INSTITUTO DE EDUCACIÓN SECUNDARIA SERPIS

Extracción y gestión de ofertas laborales de varias fuentes

Proyecto de Desarrollo de Aplicaciones Multiplataforma

Ciclo Formativo Desarrollo de Aplicaciones Multiplataforma
Departamento de informática

Autor: Santo Manresa, José Manuel
Tutor: Badal Carsi, Sergio
Curso: 2022/2023

Resumen

El proyecto ‘*Extracción y gestión de ofertas laborales de varias fuentes’* consta de dos partes claramente diferenciadas, una en la que se obtiene la información, y la otra en la que se presenta en una aplicación para que se haga uso de ella. Más concretamente, durante la primera parte, se extrae información de varias páginas webs de ofertas laborales, para procesar y guardar la que interese en nuestra propia base de datos relacional. Esta parte será invisible para el usuario final, aunque podrá configurar ciertos criterios o parámetros, de la búsqueda de ofertas. La segunda parte es la que respecta a la aplicación, que podrá manejar el usuario. En ella se mostrará la información, que se ha ido almacenando de las fuentes externas, en nuestra base de datos. Una vez cargada la información en la aplicación, el usuario podrá visualizar las ofertas, ordenarlas según la columna o criterios que desee, filtrar por distinto parámetros, etc. También contará con la posibilidad de señalar que ofertas le gustan y cuáles no, para seguir mejor las que le interesen o borrar las que no. Además podrá modificar o añadir información a estas ofertas, por ejemplo asignando una entrevista de trabajo en el caso de que surja durante el seguimiento de la oferta, con datos como la fecha y notas a tener en cuenta para prepararse para la entrevista.

Resum

El projecte 'Extracció i gestió d'ofertes laborals de diverses fonts' consta de dues parts clarament diferenciades, una en què s'obté la informació, i l'altra en què es presenta en una aplicació perquè se'n faci ús. Més concretament, durant la primera part, s'extreu informació de diverses pàgines web d'ofertes laborals, per processar i desar la que interessi a la nostra pròpia base de dades relacional. Aquesta part serà invisible per a l'usuari final, encara que podrà configurar certs criteris o paràmetres, de la recerca d'ofertes. La segona part és la que fa referència a l'aplicació, que podrà gestionar l'usuari. S'hi mostrarà la informació, que s'ha anat emmagatzemant de les fonts externes, a la nostra base de dades. Un cop carregada la informació a l'aplicació, l'usuari podrà visualitzar les ofertes, ordenar-les segons la columna o criteris que desitgi, filtrar per diferents paràmetres, etc. També comptarà amb la possibilitat d'assenyalar quines ofertes li agraden i quines no, per seguir millor les que us interessi o esborrar les que no. A més, podrà modificar o afegir informació a aquestes ofertes, per exemple assignant una entrevista de treball en el cas que sorgeixi durant el seguiment de l'oferta, amb dades com la data i notes a tenir en compte per preparar-se per a l'entrevista.

Abstract

The 'Extraction and management of job offers from various sources' project consists of two clearly differentiated parts, one in which the information is obtained, and the other in which it is presented in an application so that it can be used. More specifically, during the first part, information is extracted from various web pages of job offers, to process and save the information that interests us in our own relational database. This part will be invisible to the end user, although they will be able to configure certain criteria or parameters for the search for offers. The second part is related to the application, which the user can manage. It will show the information, which has been stored from external sources, in our database. Once the information is loaded in the application, the user will be able to view the offers, order them according to the column or criteria they want, filter by different parameters, etc. You will also have the possibility of indicating which offers you like and which you do not, to better follow those that interest you or delete those that do not. You can also modify or add information to these offers, for example assigning a job interview in the event that it arises during the follow-up of the offer, with data such as the date and notes to take into account to prepare for the interview.

Índice

Índice de imágenes

Extracción y gestión de ofertas laborales de varias fuentes

José Manuel Santo Manresa

1 - Justificación

Se trata de un proyecto muy interesante que mezcla a la vez backend y frontend, y programación de escritorio y web. Se utilizan dos lenguajes de programación distintos y multitud de herramientas con distintos propósitos que se van relacionando y encajando entre sí para conseguir un objetivo global.

Tenemos toda una fase que podríamos denominar principalmente de backend, con la creación de una base de datos relacional y un sistema de clases que a través de un ORM de python nos permite trabajar muy cómodamente con los datos desde lo que sería equivalente al servidor. Pero además cuenta con elementos de frontend ya que con el scraping se explora y analiza cómo están construidas y estructuradas las páginas web externas.

Luego la segunda fase es más similar a un frontend, ya que desarrollaremos con tecnologías web toda la parte visual e interactiva de una aplicación que funciona en un navegador. Pero aún así también cuenta con elementos de backend ya que utilizamos otro ORM, en este caso de javascript para acceder a los datos. En una aplicación web que hiciera uso de una API normalmente ya se encargaría el propio backend a través de endpoints de devolvernos la información que queramos directamente en formato json. Pero en este caso podemos modificar los datos desde ambos lenguajes y sistemas. Aunque el de python normalmente solo se encarga de guardar los nuevos datos y es el de javascript es el que se encarga de trabajar con ellos.

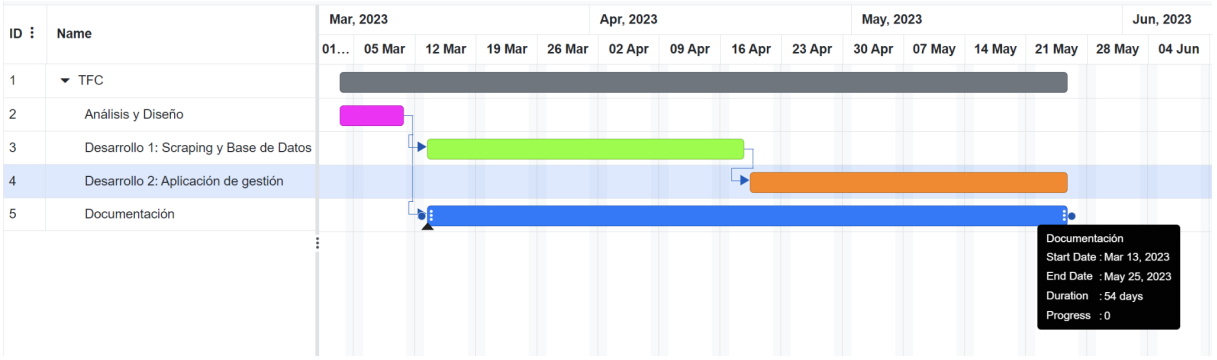
De este modo se tocan multitud de tecnologías y herramientas que además son muy utilizadas y demandadas en la actualidad.

2 - Gestión del proyecto

2.1 - Planificación inicial

Durante la planificación inicial, el proyecto se planteó como 3 grandes bloques. Primero estaría el análisis y diseño, luego el desarrollo y por último la documentación. Pero a nivel de tiempo, la documentación se pone a la par que el desarrollo, para ir documentando poco a poco los avances y no dejarlo para el final, y se comenzaría una vez estuviera claro el bloque de análisis y diseño.

A su vez el desarrollo se dividió en dos partes, por un lado la primera parte en la que se realiza el scraping y se crea la base de datos con la información obtenida, y por otro la segunda parte, que se iniciaría una vez completada la primera, en la que se desarrolla la aplicación en sí.

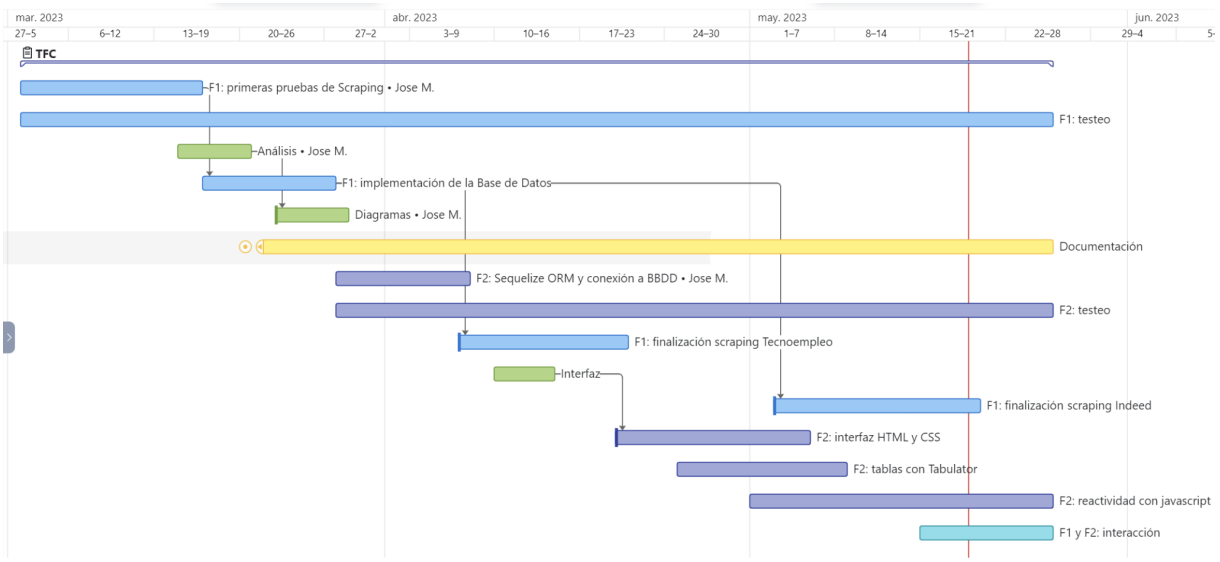


Img. 1 - Diagrama de Gantt inicial

Como es fácil de imaginar esta programación tan sencilla y con una estructura tan básica no se ha cumplido en absoluto. En el siguiente apartado explicaremos los motivos y cómo se ha gestionado finalmente el tiempo.

2.2 - Planificación desarrollada finalmente

En relativamente poco tiempo hubo que alterar la planificación inicial, y al final ha sido un proceso mucho más iterativo, en el que se ha ido saltando entre estos 4 bloques principales.
Igualmente los bloques de desarrollo se han dividido en más bloques, que se han ido alternando. Por ejemplo no ha sido necesario tener 100% terminado toda la fase del scraping para poder comenzar la aplicación, ya que por un lado se podía comenzar a crear la interfaz sin tener aún los datos, y por otro aunque el scraping se realiza de dos webs distintas, con tener terminado el método para extraer de una ya se pueden realizar pruebas para cargar en la aplicación



Img. 2 - Diagrama de Gantt final

3 - Herramientas utilizadas

3.1 - Herramientas de análisis y documentación

A continuación se nombran las herramientas empleadas durante el proceso de diseño y documentación:

H1. Google docs

Esta ha sido la principal herramienta a la hora de redactar y dar forma a la memoria del proyecto, se trata de un software similar a Microsoft Word, pero online y gratuito, especialmente útil para trabajar en equipo con más personas. Quizás no cuenta con tantas herramientas y opciones como Word o el equivalente de LibreOffice, pero tiene las opciones necesarias para la mayoría de trabajos y cuenta con una interfaz moderna y sencilla que también lo hacen muy cómodo para cualquier proyecto. Además te evita tener que instalar pesados paquetes de software, con la contrapartida de que necesitas internet para poder trabajar en todo momento.

https://www.google.com/docs/about/

H2. Draw.io

Draw.io es un software utilizado para diseñar todo tipo de diagramas en el navegador, sin necesidad de descargar ni instalar ningún software, aunque también tiene una versión de escritorio. Permite crear diagramas de clases, diagramas de Entidad-Relación, mapas conceptuales, esquemas sencillos de páginas web etc.

https://drawio-app.com/

H3. Photopea.com

Photopea es un editor de imágenes con una interfaz muy similar a la de Photoshop, pero que se puede utilizar de forma gratuita desde el propio navegador sin instalar ningún software. Si ya conoces Photoshop la adaptación es más sencilla que a otras herramientas como Gimp, y cuentas con la mayoría de herramientas básicas para el retoque de imágenes, es decir permite usar capas, máscaras, distintos modos de fusión, ajustes de iluminación y color, etc.

https://www.photopea.com/

3.2 - Herramientas de la fase de scraping

A continuación se nombran las herramientas empleadas durante la primera fase del proyecto, la del scraping, donde recopilamos y guardamos la información de fuentes externas:

H4. Python 3.11

Se trata de un lenguaje de programación interpretado, de alto nivel y muy popular.

> "Python es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). Los desarrolladores utilizan Python porque es eficiente y fácil de aprender, además de que se puede ejecutar en muchas plataformas diferentes. El software Python se puede descargar gratis, se integra bien a todos los tipos de sistemas y aumenta la velocidad del desarrollo."
>
> Amazon AWS. *¿Qué es Python? - Explicación del lenguaje Python - AWS*. <https://aws.amazon.com/es/what-is/python/> [Consulta: 18 de mayo de 2023]

https://www.python.org/

H5. PyCharm

Se trata un software para escribir código y desarrollar aplicaciones de Python, lo que se conoce como un IDE(Integrated Development environment), estos programas además de un editor de código ofrecen herramientas para facilitar al programador sus tareas, como compiladores, gestión de los directorios, resaltado de errores en el código o mensajes de advertencia etc.

> "Desarrollado por JetBrains, PyCharm es el IDE más popular para Python hasta la fecha. Compatible con Windows, Linux y macOS, PyCharm contiene módulos y paquetes que ayudan a los desarrolladores a programar software con Python más rápido y con menos esfuerzo. También se puede personalizar para responder a las necesidades específicas de un proyecto."
>
> DataScientest. *PyCharm: Todo sobre el IDE de Python más popular.* <https://datascientest.com/es/pycharm> [Consulta: 18 de mayo de 2023]

https://www.jetbrains.com/es-es/pycharm/

H6. Selenium

Se trata de un conjunto de herramientas, principalmente pensadas para la automatización y el testeo de programas escritos en distintos lenguajes de programación, por lo que se puede importar y utilizar desde lenguajes como Java, Python, JavaScript etc.

Además de ser una de las herramientas más utilizadas para el testing, su capacidad para automatizar procesos y crear fácilmente referencias a objetos que se encuentran en el HTML de una web, a través de ids o XPath, lo han convertido en una herramienta muy apta para realizar scraping, es decir extraer información de un sitio web.

Es especialmente útil cuando no tienes que extraer toda la información de una página estática, si no que tienes que recopilar distintos tipos de información, haciendo click en distintos elementos y pasando de una url a otra, pudiendo obtener datos de webs dinámicas.

En el siguiente párrafo extraído de FreeCodeCamp se explica muy bien, y además no solo eso si no que el artículo es muy interesante, ya que muestra con código de Python cómo crear un pequeño bot para hacer scraping:

> “Selenium es una herramienta diseñada para ayudarle a ejecutar pruebas automatizadas en aplicaciones web. Está disponible en varios lenguajes de programación.Aunque no es su propósito principal, Selenium también se usa en Python para web scraping, porque puede acceder a contenido renderizado en JavaScript (lo que las herramientas de scraping normales como BeautifulSoup no pueden hacer).Selenium también es útil cuando necesita interactuar con la página de alguna manera antes de recopilar los datos, como hacer clic en botones o completar campos.”
>
> FreeCodeCamp. *Cómo codificar un scraping Bot con Selenium y Python.* <https://www.freecodecamp.org/espanol/news/como-codificar-un-scraping-bot-con-selenium-y-python/> [Consulta: 18 de mayo de 2023]

https://www.selenium.dev/

H7. Chrome Driver

Es una herramienta gratuita y open source para pruebas automatizadas de aplicaciones web. Se trata del driver que nos permite lanzar y configurar el

navegador Chrome desde el código de Python y Selenium, y nos permite navegar con el explorador, realizar entrada de usuario, es decir interactuar con la web, ejecutar javascript etc.

https://chromedriver.chromium.org/

H8. SQLAlchemy

Se trata de uno de los ORM(Object Relational Mapper) más populares para el lenguaje de programación Python. Es el equivalente de lo que sería Hibernate para Java, es decir nos permite trabajar con las típicas tablas de una base de datos SQL(relacional) como si fueran clases u objetos en nuestro lenguaje de programación, con lo que podremos seleccionar, insertar, actualizar y eliminar datos cómodamente.

https://www.sqlalchemy.org/

H9. Alembic

Es como un controlador de versiones similar a git, pero enfocado a bases de datos, es decir nos permite ir modificando la estructura de nuestra base de datos de forma muy útil durante el desarrollo, adaptándola a las necesidades que vayan surgiendo, y dejando registrado las distintas versiones o fases por las que ha pasado por si necesitamos volver atrás. Esta herramienta está pensada para usarla en conjunto con SQLAlchemy para Python.

https://alembic.sqlalchemy.org/en/latest/index.html

H10. Psycopg2

Se trata de un complemento para Python para conectar con bases de datos de PostgreSQL

> “Psycopg is the most popular PostgreSQL database adapter for the Python programming language. Its main features are the complete implementation of the Python DB API 2.0 specification and the thread safety (several threads can share the same connection). It was designed for heavily multi-threaded

> applications that create and destroy lots of cursors and make a large number of concurrent “INSERT”s or “UPDATE”s.”
>
> PyPI. *Pscycopg2.* <https://pypi.org/project/psycopg2/> [Consulta: 18 de mayo de 2023]

https://pypi.org/project/psycopg2/

H11. PostgreSQL

Es una de las bases de datos de tipo relacional, SQL, más usadas en el mundo, veamos como la describe el siguiente artículo de Platzi:

> “PostgreSQL, o también conocido como Postgres, es un sistema de gestión de bases de datos relacionales (RDBMS) libre y de código abierto (Open Source) que hace énfasis en la extensibilidad y el cumplimiento de SQL.
>
> Es gratuito y libre, además de que hoy nos ofrece una gran cantidad de opciones avanzadas. Una característica interesante de PostgreSQL es el control de concurrencias multiversión; o MVCC por sus siglas en inglés. Este método agrega una imagen del estado de la base de datos a cada transacción. Esto nos permite hacer transacciones eventualmente consistentes, ofreciéndonos grandes ventajas en el rendimiento.”
>
> Platzi. *PostgreSQL: qué es, cómo funciona y cuáles son sus ventajas.* <https://platzi.com/blog/que-es-postgresql/> [Consulta: 18 de mayo de 2023]

https://www.postgresql.org/

H12. pgAdmin4

pgAdmin es una interfaz gráfica que nos permite manejar y editar nuestras bases de datos en PostgreSQL. Es la herramienta oficial que ofrece PostgreSQL, y aunque también se puede interactuar con las bases de datos mediante sentencias SQL en la terminal, este programa y su interfaz facilitan muchos procesos, también puede por ejemplo generar automáticamente un diagrama a partir de una base de datos

https://www.pgadmin.org/download/

3.3 - Herramientas de la fase de aplicación

A continuación se nombran las herramientas empleadas durante la segunda fase del proyecto, para la creación de la aplicación que manejará el usuario final:

H13. VS Code

Visual Studio Code, se trata de un editor de código desarrollado por Microsoft, pero gratuito y multiplataforma. Admite todo tipo de plugin que le aportan multitud de funcionalidades y soporte para todo tipo de lenguajes como JavaScript, Python, Ruby, PHP, HTML, CSS, C#, etc.

https://code.visualstudio.com/

H14. Node.js

Dado que JavaScript es un lenguaje de programación creado para darle dinamismo a los navegadores web, su uso está ligado a la web y se necesita un explorador para ejecutarlos. Gracias a la tecnología de Node.js se puede ejecutar este lenguaje también en escritorio sin necesidad de abrir un navegador. En el siguiente texto de la empresa Mozilla, los desarrolladores de Firefox, se explica muy bien en qué consiste Node.js:

> "Node (o más correctamente: Node.js) es un entorno que trabaja en tiempo de ejecución, de código abierto, multi-plataforma, que permite a los desarrolladores crear toda clase de herramientas de lado servidor y aplicaciones en JavaScript. La ejecución en tiempo real está pensada para usarse fuera del contexto de un explorador web (es decir, ejecutarse directamente en una computadora o sistema operativo de servidor). Como tal, el entorno omite las APIs de JavaScript específicas del explorador web y añade soporte para APIs de sistema operativo más tradicionales que incluyen HTTP y bibliotecas de sistemas de ficheros."
>
> Mozilla. *Introducción a Express/Node.* <https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction> [Consulta: 18 de mayo de 2023]

https://nodejs.org/es

H15. Electron

Electron es un framework para crear aplicaciones de escritorio usando JavaScript, HTML y CSS. Es un framework muy útil para crear aplicaciones multiplataforma que pueden ser ejecutadas en cualquier sistema operativo, a partir de tecnologías web conocidas y que están muy extendidas, por lo que facilita mucho el proceso de aprendizaje.

https://www.electronjs.org/es/

H16. Sequelize

Se trata de ORM, al igual que SQLAlchemy, solo que esta vez para el lenguaje de programación JavaScript y también TypeScript(una variante con algunas características propias), y aunque hay varias opciones se trata de uno de los más extendidos. Es compatible con todo tipo de bases de datos relacionales como PostgreSQL, MySQL, Oracle, SQLite etc. Facilitará mucho la tarea de interactuar con los datos gracias a la creación de clases y objetos de JavaScript.

https://sequelize.org/

H17. Tabulator

Tabulator, una herramienta gratuita para crear tablas dinámicas en JavaScript, que proporciona muchas funcionalidades para crear las tablas y configurarlas a nuestro gusto, así como características para hacerlas muy editables.

https://tabulator.info/

H18. Xel

Xel es un conjunto de herramientas de widgets HTML para crear aplicaciones web o de Electron más rápidamente y con un diseño moderno. Esto ahorra muchas líneas de código y además nos asegura un diseño uniforme en toda la aplicación.

Algunos de los widgets incluidos son:

Botones, pestañas, controles deslizantes, casillas de verificación, interruptores, menús, barras de menú, menús contextuales, entradas de texto, entradas numéricas, diálogos, barras de progreso, etc.

https://xel-toolkit.org/

4 - Descripción del proyecto

4.1 - Análisis

ANÁLISIS DE REQUISITOS

Se requiere una aplicación desde la cual reunir las distintas ofertas laborales de distintas webs y a la vez poder filtrarlas, administrarlas, visualizarlas etc. Una especie de sistema gestor de ofertas, con funcionalidades que faciliten manejarlas y hacer seguimiento de las que nos interesen. Se plantea como una aplicación de escritorio compatible con los distintos sistemas operativos.

Requisitos funcionales:

RF1- Guardar las ofertas con su información y sus links.

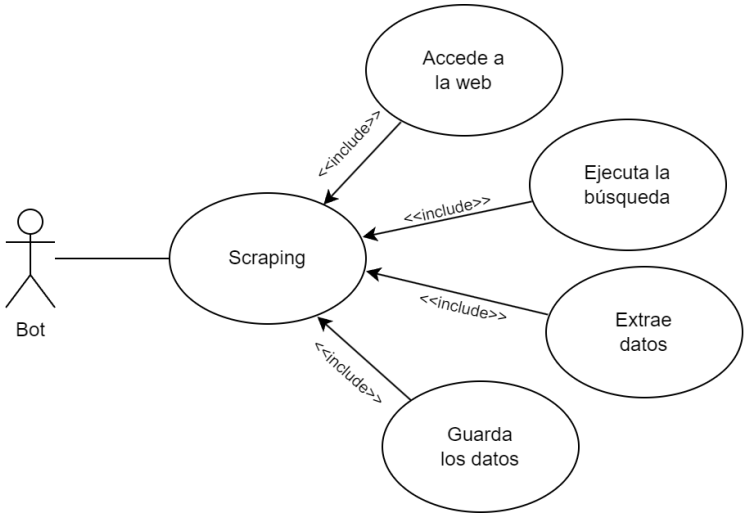
RF2- Filtrar y ordenar por distintos campos como sueldo, experiencia requerida etc.

RF3- Posibilidad de que el usuario marque el nivel de conocimientos de las tecnologías que se suelen pedir, y así filtrar por las que más domine.

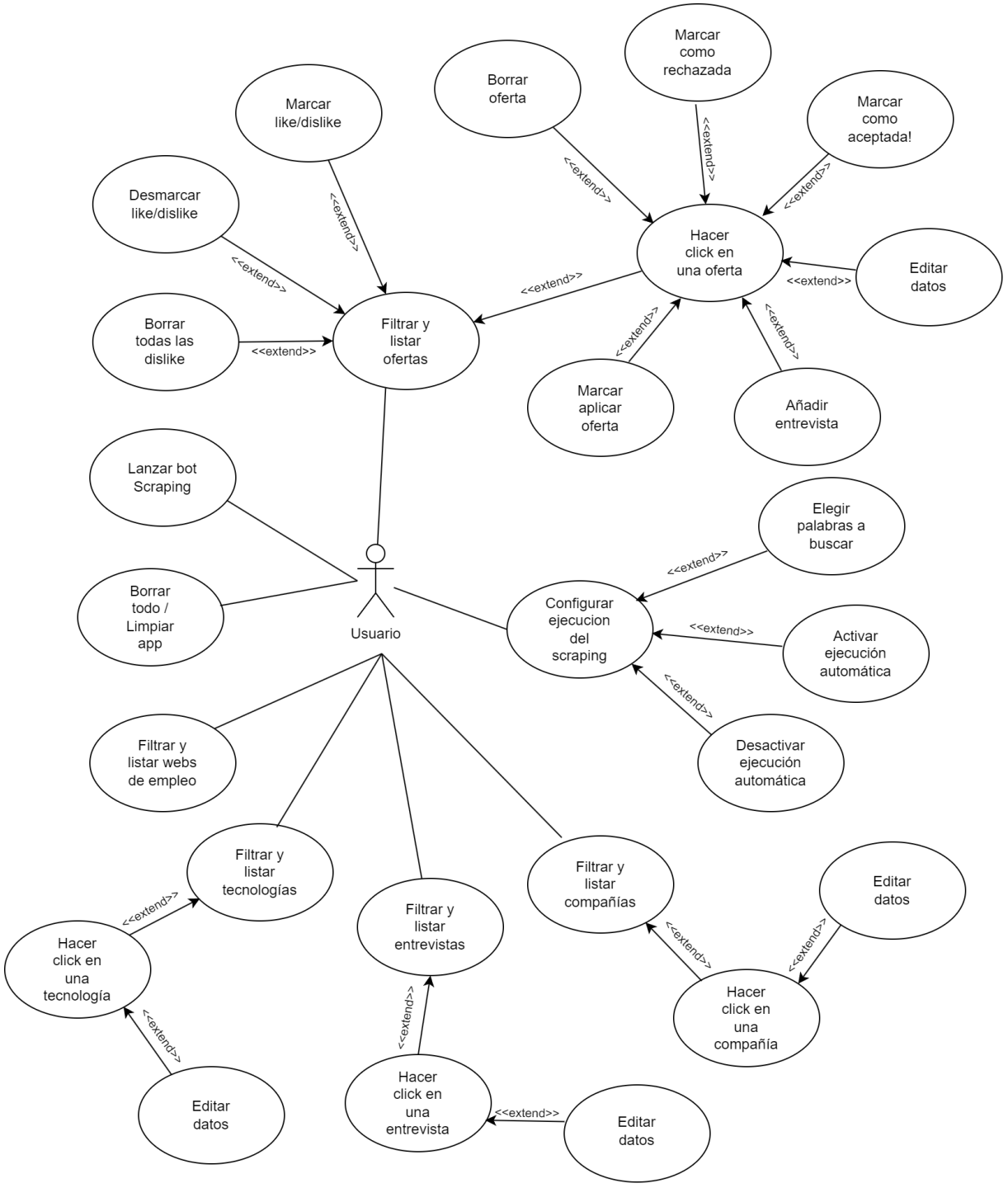
RF4- Obtención de información de las distintas empresas, por ejemplo con la web de glassdoor, para poder visualizar su reputación y comentarios/opiniones de empleados y exempleados.

RF5- Posibilidad de añadir notas personales que nos sean de utilidad a cada oferta.

RF6- Posibilidad de marcar distintos estados para cada oferta, por ejemplo si nos gusta, si estamos en fase de entrevistas, si hemos sido rechazados etc.



Img. 3 - Diagrama de Casos de Uso, bot-scraping



Img. 4 - Diagrama de Casos de Uso, usuario-App

Este diagrama representa cómo se relaciona el usuario con la aplicación. Nada más iniciarla se puede ver que puede realizar distintas acciones, entre ellas:

- Borrar todos los datos almacenados, limpiando la aplicación.
- Configurar algunos parámetros del scraping.
- Lanzar o ejecutar el robot de scraping.

- Filtrar y listar alguno de los distintos tipos de información, es decir ver datos de las ofertas, las compañías, las webs de donde se han extraído, las tecnologías que se piden, o las entrevistas(estos datos los irá añadiendo el usuario).

Las ofertas de trabajo son los datos con los que más opciones de interactuar tiene el usuario, pudiendo por un lado seleccionar una, lo que nos muestra su información y opciones en el panel lateral, para editar sus datos, añadir una entrevista, o marcar un estado, o borrarla. Con las ofertas también se puede interactuar directamente desde la visualización en la tabla de ofertas, se pueden marcar las que le gusten o no al usuario, sin necesidad de seleccionarla primero, o borrar todas las que no le gusten.

Requisitos no funcionales:

RFN1- El scraping de las distintas webs de empleo debe realizarse sin el uso de herramientas de scraping de pago, se creará el código específico para conseguir solo la información que deseamos y realizar el correcto tratamiento e inserción en una base de datos.

RFN2- Evitar el bloqueo de la IP o el baneo de la cuenta con la que se realice el scraping.

RFN3- Compatibilidad con Windows, Mac y Linux.

RFN4- Preferible uso de tecnologías web, como html, css, javascript, adaptadas a escritorio, no solo por compatibilidad en distintas plataformas, si no para facilitar una posible adaptación a futura página web con la herramienta totalmente online.

RFN5- Terminarlo en un par de meses.

4.2 - Diseño

En primer lugar vamos a hablar sobre cómo vamos a recopilar los datos de las ofertas laborales, es decir de web scraping.

El web scraping consiste en la extracción de forma automatizada de los distintos datos que nos muestre una página web, posteriormente podemos tratarlos para guardarlos como más nos convenga.

¿Para qué se hace web scraping?

Según este artículo de Rafael Zambrano para Open Webinars:

> “Estas técnicas nos permiten hacer muchas cosas con los datos de la web, que cada vez son más valiosos. Entre su gran número de aplicaciones prácticas, vamos a destacar algunas:
>
> -Alimentar una base de datos.
>
> -Hacer una migración de un sitio web.
>
> -Recopilar y ofrecer datos de varias webs.
>
> -Generar alertas.
>
> -Monitorear precios de la competencia.
>
> -Localizar ítems o stock en ecommerces.
>
> -Recolectar fichas de productos.
>
> -Detectar cambios en una web.
>
> -Analizar enlaces de una web para buscar links rotos.
>
> El web scraping, en resumen, nos permite sacar datos de la web, que es la mayor fuente de datos que existe y existirá.”
>
> OpenWebinars. *Qué es y por qué usar el Web Scraping.* <https://openwebinars.net/blog/que-es-por-que-usar-web-scraping/> [Consulta: 19 de marzo de 2023]

Problemas al extraer datos web

Aunque hay cierta polémica sobre el tema, a día de hoy el web scraping es completamente legal, y así lo ha ratificado la justicia en diversas ocasiones, al final, no se está extrayendo ninguna información confidencial de la web scrapeada, simplemente se automatiza la obtención de una información que cualquier usuario puede ver con métodos manuales.

Aun así muchas webs implementan sistemas anti scraping, ya que aunque no se obtenga información confidencial, se obtiene una gran cantidad de datos, en un volumen que a las empresas implicadas no les interesa compartir, ya que por ejemplo se podría usar por empresas de la competencia. En el caso de LinkedIn por ejemplo, ofrece planes de suscripción Premium con características para obtener información de muchos usuarios y empresas, que no sería necesario con el uso de

herramientas automatizadas, y por tanto no les conviene. Es por eso que estas webs tratarán de bloquear el acceso a las mismas si detectan que se hace a través de herramientas automatizadas, pudiendo bloquear la IP o la cuenta del usuario.

Otro de los problemas es que hay que analizar cada web y crear el código específico para la estructura de la misma, por lo que si en cualquier momento la web cambia, incluso solo con el cambio de nombre en el id o clase de un elemento, podría hacer que script dejase de funcionar, o no extrayera correctamente toda la información. Por lo que no solo hay que crear programas específicos para cada web y tipo de datos que deseemos, sino que también hay que estar comprobando habitualmente si el script sigue funcionando correctamente.

¿Cómo se ha realizado el scraping?

Para realizar el scraping utilizaremos las herramientas mencionadas en el punto 7, es decir Python, PyCharm y Selenium.

Respecto al problema del baneo o bloqueo de la IP, tras realizar unas cuantas pruebas en varias páginas hemos comprobado, que implementando algunos tiempos de espera, entre las acciones que va realizando Selenium para extraer los datos, no suelen detectar el uso de un software automatizado.

Para ello no solo implementaremos tiempos de espera de duración aleatorios (dentro de un rango), sino que también implementaremos algunos descansos más largos que saltarán de forma aleatoria cada ‘x’ tiempo u operaciones.

Con estos métodos no he tenido problema ni ningún aviso, ni siquiera un captcha a resolver para comprobar si soy humano, por parte de las dos webs a las que extraemos los datos.

Otro problema que hemos encontrado y tenido en cuenta es la posibilidad de obtener ofertas repetidas al extraer datos de varias webs. Tras investigar el tema hemos planteado una posible solución, pero no la hemos implementado en esta primera versión dada la urgencia del proyecto por parte del cliente y que ahora mismo tampoco extraemos de muchas fuentes. La resolución del problema puede encontrarse en el apartado de 5-Trabajos futuros.

Especificaciones de Software:

Elección de software

S1 - Python

- ¿Por qué Python?

https://www.tiobe.com/tiobe-index/

Como se puede ver en el enlace, según el índice TIOBE, Python es ahora mismo el lenguaje más popular en todo el mundo, ocupando el primer puesto desde hace tiempo. Además es un lenguaje especialmente utilizado en el análisis y manejo de datos. Estos motivos lo convierten en una elección idónea para la primera fase del proyecto.

S2 - PostgreSQL

- ¿Por qué PostgreSQL?

https://db-engines.com/en/ranking

Respecto a PostgreSQL, como se puede ver en la web db-engines, se trata de un tipo de base de datos también muy popular, sólo superada por Oracle, MySQL y Microsoft SQL Server. Por un lado Oracle y Microsoft SQL Server están enfocadas más bien a bases de datos empresariales muy grandes, y además de todas estas PostgreSQL es la única Open Source, ya que MySQL es de Oracle. Además aunque esta posicionada en cuarto lugar su popularidad y uso está creciendo de forma continuada estos últimos años, por lo que de nuevo parece una buena elección tanto para uso personal como profesional.

S3 - Electron.js

- ¿Por qué Electron.js?

Dada la urgencia del proyecto, se ha optado por crear una primera versión como un programa de escritorio monousuario. Pero en un futuro con un mayor tiempo de desarrollo, dadas las características del software, como la necesaria conexión a internet, probablemente sería más útil crear una versión 100% web, con distintos usuarios, y guardando los datos en servidores en vez de en local.

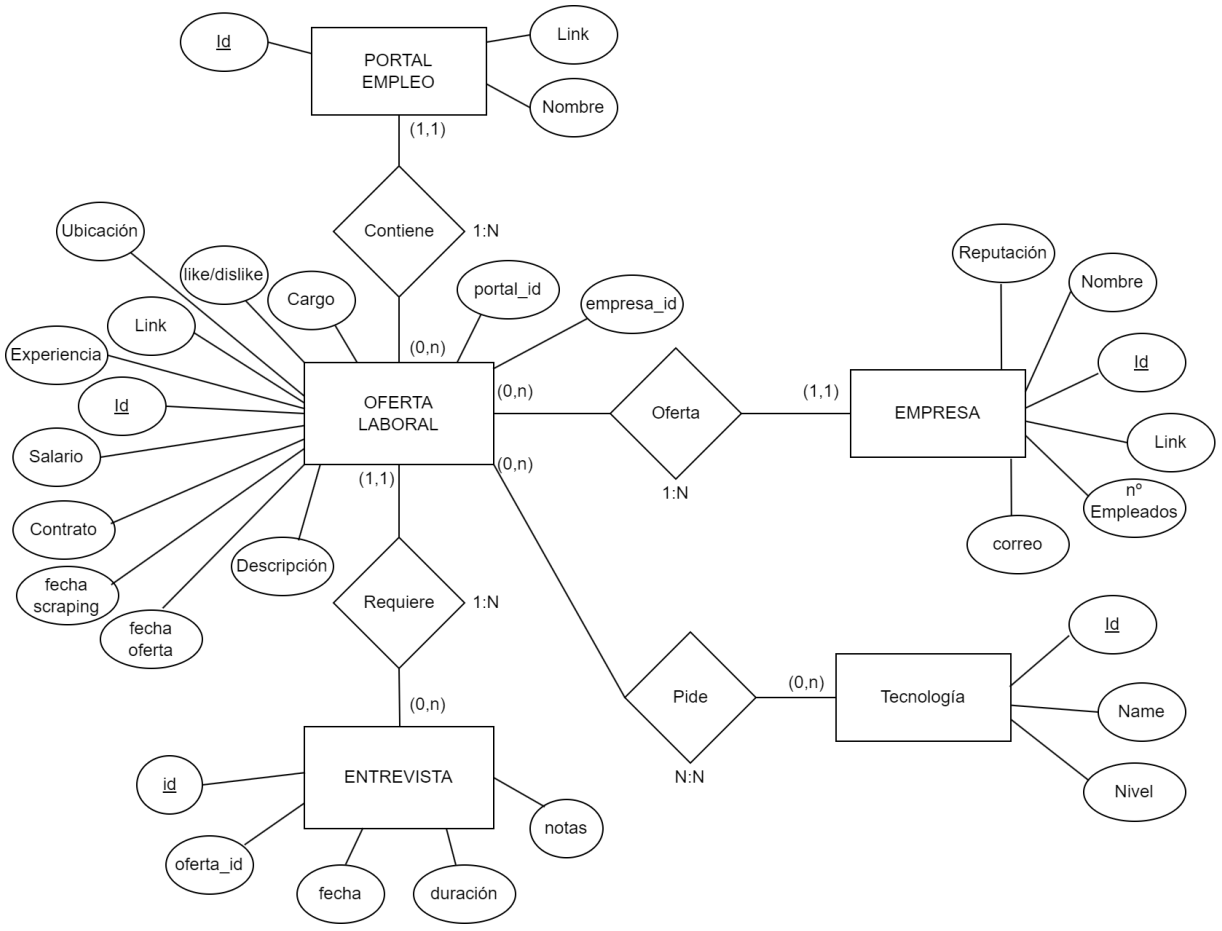
Por este motivo y por la compatibilidad con los distintos sistemas operativos se ha elegido la herramienta de desarrollo Electron, ya que al trabajar con tecnologías del navegador como HTML, CSS y JavaScript, puede hacer más sencillo en el futuro la conversión a la versión web.

Especificaciones de Hardware:

H1 - Cualquier ordenador medianamente actual debería tener suficiente potencia para ejecutar satisfactoriamente la aplicación.

H2 - Se recomienda una buena conexión a internet, tanto en velocidad como estabilidad, para no tener problemas al visitar las distintas páginas externas de las que extraemos información.

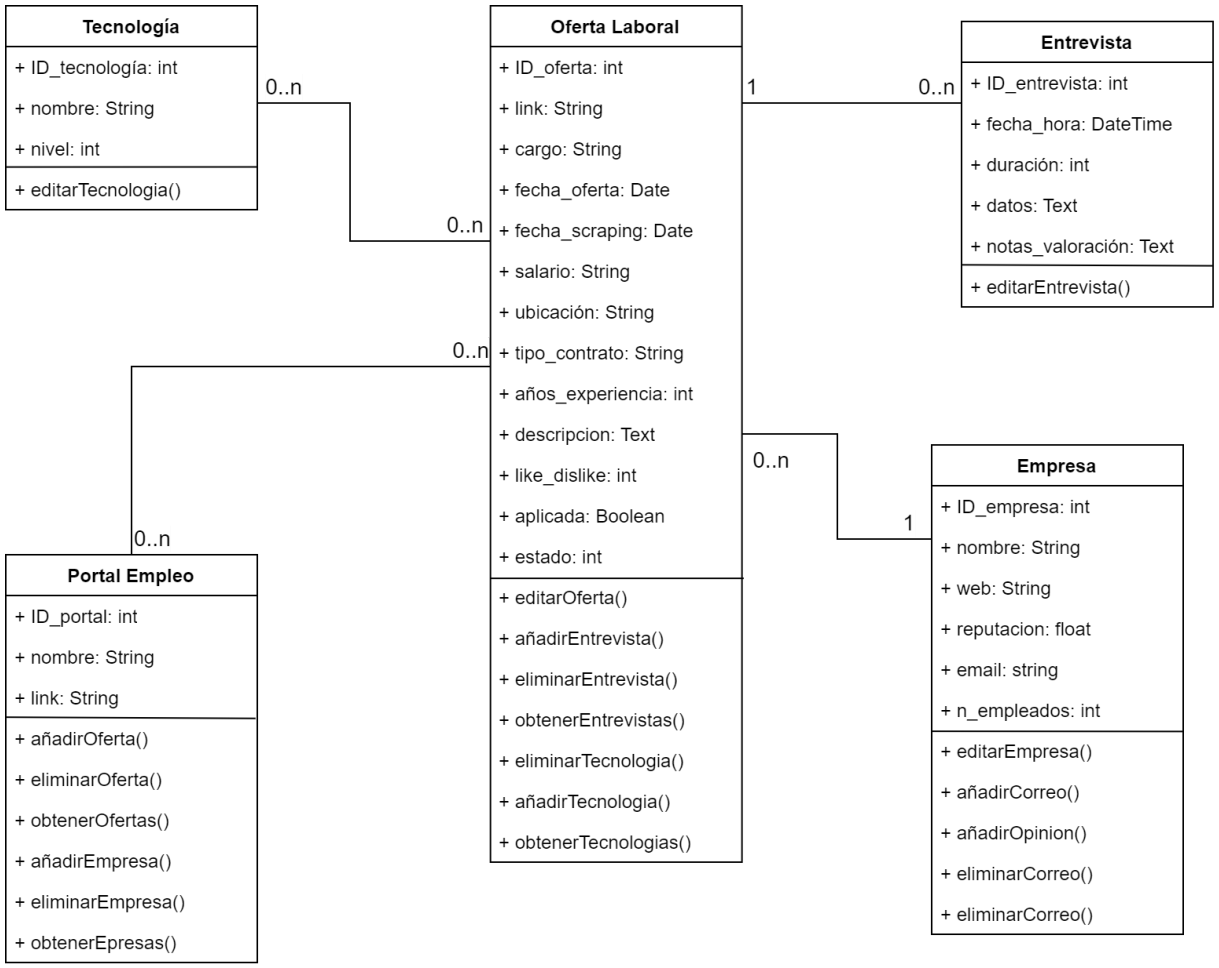
A continuación se muestra el diagrama de Entidad-Relación diseñado para almacenar los datos de todos los elementos extraídos durante el scraping y necesarios para la aplicación:



Img. 5 - Diagrama Entidad-Relación

Ya que se ha estado trabajando con ORM y clases para guardar los datos y trabajar con ellos, ahora se mostrará el diagrama de clases utilizado para el diseño de las mismas.

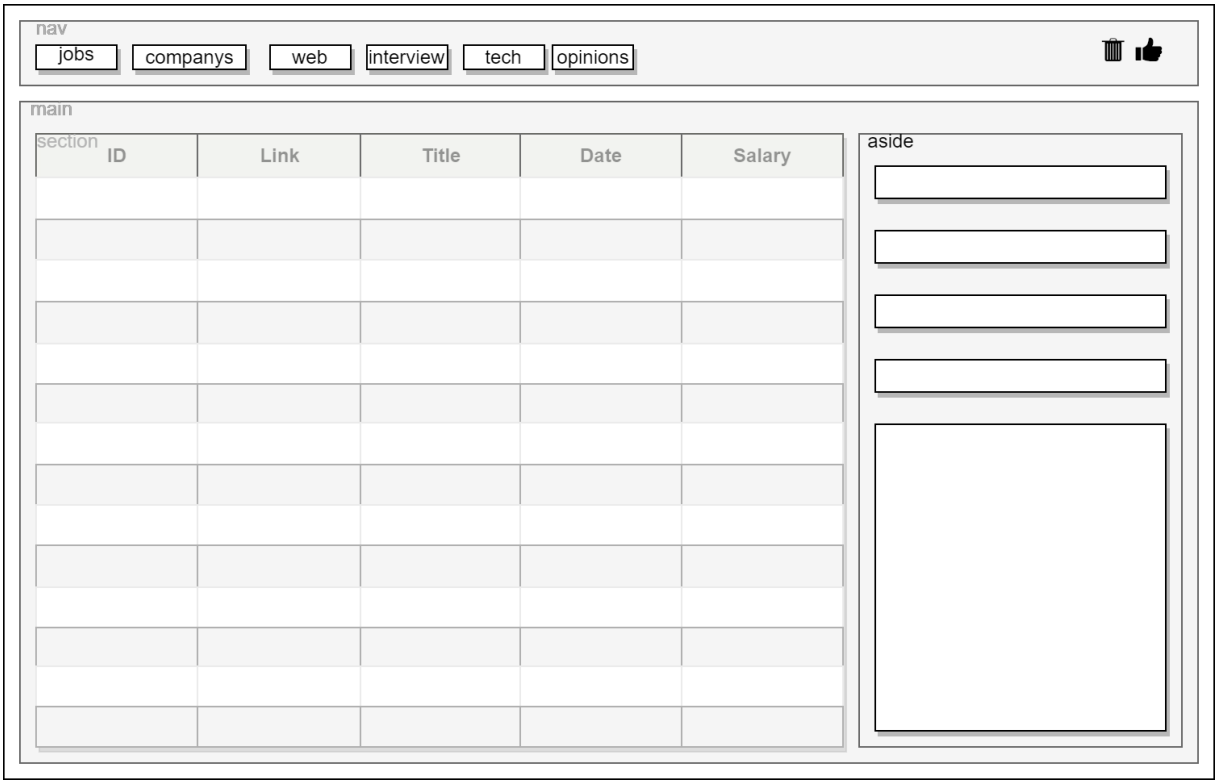
Como el usuario final sólo interactúa con la interfaz gráfica, y dado el escaso margen de tiempo, se han dejado los atributos de forma pública para acceder directamente y no necesitar métodos extra para los getters y setters:



Img. 6 - Diagrama de Clases UML

Respecto al diseño de la aplicación, dado que lo fundamental será la visualización de los datos recogidos, el cuerpo principal de la misma será una tabla donde verde forma ordenada esa información, con una fila por oferta, y distintas columnas para los distintos elementos y datos de la oferta. Además dado que además de ofertas laborales tenemos información guardada en otras tablas como empresas o entrevistas lo ideal será tener distintas pestañas en la parte superior, que muestren tablas diferentes. El otro elemento principal será un panel lateral, que se rellenará con los datos de la fila seleccionada, y nos permitirá visualizar mejor la información de la oferta específica, así como modificarlos etc.

A continuación se muestra una primera versión del diseño de la aplicación, principalmente en cuanto a distribución de los elementos:



Img. 7 - Diseño inicial de la Interfaz

4.3 - Implementación

Como se ha comentado anteriormente, el desarrollo del proyecto se divide en dos bloques principales, el primero el del scraping que será 'invisible' para el usuario final, donde se obtiene y recopila información de distintas fuentes y se guarda en la base de datos, y el segundo en el que se desarrolla la aplicación como una interfaz gráfica desde la que el usuario final pueda hacer uso visualizando, manejando y filtrando esa información recopilada.

Así pues, se va a comenzar con la implementación del primero de los bloques.

4.3.1 - SCRAPING Y BASE DE DATOS

Lo primero será configurar selenium para poder automatizar los procesos de extracción de la información:

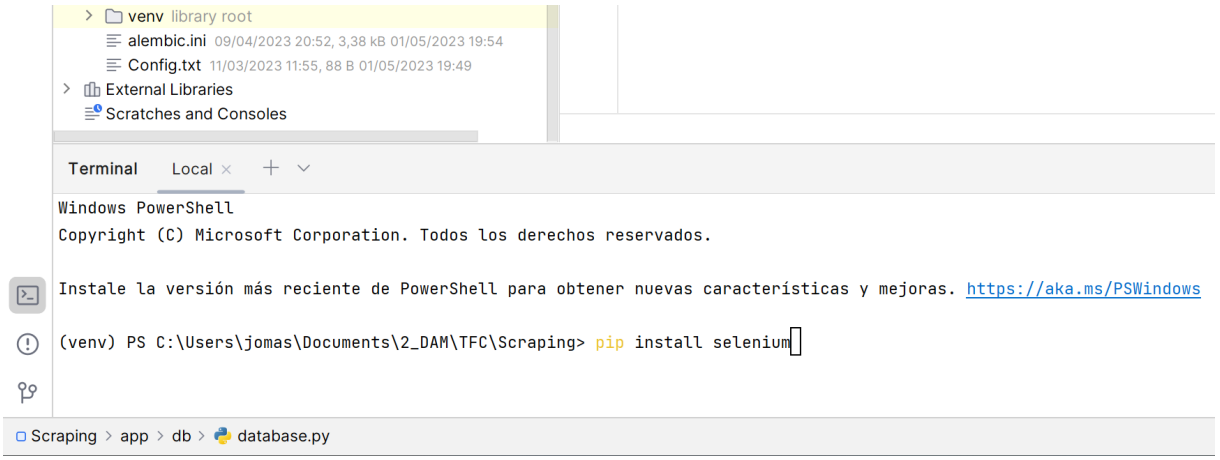
Primero se crea un proyecto de python en PyCharm, generando un 'virtual environment' donde se instalan todas las librerías necesarias para el proyecto.

Un entorno virtual es un entorno Python en el que el intérprete Python, las bibliotecas y los scripts instalados en él, están aislados de los instalados en otros entornos virtuales, y (por defecto) cualquier biblioteca instalada en un «sistema» Python, es decir, uno que esté instalado como parte de tu sistema operativo.

https://docs.python.org/es/3.8/library/venv.html#:~:text=Un%20entorno%20virtual%20es%20un,parte%20de%20tu%20sistema%20operativo.

Lo primero que se necesitará será instalar y configurar Selenium para poder automatizar los procesos de extracción de la información, se instalarán las librerías en la terminal de PyCharm del proyecto, en el virtual enviroment:

```
> pip install selenium
```



Img. 8 - Instalación de paquetes en la terminal de PyCharm

Luego en el archivo de python se importará la librería y las opciones para lanzar el navegador:

```
from selenium import webdriver

from selenium.webdriver.chrome.options import Options

options = Options()
```

Para lanzar el navegador en pantalla completa:

```
options.add_argument("start-maximized")
```

Para no ver el navegador:

```
options.add_argument("--headless")
```

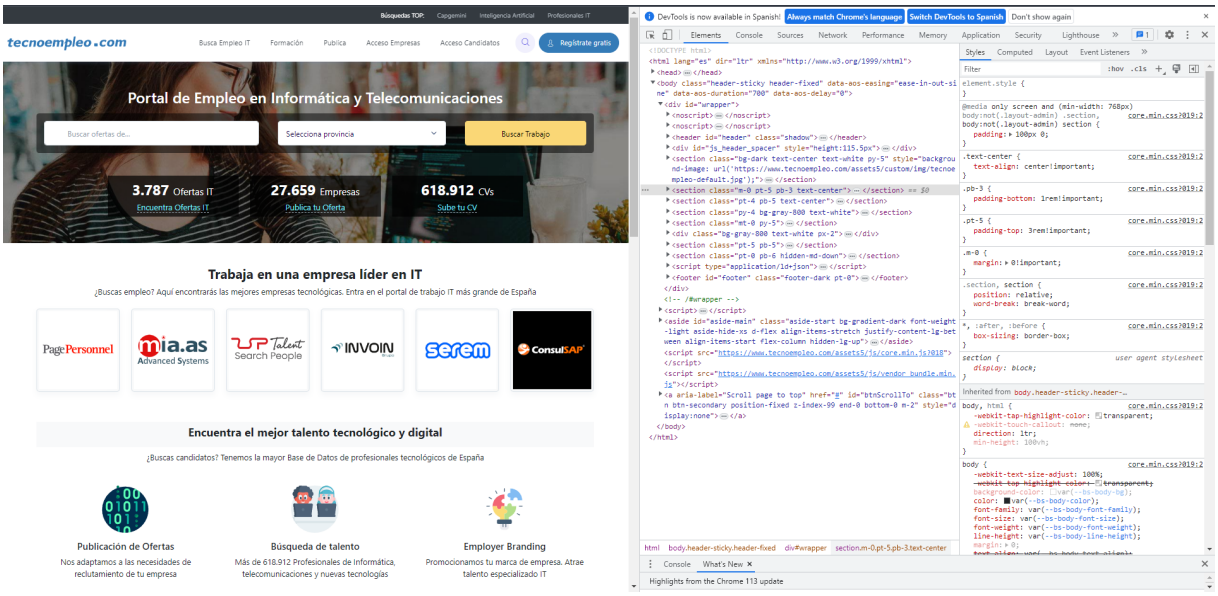
Se añaden las opciones al driver de Chrome:

```
driver = webdriver.Chrome(options=options)
```

Se le pasará al driver la url de la web a la que se desee navegar:

```
driver.get("https://es.indeed.com/")
```

Se ejecuta el script y ya se lanzaría el navegador que carga la página indicada. Una vez en el sitio web, se debe explorar el DOM, inspeccionando la página con las herramientas del navegador

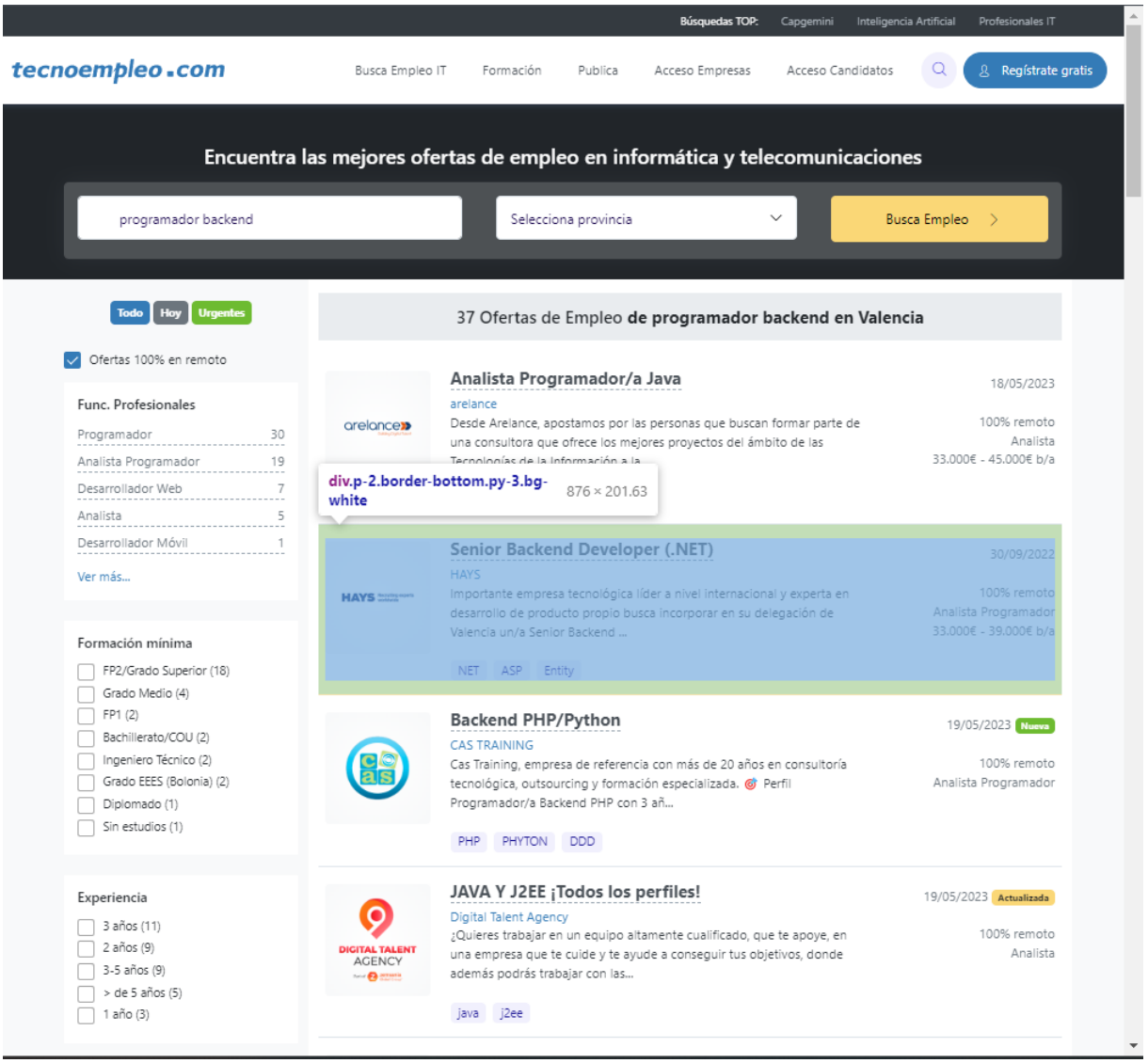


Img. 9 - Inspeccionando web

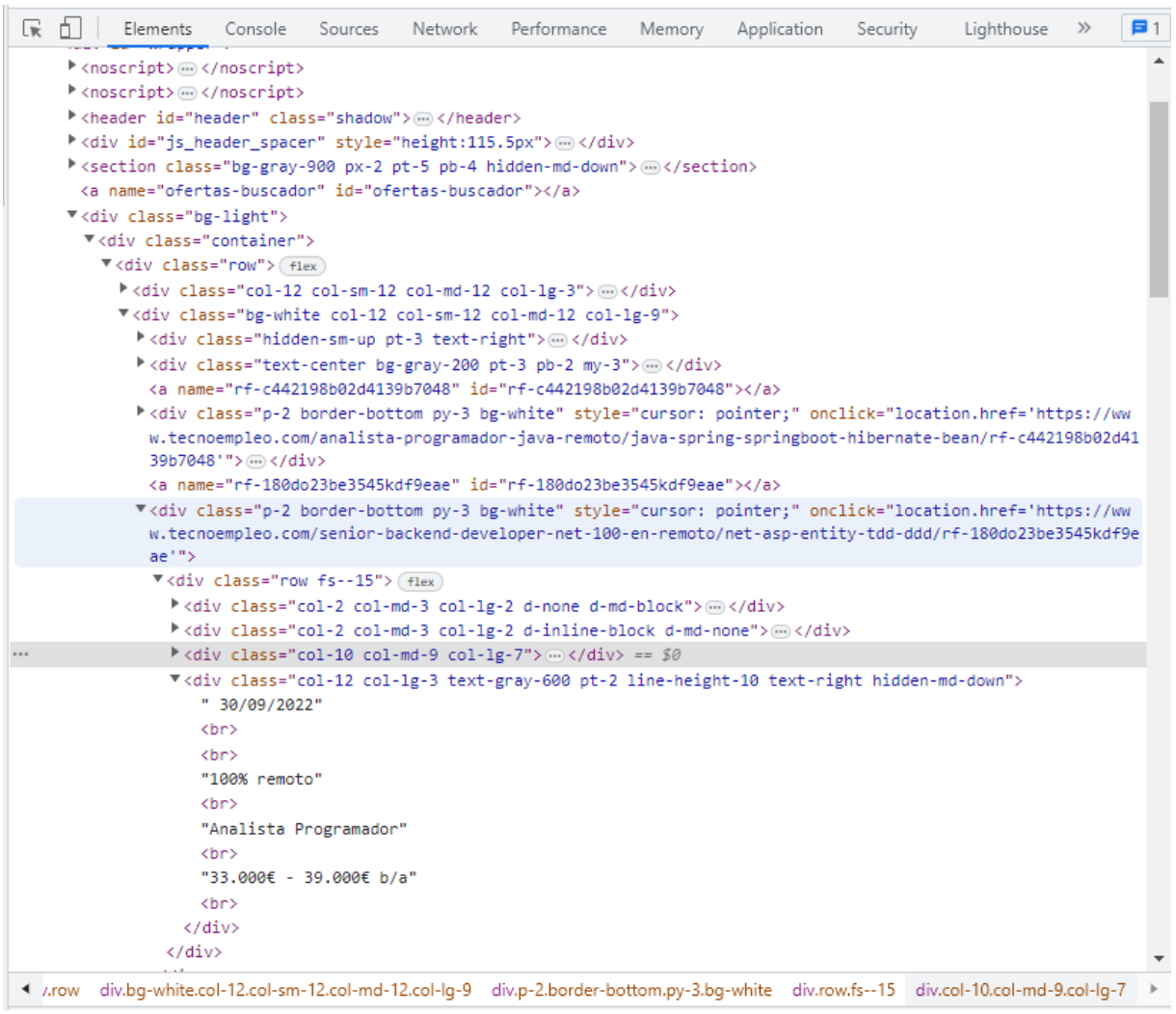
```
# REALIZAMOS LA BÚSQUEDA
input_search = driver.find_element(By.XPATH, "/html/body/div[@id='wrapper']
               /section[1]/div[@class = 'container']/form//input")

input_search.send_keys('programador backend')
time.sleep(2)
input_search.send_keys(Keys.ENTER)
```

Como se puede ver, tras analizar el DOM de la página se ha encontrado el XPath que permitirá identificar el input o caja de búsqueda, a la que se le pasarán las palabras a buscar, según el tipo de empleo que nos interesa y luego la tecla enter para ejecutar la búsqueda.

Así se llegaría a la vista de resultados de la búsqueda, con un listado de ofertas similar al siguiente:



Img. 10 - Lista de resultados, visualización del contenedor de una oferta



Img. 11 - Lista de resultados, estructura HTML

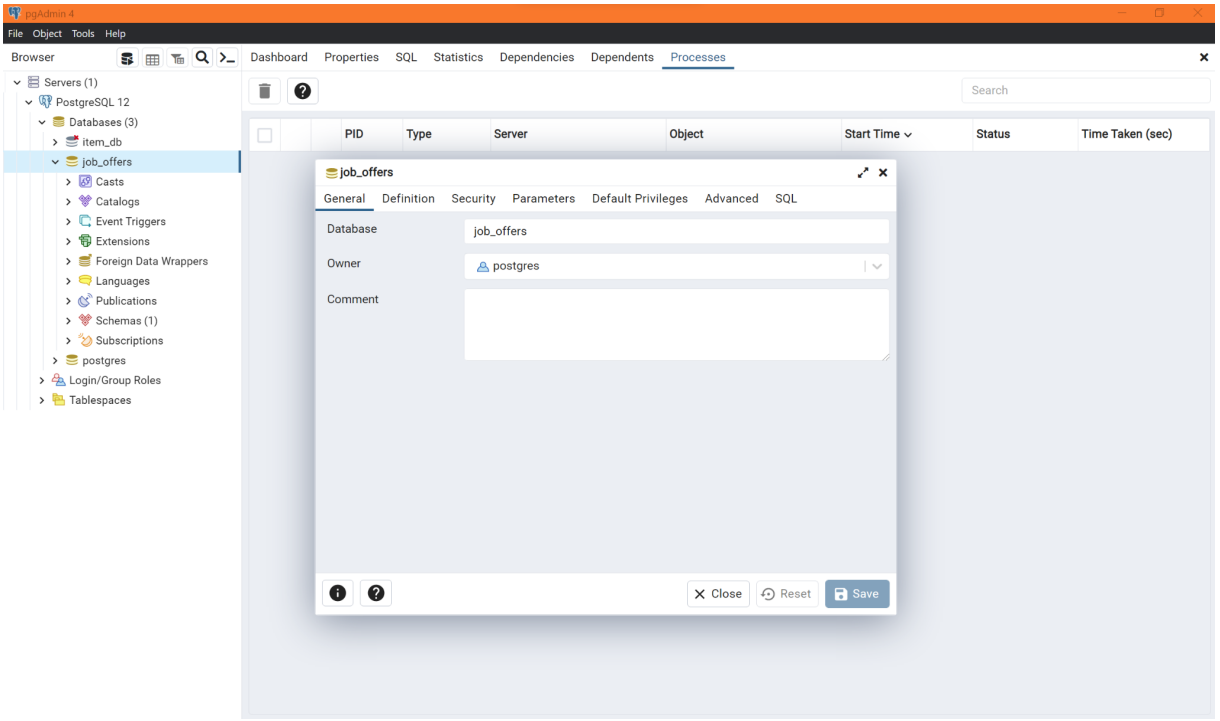
Analizando el DOM se extrae la caja o contenedor de las ofertas y luego cada una de ellas. Luego de cada oferta se obtienen los campos que interesen pudiendo guardarlos en variables, para pasarlos con cada oferta a la base de datos, o añadirlos a distintas listas para pasarlos todos de golpe al terminar de recorrer las ofertas:

```
container = driver.find_element(By.XPATH,
            "/html/body/div[@id = 'wrapper']/div[@class ='bg-light']
             /div[@class = 'container']/div[@class = 'row']/div[2]")

listado = container.find_elements(By.XPATH, "./a")

print('número de ofertas: ', len(listado))

contador = 1
```

```python
for li in listado:
    oferta = container.find_element(By.XPATH,
                                    "./div[contains(@class, 'p-2 border-bottom py-3')][{}]".format(contador))

    try:
        print('\n---- OFERTA: ', contador, '----')
        contador += 1
        print(li.get_attribute('id'))
        driver.execute_script("arguments[0].scrollIntoView();", li)
        time.sleep(2)

        link = oferta.find_element(By.XPATH,
                                   "./div[contains(@class, 'row')]/div[3]/h3/a").get_attribute('href')
        print('--> LINK OFERTA: ', link)
        link = link.split('www.tecnoempleo.com/')[1]
        link_list.append(link)

        cargo = oferta.find_element(By.XPATH,
                                    "./div[contains(@class, 'row')]/div/h3").text
        print('--> CARGO: ', cargo)
        cargo_list.append(cargo)

        empresa = oferta.find_element(By.XPATH,
                                      "./div[contains(@class, 'row')]/div[3]/a").text
        print('--> EMPRESA: ', empresa)
        empresa_list.append(empresa)

        empresa_link = oferta.find_element(By.XPATH,
                                           "./div[contains(@class, 'row')]/div[3]/a").get_attribute('href')
        print('--> EMPRESA LINK: ', empresa_link)
        empresa_link_list.append(empresa_link)

        info_lateral = oferta.find_element(By.XPATH, "./div[contains(@class, 'row')]/div[4]").text

        fecha = info_lateral.split('\n')[0]
        fecha = fecha.split(' ')[0]
        print('--> FECHA: ', fecha)
        fecha_list.append(fecha)

        año = int(fecha.split('/')[2])
        mes = int(fecha.split('/')[1])
        dia = int(fecha.split('/')[0])
        print('offer_date(',año,'-',mes,'-',dia,')')

        company = Company(name=empresa)
        print(company)
        company_id = insert_company(company)

        job = JobOffer(link=link, web=2, company=company_id, position=cargo, offer_date=datetime(año, mes, dia))
        print(job)
        insert_job(job)

    except Exception as ex:
        print(ex)
```

Img. 12 - Código Python para extraer datos de una oferta

Pero, ¿cómo guardar esta información en una base de datos? Ahora es cuando entra en juego el ORM SQLAlchemy, Alembic, PostgreSQL, etc. Primero se instala PostgreSQL y pgAdmin4, la interfaz gráfica donde crearemos la base de datos:



Img. 13 - Interfaz de pgAdmin4

Una vez creada se volverá a PyCharm para instalar SQLAlchemy, psycopg2 y conectar con la base de datos, en la terminal de nuestro entorno virtual se escribe:

```
> pip install SQLAlchemy
```

Y luego:

```
> pip install psycopg2
```

Ahora en un archivo de python que se llamará database.py se debe importar:

```
from sqlalchemy.orm import declarative_base, sessionmaker

from sqlalchemy import create_engine
```

Aquí importamos dependencias de SQLAlchemy y a continuación se realiza la conexión:

```
engine =
create_engine("postgresql+psycopg2://postgres:1415@localhost/job_offers",
echo=False)

Base = declarative_base()
```

```
SessionLocal=sessionmaker(bind=engine)
```

La opción echo sirve para visualizar las sentencias SQL que ejecutará SQLAlchemy, durante el desarrollo puede ser interesante activarlo.

Ahora se debe crear las clases o modelos que serán equivalentes a las tablas de PostgreSQL, y de hecho se crearán automáticamente en la base de datos gracias al ORM y a Alembic, pero veamos cómo se crean y configuran:

```
from datetime import datetime
from sqlalchemy import Column, Integer, String, DateTime, Text, ForeignKey
from app.db.database import Base

class JobOffer(Base):
    __tablename__ = 'job_offer'

    id = Column(Integer, primary_key=True, autoincrement=True)
    web = Column(Integer,
                 ForeignKey(
                     "web.id",
                     name="job_offer_web_id_fkey",
                     ondelete="CASCADE",
                     onupdate="CASCADE"
                 ), nullable=False)
    link = Column(String, unique=True, nullable=False)
    company = Column(Integer,
                     ForeignKey(
                         "company.id",
                         name="job_offer_company_id_fkey",
                         ondelete="CASCADE",
                         onupdate="CASCADE"
                     ), nullable=False)
    position = Column(String, nullable=False)
    offer_date = Column(DateTime, nullable=False)
    scraping_date = Column(DateTime, default=datetime.utcnow)
    salary = Column(String)
    location = Column(String)
    experience = Column(Integer)
    contract = Column(String)
    description = Column(Text)
    like_dislike = Column(Integer)
    state = Column(Integer)
```

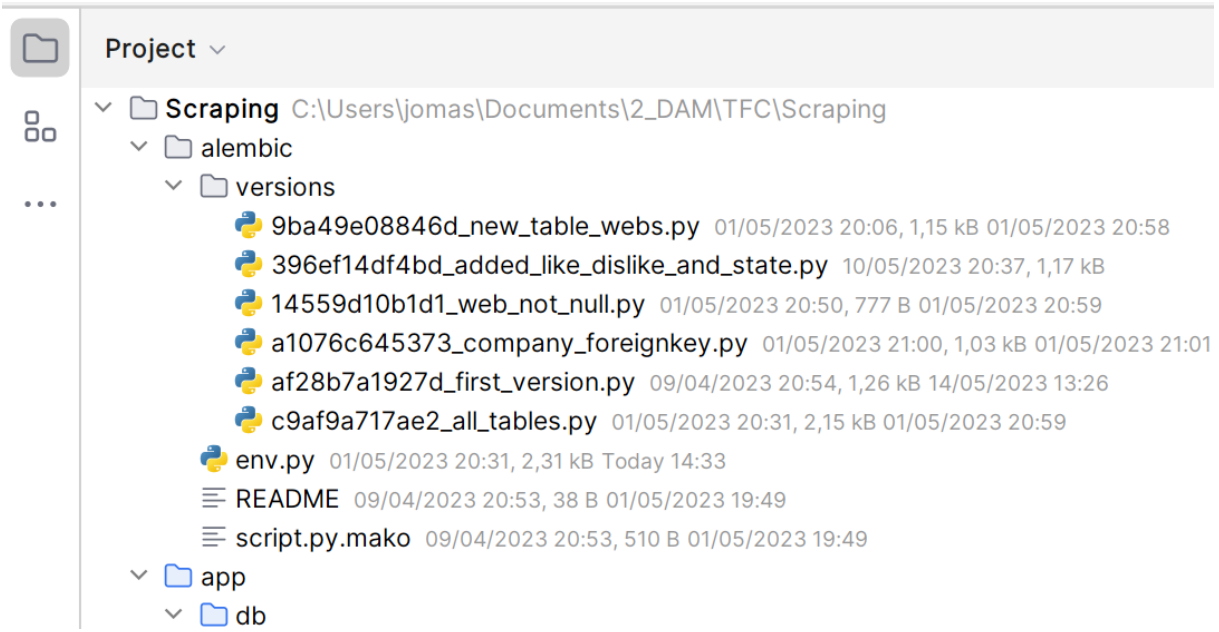
Una vez creado el modelo, para poder empezar a realizar pruebas metiendo datos se debe ejecutar Alembic para que sincronice los modelos del ORM con la base de datos, de nuevo lo primero será instalarlo en el entorno de python:

```
> pip install alembic
```

Luego se ejecutan la siguiente sentencia:

```
> alembic init .
```

Creados los directorios de Alembic:



Img. 14 - Archivos de las distintas versiones de la Base de Datos con Alembic

Ya están creados los directorio alembic, versions y el archivo env.py, que se debe configurar, para indicarle donde se encuentran los modelos que deberá tener en cuenta para generar las tablas automáticamente. A continuación se muestra el código de cómo se le pasan los modelos y en que líneas del archivo se encuentra esta parte del código a configurar:

```
19 | from app.db.database import Base
20 |
21 | from app.db.models.job_offer import JobOffer
22 | from app.db.models.web import Web
23 | from app.db.models.company import Company
24 | from app.db.models.technology import Tech
25 | from app.db.models.interview import Interview
26 |
27 | target_metadata = Base.metadata
```

Básicamente se le debe pasar a `target_metadata` el valor `Base` con la metadata, e importar cada modelo, que recordemos se creaba importando a su vez `Base`:

```
from app.db.database import Base

class JobOffer(Base):
    __tablename__ = 'job_offer'
```

A continuación ya se puede ejecutar en la terminal el comando de Alembic que nos generará las tablas en la base de datos a partir de los modelos:

```
> alembic revision --autogenerate -m "create tables"
```

Ejemplo de una primera revisión de Alembic, solo con el modelo de JobOffer:

```
"""first version
Revision ID: af28b7a1927d
Revises:
Create Date: 2023-04-09 20:54:49.342627
"""
from alembic import op
import sqlalchemy as sa

# revision identifiers, used by Alembic.
revision = 'af28b7a1927d'
down_revision = None
branch_labels = None
depends_on = None
```

```
def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table('job_offer',
    sa.Column('id', sa.Integer(), autoincrement=True, nullable=False),
    sa.Column('link', sa.String(), nullable=False),
    sa.Column('company', sa.String(), nullable=False),
    sa.Column('position', sa.String(), nullable=False),
    sa.Column('offer_date', sa.DateTime(), nullable=False),
    sa.Column('created_date', sa.DateTime(), nullable=True),
    sa.Column('salary', sa.String(), nullable=True),
    sa.Column('location', sa.String(), nullable=True),
    sa.Column('experience', sa.Integer(), nullable=True),
    sa.Column('contract', sa.String(), nullable=True),
    sa.Column('description', sa.Text(), nullable=True),
    sa.PrimaryKeyConstraint('id')
    )
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_table('job_offer')
    # ### end Alembic commands ###
```

Ahora se pueden insertar ofertas en la base de datos, para ello además se creará un métodos que en caso de existir ya una oferta con ese mismo link, que nos sirve de identificador, lo que hará será actualizar los datos de la misma, siempre que los halla recogido correctamente, es decir si algún dato esta como null (o None en Python), este no se actualiza.

A continuación el código creando un objeto JobOffer con datos de prueba y el código del método para insertar y actualizar ofertas:

```python
from datetime import datetime
from sqlalchemy import select
from app.db.database import SessionLocal
from app.db.models.job_offer import JobOffer
from app.db.models.web import Web
from app.db.models.company import Company
from app.db.models.technology import Tech

job = JobOffer(link='invent_link2352352', company='Microsoft', position='Backend .NET developer',
               offer_date=datetime(2023, 3, 2), salary='2.000€', location='Valencia', experience=2,
               description='Server-side application development using C#')

session = SessionLocal()

def insert_job(job):
    link = job.link
    print('------ INSERT JOB ------')
    try:
        session.add(job)
        session.commit()
        print('------ NEW JOB ------')
        job = session.query(JobOffer).filter_by(link=link).first()
        print('job.id: ', job.id)
        return job.id
    except:
        print('------ Job UPDATE------')
        session.rollback()
        job_db = session.query(JobOffer).filter_by(link=link).first()
        if job.position != None:
            job_db.position = job.position
        if job.offer_date != None:
            job_db.offer_date = job.offer_date
        if job.scraping_date != None:
            job_db.scraping_date = job.scraping_date
        if job.salary != None:
            job_db.salary = job.salary
        if job.location != None:
            job_db.location = job.location
        if job.experience != None:
            job_db.experience = job.experience
        if job.contract != None:
            job_db.contract = job.contract
        if job.description != None:
            job_db.description = job.description

        session.commit()
        print('job.id: ', job_db.id)
        return job_db.id
#insert_job(job)
```

Img. 15 - Código Python para insertar y actualizar ofertas

De esta forma ya se tendría una oferta en la base de datos, y además el método permite detectar si ya existe esa oferta y actualizar los datos. Ahora ya solo se trata de utilizar la información del scraping para crear modelos e insertarlos.

Aplicando esta lógica al resto de la información ya se tendría el contenido de la base de datos listo. Ahora se pasará a la aplicación que manejará el usuario final.

4.3.2 - APLICACIÓN

El primer paso será crear un proyecto de node.js y electron. Se crea la carpeta donde se quiera tener el proyecto, en este caso se llamará App. Una vez situados en la carpeta dentro de la terminal que proporciona VS Code se ejecuta el siguiente código:

```
> npm init
```

Se rellenan los campos requeridos para el archivo package.json, a continuación se instala electron:

```
> npm install --save electron
```

A continuación se crea el archivo main.js.

```
const { app, BrowserWindow } = require('electron')
const remoteMain = require('@electron/remote/main')
process.env.ELECTRON_DISABLE_SECURITY_WARNINGS = 'true'

remoteMain.initialize()

function createWindow () {
  let win = new BrowserWindow({
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false
    }
  })
  win.maximize()

  win.loadFile('index.html')

  remoteMain.enable(win.webContents)
  win.webContents.openDevTools()
  win.on('closed', () => {
```

```
    win = null
  })
}
app.on('ready', createWindow)
```

Este es el fichero que contiene la configuración de electron y se encarga de crear las ventanas y controlar la aplicación, pero para añadir lógica asociada a un archivo html se creará un nuevo fichero index.js, asociado a otro llamado index.html. A continuación se instala electron remote:

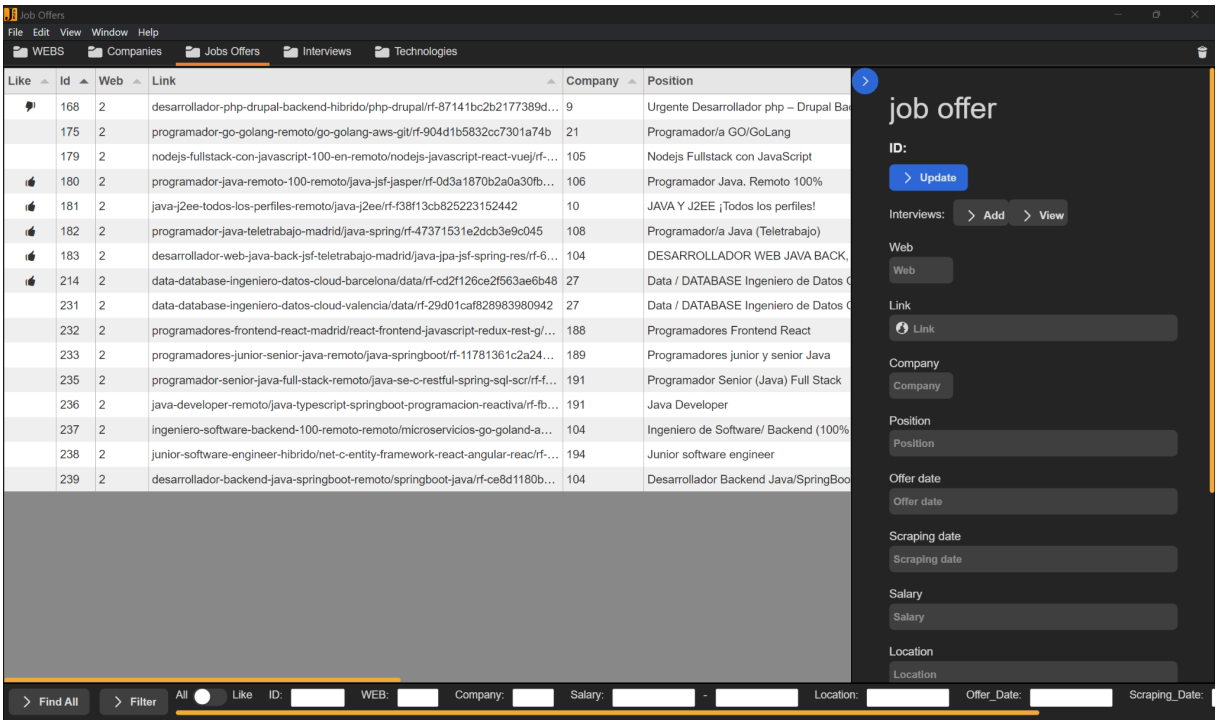
```
> npm install --save @electron/remote
```

A partir de aquí en general ya sería exactamente igual que crear una página web, se tendrá uno o varios archivos .html, con archivos .css para aplicar estilos y archivos .js para aplicar lógica, eventos y hacer la página reactiva.

Se intentará aplicar un diseño a la app con una distribución o esquema similar al planteado en el apartado de diseño, aplicando las distintas herramientas de html, css, photon y xel.

Básicamente la interfaz consta de 4 áreas, el header, que contiene la barra de navegación, el footer que contiene los elementos y botones para buscar y filtrar, y el main, que a su vez se divide en dos, lo que sería el contenedor principal donde se muestran los datos en tablas y un aside, o panel lateral, que mostrará información de la fila seleccionada en la tabla, pudiendo visualizar mejor sus datos y modificarlos.

Img. 16 - Interfaz de la aplicación recién iniciada



Img. 17 - Interfaz de la aplicación con datos cargados en la tabla jobs

Tanto el código html como css de la app en general es bastante estándar, pero si se ha tenido que realizar cierta investigación para implementar las pestañas de forma que cambie toda la interfaz en función de la pestaña clickada. A continuación se muestra parte del código html:

Dentro del header:

```
<nav>
   <x-tabs>
        <x-tab data-target = "#container_portals, #portals_aside,
        #footer_portals">
          <x-icon href="#folder"></x-icon>
          <x-label>WEBS</x-label>
        </x-tab>

        <x-tab data-target = "#container_companies, #companies_aside,
        #footer_companies">
          <x-icon href="#folder"></x-icon>
          <x-label>Companies</x-label>
        </x-tab>

        <x-tab selected data-target = "#container_jobs, #jobs_aside,
        #footer_jobs">
          <x-icon href="#folder"></x-icon>
```

```
          <x-label>Jobs Offers</x-label>
        </x-tab>

        <x-tab data-target = "#container_interviews, #interviews_aside,
        #footer_interviews">
          <x-icon href="#folder"></x-icon>
          <x-label>Interviews</x-label>
        </x-tab>

        <x-tab data-target = "#container_technologies, #technologies_aside,
        #footer_technologies">
          <x-icon href="#folder"></x-icon>
          <x-label>Technologies</x-label>
       </x-tab>
   </x-tabs>
</nav>
```

Dentro del main:

```
<section id="main_container">
      <div data-content id="container_portals">
      </div>

      <div data-content id="container_companies">
      </div>

      <div data-content id="container_jobs" class="active">
      </div>

      <div data-content id="container_interviews">
      </div>

      <div data-content id="container_technologies">
      </div>
</section>
```

Como se puede observar en el código, las pestañas tienen un data-target y los distintos contenedores dentro del main un data-content. Además el contenedor que se muestra en pantalla tendrá asociada la clase de css llamada ‘active’, el container de jobs la trae por defecto para que sea el contenedor que se muestra por defecto al iniciar la aplicación. Luego con un poco de lógica en javascript se va cambiando esta clase active a los contenedores que corresponda, tanto en el section como en el aside del main o el footer. A continuación se muestra el código en javascript:

```
// *** TABS ***
const targets = document.querySelectorAll('[data-target]')
const content = document.querySelectorAll('[data-content]')
targets.forEach(target => {
  target.addEventListener('click', () => {
    content.forEach(c => {
      c.classList.remove('active')
    })

    const t = document.querySelector(target.dataset.target)
    t.classList.add('active')

    if (JOBS.classList.contains('active')) {
      ASIDE_JOBS.classList.add('active')
      FOOTER_JOBS.classList.add('active')
    } else if (COMPANIES.classList.contains('active')) {
      ASIDE_COMPANIES.classList.add('active')
      FOOTER_COMPANIES.classList.add('active')
    } else if (PORTALS.classList.contains('active')) {
      ASIDE_PORTALS.classList.add('active')
      FOOTER_PORTALS.classList.add('active')
    } else if (INTERVIEWS.classList.contains('active')) {
      ASIDE_INTERVIEWS.classList.add('active')
      FOOTER_INTERVIEWS.classList.add('active')
    } else if (TECHNOLOGIES.classList.contains('active')) {
      ASIDE_TECHNOLOGIES.classList.add('active')
      FOOTER_TECHNOLOGIES.classList.add('active')
    }
  })
})
```

Img. 18 - Código javascript para el funcionamiento de las pestañas

Este código se encuentra dentro del archivo index.js, asociado al index.html. A continuación se muestra también cómo se importan las distintas librerías que se requieren para funcionar, por ejemplo Tabulator para las tablas o Sequelize como ORM para la conexión con la base de datos e interactuar con ella, veamos:

```
const Tabulator = require('tabulator-tables')

const shell = require('electron').shell

const { Sequelize, DataTypes, Model, QueryTypes } = require('sequelize')
```

Ahora se realiza la conexión a la base de datos:

```
const sequelize = new Sequelize('postgres://postgres:1415@localhost:5432/
job_offers')
```

Ahora se crea la clase o modelo del ORM que servirá para producir objetos equivalentes a las filas de la tabla job_offer de la base de datos:

```
// - JOBS -
class Job extends Model { }
Job.init({
  // Model attributes are defined here
  web: {
    type: DataTypes.INTEGER,
    allowNull: false
  },
  link: {
    type: DataTypes.STRING,
    allowNull: false
  },
  company: {
    type: DataTypes.INTEGER,
    allowNull: false
  },
  position: {
    type: DataTypes.STRING,
    allowNull: false
  },
  offer_date: {
    type: DataTypes.DATE,
    allowNull: false
  },
  scraping_date: {
    type: DataTypes.DATE
  },
  salary: {
    type: DataTypes.STRING
  },
  location: {
    type: DataTypes.STRING
  },
  experience: {
    type: DataTypes.INTEGER
  },
  contract: {
    type: DataTypes.STRING
  },
  description: {
    type: DataTypes.TEXT
  },
  like_dislike: {
    type: DataTypes.INTEGER
  }
}, {
  // Other model options go here
  sequelize, // We need to pass the connection instance
  modelName: 'Job', // We need to choose the model name
  tableName: 'job_offer',
  timestamps: true, // don't forget to enable timestamps!
  createdAt: false, // I don't want createdAt
  updatedAt: false // I want updatedAt to actually be called updateTimestamp
})
```

Img. 19 - Código javascript para crear la clase Job con el ORM Sequelize

Ahora se puede testear creando un objeto tipo job y guardandolo en la base de datos:

```
const job1 = Job.build({ web: 2, link: 'job1.com', company: 'Gran empresa',
position: 'developer invent', offer_date: '1299-01-18' })

job1.save()
```

O probar a cargar una oferta de la base de datos:

```
let job = await Job.findOne({
      where: {
            id: 5
      }
})
console.log(job.position)
```

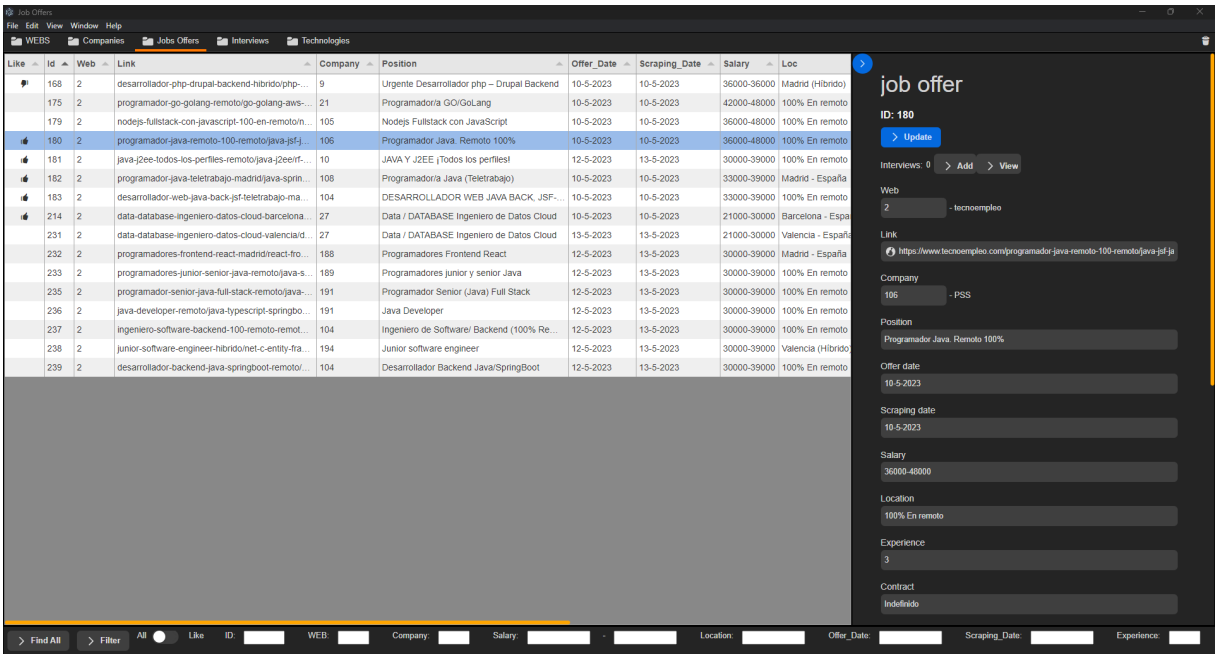
Ahora que ya se puede cargar información de la base de datos veamos cómo crear una tabla con Tabulator:

```
const tableJobs = new Tabulator(JOBS, {
  columns: [
    { title: 'Id', field: 'id', sorter: 'number' },
    { title: 'Web', field: 'web', sorter: 'number' },
    { title: 'Link', field: 'link', sorter: 'string', width: 500 },
    { title: 'Company', field: 'company', sorter: 'number' },
    { title: 'Position', field: 'position', sorter: 'string', width: 300 },
    { title: 'Offer_Date', field: 'offer_date', sorter: 'string', formatter: function(cell) {
      var value = cell.getValue()
      value = value.getDate() + '-' + (value.getMonth() + 1) + '-' + value.getFullYear()
      return value
    }},
    { title: 'Scraping_Date', field: 'scraping_date', sorter: 'string', formatter: function(cell){
      var value = cell.getValue()
      value = value.getDate() + '-' + (value.getMonth() + 1) + '-' + value.getFullYear()
      return value
    }},
    { title: 'Salary', field: 'salary', sorter: 'string' },
    { title: 'Loc', field: 'location', sorter: 'string' },
    { title: 'Ex', field: 'experience', sorter: 'number' },
    { title: 'Contract', field: 'contract', sorter: 'string' },
    { title: 'Description', field: 'description', sorter: 'string', width: 500 }
  ],
  selectable: 1,
  initialSort: [
    { column: 'id', dir: 'asc' } // sort by this first
  ]
})
```

Img. 20 - Código javascript para generar la tabla Jobs con Tabulator

Ahora que se tiene la tabla se pueden cargar los datos de la siguiente manera:

```
const btnAllJobs = document.getElementById('btn_jobs_find_all')

btnAllJobs.addEventListener('click', () => {
        const jobs = await Job.findAll()
        tableJobs.clearData()
        tableJobs.replaceData(jobs)
})
```



Img. 21 - Interfaz con datos de una oferta cargados en el panel aside

Con esto ya se tendrían cargados todos los datos en la tabla de ofertas laborales. Por supuesto hay mucho más código que puede resultar interesante como el que respecta a los botones para aplicar filtros, o la función, que al hacer click en una fila de la tabla, rellena los campos correspondientes en el aside, o el botón para añadir entrevistas a una oferta laboral, que nos la genera automáticamente y nos lleva a la pestaña de entrevistas, pero para no alargar excesivamente la memoria se dejará para el anexo de código.

4.4 - Pruebas

Durante todo el desarrollo del código se ha ido testeando el correcto funcionamiento del código, es decir cada método nuevo se a puesto a prueba varias veces, tanto al crearlo como al modificarlo directamente o cambiar partes del programa que pudieran afectarlo.

Además una vez avanzadas tanto la fase de scraping como la de la aplicación se han realizado pruebas en profundidad del buen funcionamiento de ambas aplicaciones trabajando en conjunto.

También se ha probado a ejecutar los programas una vez compilados y creado el ejecutable para comprobar el funcionamiento final.

4.5 - Documentación

Documentos del proyecto:

- **Memoria del Proyecto**
- **Anexos:**

    I - Código

    II - Instrucciones de instalación y ejecución

    III - Manual de uso

5 - Trabajos futuros

Dadas las características del proyecto y la necesidad de conexión a internet por parte de la aplicación, lo más conveniente como se ha comentado anteriormente es que en un futuro la aplicación sea completamente web, accediendo a ella desde cualquier navegador, registrando a cada usuario, y almacenando la información en servidores en vez de en los equipos de los propios usuarios.

Pero esta forma de trabajar requería de crear una API, un servidor etc. Motivos por los que dado los límites de tiempo requeridos por el cliente, se ha optado por una primera solución más sencilla, monousuario, que trabaja a nivel local. Aún así esta solución está creada ya con tecnologías web, que no solo permiten su uso en cualquier sistema operativo, si no que también facilitarán la adaptación del proyecto a su futura versión web.

Además aunque en esta versión inicial sólo se extraen ofertas de dos sitios web, la idea sería desarrollar más adelante el código necesario para scrapear otros sitios web, como LinkedIn, Infojobs, etc.

Pero esto aumentaría la necesidad de solucionar un problema que si bien se ha tenido en cuenta en esta primera versión, no se ha visto necesario solucionar aún. El problema, sería que al aumentar el número de sitios webs distintos, sería mucho más probable que se repitieran ofertas laborales, que siendo la misma estuvieran anunciadas a la vez en distintas webs.

Aunque no se han aplicado soluciones en esta versión si se ha tenido en cuenta para desarrollos futuros. A pesar de que no se han implementado de momento, si se ha realizado una profunda investigación de posibles soluciones.

Para solucionar esto seguramente se podrían aplicar varias estrategias, pero la principal que se ha considerado sería la de aplicar un algoritmo de coincidencia o similitud. Investigando el tema se ha encontrado que ya existe uno, llamado la similitud de Jaro–Winkler, que da como resultado un valor entre 0 y 1 de grado de similitud. Como se puede observar en los siguientes enlaces, este algoritmo es bastante fácil de implementar en el lenguaje de programación que se desee:

https://www.analyticslane.com/2020/06/24/la-similitud-de-jaro-winkler/

```
# Python3 implementation of above approach
from math import floor, ceil

# Function to calculate the
# Jaro Similarity of two s
def jaro_distance(s1, s2):

    # If the s are equal
    if (s1 == s2):
        return 1.0

    # Length of two s
    len1 = len(s1)
    len2 = len(s2)

    # Maximum distance upto which matching
    # is allowed
    max_dist = floor(max(len1, len2) / 2) - 1

    # Count of matches
    match = 0

    # Hash for matches
    hash_s1 = [0] * len(s1)
    hash_s2 = [0] * len(s2)

    # Traverse through the first
    for i in range(len1):

        # Check if there is any matches
        for j in range(max(0, i - max_dist),
                       min(len2, i + max_dist + 1)):

            # If there is a match
            if (s1[i] == s2[j] and hash_s2[j] == 0):
                hash_s1[i] = 1
                hash_s2[j] = 1
                match += 1
                break
```

```
    # If there is no match
    if (match == 0):
        return 0.0

    # Number of transpositions
    t = 0
    point = 0

    # Count number of occurrences
    # where two characters match but
    # there is a third matched character
    # in between the indices
    for i in range(len1):
        if (hash_s1[i]):

            # Find the next matched character
            # in second
            while (hash_s2[point] == 0):
                point += 1

            if (s1[i] != s2[point]):
                t += 1
            point += 1
    t = t//2

    # Return the Jaro Similarity
    return (match/ len1 + match / len2 +
            (match - t) / match)/ 3.0

# Driver code
s1 = "CRATE"
s2 = "TRACE"

# Prjaro Similarity of two s
print(round(jaro_distance(s1, s2),6))

# This code is contributed by mohit kumar 29
```

Img. 22 - Código python para implementar el algoritmo de similitud Jaro-Winkler

Output: 0.733333

GeeksforGeeks. *Jaro and Jaro-Winkler similarity.* <https://www.geeksforgeeks.org/jaro-and-jaro-winkler-similarity/> [Consulta: 18 de mayo de 2023]

Gracias a este algoritmo se podrían recorrer los resultados de la base de datos, comparando por ejemplo el título del puesto de trabajo, el nombre de la compañía, o la descripción del trabajo, y se podrían encontrar coincidencias sin la necesidad de que por ejemplo el título coincida al 100%, ya que pueden haberse escrito de forma ligeramente diferente de una web a otra a pesar de ser la misma oferta.

El principal trabajo de implementar este método sería testear a partir de qué porcentaje de coincidencia se puede asegurar que dos ofertas son las mismas. Básicamente habría que realizar múltiples pruebas y analizar los resultados para valorar correctamente qué porcentaje de coincidencia aplicar para eliminar ofertas repetidas.

6 - Conclusiones

Para la realización de este proyecto se han encontrado diversas dificultades principalmente en la segunda fase, la de la aplicación javascript, dado que no es un lenguaje con el que se tuviera mucha experiencia.

Especialmente se ha dedicado bastante tiempo a investigar distintos ORM para bases relacionales de javascript, ya que anteriormente sólo se habían realizado pequeños ejercicios conectando a bases no relacionales de MongoDB.

Así pues, hubo que informarse sobre qué opciones eran las más utilizadas y aconsejadas hoy en día para javascript y bases SQL. Una vez elegido Sequelize como ORM, también hubo que leer bastante de su documentación para conseguir hacerlo funcionar en todos los aspectos que necesitaba la app.

https://sequelize.org/

Otro aspecto de la aplicación al que se tuvo que dedicar bastante tiempo de investigación y desarrollo fue el de la creación de las tablas que muestran la información. Esta es una parte fundamental de la aplicación, y además ocupa la mayor parte de la pantalla, por que era muy importante que se vieran y funcionaran bien, así como que reaccionen a las necesidades del usuario para adaptar las columnas, u ordenarlas de forma ascendente o descendente según la columna y fueran reactivas para detectar clicks en una determinada fila o celda como la del like.

Es por ello por lo que aunque al principio se empezó a desarrollar en javascript el código desde 0, para hacer por ejemplo, que las columnas se pudieran cambiar de

anchura manualmente por el usuario, rápidamente se descartó este camino debido a lo extenso y complicado que se estaba volviendo el código y que aún habían muchas funcionalidades por implementar. Por ello mismo se volvió a realizar una tarea de investigación de distintas librerías de javascript con herramientas para trabajar con tablas más fácilmente. En el siguiente enlace podemos ver un ejemplo de algunas librerías:

https://www.jqueryscript.net/blog/best-data-table-grid.html

Entre ellas se encuentra Tabulator, que fue la elegida finalmente.

https://tabulator.info/

Pero de nuevo también una vez seleccionada se ha tenido que estudiar detenidamente la documentación para poder llevar a cabo todas las funcionalidades necesarias para la app.

Al final se ha podido trabajar y mejorar en el conocimiento no solo de javascript, si no también de diferentes librerías, y probar cómo se integran dentro de un proyecto. También se ha tenido que repasar y profundizar en conceptos de css para que la interfaz se muestre siempre correctamente y se adapte a distintas proporciones y usos.

7 - Bibliografía y webgrafía

Libros:

-Emiliano Barrios, A. (2019). *CSS*. Madrid: GRUPO ANAYA S.A.

-Wexler, J. (2019). *Get Programming with Node.js.* Shelter Island, NY: Manning Publications Co.

Webs:

-GeeksforGeeks. *Jaro and Jaro-Winkler similarity.* <https://www.geeksforgeeks.org/jaro-and-jaro-winkler-similarity/> [Consulta: 18 de mayo de 2023]

-AnalyticsLane. *La similitud de Jaro–Winkler.* <https://www.analyticslane.com/2020/06/24/la-similitud-de-jaro-winkler/> [Consulta: 18 de mayo de 2023]

-Mozilla. *Introducción a Express/Node.* <https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction> [Consulta: 18 de mayo de 2023]

-Amazon AWS. *¿Qué es Python? - Explicación del lenguaje Python - AWS*. <https://aws.amazon.com/es/what-is/python/> [Consulta: 18 de mayo de 2023]

-DataScientest. *PyCharm: Todo sobre el IDE de Python más popular.* <https://datascientest.com/es/pycharm> [Consulta: 18 de mayo de 2023]

-FreeCodeCamp. *Cómo codificar un scraping Bot con Selenium y Python.* <https://www.freecodecamp.org/espanol/news/como-codificar-un-scraping-bot-con-selenium-y-python/> [Consulta: 18 de mayo de 2023]

-PyPI. *Pscycopg2.* <https://pypi.org/project/psycopg2/> [Consulta: 18 de mayo de 2023]

-Platzi. *PostgreSQL: qué es, cómo funciona y cuáles son sus ventajas.* <https://platzi.com/blog/que-es-postgresql/> [Consulta: 18 de mayo de 2023]

-Tabulator. *Tabulator Info.* <https://tabulator.info/> [Consulta: 24 de abril de 2023]

-jQueryScript. *10 Best Data Table/Grid Systems In JavaScript (2023 Update)*. <https://www.jqueryscript.net/blog/best-data-table-grid.html> [Consulta: 21 de abril de 2023]

-Sequelize. <https://sequelize.org/> [Consulta: 1 de abril de 2023]

-DocsPython. *venv — Creación de entornos virtuales.* <https://docs.python.org/es/3.8/library/venv.html#:~:text=Un%20entorno%20virtual%20es%20un,parte%20de%20tu%20sistema%20operativo> [Consulta: 1 de marzo de 2023]

-OpenWebinars. *Qué es y por qué usar el Web Scraping.* <https://openwebinars.net/blog/que-es-por-que-usar-web-scraping/> [Consulta: 19 de marzo de 2023]

-TIOBE. *Index.* <https://www.tiobe.com/tiobe-index/> [Consulta: 15 de mayo de 2023]

-DBEngines.*Ranking popularity*. <https://db-engines.com/en/ranking> [Consulta: 15 de mayo de 2023]

-Google. *Google Docs: Online Document Editor.* <https://www.google.com/docs/about/> [Consulta: 6 de abril de 2023]

-Draw.io. *drawio-app.* <https://drawio-app.com/> [Consulta: 1 de marzo de 2023]

-Photopea. <https://www.photopea.com/> [Consulta: 1 de marzo de 2023]

-Python. <https://www.python.org/> [Consulta: 1 de marzo de 2023]

-JetBrains. *PyCharm: el IDE de Python para desarrolladores profesionales.* <https://www.jetbrains.com/es-es/pycharm/> [Consulta: 1 de marzo de 2023]

-Selenium. <https://www.selenium.dev/> [Consulta: 1 de marzo de 2023]

-ChromeDriver. <https://chromedriver.chromium.org/> [Consulta: 1 de marzo de 2023]

-SQLAlchemy. <https://www.sqlalchemy.org/> [Consulta: 1 de marzo de 2023]

-Alembic. <https://alembic.sqlalchemy.org/en/latest/index.html> [Consulta: 1 de marzo de 2023]

-PyPi. *psycopg2.* <https://pypi.org/project/psycopg2/> [Consulta: 1 de marzo de 2023]

-PostgreSQL. <https://www.postgresql.org/> [Consulta: 21 de marzo de 2023]

-pgAdmina. <https://www.pgadmin.org/download/> [Consulta: 21 de marzo de 2023]

-VisuelStudioCode.<https://code.visualstudio.com/> [Consulta: 15 de abril de 2023]

-Node.js. <https://nodejs.org/es> [Consulta: 15 de abril de 2023]

-Electron.js. <https://www.electronjs.org/es/> [Consulta: 15 de abril de 2023]

-Sequelize. <https://sequelize.org/> [Consulta: 15 de abril de 2023]

-Xel. *xel-toolkit*. <https://xel-toolkit.org/> [Consulta: 15 de abril de 2023]

8 - Anexos