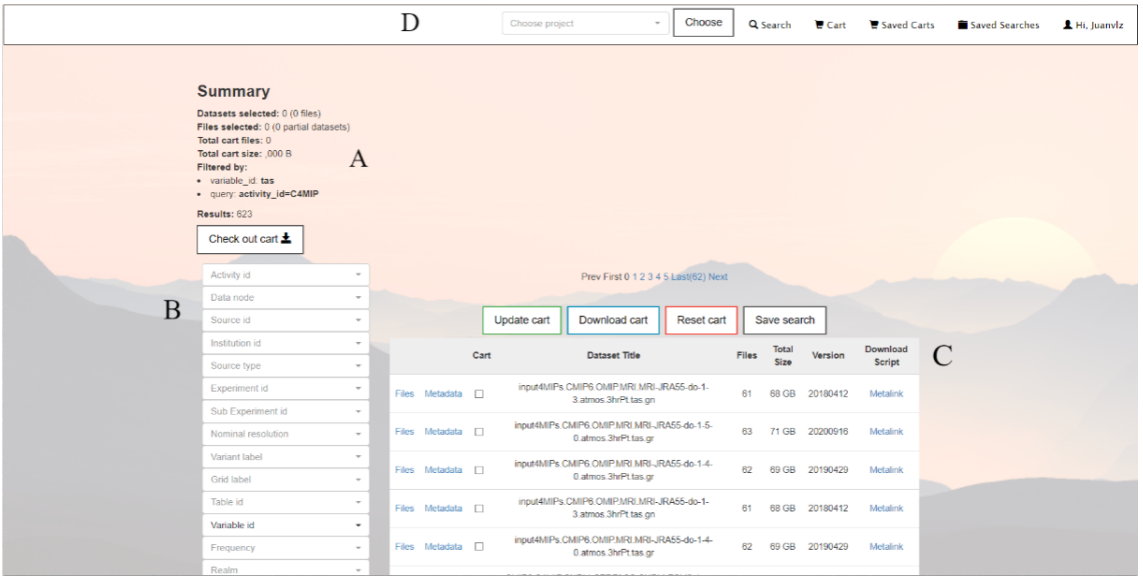

Figura 15: Vista principal

7.2. Capa de presentación

La capa de presentación se hace cargo de la comunicación entre el usuario final y el propio sistema. En este caso proporciona una interfaz de usuario en la cuál se pueden llevar a cabo todas las acciones desarrolladas anteriormente en los casos de uso. Este sistema se divide principalmente en 5 vistas: elección de filtros para realizar la búsqueda, lista de resultados, vista del carro actual, lista de búsquedas guardadas y lista de carros guardados por el usuario.
Desde cualquiera de las vistas, si el usuario ha iniciado sesión el sistema, es capaz de ver su historial de búsqueda y sus carros que han sido checkeados para una futura descarga.
Adicionalmente, el usuario siempre puede ir a la pestaña de búsqueda estando el enlace en el botón *Search* situado en el *header*, en la parte superior de la pantalla.

Como se puede ver en la figura 15, en la seccion marcada con una **A**, se le muestra al usuario un breve resumen del actual carro y de la búsqueda realizada, además de un botón para hacer un *check-out del carro*. En la sección maracada con una **B**, el usuario puede volver a filtrar para realizar una nueva búsqueda, y por último, en la sección **C**, se le muestra al usuario una tabla con los resultados de la búsqueda. Asimismo, es esta sección el usuario puede usar los botones para actualizar el carro, descargar el carro, borrarle o guardar la búsqueda, además de cambiar de página. Por último, en la sección **D**, aparece la opción de cambiar de proyecto, de volver a la ventana de búsqueda, de ir a la pestaña del carro, de ver los carros guardados y de ir a la vista de las búsquedas guardadas.

7.2.1. Vista de elección de filtros

En esta vista, el usuario selecciona los *facets* y parámetros que considera para realizar una nueva búsqueda. Cuando haya acabado, pulsa el botón de *Search*.

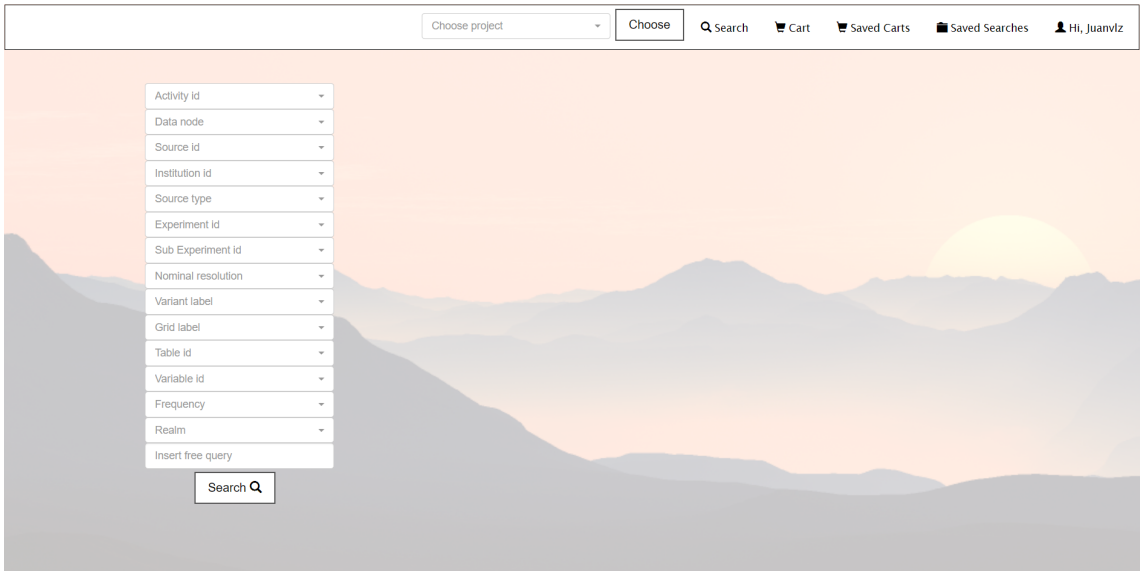

Figura 16: Vista de elección de filtros

7.2.2. Vista del resultado de la búsqueda

En esta vista, el usuario puede ver los *datasets* resultantes de la búsqueda en forma de tabla, esta tabla está compeusta por una paginación y 4 botones que permiten hacer acciones sobre la búsqueda. Al usuario se le permite añadir *datasets* y *files* al carro, descargarse el carro, hacer un *check-out* del carro, vaciar el carro o guardar la búsqueda en el historial de búsquedas. Además puede volver a buscar usando filtros diferentes. Asimismo, el usuario tiene la posibilidad de ver los metadatos de cada *dataset* y los ficheros que componen los *datasets*. También puede descargar directamente el archivo de descarga de cada *dataset*.

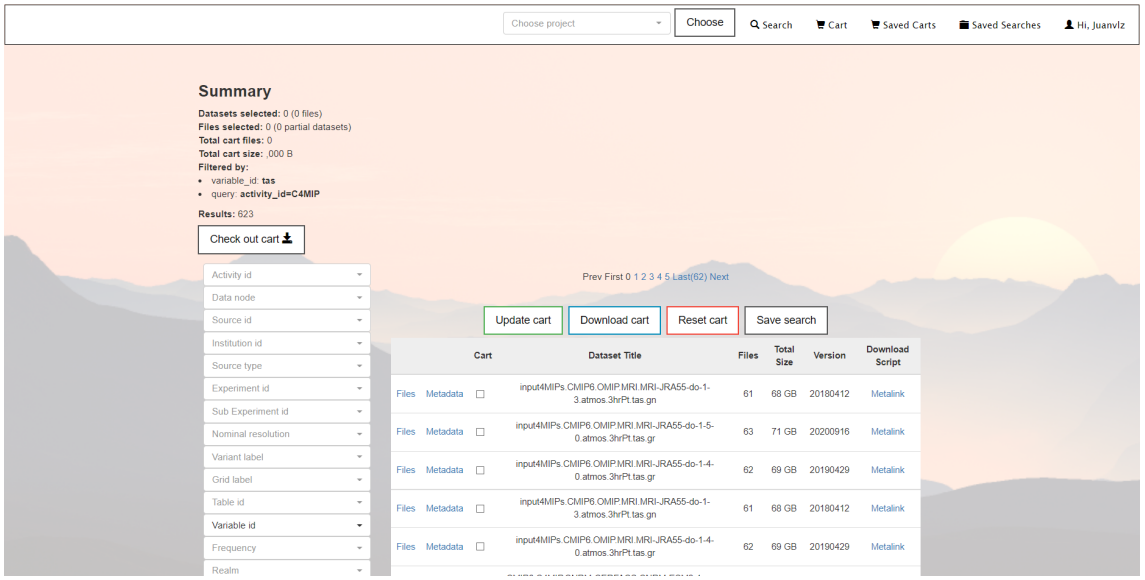

Figura 17: Vista de lista de resultados.

7.2.3. Vista de los *files* de un *dataset*

En esta vista, el usuario puede ver en forma de tabla los ficheros que componen un *dataset*, en la parte superior qde la tabla se encuentra un botón que permite al usuario añadir o eliminar *files* del carro. Además, el usuario es capaz de descarga el archivo de descarga de ese fichero y de ver los metadatos del fichero que desee.

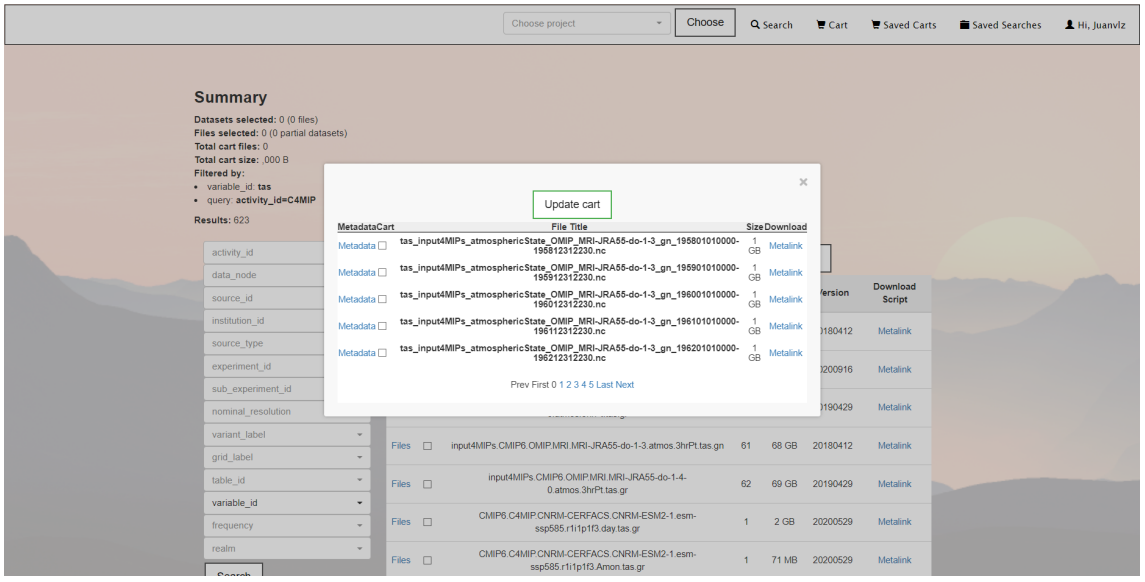

Figura 18: Vista de lista de ficheros de un *dataset*.

7.2.4. Vista de los metadatos de un *dataset*

En esta pestaña, el usuario puede ver en un *pop-up*, los metadatos del *dataset* escogido, además, puede añadir o eliminar ese *dataset* del carro. Asismismo, el usuario puede descargar el fichero de descarga de ese *dataset*.

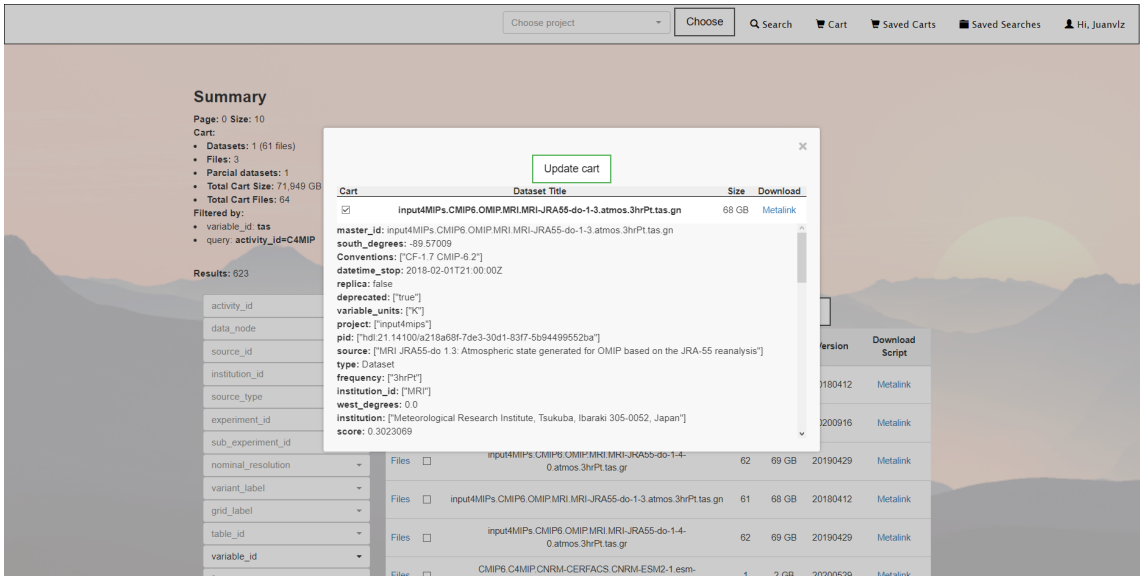

Figura 19: Vista de los metadatos de un *dataset*.

7.2.5. Vista del carro

En esta página, el usuario puede ver los elementos que ha añadido en el carro, tanto *datasets* como ficheros y que están persistidos en la sesión, es decir, una vez finalice la sesión, el carro se vaciará.
El usuario puede ver los metadatos de los *datasets* y descargar el archivo de descarga, tanto del carro al completo, es decir, un archivo de descarga que contiene todos los elementos de conforman el carro, o archivos de descarga independientes. Además, el usuario es capaz de actualizar el carro o de vaciarle al completo.

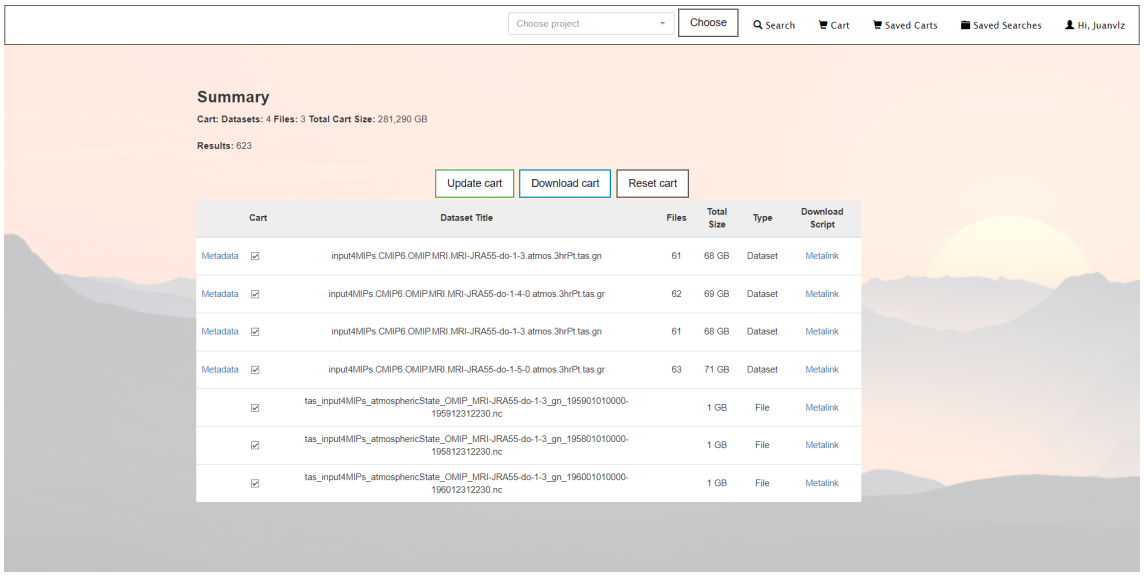

Figura 20: Vista del carro.

7.2.6. Vista de la lista de búsquedas guardadas

Esta vista es un historial de búsquedas, en ella aparece en forma de tabla un historial de las búsquedas filtradas que el usuario guardó.
El usuario tiene la posibilidad de recuperar la búsqueda y continuar buscando, también puede duplicar la búsqueda para cambiarla pero dejando la anterior guardada, cambiar el nombre de la búsqueda y eliminarla.
Cuando un usuario quiere recuperar la búsqueda, se le redirige a la pestaña de resultados con la búsqueda que fue persistida anteriormente.

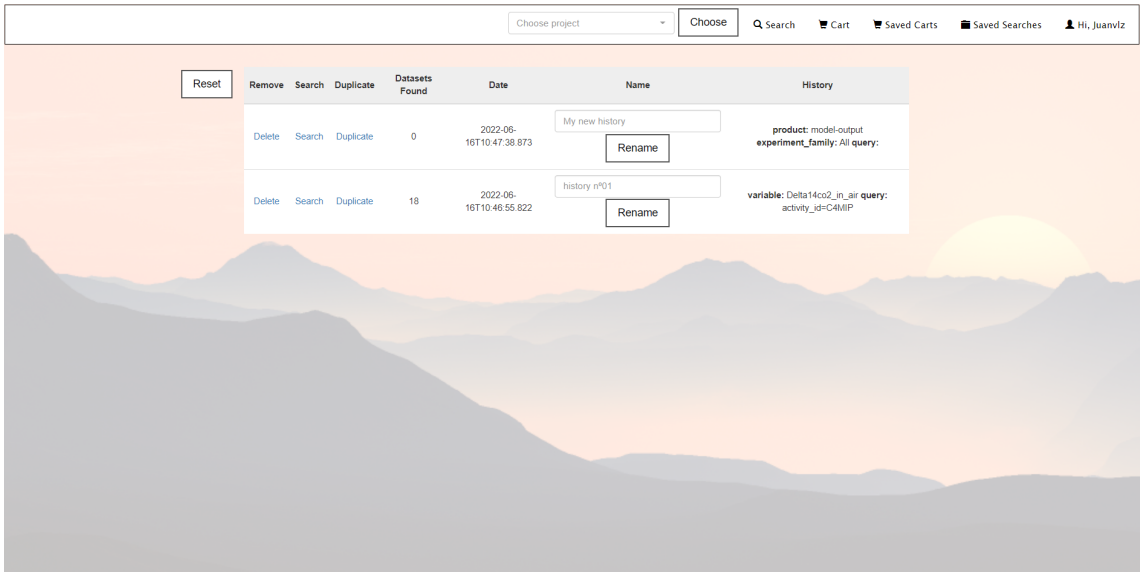

Figura 21: Vista de las búsquedas guardadas.

7.2.7. Vista de la lista de carros guardados

En la presente vista, el usuario puede observar una lista de los carros que ha ido guardando durante el tiempo, es decir, de los carros que ha persistido.
Al usuario se le ofrece la posibilidad de descargar el carro más rápidamente, ya que el proceso de conseguir las replicas se hizo en el momento de guardarlo, puede recuperar el carro para poder seguir añadiéndole ficheros, también puede borrarle y ver los ficheros que componen el carro.

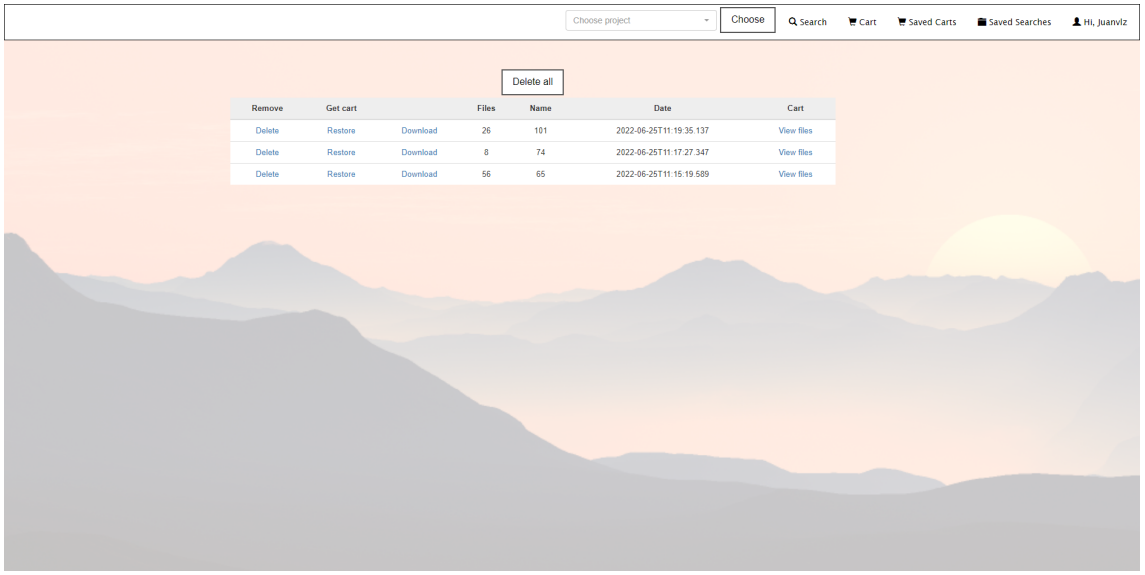

Figura 22: Vista de los carros guardados.

7.3. Capa de negocio

En la capa de negocio es donde se encuentra desarrollada la funcionalidad del sistema. Esta capa recibe mensajes desde la capa de presentación y responde con resultados que puedes ser obtenidos de la capa de datos. Esta capa se ha desarrollado con Spring Boot [24] y contiene toda la lógica de negocio, está compuesta por controladores y servicios. Todas las vistas descritas en la capa de presentación se comunican con esta capa para obtener los datos correspondientes. Dependiendo de la función que sea usará un controlador u otro.
Los controladores están anotados con @Controller, que es simplemente una especialización de la clase @Component, que le permite a Spring auto-detectar las clases de implementación a través del escaneo del classpath.
Para inyectar los servicios en los controladores, han sido anotados con @Service, y en el controlador se declaran con la anotación @Autowired, que permite inyectar una dependencia en otra.
A continuación se describen los 4 controladores que componen la capa de negocio.

- **SearchController**
  Este controlador se encarga de las principales páginas del servicio, desde la elección de un proyecto hasta la vista de resultados.

- **SavedSearchesController**
  Este controlador está compuesto por los métodos que tienen que ver con el historial de búsquedas, exactamente las funcionalidades de: mostrar el historial, borrar el historial, duplicar, renombrar, eliminar y recuperar una búsqueda.

- **CartController**
  Este controlador se ocupa de todo lo relacionado con los carros, es decir, mostrar la lista de carros, recuperar, descargar, borrar, listar ficheros y hacer un *check-out* del carro. Además de mostrar los metadatos de los *datasets* del carro.

- **UsersController** Los métodos de este controlador solo pueden ser usados por usuarios con rol ADMIN, estos métodos son: mostrar una lista de los usuarios, borrar los usuarios y recuperar un usuario.

Se han usado 7 servicios, 6 de ellos son una capa intermedia entre los repositorios y los controladores y simplemente tienen los métodos *delete*, *update*, *get* y *getAll*, mientras que el servicio principal que se usa para obtener todos los de datos de la base de datos de ESGF, es decir, el servicio que hace peticiones a la API de ESGF es diferente.

- **SearchService**
  Como todos los servicios, es anotado con @Service y tiene los métodos:

  - getFacets(String uri)
  - getDatasets(String uri, int offset, int limit)
  - getDocs(int offset, int limit, String datasetId)
  - getReplicas(FileDataset file)

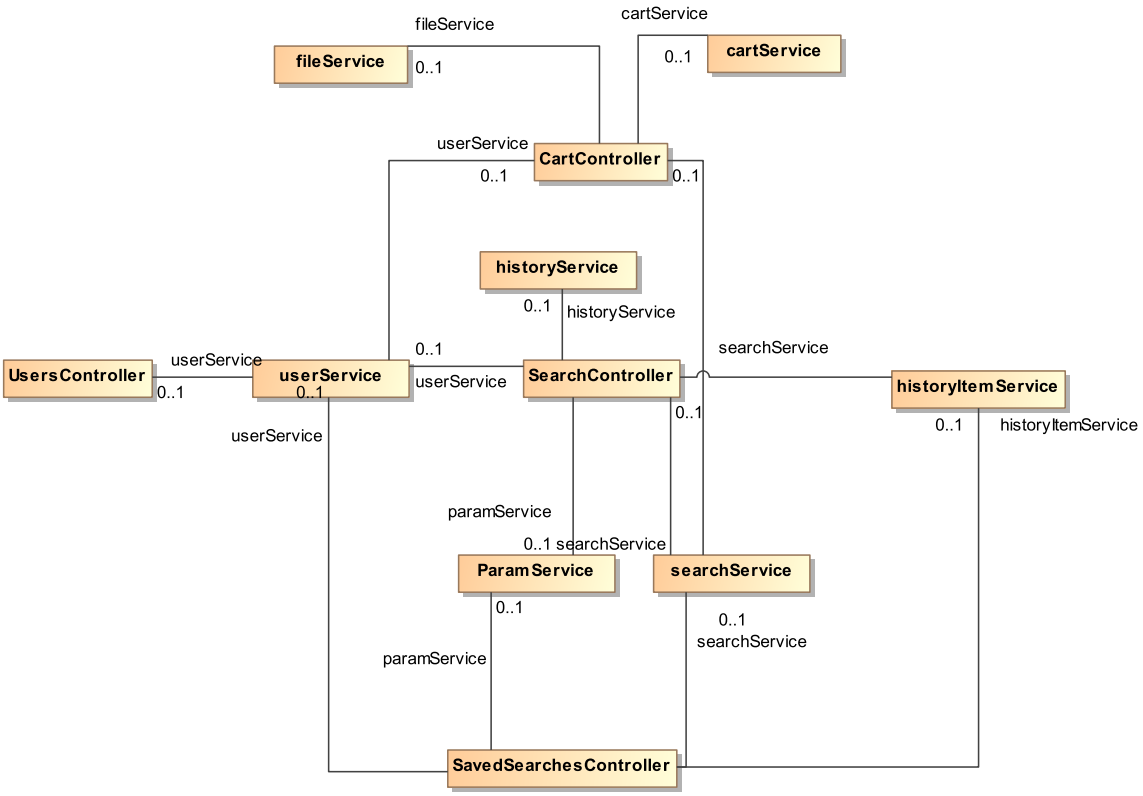

Figura 23: Diagrama de clases de los controladores y servicios.

7.4. Capa de datos

En este apartado se explicará la capa de datos implementada en el sistema.
Se usan dos formas de persistir o usar información persistida.

- **The ESGF Search RESTful API** [25]
  Esta base de datos la proporciona ESGF para acceder a datos climáticos almacenados en diferentes nodos. Usa Solr como motor de búsqueda, permitiéndonos así hacer filtrado de búsquedas y hacer búsquedas por facetas.
  En este proyecto se usa principalmente para obtener tanto los *facets* que nos permiten filtrar datos, como para obtener los datos resultados de una búsqueda por facetas.
  Para acceder a esta base de datos se usa un servicio ofrecido por ESGF, en el cual el cliente forma una petición que es procesada por la máquina de Solr, retornando una respuesta en formato XML o JSON.

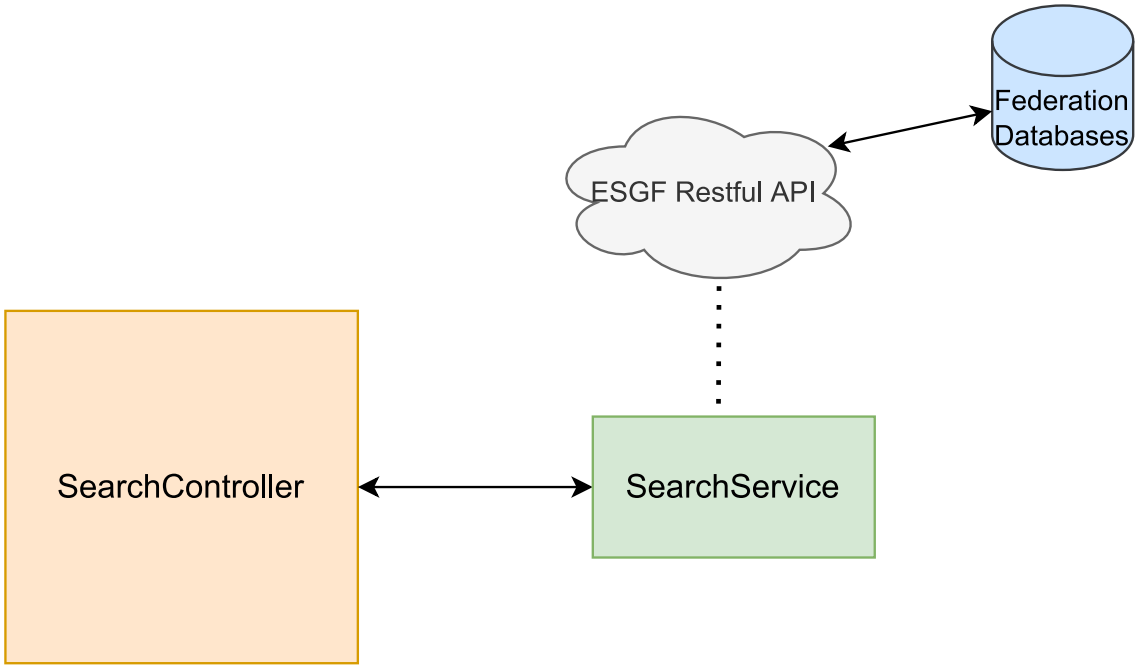

Figura 24: ESGF Search RESTful API

- **Apache Derby** [6]
  Esta base de datos se usa en modo embebido. De esta manera el motor de Derby no se ejecuta en un proceso diferente, sino que en su lugar, Derby se ejecuta dentro de la misma máquina virtual de Java (JVM) que al aplicación.

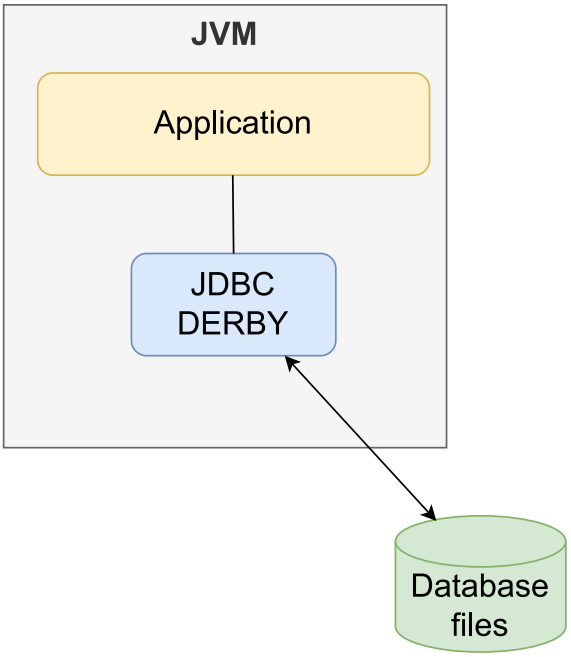

Figura 25: Embebbed Derby

Para gestionar esta base de datos se usa Spring Data JPA [26], parte de Spring Data, que facilita la implementación de repositorios basados en JPA. En esta base de datos se guarda exclusivamente datos de los **usuarios** que inician sesión en el sistema. En concreto, se almacena un identificador único, su nombre de usuario, su nombre completo y el nombre del proveedor de estos datos, es decir, Github [27] u ORCID [28]. Además, el usuario puede persistir búsquedas a modo de historial y hacer *check-outs* de carros.
De una **búsqueda** se persiste: un identificador único, un nombre, el número de documentos encontrados en la búsqueda, la fecha en la que se guardó la búsqueda y los filtros aplicados.
De un **carro** se almacena: un identificador único, la fecha en la que se hizo el *check-out*, y los documentos que componen el documento con los metadatos necesarios para realizar la descarga del carro más adelante.

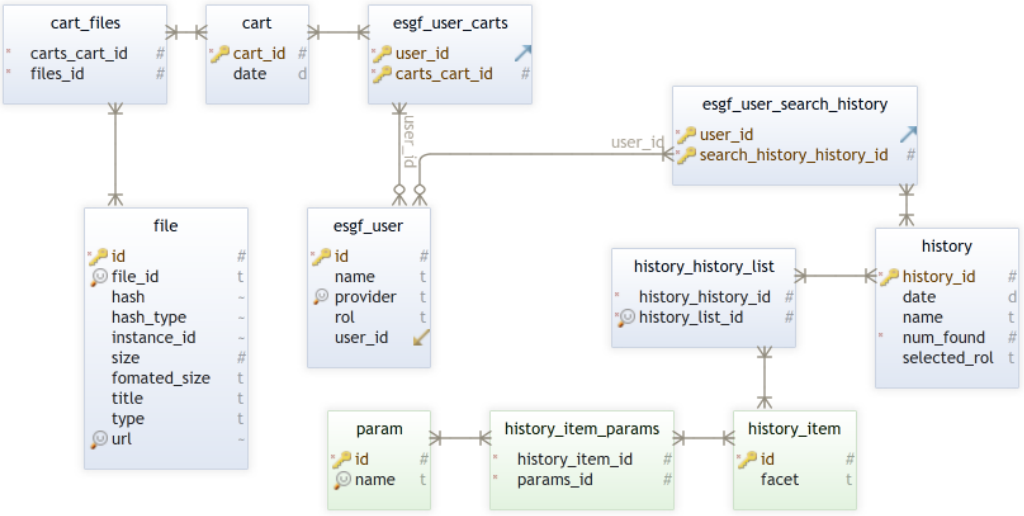

Figura 26: Diagrama de entidades de la base de datos relacional Derby generado con DbSchema.

Para configurar la base de datos desde Spring, se han anotado las clases que serán entidades en la base de datos con @Entity, las clases con esta anotación deben tener un atributo anotado con @Id, que hará de clave primaria y un constructor vacío. Los atributos se mapean automáticamente a la entidad, pero pueden ser modificados con la anotación @Column.
Para mapear las listas, se usan las anotaciones: @OneToMany, @ManyToMany o @ManyToOne.

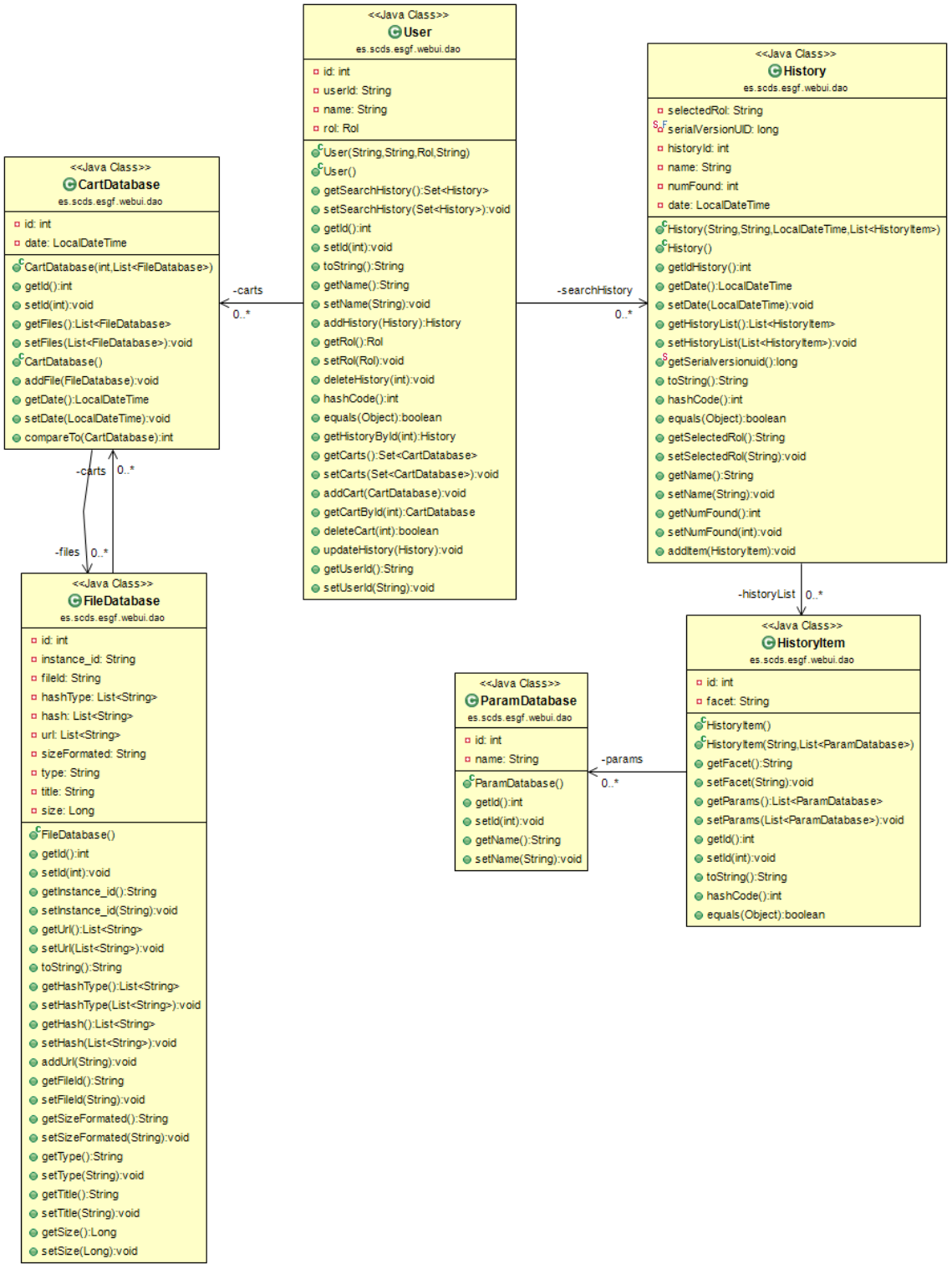

Figura 27: Diagrama de clases de la capa DAO generado con ObjectAid.

8. Pruebas y resultados

En esta sección se va a hablar sobre las pruebas realizadas sobre el proyecto para verificar el correcto funcionamiento. Cabe destacar que al tratarse de una tecnología desconocida, las pruebas unitarias han sido una parte bastante complicada. Se han tratado las pruebas unitarias, de sistema, y de aceptación.

8.1. Pruebas unitarias

Las pruebas unitarias consisten en aislar una parte de código y comprobar su correcto funcionamiento. Pueden ser de una clase o de un método.
Han sido desarrolladas al final del proyecto puesto que en cada versión incremental se realizaban pruebas de aceptación. Se ha usado Spring Testing para probar el correcto funcionamiento de la aplicación y se ha comprobado que los controladores funcionan correctamente, es decir, que redirigen al usuario a las correctas vistas.
Se han probado los métodos del controlador **SearchController** haciendo uso de la clase de ayuda **ModelAndViewAssert**, la cual nos permite simplificar escenarios de prueba que tratan con Spring Web MVC y ModelAndView. Con esta clase se pueden comparar elementos como los atributos que contiene la vista o el nombre de la vista. [29]

8.2. Pruebas de Sistema

En las pruebas de sistema se desea comprobar que la aplicación cumple con los requisitos no funcionales descritos al comienzo del proyecto.

8.2.1. Pruebas de usabilidad

En este apartado se sea verificar que se cumplan los requisitos RNF01 y RNF02, es decir, que el tiempo de aprendizaje de la aplicación sea menor a una hora y que la interfaz de usuario sea intuitiva. Para realizar esta prueba, se ha dejado que una persona no familiarizada con la aplicación tratase de entender las diferentes funciones que componen la aplicación y el resultado ha sido correcto.

8.2.2. Pruebas de rendimiento

Se ha comprobado como rinde la aplicación a eventos en los que se considera que la cantidad de objetos a persistir en la base de datos es enorme y no se aprecia ningún problema.
El único problema que hay que arreglar en un futuro es cuando una petición es tan grande que genera un *time-out* y hace que se reinicie la sesión, esto se arreglará generando una tarea asíncrona que se encargue de persistir los datos.

8.3. Pruebas de aceptación

Las pruebas de aceptación han sido realizadas en cada reunión por los dos tutores del proyecto. En las reuniones, se comprobaba que el trabajo que estaba pendiente de ser realizado funcionase.

Al final del proyecto se ha probado la aplicación ya desplegada y así se verificó que los objetivos que el proyecto tenía en un principio se cumplieran.

9. Conclusiones y trabajos futuros

En esta sección se tratara de explicar el cumplimiento o no de los objetivos iniciales, de los problemas que ha habido, y de lo que queda por hacer en un futuro.
Como objetivo principal, en este proyecto se debía desarrollar un servicio web que ofreciese a los investigadores la posibilidad de filtrar *datasets* y obtener resultados para su descarga. Este objetivo se ha cumplido, así como el resto de objetivos que hacen del sistema una aplicación completa y funcional.

Debido a que la tecnología a utilizar, es decir, Spring, era nueva para mí, al principio los problemas eran más sobre como hacer conseguir que se vieran las vistas o como comunicar las vistas con el controlador. Por ejemplo, el primer problema real con el me encontré fue la manera de conseguir que desde la vista se pudiera mandar al controlador todos los filtros aplicados usando solo los mecanismos que JSP intregado con Spring nos ofrecía. Se solucionó usando un formulario sin modelo de Spring. Otro gran escollo que tuve que superar fue la manera de implementar una paginación en la cuál los datos a mostrar no están en el servidor. Este problema se debe a que los *datasets* que se muestran en la vista de resultados son consultados a través de la API de ESGF de 10 en 10 para no saturar el servidor con datos innecesarios que nunca serán mostrados. Se solucionó usando los parámetros de **offset** y **limit** que la API nos ofrece y haciendo una llamada cada vez que el usuario quiera una página diferente.

Algún que otro problema vino de la mano de la persistencia de la información en una base de datos. Al principio se consiguió implementar una base de datos que cumpliera con los requisitos pero crecía exponencialmente a medida que los usuarios añadían carros a su perfil. Se solventó el problema haciendo que los usuarios y los ficheros tengan una relación de n a n. Es decir, un usuario puede tener 0 o más ficheros y un fichero puede estar relacionado con 1 o más usuarios. Así, se consigue que como mucho en la base de datos estarán todos los ficheros que tiene ESGF en su base de datos de SOLR.

Se ha intentado seguir la planificación establecida al comienzo del proyecto, pero al surgir percances inesperados que han tomado más tiempo de lo esperado, se ha hecho difícil conseguir ajustarse a los plazos establecidos. A pesar de esto, el proyecto ha resultado exitoso y se va a seguir trabajando para dejarle perfecto.

Como trabajos futuros que se van a realizar una vez se entregue este proyecto, se va a añadir la funcionalidad de permitir a los usuarios iniciar sesión con ORCID, ya que no fue posible hacerlo durante el desarrollo puesto que era necesario que la aplicación estuviera desplegada. Además se van a corregir ciertos errores que no afectan al flujo principal de la aplicación pero que no han podido ser solventados. Asimismo, una gran mejora que se podría hacer en la aplicación sería usar Javascript y con ello un framework de Javascript que permita crear una mejor apariencia del sistema.

Del mismo modo, en un futuro se pretende desarrollar una aplicación Restful para gestionar las búsquedas desde una aplicación de consola, consiguiendo así un desacople entre la generación de las vistas JSP y la API. Esta aplicación permitiría a los investigadores acceder a los datos directamente desde sus consolas sin necesidad de usar un navegador web, aunque esta opción también estaría disponible.

Por mi parte, la realización de este proyecto me ha permitido entender muchas de las cosas que estudiamos en la carrera pero que se quedan en el aire o que no llegamos a ver en un entorno real. Me ha ayudado mucho a aprender a corregir errores y he aprendido algunas técnicas de depuración de fallos. Asimismo, me he dado cuenta que no todo sale como en un principio planificas debido a ciertas dificultades que te encuentras por el camino, las cuáles son mas difíciles de corregir que lo que uno en una primera instancia piensa. También he comprendido que aveces hay cosas que nunca se han hecho o que no son fáciles de encontrar ya desarrolladas. En estas ocasiones la realización de una función se hace más complicada puesto que igual es necesario entender o investigar a fondo una cierta herramienta. Por ejemplo, al desarrollar el inicio de sesión usando el Oauth2 de Github, se me hizo complicado que funcionase a la perfección como se requería.

En conclusión, me gustaría dar las gracias a mis dos tutores, Antonio y Zequi, por ayudarme con todo y juntos conseguir que este proyecto salga adelante. Desarrollar este proyecto me ha parecido una buena manera de dar por finalizada la carrera y de dar un uso real a muchos de los conocimientos aprendidos durante estos 4 años.

Referencias

[1] "ESGF Federation Design." [Online]. Available: https://esgf.llnl.gov/federation-design.html

[2] "Authorization Code Grant." [Online]. Available: https://docs.wso2.com/display/IS530/Authorization+Code+Grant

[3] "Metalink." [Online]. Available: https://en.wikipedia.org/wiki/Metalink

[4] R. Johnson, "Spring Framework Documentation," Jun. 2022. [Online]. Available: https://docs.spring.io/spring-framework/docs/current/reference/html/

[5] "JSP." [Online]. Available: https://en.wikipedia.org/wiki/Jakarta_Server_Pages

[6] "Apache Derby Embebbed." [Online]. Available: https://db.apache.org/derby/papers/DerbyTut/embedded_intro.html

[7] "aria2." [Online]. Available: https://github.com/aria2/aria2

[8] "OAuth." [Online]. Available: https://en.wikipedia.org/wiki/OAuth

[9] "DBSchema." [Online]. Available: https://dbschema.com/?AFFILIATE=144826&__c=1

[10] "JAVA." [Online]. Available: https://en.wikipedia.org/wiki/Java_(programming_language)

[11] "CSS." [Online]. Available: https://en.wikipedia.org/wiki/CSS

[12] "HTML." [Online]. Available: https://en.wikipedia.org/wiki/HTML

[13] "JSON." [Online]. Available: https://es.wikipedia.org/wiki/JSON

[14] "XML." [Online]. Available: https://es.wikipedia.org/wiki/Extensible_Markup_Language

[15] "YAML." [Online]. Available: https://www.redhat.com/es/topics/automation/what-is-yaml

[16] J. Richer, A. Sanso, and I. Glazer, *OAuth 2 in action*. Shelter Island, NY: Manning Publications, 2017, oCLC: ocn945947689.

[17] "SpringAndOauth2." [Online]. Available: https://spring.io/guides/tutorials/spring-boot-oauth2/

[18] E. D. Hardt, "OAuth2.0," Oct. 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749

[19] "Earth System Grid Federation." [Online]. Available: https://esgf.llnl.gov

[20] I. Sommerville, *Software engineering*, 9th ed. Boston: Pearson, 2011, oCLC: ocn462909026.

[21] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*, 01 2010.

[22] "Spring profiles." [Online]. Available: https://docs.spring.io/spring-boot/docs/1.2.0.M1/reference/html/boot-features-profiles.html

[23] F. Ruiz, "Diseño de Software." [Online]. Available: https://www.ctr.unican.es/asignaturas/is1/is1-t04-trans.pdf

[24] "Spring." [Online]. Available: https://spring.io

[25] "The ESGF Search RESTful API." [Online]. Available: https://esgf.github.io/esg-search/ESGF_Search_RESTful_API.html

[26] "Spring Data JPA." [Online]. Available: https://spring.io/projects/spring-data-jpa

[27] "Github OAuth." [Online]. Available: https://docs.github.com/es/developers/apps/building-oauth-apps/authorizing-oauth-apps

[28] "API Tutorial: Get an Authenticated ORCID iD." [Online]. Available: https://info.orcid.org/documentation/api-tutorials/api-tutorial-get-and-authenticated-orcid-id/

[29] "ModelAndViewAssert." [Online]. Available: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/test/web/ModelAndViewAssert.html