RASTREADOR DE VIVIENDAS ADAPTADAS

VNiVERSiDAD D SALAMANCA

Trabajo de Fin de Grado
INGENIERÍA INFORMÁTICA 2023

Autor
Adrián Torre Salinero

Tutores
Sara Rodríguez González
Guillermo Hernández González

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

Facvltad D Ciencias
VNiVERSiDAD D SALAMANCA

Certificado de los tutores TFG

Dña. Sara Rodríguez González, profesora del Departamento de Informática y Automática de la Universidad de Salamanca, y D. Guillermo Hernández González, profesor del Departamento de Informática y Automática de la Universidad de Salamanca

HACEN CONSTAR:

Que el trabajo titulado"Rastreador de viviendas adaptadas", que se presenta, ha sido realizado por Adrián Torre Salinero, con DNI 70922271T y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Ingeniería Informática en esta Universidad.

Salamanca, de julio de 2023

Fecha: 2023.07.03 22:47:37 +02'00'

Firmado por HERNANDEZ GONZALEZ GUILLERMO - ***1124** el día 03/07/2023 con un certificado emitido por AC FNMT Usuarios

Fdo.: Sara Rodríguez González

Fdo.: Guillermo Hernández González

Resumen

En este proyecto se ha desarrollado un buscador web de viviendas en el que se muestran viviendas adaptadas. Para ello, el sistema es capaz de buscar la información de las viviendas en una web, concretamente en todopisos y, filtrar las viviendas por aquellas que sean adaptadas o accesibles.

Las viviendas accesibles son aquellas que no presentan limitaciones para la movilidad de las personas que las habitan. Para saber si una vivienda es accesible, el sistema debe de ser capaz, mediante el procesamiento de lenguaje natural, de diferenciar las descripciones de las viviendas accesibles de las que no lo son. Concretamente se ha utilizado la técnica de *Latent Dirichlet Allocation* (LDA), que permite la autogeneración de tópicos dado un conjunto de observaciones. Uno de los tópicos generados es el que se utiliza para definir las viviendas accesibles.

La motivación principal de este proyecto surge de la petición del CRMF del imserso, representando una necesidad real de este centro. Las personas que forman parte de este centro trasladaron la necesidad que tenían de un buscador de estas características.

Para el desarrollo del proyecto se utiliza la metodología Scrum debidamente adaptada a las características de un proyecto individual, ya que Scrum fue concebida para proyectos grupales. Se ha decidido utilizar esta metodología debido al corto tiempo de desarrollo y el desconocimiento de las tecnologías usadas. Los *sprints* permiten el aprendizaje e implementación de herramientas y funcionalidades para añadirlas a un prototipo funcional incremental.

La mayoría del proyecto está desarrollado en Python, apoyándose en HTML, CSS y JavaScript para las interfaces de usuario y en un servidor de base de datos relacional MariaDB para el almacenamiento de los datos.

El sistema desarrollado tiene una interfaz sencilla y fácil de entender. Para asegurarse de que esto es así, se han realizado pruebas a diferentes usuarios, además de un diseño de interfaz el cual está basado en las interfaces de los portales web más conocidos.

Aunque el sistema implementa un sistema de usuarios, el cual da acceso a ciertas funcionalidades para aquellos usuarios que decidan registrarse en el sistema como tener una lista de viviendas favoritas, la búsqueda de viviendas dentro de la web está permitida sin la necesidad de registrarse en el sistema.

Los usuarios registrados en el sistema tienen acceso a funcionalidades que no tienen los usuarios no registrados, como la posibilidad de añadir viviendas a la lista de favoritas y acceder a la misma.

Para recoger las viviendas de la web seleccionada se utiliza un sistema scraper, implementado con la herramienta Scrapy. La información recogida por el proceso scraper y pasada por el filtro generado por el modelo LDA, debe de ser confirmada por un usuario administrador, para evitar mostrar viviendas en la página que no sean accesibles. Además, los usuarios administradores pueden editar la información de las viviendas para solventar posibles fallos en la extracción de los mismos, ya sea caracteres que no han sido reconocidos, descripciones incompletas...

El sistema está integrado con la web original, permitiendo visitar las webs específicas de cada vivienda directamente desde la web creada, ya sea para acceder a otros datos de la vivienda, contactar con el vendedor original...

Palabras Clave

Buscar de viviendas, viviendas adaptadas, Flask, Scraper, LDA, Docker, MariaDB, Python, HTML, Scrum

Abstract

In this project, a housing web search engine has been developed in which adapted housing is shown. To do this, the system must search for information on the homes on a website, specifically at https://www.todopisos.es, and filter the homes by those that are adapted or accessible.

Accessible homes are those that do not present limitations for the mobility of the people who inhabit them. To find out if a home is accessible, the system must be capable, through natural language processing, of differentiating the descriptions of accessible homes from those that are not. Specifically, LDA has been used, which allows the self-generation of topics given a set of observations. One of the topics generated is the one used to define affordable housing.

The main motivation of this project arises from the request of the CRMF of the imserso, representing a real need of this center. The people who are part of this center conveyed the need they had for a search engine of these characteristics.

For the development of the project, the Scrum methodology is used, duly adapted to the characteristics of an individual project, since Scrum was conceived for group projects. It has been decided to use this methodology due to the short development time and the lack of knowledge of the technologies used. The sprints allow the learning and implementation of tools and functionalities to add them to an incremental functional prototype.

The majority of the project is developed in Python, relying on HTML, CSS and JavaScript for user interfaces and a relational database server MariaDB for data storage.

The developed system has a simple and easy to understand interface. To make sure that this is the case, tests have been carried out on different users, in addition to an interface design which is based on the interfaces of the most popular web portals.

Although the system implements a user system, which gives access to certain features for those users who decide to register in the system, such as having a list of favorite homes, the search for homes within the web is allowed without the need to register in the system.

Users registered in the system have access to functions that non-registered users do not have, such as the ability to add homes to the favorites list and access it.

To collect the homes from the selected web, a scraper system is used, implemented with the Scrapy tool. The information collected by the scraper process and passed through the filter generated by the LDA model must be confirmed by an administrator user, to avoid showing homes on the page that are not accessible. In addition, administrator users can edit the information of the dwellings to solve possible errors in their extraction, whether it be characters that have not been recognized, incomplete descriptions...

The system is integrated with the original website, allowing you to visit the specific websites of each home directly from our website, either to access other data about the home, contact the original seller...

Keywords

Search for housing, adapted housing, Flask, Scraper, LDA , Docker , MariaDB , Python , HTML, Scrum

Tabla de contenido

Tabla de tablas

Tabla de imágenes

1. Introducción

En el presente documento se expone la información necesaria para la memoria del Trabajo de Fin de Grado titulado «Buscador de viviendas adaptadas», desarrollado por el alumno Adrián Torre Salinero, cuyo resultado es un software que integra viviendas accesibles detectadas en la web todopisos (1).

Este trabajo de final de grado se ha llevado a cabo en colaboración con el CRMF (Centro de Recuperación de personas con discapacidad Física) (2) del Imserso. Se mantuvo una reunión con ellos a finales de noviembre de 2022 en la que expresaron la necesidad del proyecto y dieron unas indicaciones orientativas de lo que necesitaban, pero con libertad a la hora de ampliar funcionalidades.

Un sistema de búsqueda de viviendas (3) es un sistema que dispone de información de viviendas y permite consultar y filtrar esta información. Normalmente, este tipo de sistemas son portales web, en los cuales se selecciona compra o alquiler, ciudad y un rango de precios. La aplicación muestra las viviendas de las cuales tiene información y se corresponde con los filtros introducidos por el usuario.

Muchos otros portales ofrecen otro tipo de funcionalidades como cálculo de hipotecas o servicios orientados a inmobiliarias, pero la funcionalidad básica es la de mostrar la información de las viviendas. Normalmente esta información la obtienen a través de los propios usuarios, ya sean inmobiliarias o particulares, que deciden anunciar su vivienda en el portal.

Muchas de estas aplicaciones tienen un sistema de usuarios que proporciona funcionalidades extras, como la propia funcionalidad de publicar un anuncio de vivienda o generar una lista con viviendas marcadas como favoritas. Además, las interfaces de la mayoría de portales de viviendas son muy parecidas y muestran información similar: precio de la vivienda, ciudad en la que se encuentra, descripción de la vivienda, fotos de la misma, etc.

Es importante para el proyecto tener como referencia los principales portales utilizados para buscar viviendas, como lo son fotocasa (4) o idealista (5).

Una vivienda adaptada o accesible se trata de una vivienda que no ofrece barreras arquitectónicas que dificulten la movilidad de las personas que la habitan (6). Este tipo de viviendas son especialmente importantes para las personas que padecen reducciones de movilidad, ya sea por enfermedades o por deterioro de las capacidades motrices.

Actualmente, existen gran cantidad de viviendas que presentan barreras físicas y que son totalmente inhabitables por este tipo de personas (7). Muchos de los aspectos que para personas sin este tipo de necesidades son banales, como un escalón en la entrada del portal, para las personas con estas necesidades son aspectos cruciales a la hora de elegir la vivienda en la que van a vivir.

Se entiende que las viviendas adaptadas o accesibles pueden tener diferentes grados de adaptación en función de la reducción de movilidad que padezca la persona.

La propuesta realizada por el CRMF que corresponde a este trabajo, viene precedida por una necesidad real a la que el centro se enfrenta. Los buscadores de viviendas más populares no ofrecen una opción para filtrar

las viviendas por aquellas que son adaptadas. Las personas que integran el CRMF encuentran muchas complicaciones a la hora de encontrar este tipo de viviendas en los buscadores.

Este proyecto viene a solucionar esta necesidad, mediante un sistema que permita encontrar viviendas accesibles en la red, almacenarlas y generar un buscador alrededor de ellas. Como es lógico, el número de viviendas disponibles que sean adaptadas va a ser mucho menor de las que no lo son, pero el buscador aspira a cumplir las necesidades del centro. Además se pretende darle un alcance de ámbito nacional al buscador, buscando viviendas en toda España.

La estructura del resto de esta memoria es la siguiente:

- Objetivos del trabajo de Fin de grado
- Conceptos teóricos
- Técnicas y herramientas
- Aspectos relevantes del desarrollo
- Conclusiones y líneas de trabajo a futuro
- Bibliografía

2. Objetivos del sistema

Antes de comenzar cualquier proyecto *software* es fundamental la definición de los objetivos del mismo. Dichos objetivos deben de ser generalistas, abarcando diversas opciones de escalabilidad y funcionalidad, pero cumpliendo estrictamente con las necesidades del cliente. En el caso concreto de este proyecto de final de grado, se ve muy bien reflejado, ya que, en la toma de requisitos con el cliente, dio bastante libertad, pero con un objetivo claro: ser capaz de crear un portal de viviendas accesibles.

Es por ello, que los objetivos del sistema desarrollado para este trabajo de fin de grado son bastante generalistas, dando pie a agregar funcionalidades, pero cumpliendo estrictamente con las necesidades del cliente.

Es importante recalcar que los requisitos no son solo peticiones expresas del cliente o la funcionalidad relacionada con los mismos, sino que son todos aquellos aspectos que se detectan en el proceso de desarrollo como requisitos para que el proyecto *software* funcione. Pueden ser extraídos de las peticiones del cliente o necesidades derivadas del proceso de diseño, herramientas utilizadas, modelo de despliegue...

El objetivo de este apartado es mostrar y explicar los objetivos del sistema. Primero, se explicarán los objetivos funcionales. que son aquellos objetivos que representan una funcionalidad de la aplicación. Después se explicarán los objetivos no funcionales, que vienen representados por requisitos que no representan una funcionalidad propia dentro de la aplicación.

2.1. Objetivos funcionales

Los objetivos funcionales son aquellos derivados directamente de las necesidades que plantean los clientes y cómo la aplicación puede satisfacerlas.

OBJ-01	**Gestión de usuarios**
Versión	1.0
Autor	Adrián Torre Salinero
Fuente	CRMF
Descripción	Permitir el alta y modificación de usuarios y/o administradores.
Subobjetivos	-
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Guardar los datos de los diferentes usuarios del sistema.

Tabla 1 - OBJ_01 - Gestión de usuarios

El primer objetivo del sistema y propio de casi cualquier aplicación de estas características es la gestión de usuarios. Este objetivo hace referencia a todas las funcionalidades que el sistema debe aportar relacionados con la información de los usuarios. A grandes rasgos, debe guardar y manejar la información de registro de

los usuarios, debe iniciar la sesión de los mismos con los datos de registro obtenidos y debe de diferenciar las diferentes funcionalidades que los diferentes tipos de usuarios tienen.

Varias de las decisiones relacionadas con este objetivo son críticas, ya que existen funcionalidades dentro de la aplicación a las que, si accedieran usuarios no deseados, podría cambiar el uso de la aplicación negativamente. Por ejemplo, si alguien que no es administrador, usuario el cual tiene acceso a funcionalidades especiales, pudiera editar la información de las viviendas, muchas de las funcionalidades asociadas a la vivienda podrían funcionar erróneamente, como por ejemplo en los filtros de ciudad y precio de las mismas.

Además, el sistema debe ser capaz de registrar nuevos usuarios administradores. Para ello, ofrece la funcionalidad de ascender a un usuario como administrador. Es una funcionalidad exclusiva de otros administradores y permite elegir a un usuario estándar y convertirlo en administrador.

Se habilita una opción para ver los datos del usuario, donde se mostrarán los datos incluidos en el registro excluyendo la contraseña. El sistema debe de ser capaz de proporcionarle un formulario de cambio de contraseña al usuario, donde introduciendo la contraseña antigua se le permita establecer una nueva contraseña.

OBJ-02	**Gestión de viviendas**
Versión	1.0
Autor	Adrián Torre Salinero
Fuente	CRMF
Descripción	Gestión de la información relativa a las viviendas de la base de datos.
Subobjetivos	-
Importancia	Vital
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Buscar viviendas , guardar la información relacionada con cada una de ellas y permitir su eliminación de la base de datos, así como la manipulación de datos por parte del usuario (filtrar viviendas por precio, zona, añadir a favoritos...)

Tabla 2 - OBJ_02 - Gestión de viviendas

Este segundo y último objetivo es el relacionado con la gestión de la información de las viviendas. Este objetivo es especialmente generalista, lo que permite ajustar el alcance y las funcionalidades de la aplicación en tiempo de desarrollo. Las funcionalidades relacionadas con este objetivo se encargarán de buscar la información de las viviendas en la web, filtrar la información de las viviendas obtenidas y guardar únicamente aquellas que el filtro aplicado considere como accesible y manejar la información de las viviendas.

Tanto la búsqueda de la información de las viviendas como el filtrado de las mismas deben realizarse de forma automática por el sistema. Son tareas que se el sistema realizará de forma periódica para ir ampliando la base de viviendas del sistema. Una vez ha filtrado las viviendas, las almacena como viviendas sin verificar y las guarda en la base de datos del sistema en la tabla correspondiente.

Debe diferenciar la información de las viviendas mostrada a los usuarios estándar que, a los usuarios administradores, ya que para los administradores debe ser posible modificar la información de las viviendas cuando acceda a ellas. Además, las viviendas encontradas, las cuales han pasado el filtro y el sistema las considera accesibles, deben de ser verificadas por los administradores antes de que se muestren al resto de usuarios. Antes de verificar las viviendas pueden editar también su información.

Los usuarios registrados pueden añadir cualquier vivienda que ya haya sido verificada a su lista de favoritos. En esta lista aparecerán las viviendas igual que si lo hicieran en una búsqueda normal. El sistema debe ser capaz de manejar de forma ágil la relación entre los usuarios y las viviendas, ya que se espera que en el funcionamiento normal de la aplicación un usuario añada y borre varias viviendas de su lista de favoritos.

2.2. Objetivos no funcionales

Los objetivos no funcionales son aquellos que no representan una funcionalidad propia de la aplicación, pero que son necesarios para el correcto funcionamiento de la misma. Suele tratarse de conceptos más abstractos que en los objetivos funcionales. Representan condiciones y restricciones del sistema.

NFR-0001	**Fiabilidad**
Versión	1.0
Autores	Adrián Torre Salinero
Fuentes	CRMF
Objetivos asociados	
Requisitos asociados	
Descripción	El sistema debe de funcionar correctamente el mayor tiempo posible.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	El servidor deberá de ser capaz de abastecer varias peticiones simultáneamente.

Tabla 3 - NFR-0001 - Fiabilidad

NFR-0002	Seguridad
Versión	1.0
Autores	Adrián Torre Salinero
Fuentes	CRMF
Objetivos asociados	
Requisitos asociados	
Descripción	El sistema debe de ser capaz de ofrecer un protocolo seguro para el intercambio de información con los clientes de la página. Además, la información sensible del usuario (contraseña) debe ser almacenada de forma segura.
Importancia	Alta
Urgencia	Baja
Estado	En desarrollo
Estabilidad	Alta
Comentarios	

Tabla 4 - NFR-0002 – Seguridad

NFR-0003	Facilidad de uso
Versión	1.0
Autores	Adrián Torre Salinero
Fuentes	CRMF
Objetivos asociados	
Requisitos asociados	
Descripción	Tanto las interfaces como las funcionalidades de la aplicación deben de ser fáciles de aprender y entender por parte de todos los usuarios.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	

Tabla 5 - NFR-0003 -Facilidad de uso

Es importante resaltar la importancia de la facilidad de uso. Al ser una aplicación destinada a personas con movilidad reducida, se tiene que tener en cuenta que el sistema tiene un público general, pero mayoritariamente gente mayor, ya que en personas de avanzada edad es más común tener algún tipo de problema de movilidad. Es por ello que las interfaces son simples y fáciles de comprender. Se han realizado pruebas con diferentes tipos de usuarios para comprobar este aspecto.

En cuanto a seguridad, el sistema principalmente protege los datos de acceso al sistema, encriptando la contraseña. Además, se han realizado pruebas de seguridad para que no sea posible acceder a funcionalidades prohibidas o a información sensible de los usuarios mediante la manipulación de la URL. Además, el sistema debe diferenciar las funcionalidades exclusivas de administradores, usuarios registrados y usuarios no registrados. El sistema de despliegue elegido facilita que la disponibilidad del sistema sea muy buena.

2.3. Objetivos personales

Para el proyecto se ha tenido que tener en cuenta la limitación temporal de la entrega del mismo, lo que ha influido sensiblemente en el coste total del proyecto. Se han tenido que realizar varios ajustes para determinar un coste temporal que se ajustara a la fecha estipulada.

Esto ha influido en la priorización de tareas. Personalmente, quería tener un proyecto que se pudiera diferenciar de otros proyectos similares. Esto implica priorizar y asignar más recursos a aquellas tareas que tengan como resultado una funcionalidad realmente diferencial en la aplicación. Específicamente, el proceso de filtrado de las viviendas para hacer que sean accesibles, al que se ha dedicado todo el tiempo necesario para perfeccionar su funcionamiento.

Personalmente, me interesaba mucho aprender tecnologías para el procesamiento de lenguaje natural y el proceso de crawling y scraping. Me interesa mucho el avance de la informática en el contexto de comprensión de actividades humanas. En el caso concreto del proyecto, se ha conseguido que el sistema sea capaz de entender las descripciones para saber si son accesibles o no. Es el primer paso del que espero que sean muchos en mi desarrollo personal en este ámbito.

Además, las viviendas adaptadas son una parte importante en mi vida, ya que familiares muy cercanos tienen una vivienda adaptada a sus limitaciones físicas. Conozco la necesidad de las personas en estas condiciones de encontrar una vivienda que les haga la vida más fácil.

La parte afectada por la asignación de prioridades ha sido las interfaces de usuario, las cuales cumplen con los requisitos del sistema, especialmente el requisito de facilidad de uso, pero son interfaces simples.

Cabe destacar que todos los objetivos descritos son fruto del objetivo primordial de este trabajo, que es cumplir con los requisitos de un trabajo de final de grado, tanto por cantidad como por calidad del trabajo.

3. Conceptos teóricos

En este apartado de la memoria se presenta una definición de los conceptos más importantes relacionados con el trabajo de fin de grado. Es importante comprender el significado de cada uno de los términos mostrados, ya que se utilizarán a lo largo de la memoria y son partes imprescindibles para la comprensión tanto de lo que realmente hace la aplicación como de la memoria en sí.

3.1. Web Scraping

El web scraping (8) es una técnica utilizada para extraer información a través de páginas web. Es un proceso *software* que se lleva a cabo para extraer información determinada de una url, incluso para aquellas que están diseñadas para el uso humano, es decir, aquellas que no están en un formato específicamente diseñado para su procesamiento automatizado.

Es una herramienta de gran ayuda en el proyecto, ya que permite obtener información de gran cantidad de viviendas con un coste asociado bajo. Para ello se debe proporcionar al *software* una lista de URL correspondientes a las viviendas. Para buscar las viviendas, inicialmente se eligieron los buscadores de viviendas más utilizados, como *fotocasa* e *idealista*, pero prohíben en sus términos y condiciones el uso de este tipo de tecnologías. Por ejemplo, en el aviso legal de idealista:

<<Puedes navegar por la Web y Apps, registrarte para guardar búsquedas y favoritos, contactar con anunciantes, publicar inmuebles en venta o alquiler, así como contratar otros servicios adicionales. No puedes hacerles daño a terceros ni a idealista, hacer actos contrarios a la Ley, **usar mecanismos automáticos para copiar o extraer nuestro contenido**, hacer contactos fraudulentos ni utilizar las claves de acceso de otros sin su permiso.>>

Además, implementaban un sistema anti-crawling, que no permitía un acceso recursivo a sus URLs. Idealista ofrece una API para recoger información de viviendas, pero no satisface las necesidades de la aplicación ya que las viviendas diarias que te permite consultar son muy reducidas.

Por estas razones, se decidió utilizar una página menos conocida, pero que cumplía con las necesidades de la aplicación y permitía el uso de scraping en su web. La web elegida ha sido todopisos (1). Se ha elegido esta web por encima de otras debido a varios factores:

- Tamaño de la página. El portal seleccionado, pese no ser uno de los más conocidos, es bastante grande. Dispone de una gran cantidad de viviendas que proporciona una base fiable con la que trabajar. Además, tiene ciudades de toda España, lo que permite que la aplicación pueda tener un alcance considerable. Teniendo en cuenta el número total de viviendas que se han añadido a la página desde su creación (unas 7.400.000 viviendas) y el año de creación del portal (2005) se puede hacer un cálculo aproximado sobre la cantidad de viviendas mensuales que se cuelgan en la página, unas 34.200 mensuales. Estos datos concuerdan con los observados en el periodo de desarrollo del proyecto

- Estructura de la URL. Para poder especificar al *software* las URLs que debe visitar, hay que construirlas primero. En este aspecto, la estructura utilizada por la página (https://www.todopisos.es/inmueble/n donde n es un identificador único de cada anuncio colgado en la página) permite que la construcción de los sitios web a visitar sea muy sencillo. Además, es fácil saber cuáles han sido las viviendas que ya se han visitado almacenando la última visitada e incrementando ese número.
- Estructura Html de la página. Los datos más importantes de la página web vienen etiquetados en el código html de la web con una clase única, lo que facilita mucho la extracción de los datos de la web.

3.2. Procesamiento de Lenguaje Natural (NLP)

El procesamiento de lenguaje natural (9) (NLP, Natural Language Processing) es una tecnología de *machine learning* (10) que brinda a las computadoras la capacidad de interpretar, manipular y comprender el lenguaje humano. Es una parte importante del proyecto, ya que el sistema deberá de ser capaz de distinguir las viviendas adaptadas o accesibles de las que no lo son. Esto lo va a conseguir a través de las descripciones de las viviendas.

Es importante que este proceso se haga de manera exitosa, ya que la funcionalidad de la aplicación se basa en el filtrado realizado por el sistema tras procesar la información obtenida por el proceso de web scraping. Un mal funcionamiento de este procesamiento afecta de forma muy negativa al funcionamiento de la aplicación, ya sea bloqueando viviendas que sean accesibles o dejando pasar muchas viviendas que no lo sean. Aunque se necesitará de una confirmación posterior por parte de un administrador, es importante que la cantidad de viviendas no accesibles que el sistema detecta como que sí lo son sea lo más reducida posible.

Aunque se han barajado varias opciones a lo largo del desarrollo del proyecto para implementar un sistema de procesamiento de lenguaje que cumpla las necesidades del cliente, finalmente se ha decidido por utilizar LDA, el cual es explicado en el apartado 3.6. LDA.

La técnica de filtrado planteada en un inicio fue la búsqueda de elementos clave en las descripciones de las viviendas. Aunque esta herramienta es utilizada finalmente para reducir el número de viviendas erróneas del filtrado, es un proceso demasiado costoso para implementarlo en un volumen alto de trabajo. LDA es un modelo matemático que permite que el proceso de filtrado de grandes cantidades de información se realice de una manera más eficiente.

También se planteó el uso de otras herramientas, como NLTK, pero finalmente esta herramienta solo ha tenido un papel de apoyo a la tecnología elegida.

3.3. Contenedores de Software

Los contenedores de *software* (11) son una herramienta de despliegue que permite la ejecución de aplicaciones de forma aislada y portátil. Proporcionan un entorno en el que se pueden empaquetar aplicaciones y sus dependencias. Esto facilita en gran medida su implementación y ejecución en diferentes entornos.

Una de las principales características por las que el proyecto tiene este modelo de despliegue es la portabilidad de los contenedores, lo que permite que se puedan ejecutar en diferentes sistemas operativos de una forma muy rápida. Concretamente, se busca que sea portable a todos los sistemas UNIX, ya que se requiere de la herramienta cron para el correcto funcionamiento de la aplicación.
Otro de los motivos para la elección de este modelo, es la rapidez en la implementación del mismo. La herramienta elegida, Docker, es fácil de configurar y de exportar.

3.4. Bases de datos relacionales

Una base de datos relacional (12) es un tipo de base de datos que almacena y proporciona acceso a datos relacionados entre sí. Las bases de datos relacionales se basan en el modelo relacional, que es una forma intuitiva y directa de representar datos en tablas. En una base de datos relacional, las filas de las tablas son registros, con un ID único llamado clave. Las columnas representan los atributos del registro.
En el sistema se ha decidido utilizar este tipo de base de datos ya que la información que se va a almacenar tiene siempre la misma estructura, ya que el sistema scraper busca la misma información de las viviendas y todos los usuarios tienen los mismos datos de registro. Además, al almacenar datos relacionados, la gestión de los mismos se realiza de una forma segura, con unas normas y de modo uniforme.
Este sistema de base de datos es deficiente en la gestión de datos gráficos o multimedia, pero esto no representa una desventaja en el sistema ya que no es necesario almacenar ese tipo de datos. Además, la fragmentación de tablas, que suele ser otro problema de las bases de datos relacionales al no poder emplear entidades subordinadas, no es grande en esta aplicación, ya que el sistema de claves externa ayuda a solventar la posible fragmentación. Esto, unido al reducido número de tablas que maneja la aplicación, propicia que la velocidad de las consultas no se vea afectada.

3.5. API-REST

Para explicar este concepto es necesario dividir el nombre. API (13) (del inglés Application Programming Interface) es un conjunto de definiciones y de protocolos que permiten la comunicación entre dos aplicaciones.
Gracias a este tipo de *software*, es posible ahorrar tiempo de desarrollo, decrementando el coste total. En este trabajo se utiliza una API para comunicar la interfaz de la aplicación con el módulo de persistencia. Implementar una API en la aplicación presenta varias ventajas:

- Permite la reutilización del código. Esto permite reducir el coste y tiempo asociado al desarrollo de las tareas
- Es sencilla de construir
- No consume muchos recursos
- Permite definir puntos de acceso determinados

REST (14) (REpresentational State Transfer), por otro lado, hace referencia a la forma de interactuar con el servidor HTTP. Es un servicio que no tiene estado, dando a las peticiones independencia entre ellas. Para que el servidor recordase la sesión, sería necesario un usuario, token u otro tipo de credencial. En el caso concreto de este proyecto se va a utilizar el usuario para mantener la sesión en el servidor.

Algunas de las características más importantes de API-REST son que se utilizan los métodos HTTP (GET, POST, PUT, DELETE), el uso de URI (UniformResourceIdentifier) único para identificar a los recursos del sistema y la representación de los datos a través de, en el caso concreto de este trabajo, JSON.

3.6. LDA

En aprendizaje automático, la Asignación Latente de Dirichlet (LDA,Latent Dirichlet Allocation) (15) es un modelo que permite que conjuntos de observaciones puedan ser explicados por grupos no observados llamados tópicos. El modelo parte de la base de la existencia de estos tópicos o temas que aparecen en diferentes proporciones en cada uno de los textos. Se pueden ajustar los parámetros de las distribuciones para encontrar un conjunto de temas explicados como distribuciones discretas de palabras y los textos como mezclas ponderadas de estos temas, con pesos diferentes.

Esto permite encontrar tópicos generados por el modelo que se asignen a las viviendas accesibles. Se establece un peso mínimo en el tópico elegido para que se considere accesible, mediante la realización de diferentes pruebas. Es lógico pensar que muchos de los tópicos generados serán similares, ya que las descripciones de las viviendas suelen tener unos términos comunes a todas, como puede ser «habitación» o «baño». Además, la cantidad de viviendas que especifican en su descripción que son accesibles no es muy alta, por lo que se necesitará un conjunto amplio de tópicos y deben de ser diferentes entre sí. Estas características son ajustables en la generación del modelo LDA.

En el proyecto se utilizó una colección de descripciones extraídas por el sistema scraper para entrenar al modelo LDA y la generación de los tópicos. Uno de los tópicos generados por el modelo será el utilizado para el proceso de filtrado de viviendas.

Aunque puedan surgir problemas en la generación de los grupos, ya que la información que aparece en las descripciones de las viviendas muchas veces tiene varios elementos en común, como número de habitaciones, baños, si tiene terraza..., es la mejor herramienta combinada con el scraping para generar el modelo de procesamiento de lenguaje funcional de la aplicación.

4. Técnicas y herramientas

En este apartado se procede a explicar las técnicas y herramientas concretas utilizadas para el desarrollo e implementación de las funcionalidades del proyecto.

4.1. Entorno de desarrollo principal

El entorno de desarrollo se basa en un sistema operativo Windows. Se ha elegido este sistema operativo para alojar el entorno de desarrollo por dos motivos: El primero de ellos, es porque es el sistema operativo instalado en el portátil del desarrollador de este trabajo de fin de grado. El segundo, por la comodidad y familiaridad con el sistema operativo, ya que es el que más ha utilizado el desarrollador. Además, en ese sistema operativo se encuentran los editores de código con los que el desarrollador está familiarizado.

4.1.1. Visual Studio Code

Visual Studio Code (16) es uno de los editores de código más conocidos, utilizados y flexibles (17). Dispone de extensiones que se adaptan a todo tipo de lenguajes de programación y necesidades del programador. Debido a que en el proyecto se va a trabajar con diferentes lenguajes, se ha decidido utilizar este editor de código. Además, es el editor de código con el que más familiarizado se encuentra el desarrollador, habiéndolo usado tanto en el entorno laboral como en el entorno estudiantil.
La extensión más importante que se ha utilizado es Pylance (18), la cual permite probar y encontrar errores en el código de una forma sencilla.

4.1.2. Python

Python (19) es uno de los lenguajes de programación más utilizados hoy en día (17), debido a su amplia variedad de bibliotecas que ofrecen funcionalidades de gran valor para el desarrollo de un proyecto *software*. De hecho, esta es la gran razón por la que se ha decidido basar el back-end del proyecto en este lenguaje de programación, ya que ofrecía soluciones a la mayoría de desafíos planteados en el desarrollo del proyecto. Cabe destacar la rapidez en el aprendizaje en este lenguaje, ya que al comienzo del proyecto el desarrollador era un total inexperto y ha conseguido avanzar mucho y rápido en sus conocimientos.
Utilizando Python y sus bibliotecas se han construido 2 de los grandes bloques de la aplicación:

4.1.2.1. API-REST

El desarrollo de una API (20) en un proyecto de estas características es fundamental. Permite controlar los puntos de acceso a la información, realizando comprobaciones de seguridad y definiendo estos puntos de acceso. Se deben de definir las conexiones y el modo de comunicación con la API, que actúa como «controlador» en la aplicación. Para ello se ha utilizado una herramienta proporcionada por una biblioteca de Python : Flask.

4.1.2.1.1. Flask

Flask (21) es un *framework* escrito en Python que permite el desarrollo de aplicaciones web rápidamente y con un número de líneas de código reducido. Se basa en la especificación WSGI. El uso de este *framework* se debe principalmente a la rapidez con la que se puede desarrollar una aplicación web funcional, ya que con un simple script de 10 líneas podemos tener un servidor funcionando. Flask trae de serie un servidor de pruebas, ideal para entornos de desarrollo para encontrar posibles errores, pero ineficiente en cuanto a entorno comercial se refiere. Esta funcionalidad es ideal para el desarrollo de un proyecto como el mío, ya que permite avanzar en la funcionalidad de la aplicación y probarla sin tener que desarrollar toda una estructura de servidor.

Flask se caracteriza por la definición de rutas para el acceso a la información. A continuación, se muestra un ejemplo de ruta, en este caso la página principal de la aplicación.

```python
@app.route('/')
def main():

    return render_template('PagPrincipal.html')
```

Imagen 1 - Endpoint principal de la aplicación

Como se puede ver en Imagen 1 con 3 líneas de código se ha definido el endpoint principal de la aplicación. Se debe utilizar @app.route() para agregar la ruta a los *endpoints* de la aplicación, y después definir la función que da servicio a ese *endpoint*; en el caso que se muestra en Imagen 1, únicamente deberá renderizar el HTML correspondiente a la página principal de la app.

En la definición de la ruta se pueden añadir otros parámetros, como si se quiere utilizar algún parámetro en la función o los métodos de comunicación de REST que se pueden utilizar en el *endpoint*. Estas especificaciones permiten tanto personalizar el *endpoint* en función de diferentes valores, como actuar de forma diferente en función del método utilizado. En la aplicación se utilizan ambos procedimientos para desarrollar algunas de las funciones de la aplicación, por ejemplo:

```python
@app.route('/cambiarContraseña/<int:user_id>',methods=['GET', 'POST'])
@login_required
def cambiarContraseña(user_id):
    user=load_user(user_id)
    if request.method=='POST':
        oldPassword=request.form['oldPassword']
        newPassword=request.form['newPassword']
        repeatPassword=request.form['repeatPassword']
        if repeatPassword != newPassword:
            return render_template('CambiarContraseña.html',user=user,user_id=user.id, error="La contraseña no ha sido confirmada con
        else:
            control=accesoBaseDatos.cambiarPass(user.id,oldPassword,newPassword)
            if control:
                return redirect(url_for('verDatos', user_id=user.id))
            else:
                return render_template('CambiarContraseña.html', user=user,error="La contraseña antigua no es correcta")
    return render_template('CambiarContraseña.html',user=user)

@app.route('/ascenderUsuario',methods=["GET","POST"])
```

Imagen 2 – Endpoint cambiarContraseña

El código mostrado en Imagen 2 se corresponde con la función de cambio de contraseña de la aplicación. A diferencia del código de la página principal, aquí se ha añadido el parámetro user_id a la ruta del endpoint,

lo que permite crear páginas individuales a cada usuario. Además, se definen los métodos GET y POST para el acceso a este *endpoint*, actuando de forma diferente en función del método utilizado en la petición.

4.1.2.1.2. Flask.login_required

El *decorator* login_required ofrecido por flask permite limitar el acceso a un *endpoint* únicamente a usuarios que tengan una sesión iniciada en el sistema. Se puede observar su uso en la función mostrada en Imagen 2, donde sólo tiene sentido que tengan acceso al cambio de contraseña aquellos usuarios que tengan sesión en el mismo. Este decorador aporta bastante seguridad en el acceso a información sensible. Es utilizado múltiples veces a lo largo de la aplicación para limitar el acceso a la información

4.1.2.1.3. Definición de usuario

Es importante definir la clase usuario con la que va a trabajar flask. Esta definición de usuario no tiene que tener necesariamente los mismos parámetros que la definición de usuario realizada en la base de datos del sistema, solo debe de tener aquellos que ayuden al control de acceso y que proporcionen una utilidad real. En el caso de la aplicación, la definición de la clase usuario luce de la siguiente manera:

```
class User(UserMixin):
    def __init__(self, id,role):
        self.id = id
        self.role = role
```

Imagen 3 - Clase User (UserMixin)

En la clase User mostrada en Imagen 3, se definen los parámetros de id y de rol. El valor id es único para cualquier usuario, y el valor rol diferencia a los diferentes tipos de usuario en el sistema, siendo estos administrador y usuario. Este último parámetro en conjunto con el *decorator* login_required es la base de la protección de información sensible de la aplicación, ya que es usado de una forma similar, limitando el acceso a cierta información únicamente a los usuarios administradores.

4.1.2.1.4. Request

En la comunicación con una API-REST el papel de las peticiones es fundamental. Una *request* (22) o petición es una llamada que el cliente realiza para enviar o solicitar información a través de los protocolos definidos por REST. Existen varios tipos de *requests*, y los principales métodos son GET, POST, PUT y DELETE. Las peticiones realizadas en la API únicamente utilizan los métodos GET y POST.

El método GET es normalmente utilizado para pedir información. En el proyecto se usa en repetidas ocasiones para consultar la información de las viviendas y los usuarios. El método POST se utiliza generalmente cuando se quiere enviar datos al servidor, ya sea para su almacenamiento o su procesamiento.

Además del método, es importante tener en cuenta la siguiente información:

- URL. Se necesita una URL a la que enviar la petición. Esta URL debe ser definida por la API y permitir uno de los métodos de REST. Cada URL de la API tiene una funcionalidad asociada diferente, y hay veces que presentan restricciones de seguridad.
- Body. Es el cuerpo de la solicitud y se utiliza para enviar información adicional en la petición. La utilizaré en aquellas peticiones que utilicen el método POST y principalmente se utilizan para mandar información relativa a las viviendas o a los usuarios, en función de la funcionalidad que se esté accediendo en ese momento.

4.1.2.2. Gestión de viviendas

Para la gestión de las viviendas, objetivo fundamental de la aplicación, se ha utilizado una serie de herramientas que permiten recoger la información de las viviendas, a través de un *software* de web scraping, y que permite filtrarlas y diferenciar las viviendas accesibles o adaptadas de las que no lo son. Todas las tecnologías descritas en este apartado han sido utilizadas en este proyecto con la única función de satisfacer estas necesidades.

4.1.2.2.1. Scrapy

Scrapy (23) es una herramienta de extracción de datos de código abierto que tiene como objetivo principal extraer la información de un sitio web determinado. Es una herramienta que sirve para el desarrollo arañas web y realizar tareas de web scraping de manera eficiente. Se ha elegido esta implementación por varios motivos:

- Posibilidad de crear y ejecutar crawler o arañas web. Permite crear y ejecutar rastreadores web que funcionan de manera automática y sistematizada para inspeccionar los datos de determinados sitios web.
- Implementación en Python. Conviene su implementación en Python para favorecer la fácil integración del proyecto.
- Puede ejecutarse en sistemas Windows, Linux, mac y BSD, lo que beneficia enormemente a la `portabilidad del sistema.
- Destaca por la velocidad y resistencia en la extracción de datos, lo que beneficia a la fiabilidad del sistema

Entrando más en la implementación concreta, se debe utilizar la biblioteca scrapy de Python para crear un proyecto Scrapy. Una vez creado el proyecto, dentro de la carpeta spiders, generada en la creación del proyecto, se deben crear los archivos de araña. En el caso de este proyecto, solo se ha necesitado de una araña, pero en un futuro podrían crearse nuevas arañas para recoger información de otros sitios web.
En el script correspondiente a la araña se debe definir la estructura de la misma. Hay algunos parámetros que son clave para el funcionamiento de la araña:

- Definición de la clase Spider: Todo el contenido del script debe de estar dentro de una clase Spider.
- Nombre de la araña
- Conjunto de urls que debe de visitar la araña
- Definición de la función parse, la cual se encarga de definir el procesamiento de la respuesta para extraer la información deseada y manejar excepciones

Una vez creada la araña, simplemente se debe ejecutarla. En el caso concreto de la araña implementada en el proyecto, la definición de las urls se realiza siguiendo el patrón explicado en 3.1.- Web Scraping, almacenando el id correspondiente a la última vivienda visitada con éxito. De esta manera se asegura no visitar viviendas repetidas. Para evitar bucles de acceso a urls no disponibles, se realizan ciclos de 1000 viviendas en cada ejecución. Gracias a las características de Scrapy, este proceso supone un coste muy bajo.

En cuanto al comportamiento de la araña, deberá de buscar en el html de la web visitada los valores que se quieren almacenar de la vivienda. Concretamente busca los valores de precio, descripción y título. Antes de guardar la información en la base de datos del sistema, debe de pasar el filtro para ver si es accesible o no. Una vez decidido, si es accesible la guarda en la base de datos del sistema a través de una función del módulo de acceso a la base de datos. Antes de almacenar la vivienda en la base de datos, se busca la ciudad a la que pertenece la ciudad en la descripción de la misma. Se incluye esta información, la url de la vivienda y los datos recogidos por la araña.

4.1.2.2.2. NLTK

Natural Language ToolKit (24) (NLTK) es conjunto de librerías y programas que permite llevar a cabo muchas tareas relacionadas con el procesamiento del lenguaje natural. En el proyecto ha supuesto una herramienta de apoyo para la herramienta realmente elegida para el procesamiento del lenguaje natural, pero ofrece funciones realmente útiles para habilitar la otra herramienta.

A grandes rasgos, se ha utilizado NLTK para transformar el texto plano extraído de la web a términos que permitan el procesamiento computacional del texto. Se transforma la descripción extraída de la url de la vivienda, eliminando los signos de puntuación, números y palabras de una sola letra. Una vez reducido el texto, se extraen las raíces etimológicas de las palabras restantes. Este conjunto de raíces se denominan tokens, y son la entrada del modelo LDA, que es la herramienta que realmente se utiliza para el procesamiento de lenguaje.

4.1.2.2.3. Gensim

Gensim (25) es una herramienta que permite implementar el algoritmo LDA. Utiliza la técnica de topicmodeling, que es una técnica no supervisada de NLP para la extracción de tópicos de conjuntos (26). Para poder utilizar esta herramienta es importante tener un conjunto de documentos, los cuales son secuencias de tokens, llamado corpus, que servirá como base para la herramienta. Este conjunto de tokens se obtiene

gracias a scrapy, extrayendo las descripciones de las viviendas, y NLTK, transformando estas descripciones en tokens. Es importante que, a la hora de crear el diccionario, se incluyan los tokens como una secuencia de secuencias, concretamente una lista de tuplas en Python, ya que la distribución de los tokens importa en la definición de los tópicos. Usando este diccionario y el corpus de las viviendas se crea el modelo LDA.

En la creación del modelo LDA, pueden especificarse una serie de parámetros que permiten que el modelo se ajuste lo máximo posible a las necesidades del proyecto. Estos parámetros se deben ajustar en función a varios parámetros, como el tamaño del corpus o lo diferentes que son entre sí los componentes del corpus.

```
diccionario = Dictionary(df.Tokens)
print(f'Número de tokens: {len(diccionario)}')

corpus = [diccionario.doc2bow(Descripcion) for Descripcion in df.Tokens]

lda = LdaModel(corpus=corpus, id2word=diccionario,
               num_topics=100, random_state=42,
               chunksize=509, passes=20, alpha='auto')
```

Imagen 4 - Creación de modelo LDA y diccionario

4.1.2.2.4. Geonamescache

Geonamescache (27) es una biblioteca de Python que ofrece funciones para trabajar con ciudades, países... En el caso de este proyecto, se utiliza para encontrar las ciudades a las que pertenecen las viviendas, ya que era un proceso más rápido y sencillo que encontrarlas en el html debido a la estructura del mismo.

Geonamescache proporciona los nombres de las ciudades de España y Portugal, lo que permite que la araña antes de guardar la información en la base de datos busque la ciudad en la descripción de la vivienda.

```
gc = GeonamesCache()
# Obtener todas las ciudades
ciudades = gc.get_cities()
# Función para buscar ciudades en un texto
def buscar_ciudad(texto):
    for ciudad in ciudades.values():
        nombre_ciudad = ciudad['name']
        # Comprobar si el nombre de la ciudad está presente en el texto
        if re.search(r'\b{}\b'.format(nombre_ciudad), texto, re.IGNORECASE):
            # Asignar el nombre de la ciudad al campo "city" del JSON
            return nombre_ciudad
```

Imagen 5 - Uso geonamescache para encontrar las viviendas

4.1.3. Módulo de persistencia

El módulo de persistencia de una aplicación es el módulo encargado de garantizar tanto el almacenamiento como la recuperación de los datos a largo plazo. El módulo de persistencia de esta aplicación principalmente se centra en un servidor de base de datos mariaDB y un módulo de acceso a la misma. Adicionalmente, existen algunos archivos estáticos o variables de sistema que ayudan al correcto comportamiento de la aplicación.

4.1.3.1. MariaDB

MariaDB (28) es un sistema de gestión de bases de datos derivado de mySQL (29). Al derivar de mySQL se trata de un modelo de base de datos relacional, por lo que la información se almacena en forma de tablas, donde cada fila está identificada con un valor o clave única llamado id y cada columna representa un atributo del elemento identificado por id.

Aunque este tipo de sistemas son bastante rígidos en cuanto a la estructura de los datos, la información que se almacena en el sistema tiene siempre los mismos componentes, por lo que esta, que es su principal desventaja, no tiene efecto en el sistema.

4.1.3.2. Acceso base de datos

Para el acceso a la base de datos, se ha decidido implementar un módulo que proporciona funciones para realizar operaciones específicas en la base de datos. La implementación de este módulo ha sido realizada para separar la lógica de negocio de la estructura interna de la base de datos. De esta forma, el controlador puede hacer operaciones en las tablas sin la necesidad de conocer su estructura interna.

4.1.3.2.1. Mysql.connector

Para realizar la conexión a la base de datos, en el módulo de acceso se utiliza la biblioteca mysql.connector (30) de Python. Permite, introduciendo los parámetros de acceso, conectarse al servidor de base de datos de MariaDB.

4.1.4. Interfaz de Usuario

La interfaz de usuario representa la vista del proyecto. Es el módulo con el que el usuario interactúa. Es importante definir la comunicación entre la interfaz y la API-REST de flask. Para el diseño de la interfaz de usuario se ha utilizado HTML, CSS y JavaScript.

El diseño de la interfaz de usuario se basa principalmente en la facilidad de uso. Las diferentes funcionalidades del sistema son fáciles de acceder y utilizar. Existen interfaces que cumplen la misma funcionalidad o una muy similar, pero adaptadas al tipo de usuario que la está utilizando. Por ejemplo, aunque la funcionalidad es muy similar a la hora de buscar viviendas, la interfaz que muestra las viviendas a los usuarios registrados muestra si la vivienda está en la lista de favoritas del usuario en cuestión. Esto como es obvio, en el caso de un usuario sin registrar no aparece. Ocurre algo parecido con la pantalla principal, donde en los 3 casos sigue una estructura similar pero con opciones diferentes.

El diseño de las interfaces se ha basado en las interfaces de los principales buscadores de viviendas existentes. Principalmente de fotocasa e idealista.

4.1.4.1. HTML y CSS

HTML (31) (Lenguaje de Marcas de Hipertexto, Hyper Text MarkupLanguage) es el componente más básico de la web. Como su nombre indica, es un lenguaje marcado, que permite indicar la estructura de un documento mediante etiquetas. Ofrece gran adaptabilidad, una estructuración lógica y es fácilmente interpretable tanto por personas como por máquinas.

Toda la estructura de las interfaces del proyecto está definida en HTML. Para realizar la comunicación entre la interfaz y la API-REST se debe configurar un *request* indicando el punto de acceso que se quiere utilizar, el método de REST que se quiere utilizar y, si se necesita, información adicional para la consulta.

CSS (32) (Hojas de Estilo en Cascada, Cascading Style Sheets) es el lenguaje de estilos utilizado para describir la presentación de documentos HTML. Describe como debe de renderizar el elemento estructurado en la pantalla. Permite definir clases de estilo las cuales pueden ser asociadas a elementos HTML.

4.1.4.2. JavaScript

JavaScript (33) (JS) es un lenguaje de programación interpretado orientado a objetos. Se utiliza en la mayoría de páginas webs modernas para dotarlas de mejoras dinámicas e interactivas. Es el caso de la aplicación. Se ha utilizado este lenguaje para tener una experiencia de usuario más fluida y dinámica.

4.2. Entorno virtual (VirtualBox)

Oracle VM VirtualBox (34) es un *software* de virtualización. Por medio de esta aplicación es posible instalar otros sistemas operativos adicionales, conocidos como sistemas invitados dentro de otro sistema operativo anfitrión. En el caso de este proyecto el sistema operativo anfitrión es Windows 11 y el sistema operativo invitado es Ubuntu (35) 20.04. Se ha elegido esta opción de *software* de virtualización debido a las diferentes facilidades que ofrece a la hora de compartir recursos (fundamental para utilizar los archivos alojados en el sistema de almacenamiento del sistema anfitrión como si fueran propios), así como facilidades para la redirección de puertos (necesario para el acceso al servidor de base de datos alojado en el sistema anfitrión).

Además, se ha decidido virtualizar un sistema operativo Linux debido a la herramienta cron que ofrecen estos sistemas operativos. Es una herramienta útil en el desarrollo de aplicaciones cuando se necesita una ejecución periódica de scripts. En el caso de este proyecto, se necesita la ejecución periódica de:

- La araña scrapy, para que pueda encontrar viviendas constantemente.
- El script verificarViviendas, que accede a la función verificarViviendasCRON del componente de acceso a la base de datos del sistema, para eliminar o modificar las viviendas que ya no están disponibles en la página web.

4.2.1. Compartir recursos

Para poder compartir una carpeta del sistema anfitrión con el sistema virtualizado por VirtualBox es necesario insertar la imagen CD de las GuestAdditions (vboxguestadditions.iso). VirtualBox GuestAdditions es un

conjunto de controladores y utilidades que mejoran la integración entre el sistema anfitrión (Windows 11) y el virtualizado (Ubuntu 20.04).

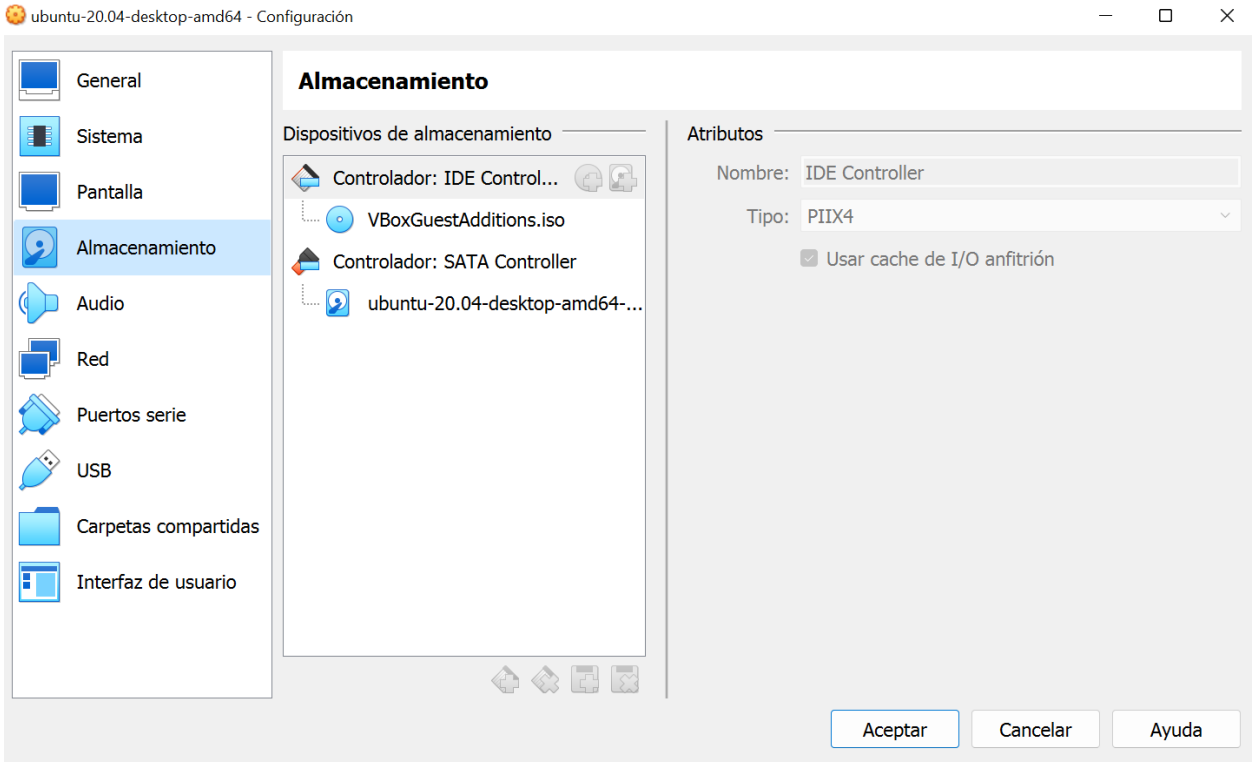


Imagen 6 - VBoxGuestAdditions

Una vez instalada la imagen y todos sus controladores y utilidades de VirtualBox GuestAdditions, se debe añadir desde el menú de configuración de VirtualBox las carpetas que se necesiten compartir. En este caso, la carpeta correspondiente a TFG. Además, se pueden configurar un auto montaje en una ruta concreta del sistema de almacenamiento del sistema virtualizado, que en este caso es /PruebaServer.

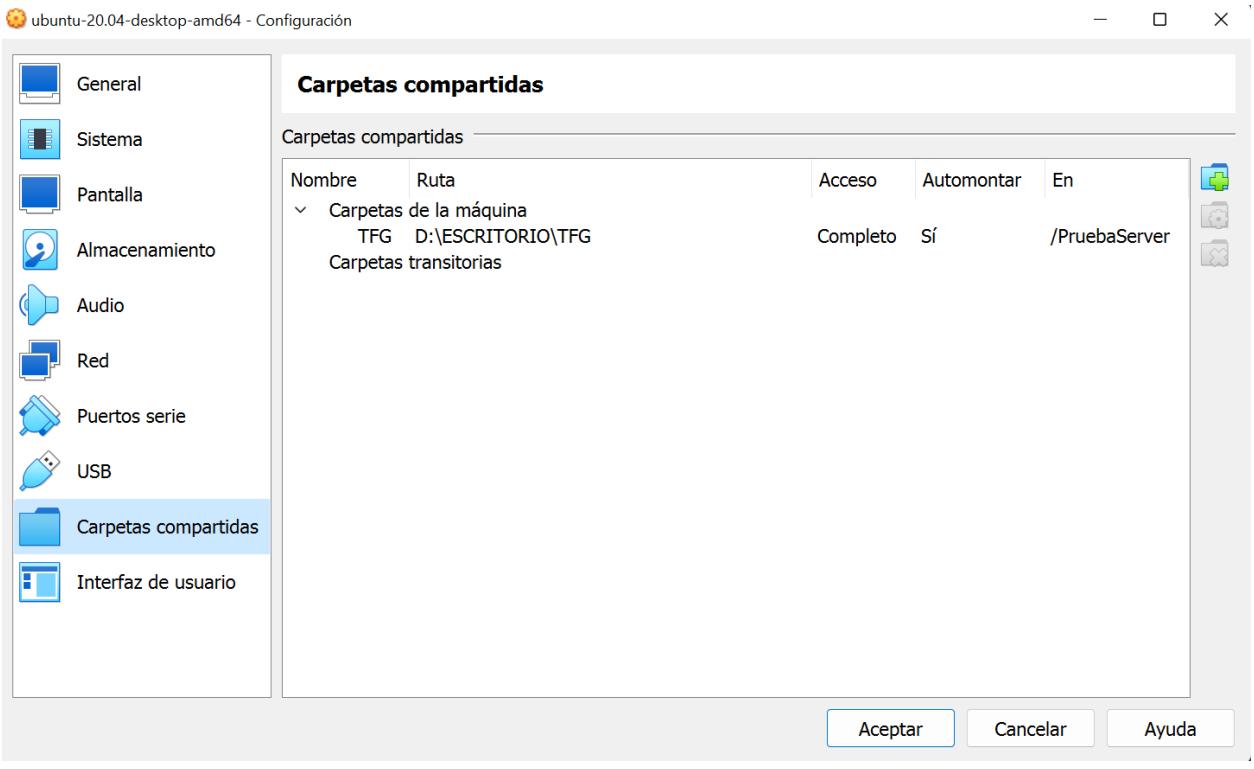


Imagen 7 - Compartir carpetas VirtualBox

4.2.2. Habilitar comunicación entre puertos

Para un correcto funcionamiento de la aplicación en la máquina virtual, es necesario habilitar la comunicación entre puertos de ambos sistemas. Hay que realizar principalmente dos cambios.

El primero de los cambios es permitir el acceso al servidor de bases de datos desde IPs externas. Para que las aplicaciones ejecutadas en contenedores de *software* externos al sistema anfitrión del servidor de bases sean capaces de acceder a la información. Para ello se debe editar el fichero de configuración del servidor de MariaDB my.ini, añadiendo la línea ‹bind-address=0.0.0.0».

4.2.3. Cron

Cron (36) es una herramienta proporcionada por los sistemas operativos Linux. Permite la ejecución de órdenes de manera periódica. Permite de una manera sencilla una programación de tareas. En el proyecto se utiliza para ejecutar periódicamente la araña Scrapy y para verificar la disponibilidad de las viviendas en la web original. Debido el modelo de implementación utilizado, las tareas del cron están definidas en el fichero cronjob. El modelo de implementación será el encargado de utilizar la herramienta cuando se ejecute. En el cronjob se debe indicar la ruta del fichero y la frecuencia igual que en los ficheros normales de cron. Ambas tareas se realizan cada hora

```
0 * * * * root python /app/VerificarViviendas.py
0 * * * * root python -m scrapy crawl todopisos
```

Imagen 8 – cronjob

4.2.4. Docker

Para el modelo de despliegue de la aplicación se ha decidido utilizar contenedores de *software*. Concretamente, se ha decidido utilizar la herramienta Docker (37). Los contenedores de *software* almacenan todo lo necesario para su ejecución, incluyendo bibliotecas, herramientas del sistema, código y tiempo de ejecución. Se ha elegido este modelo por varias razones:

- Rapidez. Dockerizar una aplicación web es un proceso relativamente rápido, es fácil de probar y es fácil aprender a usarla. Debido a las limitaciones temporales del proyecto era un modelo ideal en este aspecto.
- Escalabilidad. Las imágenes generadas por Docker pueden implementarse en cualquier ordenador con cualquier sistema operativo. Esto permite ajustar la escalabilidad del proyecto rápidamente según las necesidades que pueda tener el cliente.

En el apartado 5.3. Despliegue se comenta como utilizar esta herramienta.

5. Aspectos Relevantes del Desarrollo

Se recogen en este apartado algunos de los aspectos más relevantes en cuanto a la gestión y al desarrollo del proyecto. Los anexos entregados junto a este documento relatan con más detalle los siguientes temas:

Anexo	Tema Tratado
Anexo I	Planificación Temporal
Anexo II	Especificación y análisis de requisitos
Anexo III	Diseño del sistema
Anexo IV	Documentación técnica
Anexo V	Manual de usuario

Tabla 6 - Anexos

5.1. Planificación Temporal

Debido a las características del proyecto (incertidumbre de tecnologías utilizadas y limitaciones temporales) se ha decidido utilizar una metodología ágil. Más concretamente se ha decidido utilizar los sprints de Scrum (38), que se basan en el desarrollo intensivo de funcionalidades. La idea es diseñar tareas que engloben algún tipo de funcionalidad y que en el resultado final de cada tarea sea un prototipo funcional que tenga implementada la funcionalidad. Esto proporciona un prototipo incremental, al cual se le van añadiendo funcionalidades.

Es especialmente útil esta metodología si se piensa en añadir funcionalidades en un futuro, ya que es sencillo introducir nuevas funcionalidades al prototipo incremental. Además, permite ir definiendo la funcionalidad y alcance del proyecto en tiempo de desarrollo teniendo en cuenta principalmente la limitación temporal establecida por la fecha de entrega del mismo.

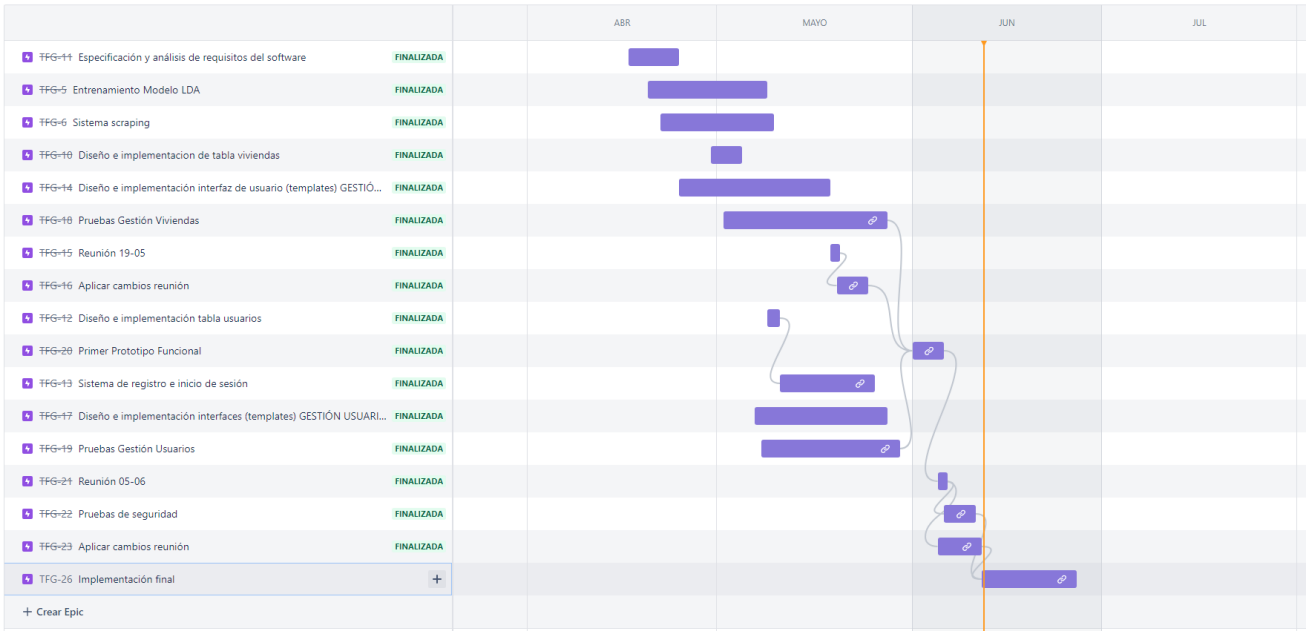


Imagen 9 - Planificación temporal del proyecto

En general, los tiempos estimados para el desarrollo de las tareas ha sido adecuado, ya que cuando se realizó la estimación se tuvieron en cuenta varios factores que tuvieron efecto en el coste temporal de desarrollo de las tareas, como por ejemplo la inexperiencia en las tecnologías utilizadas o simplemente el desconocimiento sobre que tecnología utilizar. Esta última es la que más efecto negativo ha tenido, ya que al tiempo de implementación de la funcionalidad hay que sumarle el tiempo de investigación de tecnologías.

En total, el desarrollo del trabajo ha constado de unas 360 horas cumpliendo con todas las tareas y objetivos planteados al inicio del proyecto, superando las horas inicialmente planteadas, 320 horas.

Para una descripción más detallada, consultar el Anexo I - Plan de proyecto *Software*

5.2. Arquitectura y diseño

El patrón principal utilizado en la arquitectura y diseño del sistema es el patrón Modelo-Vista-Controlador (MVC) (39), en el cual se diferencian la Vista, interfaces con la que interactúa el usuario y se corresponde con la representación de los datos, del modelo, que es la implementación real del módulo de persistencia del sistema y es dónde se almacenan los datos. Para ello existe un módulo intermedio, llamado controlador, que maneja las peticiones y respuestas entre vista y controlador.

La implementación concreta del controlador de la aplicación, una API-REST que tramita las peticiones y un servidor que las recibe, permite agregar funcionalidades fácilmente, ya que únicamente haría falta definir un nuevo endpoint en la API. Concretamente en el sistema se utiliza el servidor de prueba que implementa Flask.

Las interfaces se comunican con el controlador a través de request siguiendo el protocolo REST, estas peticiones son manejadas por el controlador y a través del módulo de acceso a la base de datos, realiza las consultas al modelo.

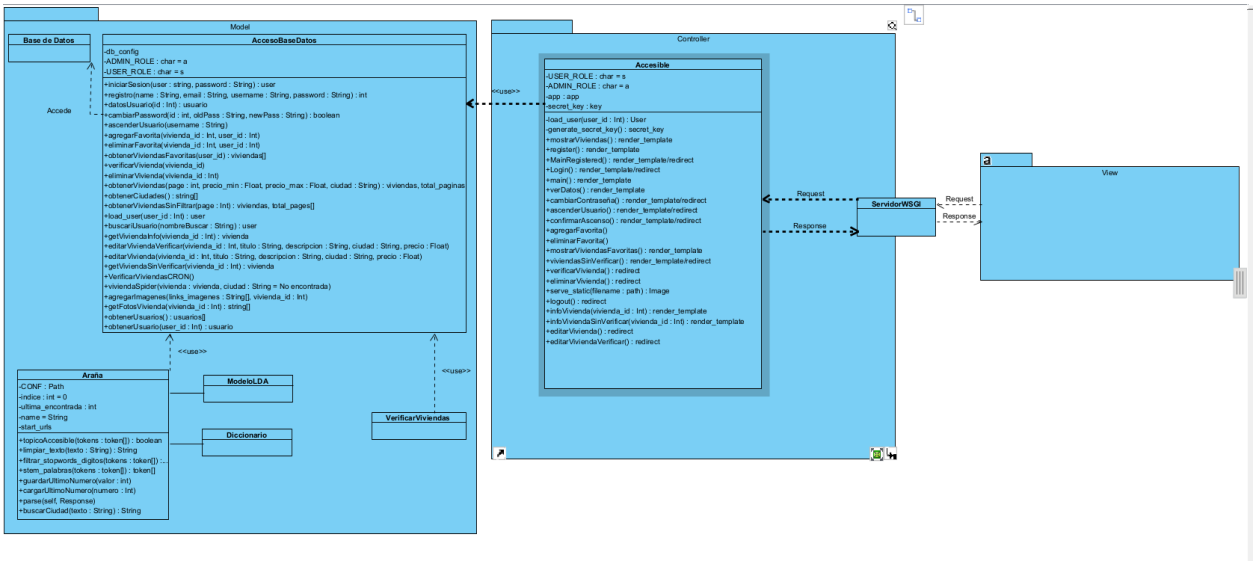


Imagen 10 - Diagrama de clases de diseño / Arquitectura del proyecto

Por temas de legilibilidad del diagrama, no se muestran todas las interfaces que se encuentran en la vista. Para una explicación más detallada de cada uno de los componentes de los módulos y, en general, del proceso de diseño en el desarrollo del proyecto, consulte el Anexo III - Especificación de Diseño.

5.2.1. Modelo LDA

En el caso concreto de este proyecto, tras un proceso de scraping largo, se consiguió un corpus compuesto por la descripción de 200.000 viviendas extraídas de la web seleccionada. Utilizando este corpus, y al tratarse de, en muchos casos, descripciones similares, hubo la necesidad de crear un modelo LDA con un número de tópicos alto, para que permita encontrar un tópico que sea de viviendas accesibles o adaptadas. Además, mediante el ajuste de los parámetros de Gensim, los tópicos definidos son altamente diferentes entre sí. Esto se consigue mediante el ajuste del parámetro *eta*, que controla la distribución de palabras en los tópicos e influye en la diversidad de los tópicos generados. Un valor pequeño de *eta*, como 0.05, genera tópicos más distintos y especializados. El número de tópicos se consigue ajustado el parámetro num_topics de la función, que tras varias pruebas, se ha elegido 75 tópicos. Se ha elegido un valor tan alto de tópicos para conseguir un tópico que utilizará tokens clave para identificar viviendas accesibles con un valor alto en su peso ponderado. Gensim permite la consulta de la estructura de los tópicos, mediante la función show_topics(). A esta función se le puede especificar el número de tópicos mostrados y los tokens por tópico que se quieren mostrar. Abriendo la salida de esta función en un editor de texto, se puede realizar un análisis rápido de los tópicos.

Estos tópicos, la mayoría de ellos y aunque se haya ajustado el valor *eta*, hacen referencia a los mismos términos, comunes a todas las descripciones de las viviendas. Esto es normal teniendo en cuenta la estructura general de la descripción de una vivienda: Número de habitaciones, baños, estructura de la casa... De todos los tópicos generados, hay un tópico que destaca para el propósito del proyecto:

0.059*"proteg" + 0.057*"arquitecton" + 0.051*"levant" + 0.045*"sof" + 0.037*"barrer" + 0.037*"limpiez" + 0.031*"deni" + 0.030*"bellez" + 0.029*"bienest"

El texto mostrado se corresponde con los tokens que más peso tienen en el tópico y el peso ponderado que aportan. Como se puede ver aparecen términos como «arquitecton», «barrer» y «bienest», que corresponden a barreras arquitectónicas y bienestar, que son términos que hacen referencia directa a las viviendas accesibles, ya sea porque explican que las viviendas no tienen barreras arquitectónicas o que buscan el bienestar de la persona inquilina. Se elige este tópico ya que estas palabras aportan mucho peso dentro del tópico e identifica bien a las viviendas adaptadas.

Una vez definido el modelo LDA y el diccionario, se usan en la función topicoAccesible para determinar si una vivienda es accesible o no.

```
def topicoAccesible (vivienda):
    """
    En esta función se decide si la descripción de una vivienda se corresponde con la de una vivienda accesible
    :vivienda Conjunto de tokens presente en la descripción de la vivienda
    """


    bow_articulo_nuevo = diccionario.doc2bow(vivienda)
    i=0

    for topico in lda[bow_articulo_nuevo]:
        i+=1
        if topico[0] == topico: #Valor del tópico de nuestro modelo LDA que contiene viviendas accesibles


            if topico[1] > valor_corte: #Valor de corte

                return True

            else:

                return False

    return False
```

Imagen 11 - Función topicoAccesible

El valor de valor_corte se ha determinado tras realizar varias pruebas, en las cuales los valores probados anteriormente dejaban pasar demasiadas viviendas no accesibles o no dejaban pasar casi ninguna vivienda.

5.2.2. Diseño de datos en la base de datos

En el servidor MariaDB de base de datos se han definido varias tablas para almacenar la información de la base de datos. Para el diseño de la base de datos se ha tenido en cuenta las limitaciones de los modelos de bases de datos relacionales y las necesidades del sistema. A grandes rasgos:

- Tabla users: Almacena la información de los usuarios registrados en el sistema
- Tabla infoViviendas: Almacena la información de las viviendas verificadas como accesibles
- Tabla viviendasFavoritas: Guarda relaciones entre usuarios y viviendas. Las viviendas están en la lista de favoritas del usuario al que están relacionados

- Tabla viviendasSinFiltrar: Guarda la información de las viviendas que han pasado el filtro de procesamiento de lenguaje natural pero no han sido verificadas por un administrador como accesibles
- Tabla imagenes: Almacena la información de las imágenes relacionadas con las viviendas.

El único problema que podría surgir al utilizar un modelo de base de datos relacional es la tabla imagenes, ya que este tipo de modelos no es muy bueno para manejar datos multimedia. Por ello, se guarda la url de obtención del recurso en lugar de la imagen en sí.

Se usan las claves externas para establecer las relaciones entre los usuarios y las viviendas en la tabla de viviendasFavoritas, para asegurar la existencia de ambos elementos en la base de datos. También se utilizan claves externas para establecer las relaciones entre las imágenes y las viviendas. Para poder realizar estas relaciones, es necesario que, tanto las viviendas como los usuarios, estén identificados por un parámetro único llamado id. La relación que se establece entre viviendas y usuarios se realiza creando una tupla en la que aparezcan los identificadores de la vivienda y el usuario. Para la imagen es algo similar, relacionando la url de la imagen con el id de la vivienda a la que corresponde.

5.2.3. Interfaz

En este apartado se explican algunos conceptos del diseño de las interfaces. Concretamente, se explicará el uso de HTML, CSS y JavaScript en la aplicación. Para información más detallada, consulte el Anexo IV - Documentación técnica de programador.

Cabe destacar que en el caso de este proyecto se ha incluido la hoja de estilos y las funciones de javaScript dentro de los propios archivos HTML, mediante las etiquetas <style> y <script>.

5.2.3.1. Diseño HTML y CSS

Como se ha explicado en 4.1.4.1.- HTML y CSS, estos lenguajes se utilizan para implementar la interfaz de la aplicación. Explicar en profundidad todas y cada una de las interfaces y sus correspondientes hojas de estilo sería repetitivo. Se procede a explicar un ejemplo genérico que sirve de base para el resto de interfaces.

```
.sidebar {
    background-color: #4CAF50;
    padding: 10px;
    width: 100%;
    text-align: center;
    position: fixed;
    top: 0px;
}
```

Imagen 12 - Clase de estilo sidebar

En Imagen 12 se muestra la clase definición de la clase de estilo sidebar. Esta clase se utiliza para la barra que aparece en la parte superior de la interfaz mostrarViviendas. En esta clase se especifica el color del elemento, que el texto debe de estar centrado en este contenedor y la posición y tamaño del elemento. Para poder utilizar la clase debemos de asignársela a un elemento de la interfaz.

```
<div class="sidebar">···
</div>
```

Imagen 13 - Asignación clase sidebar

Al asignarle la clase *sidebar* al contenedor, en este caso que almacena los campos del formulario para filtrar las viviendas mostradas y un botón para volver a la página principal, se comporten como describe la clase.

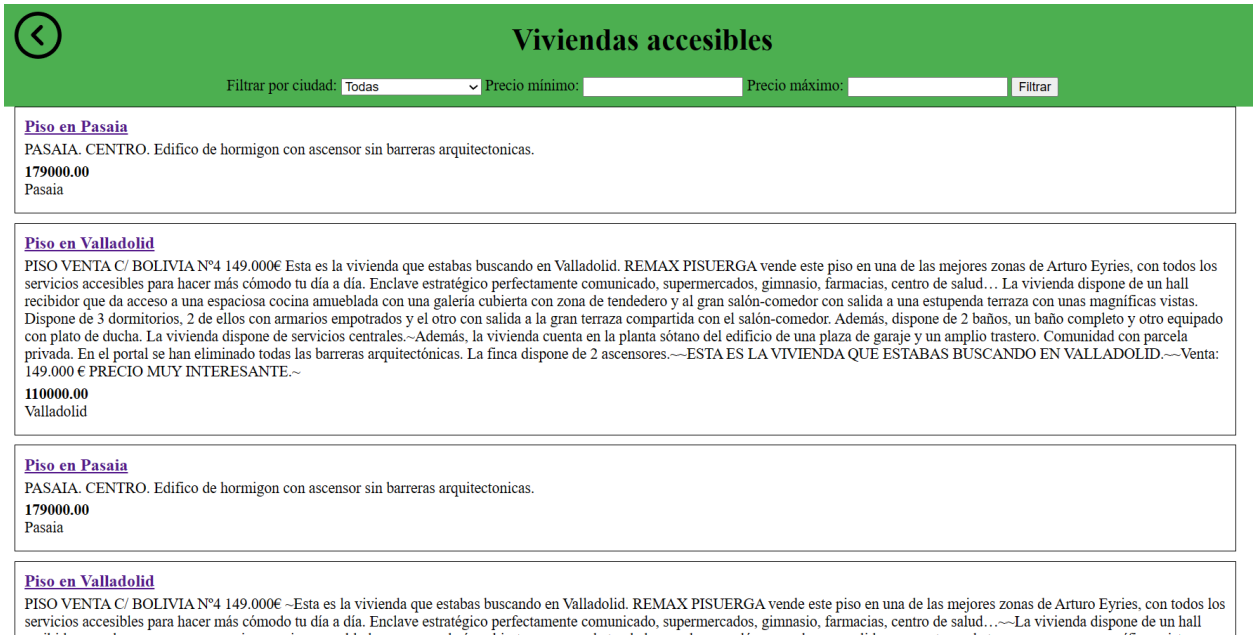


Imagen 14 - Interfaz mostrar viviendas.

Como se observa en Imagen 14, la barra de la parte superior tiene la posición establecida por el estilo y todos los elementos, salvo el botón para volver a la página principal que tiene una clase que lo coloca en ese espacio, se sitúan centrados.

```
.content-container {
    margin-top: 100px;
    padding: 20px;
}
.vivienda {
    margin-bottom: 10px;
    padding: 10px;
    border: 1px solid #000;
}
```

Imagen 15 - Hojas de estilo contenedor de vivienda

Otro de los elementos que destaca dentro de la interfaz mostrarViviendas, son los contenedores de información que almacenan la información relativa a la vivienda. Se crea un contenedor de contenido en el que poder crear otros elementos con la clase *content-container*. Los contenedores individuales de las viviendas se crean con la clase vivienda, que rodea a cada uno de ellos con una línea negra. Con los estilos mostrados en Imagen 15 y los estilos de los diferentes atributos de una vivienda (título, descripción, precio y ciudad) se crea bucle *for* para crear los contenedores de todas las viviendas.

```
{% if viviendas %}
<div class="content-container">
    {% for vivienda in viviendas %}

        <div class="vivienda">
            <div class="titulo"><a href="{{ url_for('infoVivienda', vivienda_id=vivienda[0]) }}">{{ vivienda[1] }}</a></div>
            <div class="descripcion">{{ vivienda[5] }}</div>
            <div class="precio">{{ vivienda[3] }}</div>
            <div class="descripcion">{{ vivienda[4] }}</div>
        </div>

    {% endfor %}
</div>
{% else %}
    <p>No se encontraron viviendas.</p>
{% endif %}
```

Imagen 16 - HTML para mostrar contenedores de viviendas

En la Imagen 16 se muestra el bucle *for* mencionado, creando el contenedor de las viviendas y creando cada uno de los contenedores individuales de las viviendas.

Las solicitudes son enviadas desde los archivos HTML. A continuación, se muestra un ejemplo de cómo se realiza la petición para obtener la información de las viviendas en función de los filtros introducidos.

```
<div class="search-bar">
    <form action="{{ url_for('mostrar_viviendas') }}" method="GET">
        <input class="search-input" id="city-input" type="text" name="ciudad" placeholder="Ciudad">
        <input class="search-input" id="min-price-input" type="number" name="precio_min" placeholder="Precio min">
        <input class="search-input" id="max-price-input" type="number" name="precio_max" placeholder="Precio max">
        <button type="submit" class="search-button">
            <img src="{{ url_for('serve_static', filename='PNG/lupa.jpg') }}" alt="Buscar" class="boton_lupa" />
        </button>
    </form>
</div>
```

Imagen 17 – Request mostrar_viviendas

En Imagen 17 se muestra como se realiza la petición a mostrar_viviendas utilizando el método GET. En él se define un formulario en el que se recogen los datos del precio máximo, precio mínimo y ciudad. Estos datos son enviados junto con la petición al pulsar el botón identificado por la clase *search-button*. Como se puede ver, se hace otra petición a la API, en este caso para recuperar una imagen y asignársela al botón. En esta petición se puede ver como se pasa la variable *filename* como parámetro a la URL.

Para acceder a todas las funcionalidades de la aplicación se utilizan los *requests*, modificando la información del *body* o el método utilizado pero el esquema general es el mostrado en la Imagen 17.

5.2.3.2. Funciones JavaScript

Para mejorar la agilidad de las interfaces de la aplicación, se han creado algunas funciones en JavaScript.

Uno de los casos en los que se ha utilizado JS ha sido a la hora de agregar y borrar las viviendas de la lista de favoritas. Cuando el usuario pulsa en el botón habilitado para estas tareas, el código JS se encarga tanto de cambiar la imagen del botón (por ejemplo si el usuario la agrega a favoritas, cambia la imagen de no favorita a favorita) y realizar la consulta a la API de manera dinámica.

```
// Obtener todos los botones con las clases "agregar-favorita" y "eliminar-favorita"
const botones = document.querySelectorAll('.agregar-favorita, .eliminar-favorita');

// Iterar sobre cada botón y agregar un evento de clic
botones.forEach((boton) => {
    boton.addEventListener('click', (event) => {
        event.preventDefault(); // Evitar el comportamiento predeterminado del botón

        const viviendaId = boton.dataset.vivienda;

        // Crear una instancia de XMLHttpRequest
        const conexion = new XMLHttpRequest();

        // Determinar la acción según la clase del botón
        let url, nuevaClase, nuevaImagen;
        if (boton.classList.contains('agregar-favorita')) {
            url = '/agregarFavorita';
            nuevaClase = 'eliminar-favorita';
            nuevaImagen = "{{ url_for('serve_static', filename='PNG/corazonLLeno.png') }}";
        } else {
            url = '/eliminarFavorita';
            nuevaClase = 'agregar-favorita';
            nuevaImagen = "{{ url_for('serve_static', filename='PNG/corazonVacio.png') }}";
        }
```

Imagen 18 - Cambiar botones JS

En Imagen 18 se muestra como el código JS actúa, cambiando el funcionamiento del botón y actualizando su imagen en tiempo de ejecución. Es importante darse cuenta como varía su comportamiento en función de la clase que tenía el botón cuando fue pulsado. Esta información se establece inicialmente con los datos de la lista de favoritas del usuario.

```
        // Configurar la solicitud
        conexion.open('POST', url);
        conexion.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

        // Configurar la función de respuesta
        conexion.onload = function() {
            if (conexion.status === 200) {
                // La solicitud se completó exitosamente, puedes realizar acciones adicionales si es necesario

                // Cambiar la clase y la imagen del botón
                boton.classList.remove('agregar-favorita', 'eliminar-favorita');
                boton.classList.add(nuevaClase);
                const imagen = boton.querySelector('img');
                imagen.src = nuevaImagen;

                console.log('Acción realizada correctamente');
            }
        };

        // Enviar la solicitud con el ID de la vivienda
        conexion.send(`vivienda_id=${viviendaId}`);
```

Imagen 19 - Configuración de la solicitud

Una vez establecidas la nueva clase, nueva imagen y la URL de acceso a la API, se realiza la conexión y se realiza la acción correspondiente al id de la vivienda en cuestión.

Además de en ese caso, otro de los casos en los que la funcionalidad implementada en JS se hace notar es a la hora de ascender usuarios a administradores, más concretamente, a la hora de buscar el usuario al que se quiere ascender a administrador.

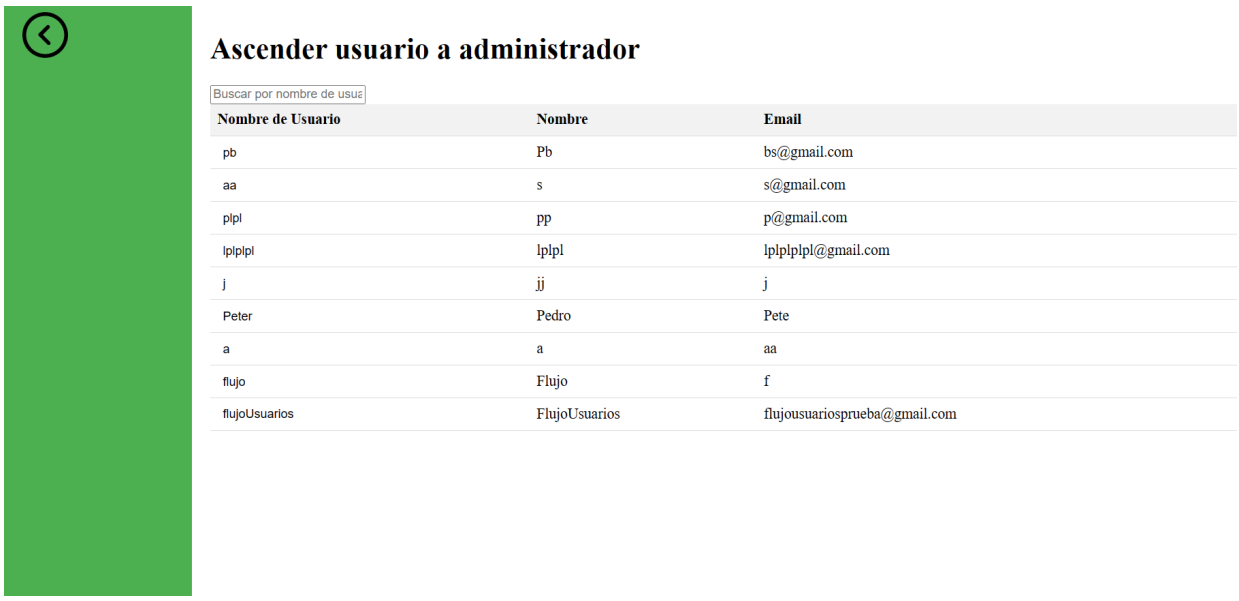


Imagen 20 - Interfaz ascender usuario a administrador

En la interfaz ascender usuario a administrador se muestra una tabla con todos los usuarios del sistema. Para buscar el usuario al que se quiere ascender hay una barra de búsqueda situada en la parte superior que permite buscar el nombre de usuario. Al escribir se van actualizando los datos de la tabla, mostrando únicamente los usuarios que coincidan total o parcialmente con el texto introducido. Esto se logra a través de una función de JS.

```
function buscarUsuarios() {
    var input = document.getElementById("busqueda");
    var filtro = input.value.toUpperCase();
    var tabla = document.getElementById("tablaUsuarios");
    var filas = tabla.getElementsByTagName("tr");

    for (var i = 0; i < filas.length; i++) {
        var celdaUsuario = filas[i].getElementsByTagName("td")[0];
        if (celdaUsuario) {
            var textoUsuario = celdaUsuario.textContent || celdaUsuario.innerText;
            if (textoUsuario.toUpperCase().indexOf(filtro) > -1) {
                filas[i].style.display = "";
            } else {
                filas[i].style.display = "none";
            }
        }
    }
}
```

Imagen 21 - buscarUsuario JS

La función definida en Imagen 21 es llamada por el evento *onkeyup* de la barra de búsqueda, por lo que cada vez que se escriba una letra, la tabla se actualizará.

5.2.3.3. Static Files

Las interfaces implementadas en HTML en muchos casos utilizan imágenes para utilizarlas como botones. Para este tipo de casos, existe una ruta dentro del proyecto en la cual se almacenan estas imágenes, la car-

peta static_files. La API de flask permite el acceso a este tipo de documentos a través de un endpoint específico.

```python
@app.route('/static/<path:filename>')
def serve_static(filename):

    """
    Define un endpoint para el acceso a archivos estáticos del sistema, como imágenes
    :filename nombre del archivo al que se quiere acceder
    """
    root_dir = os.path.dirname(os.getcwd())
    return send_from_directory(os.path.join(root_dir, 'static'), filename)
```

Imagen 22 - Endpoint para el acceso a documentos estáticos (static)

A este endpoint definido en la API es necesario pasarle como parámetro el nombre del archivo que quiere ser utilizado. La función utiliza la ruta del propio script y le agrega la ruta relativa /static/filename para acceder al documento concreto. Es utilizado en casi todas las interfaces y, como ejemplo en Imagen 20, donde el botón de la parte superior izquierda lo utiliza para recuperar la imagen de la flecha.

```html
<button class="back-button" >
    <img src="{{ url_for('serve_static', filename='PNG/volver.png') }}" alt="Imagen" class="image-style" />
</button>
```

Imagen 23 - Acceso al endpoint acceso

De esta forma se asigna la imagen de la flecha al botón de la parte superior izquierda.

5.2.4. Módulo de acceso a la base de datos

En este módulo se implementan todas las funciones de acceso a la base de datos utilizando mysql.connector. En este módulo se definen funciones para satisfacer todas las posibles consultas asociadas a las funcionalidades del sistema.

Para poder acceder al servidor de MariaDB, se tienen que especificar los parámetros de acceso al mismo.

```python
db_config = {
    'host': '192.168.1.31',
    'user': 'root',
    'password': '16112000',
    'database': 'viviendasaccesibles',
    'port' : '3300'
}
```

Imagen 24 - Parámetros de acceso a la base de datos

Una vez creada la variable con los parámetros de acceso como se muestra en Imagen 24, solo hay que crear una conexión con la base de datos y un cursor que navegue a través de esa conexión. Una vez se ha creado correctamente el cursor, simplemente hay que utilizarlo escribiendo una consulta normal de SQL. A continuación, un ejemplo:

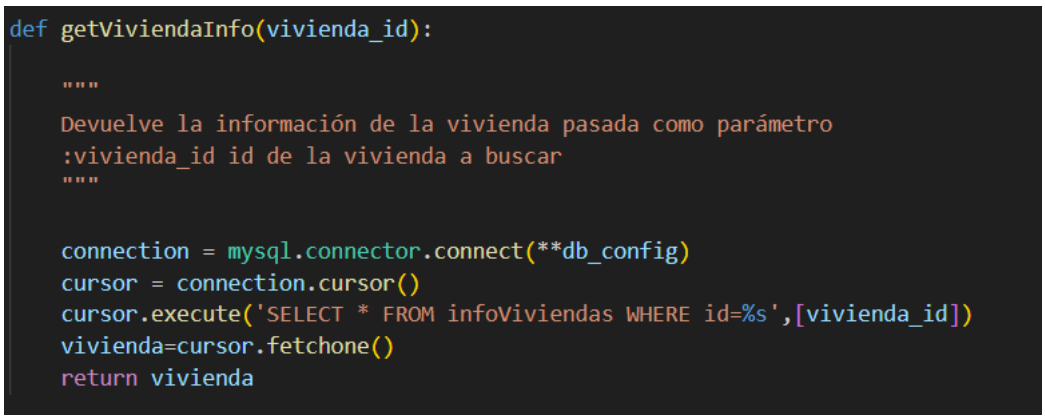


Imagen 25 - Ejemplo acceso base de datos

En el caso concreto del ejemplo mostrado en Imagen 25, se consulta la información de una vivienda pasado el id. Es por ello que solo se recupera una tupla mediante cursor.fetchone(). En los casos que pueden devolverse más de una tupla, se utiliza cursor.fetchall().

La mayoría de las funciones definidas en el módulo son ejecutadas mediante interacciones del usuario con la interfaz, que tras ser manejadas por la API-REST de flask, llegan a la base de datos. Entre estas funciones destacan las consultas de las viviendas, inicio de sesión, registro, listas de viviendas favoritas, información de las viviendas... Todas estas funciones siguen una estructura parecida a la mostrada en Imagen 25.

Además de este tipo de funciones, que son la mayoría del script, también se ha incluido una función de acceso a la base de datos para la araña Scrapy, por el mismo motivo, para separar la estructura interna de la base de datos de la recolección de la información. La función definida para la araña Scrapy es viviendaSpider.

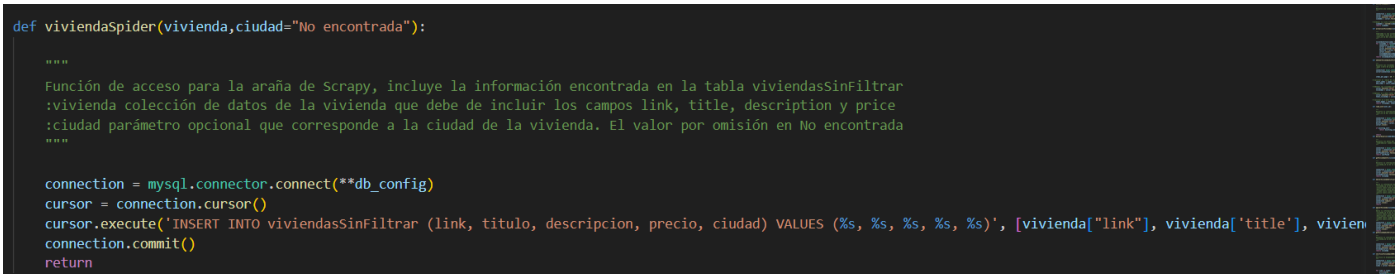
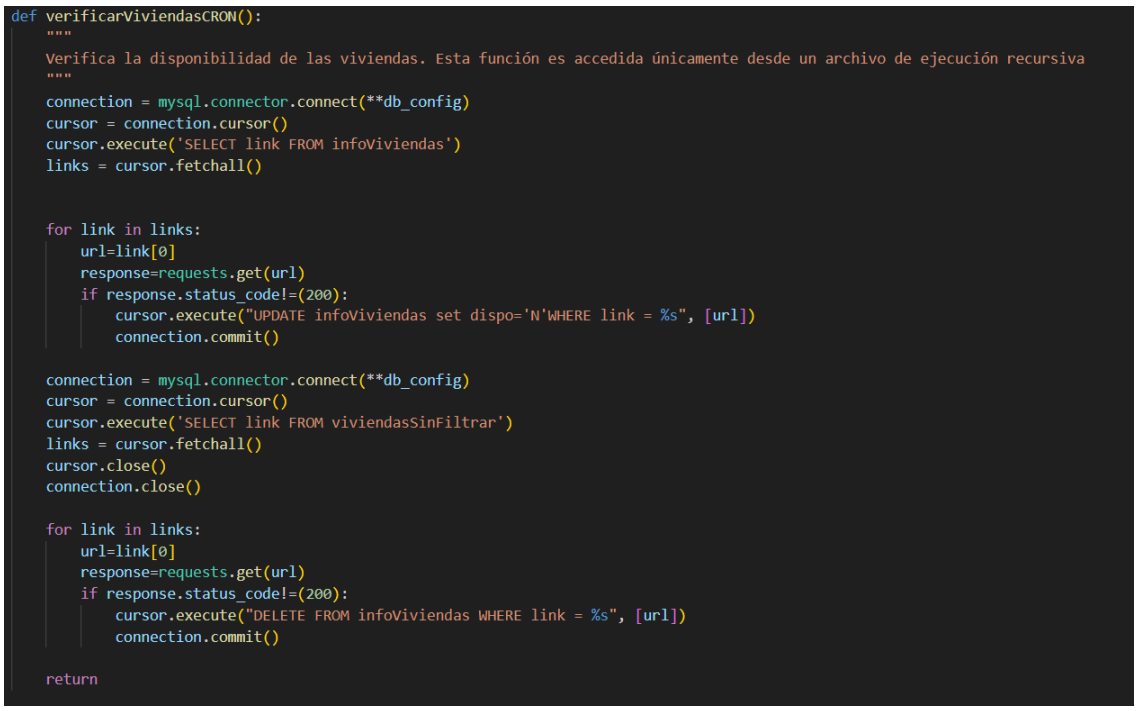


Imagen 26 - Función viviendaSpider

En la función definida en Imagen 26, se requieren dos parámetros, la información de la vivienda, donde vienen incluidos link, título, descripción y precio, y un parámetro opcional de ciudad, cuyo valor por omisión es «No encontrada». Con la sentencia *connection.commit()* se aplican los cambios realizados en la base de datos, concretamente en esta función la adición de una nueva vivienda a la tabla viviendasSinFiltrar.

También se define una función para el acceso recursivo de un script para verificar las viviendas. Más adelante se explicará cómo se utiliza este script, pero en resumen ejecuta esta función cada cierto tiempo.

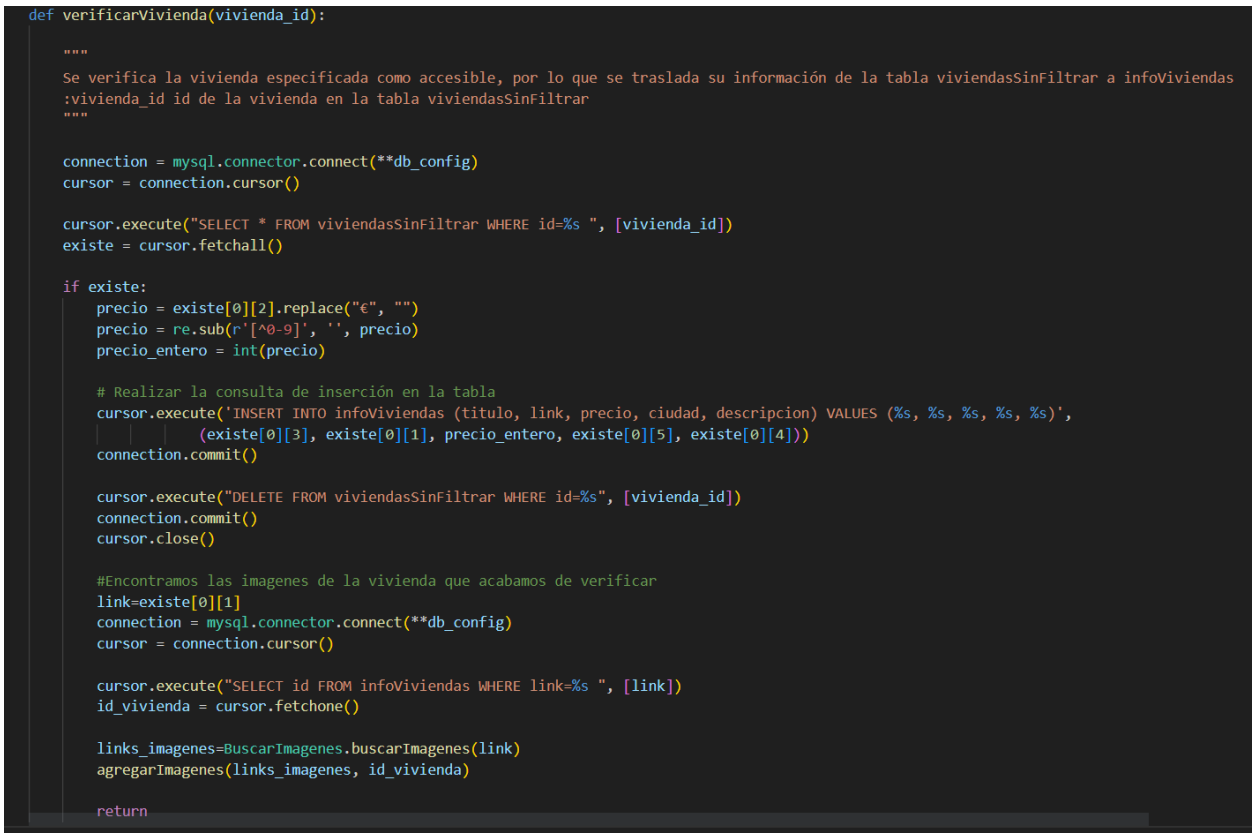


```python
def verificarViviendasCRON():
    """
    Verifica la disponibilidad de las viviendas. Esta función es accedida únicamente desde un archivo de ejecución recursiva
    """
    connection = mysql.connector.connect(**db_config)
    cursor = connection.cursor()
    cursor.execute('SELECT link FROM infoViviendas')
    links = cursor.fetchall()


    for link in links:
        url=link[0]
        response=requests.get(url)
        if response.status_code!=(200):
            cursor.execute("UPDATE infoViviendas set dispo='N'WHERE link = %s", [url])
            connection.commit()

    connection = mysql.connector.connect(**db_config)
    cursor = connection.cursor()
    cursor.execute('SELECT link FROM viviendasSinFiltrar')
    links = cursor.fetchall()
    cursor.close()
    connection.close()

    for link in links:
        url=link[0]
        response=requests.get(url)
        if response.status_code!=(200):
            cursor.execute("DELETE FROM infoViviendas WHERE link = %s", [url])
            connection.commit()

    return
```

Imagen 27 - Función verificarViviendasCRON

En la función verificarViviendasCRON, se visitan todos las URL de las viviendas almacenadas en las tablas de las bases de datos viviendasSinFiltrar e infoViviendas. Lo que busca es conocer si la web está disponible o no. Si lo está no hace nada. En caso de que no esté disponible, si es una vivienda ya verificada se cambia el valor del campo dispo a N, indicando que no está disponible. En caso de que no esté verificada, la borra directamente de la tabla.

Para no hacer la memoria muy repetitiva, no voy a mostrar detalladamente la implementación de cada función ya que todas siguen la misma estructura y no hay mucho que añadir. Procedo a enseñar las dos funciones que más se diferencia del resto, verificarVivienda y mostrarViviendas.



```python
def verificarVivienda(vivienda_id):

    """
    Se verifica la vivienda especificada como accesible, por lo que se traslada su información de la tabla viviendasSinFiltrar a infoViviendas
    :vivienda_id id de la vivienda en la tabla viviendasSinFiltrar
    """

    connection = mysql.connector.connect(**db_config)
    cursor = connection.cursor()

    cursor.execute("SELECT * FROM viviendasSinFiltrar WHERE id=%s ", [vivienda_id])
    existe = cursor.fetchall()

    if existe:
        precio = existe[0][2].replace("€", "")
        precio = re.sub(r'[^0-9]', '', precio)
        precio_entero = int(precio)

        # Realizar la consulta de inserción en la tabla
        cursor.execute('INSERT INTO infoViviendas (titulo, link, precio, ciudad, descripcion) VALUES (%s, %s, %s, %s, %s)',
                       (existe[0][3], existe[0][1], precio_entero, existe[0][5], existe[0][4]))
        connection.commit()

        cursor.execute("DELETE FROM viviendasSinFiltrar WHERE id=%s", [vivienda_id])
        connection.commit()
        cursor.close()

        #Encontramos las imagenes de la vivienda que acabamos de verificar
        link=existe[0][1]
        connection = mysql.connector.connect(**db_config)
        cursor = connection.cursor()

        cursor.execute("SELECT id FROM infoViviendas WHERE link=%s ", [link])
        id_vivienda = cursor.fetchone()

        links_imagenes=BuscarImagenes.buscarImagenes(link)
        agregarImagenes(links_imagenes, id_vivienda)

        return
```

Imagen 28 - Función verificarVivienda

La función verificarVivienda es ejecutada cuando un administrador verifica como accesible o adaptada una de las viviendas que están en la tabla viviendasSinFiltrar. Cuando esto ocurre, la función, que recibe el id de la vivienda en la tabla viviendasSinFiltrar mediante el parámetro de entrada vivienda_id, incluye la información de la vivienda en la tabla infoViviendas, y la borra de la tabla viviendasSinVerificar.

Después, obtiene su nuevo id en la tabla infoViviendas, ya que es necesario para la función buscarImagenes. Esta función, ubicada en un script auxiliar llamado BuscarImagenes, busca las imágenes a través de la URL de la vivienda. Este proceso es muy similar al realizado por Scrapy, pero se realiza ahora ya que es necesario saber el id de la vivienda en la tabla infoViviendas para agregarlo a la tabla imagenes, por la estructura de la misma. En esta tabla se especifica que el campo vivienda_id es una clave externa de la tabla infoViviendas, por lo que es necesario que la vivienda esté en esta clase.

También podría haberse creado otra tabla, para las imágenes de las viviendas sin verificar, y trasladar este proceso de búsqueda de a la araña Scrapy. Ambos enfoques parecen válidos, pero se ha considerado usar la opción de una sola tabla ya que las viviendas sin verificar no tienen por qué ser accesibles, por lo que buscar las imágenes de estas viviendas previo a su confirmación podría llevar a almacenar información inservible.

```python
def obtenerViviendas(page,precio_min, precio_max, ciudad):

    """
    Se devuelven todas las viviendas que pasen los filtros pasados
    :page número de paginación, usado en la interfaz
    :precio_min precio mínimo de las viviendas
    :precio_max precio máximo de las viviendas
    :ciudad ciudad donde buscar las viviendas
    """

    connection = mysql.connector.connect(**db_config)
    cursor = connection.cursor()


    # Consulta SQL para obtener las viviendas filtradas por preci
    query = "SELECT * FROM infoviviendas WHERE dispo='s'"
    params = []
    if precio_min:
        query += " AND precio >= %s"
        params.append(precio_min)
    if precio_max:
        query += " AND precio <= %s"
        params.append(precio_max)
    if ciudad:
        query += " AND ciudad = %s"
        params.append(ciudad)

    # Obtener el número total de viviendas
    cursor.execute(query, params)
    viviendas = cursor.fetchall()  # Leer completamente los resul

    total_viviendas = len(viviendas)
```

Imagen 29 - Función obtenerViviendas (I)

```python
# Obtener el número total de viviendas
cursor.execute(query, params)
viviendas = cursor.fetchall()  # Leer completamente los resultados

total_viviendas = len(viviendas)

# Calcular el número de páginas
viviendas_por_pagina = 20
total_paginas = (total_viviendas + viviendas_por_pagina - 1) // viviendas_por_pagina

# Calcular el desplazamiento para la paginación
offset = (page - 1) * viviendas_por_pagina

# Consulta SQL para obtener las viviendas paginadas
query += " LIMIT %s, %s"
params.extend([offset, viviendas_por_pagina])
cursor.execute(query, params)
viviendas = cursor.fetchall()
return viviendas, total_paginas
```

Imagen 30 - Función obtenerViviendas (II)

La función obtenerViviendas tiene 4 parámetros, los cuales son parámetros para ajustar la búsqueda, salvo en el caso del parámetro page. En la interfaz, las viviendas se muestran en páginas de 20 viviendas. El parámetro page representa la página de la cual se debe pasar información. La variable *query* representa la sentencia que se va a ejecutar, y params representan los parámetros para las mismas. Teniendo en cuenta

los valores de los parámetros precio_min, precio_max y ciudad se conforma la sentencia definitiva. Además, solo se muestran aquellas viviendas cuya disponibilidad sea s.

Después de ejecutar la sentencia, se calcula el número de viviendas que han satisfecho la consulta. Esto se realiza para calcular el número de páginas que va a haber. Más tarde ,utilizando el parámetro de page, se limitan los resultados de la consulta a los que le corresponde a la página

5.3. Despliegue

Para utilizar Docker, es necesario crear un archivo Dockerfile en el que se especifique las características del que se quiere crear la imagen. Para ello se ha guardado todo lo necesario para el funcionamiento de la aplicación en una carpeta app. Para generar la imagen, el contenido de Dockerfile debe ser el siguiente:

```
# Establece la imagen base de Python
FROM python:3.9

# Establece el directorio de trabajo dentro del contenedor
WORKDIR /app

# Copia todos los archivos de la aplicación al contenedor
COPY ./app .

# Instala las dependencias
RUN pip install --no-cache-dir -r requirements.txt

# Copia el archivo cronjob al contenedor
COPY cronjob /etc/cron.d/cronjob

# Otorga los permisos adecuados al archivo cronjob
RUN chmod 0644 /etc/cron.d/cronjob

# Establece el comando para ejecutar el cron y la aplicación Flask
CMD service cron start && python mostrarViviendas.py
```

Imagen 31 - Dockerfile

Donde:

- ./app es la carpeta creada para almacenar el proyecto
- requirements.txt es el documento que indica las bibliotecas que debe de instalarse para el correcto funcionamiento de la aplicación. Se deben incluir todas las bibliotecas explicadas a lo largo de la memoria.
- cronjob es el documento mostrado en Imagen 8
- mostrarViviendas.py es el archivo donde se define la API de flask, explicada en el módulo de gestión.

Utilizando este DockerFile se puede crear la imagen para poder ejecutar el programa en un contenedor Docker. Esto se consigue gracias a la orden:

dockerbuild -t viviendasaccesibles

Donde viviendasaccesibles es el nombre que se le da a la imagen y . representa la ruta donde se encuentra Dockerfile. Esta orden es ejecutada en el mismo directorio donde se encuentra Dockerfile. De esta forma, se crea la imagen de Docker, que puede ser exportada mediante zip o tar. Para ejecutar la aplicación en un contenedor *software* de Docker, se debe ejecutar la orden:

docker run -p 5000:5000 viviendasaccesibles

Esta orden ejecuta un contenedor utilizando la imagen viviendasaccesibles y mapea el puerto 5000 del contenedor al puerto 5000 del sistema operativo host. Este puerto es el que utiliza el servidor WSGI de prueba de flask.
Para exportar esta imagen, se puede comprimir en un archivo .tar. Para ello se debe de ejecutar la orden:

dockersave -o viviendasaccesibles.tar viviendasaccesibles

Donde viviendasaccesibles.tar es la imagen comprimida de la aplicación. Para utilizarla solo se debe usar la orden load de manera similar a como se ha hecho con sabe

docker load -i viviendasaccesibles.tar

Después de importarla, se ejecuta con la orden run de docker, como se mostró anteriormente.

5.4. Sistema Final

En este apartado se procede a mostrar algunas de las interfaces finales de la aplicación, relacionadas con las funcionalidades y las herramientas descritas a lo largo de este documento. Para una explicación más detallada de las interfaces finales de la aplicación, consulte el Anexo V – Manual de Usuario.

Imagen 32 - Interfaz Principal

En Imagen 32 se muestra la pantalla principal de la aplicación. En ella aparecen en la parte central 3 campos para introducir filtros en la búsqueda de viviendas. Para realizar la búsqueda aplicando los filtros escritos se debe pulsar el botón de la lupa. En la parte superior derecha, aparecen dos botones para registrar un nuevo usuario o para iniciar sesión. Al pulsar en el botón de la lupa se ejecuta la búsqueda con los valores deseados, en ese caso ninguno.

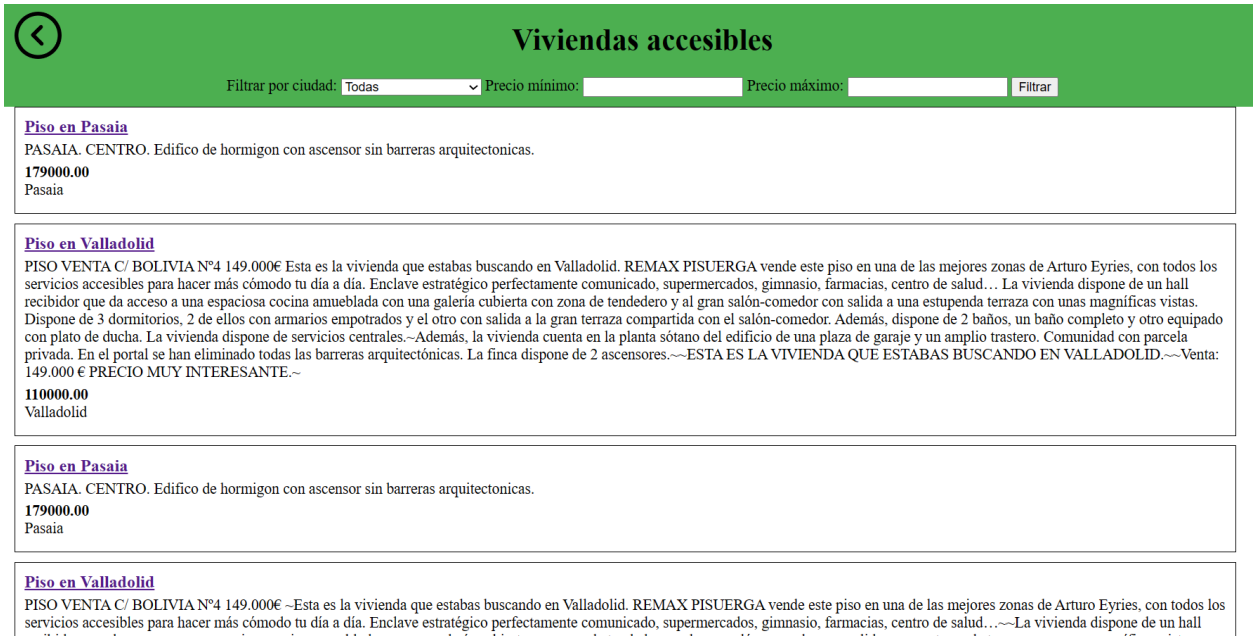


Imagen 33 - Resultados de la búsqueda

Se redirige al usuario a Imagen 33 donde se muestran todas las viviendas almacenadas en la base de datos que se ajusten a los filtros introducidos, como se ha dicho, en este caso sin filtros. En esta página se puede

ver las descripciones, precios y ciudades de las viviendas que han pasado los filtros. Pulsando sobre cualquiera de los títulos de las viviendas, se redirige al usuario a la página individual de la vivienda.

En la parte superior se encuentra una barra en la que podemos ajustar de nuevo los filtros, además de un botón para volver a la página principal de la aplicación.

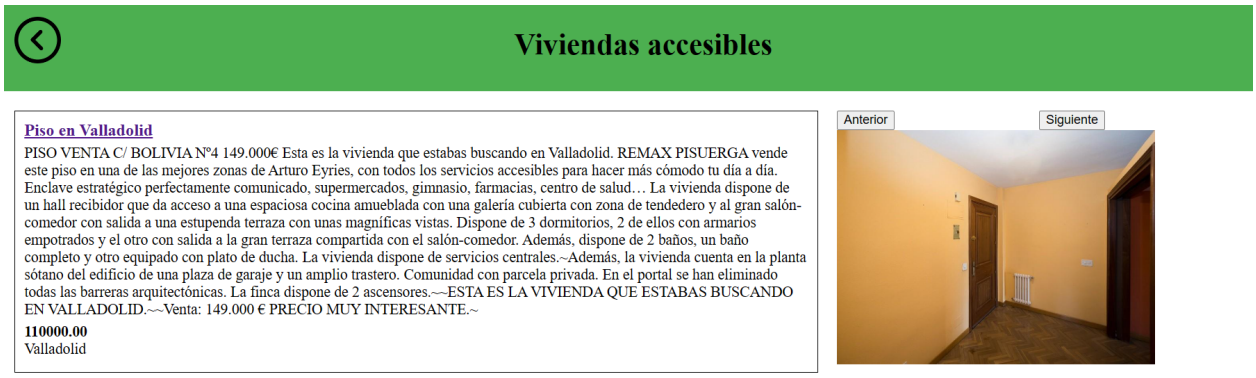
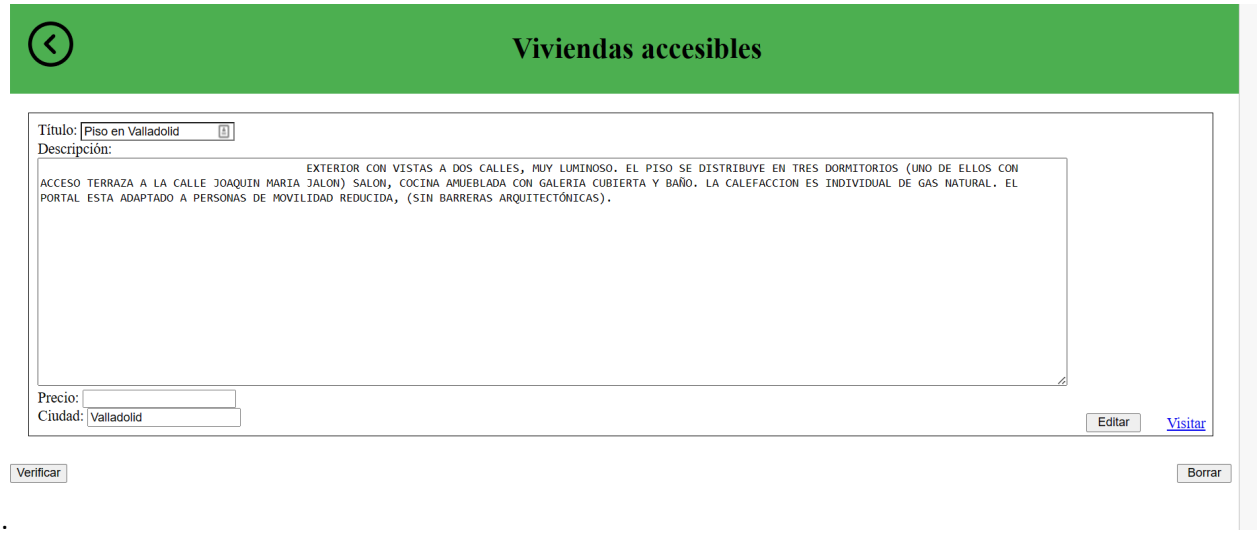


Imagen 34 - Información Vivienda

En la página individual de la vivienda se pueden ver los mismos datos que aparecían en la búsqueda general, además de un carrusel de imágenes por las que se pueden navegar a través de los botones colocados encima de las mismas. Al pulsar de nuevo sobre el título de la vivienda, se redirige a la página original donde fue encontrada la vivienda , donde podrá encontrar más datos de la vivienda. Al pulsar el botón de la esquina superior izquierda, se redirige al usuario a la página de búsqueda.



.

Imagen 35 - Información vivienda sin verificar

Los administradores pueden editar la información que aparece en los campos de las viviendas. En el caso de Imagen 35 ,se muestra la página individual de una vivienda sin verificar, pero para las viviendas verificadas es similar. En la interfaz de individual de las viviendas aparecerán los campos de información de la misma,

los cuales pueden ser editados. Para editar la información de una vivienda solo se deben cambiar los datos escritos y darle a editar. Al darle a editar se actualizarán los valores de la vivienda con los valores que aparezcan en los campos. Existe un botón de verificar, para verificar la vivienda e incluirla con el resto de viviendas en las búsquedas estándar, y un botón de borrar para eliminarla del sistema. El botón visitar situado a la derecha del botón editar permite visitar la página original de la vivienda. Al verificar una vivienda, se buscarán automáticamente las imágenes en la web original.

Aunque esta página corresponde a una vivienda sin verificar, los usuarios administradores tienen una página similar a esta para las viviendas ya verificadas, pudiendo editar la información de las viviendas siempre que sea necesario.

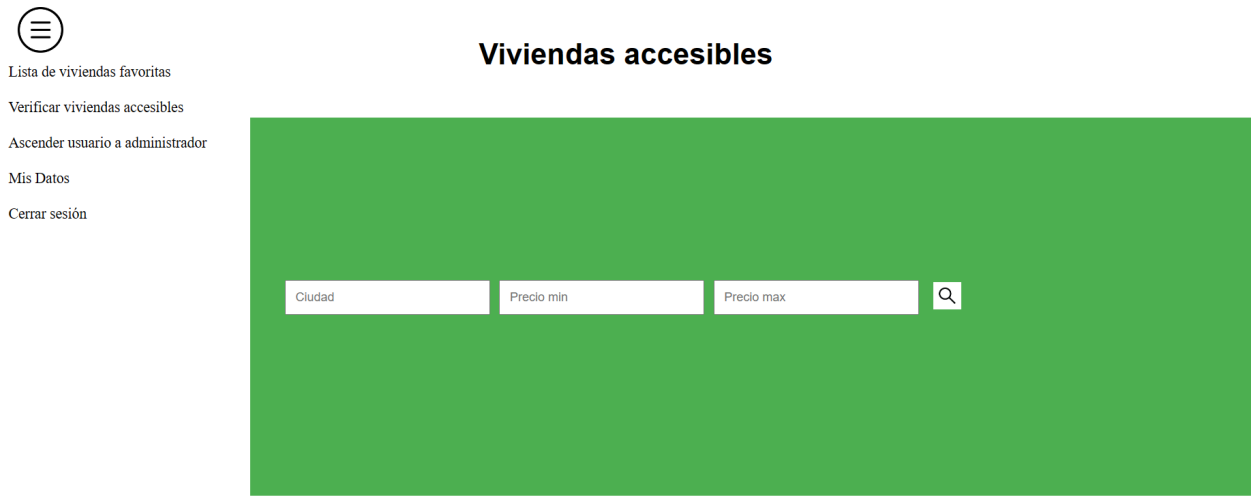


Imagen 36 - Interfaz principal administrador

En Imagen 36 se muestra la interfaz principal de un usuario administrador, donde se puede ver que es muy similar a la interfaz principal, pero eliminando las opciones para iniciar sesión y registrarse y agregando un menú con las opciones propias de un administrador. Para los usuarios registrados la interfaz es idéntica, cambiando las opciones del menú por las propias de un usuario registrado

5.5. Pruebas realizadas

Para el correcto desarrollo del sistema, se han realizado principalmente dos conjuntos de pruebas. El primer conjunto de pruebas realizado se corresponde con las pruebas realizadas a usuarios para ver si comprendían las interfaces de la página web y sabían acceder a las todas las funcionalidades de la misma. El otro conjunto de pruebas, están relacionados con la seguridad del sistema. Concretamente, con el acceso a información sensible a través del manejo de URL.

5.5.1. Pruebas a usuarios reales

Como se ha comentado anteriormente, en este conjunto de pruebas se pide a los usuarios que realicen determinadas tareas para verificar la facilidad de uso y entendimiento de las interfaces. Se han realizado pruebas a un grupo de 6 usuarios, entre los que se incluyen personas jóvenes (20-22 años), personas de mediana edad (47 años) y personas de la avanzada edad (75 años). En el caso de los usuarios jóvenes, como es obvio, han tenido gran relación con internet y utilizan una gran cantidad de páginas web diariamente. Es de esperar que estos usuarios sean capaces de entender todas las interfaces sin ningún problema y, además, es posible que aporten información importante para realizar cambios en las interfaces.

Los usuarios más mayores, no han tenido mucha relación con el mundo de internet, en especial el usuario de avanzada edad, por lo que las pruebas realizadas a estos usuarios son claves para saber si las interfaces del sistema son fáciles de usar y entendibles.

En las pruebas realizadas, todos los usuarios fueron capaces de realizar con éxito las tareas que se le pidieron. Tanto registrarse en el sistema, como buscar y filtrar viviendas, añadirlas a favoritas, ver la lista de favoritas, ver los datos de usuario y navegar por la aplicación utilizando los menús del sistema. Esto es especialmente positivo en el caso de los usuarios mayores, ya que es muy importante que fueran capaces de hacerlo.

Realizando estas pruebas, algunos de los usuarios, concretamente los jóvenes, propusieron algunos cambios que mejoran la experiencia de usuario, como mantener los filtros cuando se realiza una búsqueda o que te redirija a la página de inicio de sesión tras finalizar el registro.

En resumen, los resultados de las pruebas a usuarios son muy satisfactorios, de los cuales se entienden que la aplicación es fácil de usar y tiene una interfaz poco compleja.

5.5.2. Pruebas de seguridad

Como se ha comentado antes, las pruebas de seguridad realizadas en el sistema se basan en restringir el acceso a información sensible o a funcionalidades prohibidas mediante la manipulación de la URL.

Para realizar las pruebas, primero se ha intentado visitar las funcionalidades de usuario registrado sin registrarme anteriormente en el sistema. En este caso no ha surgido ningún problema, ya que mediante el *decorator* @login_required de flask es muy sencillo evitar estos accesos. Este *decorator* ya se había implementado previo al inicio de las pruebas, por lo que este primer conjunto de pruebas.

A continuación, se ha intentado acceder a funcionalidades exclusivas de administrador estando registrado como un usuario normal. En este caso tampoco han sufrido ningún problema, ya que en este tipo de funcionalidades se comprueba el rol del usuario que intenta acceder a la funcionalidad. Si no es administrador, deniega el acceso.

El último conjunto de pruebas se correspondía con intentar acceder a información de otros usuarios siendo un usuario normal. Esto debería de estar prohibido y solo permitido para administradores en algunos casos concretos. En estas pruebas si surgieron algunos problemas, ya que al introducir la url:

/verDatos/usuario

El usuario registrado era capaz de acceder a información sensible de otros usuarios. Es por ello que se ha modificado la URI para acceder a este endpoint, eliminando el parámetro usuario y mostrando la información del usuario que accede al endpoint.

En resumen, estas pruebas han permitido mejorar algo la seguridad de la aplicación, por lo menos en primera instancia, aunque el planteamiento inicial de la aplicación era bastante seguro, al menos en este aspecto.

6. Conclusiones

Este es el apartado final del documento. Se va a explicar cómo el sistema cumple con los requisitos planteados por el cliente con las funcionalidades implementadas. Además, se dará alguna idea de trabajo a futuro para mejorar la aplicación. Para poder entender la necesidad del proyecto y cómo este se diferencia de otros proyectos similares hay que destacar que existen muchos buscadores de viviendas que aportan una funcionalidad similar a la de la aplicación, como los mencionados anteriormente fotocasa e idealista, o la página de donde se ha extraído la información, todopisos. Estos buscadores permiten el filtrado de viviendas en función de unos parámetros. La gran diferencia de los buscadores ya existentes y el buscador desarrollado en este proyecto es la forma de extracción de la información. Mientras que todos los buscadores mencionados obtienen la información de las viviendas mediante los propios anunciantes de las mismas, mediante formularios, la aplicación correspondiente a este proyecto encuentra la información utilizando un sistema de web scraping.

En muchos de los buscadores no existe una opción que permita filtrar las viviendas en función de si están adaptadas o no. La única web de la cual se tiene conocimiento que aporta una funcionalidad similar a la implementada en la aplicación es la web idealista, que permite la búsqueda en una selección de viviendas que tienen marcadas como accesibles. De nuevo, la gran diferencia con la aplicación desarrollada es el método de obtención de la información. En este caso, mientras que idealista muestra como accesibles aquellas viviendas que los anunciantes han indicado en el formulario que son accesibles, la aplicación desarrollada encuentra las viviendas accesibles mediante el procesamiento de lenguaje natural, lo que potencialmente podría suponer un aumento en el número de viviendas consideradas accesibles. Es difícil calcular el aumento de viviendas, ya que realizar pruebas en la web idealista siguiendo el esquema utilizado por la aplicación es imposible, debido a la prohibición de procesos de crawling definidos por la página.

Para el desarrollo de este proyecto no se ha tenido constancia de ningún otro buscador que aporte una funcionalidad similar relacionada con las viviendas accesibles. Existen algunas web que intentan ofrecer una funcionalidad similar, mostrando anuncios de viviendas accesibles, pero ninguna de ellas tienen un volumen de viviendas que pueda ser utilizado como referencia, ya que la mayoría de las páginas disponen de menos de 20 anuncios.

6.1. Funcionalidad del sistema

Con las funcionalidades explicadas a lo largo del documento se puede llegar a dos conclusiones:

- Los requisitos planteados inicialmente por el cliente han sido satisfechos. La necesidad de encontrar viviendas adaptadas se ha cumplido con creces con el sistema scraper y el filtro implementado. La gran

mayoría de las viviendas que pasan el filtro son realmente accesibles, o al menos, así lo indican en la descripción de alguna forma. Concretamente, tras una revisión de 125 viviendas realizada por el desarrollador, únicamente se encontró una vivienda que no fuera accesible. Aunque es una muestra amplia y permite decir que ha sido un éxito, convendría revisar todas las viviendas encontradas por el algoritmo para unos resultados más fiables sobre la precisión del proceso de filtrado. Además, como se pidió, las viviendas deben ser verificadas por los usuarios administradores antes de ser mostradas al resto de usuarios. El sistema de usuarios funciona perfectamente y tiene todos los usuarios que aparecían en las peticiones del cliente. En general, tiene la estructura y la funcionalidad de un sistema de búsqueda de viviendas clásico, pero únicamente mostrando viviendas adaptadas.

- De la misma forma, los requisitos planteados por el cliente eran algo abiertos en cuanto a la implementación de los mismos. Esto ofrece mucha flexibilidad en la implementación, lo cual se ha tenido en cuenta a lo largo del desarrollo del proyecto. En el caso concreto de este proyecto, se ha intentado hacer un sistema lo más sencillo de utilizar posible, descartando en el proceso de desarrollo toda funcionalidad superflua que aporte poco valor al resultado final. Además, la recogida de datos de las viviendas recoge los datos clave de la vivienda, dejando otros datos secundarios e información de contacto con el anunciante delegado en la página original de la vivienda encontrada.

El proyecto proporciona al usuario una experiencia satisfactoria, cumpliendo con las expectativas del mismo y mostrando las viviendas según los parámetros introducidos como filtro. Además, el sistema de viviendas favoritas permite la visualización rápida de aquellas viviendas que realmente interesen al usuario. El sistema implementado para evitar mostrar viviendas que ya no están disponibles también aporta a lograr una buena experiencia, ya que intentar acceder a una vivienda que no está disponible puede ser un proceso que frustre a los usuarios.

La estructura del proyecto permite ampliar funcionalidades según las necesidades que vayan surgiendo en un futuro desarrollo, por lo que la aplicación de un prototipo incremental se ha considerado exitosa, ya que permite ampliar la funcionalidad a través de varios frentes (consulta de nuevas páginas, funcionalidades de la API, funciones en el módulo de acceso a la base de datos...)

El diseño del sistema aplicando el patrón MVC también ha sido exitoso, ya que la representación de los datos realizada en la vista está totalmente separada de la implementación real de los datos almacenados en el Modelo. El controlador es el que lleva la lógica del negocio y tampoco tiene conocimiento de la estructura interna del módulo de persistencia. Los 3 principales módulos del sistema están separados entre ellos y no tienen información de la estructura interna de los otros módulos.

Se concluye, por lo tanto, que el sistema cumple con todas las necesidades planteadas, eligiendo las diferentes herramientas planteadas a lo largo del documento y dando como resultado final una aplicación dockerizada para ejecutarse en un contenedor de *software*.

6.2. Trabajo a futuro

Debido al límite temporal del proyecto, hay varios aspectos que son funcionales, pero tienen margen de mejora.

El primero punto de mejora es el sistema de despliegue. El modelo elegido permite adaptarse a otros modelos de despliegue que tengan un alcance mayor. Como idea, AWS ofrece ECS (40) para escalar un despliegue basado en contenedores. Es un servicio de orquestación de contenedores totalmente administrado que facilita la implementación, la administración y el escalado de las aplicaciones en contenedores. Es una opción interesante para escalar el proyecto a un contexto empresarial, ya que AWS es uno de los mayores proveedores de servicios en la nube (41).

Aunque la idea de utilizar ECS de AWS es la propuesta por el desarrollador, pero existen otras opciones válidas. Al haber elegido un contenedor de *software* como modelo de despliegue, el proyecto se adapta fácilmente a otros tipos de despliegue.

La estructura del proyecto permite añadir nuevas funcionalidades de una forma sencilla. Por ejemplo, quizá sería conveniente añadir campos para filtrar según la adaptación que tenga la vivienda en concreto. Esta función puede realizarse explorando otras opciones de NLP, como la introducción de términos clave que diferencien cada una de las posibles adaptaciones que puedan tener las viviendas. También se puede entrenar un sistema de clasificadores mediante análisis de datos para que clasifique las viviendas accesibles en función de la adaptación incorporada a la vivienda. La idea es que se pueda ampliar la funcionalidad si así se considera. En todo caso, este proceso conlleva un análisis de datos que se escapa del alcance planteado para el proyecto y queda a elección de futuros desarrolladores.

Además. el diseño de interfaces de la aplicación, como se ha comentado anteriormente en la memoria, ha tenido una prioridad baja en cuanto a estética a lo largo del desarrollo del proyecto, dando como resultado una interfaz funcional y comprensible, pero con gran margen de mejora. Por tanto, el proceso de diseño de interfaces en el proyecto se ha centrado en la simplificación y en la fácil comprensión de la misma, dejando de lado aspectos estéticos que es un claro factor a mejorar en un futuro.

Podría plantearse la idea de crear nuevas arañas Scrapy para realizar la obtención de información de viviendas de otros sitios web. Para poder realizar esto, además de configurar la araña en cuestión, habría que implementar un sistema para detectar viviendas que puedan aparecer en ambos sitios web, evitando la redundancia en la información almacenada en la base de datos.

Estas son las principales rutas que futuros desarrolladores podrían tener en cuenta en un futuro desarrollo del proyecto.

7.Bibliografía

1. *Página web de todopisos*. Recuperado por última vez el 30 de Junio de 2023, de https://www.todopisos.es

2. *Página web del CRMF en Salamanca*. Recuperado por última vez el 30 de Junio de 2023, de https://crmfsalamanca.imserso.es/web/crmf-de-salamanca

3. *Artículo en the balance sobre buscadores de viviendas*. Recuperado por última vez el 30 de Junio de 2023, de https://www.thebalance.com/what-is-a-real-estate-search-engine-4785217

4. *Sitio web de fotocasa*. Recuperado por última vez el 30 de Junio de 2023, de https://www.fotocasa.es

5. *Sitio web de idealista*. Recuperado por última vez el 2023 de Junio de 30, de https://www.idealista.com

6. *Definición de vivienda adaptada por kronoshomes*. Recuperado por última vez el 30 de Junio de 2023, de https://www.kronoshomes.com/es/ideas/que-es-una-vivienda-adaptada/#:~:text=Una%20vivienda%20accesible%20o%20adaptada,en%20cuenta%20sus%20necesidades%20concretas

7. *Porcentaje de viviendas accesibles a lo largo de los años en España*. Recuperado por última vez el 30 de Junio de 2023, de https://www.api.cat/noticias/la-vivienda-en-espana-suspende-en-accesibilidad/

8. *Artículo "What is Web Scraping and How Does Web Crawling Work?" en-PromptCloud*. Recuperado por última vez el 30 de Junio de 2023, de https://www.promptcloud.com/blog/web-scraping-introduction/

9. *Steven Bird, Ewan Klein y Edward Loper*. Recuperado por última vez el 30 de Junio de 2023, de Natural Language Processing with Python: https://www.nltk.org/book

10. *Artículo "A Few Useful Things to Know About Machine Learning" de Pedro Domingos.* . Recuperado por última vez el 30 de Junio de 2023, de https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf

11. *"What are Containers and Why Do You Need Them?" en Docker Documentation*. Recuperado por última vez el 30 de Junio de 2023, de https://docs.docker.com/get-started/overview/

12. *Definición de bases de datos relacionales y sus principios por techopedia*. Recuperado por última vez el 30 de Junio de 2023, de https://www.techopedia.com/definition/1236/relational-database

13. *Documentación oficial de Mozilla Developer Network (MDN) para API* . Recuperado por última vez el 30 de Junio de 2023, de https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction

14. *Documentación oficial de Mozilla Developer Network (MDN) para REST*. Recuperado por última vez el 30 de Junio de 2023, de https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#rest

15. *Blei, D. M., Ng, A. Y., &Jordan, M. I. (2003).* Obtenido de Latent Dirichlet Allocation. Journal of Machine Learning Research, 3, 993-1022. Recuperado por última vez el 30 de Junio de 2023: https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf
16. *Sitio oficial de Visual Studio* . Recuperado por última vez el 30 de Junio de 2023, de https://code.visualstudio.com/
17. *Estudio sobre uso de tecnologías realizado por StackOverflow*. Recuperado por última vez el 30 de Junio de 2023, de https://survey.stackoverflow.co/2023/#technology
18. *Documentación oficial de Pylance*. Recuperado por última vez el 30 de Junio de 2023, de https://pylance-release.walkme.com/docs/
19. *Sitio oficial de Python*. Recuperado por última vez el 30 de Junio de 2023 de https://www.Python.org/
20. *Guía de Google Cloud para el diseño de API* . Recuperado por última vez el 30 de Junio de 2023, de https://cloud.google.com/apis/design
21. *Documentación oficial de Flask*. Recuperado por última vez el 30 de Junio de 2023, de https://flask.palletsprojects.com/
22. *Documentación oficial de requests*. Recuperado por última vez el 30 de Junio de 2023, de https://docs.Python-requests.org/
23. *Documentación oficial de Scrapy*. Recuperado por última vez el 30 de Junio de 2023, de https://docs.scrapy.org/
24. *Documentación oficial de NLTK* . Recuperado por última vez el 30 de Junio de 2023, de https://www.nltk.org/
25. *Documentación oficial de Gensim*. Recuperado por última vez el 30 de Junio de 2023, de https://radimrehurek.com/gensim/
26. *M. Manikandan, R. Pushpa.* Recuperado por última vez el 30 de Junio de 2023, de Topic modeling and sentiment analysis for customer reviews on social media. Estudio que usa gensim y NLTK para el análisis de reseñas: https://www.sciencedirect.com/science/article/pii/S2212017314005369
27. *Repositorio de GitHub de Geonamescache*. Recuperado por última vez el 30 de Junio de 2023, de https://github.com/yaph/geonamescache
28. *Documentación oficial de MariaDB*. Recuperado por última vez el 30 de Junio de 2023, de https://mariadb.com/kb/en/documentation/
29. *Documentación oficial de MySQL*. Recuperado por última vez el 30 de Junio de 2023, de https://dev.mysql.com/doc/
30. *Guía de desarrollador de MySQL Connector*. Recuperado por última vez el 30 de Junio de 2023, de https://dev.mysql.com/doc/connector-Python/en/
31. *Documentación oficial de HTML en Mozilla Developer Network (MDN)* . Recuperado por última vez el 30 de Junio de 2023, de https://developer.mozilla.org/es/docs/Web/HTML

32. *Documentación oficial de CSS en Mozilla Developer Network (MDN)* . Recuperado por última vez el 30 de Junio de 2023, de https://developer.mozilla.org/es/docs/Web/CSS
33. *Documentación oficial de JavaScript en Mozilla Developer Network (MDN)* . Recuperado por última vez el 30 de Junio de 2023 de https://developer.mozilla.org/es/docs/Web/JavaScript
34. *Documentación oficial de VirtualBox*. Recuperado por última vez el 30 de Junio de 2023, de https://www.virtualbox.org/wiki/Documentation
35. *Documentación oficial de Ubuntu* . Recuperado por última vez el 30 de Junio de 2023, de https://ubuntu.com/documentation
36. *Manual de cron en sistemas Linux*. Recuperado por última vez el 30 de Junio de 2023, de https://linux.die.net/man/8/cron
37. *Documentación Oficial de Docker*. Recuperado por última vez el 30 de Junio de 2023, de https://docs.docker.com/
38. *Scrum Guide*. Recuperado el 30 de Junio de 2023, de https://www.scrumguides.org/
39. *Model-View-Controller (MVC) Pattern" en Microsoft*. Recuperado por última vez el 30 de Junio de 2023, de https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649643(v=pandp.10)
40. *Sitio oficial de ECS por AWS*. Recuperado por última vez el 30 de Junio de 2023, de https://aws.amazon.com/es/ecs/
41. *Artículo sobre cuota de mercado en alojamiento web*. Recuperado por última vez el 30 de Junio de 2023, de https://kinsta.com/es/cuota-de-mercado-de-aws/
/