UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Impact Factor Oracle
Documentación Técnica

Presentado por Gadea Lucas Pérez
en Universidad de Burgos — 10 de mayo de 2023
Tutores: Virginia Ahedo y Álvar Arnaiz

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

La planificación temporal es esencial para el éxito de cualquier proyecto, especialmente en el desarrollo de *software*. En esta sección se detallará cómo se ha llevado a cabo el cronograma siguiendo una metodología ágil tipo Scrum, mediante el uso de *sprints*. Se describirán los pasos necesarios para planificar y gestionar de manera eficiente el tiempo y los recursos disponibles, asegurando así el cumplimiento de los objetivos del proyecto en el plazo establecido.

A.2. Metodología

La metodología Scrum es un marco de trabajo ágil que se utiliza para la gestión de proyectos. Se basa en la colaboración entre el equipo de desarrollo y el cliente, así como en la entrega continua de entregables funcionales en ciclos cortos de tiempo conocidos como *sprints*. Scrum se centra en la flexibilidad y la adaptabilidad, permitiendo más fácilmente la adaptación y los cambios de manera rápida y efectiva ante situaciones inesperadas.

En el contexto del presente trabajo, el uso de la metodología Scrum será beneficioso puesto que nos permitirá tener un enfoque más dinámico y flexible, lo que nos permitirá adaptarnos a las necesidades del proyecto a medida que avanzamos. Además, Scrum nos proporcionará una mayor transparencia y comunicación, lo que nos permitirá tomar decisiones informadas y asegurar que el proyecto se entregue a tiempo y con éxito.

Por otro lado, se ha complementado la metodología Scrum con el modelo Canvas[1]. Esta combinación nos permitirá visualizar y organizar de manera efectiva las tareas, los hitos y los recursos disponibles, facilitando así la gestión y el seguimiento del proyecto.

A.3. Herramienta

ZenHub es una herramienta de gestión de proyectos que se utiliza en muchos entornos empresariales para mejorar la eficiencia en el seguimiento y administración de tareas. Con ZenHub, es posible planificar y visualizar fácilmente las actividades de un proyecto, monitorear el progreso de cada tarea y hacer un seguimiento de los plazos de entrega. También ofrece funciones útiles como integración con GitHub, herramientas de informes y análisis de datos, y seguimiento de errores y problemas.

A.4. Planificación temporal

El presente proyecto comienza en septiembre y se extiende hasta junio. Durante este período de tiempo, es importante asegurarnos de que todas las tareas y hitos estén claramente definidos.

Para lograr esto, se han realizado reuniones periódicas para discutir el progreso del proyecto y asegurarnos de que estamos en el camino correcto. Además, hemos implementado *sprints* regulares para asegurarnos de que estamos avanzando de manera constante y cumpliendo con nuestras metas a tiempo.

Con esta planificación temporal sólida, estamos seguros de que podremos completar el proyecto a tiempo y cumplir con los objetivos establecidos. Sin embargo, es importante ser flexibles y estar preparados para hacer ajustes según sea necesario a medida que avanzamos en el proyecto, puesto que se realiza al mismo tiempo que trascurre el curso académico.

[1]El modelo Canvas se refiere comúnmente a un lienzo o plantilla visual utilizada para estructurar y desarrollar ideas de negocio.

En las siguientes tablas se recogen los distintos *sprints* junto con su duración, objetivos y tareas. Además, se adjuntará el gráfico *burndown*[2] de cada uno de los *sprints*.

Sprt.	Duración	Objetivos	Tareas
1	19/09/2022 03/10/2022	Comenzar el desarrollo del proyecto	▪ Elegir un modelo de referencias bibliográficas. ▪ Definición de conceptos.

Tabla A.1: Sprint 1

Figura A.1: Sprint 1: Burndown chart

Sprt.	Duración	Objetivos	Tareas

[2]Gráfico burndown: Se trata de una herramienta comúnmente utilizada en la gestión de proyectos ágiles para realizar un seguimiento de un *sprint*. Este gráfico muestra la cantidad de trabajo que queda por hacer en relación con el tiempo disponible para hacerlo. A medida que se completa el trabajo, la línea del gráfico debería disminuir hasta alcanzar cero en el momento en que finaliza el *sprint*.

Sprt.	Duración	Objetivos	Tareas
2	03/10/2022 17/10/2022	Tareas de investigación y documentación	▪ Búsqueda de antecedentes. ▪ Familiarizarse con la memoria en LaTeX.

Tabla A.2: Sprint 2

Figura A.2: Sprint 2: Burndown chart

Sprt.	Duración	Objetivos	Tareas
3	17/10/2022 14/11/2022	Investigación y creación de un prototipo inicial	▪ Búsqueda de información sobre MIAR. ▪ Documentación sobre el JCR y el índice de impacto. ▪ Prototipo que permita la búsqueda de artículos en GS. ▪ Extracción de datos: ¿qué datos se pueden extraer? ▪ Reflexión sobre el diseño de la BBDD. ▪ Prueba del prototipo.

Tabla A.3: Sprint 3

Figura A.3: Sprint 3: Burndown chart

Sprt.	Duración	Objetivos	Tareas
4	14/11/2022 28/11/2022	Base de datos y prototipo	▪ Creación de la base de datos en MariaDB. ▪ Prueba de la BBDD. ▪ Mejoras del prototipo.

Tabla A.4: Sprint 4

Figura A.4: Sprint 4: Burndown chart

NOTA

Como se puede observar en el gráfico *burndown* anterior, las tareas no fueron cerradas a tiempo. Sin embargo, es preciso aclarar que se trata de un error en la configuración de las tareas. Si bien todas las tareas fueron completadas a tiempo, fue preciso borrarlas en ZenHub y volverlas a crear más adelante cuando el *sprint* ya había terminado debido a que estaban inicialmente mal configuradas.

Sprt.	Duración	Objetivos	Tareas
5	28/11/2022 12/12/2022	Base de datos y prototipo	▪ Se muda la BBDD a Postgres. ▪ Pruebas e investigación para extraer el DOI.

Tabla A.5: Sprint 5

Figura A.5: Sprint 5: Burndown chart

Sprt.	Duración	Objetivos	Tareas
6	09/01/2023 23/01/2023	Documentación y obtención del DOI	▪ Obtención del DOI a través de Crossref. ▪ Avance de la memoria y los anexos. ▪ Mejora de la BBDD.

Tabla A.6: Sprint 6

Figura A.6: Sprint 6: Burndown chart

Sprt.	Duración	Objetivos	Tareas
7	06/02/2023 20/02/2023	Preparación para la conexión remota a los servidores de la universidad.	▪ Conectarse a la VPN de la universidad. ▪ Crear un servicio virtualizado en Miniconda. ▪ Mejorar la BBDD.

Tabla A.7: Sprint 7

Figura A.7: Sprint 7: Burndown chart

Sprt.	Duración	Objetivos	Tareas
8	20/02/2023 06/03/2023	Conexión a los servidores y mejora de los prototipos. Avances en la API y documentación.	▪ Mejorar el prototipo. ▪ Prototipo con varios hilos de ejecución (usar servidores de la universidad). ▪ Nuevo prototipo con Selenium. ▪ Sistema inteligente de requests. ▪ Revisión de PoP. ▪ API: Autentificación y back-end.

Tabla A.8: Sprint 8

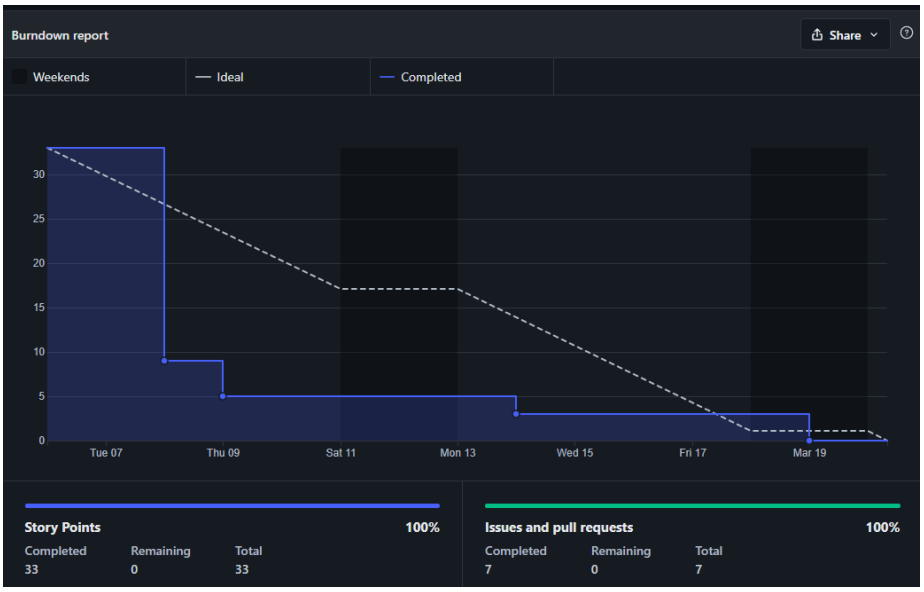

Figura A.8: Sprint 8: Burndown chart

Sprt.	Duración	Objetivos	Tareas
9	06/03/2023 20/03/2023	Finalización del prototipo de extracción de Crossref.	▪ Plan de prueba para Crossref. ▪ Prototipo de Crossref. ▪ Comprobar límite de llamadas a Crossref. ▪ Crear entorno virtualizado. ▪ Fichero requirements. ▪ Licencia Crossref para Cited-by. ▪ Cálculo del IF.

Tabla A.9: Sprint 9

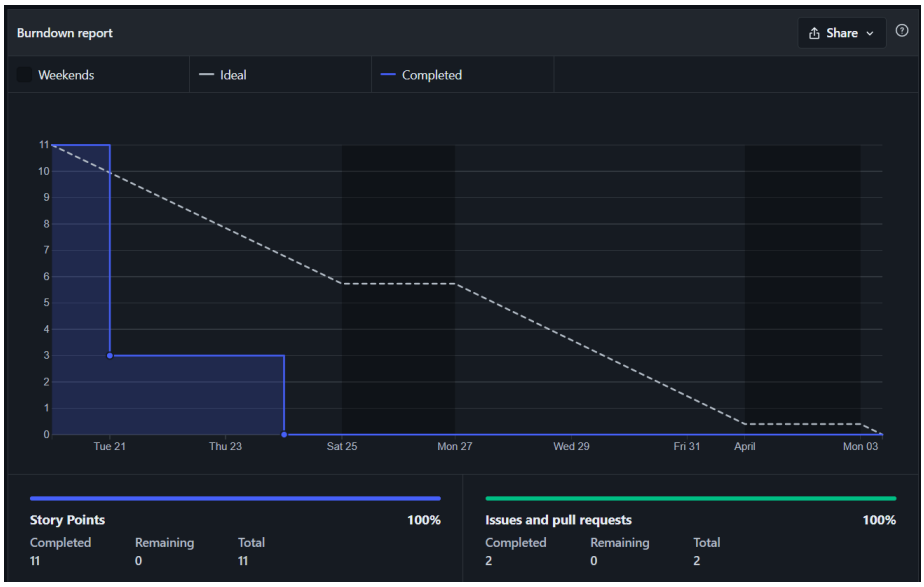

Figura A.9: Sprint 9: Burndown chart

Sprt.	Duración	Objetivos	Tareas
10	20/03/2023 03/04/2023	Procesamiento de los datos.	▪ Normalizar listados JCR. ▪ Comparar JCR obtenidos con los valores reales.

Tabla A.10: Sprint 10

Finalmente, se incluirá un gráfico acumulativo[3] que muestra la cantidad total de trabajo realizado de forma más generalizada.

[3]Gráfico acumulativo: En este gráfico, el eje vertical representa la cantidad total de trabajo completado, mientras que el eje horizontal representa el tiempo transcurrido durante el proyecto. A medida que se completa el trabajo, la línea del gráfico se eleva, lo que indica el progreso general del proyecto

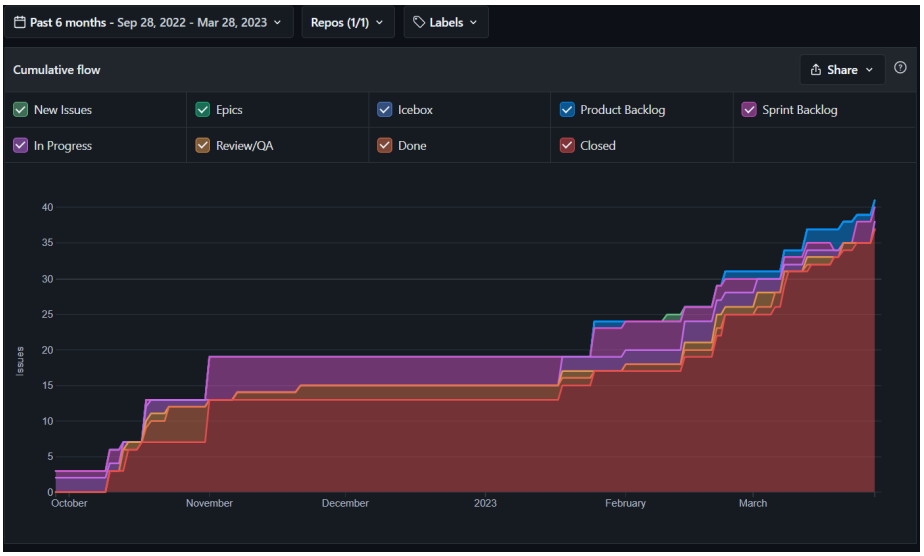

Figura A.10: Gráfico acumulativo de todo el proyecto

A.5. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

B.2. Objetivos generales

B.3. Catálogo de requisitos

B.4. Especificación de requisitos

Apéndice C

Especificación de diseño

C.1. Introducción

C.2. Diseño de datos

C.3. Diseño procedimental

C.4. Diseño arquitectónico

C.5. API

Se pretende desarrollar una API REST sencilla que utilizará una arquitectura cliente-servidor de dos capas.

La funcionalidad principal de la API será predecir el índice de impacto de una lista de revistas elegida por el usuario. Además, se proporcionarán gráficos e información adicional para ayudar a los usuarios a entender mejor los resultados. Por otro lado, se podrán distinguir dos perfiles de usuario: el usuario normal y el administrador. El administrador tendrá permisos para "reentrenar" la red que predice los índices y gestionar al resto de usuarios.

El *backend* se programará en Python y el *frontend* con HTML, CSS y JavaScript. Para ello, se utilizará Flask, que se trata de un *framework* de Python para el desarrollo de aplicaciones web.

Para la conexión con la base de datos, se hará uso de *psycopg2*, que es un módulo de Python que proporciona una interfaz para conectarse y

interactuar con bases de datos PostgreSQL. Es una de las librerías más populares y ampliamente utilizadas para trabajar con PostgreSQL en Python. Además, es compatible con la mayoría de las versiones de Python y es muy fácil de utilizar.

Por otro lado, para la autenticación, se han evaluado varias opciones que ofrece Flask, a saber: Flask-Login, Flask-Security y Flask-JWT. Finalmente, se ha elegido Flask-Login debido a su sencillez y facilidad de uso. Esta opción permite al usuario iniciar sesión con un nombre de usuario y una contraseña y almacena la información de la sesión en las *cookies* del navegador. Este paquete incluye la protección de rutas con decoradores.

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

D.3. Manual del programador

Esta sección está destinada a proporcionar información detallada sobre cómo utilizar el programa desarrollado en el proyecto. Aquí se describen las funciones y características clave de dicho software, así como los detalles sobre la configuración y la integración del programa con otros sistemas.

Base de datos

Para aquellos usuarios de sistemas operativos Windows, es importante tener en cuenta un detalle en el momento de obtener información de la base de datos en PostgresSQL. Cuando se recolectan los datos en formato CSV, es necesario cambiar la ruta de generación de estos archivos a la carpeta «Public» [1]. Esto se debe a que el usuario por defecto que genera PostgresSQL (usuario «postgres»), no tiene los permisos suficientes para leer e importar datos de los CSV en cuestión ya que, para ello, se utiliza el comando *copy* (que solo puede ser ejecutado por «Superusers»)[1]. Sin embargo, la carpeta

[1]Directorio que permite compartir archivos entre distintos usuarios en un mismo sistema Windows. Microsoft ha mantenido este directorio desde la versión de WindowsXP.

«Public» cuenta con los permisos necesarios para que el usuario que crea Postgres pueda acceder y leer los datos. Por lo tanto, es importante seguir este paso para garantizar que se pueda obtener correctamente la información recogida previamente.

D.4. Compilación, instalación y ejecución del proyecto

D.5. Pruebas del sistema

En el desarrollo de *software*, es esencial contar con una metodología adecuada para garantizar la calidad del producto final. En este sentido, el Test Driven Development (TDD) se ha convertido en una de las metodologías más populares en la actualidad. TDD consiste en escribir las pruebas antes de comenzar a desarrollar el código, lo que permite detectar errores de manera temprana y asegurar que el *software* cumpla con los requisitos establecidos. En este apartado, presentaremos los proyectos de pruebas que hemos llevado a cabo utilizando TDD como metodología principal.

Esta metodología se divide en tres fases principales: la fase roja, la fase verde y la fase morada.

- En la fase roja, se escriben las pruebas unitarias para el código que se desea implementar, las cuales deben fallar inicialmente (puesto que aún no existe el código a probar).
- En la fase verde, se escribe el código mínimo necesario para que las pruebas pasen satisfactoriamente.
- Finalmente, en la fase morada, se realiza la refactorización del código escrito anteriormente para mejorar su calidad y mantenerlo limpio y legible.

Este enfoque iterativo permite a los desarrolladores garantizar la calidad y funcionalidad del software en cada paso del proceso de desarrollo y ayuda a reducir los errores y *bugs* que puedan surgir en el proceso.

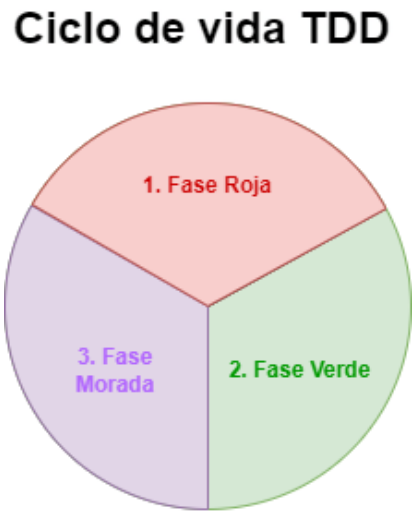

Figura D.1: Fases del la metodología TDD

Prototipo de Crossref

Este es el prototipo final para la extracción de los datos bibliográficos. A continuación, se documentará la información más relevante del proyecto de pruebas llevado a cabo.

Diseño y arquitectura

Puesto que el *software* a probar está compuesto de un solo *script* de Python, se ha generado un único módulo de pruebas llamado `crossref_test.py`.

Fase Roja

A lo largo de esta fase se definen las pruebas necesarias para crear un código que extraiga correctamente los datos de Crossref. Los casos de prueba diseñados son los siguientes:

ID	Nombre	Descipción	Salida esperada

ID	Nombre	Descipción	Salida esperada
C01	Revista existente	Método test_journal_exist. Se trata de extraer una revista que previamente se conoce que existe.	True
C02	Revista inexistente	Método test_journal_exist. Se trata de extraer una revista que previamente se conoce que no existe.	False
C03	Cargar datos de entrada	Método test_load_data. Se comprueba que se cargan bien los datos del CSV de entrada.	Carga correcta
C04	Formateado I	Método test_formatData. Se comprueba que se eliminan bien las etiquetas HTML y los espacios.	Cadena correcta
C05	Formateado II	Método test_formatData. Se comprueba que se eliminan bien las etiquetas «href».	Cadena correcta
C06	Formateado III	Método test_formatData. Se comprueba que una cadena correcta no cambia.	Cadena sin modificaciones
C07	Formateado IV	Método test_formatData. Se comprueba que no hay campos vacíos (rellenar con «n.m.»).	Cadena sin valores vacíos
C08	Formato de salida I	Método test_getArticles_returns_list_of_dicts. Se comprueba que se obtiene una lista de artículos.	Lista de diccionarios
C09	Formato de salida II	Método test_getArticles_returns_valid_keys. Se comprueba que las claves del diccionario son correctas.	Claves correctas
C10	Número de resultados	Método test_getArticles_correct_ article_count. Se comprueba que se obtiene el número correcto de artículos que aparecen en Crossref.	Número correcto

ID	Nombre	Descipción	Salida esperada
C11	Excepción de ISSN	Método test_getArticles_invalid_issn_raises_error. Se intenta buscar un ISSN incorrecto.	False
C12	Excepción de año I	Método test_getArticles_invalid_year. Se introducen los años de búsqueda en el orden incorrecto.	False
C13	Excepción de año II	Método test_getArticles_invalid_year. Se introducen años inválidos.	False
C14	Ejecución completa I	Método test_main. Pre-requisito: CSV de entrada de datos. Se comprueba que se genera una gráfica al finalizar el proceso. Post-requisito: Imagen con la gráfica generado.	True
C15	Ejecución completa II	Método test_main. Precondición: CSV de entrada de datos. Se comprueba que se generan los ficheros CSVs de resultados al finalizar el proceso. Post-requisito: Ficheros CSV generados.	True
C16	Ejecución completa III	Método test_main. Precondición: CSV de entrada de datos. Se comprueba que se genera el fichero de log al finalizar el proceso. Postcondición: Ficheros de log generados.	True
C17	Ejecución completa IV	Método test_main. Precondición: CSV de entrada de datos y fichero de log generado correctamente. Se comprueba que la información de log es correcta.	True

Tabla D.1: Casos de prueba para el prototipo de Crossref

Fase Verde

Tras la creación del código, finalmente se consigue una versión que pasa todos los test programados en la fase anterior, como se puede observar en la siguiente ilustración:

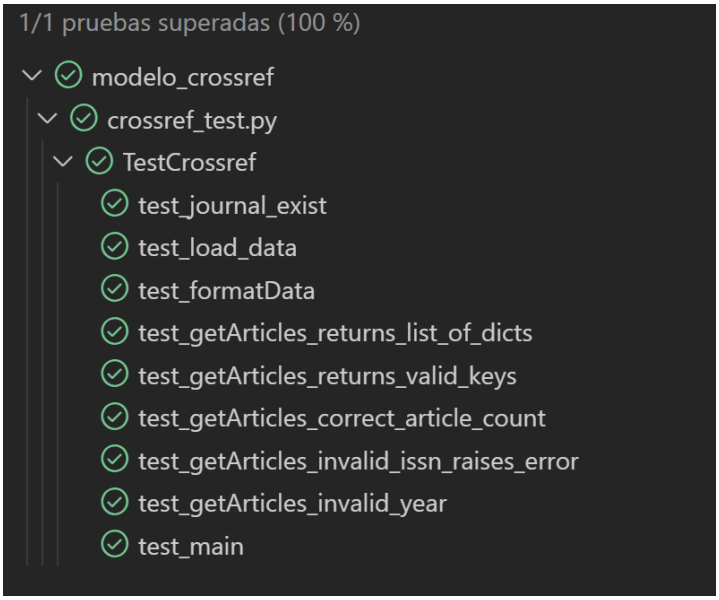

Figura D.2: Pruebas pasadas exitosamente

Fase Morada

En esta fase se han realizado acciones de refactorización. A saber:

- Documentación de código: Se han incluido las cabeceras y comentarios más importantes en el código para facilitar su comprensión y claridad. Asimismo, se han eliminado comentarios redundantes o innecesarios.
- Ordenación de *imports*: Se han ordenado los *imports* del código de forma que se identifique fácilmente la naturaleza de los paquetes y módulos empleados. De esta forma, se agrupan al inicio los módulos integrados de Python, posteriormente los paquetes de terceros y, finalmente, módulos o paquetes locales.
- Mensajes de salida I: En lugar de imprimir mensajes de salida en la consola, (ant. `print("[ERROR] Se ha producido un error")`), ahora se utilizan registros de *logging* para registrar los eventos de la función

(por ejemplo, `logging.error('Se ha producido un error')`). Los registros son una forma más eficiente y flexible de registrar información que imprimir en la consola.

- Mensajes de salida II: Utiliza f-strings en lugar de concatenar cadenas para mejorar el rendimiento y la legibilidad del código. Ejemplo: `logging.error('Se ha producido un error'+ codigo_error)` por `logging.error(f'Se ha producido un error {codigo_error}')`.
- Acceso a diccionarios I: En lugar de especificar los parámetros de búsqueda utilizando una cadena de consulta, se puede usar un diccionario y pasarlos como argumentos de palabras clave al método `requests.get()`. Esto puede hacer que el código sea más legible.
- Acceso a diccionarios II: En lugar de crear un diccionario vacío .ªuthorz luego verificar si existe una lista de autores, se puede usar el método `get()` directamente en la lista de autores para obtener el primer elemento de la lista.
- Lista de comprensión: En lugar de crear un diccionario temporal publicacion para cada artículo, se puede usar una lista de comprensión para crear una lista de diccionarios de artículos. Esto puede hacer que el código sea más conciso.

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

Entorno virtualizado

Se trata de un entorno aislado que te permite instalar y gestionar de forma independiente las bibliotecas y paquetes de Python que se necesitan para el proyecto, sin afectar a otros proyectos o al sistema operativo. Esto nos permite evitar problemas de dependencias y compatibilidad entre diferentes versiones de bibliotecas, además de facilitar la gestión y el mantenimiento del proyectos.

Miniconda, por su parte, es una distribución de Python que incluye un gestor de paquetes llamado Conda, que te permite instalar y gestionar de forma sencilla bibliotecas y paquetes de Python.

Así pues, se procede a crear un entorno virtualizado de Python utilizando Miniconda. El entorno en cuestión se encuentra empaquetado en la carpeta comprimida `environment.zip`. Para poder utilizar este entorno, bastará con realizar los siguientes pasos:

1. Instalar MiniConda en el dispositivo en el que se quiera ejecutar el prototipo.
2. Crear el entorno usando el siguiente comando en la consola de MiniConda: `conda env create -f environment.yml`. Es preciso estar en el mismo directorio que el fichero `environment.yml` o especificar la ruta completa.
3. A continuación, se debe activar el entorno con el siguiente comando: `conda activate crossref_entorno`.
4. Ejecutar el prototipo de Crossref.
5. Una vez finalizado, para desactivar el entorno, bastará con ejecutar el siguiente comando: `conda deactivate`.

Bibliografía

[1] Jorge Domínguez Chávez. *CLIENTE PSQL DE POSTGRESQL*. 04 2020.