



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**título del TFG
Documentación Técnica**



Presentado por nombre alumno
en Universidad de Burgos — 27 de junio
de 2023

Tutor: nombre tutor

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	iv
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	2
A.3. Estudio de viabilidad	9
Apéndice B Especificación de Requisitos	13
B.1. Introducción	13
B.2. Objetivos generales	13
B.3. Catálogo de requisitos	13
B.4. Especificación de requisitos	13
Apéndice C Especificación de diseño	15
C.1. Introducción	15
C.2. Diseño de datos	15
C.3. Diseño procedimental	17
C.4. Diseño arquitectónico	17
Apéndice D Documentación técnica de programación	21
D.1. Introducción	21
D.2. Estructura de directorios	21
D.3. Manual del programador	23

D.4. Compilación, instalación y ejecución del proyecto	23
D.5. Pruebas del sistema	23
Apéndice E Documentación de usuario	25
E.1. Introducción	25
E.2. Requisitos de usuarios	25
E.3. Instalación	25
E.4. Manual del usuario	25
Apéndice F Anexo de sostenibilización curricular	27
F.1. Introducción	27
Bibliografía	29

Índice de figuras

Índice de tablas

B.1. CU-1 Nombre del caso de uso.	14
---	----

Apéndice A

Plan de Proyecto Software

A.1. Introducción

El proyecto propuesto se centra en el desarrollo de la herramienta *OLIVIA-Finder*, la cual ha sido diseñada con el objetivo de extraer datos de paquetes y sus dependencias de los repositorios de software *CRAN*, *Bioconductor*, *PyPI* y *npm*.

Dado que la herramienta en sí misma carece de elementos visuales o atractivos más allá de la presentación de los datos en forma de una lista de enlaces entre nodos de la red, se ha decidido complementarla con un análisis básico de las redes generadas desde la perspectiva de la ciencia de redes. Esto se realiza con el propósito de evitar que el trabajo resulte monótono y brindar un enfoque más completo y enriquecedor al proyecto.

En general, el proyecto se puede describir en tres etapas fundamentales. La primera etapa consiste en una exhaustiva investigación y documentación, donde se realiza un estudio en profundidad de los repositorios *CRAN*, *Bioconductor*, *PyPI* y *npm*, así como de las técnicas y herramientas utilizadas para la extracción de datos y análisis de redes de dependencias. Esta etapa sienta las bases teóricas necesarias para comprender el contexto en el que se desarrollará la herramienta y el análisis posterior.

La segunda etapa se enfoca en el diseño y desarrollo de la herramienta *OLIVIA-Finder*. Aquí, se aplican los conocimientos adquiridos durante la etapa de investigación para implementar una solución eficiente y robusta que permita la extracción de datos de los repositorios mencionados. Se deben tener en cuenta diversos aspectos técnicos, como el manejo de solicitudes web, el procesamiento de datos y la manipulación de estructuras de red.

La tercera etapa del proyecto implica el análisis experimental de las redes generadas. Una vez obtenidos los datos de los paquetes y sus dependencias, se aplican técnicas de la ciencia de redes para examinar características importantes, como la *centralidad de grado*, el algoritmo *PageRank* y otras métricas relevantes. Este análisis proporciona una comprensión más profunda de la estructura y las propiedades de las redes de dependencias en los repositorios estudiados, permitiendo identificar paquetes críticos, vulnerabilidades potenciales y relaciones significativas entre los elementos de la red.

A.2. Planificación temporal

En el ámbito de los proyectos modernos de software, es común utilizar metodologías ágiles, como *Scrum* o *Kanban*. Estas metodologías reconocen la naturaleza dinámica del proceso iterativo que implica el establecimiento de requisitos, diseño, desarrollo y validación de un producto de software. Por lo tanto, se enfocan en la planificación adaptativa y la mejora continua a través de entregas tempranas.

Sin embargo, en nuestro caso, debemos cuestionar la aplicabilidad práctica de las metodologías ágiles tal como están concebidas. Esto se debe a que el conjunto de partes interesadas, que se limita a un cliente académico prototípico, y sobre todo al hecho de que el equipo de trabajo es unipersonal. Estas circunstancias particulares plantean dudas sobre la eficacia de la implementación de las metodologías ágiles en nuestro contexto.

No obstante, podemos establecer similitudes entre nuestro proceso y el marco de trabajo *Scrum*, debido a la presencia de *sprints*. Aunque no hemos seguido rigurosamente la estructura de los *sprints* debido a la falta de objetivos claramente definidos en el proyecto, los *sprints* nos han permitido representar la actividad realizada y las etapas en las que hemos dividido el trabajo.

Por otro lado, se ha de tener en cuenta que las tareas de investigación incluidas en el proyecto son difíciles de planificar. El proceso de investigación implica continuos replanteamientos y el alcance de los resultados debe ser constantemente modelado o redefinido a medida que avanzamos dentro del límite temporal del que se dispone.

Sprint 0: Kick of meeting

Este *sprint* representó nuestra primera aproximación a la temática del proyecto. Desde el principio, quedó claro que el modelo matemático pro-

porcionado por OLIVIA en el Trabajo de Fin de Grado anterior estaba más allá de nuestra comprensión absoluta, y nuestro enfoque se centraría principalmente en la experimentación.

Por lo tanto, fue necesario realizar un esfuerzo inicial para comprender los fundamentos básicos de lo que el modelo de OLIVIA nos permitía hacer. En este sentido, los Jupyter Notebooks proporcionados en el TFG anterior fueron de gran utilidad, ya que mostraban la funcionalidad en casos de estudio concretos y ofrecían un análisis complementario.

Desde el principio, nos encontramos con problemas técnicos. En primer lugar, las dependencias de OLIVIA requieren una actualización, ya que se están utilizando versiones algo desactualizadas de algunos paquetes y herramientas de análisis, y el *Dependabot de GitHub* insiste en su actualización “*Bump numpy from 1.18.5 to 1.22.0*”. En concreto, se ha identificado que esta versión de *NumPy* presenta vulnerabilidades de alto riesgo, como la referida al *NumPy NULL Pointer Dereference*.

Sin embargo, no es posible actualizar la biblioteca, ya que existen métodos esenciales para la funcionalidad implementada en OLIVIA que se han vuelto obsoletos en la versión actualizada. Además, en cuanto a la versión de Python, el código debe ejecutarse en la versión 3.8 debido a la presencia de dependencias clave para OLIVIA, como *intbitset*, que no están disponibles para otras versiones de Python. Esto supuso un problema al ejecutar los notebooks en *Google Colab* debido a las dificultades para instalar la versión específica de Python que requeríamos.

Además, fue necesario realizar una introducción a la temática mediante una lectura superficial de la memoria del TFG de OLIVIA. Esto también nos permitió familiarizarnos con el hecho de que estas memorias se redactan utilizando \LaTeX , una tecnología que nos resultaba completamente desconocida en ese momento.

En conclusión, este *sprint* inicial implicó la familiarización con el modelo de OLIVIA, la comprensión de sus fundamentos y la resolución de desafíos técnicos relacionados con las dependencias y las versiones de Python. Además, se llevó a cabo una introducción a la temática a través del estudio de la memoria del TFG anterior, lo que también nos permitió adquirir conocimientos sobre el uso de \LaTeX en la redacción de documentos científicos.

La duración de este *sprint* ha sido de 30 días aproximadamente, realizando 2 reuniones y con unas 25 horas de trabajo.

Sprint 1 - Data collection

En esta etapa de investigación nos enfocamos en el manejo de datos. Inicialmente, utilizamos el conjunto de datos de *libraries.io*, el cual resultó útil desde una perspectiva histórica. Sin embargo, presentó importantes limitaciones, como la falta de actualización periódica. Al enfrentarnos a los desafíos derivados de este conjunto de datos, pudimos constatar que trabajar con grandes volúmenes de datos no es trivial. Fue necesario utilizar un disco duro externo para almacenar el conjunto de datos, del cual solo nos interesaba la lista de enlaces entre paquetes y dependencias contenida en uno de los archivos CSV. Debido a la gran cantidad de líneas y su considerable tamaño, resultó difícil manipular y filtrar los datos.

La falta de actualización de los datos fue un aspecto clave a mejorar, por lo cual exploramos otras fuentes de información, como la API de *libraries.io* y el conjunto de datos de *BigQuery* proporcionado por los mismos desarrolladores. El uso de la API quedó descartado debido a sus limitaciones técnicas para realizar un escaneo completo del repositorio. Por otro lado, el conjunto de datos de *BigQuery* también presentó problemas similares a los archivos CSV en términos de falta de actualización.

Como alternativa, se propuso utilizar técnicas de *web scraping* para recolectar información de los sitios web de los repositorios de software de nuestro interés, comenzando con la recolección de información de CRAN. Este esfuerzo dio resultados positivos, y logramos obtener una lista actualizada de la red de CRAN. Los éxitos obtenidos en este proceso nos llevaron a considerar el desarrollo de una herramienta que pudiera obtener esta información para los repositorios de interés, ya que esto generaría un nuevo conjunto de datos utilizable en OLIVIA y en trabajos anteriores que hayan utilizado datos de *libraries.io*, lo que permitiría actualizar sus resultados de manera significativa.

Como resultado al finalizar este *sprint*, logramos obtener una red actualizada de CRAN gracias a que desarrollamos un prototipo básico pero todavía inmaduro de un código en Python que nos permitió realizar la recolección de datos.

Es importante destacar que durante este *sprint* se abordó uno de los principales desafíos que hemos enfrentado. Como era de esperar, los servidores web implementan medidas de seguridad para evitar comportamientos maliciosos. Debido a la naturaleza del proceso de escaneo de un repositorio, que implica realizar numerosas solicitudes web a un mismo servidor desde una misma dirección IP en un período de tiempo relativamente corto, esta

actividad a menudo resulta en la prohibición temporal de acceso al servidor web para los equipos asociados a esa dirección IP. Para solucionar este problema, se implementó la funcionalidad de ocultar las solicitudes detrás de servidores proxy, que enmascaran la dirección IP de origen al servidor web. Por lo tanto, fue necesario desarrollar esta funcionalidad para que fuera posible llevar a cabo esta tarea.

En conclusión, este *sprint* fue clave para obtener un conjunto de datos actualizado y establecer los primeros pasos en el desarrollo de una herramienta de recolección de datos. Además, se logró resolver el desafío de la prohibición de acceso a los serv

idores web mediante la implementación de rotación de proxy. Estos avances sientan las bases para continuar con el desarrollo del proyecto y alcanzar los objetivos planteados.

La duración de este *sprint* ha sido aproximadamente de 30 días, que en nuestra metodología de trabajo equivalen aproximadamente a 2 reuniones y a 60 horas de trabajo.

Sprint 2 - Library implementation and interpretation of some data

En esta etapa, nuestro enfoque se centró en el desarrollo de una herramienta genérica con el propósito de llevar a cabo una extracción sencilla de la red de dependencias de los repositorios *CRAN*, *Bioconductor*, *PyPI* y *npm*. Nos enfrentamos al primer desafío de obtener una lista completa de los paquetes disponibles en cada repositorio, que serviría como punto de partida para nuestra recopilación de datos. Cabe destacar que esta tarea no es sencilla en la mayoría de los casos, ya que dichas listas no siempre están disponibles.

Es importante mencionar que cada repositorio de software presenta sus peculiaridades distintivas. En el caso particular de *CRAN* y *Bioconductor*, el proceso se basó exclusivamente en técnicas de *web scraping*, aprovechando los datos de interés que se encuentran directamente en las listas de paquetes publicadas en sus respectivos sitios web.

El análisis de *CRAN* resultó ser el más sencillo de todos, ya que su página web es simple, robusta y aparentemente sólida a lo largo del tiempo, lo que proporciona una implementación bastante estable para este repositorio. Por otro lado, inicialmente se esperaba que *Bioconductor* fuese más sencillo debido a la menor cantidad de paquetes en comparación con los otros

repositorios en los que estamos trabajando. Sin embargo, nos encontramos con un problema en el servidor web de *Bioconductor*, específicamente en la página que muestra el listado de paquetes disponibles, ya que no era accesible mediante una simple solicitud web, como las que solemos realizar utilizando la biblioteca *requests* de Python.

En *Bioconductor*, se utilizaba JavaScript para cargar dinámicamente la lista de paquetes en tiempo de ejecución sobre la página. Esta situación dificulta la obtención de los datos, ya que la biblioteca *requests* no es compatible con la carga de JavaScript. Como alternativa, tuvimos que recurrir a la biblioteca *selenium*, la cual ofrece funcionalidades más avanzadas al actuar como un navegador *headless* (sin interfaz gráfica) que se comporta de manera similar a un navegador de escritorio convencional al que estamos acostumbrados, pero que admite la automatización de tareas. Gracias a *selenium*, logramos extraer la lista de paquetes de *Bioconductor* y, a partir de ella, procedimos de manera similar a como lo habíamos hecho anteriormente, obteniendo los datos de interés mediante técnicas de *web scraping*.

El repositorio *PyPI*, también publica una lista de paquetes. Sin embargo, es importante destacar que esta lista contiene muchos paquetes obsoletos e inexistentes, lo que la convierte en un punto de partida necesario pero no del todo óptimo. Utilizando esta lista de paquetes y aprovechando la API proporcionada por *PyPI* para obtener metadatos de los paquetes, pudimos extraer la red de dependencias correspondiente.

En esta etapa, nos percatamos del tiempo considerablemente elevado requerido para llevar a cabo esta recopilación, así como de la necesidad de cuidar la implementación de la herramienta para evitar el desperdicio de memoria. Estos problemas surgidos debido al tamaño de los datos nos proporcionan una perspectiva clara de la importancia de gestionar eficientemente los recursos cuando se trabaja con cantidades masivas de información. En cuanto al tiempo necesario para la recolección, se realizó un esfuerzo por optimizar el proceso mediante técnicas de concurrencia, lo cual nos permitió realizar solicitudes web de forma concurrente en lugar de secuencial, como habíamos estado haciendo hasta ese momento. Estas mejoras significativas en el rendimiento de la herramienta se tradujeron en una reducción significativa del tiempo requerido y el consumo de memoria.

Finalmente, logramos generar el conjunto de datos para el repositorio *npm*. Es importante destacar que este repositorio ha sido el más desafiante de abordar. En primer lugar, no existe una forma sencilla de obtener una lista completa de los paquetes existentes en *npm*. Además, el repositorio oficial de paquetes no es único, ya que existen repositorios espejo (*mirrors*)

alternativos que difieren tanto en el número de paquetes como en los paquetes que contienen. Para abordar este problema, decidimos utilizar la lista de paquetes que pudimos extraer de uno de estos *mirrors* y complementar con los paquetes que teníamos en el conjunto de datos de *libraries.io*. De esta manera, obtuvimos una lista de nombres de paquetes a los cuales dirigir nuestros esfuerzos. A partir de esta lista, aplicamos la metodología utilizada previamente para el resto de los repositorios. En el caso de *npm*, gracias a su API, pudimos recopilar los metadatos necesarios para construir el conjunto de datos correspondiente.

Llevar a cabo este *sprint* ha supuesto una duración de 60 días, lo que aproximadamente corresponde con unas 3 reuniones y alrededor de 150 horas de trabajo.

Sprint 3 - Library refactoring

Tras el análisis de los datos recolectados, observamos que los conjuntos de datos de *libraries.io* proporcionan información sobre todas las versiones existentes de un paquete. En otras palabras, consideramos como *dependencias* de un paquete todas las dependencias que hayan existido para cada una de sus versiones. Sin embargo, desde una perspectiva de desarrollo de software, esto no es correcto, ya que sobrecargamos las dependencias de un paquete con dependencias de versiones antiguas que ya no se utilizan en el ciclo de vida actual de esas bibliotecas. Por lo tanto, es importante tener en cuenta esta información en los análisis posteriores que realicemos, donde será necesario aplicar un filtrado adecuado si deseamos utilizar los datos de *libraries.io*. Otro aspecto interesante es que no todas las *dependencias* de un paquete se encuentran presentes en el repositorio al que pertenece ese paquete, e incluso es posible que no utilicen el mismo lenguaje de programación. Tomemos como ejemplo un paquete en Python que depende de un binario escrito en C. Este fenómeno es muy común en la red de *Bioconductor*, cuyos paquetes dependen en gran medida de paquetes existentes en *CRAN*. A estas dependencias las hemos bautizado como *dependencias foráneas*.

Con el objetivo de mejorar la funcionalidad y flexibilidad de la biblioteca, se llevó a cabo una refactorización del diseño para adaptarlo y permitir el uso y la combinación de diversas fuentes de datos. Se proporciona soporte para *web scraping*, conjuntos de datos en formato *CSV*, la API de *libraries.io* y repositorios de *GitHub*. El uso combinado de diferentes fuentes de datos nos brinda la capacidad de buscar en fuentes alternativas cuando un paquete solicitado no se encuentra en la fuente principal. El soporte de archivos *CSV* nos permite considerar los conjuntos de datos de *libraries.io* como fuente

de información, tanto a través de su API como de los archivos en sí. La implementación para repositorios de *GitHub* nos proporciona información adicional al utilizar *GitHub* como fuente de datos, ya que es el sistema de control de versiones por excelencia donde se encuentran la mayoría de los proyectos de software de código abierto publicados. De esta manera, tenemos acceso a un repositorio de un nivel inferior, ya que no pertenece a un gestor de paquetes específico de un lenguaje de programación, sino que se basa su relevancia principalmente en un ámbito más cercano al desarrollo. Además, la biblioteca ofrece otras funcionalidades, como la obtención en tiempo de ejecución de una red de dependencias transitiva para un paquete en particular. También proporciona persistencia de datos en forma de objetos serializados y la capacidad de exportar datos en formato *CSV* compatible con *OLIVIA*.

Una vez concluida esta etapa, procedimos a publicar los conjuntos de datos en *Zenodo*, con el fin de hacerlos accesibles para la comunidad científica. Además, dedicamos nuestros esfuerzos a mejorar la documentación del código fuente y generar una documentación completa de la biblioteca. Esta documentación está diseñada para ser accesible desde la web y se encuentra alojada en *GitHub Pages*. La publicación de los conjuntos de datos en *Zenodo* permite a otros investigadores y desarrolladores acceder a los datos recopilados y utilizarlos en sus propios proyectos o investigaciones. De esta manera, promovemos la transparencia y el intercambio de información entre la comunidad científica. En cuanto a la documentación de la biblioteca, nos esforzamos por ofrecer una guía clara y concisa sobre cómo utilizar la biblioteca, qué funcionalidades y características ofrece, así como ejemplos de uso. La documentación se ha estructurado de manera que sea fácil de navegar y buscar información relevante. Al alojarla en *GitHub Pages*, proporcionamos un acceso práctico y amigable para los usuarios, quienes pueden acceder a la documentación directamente desde el sitio web del proyecto en *GitHub*. Este *sprint* ha sido difícil de calcular el tiempo que ha llevado realizarlo, debido a que ha habido saltos hacia *sprints* anteriores cuando estamos trabajando en este. Ha sido uno de los más complejos. Se calcula un periodo de 60 días, 4 reuniones y aproximadamente 150 horas de trabajo.

Sprint 4 - Análisis de datos

En la etapa final de este estudio, se utilizaron los datos recopilados para realizar un análisis con una perspectiva evolutiva de las redes de dependencias presentes en los repositorios *CRAN*, *Bioconductor*, *PyPI* y *npm*, utilizando la ciencia de redes como marco de referencia.

En este análisis, se lleva a cabo una comparativa entre el estado de la red según los datos obtenidos de *libraries.io* y los datos obtenidos mediante *web scraping*. Se realiza un análisis desde el punto de vista de la centralidad de grado y el algoritmo *PageRank*, con el objetivo de identificar y argumentar cuáles son los paquetes más vulnerables y por qué. Además, se calculan algunas de las métricas proporcionadas por el modelo de red de *OLIVIA* y se establecen relaciones entre ellas.

Este análisis ofrece una perspectiva a nivel micro al examinar los extremos de la red, como los paquetes más destacados, y una visión más macro al analizar distribuciones y propiedades más generales, como el grado medio. Su objetivo principal es proporcionar una comprensión más profunda de la composición de estas redes de dependencias, sin abordar un análisis exhaustivo de toda la red.

Este *sprint* en cuanto a duración fue el más corto y de una duración de 15 días, en los que hubo 3 reuniones, y aproximadamente ocupó unas 30 horas de trabajo.

A.3. Estudio de viabilidad

Considerando la dirección y metas de las actividades planificadas, procederemos a evaluar la factibilidad de las mismas, teniendo en cuenta su enfoque como un proyecto de investigación, desarrollo e innovación.

Viabilidad económica

Presupuesto

Para realizar una estimación precisa del presupuesto del proyecto, hemos considerado diferentes aspectos relacionados con los costos involucrados. En primer lugar, hemos establecido una tarifa base de 15€ por hora de trabajo para el autor del proyecto, incluyendo los costos salariales, complementos y seguridad social. Cabe destacar que no se han identificado gastos significativos en material o hardware, a excepción del consumo eléctrico durante la recolección de datos.

En cuanto al software utilizado, nos complace informar que todos los productos empleados en el desarrollo del proyecto han sido de coste cero. Sin embargo, es importante mencionar que existe la posibilidad de incurrir en gastos adicionales si se decide utilizar servicios de computación en la nube de pago, aunque estos no son estrictamente necesarios para el proyecto.

En términos de *costos indirectos*, los cuales abarcan los gastos generales imputables al proyecto, los hemos incluido en la partida correspondiente. Siguiendo una práctica común, hemos asignado un *15 %* del presupuesto de personal directo como gastos indirectos. De esta manera, hemos obtenido una aproximación al presupuesto total del proyecto de $415 \text{ horas} \times 15\text{€/hora} \times 1,15 = 7158,75\text{€}$. Es importante señalar que esta cifra no contempla posibles gastos derivados de la publicación y difusión de los resultados, los cuales podrían añadirse en etapas posteriores.

En cuanto al salario del profesor, quien cuenta con una mayor experiencia y conocimientos especializados, se ha establecido una tarifa base de *30€* por hora. Considerando que ha habido un total de 14 reuniones, cada una con una duración aproximada de *1.5* horas, y aplicando los mismos gastos indirectos del *15 %*, el costo total asociado a estas reuniones sería de $14 \text{ reuniones} \times 1,5 \text{ horas} \times 30\text{€/hora} \times 1,15 = 724\text{€}$.

En resumen, según nuestras estimaciones, el desarrollo de este proyecto conllevaría un costo aproximado de *9283.5€*. Este presupuesto abarca los salarios del autor del trabajo y del profesor, los gastos indirectos correspondientes y otros aspectos operativos relevantes.

Por último, es importante mencionar que, debido a la naturaleza del proyecto como una iniciativa de código abierto e investigación, no se permite la venta ni la comercialización del producto resultante con fines comerciales. El objetivo principal es promover la colaboración y el intercambio de conocimientos en la comunidad científica.

Financiación

Los programas nacionales o europeos de financiación destinados a la *investigación, desarrollo e innovación (I+D+i)* podrían constituir una opción viable para sufragar total o parcialmente el proyecto. Por lo general, resulta necesario contar con una *afiliación* a un marco empresarial o académico específico que permita acceder a los diversos *instrumentos de financiación* disponibles.

Una interesante posibilidad que merece ser estudiada, y que ha cobrado relevancia en el contexto del movimiento de *Ciencia Abierta*, es el *crowdfunding*. Esta modalidad persigue la obtención de dinero para el proyecto mediante contribuciones económicas realizadas por personas interesadas en contribuir al progreso de la *investigación científica transparente* y con *resultados públicos, accesibles y gratuitos*. Algunos ejemplos son *Kickstarter* o *Indiegogo*.

En el ámbito del *crowdfunding*, es de una *importancia capital* presentar de forma atractiva la idea del proyecto, destacando su *relevancia e impacto* en la *comunidad científica* y en la sociedad en general. Asimismo, es fundamental establecer una estrategia de comunicación efectiva para alcanzar a potenciales colaboradores y motivarlos a realizar sus aportaciones económicas.

Viabilidad legal

La viabilidad legal del proyecto software que hemos desarrollado es favorable, considerando las siguientes características:

Naturaleza *opensource* y gratuita:

Todo el proyecto ha sido diseñado bajo una filosofía de código abierto y gratuito, lo que implica que el software y sus componentes están disponibles para ser utilizados, modificados y distribuidos sin restricciones. Esta elección nos permite fomentar la colaboración y el acceso abierto a la tecnología desarrollada.

La biblioteca que hemos desarrollado para el proyecto tiene una naturaleza experimental y no comercial. Esto implica que su propósito es explorar nuevas ideas y soluciones tecnológicas, sin intención de generar beneficios económicos directos a través de su venta o licenciamiento.

Recursos extraídos de la red:

Los recursos utilizados en el proyecto, como bibliotecas y herramientas, han sido obtenidos de fuentes abiertas y accesibles en Internet. Al utilizar estos recursos, hemos respetado las condiciones y términos de uso establecidos por los autores y las licencias correspondientes.

El conjunto de datos utilizado en el proyecto ha sido recolectado de fuentes accesibles en la red, donde cualquier persona puede acceder a ellos. Nos hemos esforzado en recopilar y proporcionar estos datos a la comunidad, siguiendo los principios de transparencia y acceso abierto. En el caso de uso de estos datos, solo exigimos la atribución a los autores del Trabajo Fin de Grado correspondiente.

Aprovechamos este momento para dar atribución al creador del conjunto de datos de *libraries.io* llamado *Tidelift*.

Licencia de código abierto:

Con respecto a la clasificación de la licencia adecuada para el proyecto, considerando los aspectos mencionados, una opción apropiada podría ser la

licencia *Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)*. Esta licencia permite utilizar, modificar y distribuir el software y sus componentes, siempre y cuando se atribuya adecuadamente a los autores originales y se comparta bajo una licencia similar. Esta elección refuerza los principios de acceso abierto y fomenta la colaboración en la comunidad de desarrolladores y usuarios del software.

Apéndice B

Especificación de Requisitos

B.1. Introducción

Una muestra de cómo podría ser una tabla de casos de uso:

B.2. Objetivos generales

B.3. Catálogo de requisitos

B.4. Especificación de requisitos

CU-1	Ejemplo de caso de uso
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-xx, RF-xx
Descripción	La descripción del CU
Precondición	Precondiciones (podría haber más de una)
Acciones	<ol style="list-style-type: none"> 1. Pasos del CU 2. Pasos del CU (añadir tantos como sean necesarios)
Postcondición	Postcondiciones (podría haber más de una)
Excepciones	Excepciones
Importancia	Alta o Media o Baja...

Tabla B.1: CU-1 Nombre del caso de uso.

Apéndice C

Especificación de diseño

C.1. Introducción

C.2. Diseño de datos

Estructuras de datos

La información relativa a la red de dependencias de paquetes se almacena en un formato híbrido que contiene varias estructuras de datos con distintos objetivos:

- Almacenar la información concreta de paquetes y dependencias del repositorio, en un formato que permita realizar de forma eficiente las operaciones más comunes relacionadas con el objeto de la biblioteca.
- Ser capaz de aceptar extensibilidad por si en un determinado momento hubiese que ampliar o variar los datos que almacenamos.

De esta forma, representamos cada paquete mediante una estructura de datos específica para contener los atributos que deseamos representar. La red de dependencias queda representada como un diccionario de paquetes. Esta representación permite acceder a un paquete de forma eficiente, lo cual resulta importante para el correcto funcionamiento de la herramienta.

Datos de entrada

Podemos construir la red de dependencias a través de un fichero CSV que represente la lista de enlaces. Los paquetes se pueden reconstruir a partir de una estructura tipo diccionario con los atributos implementados. Las listas de paquetes se dan en formato lista de objetos paquete.

Un problema que se ve es la redundancia de los nombres de nodos, pero realmente es necesario si queremos almacenar datos más concretos sobre la relación de dependencia, como la versión concreta de una dependencia que usa un paquete, etc.

Datos de salida

La red se puede exportar a formato de grafo dirigido de *NetworkX* y también como un diccionario de listas de dependencias.

La estructura de datos *Paquete* se puede exportar como un diccionario, mientras que un conjunto de paquetes se exportará como una lista de objetos *paquetes*.

Ficheros de entrada

Podemos emplear un archivo CSV como fuente de datos.

Podemos emplear objetos serializados de tipo *PackageManager*.

Ficheros de salida

Podemos generar ficheros *CSV* con la representación de la red o exportarlo como un dataframe de *Pandas*, lo que nos permite exportarlo a otros formatos como *JSON*, *HTML*, etc.

También podemos realizar un exportado mediante la serialización del objeto *PackageManager* como estructura global de persistencia para un repositorio. Este tipo de archivo se le ha dado una extensión *.olvpm* para poder identificarlo correctamente.

C.3. Diseño procedimental

C.4. Diseño arquitectónico

Modulo `olivia_finder.package`

Package es uno de los módulos base de *olivia-finder*, cuya funcionalidad radica en constituir una estructura de datos que contiene la representación del paquete. Está compuesto por atributos en formato *string* y una lista de dependencias (*lista de objetos paquete*).

En nuestra implementación, los atributos que almacenamos son el nombre, la versión, la URL y las dependencias. Las principales funcionalidades que implementa son la representación como un diccionario, la carga a partir de una estructura de tipo diccionario y la enumeración de las dependencias.

Modulo `olivia_finder.package_manager`

El módulo *Package Manager* es un wrapper de los datos que conforman la red de dependencias de un repositorio y de la funcionalidad del objeto *DataSource*, que representa la fuente de datos de un repositorio. Aporta funcionalidades para la carga, obtención y persistencia de los datos.

Para inicializar el objeto, necesitamos pasarle como argumento una lista de los *data sources* que deseemos usar. Está pensado para tener como mínimo un *data source* principal y una serie de *data sources* auxiliares opcionales, donde se realizará la búsqueda si no se obtiene un resultado mediante el principal.

Respecto a la carga de datos en sus estructuras, podemos emplear ficheros de persistencia *.olvpm* o archivos CSV con la lista de enlaces.

La obtención de datos se puede realizar de forma individual para un paquete o para una lista de paquetes. Una vez obtenidos los datos, si se desea, se pueden almacenar en la estructura interna de la clase, que es un diccionario, y recuperarlos desde ahí. Por lo tanto, si queremos usar los datos directamente desde las estructuras de la clase, primero debemos inicializar los datos.

Una funcionalidad interesante es la obtención de redes de dependencias transitivas a partir de un paquete y dada una profundidad de búsqueda.

La exportación de los datos se puede representar mediante un objeto diccionario de nodos que contienen listas de nombres de paquetes, es decir,

una lista de adyacencia. El principal formato de exportación que vamos a tratar son los archivos CSV en formato lista de enlaces, debido a su naturaleza legible y su integración con OLIVIA. También es interesante su representación como grafo dirigido de *NetworkX* (*nx.DiGraph*).

La persistencia de los datos se lleva a cabo mediante la serialización del objeto utilizando el módulo *pickle* de Python, o la generación de archivos CSV de lista de enlaces.

Se incorporan excepciones características para representar los problemas más comunes en un contexto más descriptivo sobre la actuación de la herramienta.

Paquete *olivia_finder.utilities*

El paquete *utilities* contiene un conjunto de módulos que implementan funcionalidades de utilidad para la aplicación, aunque no forman parte central de ella. Está compuesto por los siguientes módulos:

El módulo *config* implementa una funcionalidad para la carga de la configuración definida en el archivo *config.ini*¹, donde se especifican las configuraciones necesarias.

El módulo *exception* implementa una excepción genérica para la herramienta, que puede ser utilizada para capturar y manejar situaciones excepcionales.

El módulo *logger* implementa funcionalidades para llevar un registro de los eventos ocurridos durante la ejecución de la aplicación.

El módulo *singleton_decorator* implementa un patrón de diseño singleton basado en el decorador, que se puede aplicar a cualquier clase para garantizar que solo se cree una única instancia de dicha clase.

El módulo *utilities* proporciona algunas funcionalidades concretas y frecuentemente utilizadas en la aplicación.

Paquete *olivia_finder.myrequests*

Este paquete implementa la funcionalidad de realizar *peticiones web* de forma concurrente y mediante rotación de *proxies* y *user agents* para ocultar la identidad del realizador de la petición.

Está compuesto por los siguientes módulos y paquetes:

¹Archivo de configuración de la biblioteca

El módulo *proxy_handler* se encarga de gestionar los proxies que se utilizarán para realizar las peticiones. Lleva un control del número de usos de cada proxy y gestiona su rotación para evitar la repetición en peticiones concurrentes. Además, obtiene nuevos proxies cuando ha agotado los existentes. Implementa el patrón de diseño *singleton*.

El paquete *proxy_builders* implementa la funcionalidad de obtención de una lista de proxies de Internet. La funcionalidad está descrita en la clase abstracta *ProxyBuilder* del módulo *proxy_builders*, que se encarga de implementar las funcionalidades comunes y sirve como interfaz para las distintas clases que implementan *ProxyBuilder*. Estas clases se encuentran en los módulos *list_builder* y *ssl_proxies*.

El módulo *proxy_builders.list_builder* se encarga de obtener una lista de proxies almacenada en un servidor web en formato *txt*.

El módulo *proxy_builders.ssl_proxies* realiza la misma función, pero obtiene los datos de los proxies mediante *web scraping* de un sitio web que ofrece este servicio.

El módulo *useragent_handler* realiza una función similar al módulo *proxy_handler*, pero con los *user agents*. Implementa la obtención a través de un archivo estático que contiene una lista de *user agents* y también mediante *web scraping* de una página web que contiene un listado. Gestiona la rotación de los *user agents* para evitar su repetición. También implementa el patrón de diseño *singleton*.

El módulo *job* es un *wrapper* de los atributos de una petición web. Encapsula la URL de destino, los parámetros de la petición y almacena la respuesta del servidor.

El módulo *request_worker*, que hereda de la clase *Thread*, se encarga de realizar las peticiones web asignadas a él.

Por último, el módulo *request_handler* implementa toda la lógica del paquete *myrequests*. Se encarga de gestionar los trabajos de petición web a realizar y asignarlos a los *workers*, que se encargan de realizar estas peticiones de forma concurrente. Recoge los trabajos realizados y los devuelve.

Estos módulos y paquetes en conjunto permiten realizar peticiones web de forma eficiente, garantizando la anonimización del realizador de la petición mediante el uso de proxies y user agents rotativos.

Paquete `olivia_finder.data_source`

Este paquete implementa la funcionalidad de obtención de datos desde distintas fuentes.

En primer lugar, tenemos el módulo `data_source` que implementa una clase abstracta que sirve como interfaz para las distintas clases de este paquete. Un `data_source` debe implementar la lógica de obtención de una lista de paquetes disponibles y la obtención de los datos de los paquetes.

Tenemos 3 módulos que implementan `DataSource`:

El módulo `data_source.scrapers_ds` implementa la obtención de datos de los paquetes mediante *web scraping* y proporciona una interfaz para las implementaciones concretas de `scraper` para los repositorios de software. Las funcionalidades que aporta incluyen la generación y lanzamiento de trabajos de petición web correspondientes a la obtención de los datos de los paquetes, así como la recolección de los datos de estas peticiones.

Las implementaciones concretas del módulo `repository_scrapers.scrapers_ds` se encuentran dentro del subpaquete `repository_scrapers`, el cual contiene los módulos `repository_scrapers.bioconductor`, `repository_scrapers.cran`, `repository_scrapers.npm` y `repository_scrapers.pypi`, que realizan la labor descrita para cada uno de estos repositorios. Además, incluye el módulo `repository_scrapers.github`, que permite realizar una labor similar a través de la sección *Insights > Dependency graph > Dependencies*, que se puede habilitar en un repositorio de GitHub.

El módulo `data_source.csv_ds` proporciona soporte para usar un dataset en formato CSV como fuente de datos. La funcionalidad que ofrece es similar a la mencionada anteriormente, pero adaptada y optimizada para este tipo de archivo.

El módulo `data_source.librariesio_ds` realiza la misma función que los anteriores, pero utiliza la API de `data_source.libraries.io`. No todas las funcionalidades han podido ser implementadas en este módulo, por ejemplo, la lista de paquetes de un repositorio no está soportada. Sin embargo, ofrece un conjunto de datos más amplio al acceder directamente al conjunto de datos de `libraries.io` a través de su API. Para poder utilizar esta funcionalidad, es necesario disponer de una clave de API de `libraries.io`.

Apéndice D

Documentación técnica de programación

D.1. Introducción

Se recoge en este apartado la información más relevante para la extensión o adaptación del código de la biblioteca.

D.2. Estructura de directorios

El proyecto de software está disponible en el repositorio público <https://github.com/dab0012/olivia-finder>. Este repositorio incluye los objetos de persistencia, los datos extraídos en los análisis y todo el conjunto de scripts y demás material utilizado. Sin embargo, si lo que le interesa son los datos generados de las redes de dependencias, es recomendable acceder a ellos desde los siguientes mirrors en Zenodo: `xxxxxxxxxxxxx`.

Estructura de directorios del repositorio

El repositorio alberga una amplia gama de contenido, que incluye tanto la biblioteca Olivia-Finder que hemos desarrollado como la biblioteca OLIVIA realizada en el anterior Trabajo de Fin de Grado. Además, se encuentra disponible documentación detallada sobre ambas bibliotecas, que proporciona información exhaustiva sobre su funcionamiento y características.

Dentro del repositorio, también se pueden encontrar una serie de notebooks que ofrecen demostraciones interactivas de la funcionalidad de las

bibliotecas. Estos notebooks permiten a los usuarios explorar y experimentar con los datos generados por las redes que hemos creado, lo que facilita la comprensión y el análisis de los resultados obtenidos.

Además del código y la documentación, el repositorio cuenta con una variedad de material auxiliar, como scripts y funcionalidades adicionales. Estos recursos complementarios ofrecen soporte adicional para aquellos interesados en ampliar o personalizar las funcionalidades de las bibliotecas.

```
olivia_finder
|-- docs
|-- notebooks
|   |-- common
|   |-- olivia
|   |-- olivia_finder
|   |-- resources
|   '-- results
|       |-- csv_datasets
|       |-- network_models
|       |-- package_lists
|       '-- package_managers
|-- olivia
|   |-- docs
|   '-- olivia
'-- olivia_finder
    |-- diagrams
    '-- olivia_finder
```

Sobre la raíz del repositorio, se encuentra el directorio *docs* contiene la documentación de la biblioteca, mientras que el directorio *notebooks* contiene los notebooks que ofrecen demostraciones interactivas de las bibliotecas.

Dentro de *notebooks*, el directorio *common* contiene notebooks que ofrecen funcionalidades comunes a ambas bibliotecas, mientras que los directorios *olivia* y *olivia_finder* contienen notebooks que ofrecen funcionalidades específicas de cada biblioteca. El directorio *resources* contiene los algun scripts y recursos adicionales de utilidad. Por último, el directorio *results* contiene los datos generados por las redes de dependencias, que se encuentran organizados en subdirectorios según su tipo.

En la raíz del repositorio, se encuentran los directorios *olivia* y *olivia_finder*, que contienen las bibliotecas OLIVIA y Olivia-Finder, respectivamente. Dentro de cada uno de estos directorios, se encuentra el directorio *olivia* o *olivia_finder*, que contienen el código de las bibliotecas.

D.3. Manual del programador

D.4. Compilación, instalación y ejecución del proyecto

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

- E.1. Introducción
- E.2. Requisitos de usuarios
- E.3. Instalación
- E.4. Manual del usuario

Apéndice F

Anexo de sostenibilización curricular

F.1. Introducción

Este anexo incluirá una reflexión personal del alumnado sobre los aspectos de la sostenibilidad que se abordan en el trabajo. Se pueden incluir tantas subsecciones como sean necesarias con la intención de explicar las competencias de sostenibilidad adquiridas durante el alumnado y aplicadas al Trabajo de Fin de Grado.

Más información en el documento de la CRUE https://www.crue.org/wp-content/uploads/2020/02/Directrices_Sostenibilidad_Crue2012.pdf.

Este anexo tendrá una extensión comprendida entre 600 y 800 palabras.

Bibliografía
