



Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**



Grado en Ingeniería Informática

Trabajo Fin de Grado

Aplicación Web para Marketplace de Permutas

Autor: Marta Corrochano Garrido

Tutor(a): Julio Mariño Carballo

Madrid, Junio 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Aplicación Web para Marketplace de permutas Circulares

Mayo 2022

Autor: Marta Corrochano Garrido

Tutor:

Julio Mariño Carballo

Lenguajes y Sistemas Informáticos

ETSI Informáticos

Universidad Politécnica de Madrid

Resumen

El proyecto que se presenta en esta memoria tiene como objetivo explicar en detalle el desarrollo de una aplicación web que actúa como “marketplace” para la realización de permutas inmobiliarias. Se trata así de una plataforma digital que ejerce de intermediadora entre usuarios que deseen intercambiar sus propiedades de manera permanente. En concreto se ha llevado a cabo el desarrollo del software “back-end” de la aplicación. Por lo que en esta memoria se encuentra el diseño del modelo de datos, la creación de la base de datos, además de la API que cumple con los requisitos encontrados.

Para poder entender la funcionalidad de la aplicación, se ha realizado un análisis del sector inmobiliario y se describe el modelo de negocio elegido para desarrollar la actividad de permuta.

Al final del proyecto se han adjuntado las pruebas realizadas para la comprobación del cumplimiento de los requisitos especificados.

Abstract

The project presented in this dissertation aims to explain in detail the development of a web application that acts as a marketplace for real estate swaps. It is thus a digital platform that behaves as an intermediary between users who wish to exchange their properties on a permanent basis. Specifically, it has been carried out the development of the application back-end software. That is why in this report it is found the design of the database as well as the API that meets the requirements.

In order to understand the functionality of the application, an analysis of the real estate sector in these days has been done and the business model chosen is described.

At the end of the project, the tests performed to verify compliance with the specific requirements have been attached.

Tabla de contenidos

1	<i>Introducción</i>	1
1.1	Propósito y contextualización del tema	1
1.2	Objetivos	2
1.3	Estructura de la Memoria	2
1.4	Modelo de negocio	3
1.5	Estado del arte	8
2	<i>Análisis</i>	10
2.1	Casos de uso	10
2.2	Especificación de requisitos	10
3	<i>Diseño</i>	12
3.1	Diagrama entidad relación	12
3.2	Tablas y Atributos	12
3.3	Validación del diseño	17
3.4	Lógica del diseño	21
4	<i>Implementación</i>	23
4.1	Tecnologías empleadas	23
4.2	Desarrollo	25
4.2.1	Base de datos	25
4.2.2	API REST	27
4.2.2.1	Diagrama UML	31
4.2.2.2	Verbos utilizados	32
4.2.2.3	Códigos http de respuesta	32
4.2.2.4	Identificación de URI y patrones de identificación de recursos por requisito	33
5	<i>Pruebas</i>	45
6	<i>Conclusiones y trabajos futuros</i>	69
6.1	Futuras ampliaciones	69
6.2	Conclusiones	69
7	<i>Análisis de Impacto</i>	70
8	<i>Bibliografía</i>	71

Tabla de Ilustraciones

Ilustración 1. Evolución de las permutas de viviendas en España. Fuente: datos del INE [18]	4
Ilustración 2. Generación de Tabla usuario en Workbench.	13
Ilustración 3. Ejemplo de Tabla usuario en Workbench.	14
Ilustración 4. Generación de Tabla inmueble en Workbench.	14
Ilustración 5. Ejemplo de Tabla inmueble en Workbench.	14
Ilustración 6. Generación de Tabla favoritos en Workbench.	15
Ilustración 7. Ejemplo de Tabla favoritos en Workbench.	15
Ilustración 8. Generación de Tabla permuta en Workbench.	16
Ilustración 9. Ejemplo de Tabla permuta en Workbench.	16
Ilustración 10. Generación de Tabla mensajes en Workbench.	16
Ilustración 12. Ejemplo de Tabla mensajes en Workbench.	17
Ilustración 35. Dependencias Maven.	24
Ilustración 36. Creación de un Modelo de Datos.	25
Ilustración 37. Especificación de metadatos.	25
Ilustración 38. Modelo EER.	26
Ilustración 39. Conexión MySQL Workbench.	26
Ilustración 40. Script del Modelo de Datos en la conexión.	27
Ilustración 41. Definición de un nuevo servidor en Eclipse.	28
Ilustración 42. Modificación en los recursos configurados del servidor.	28
Ilustración 43. Conexión a la base de datos.	29
Ilustración 44. Directorio del proyecto en Eclipse.	30
Ilustración 45. Prueba de registro.	45
Ilustración 46. Tabla usuario tras registrar el primer usuario.	46
Ilustración 47. Tabla inmueble tras registrar el primer usuario.	46
Ilustración 48. Tabla inmueble.	47
Ilustración 49. Prueba errónea de registro.	47
Ilustración 50. Prueba de login.	48
Ilustración 51. Prueba de login errónea.	48
Ilustración 52. Prueba de consulta de datos de usuario.	49
Ilustración 53. Prueba errónea de consulta de datos de usuario.	49
Ilustración 54. Prueba de modificación de datos de usuario.	50
Ilustración 55. Tabla usuario tras la modificación del usuario 1.	50
Ilustración 56. Prueba errónea de modificación de datos de usuario.	50
Ilustración 57. Prueba de registro de un inmueble.	51
Ilustración 58. Tabla inmueble con un nuevo inmueble.	51
Ilustración 59. Prueba errónea de registro de inmueble.	51
Ilustración 60. Prueba para visualizar las propiedades de un usuario.	52
Ilustración 61. Prueba errónea para visualizar las propiedades de un usuario.	53
Ilustración 62. Prueba para realizar un búsqueda con filtro.	54
Ilustración 63. Prueba para añadir un inmueble a la lista de favoritos.	54
Ilustración 64. Tabla favoritos tras añadir varios inmuebles a las listas.	54
Ilustración 65. Prueba errónea para añadir un inmueble a la lista de favoritos.	55
Ilustración 66. Prueba de consulta de lista de favoritos de un usuario.	56
Ilustración 67. Prueba errónea de consulta de lista de favoritos de un usuario.	56
Ilustración 68. Prueba para generación de una permuta.	57
Ilustración 69. Tabla permuta tras generar una nueva permuta.	57
Ilustración 70. Prueba errónea de generación de permuta.	57
Ilustración 71. Prueba para almacenar mensajes de una permuta.	58

Ilustración 72. Tabla mensajes con los mensajes almacenados.	58
Ilustración 73. Prueba errónea para almacenar mensajes de una permuta. ..	58
Ilustración 74. Prueba para visualizar las permutas solicitadas a un usuario.	59
Ilustración 75. Prueba errónea para visualizar permutas solicitadas a un usuario.	59
Ilustración 76. Prueba para visualizar los mensajes almacenados de una permuta.	60
Ilustración 77. Prueba errónea para visualizar los mensajes de una permuta.	60
Ilustración 78. Tabla permuta.	61
Ilustración 79. Prueba para eliminar una permuta.	61
Ilustración 80. Tabla permuta resultado.	62
Ilustración 81. Prueba errónea eliminación de una permuta.	62
Ilustración 82. Prueba para modificar la hipoteca asociada de un inmueble..	62
Ilustración 83. Prueba para modificar la hipoteca asociada de un inmueble..	63
Ilustración 84. Tabla inmueble modificada.	63
Ilustración 85. Tabla permuta modificada.	63
Ilustración 86. Prueba errónea para modificar el precio o la hipoteca de un inmueble.	64
Ilustración 87. Tabla favoritos.	64
Ilustración 88. Prueba para eliminar un inmueble.	65
Ilustración 89. Nueva tabla favoritos.	65
Ilustración 90. Prueba errónea para eliminar inmuebles.	65
Ilustración 91. Prueba para visualizar las permutas enviadas por un usuario.	66
Ilustración 94. Prueba para visualizar permutas enviadas por un usuario. ...	66
Ilustración 95. Prueba para visualizar las permutas enviadas por un usuario.	66
Ilustración 96. Prueba para visualizar permutas enviadas por un usuario.	66
Ilustración 97. Prueba errónea para visualizar las permutas enviadas por un usuario.	67
Ilustración 98. Prueba para generar un "contrato".	67
Ilustración 99. Tabla inmueble resultante.	68
Ilustración 100. Tabla favoritos resultante.	68
Ilustración 101. Tabla permuta resultante.	68
Ilustración 102. Tabla favoritos resultante.	68
Ilustración 103. Tabla permuta resultante.	68
Ilustración 104. Prueba errónea al generar el contrato.	68

Índice de Tablas

<u>Tabla 1. Petición de registro.</u>	33
<u>Tabla 2. Pe.</u>	33
<u>Tabla 3. Petición de perfil de usuario.</u>	34
<u>Tabla 4. Petición de modificación de datos de perfil.</u>	34
<u>Tabla 5. Petición de registro de inmuebles.</u>	35
<u>Tabla 6. Petición de visualización de propiedades de un usuario.</u>	36
<u>Tabla 7. Petición de búsqueda.</u>	37
<u>Tabla 8. Petición de creación de favoritos.</u>	37
<u>Tabla 9. Petición de visualización de favoritos de un usuario.</u>	38

<u>Tabla 10. Petición de creación de solicitud de permuta.</u>	38
<u>Tabla 11. Petición de almacenamiento de mensajes por un usuario.</u>	38
<u>Tabla 12. Petición de visualización de permutas solicitadas a un usuario.</u>	40
<u>Tabla 13. Petición de visualización de mensajes relacionados a una permuta.</u>	40
<u>Tabla 14. Petición de eliminación de una permuta.</u>	41
<u>Tabla 15. Petición de modificación del precio de un inmueble.</u>	41
<u>Tabla 16. Petición de modificación de la hipoteca asociada a un inmueble.</u>	41
<u>Tabla 17. Petición de eliminación de un inmueble.</u>	42
<u>Tabla 18. Petición de visualización de permutas enviadas por un usuario.</u>	42
<u>Tabla 19. Petición de generación de un "contrato".</u>	44
<u>Tabla 24. Tabla usuario.</u>	46

Tabla de Figuras

Figura 1. Diagrama permutas circulares.	7
Figura 2. Diagrama resultado de la permuta circular.	8
Figura 3. Diagrama Entidad/Relación.	12
Figura 4. Diagrama EER de Workbench.	13
Figura 5. Permuta circular valorada.	22
Figura 6. Diagrama UML del proyecto crud.	32

1 Introducción

1.1 Propósito y contextualización del tema

Actualmente, la tendencia por la transformación digital está alcanzando cada vez más popularidad en todos los sectores. En concreto, en esta memoria nos centraremos en el sector inmobiliario. Europa Press anunció en diciembre de 2021 que durante el último año la digitalización en el sector inmobiliario español había mejorado en un 28%. Publicó que la razón principal de este aumento acelerado había sido la pandemia, del mismo modo que muchos medios de comunicación. Sin embargo, tras una encuesta realizada por la consultora inmobiliaria CBRE España, acerca de las tendencias de transformación digital, el 47% de los encuestados afirmó estar dispuestos a llevar a cabo estrategias de digitalización en sus empresas, independientemente de la pandemia [1].

Según el informe “Tendencias en el mercado inmobiliario en Europa 2021” publicado por PricewaterhouseCoopers y Urban Land Institute, la digitalización es la prioridad para todas las empresas inmobiliarias en estos momentos [2]. Además, el informe describe la innovación y el compromiso con la ESG (“Environmental, Social, Governance”) como principales tendencias junto con la transformación digital.

Con respecto a la digitalización, debemos introducir el término “PropTech”, que proviene de “property technology” y se define como el conjunto de nuevas tecnologías utilizadas en el sector inmobiliario. Según ESCP Business School, entre junio de 2020 y junio de 2021 se destinaron 856 millones de euros en España, constituyendo así el segundo país más avanzado en “PropTech” [3].

Dentro de las tecnologías que ofrece “PropTech” destacan los “marketplaces” inmobiliarios. Se trata de plataformas que actúan como intermediarios entre compradores y vendedores [4].

La competitividad conducida por las tendencias comentadas, nos ha llevado a crear nuestro propio “marketplace” a través de la aplicación PermutApp. Se trata de una aplicación web de usuario que ofrece un servicio innovador dentro del sector inmobiliario. Este servicio requiere de una explicación detallada debido a que introduce un nuevo modelo de negocio al sector inmobiliario.

Dentro del sector inmobiliario podemos encontrar las dos áreas de negocio más populares: la compra/venta y el alquiler de inmuebles. Pero, también existen otros servicios como es el de las permutas, en el cual se centra PermutApp. No se trata de un negocio innovador, ya que el intercambio de propiedades no es más que un trueque y por lo tanto se ha practicado desde el inicio del comercio. Sin embargo, es en la década de los años 70, durante el auge de la construcción, cuando comienza a conocerse como un método alternativo a la venta.

Se puede definir permuta como el intercambio de viviendas entre varios propietarios, que en el caso de existir una diferencia entre los valores de los inmuebles, el propietario deudor deberá abonar al acreedor.

Desde la década de los años 70, la forma de permuta más común era el intercambio de un terreno a una constructora a cambio de un inmueble. Sin embargo, en la actualidad se realizan permutas en casos como la necesidad de cambiar de casa con urgencia y no poder o querer pedir una hipoteca para comprar una vivienda [5].

Cabe destacar como el número de permutas se ha visto aumentado en los últimos años. Del mismo modo aumentan las plataformas digitales dedicadas a la búsqueda de inmuebles y por ende sus clientes. Sin embargo, no existe ninguna plataforma digital que se centre en el sector de las permutas, por lo que PermutApp es nuestra solución y modelo de negocio innovador a este vacío en el sector.

PermutApp es una aplicación para facilitar el intercambio permanente de inmuebles entre usuarios. Para ello, nuestros usuarios deben registrar al menos una propiedad con el objetivo de intercambiarla por otra que se encuentre registrada en nuestra aplicación. El concepto que propone es un buscador de inmuebles y un intermediario para dar la posibilidad a nuestros usuarios de adquirir y a su vez poder deshacerse de una propiedad.

De este modo, nuestros usuarios podrán agilizar todos los procesos burocráticos mediante una forma segura de elegir un nuevo hogar. Esto se debe a que la aplicación acotará el mercado a aquellas personas con el mismo objetivo, por lo que la interacción entre interesados será mucho más sencilla y fiable.

Se trata de un concepto innovador que sorprenderá al mercado, ya que agilizará el proceso y eso supondrá un gran cambio en la mentalidad de las personas con predisposición de cambiar de vivienda sin demasiado esfuerzo. En el punto 1.4 se explicará mejor el modelo de negocio.

1.2 Objetivos

El objetivo principal de este proyecto es el desarrollo del “back-end” de una aplicación funcional, usable, fiable y sencilla que cumpla con las necesidades de nuestros clientes para poder así utilizarla como herramienta de un nuevo mercado en el sector inmobiliario.

Para poder conseguir el objetivo principal, será necesario comprender el problema que se quiere solucionar y diseñar y desarrollar una aplicación que cumpla todos los requisitos posibles.

El código del “back-end” de la aplicación consistirá en una API REST que deberá dar soporte a una interfaz de usuario, y almacenar los datos en una base de datos.

Por lo que se considerarán los siguientes objetivos secundarios:

1. Análisis del sector e identificación del modelo de negocio.
2. Identificación y descripción detallada de los requisitos que se quieren lograr.
3. Elección de las tecnologías a utilizar para el desarrollo.
4. Diseño y gestión de la base de datos de la aplicación.
5. Desarrollo del código.
6. Realización de un plan de pruebas.

1.3 Estructura de la Memoria

Esta memoria describe de manera detallada el desarrollo de todos los objetivos secundarios, así como los resultados y conclusiones obtenidas del proceso completo.

Para ello, dividiremos los objetivos en apartados. En primer lugar, el análisis del sector y la identificación del modelo de negocio se desarrollarán en los apartados 1.4 y 1.5. Mientras que en el siguiente capítulo se relatarán diferentes casos de uso de la aplicación y los requisitos identificados tras el análisis de las necesidades encontradas en los posibles casos de uso.

Por otro lado, el tercer capítulo trata de realizar un diseño en detalle de la implementación de la aplicación. En el podemos encontrar el diagrama E/R y las tablas y atributos de los recursos, así como un apartado para validar dicho modelo de datos.

En el cuarto capítulo se desarrollará el proceso de implementación del proyecto en el que se describe tanto el proceso de creación de la base de datos como la configuración del proyecto software en Eclipse que alberga la API.

En el apartado quinto se adjuntan las pruebas realizadas para comprobar la funcionalidad de cada requisito. Mientras que en el apartado siguiente se detallan las conclusiones y las posibles ampliaciones futuras de la aplicación.

Por último, encontramos el apartado “Análisis de Impacto” que describe las aportaciones del proyecto a la sociedad y al medio ambiente.

1.4 Modelo de negocio

Como se ha comentado, el sector inmobiliario se encuentra actualmente en un proceso de digitalización, donde las agencias convencionales están siendo sustituidas por plataformas digitales, en su gran mayoría de uso gratuito. Sin embargo, la madurez del sector inmobiliario, en términos de tecnología, todavía no cuenta con el aprobado, ya que es del 4,9 sobre 10, según el informe “Madurez digital del Sector Inmobiliario” de CBRE España [6]. Dicho informe relata los resultados de una encuesta realizada a propietarios, inversores y promotores de compañías inmobiliarias de España y Portugal. Entre los resultados obtenidos, destaca que el 30% de las empresas aún no disponen de un plan digital. Aún así, el 83% de los encuestados sí prevé un aumento de sus ingresos gracias a la inversión en digitalización en los próximos 5 años.

Mientras tanto, el 90% de las personas interesadas en cualquier servicio inmobiliario utilizan las plataformas digitales para iniciar sus procesos, ya sea para subir la información acerca de un inmueble que quieren vender o buscar propiedades en aplicaciones, según la inmobiliaria OiRealton [7].

Entre las plataformas digitales nos vamos a centrar en los “marketplaces” inmobiliarios. Se trata de las aplicaciones que ponen en contacto a compradores y vendedores, o incluso agencias, y por lo tanto, una herramienta para el servicio inmobiliario. Esta tecnología fue la primera opción, para buscar una vivienda, del 54% de las personas de una muestra de 1000 que se compraron o alquilaron un inmueble desde 2017 a 2020, según un estudio realizado por Capterra [8]. Seguidas de un 15% que utilizó una página web o aplicación propia de una agencia inmobiliaria.

Entre los resultados de la encuesta, Capterra obtuvo información relevante acerca de las características necesarias que requieren los “marketplaces”. En primer lugar, lo que más les importa a los usuarios son las imágenes de la propiedad, seguido de la protección de datos, los filtros de búsqueda, el comparador de precios y el contacto fácil con el propietario o la agencia inmobiliaria. También, pudieron conocer que el medio de contacto que prefieren los usuarios es la llamada telefónica, además del chat propio de la aplicación, ya que se demostró que el 52% lo utilizó. De estos chats, la gran mayoría prefiere tener contacto directo con un propietario y no con un “bot”.

Ante la tendencia de la transformación hacia plataformas digitales que sustituyen las funciones de las agencias convencionales, las aplicaciones deben tener presente entre sus estrategias prioritarias la innovación. Esto se debe a que un modelo innovador con un diseño sencillo, usable y fiable es el mayor

poder que las inmobiliarias y plataformas digitales del sector pueden tener para diferenciarse del resto, y así aumentar su ventaja competitiva en el sector.

Para una sociedad que busca sencillez, agilidad y control a la hora de comprar o vender propiedades, los “marketplaces” ofrecen a sus clientes estas tres características, sin embargo, en muchos casos, es la fiabilidad de las plataformas lo que puede ser un inconveniente. Esto se debe a que en la mayoría de los casos se trata de aplicaciones que se dedican únicamente a mostrar las propiedades y aportar los datos que los propietarios deseen administrar. Esto puede provocar inseguridad en los interesados, ya que en muchas ocasiones la información no es completa.

Aún así, cada vez más, las personas ven el lado positivo de utilizarlas, pues son un mecanismo de ahorros de costes y tiempo, y no solo sirven para anunciar una propiedad para que llegue a más interesados. Además, con el uso de tecnologías como la inteligencia artificial y el “Big Data”, los inmuebles registrados en las plataformas pueden llegar directamente a aquellas personas interesadas y se podrán realizar anuncios personalizados. Es por ello que, el 70% de las empresas del sector inmobiliario cuentan con equipos especializados en las tecnologías, y el número de compañías que tiene equipos de trabajo para analizar los datos recopilados y así poder mejorar en tecnología, ha aumentado en la última década [9]. Sin embargo, la mayoría de plataformas que se introducen en el sector actualmente son muy parecidas y ofrecen los mismos servicios.

Por otro lado, si analizamos los servicios que ofrecen las plataformas, vemos que destacan la compra/venta y los alquileres de inmuebles pero no encontramos ninguna plataforma que incluya la opción de permuta permanente. Como se ha mencionado, las permutas son una opción para aquellas personas que deseen agilizar el proceso de compra/venta y evitar la intervención de los bancos en hipotecas de altos intereses.

En el año 2021 se registraron 1230 operaciones en cuestión de permutas inmobiliarias, desde el mes de enero al mes de septiembre en España, según datos del INE [10]. El pasado año, se registraron el mayor número de permutas desde 2015, en el que se realizaron 7.100 movimientos. Anteriormente, esta operación se realizaba a través de agencias inmobiliarias, sin embargo, la tendencia actual es la de movimientos entre particulares que gestionan todo a su cargo.

Evolución de las permutas de viviendas en España

[Datos mensuales]



Ilustración 1. Evolución de las permutas de viviendas en España.

Fuente: datos del INE [18]

Debemos tener en cuenta que la compra/venta de una vivienda incurre en una burocracia extensa, que supone una gran cantidad de tiempo y esfuerzo para ambos interesados. Se trata de una multitud de verificaciones de la propiedad que documenten el buen estado y que cumpla con todas las leyes escritas, como son la cédula de habitabilidad, el certificado de eficiencia energética, los últimos recibos y el registro de la propiedad.

Además, muchos de las personas que desean comprar una propiedad se ven en la necesidad de incurrir en hipotecas. Según Statista, alrededor del 30% de los españoles propietarios de viviendas actualmente tienen una hipoteca [11]. Sin embargo, cada vez es más complicado conseguir que los bancos aprueben una hipoteca. Una de las razones por las que no es sencillo es que las personas deberán disponer de, al menos, el 20% del total del importe de la vivienda. También, cabe destacar que esto es un proceso largo debido a la cantidad de documentación que el banco exige.

A este periodo deberán sumarle el tiempo en el que el vendedor debe recopilar toda la documentación para poder verificar la propiedad. Este tiempo se debe a que muchos de los documentos dependerán de la firma de instituciones, y ese tiempo será por duplicado, ya que primero se deberá preparar la venta para llevar a cabo después la compra.

Por el contrario, las permutas permiten que los clientes puedan realizar un intercambio de manera rápida y sencilla, y en muchas ocasiones evitando cualquier tipo de hipoteca. Sin embargo, actualmente no existe una plataforma que facilite este servicio. Por esta razón nace PermutApp.

Este nuevo negocio propone el intercambio de propiedades con un proceso simple. En primer lugar, solo encontraremos usuarios con el mismo objetivo. Para ello, al registrarse como usuarios de la aplicación deberán incluir, al menos, una propiedad, con la intención de intercambiarla. Para ello, se ha diseñado un formulario que detalla todas las características de un inmueble y que los propietarios deberán rellenar a la hora de registrar una propiedad.

Tras el registro de la propiedad o propiedades, los usuarios podrán hacer una búsqueda personalizada de los inmuebles registrados en la aplicación por todos los usuarios. Esta búsqueda cuenta con una serie de filtros que facilitará el trabajo a los interesados. Una vez obtenida la lista de resultados, los usuarios podrán seleccionar cualquier inmueble y añadirlo a su lista de favoritos. Desde dicha lista, deberán seleccionar un inmueble y agregar que propiedad de su lista de registros desea intercambiar. La aplicación se encargará de calcular el importe resultado de la diferencia de ambos valores, teniendo en cuenta ambos precios de venta y las hipotecas pendientes, en caso de tenerlas. También, indicará cual de los usuarios será el que deberá pagar por la diferencia de estas. Este proceso generará lo que se denomina “solicitud de permuta”, que será recibida por el usuario propietario del inmueble de interés.

Dicho importe podrá ser recalculado, ya que nuestra aplicación ofrece un chat que pondrá en contacto a dos usuarios en relación a una permuta. A través de dicho chat, podrán llevarse a cabo contraofertas y los usuarios podrán dar su opinión.

Cada usuario será el responsable de enviar tantas solicitudes de permuta como considere oportunas y al mismo tiempo recibirá solicitudes de otros usuarios. De este modo, existirán numerosas posibilidades de permuta incluso sin llegar a realizar solicitudes, ya que podrá recibirlas.

Cuando los usuarios involucrados se ponen de acuerdo en llevar a cabo una permuta o intercambio, la aplicación generará un “contrato”. En dicho contrato se mostrarán los datos de los participantes, los inmuebles y la cuantía final calculada.

Con respecto a nuestros usuarios potenciales, la aplicación está dirigida a un sector muy específico, lo que nos permite centrarnos en cumplir los requisitos de dichos usuarios en más detalle. Para ello, nos hemos informado de los intereses de los clientes de dicho sector que se representan en el punto 2.1.

Para explicar en más detalle la finalidad de la aplicación se van a describir diferentes casos de posibles usuarios de PermutApp.

Un ejemplo de usuarios podría ser el caso de un matrimonio que compró una casa familiar de más de 200 metros cuadrados para vivir mientras crecían sus hijos, pero estos ya se han ido de casa, por lo que quieren cambiarse a un hogar más pequeño. Además, creen que lo mejor es deshacerse de la vivienda familiar para poder afrontar los costes de un nuevo hogar sin tener que involucrar a los bancos para pedir un préstamo. El matrimonio quiere mudarse en su mismo barrio, pero les está siendo complicado vender su vivienda actual a través de las agencias convencionales. Además, el hecho de tener que vender para luego tener que comprar les limita, ya que cuando encuentran un piso que les gusta, rápidamente deja de estar disponible debido a que otros interesados, que sí disponen del dinero, se adelantan.

En su mismo barrio se encuentra una pareja joven, propietaria de un piso de 80 metros cuadrados, que está buscando una casa de mayor tamaño para ampliar la familia. Esta pareja se enfrenta ante el problema de los bancos y la burocracia. Está intentando conseguir una hipoteca para un nuevo hogar antes de saber por cuanto dinero finalmente podrá vender su casa, por lo que le están poniendo bastantes impedimentos y unos intereses cada vez más altos. Además, le gustaría mudarse cuanto antes, pero le ocurre la misma situación que al matrimonio.

Por lo que, a través de la aplicación ambos podrían conseguir sus objetivos sin necesidad de esperar a que ninguna venta se produzca o tener que contactar a ninguna agencia.

Otro ejemplo de posibles usuarios podrían ser aquellos que necesiten o deseen aumentar su liquidez cambiando un inmueble de mayor valor por otro de menor valor. También, aquellos que deseen cambiar su residencia en la ciudad por la vida en el campo y viceversa, ya sea por trabajo o por cambiar de estilo de vida.

Por otro lado, podrían recurrir a PermutApp aquellas personas que deban cambiar de vivienda por urgencia tras haber sufrido algún percance que no les permita vivir en su casa, por ejemplo, una incapacidad de movilidad.

Estos ejemplos son algunos de los numerosos casos que podrían llevar a usuarios a utilizar PermutApp.

Además, se ha diseñado para poder integrar la funcionalidad para realizar permutas entre más de dos partes. Esto quiere decir, que no necesariamente, las permutas tienen porque hacerse entre dos usuarios, sino que al enviar y recibir solicitudes, cabe la posibilidad de poner en contacto a más interesados en la misma operación.

Para ello, cuando un usuario recibe una solicitud de permuta por uno de sus inmuebles pero este no está interesado en el inmueble propuesto por el solicitante, sino que está interesado en otro inmueble que ha visualizado en la

aplicación, podrá generar una solicitud de permuta para mostrar su interés acerca de este nuevo inmueble. De modo que el nuevo actor recibirá la solicitud de permuta, y además, tendrá acceso a visualizar las solicitudes de permuta del inmueble propuesto para intercambiar por el solicitante. Por lo que en caso de estar interesado en alguna de las permutas solicitadas a dicho inmueble, podrá mostrar su interés generando una nueva solicitud de permuta al propietario. Al final se habrá formado un ciclo, en el que cada usuario será solicitado y solicitante.

A continuación se muestra un diagrama a modo de ejemplo:

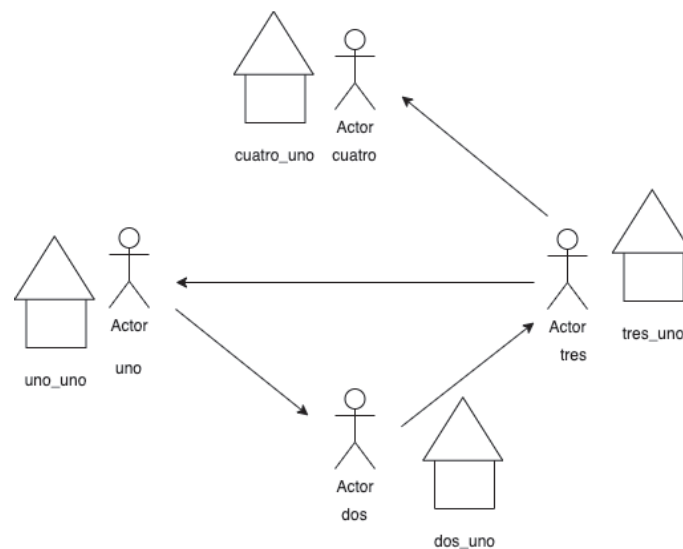


Figura 1. Diagrama permutas circulares.

En este diagrama podemos ver como el actor “uno” está interesado en el inmueble “dos_uno” del actor “dos”, por lo que le envía una solicitud de permuta. Por otro lado, vemos como el actor “dos” que ha recibido la solicitud de permuta del actor “uno” está interesado en el inmueble “tres_uno” del actor “tres”, por lo que le envía una solicitud de permuta.

En este momento el usuario “tres”, al haber recibido la solicitud del usuario “dos”, puede visualizar la cadena de permutas que se ha establecido antes de dicha solicitud, es decir la solicitud del actor “uno” al actor “dos”. En esa solicitud, usuario “uno” quiere adquirir el inmueble “dos_uno” y deshacerse de su inmueble “uno_uno”, por lo que está buscando algún interesado en su inmueble “uno_uno”. Usuario “tres” tras visualizar esta solicitud de permuta del usuario “uno” muestra su interés por el inmueble “uno_uno”, enviándole así una solicitud de permuta a este.

En el diagrama podemos ver que existe un usuario “cuatro” al que el usuario “tres” también le ha enviado una solicitud de permuta. Sin embargo, este no ha mostrado interés por ninguno de los inmuebles involucrados en la operación. Tras formarse el ciclo, el usuario “uno” que recibe la solicitud del “tres”, responde ante esta solicitud mostrando su interés por deshacerse de su inmueble “uno_uno”. Del mismo modo responde el usuario “tres” al usuario “dos” al ver que ha conseguido el inmueble “uno_uno” y que la oferta que le ofrece la permuta con usuario “dos” por la diferencia de valor entre su inmueble “tres_uno” y el inmueble “dos_uno” le interesa económicamente. Por último el usuario “dos” acepta la solicitud del usuario “uno” por su inmueble “dos_uno”.

Por lo que el usuario “uno” habría logrado adquirir el inmueble “dos_uno”, el usuario “dos” el inmueble “tres_uno” y el usuario “tres” el inmueble “uno_uno”.

Por lo que finalmente, podemos ver el resultado de la permuta circular en el siguiente diagrama, en el que podemos ver como el usuario “cuatro” no ha realizado ningún movimiento.

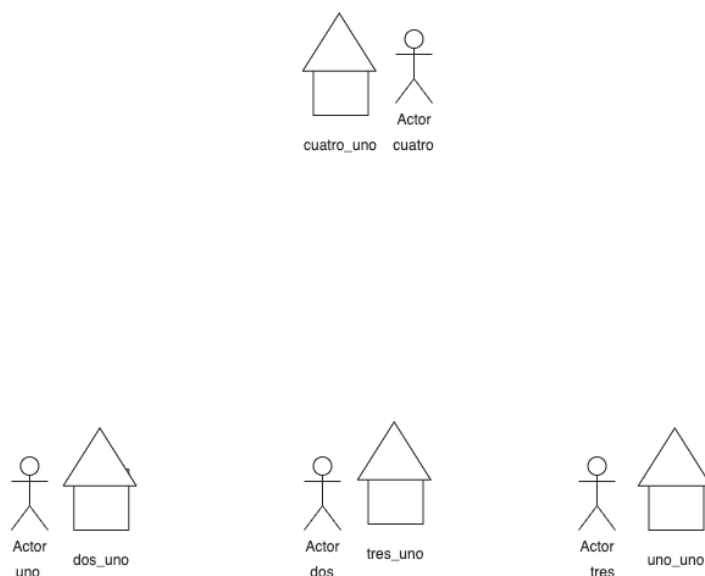


Figura 2. Diagrama resultado de la permuta circular.

En definitiva, PermutApp sería la forma más eficiente posible, en el mercado actual, de cambiar de propiedad minimizando los costes, evitando en mayor medida la intervención de los bancos y utilizando una herramienta segura y actualizada. Con tan solo una rápida búsqueda nuestros usuarios podrían lograr agilizar un proceso que sin la aplicación llevaría meses, como es el de poner a la venta una propiedad y la adquisición de otra.

Por otro lado, nuestros usuarios constituirían un sector exclusivo, que estaría formado por personas con un interés claro, que sería el de cambiar de propiedad de forma rápida, abaratando los costes que esto supone. Las condiciones necesarias para ser usuario de PermutApp, serían las de poseer una vivienda en propiedad y conocer la tecnología de usuario de una aplicación web.

Por su innovación y todas las razones explicadas, la aplicación PermutApp se puede considerar una buena estrategia en el sector inmobiliario.

1.5 Estado del arte

Actualmente, existen numerosas aplicaciones web para anunciar una vivienda para la venta, así como agencias que se han sumado a la transformación digital. Sin embargo, no existe una plataforma digital dedicada en exclusiva a realizar permutas inmobiliarias permanentes.

Podríamos considerar como nuestros mayores competidores, los “marketplaces” Idealista o Fotocasa, ya que ambos funcionan como buscadores de inmuebles y utilizan unos filtros similares a los nuestros.

Idealista es considerada la aplicación inmobiliaria más utilizada en España. Según los datos que recoge App Annie, durante el mes de febrero de 2021 tuvo 1,4 millones de usuarios y sumó más de 4,8 millones de horas de uso en su aplicación móvil [12].

Sin embargo, como se ha mencionado, ninguna de las dos aplicaciones móviles más utilizadas, ni ninguna aplicación del mercado, ofrece un servicio de intermediación exclusivo para permutas permanentes.

Además, en ambas aplicaciones, el servicio de contacto entre el propietario y el comprador interesado es externo a la aplicación, por lo que nuestro servicio tendría aspectos muy diferentes a ambas aplicaciones.

Cabe destacar que sí que existe un blog que actúa como intermediario para realizar intercambios de viviendas temporales. Pero a diferencia de PermutApp, se trata de intercambios sin coste por parte de ningún interesado y con un periodo máximo de tiempo. La web en cuestión se llama “home Exchange” [13].

2 Análisis

2.1 Casos de uso

El objetivo de esta aplicación es ofrecer una plataforma digital que ponga en contacto a personas con el fin de realizar una permuta. Por lo que, el caso de uso principal de la aplicación es la realización de un intercambio de propiedades entre usuarios. Sin embargo, en la aplicación podemos encontrar los siguientes casos de uso:

1. Registrar un usuario.
2. Registrar un inmueble.
3. Eliminar un inmueble.
4. Iniciar sesión.
5. Visualizar los datos de un usuario.
6. Modificar los datos de un usuario.
7. Visualizar las propiedades de un usuario.
8. Visualizar las propiedades registradas por todos los usuarios con disponibilidad de filtrado.
9. Añadir a la lista de favoritos de un usuario un inmueble.
10. Visualizar la lista de favoritos de un usuario.
11. Generar una solicitud de permuta a partir de un inmueble de la lista de favoritos.
12. Almacenar mensajes de una permuta.
13. Visualizar mensajes de una permuta.
14. Modificar el precio o la hipoteca de un inmueble.
15. Visualizar las permutas solicitadas por un usuario.
16. Visualizar las permutas solicitadas a un usuario.
17. Eliminar una permuta.
18. Generar un contrato.

2.2 Especificación de requisitos

Los requisitos definen el comportamiento del sistema y las funcionalidades de la aplicación.

R01: Los usuarios deben registrarse en la aplicación. Para ello el usuario debe proporcionar: nombre, apellidos, dni, domicilio, teléfono, email, contraseña y cuenta bancaria. Además, será obligatorio registrar una propiedad con el objetivo de realizar un intercambio. Para ello debe proporcionar como datos de la propiedad: ciudad, barrio, dirección, dimensión, precio, la cuantía de la hipoteca pendiente a aportar, tipo de vivienda (casa, chalet, piso, ático), el estado (obra nueva, buen estado, necesario reforma), el tipo de área exterior que tenga en el caso de que la tenga (jardín, terraza, sin exteriores) y si tiene garaje.

R02: Login de usuario. El usuario se identificará con su email y contraseña, una vez se haya registrado.

R03: Consultar los datos de un usuario desde su perfil.

R04: Modificar los datos de su perfil (domicilio, teléfono, email, contraseña, cuenta bancaria). El email solo podrá modificarse en caso de que no exista otro usuario con el mismo email.

R05: Registrar más propiedades en el perfil de un usuario.

R06: Consultar todas las propiedades registradas por un usuario.

R07: Consultar las propiedades registradas en la aplicación por todos los usuarios. Además, podrá realizar una búsqueda añadiendo los filtros que desee

entre: ciudad, barrio, un rango de dimensión, un rango de precio, que no tenga hipoteca pendiente, el tipo de vivienda, el estado, el tipo de áreas exteriores y si tiene garaje.

R08: Añadir un inmueble a la lista de favoritos de un usuario.

R09: Consultar la lista de inmuebles favoritos de un usuario.

R10: Generar una solicitud de permuta de un inmueble desde la lista de favoritos. Para ello, el usuario que esté interesado en intercambiar una propiedad por otra de su lista de favoritos, deberá indicar que inmueble entre sus propiedades quiere intercambiar. Además, al indicar dicho inmueble, se calculará la cuantía resultante de dicha permuta, es decir, la diferencia de precios entre ambos inmuebles y quien deberá ser el usuario pagador. Para calcular dicha cuantía se tendrán en cuenta los precios y las hipotecas asociadas de ambos inmuebles.

R11: Los usuarios involucrados en una permuta podrán enviar mensajes para mostrar sus intereses y sus contraofertas, en el caso de tenerlas. Esto se hará a través de un canal facilitado para cada permuta. Todos los mensajes serán almacenados.

R12: El usuario podrá consultar las permutas que se le hayan solicitado por sus inmuebles.

R13: Los usuarios involucrados en una permuta podrán recibir mensajes del solicitante. Es decir, podrá visualizar los mensajes almacenados ordenados por tiempo de envío. Todos los mensajes estarán relacionados con una única permuta.

R14: El usuario solicitado puede rechazar la solicitud de permuta, eliminándola.

R15: Los usuarios podrán modificar el precio o la cantidad de hipoteca pendiente en cualquier momento. En caso de existir permutas asociadas al inmueble modificado, se actualizará la nueva cuantía asociada a dicha permuta.

R16: Los usuarios podrán eliminar de la aplicación cualquiera de sus propiedades en cualquier momento. De modo que estas también desaparecerán de la lista de favoritos de los usuarios que lo haya seleccionado y se eliminarán las solicitudes de permutas en las que participe.

R17: El usuario podrá visualizar las permutas que haya solicitado.

R18: Una vez un usuario acepta una solicitud de permuta, se eliminarán automáticamente de la aplicación los inmuebles partícipes y se imprimirá un contrato con los datos de ambos usuarios, los datos de los inmuebles a intercambiar, el importe a pagar y el nombre del usuario que deberá aportar la cuantía.

3 Diseño

En este capítulo se realiza el diseño del modelo de datos que se ha utilizado para el desarrollo de la aplicación.

En primer lugar, se han descrito los siguientes recursos:

- Usuario: contiene los datos de los usuarios registrados.
- Inmueble: contiene los datos de los inmuebles registrados.
- Favoritos: contiene todas las listas de inmuebles favoritos de los usuarios registrados.
- Permuta: contiene los datos de todas las permutas solicitadas a todos los usuarios registrados.
- Mensajes: contiene los mensajes de todas las permutas solicitadas en la aplicación.

3.1 Diagrama entidad relación

A continuación, se describen las relaciones entre los diferentes recursos tal y como se ha diseñado la aplicación. Para ello, se ha utilizado un diagrama entidad-relación.

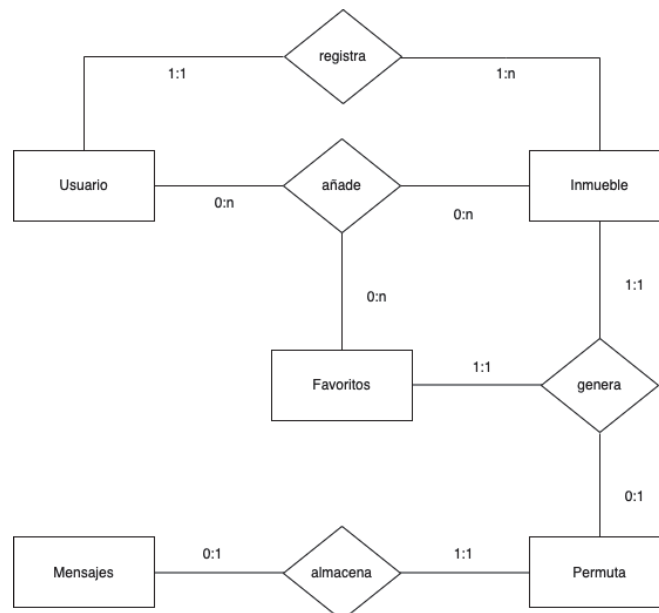


Figura 3. Diagrama Entidad/Relación.

3.2 Tablas y Atributos

En este apartado se especificarán las tablas de recursos que han sido modeladas con la herramienta MySQL Workbench. El proceso de creación de la base de datos se especifica en el apartado 4.2.1.

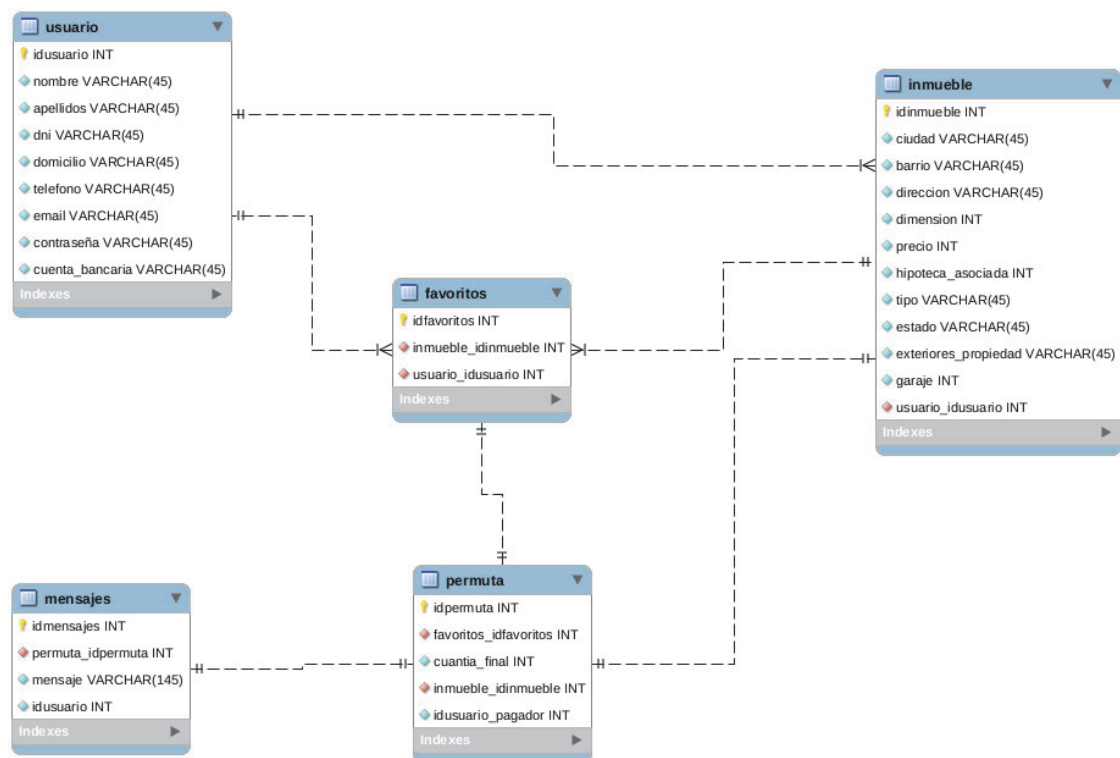


Figura 4. Diagrama EER de Workbench.

Tabla usuario:

Los datos del usuario a excepción del nombre y apellidos, solo serán compartidos con otros usuarios en el caso de generar un contrato.

- **idusuario:** PRIMARY KEY, único para cada usuario y se incrementa al añadir un nuevo usuario. No podrá modificarse como dato de usuario.
- **nombre:** no podrá modificarse como dato de usuario.
- **apellidos:** no podrá modificarse como dato de usuario.
- **dni:** no podrá modificarse como dato de usuario.
- **domicilio:** sí podrá modificarse como dato de usuario.
- **teléfono:** sí podrá modificarse como dato de usuario.
- **email:** sí podrá modificarse como dato de usuario, pero solo si el nuevo email no está registrado en la base de datos.
- **contraseña:** sí podrá modificarse como dato de usuario, por cualquier valor.
- **cuenta_bancaria:** sí podrá modificarse como dato de usuario.

usuario - Table										
Table	Columns	Indexes	Foreign Keys	Triggers	Partitioning	Options	Inserts	Privileges		
Column Name						Datatype	PK	NN	UQ	BIN
idusuario						INT	☒	☒	☐	☐
nombre						VARCHAR(45)	☐	☒	☐	☐
apellidos						VARCHAR(45)	☐	☒	☐	☐
dni						VARCHAR(45)	☐	☒	☐	☐
domicilio						VARCHAR(45)	☐	☒	☐	☐
telefono						VARCHAR(45)	☐	☒	☐	☐
email						VARCHAR(45)	☐	☒	☐	☐
contraseña						VARCHAR(45)	☐	☒	☐	☐
cuenta_bancaria						VARCHAR(45)	☐	☒	☐	☐

Ilustración 2. Generación de Tabla usuario en Workbench.

#	idusuario	nombre	apellidos	dni	domicilio	telefono	email	contraseña	cuenta_bancaria
1	1	Maria	Jiménez	9876554A	calle 1	666898765	mariajimenez@com	abcd	ES2200 0022 0033 ...
2	2	Carlos	Gómez	2356554B	calle 22	675943545	carlosgomez@com	1234	ES3600 0023 2653 ...
3	3	Juan	Gil	24678933B	calle 13	679934756	juangil@com	0000	ES8844 2500 2300 ...
4	4	Mariana	García	05669873G	calle 14	688933755	marianagarcia@com	0220	ES8486 2520 3340 ...
5	5	Alvaro	Lozano	056443768G	calle de la paz	623457689	alvarolozano@com	1020	ES4334 5225 3443 ...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 3. Ejemplo de Tabla usuario en Workbench.

Tabla inmueble:

Ningún dato podrá modificarse una vez registrado, a excepción del precio y la hipoteca asociada.

- **idinmuelle:** PRIMARY KEY, único para cada inmueble y se incrementa al añadir un nuevo inmueble.
- **ciudad:** nombre de la ciudad donde se ubica el inmueble.
- **barrio:** nombre del barrio donde se ubica el inmueble.
- **dirección:** dirección exacta donde se ubica el inmueble. Este dato solo estará disponible para el usuario propietario, quien decidirá si compartirlo.
- **precio:** precio de mercado del inmueble impuesto por el propietario.
- **hipoteca_asociada:** importe de la hipoteca pendiente de abonar del inmueble.
- **tipo:** define el tipo de inmueble registrado (piso, ático, chalet, casa).
- **estado:** describe el estado del inmueble (necesario reforma, buen estado, obra nueva)
- **exteriores_propiedad:** describe el tipo de área exterior en el caso de tenerla (balcón, terraza, jardín).
- **garaje:** indica si la propiedad tiene garaje (0 o 1).
- **usuario_idusuario:** FOREIGN KEY, identificador del usuario propietario.

Inmueble - Table										
Table	Columns	Indexes	Foreign Keys	Triggers	Partitioning	Options	Inserts	Privileges		
Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
idinmuelle	INT	☒	☒	☐	☐	☐	☐	☒	☐	
ciudad	VARCHAR(45)	☐	☒	☐	☐	☐	☐	☐	☐	
barrio	VARCHAR(45)	☐	☒	☐	☐	☐	☐	☐	☐	
direccion	VARCHAR(45)	☐	☒	☐	☐	☐	☐	☐	☐	
dimension	INT	☐	☒	☐	☐	☐	☐	☐	☐	
precio	INT	☐	☒	☐	☐	☐	☐	☐	☐	
hipoteca_asociada	INT	☐	☒	☐	☐	☐	☐	☐	☐	
tipo	VARCHAR(45)	☐	☒	☐	☐	☐	☐	☐	☐	
estado	VARCHAR(45)	☐	☒	☐	☐	☐	☐	☐	☐	
exteriores_propiedad	VARCHAR(45)	☐	☒	☐	☐	☐	☐	☐	☐	
garaje	INT	☐	☒	☐	☐	☐	☐	☐	☐	
usuario_idusuario	INT	☐	☒	☐	☐	☐	☐	☐	☐	

Ilustración 4. Generación de Tabla inmueble en Workbench.

#	idinmuelle	ciudad	barrio	direccion	dimension	precio	hipoteca_asociada	tipo	estado	exteriores_propiedad	garaje	usuario_idusuario
1	1	Madrid	canillas	calle 1	100	200000	30000	piso	buen estado	terrazza	0	1
2	7	Madrid	CORDE O...	Avenida de los ...	500	2000000	0	chalet	buen estado	jardin	1	1
3	9	Madrid	Arturo Soria	Calle de Arturo...	120	200000	60000	piso	buen estado	terrazza	1	2
4	10	Madrid	Goya	Calle de Goya	150	300000	20000	piso	buen estado	terrazza	1	2
5	11	Madrid	Malasaña	Calle del pez	80	350000	20000	piso	necesario ref...	balcón	0	2
6	12	Madrid	La Moraleja	Avenida de la ...	400	1500000	0	casa	obra nueva	jardin	1	2
7	13	Madrid	Coslada	Avenida de la c...	200	150000	30000	piso	obra nueva	terrazza	1	3
8	14	Madrid	Getage	Avenida de get...	160	130000	10000	piso	buen estado	balcón	1	4
9	15	Madrid	Alcobendas	Avenida 10	360	700000	0	ático	buen estado	terrazza	1	5
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 5. Ejemplo de Tabla inmueble en Workbench.

Tabla favoritos:

- **idfavoritos:** PRIMARY KEY, único para cada fila de favoritos y se incrementa al añadir un nuevo favorito.
- **inmueble_idinmueble:** FOREIGN KEY, identificador del inmueble que el usuario con identificador usuario_idusuario quiere añadir a su lista de favoritos. Si el inmueble es eliminado de la base de datos, las filas de la tabla favoritos en las que intervenga también se eliminarán.
- **usuario_idusuario:** FOREIGN KEY, identificador del usuario que añade el inmueble con identificador inmueble_idinmueble a su lista de favoritos.

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
idfavoritos	INT	☒	☒	☐	☐	☐	☐	☒	☐	
inmueble_idinmueble	INT	☐	☒	☐	☐	☐	☐	☐	☐	
usuario_idusuario	INT	☐	☒	☐	☐	☐	☐	☐	☐	

Ilustración 6. Generación de Tabla favoritos en Workbench.

#	idfavoritos	inmueble_idinmueble	usuario_idusuario
1	7	1	3
2	8	9	1
3	9	13	1
4	10	14	1
5	11	1	2
6	12	7	2
7	13	15	2
8	14	14	3
9	15	12	5

Ilustración 7. Ejemplo de Tabla favoritos en Workbench.

Tabla permuta:

- **idpermuta:** PRIMARY KEY, único para cada permuta generada y se incrementa al añadir una nueva permuta.
- **favoritos_idfavoritos:** FOREIGN KEY, identificador del favorito a partir del cual se quiere generar una permuta. Si la fila con identificador favoritos_idfavoritos es eliminada de la base de datos, también se eliminarán las filas de la tabla permuta donde intervenga.
- **cuantía_final:** importe que indica la diferencia de ambos precios teniendo en cuenta las hipotecas asociadas a los inmuebles. Supondrá el importa que un usuario deberá pagar al otro. Este campo se modificará cada vez que se modifique alguna variable de su ecuación en la base de datos.

$$\text{Cuantía} = |(\text{precio1-hipoteca1})-(\text{precio2-hipoteca2})|$$

- **inmueble_idinmueble:** FOREIGN KEY, el identificador corresponde a un inmueble cuyo propietario es el usuario solicitante de la permuta, es decir, el usuario que interviene en la fila favoritos con identificador favoritos_idfavoritos. Si este inmueble es eliminado o modificado en la tabla inmueble, los cambios serán reflejados en la tabla permuta. De modo que si el inmueble es eliminado, la solicitud de permuta también quedará eliminada.
- **usuario_pagador:** será el identificador que indique quién de los dos participantes de la permuta deberá pagar por la cuantía_final al otro.

permuto - Table

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
idpermuto	INT	☒	☒	☐	☐	☐	☐	☒	☐	
favoritos_idfavoritos	INT	☐	☒	☐	☐	☐	☐	☐	☐	
cuantia_final	INT	☐	☒	☐	☐	☐	☐	☐	☐	
inmueble_idinmueble	INT	☐	☒	☐	☐	☐	☐	☐	☐	
idusuario_pagador	INT	☐	☒	☐	☐	☐	☐	☐	☐	

Ilustración 8. Generación de Tabla permuto en Workbench.

#	idpermuto	favoritos_idfavoritos	cuantia_final	inmueble_idinmueble	idusuario_pagador
1	6	12	1860000	9	2
2	7	8	1860000	7	2
3	8	15	800000	15	5
4	9	14	0	13	4
5	10	8	30000	1	2
*	NULL	NULL	NULL	NULL	NULL

Ilustración 9. Ejemplo de Tabla permuto en Workbench.

Tabla mensajes:

- **idmensajes:** PRIMARY KEY, único para cada mensaje almacenado y se incrementa al añadir un nuevo mensaje.
- **permuto_idpermuto:** FOREIGN KEY, identificador de la permuto relativa al mensaje. Al eliminar una permuto se eliminarán los mensajes asociados a esa permuto de la base de datos.
- **mensaje:** texto en cuestión que se quiere enviar.
- **idusuario:** identificador del usuario emisor. El identificador deberá ser uno de los dos que intervienen en la permuto con identificador permuto_idpermuto.

mensajes - Table

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
idmensajes	INT	☒	☒	☐	☐	☐	☐	☒	☐	
permuto_idpermuto	INT	☐	☒	☐	☐	☐	☐	☐	☐	
mensaje	VARCHAR(145)	☐	☒	☐	☐	☐	☐	☐	☐	
idusuario	INT	☐	☒	☐	☐	☐	☐	☐	☐	

Ilustración 10. Generación de Tabla mensajes en Workbench.

#	idmensajes	permuta_idpermuta	mensaje	idusuario
1	8	6	Hola, estaría interesada en su inmueble del barrio Conde Orgaz, estaría interesado en negocia por mi piso de Arturo Soria?	2
2	9	6	Hola, me parece una buena propiedad y sí estaría interesada.	1
3	10	6	Me alegro, sin embargo, considero que un precio justo por su inmueble sería 1.900.000, estaría dispuesta?	2
4	11	6	No, lo mejor será seguir buscando otras opciones	1
5	12	9	Hola, me interesaría realizar una permuta por su inmueble en Getafe, ya que estoy buscando mudarme allí.	3
6	13	9	Buenas tardes, la verdad es que tanto su inmueble como su zona son lo que estaba buscando.	4
7	14	9	De acuerdo, procedamos. Además, el importe es de 0 euros, por lo que el intercambio será sencillo.	3
8	15	9	Estoy de acuerdo, generaré un contrato.	4
*	NULL	NULL	NULL	NULL

Ilustración 11. Ejemplo de Tabla mensajes en Workbench.

3.3 Validación del diseño

Para realizar la validación del diseño se ha comprobado que el modelo de datos sea el óptimo para el cumplimiento de los casos de uso descritos en el apartado 2.1. Para ello, se han realizado las consultas en lenguaje sql necesarias para su desarrollo y se ha comprobado el funcionamiento de estas.

Se han utilizado consultas simples ya que las consultas anidadas no permitían el control de errores que se requería.

1. Registrar un usuario:

Primero será necesario comprobar si dicho usuario ya estaba registrado en la aplicación. Para ello se utiliza la consulta:

```
"SELECT * FROM `permutasBD`.`usuario` WHERE `email` = 'value';"
```

siendo *value*, la dirección de correo electrónica del usuario que se va a registrar.

En el caso de no existir dicho *email* en la tabla "usuario", se procederá a hacer la inserción como:

```
"INSERT INTO `permutasBD`.`usuario`
(`nombre`,`apellidos`,`dni`,`domicilio`,`telefono`,`email`,
`contraseña`,`cuenta_bancaria`)VALUES('value1','value2',
'value3','value4','value5','value6','value7','value8');"
```

2. Registrar un inmueble:

```
"INSERT INTO `permutasBD`.`inmueble`
(`ciudad`,`barrio`,`direccion`,`dimension`,`precio`,`hipoteca_asociada`,`tipo`,`estado`,`exteriores_propiedad`,`gараje`,`usuario_idusuario`)VALUES('value1','value2','value3',
'value4','value5','value6','value7','value8','value9','value10','value11');"
```

3. Eliminar un inmueble:

```
"DELETE FROM `permutasBD`.`inmueble` WHERE `idinmueble` =value;"
```

4. Iniciar sesión:

Será necesario consultar si el usuario con un email y una contraseña determinada está en la base de datos. Para ello se realiza la siguiente consulta:

```
"SELECT * FROM `permutasBD`.`usuario` WHERE `email` = 'value1' AND `contraseña` = 'value2';"
```

5. Visualizar los datos de un usuario:

```
"SELECT * FROM `permutasBD`.`usuario` WHERE `idusuario` = value;"
```

6. Modificar los datos de un usuario:

Para modificar el domicilio, el teléfono, la contraseña o la cuenta bancaria se realizará la siguiente operación:

```
"UPDATE `permutasBD`.`usuario` SET `domicilio` = 'value1',
`telefono` = 'value2', `contraseña` = 'value2',
`cuenta_bancaria` = 'value3' WHERE `idusuario` = value4; "
```

Mientras que si se intenta modificar el email del usuario se realiza la siguiente operación que comprueba que dicho email no está registrado en la base de datos y así que no pertenece a otro usuario.

```
"SELECT * FROM `permutasBD`.`usuario` WHERE `email`
='value1' AND `idusuario` != 'value2';"
```

En caso de no pertenecer a ningún usuario, se realizará la operación anterior con *value1* y *value2*

7. Visualizar las propiedades de un usuario:

```
"SELECT * FROM `permutasBD`.`inmueble` WHERE
`usuario_idusuario` = value;"
```

8. Visualizar las propiedades registradas por todos los usuarios:

Se podrá realizar un búsqueda completa con la consulta:

```
"SELECT * FROM `permutasBD`.`inmueble`"
```

Además, se podrán añadir los siguientes filtros: ciudad, barrio, un rango de dimensión, un rango de precio, que no tenga hipoteca pendiente, el tipo de vivienda, el estado, el tipo de áreas exteriores y si tiene garaje.

A modo de ejemplo se presentan las siguientes consultas:

```
"SELECT * FROM `permutasBD`.`inmueble` WHERE `ciudad` =
'value1' AND `dimension` >= value2 AND `precio` <= value3; "
```

```
"SELECT * FROM `permutasBD`.`inmueble` WHERE `ciudad` =
'value1' AND `barrio` = 'value2' AND `tipo` = 'value3' AND
`garaje` = value4;"
```

9. Añadir a la lista de favoritos de un usuario un inmueble:

En primer lugar, se debe comprobar que el inmueble no sea una propiedad del usuario que realiza la acción. Para ello se lleva a cabo la siguiente consulta:

```
"SELECT * FROM `permutasBD`.`inmueble` WHERE `idinmueble` =
value1 AND `usuario_idusuario` != value2;"
```

Si la llamada es exitosa se realizará la operación de inserción:

```
"INSERT INTO `permutasBD`.`favoritos`
(`inmueble_idinmueble`,`usuario_idusuario`) VALUES (value1,
value2);"
```

10. Visualizar la lista de favoritos de un usuario:

```
"SELECT * FROM `permutasBD`.`favoritos` WHERE
`usuario_idusuario` = value;"
```

Para poder visualizar los datos de cada inmueble añadido como favorito por el usuario, realizamos sobre cada favorito obtenido la siguiente operación:

```
"SELECT * FROM `permutasBD`.`inmueble` WHERE `idinmueble` =
value;"
```

Siendo *value* el "inmueble_idinmueble" de cada fila de la tabla "favoritos".

11. Generar una solicitud de permuta:

Desde la lista de favoritos se llevará a cabo la operación de crear permuta. Para ello, se realizará la siguiente operación de inserción:

```
"INSERT INTO `permutasBD`.`permuta`
(`favoritos_idfavoritos`,`cuantia_final`,
`inmueble_idinmueble`,`idusuario_pagador`) VALUES (value1,
value2, value3, value4);"
```

12. Almacenar mensajes de una permuta:

```
"INSERT INTO `permutasBD`.`mensajes` (`permuta_idpermuta`,
`mensaje`, `idusuario`) VALUES (value1,'value2',value3);"
```

13. Visualizar mensajes de una permuta.

```
"SELECT * FROM `permutasBD`.`mensajes` WHERE
`permuta_idpermuta` = value order by `idmensajes`;"
```

14. Modificar el precio o la hipoteca de un inmueble:

Para modificar el precio se realizará la consulta:

```
"UPDATE `permutasBD`.`inmueble` SET `precio` = value1 WHERE
`idinmueble` = value2;"
```

Mientras que para para modificar una hipoteca se realizará la siguiente operación:

```
"UPDATE `permutasBD`.`inmueble` SET `hipoteca_asociada` =
value1 WHERE `idinmueble` = value2;"
```

En caso de estar involucrado en alguna solicitud de permuta, deberá recalcularse la cuantía resultante de esta modificación con el nuevo precio o hipoteca y modificar la tabla permuta en la que intervenga.

Para comprobar si tiene permutas, deberemos analizar el papel del inmueble.

Por un lado será necesario comprobar si se trata del inmueble de un usuario que ha solicitado alguna permuta o si se trata de un inmueble que ha sido solicitado para permutar por otro usuario.

Para comprobar el primer caso, simplemente se realizará la siguiente consulta:

```
"SELECT * FROM `permutasBD`.`permuta` WHERE
`inmueble_idinmueble` =value;"
```

Si se trata del inmueble cuyo propietario es el solicitante de la permuta, será necesario volver a realizar el cálculo de la cuantía y modificar la tabla permutas con la consulta:

```
"UPDATE `permutasBD`.`permuta` SET `cuantia_final` =
value1,`idusuario_pagador` = value2 WHERE
`idpermuta`=value3;"
```

Para comprobar si el inmueble ha sido solicitado para permutar, se debe comprobar primero si se encuentra en la lista de favoritos de algún usuario. Para ello se realiza la siguiente consulta:

```
"SELECT * FROM `permutasBD`.`favoritos` WHERE
`inmueble_idinmueble` =value;"
```

En el caso de pertenecer a alguna lista se deberá comprobar si se ha generado alguna permuta con dicha entrada de la lista de favoritos, de modo que se realiza la siguiente operación:

```
"SELECT * FROM `permutasBD`.`permuta` WHERE
`favoritos_idfavoritos` = value;"
```

En caso de que existan permutas, se realizará la misma operación para modificar la cuantía y el usuario pagador ya descrita.

15. Visualizar las permutas solicitadas por un usuario:

Primero se busca la lista de favoritos de dicho usuario con la consulta:

```
"SELECT * FROM `permutasBD`.`favoritos` WHERE
`usuario_idusuario` = value;"
```

Y después se obtienen todas las permutas que haya solicitado el usuario con la siguiente consulta:

```
"SELECT * FROM `permutasBD`.`permuta` WHERE
`favoritos_idfavoritos` =value;"
```

Para poder visualizar los datos completos de permuta, se obtendrán los datos de los inmuebles que intervienen en cada permuta.

Para ello, se realizará la siguiente operación dos veces por cada permuta obtenida:

```
"SELECT * FROM `permutasBD`.`inmueble` WHERE `idinmueble` =
value;"
```

En la primera vez, *value* consistirá en “idinmueble_usuario” de la tabla “permuta” obtenido en la operación anterior. Mientras que la segunda vez *value* consistirá en “inmueble_idinmueble” de la tabla “favoritos” relacionada con cada entrada de la tabla “permuta”.

16. Visualizar las permutas solicitadas a un usuario:

Será necesario buscar en la tabla “favoritos” cada inmueble del que el usuario sea propietario mediante la iteración de la siguiente operación:

```
"SELECT * FROM `permutasBD`.`favoritos` WHERE
`inmueble_idinmueble` = value;"
```

En caso de estar algún inmueble en la lista de favoritos de algún usuario, se realizará la siguiente operación para consultar si existe una permuta con dicho favorito:

```
"SELECT * FROM `permutasBD`.`permuta` WHERE
`favoritos_idfavoritos` = "
favoritosList.get(i).getIdfavoritos() + ";"
```

Para poder visualizar los datos completos de permuta, se obtendrán los datos de los inmuebles que intervienen. Para ello se realizan las mismas operaciones que en el caso de uso anterior.

17. Eliminar una permuta:

```
"DELETE FROM `permutasBD`.`permuta` WHERE `idpermuta`
=value;"
```

18. Generar un contrato:

Con los datos de la permuta que ha sido aceptada y por tanto para la que se pretende imprimir dicho “contrato”, realizaremos las siguientes consultas.

En primer lugar, necesitamos los datos que recuperamos con la consulta:

```
"SELECT * FROM `permutasBD`.`permuta` WHERE `idpermuta`
=value;"
```

Una vez tenemos los datos, recuperamos el nombre y apellido del usuario pagador, en caso de haberlo, con la consulta:

```
"SELECT `nombre`,`apellidos` FROM `permutasBD`.`usuario`
WHERE `idusuario` =value;"
```

Y los datos de los usuarios e inmuebles involucrados.

Para obtener los datos del usuario solicitante, necesitamos realizar la siguiente consulta:

```
"SELECT * FROM `permutasBD`.`favoritos` WHERE `idfavoritos`
= value ;"
```

Una vez tenemos los datos del favorito con el que se generó la permuta podemos obtener el usuario propietario del inmueble implicado en la fila de la tabla “favoritos”, es decir, del inmueble que se solicitó para permutar.

```
"SELECT `usuario_idusuario` FROM `permutasBD`.`inmueble`
WHERE `idinmueble` =value;"
```

Por último, obtenemos los datos de ambos usuarios mediante la consulta:

```
"SELECT * FROM `permutasBD`.`usuario` WHERE `idusuario` =  
value;"
```

Donde *value* será el “usuario_idusuario” de la tabla “inmueble” obtenido de la operación anterior y después será el obtenido de la tabla “favoritos”.

Mientras que los inmuebles se obtendrán realizando la operación:

```
"SELECT * FROM `permutasBD`.`inmueble` WHERE `idinmueble` =  
value;"
```

Donde *value* será el identificador del inmueble de la fila de la tabla “permuta”, para después repetir la operación con el identificador del inmueble de la tabla “favoritos”.

3.4 Lógica del diseño

En primer lugar, el usuario registrado deberá añadir, al menos, un inmueble en la aplicación. De manera adicional al registro, el usuario podrá añadir las propiedades que desee. Una vez registrado, accederá iniciando sesión. Tras esto, podrá realizar las búsquedas con filtro o sin filtro de inmuebles necesarias para añadir inmuebles de otros usuario que cumplan sus requisitos a su lista de favoritos. Desde la lista de favoritos se generarán las solicitudes de permuta. Para generar la solicitud de permuta se requerirá de un inmueble que pertenezca a la lista de favoritos de un usuarios y de un inmueble propio del usuario. Una vez obtenidas las dos propiedades, la aplicación generará la solicitud que indicará cual será la cuantía que uno de los usuarios deberá pagar al otro, así como quién será el pagador, en caso de existir diferencia entre los valores de los inmuebles. Para calcular dicho valor y dicho usuario, se tendrá en cuenta el precio de mercado de ambos inmuebles, que es impuesto por cada propietario, junto con las hipotecas pendientes de ambos propietarios. De modo que mediante la siguiente ecuación se calculará la diferencia a aportar:

$$\text{Cuantía} = |(\text{precio1-hipoteca1})-(\text{precio2-hipoteca2})|$$

Una vez calculado, ambos usuarios podrán ponerse en contacto a través de mensajes que serán almacenados. En esos mensajes podrán realizarse contraofertas que impliquen la modificación de precios e hipotecas de los inmuebles involucrados, y por lo tanto una revalorización de la cuantía a aportar de la permuta.

Finalmente, la permuta podrá ser aceptada por ambas partes, por lo que se generará un “contrato”, es decir, un documento que recoge los datos, y a su vez se eliminarán los inmuebles incluidos en la permuta de la aplicación. Por el contrario, puede ser rechazada, y la solicitud de permuta se eliminará, sin eliminar los inmuebles de la aplicación.

Con respecto a la funcionalidad de permutas entre más de dos partes que se ha explicado en el apartado 1.4, no se ha implementado todavía. Sin embargo, tan solo sería necesario incluir una modificación para el cálculo por la diferencia del valor de los inmuebles involucrados. Como cada usuario implicado en la permuta circular será solicitado y solicitante, se tendrá que considerar tanto la diferencia por el valor del inmueble solicitado como la diferencia por el valor del inmueble solicitante.

Para explicar este cálculo, pondremos el mismo ejemplo que hemos utilizado en el apartado 1.4, es decir con las mismas permutas, pero esta vez valorando los inmuebles, por eso se incluyen unas flechas en color rojo que se encuentran en diferentes sentidos, indicando el flujo monetario. Tendremos en cuenta los

valores ya calculados de hacer la resta del precio de mercado menos la hipoteca asociada pendiente.

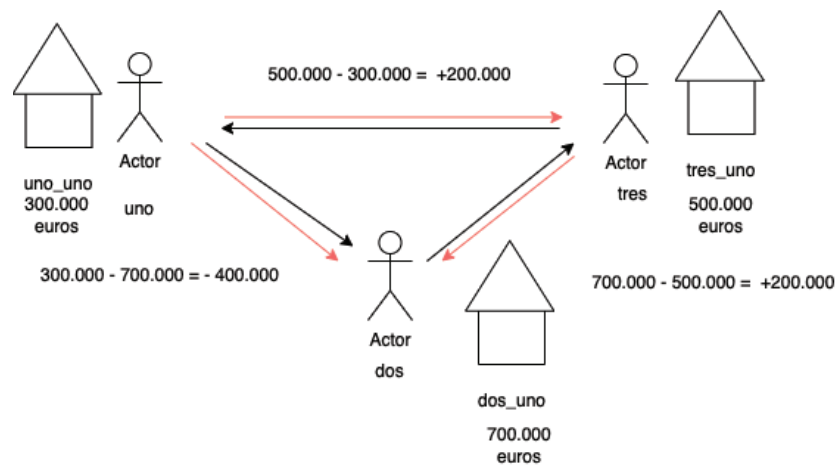


Figura 5. Permuta circular valorada.

En primer lugar, vemos que el inmueble “dos_uno” está valorado en 700.000 euros, el inmueble “tres_uno” en 500.000 y el “uno_uno” en 300.000. Vemos que la cuantía resultante de la permuta entre el actor “dos” y el actor “tres” es de 200.000 euros, esto significa que el actor “dos” se verá beneficiado en dicha cantidad, ya que se muda a una casa 200.000 más barata.

Podemos ver como el usuario “tres” deberá ser beneficiado con 200.000 euros por la misma razón, mientras que el actor “uno” deberá pagar 400.000 euros. Por lo que el único usuario deudor en este caso pagará 200.000 al usuario “tres” y otros 200.000 al usuario “dos”. De este modo, se solventa la transacción que sería necesaria entre los usuarios “dos” y “tres”.

4 Implementación

En este apartado se describe la metodología utilizada para el desarrollo de la aplicación. En concreto, en este proyecto se ha desarrollado el “back-end” de la aplicación PermutApp. Para ello, se ha diseñado el modelo de datos y se ha creado la base de datos que almacenará la información recolectada por la interfaz de usuario en los recursos previamente explicados. El software consistirá en una aplicación REST que deberá dar soporte a una interfaz de usuario.

4.1 Tecnologías empleadas

Para el desarrollo del “back-end” de la aplicación se ha implementado una API de REST. Se trata de una interfaz de programación de aplicaciones que utiliza la arquitectura REST. A través de ella, el usuario podrá hacer peticiones desde la interfaz de usuario para escribir, leer, modificar o eliminar información de la base de datos.

Se ha decidido implementar una API REST ya que permite el acceso a la base de datos desde diferentes servidores. De este modo, PermutApp podrá albergar numerosos usuarios que se conecten desde diferentes dispositivos.

El servicio REST (“Representational State Transfer”) es el tipo de arquitectura basado en sistemas distribuidos que se ha elegido para la implementación. Recibe la información por medio de http, en formato JSON a una URL [14]. JSON es un formato requerido por REST de texto ligero. Estas peticiones tienen un encabezado que contiene: el identificador de los recursos (URI), autorizaciones, las cookies e información acerca de los metadatos. La API devuelve una respuesta que también contiene un encabezado y un cuerpo, además de un código http de respuesta que indica el estado de la solicitud (OK, CREATED, INTERNAL_SERVER_ERROR, etc) [14].

Sigue una arquitectura cliente-servidor que está formada por clientes, servidores y recursos que se gestionan a través de solicitudes http. La comunicación entre el cliente y el servidor no almacena el estado, por lo que cada solicitud será independiente.

Para el desarrollo del “back-end” de la aplicación, se han utilizado las siguientes tecnologías:

- **IDE (entorno de desarrollo integrado) Eclipse:** Se ha utilizado la plataforma de desarrollo Eclipse. En ella se alberga el código y el servidor que ejecuta la aplicación. Se ha decidido utilizar Eclipse por su popularidad y facilidad para la creación de proyectos web dinámicos en java que se describirá a continuación [15]. Además, Eclipse incluye las herramientas que se necesitaban para el proyecto: Java EE y Maven.
- **Maven:** Se ha utilizado Maven como herramienta para la comprensión y gestión del proyecto software. Maven se basa en el concepto de un modelo de objetos de proyecto (POM). El archivo POM de los proyectos Maven permite al programador mucha flexibilidad a la hora de utilizar librerías para la implementación, ya que tan solo hay que añadirlas como dependencias en este archivo y se aplicará para todo el proyecto. Este proceso se explicará en el apartado 4.2.2. Por esta razón es una herramienta útil para poder compartir el proyecto, ya que los programadores se evitarán tiempos en configurar el entorno de desarrollo [16].
- **JDK (Java Development Kit):** Se ha elegido el lenguaje Java para todo el desarrollo del “back-end” de la aplicación, JavaSE-1.8. Se ha decido

utilizar este lenguaje debido a su popularidad y las facilidades que propone para programar una aplicación web dinámica en Eclipse utilizando Maven. Además, se trata así del lenguaje por excelencia que se ha desarrollado a lo largo del grado. También, por su posibilidad de ejecución en cualquier entorno compatible con java [18].

- **Jersey:** Se ha utilizado el “REST framework” Jersey que proporciona las especificaciones JAX-RS (@Path, @Get, @Post, @Consumes, @Produces, entre otras). Se trata de un “framework open source” de Eclipse que permite convertir los datos enviados y recibidos para poder ser leídos. Se trata así de una API JAX-RS estándar y portable. La totalidad del software se desarrollo entorno a esta librería. Se ha decidido utilizar Jersey por su sencillez y posibilidad de desplegarse en cualquier servidor de aplicaciones Java. En definitiva se ha utilizado porque hace posible el desarrollo de servicios RESTful de una manera más sencilla. Para ello se han añadido las siguientes dependencias al archivo POM [19].

```
<dependencies>
  <dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet</artifactId>
    <!-- use the following artifactId if you don't need servlet 2.x compatibility -->
    <!-- artifactId>jersey-container-servlet</artifactId -->
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.inject</groupId>
    <artifactId>jersey-hk2</artifactId>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-binding</artifactId>
  </dependency>

  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
    <version>8.0.13</version>
  </dependency>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.12.0</version>
  </dependency>
</dependencies>
```

Ilustración 12. Dependencias Maven.

- **Servidor de aplicaciones Apache Tomcat9:** Se ha utilizado “tomcat9” como contenedor web o “Java servlet”. Este software permite que el servidor web reciba contenido dinámico utilizando el protocolo http en Java. En nuestro caso funcionará como el propio servidor web sobre el que se ejecutará la aplicación. También, se pueden utilizar otros contenedores como Glassfish, sin embargo, Tomcat es más ligero pero con opción de añadir componentes externos. Por lo que ofrece más libertad y sencillez al programador [20].
- **Postman:** se ha utilizado la plataforma de APIs Postman para realizar las pruebas de nuestra API. Esta aplicación se ha utilizado para realizar las pruebas pertinentes a nuestra API a través de su interfaz gráfica de usuario [21].

Para la gestión de los recursos y el almacenamiento de las bases de datos se han utilizado las siguientes tecnologías:

- **SQL (Structured Query Language):** Se ha elegido el lenguaje SQL como lenguaje de definición, manipulación y control para realizar las consultas a una base de datos relacional. Se ha decidido utilizar una base de datos relacional ya que la aplicación solo necesitará almacenar información con una estructura definida. Además, SQL está entre los lenguajes de

manipulación de datos más populares y tiene una sintaxis universal y persiste en el tiempo, por lo que es sencillo aprender.

- **Mysql Workbench:** Se ha elegido MySQL como gestor para el modelado de la base de datos que almacena los recursos de la aplicación. MySQL es una aplicación de libre acceso, por lo que permite la gestión de la base de datos a nuevas aplicaciones. También, destaca por su seguridad, su escalabilidad y su velocidad [22].

4.2 Desarrollo

4.2.1 Base de datos

Para gestionar la base de datos que almacena los recursos de PermutApp se ha utilizado el gestor de SQL para bases de datos relacionales, MySQL Workbench. En primer lugar, se creó un nuevo modelo de datos. En él, se diseñó los metadatos, cada tabla y cada atributo, especificando su nombre y el tipo de datos que estos aceptan, además de otras especificaciones como las opciones de las claves primarias y las claves externas.

Una vez terminada la especificación de los metadatos, se creó el diagrama EER para determinar las relaciones entre los recursos.

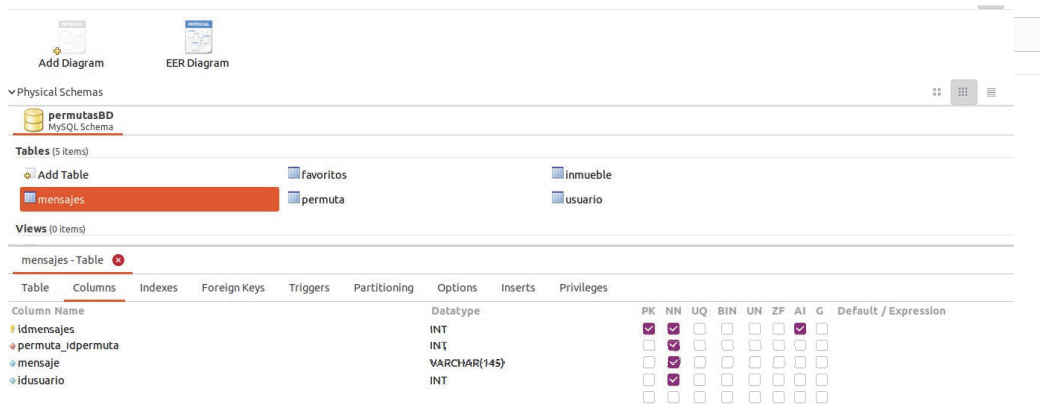


Ilustración 14. Especificación de metadatos.

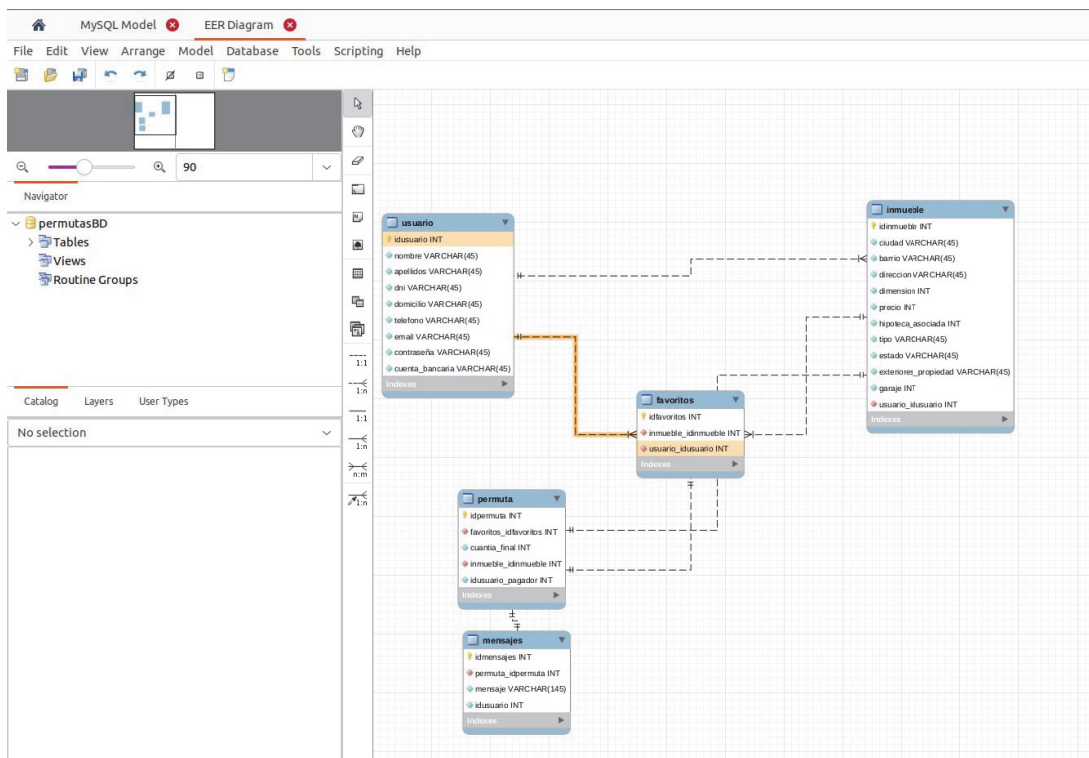


Ilustración 15. Modelo EER.

Tras finalizar con los metadatos, se exportó el “script” resultante del modelo. Para poder acceder a la base de datos, se debió crear una conexión con permisos para realizar las consultas desde la API. Para ello se añadió una nueva conexión a “localhost” con usuario: “root” y una contraseña.

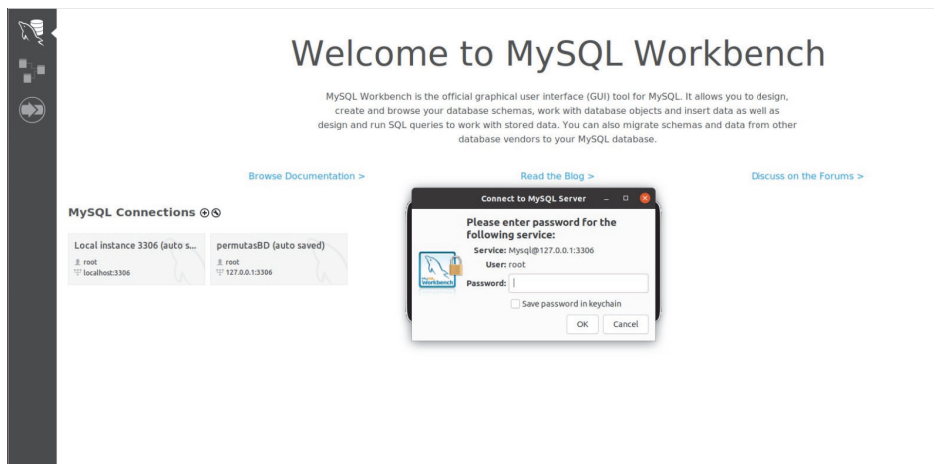
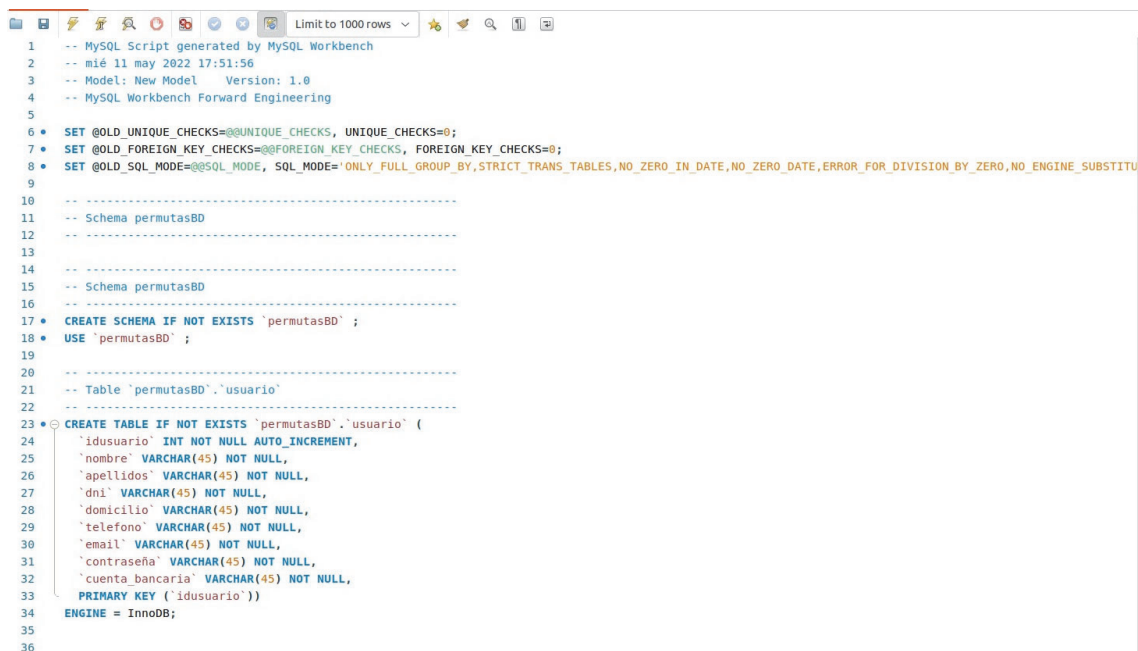


Ilustración 16. Conexión MySQL Workbench.

Tras iniciar sesión, se insertó el “script” con el nuevo modelo anteriormente creado. De este modo se originó la base de datos.



```
1 -- MySQL Script generated by MySQL Workbench
2 -- mié 11 may 2022 17:51:56
3 -- Model: New Model Version: 1.0
4 -- MySQL Workbench Forward Engineering
5
6 • SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
7 • SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
8 • SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
9
10 --
11 -- Schema permutasBD
12 --
13
14 --
15 -- Schema permutasBD
16 --
17 • CREATE SCHEMA IF NOT EXISTS `permutasBD` ;
18 • USE `permutasBD` ;
19
20 --
21 -- Table `permutasBD`.`usuario`
22 --
23 • CREATE TABLE IF NOT EXISTS `permutasBD`.`usuario` (
24   `idusuario` INT NOT NULL AUTO_INCREMENT,
25   `nombre` VARCHAR(45) NOT NULL,
26   `apellidos` VARCHAR(45) NOT NULL,
27   `dni` VARCHAR(45) NOT NULL,
28   `domicilio` VARCHAR(45) NOT NULL,
29   `telefono` VARCHAR(45) NOT NULL,
30   `email` VARCHAR(45) NOT NULL,
31   `contraseña` VARCHAR(45) NOT NULL,
32   `cuenta_bancaria` VARCHAR(45) NOT NULL,
33   PRIMARY KEY (`idusuario`))
34 ENGINE = InnoDB;
35
36
```

Ilustración 17. Script del Modelo de Datos en la conexión.

4.2.2 API REST

La API REST consistirá en un conjunto de operaciones CRUD (“create”, “read”, “update” and “delete”) por lo que el nombre que tendrá nuestra API será “crud”. Utiliza las operaciones CRUD ya que almacena la información de manera persistente en una base de datos relacional [23].

Para el desarrollo de la API REST utilizando las tecnologías explicadas, primero se creó un proyecto Maven en Eclipse. Durante el proceso de creación se añadió el filtro con el grupo: “Org.glassfish.jersey.archetypes” y el “artifact”: “Jersey-quickstart-webapp”.

También, fue necesario añadir Tomcat como nuevo servidor. En este caso se ha utilizado “Tomcat v9.0”.

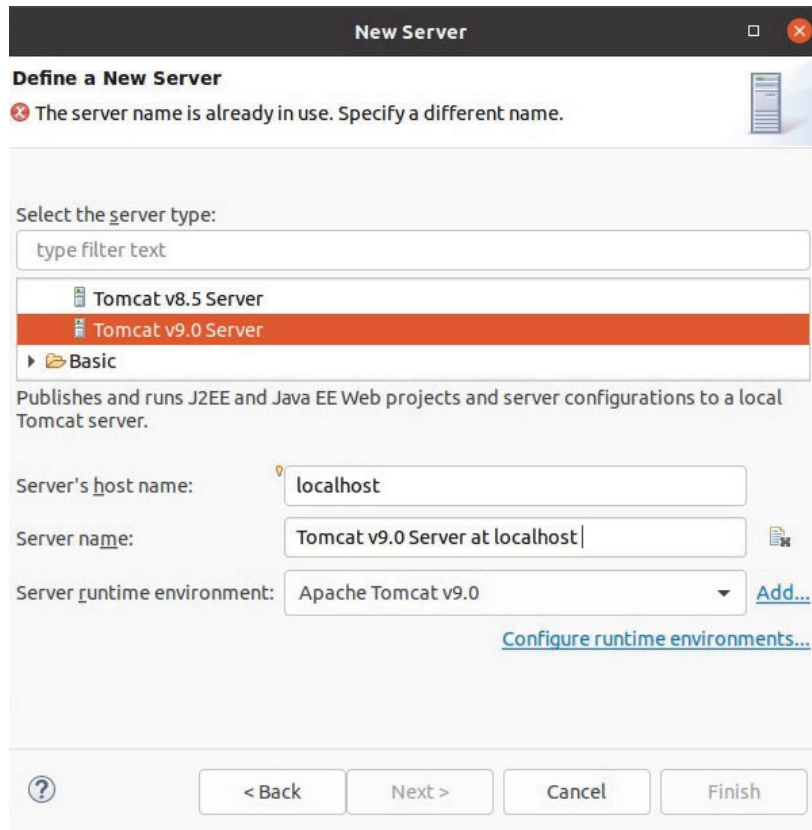


Ilustración 18. Definición de un nuevo servidor en Eclipse.

Una vez añadido, desde la pestaña servidores, se añadió el nuevo proyecto “crud” a los proyecto configurados para ser ejecutados desde el nuevo servidor.

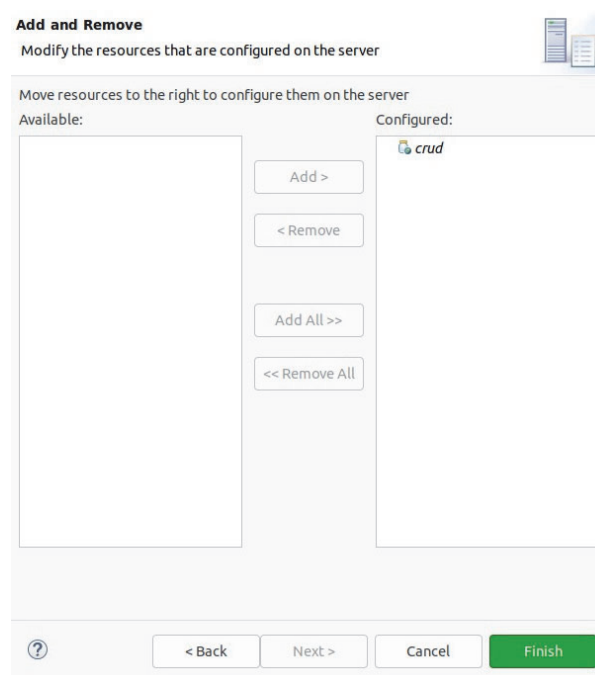


Ilustración 19. Modificación en los recursos configurados del servidor.

Al utilizar un proyecto Maven, fue necesario configurar el archivo POM. En la ilustración 12 podemos ver las dependencias utilizadas. La dependencia “mysql-connector” hará posible la conexión a la base de datos.

Para desarrollar la API se han creado las clases “resource” y “service”.

En la clase “resource” se han especificado las URI (Uniform Resource Identifier) y las especificaciones JAX-RS de cada método que llamará a su vez a métodos de la clase “service”.

Todos los métodos devuelven un tipo “Response” de la librería de “jersey”: “javax.ws.rs”, con el código http de respuesta y el cuerpo.

En el constructor de la clase “service” creamos la conexión a la base de datos del siguiente modo:

```
public crudservice() {  
    try {  
        String host = "localhost";  
        int port = 3306;  
        String db = "permutasBD";  
  
        String url = String.format("jdbc:mysql://%s:%d/%s?useSSL=false",host,port,db);  
        String uname = "root";  
        String pwd = "root";  
  
        Class.forName("com.mysql.cj.jdbc.Driver");  
        conn = DriverManager.getConnection(url,uname,pwd);  
    } catch(Exception e) {  
        System.out.println(e + "unsuccess connection");  
    }  
}
```

Ilustración 20. Conexión a la base de datos.

También, contiene los métodos que se referencian en la clase “resource”. En los métodos se realizarán las consultas SQL con las operaciones CRUD.

Por último, se han creado las clases para gestionar los recursos (usuario, inmueble, mensaje, permuta, favoritos).

También, una clase “contrato”, para la creación del contrato especificado en el requisito 18, y una clase “registro” para crear un objeto de registro que cuente con todos los datos necesarios para implementar el requisito 1.



Ilustración 21. Directorio del proyecto en Eclipse.

4.2.2.1 Diagrama UML

A continuación se detalla el diagrama UML del proyecto java.



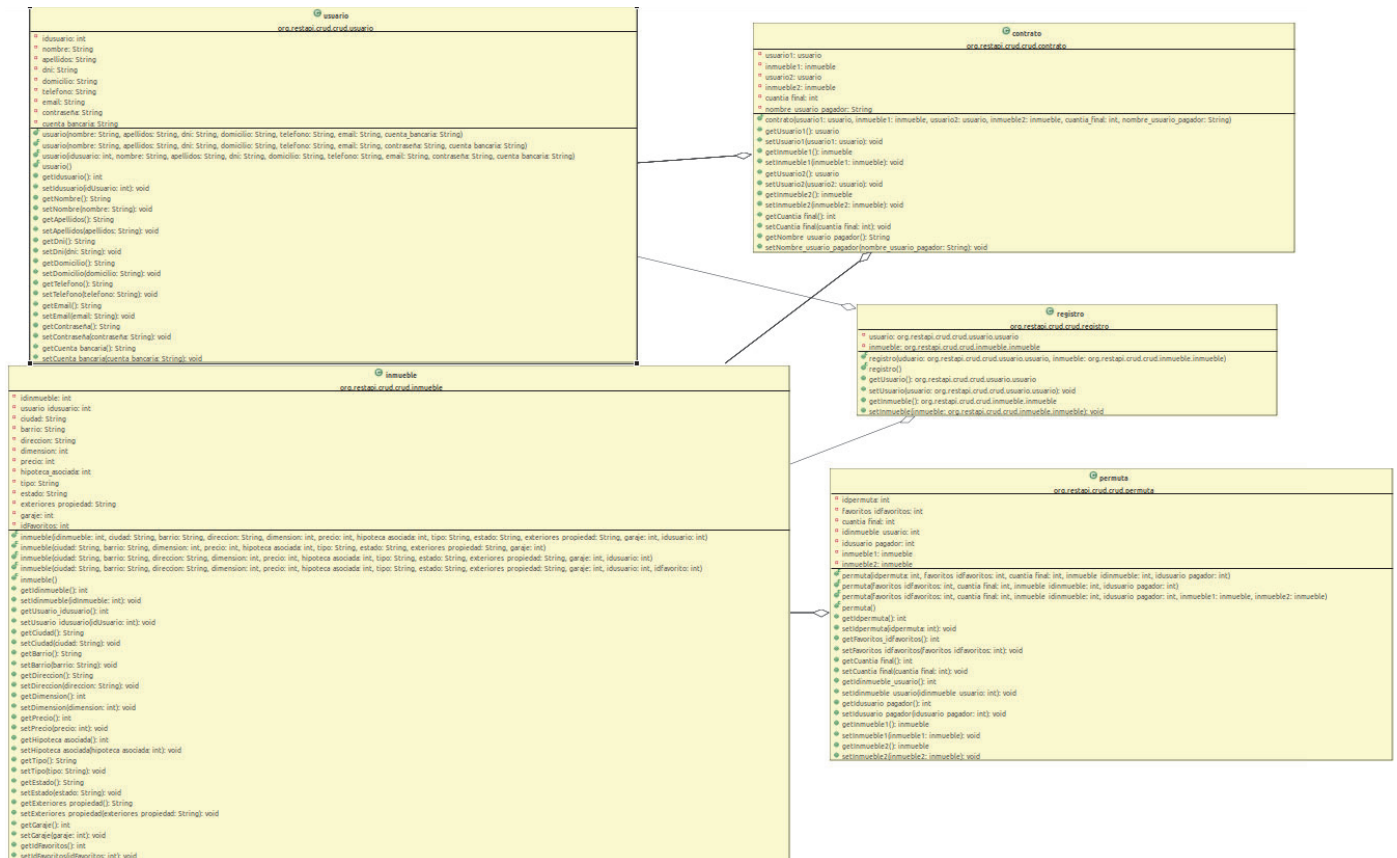


Figura 6. Diagrama UML del proyecto crud.

4.2.2.2 Verbos utilizados

Los métodos soportados por http utilizados son:

GET: para consultar la información de un recurso. Devuelve el estado de la respuesta de la petición y el cuerpo. El cuerpo contiene un JSON con los datos de los recursos requeridos en la petición.

POST: para crear un nuevo recurso. Devuelve el estado de la respuesta a la petición y el cuerpo. El cuerpo puede contener un identificador de un nuevo recurso añadido, o bien, un JSON con los datos de los recursos requeridos en la petición.

PUT: para modificar atributos de un recurso. Devuelve el estado de la respuesta de la petición.

DELETE: para eliminar un recurso determinado. Devuelve el estado de la respuesta de la petición.

4.2.2.3 Códigos http de respuesta

Las diferentes peticiones podrán devolver los siguientes códigos http que indicarán el estado de la respuesta de la petición:

200: OK

201: CREATED

404: NOT FOUND

415: UNSUPPORTED MEDIA TYPE

500: INTERNAL SERVER ERROR

4.2.2.4 Identificación de URI y patrones de identificación de recursos por requisito

1. **R01:** registro de un usuario.

Entrypoint: <http://localhost:8080/crud/webapi/permutoApp/registro>

URI	crud/webapi/permutoApp/registro
Método	POST
Cadena de consulta	
Cuerpo de la petición	Application/JSON con los datos del nuevo usuario a añadir "usuario": { "nombre": "Alvaro", "apellidos": "Lozano", "dni": "056443768G", "domicilio": "calle de la paz", "telefono": "623457689", "email": "alvarolozano@com", "contraseña": "1020", "cuenta_bancaria": "ES4334 5225 3443 2776 5665" }
Devuelve	201 y el identificador del usuario añadido si se ha creado 404 415 500 si ya existe un usuario registrado con ese email

Tabla 1. Petición de registro.

2. **R02:** login de un usuario.

Entrypoint: <http://localhost:8080/crud/webapi/permutoApp/login>

URI	crud/webapi/permutoApp/login
Método	POST
Cadena de consulta	
Cuerpo de la petición	Application/JSON con los datos del nuevo usuario que va a iniciar sesión { "email": "mariajimenez@com", "contraseña": "abcd" }
Devuelve	201 y el identificador del usuario que ha iniciado sesión 404 si el usuario o la contraseña son incorrectos 415 500

Tabla 2. Petición de login.

3. **R03:** visualizar perfil de un usuario.

Entrypoint:

<http://localhost:8080/crud/webapi/permutoApp/{idusuario}/perfil>

URI	crud/webapi/permutoApp/{idusuario}/perfil
Método	GET
Cadena de consulta	

Cuerpo de la petición	
Devuelve	200 y Application/JSON con los datos del usuario <pre> { "apellidos": "jiménez", "contraseña": "*****", "cuenta_bancaria": "ES2200 0123 0456 0000 0355", "dni": "9876554A", "domicilio": "calle 1", "email": "mariajimenez@com", "idusuario": 1, "nombre": "Maria", "telefono": "666898765" } </pre> 404 500 si el identificador del usuario es incorrecto

Tabla 3. Petición de perfil de usuario.

4. **R04:** modificar los datos de perfil de un usuario.

Entrypoint:

<http://localhost:8080/crud/webapi/permutoApp/modificarPerfil>

URI	crud/webapi/permutoApp/modificarPerfil
Método	POST
Cadena de consulta	
Cuerpo de la petición	Application/JSON con los datos del usuario que se quiera modificar y el idusuario <pre> { "idusuario": 5, "nombre": "", "apellidos": "", "dni": "", "domicilio": "", "telefono": "", "email": "alvarolozano99@com", "contraseña": "", "cuenta_bancaria": "" } </pre>
Devuelve	201 404 415 500 si el identificador del usuario es incorrecto o si ha intentado cambiar su email por uno que ya estaba registrado

Tabla 4. Petición de modificación de datos de perfil.

5. **R05:** añadir una propiedad.

Entrypoint:

<http://localhost:8080/crud/webapi/permutoApp/:idusuario/añadirInmueble>

URI	crud/webapi/permutApp/{idusuario}/añadirInmueble
Método	POST
Cadena de consulta	
Cuerpo de la petición	Application/JSON con los datos del inmueble que se quiere registrar <pre>{ "ciudad": "Madrid", "barrio": "COnde Orgaz", "direccion": "Avenida de los Madroños", "dimension": 500, "precio": 2000000, "hipoteca_asociada": 0, "tipo": "chalet", "estado": "buen estado", "exteriores_propiedad": "jardin", "garaje": 1 }</pre>
Devuelve	201 y el identificador del inmueble añadido 404 415 500

Tabla 5. Petición de registro de inmuebles.

6. **R06:** visualizar las propiedades de un usuario.

Entrypoint:

http://localhost:8080/crud/webapi/permutApp/:idusuario/propiedades

URI	crud/webapi/permutApp/{idusuario}/propiedades
Método	GET
Cadena de consulta	
Cuerpo de la petición	
Devuelve	200 y Application/JSON del listado de propiedades de un usuario <pre>[{ "barrio": "canillas", "ciudad": "Madrid", "dimension": 0, "direccion": "calle 1", "estado": "buen estado", "exteriores_propiedad": "terraza", "garaje": 0, "hipoteca_asociada": 30000, "idFavoritos": 0, "idinmueble": 1, "precio": 200000, "tipo": "piso", "usuario_idusuario": 1 }, { "barrio": "COnde Orgaz", "ciudad": "Madrid", "dimension": 0, "direccion": "Avenida de los Madroños", "estado": "buen estado", "exteriores_propiedad": "jardin", "garaje": 1, "hipoteca_asociada": 0, </pre>

	<pre> "idFavoritos": 0, "idinmueble": 7, "precio": 2000000, "tipo": "chalet", "usuario_idusuario": 1 }] 404 si el usuario no se encuentran propiedades de este usuario 500 </pre>
--	--

Tabla 6. Petición de visualización de propiedades de un usuario.

7. **R07:** realizar una búsqueda con o sin filtro de las propiedades registradas en la aplicación por todos los usuarios.

Entrypoint: <http://localhost:8080/crud/webapi/permutoApp/busqueda>

URI	crud/webapi/permutoApp/ permutoApp/busqueda
Método	GET
Cadena de consulta	
Cuerpo de la petición	Application/JSON con los datos de las opciones de filtrado <pre> { "ciudad": "Madrid", "barrio": "", "dimension": 100, "precio": 1000000, "hipoteca_asociada": 0, "tipo": "", "estado": "", "exteriores_propiedad": "", "garaje": 1 } </pre>
Devuelve	200 y Application/JSON del listado de propiedades resultado de la búsqueda <pre> [{ "barrio": "canillas", "ciudad": "Madrid", "dimension": 0, "direccion": "calle 1", "estado": "buen estado", "exteriores_propiedad": "terraza", "garaje": 0, "hipoteca_asociada": 30000, "idFavoritos": 0, "idinmueble": 1, "precio": 200000, "tipo": "piso", "usuario_idusuario": 1 }, { "barrio": "Goya", "ciudad": "Madrid", "dimension": 0, "direccion": "calle Goya", "estado": "buen estado", "exteriores_propiedad": "terraza", "garaje": 0, "hipoteca_asociada": 0, "idFavoritos": 0, "idinmueble": 3, </pre>

	<pre>"precio": 850000, "tipo": "piso", "usuario_idusuario": 3 }]</pre> 404 500
--	--

Tabla 7. Petición de búsqueda.

8. **R08:** añadir una propiedad a la lista de favoritos de un usuario.

Entrypoint:

http://localhost:8080/crud/webapi/permutApp/:idusuario/añadirFavorito/:idinmuelle

URI	crud/webapi/permutApp/{idusuario}/añadirFavorito/:idinmuelle
Método	POST
Cadena de consulta	
Cuerpo de la petición	
Devuelve	201 y el identificador del favorito creado que contiene el inmueble 404 500

Tabla 8. Petición de creación de favoritos.

9. **R09:** visualizar lista de inmuebles favoritos de un usuario

Entrypoint:

http://localhost:8080/crud/webapi/permutApp/:idusuario/favoritos

URI	crud/webapi/permutApp/{idusuario}/favoritos
Método	GET
Cadena de consulta	
Cuerpo de la petición	
Devuelve	200 y Application/JSON con el listado de los favoritos del usuario <pre>[{ "barrio": "chamartín", "ciudad": "Madrid", "dimension": 0, "direccion": "calle 14", "estado": "necesario reforma", "exteriores_propiedad": "", "garaje": 0, "hipoteca_asociada": 20000, "idFavoritos": 4, "idinmueble": 4, "precio": 250000, "tipo": "piso", "usuario_idusuario": 4 }, { "barrio": "canillas", "ciudad": "Madrid", "dimension": 0, "direccion": "calle 1", "estado": "buen estado", "exteriores_propiedad": "terrazza",</pre>

	<pre> "garaje": 0, "hipoteca_asociada": 30000, "idFavoritos": 7, "idinmueble": 1, "precio": 200000, "tipo": "piso", "usuario_idusuario": 1 }] 404 500 </pre>
--	---

Tabla 9. Petición de visualización de favoritos de un usuario.

10. **R10:** generar permuta desde la lista de favoritos.

Entrypoint:

http://localhost:8080/crud/webapi/permutoApp/:idfavoritos/generarPermuta/{idinmueble}

URI	crud/webapi/permutoApp/{idfavoritos}/generarPermuta/{idinmueble}
Método	POST
Cadena de consulta	
Cuerpo de la petición	
Devuelve	201 y el identificador de la permuta creado que contiene el inmueble 404 500 si el usuario que quiere generar la permuta no dispone de inmuebles

Tabla 10. Petición de creación de solicitud de permuta.

11. **R11:** almacenar mensaje de una permuta.

Entrypoint:

http://localhost:8080/crud/webapi/permutoApp/:idpermuta/añadirMensaje

URI	crud/webapi/permutoApp/{idpermuta}/añadirMensaje
Método	POST
Cadena de consulta	
Cuerpo de la petición	Application/JSON con el cuerpo del mensaje y el identificador del usuario que lo envía <pre> { "mensaje": "Estoy interesada en su inmueble de chamartin", "idusuario": 1 } </pre>
Devuelve	201 y el identificador del mensaje creado 404 415 500

Tabla 11. Petición de almacenamiento de mensajes por un usuario.

12. **R12:** consulta de las permutas recibidas por un usuario.

Entrypoint:

http://localhost:8080/crud/webapi/permutoApp/:idusuario/permutasSolicitadas

URI	crud/webapi/permutoApp/{idusuario}/permutasSolicitadas
-----	--

Método	GET
Cadena de consulta	
Cuerpo de la petición	
Devuelve	200 y Application/JSON con las permutas que se le hayan solicitado <pre>[{ "cuantia_final": 1770000, "favoritos_idfavoritos": 3, "idinmuelle_usuario": 7, "idpermuta": 2, "idusuario_pagador": 4, "inmueble1": { "barrio": "COnde Orgaz", "ciudad": "Madrid", "dimension": 0, "direccion": "Avenida de los Madroños", "estado": "buen estado", "exteriores_propiedad": "jardin", "garaje": 1, "hipoteca_asociada": 0, "idFavoritos": 0, "idinmuelle": 7, "precio": 2000000, "tipo": "chalet", "usuario_idusuario": 1 }, "inmueble2": { "barrio": "chamartín", "ciudad": "Madrid", "dimension": 0, "direccion": "calle 14", "estado": "necesario reforma", "exteriores_propiedad": "", "garaje": 0, "hipoteca_asociada": 20000, "idFavoritos": 0, "idinmuelle": 4, "precio": 250000, "tipo": "piso", "usuario_idusuario": 4 } }, { "cuantia_final": 620000, "favoritos_idfavoritos": 4, "idinmuelle_usuario": 3, "idpermuta": 3, "idusuario_pagador": 4, "inmueble1": { "barrio": "Goya", "ciudad": "Madrid", "dimension": 0, "direccion": "calle Goya", "estado": "buen estado", "exteriores_propiedad": "terraza", "garaje": 0, "hipoteca_asociada": 0, "idFavoritos": 0, "idinmuelle": 3, "precio": 850000, "tipo": "piso", </pre>

	<pre> "usuario_idusuario": 3 }, "inmueble2": { "barrio": "chamartín", "ciudad": "Madrid", "dimension": 0, "direccion": "calle 14", "estado": "necesario reforma", "exteriores_propiedad": "", "garaje": 0, "hipoteca_asociada": 20000, "idFavoritos": 0, "idinmueble": 4, "precio": 250000, "tipo": "piso", "usuario_idusuario": 4 } } } 404 500 </pre>
--	---

Tabla 12. Petición de visualización de permutas solicitadas a un usuario.

13. **R13:** visualizar los mensajes asociados a una permuta ordenados por tiempo de envío.
 Entrypoint:
<http://localhost:8080/crud/webapi/permutoApp/permuta/:idpermuta/mensajes>

URI	crud/webapi/permutoApp/permuta/{idpermuta}/mensajes
Método	GET
Cadena de consulta	
Cuerpo de la petición	
Devuelve	200 y Application/JSON con los mensajes almacenados por permuta <pre> [{ "idmensaje": 6, "idpermuta": 2, "idusuario": 1, "mensaje": "Estoy interesada en su inmueble de chamartin" }, { "idmensaje": 7, "idpermuta": 2, "idusuario": 4, "mensaje": "Yo también estoy interesado en su inmueble del Conde Orgaz pero creo que debería bajar el precio 100.000 euros" }] 404 500 </pre>

Tabla 13. Petición de visualización de mensajes relacionados a una permuta.

14. **R14:** eliminar una permuta.
 Entrypoint:
<http://localhost:8080/crud/webapi/permutoApp/eliminarPermuta/:idpermuta>

URI	Crud/webapi/permutoApp/eliminarPermuta/{idpermuta}
Método	DELETE
Cadena de consulta	
Cuerpo de la petición	
Devuelve	200 y el identificador de la permuta eliminada 404 500 si no se encuentra la permuta en la base de datos

Tabla 14. Petición de eliminación de una permuta.

15. **R15:** modificar el precio o la hipoteca asociada a un inmueble

Precio:

Entrypoint:

<http://localhost:8080/crud/webapi/permutoApp/:idinmueble/modificarPrecio>

URI	Crud/webapi/permutoApp/:idinmueble/modificarPrecio
Método	PUT
Cadena de consulta	Nuevo precio
Cuerpo de la petición	
Devuelve	200 404 500 si no se encuentra el inmueble en la base de datos

Tabla 15. Petición de modificación del precio de un inmueble.

Hipoteca:

Entrypoint:

<http://localhost:8080/crud/webapi/permutoApp/:idinmueble/modificarHipoteca>

URI	Crud/webapi/permutoApp/:idinmueble/modificarHipoteca
Método	PUT
Cadena de consulta	Nueva hipoteca asociada
Cuerpo de la petición	
Devuelve	200 404 500 si no se encuentra el inmueble en la base de datos

Tabla 16. Petición de modificación de la hipoteca asociada a un inmueble.

16. **R16:** eliminar un inmueble.

Entrypoint:

<http://localhost:8080/crud/webapi/permutoApp/eliminarInmueble/:idinmueble>

URI	Crud/webapi/permutoApp/eliminarInmueble/{idinmueble}
Método	DELETE
Cadena de consulta	

Cuerpo de la petición	
Devuelve	200 404 500 si no se encuentra el inmueble en la base de datos

Tabla 17. Petición de eliminación de un inmueble.

17. **R17:** visualizar permutas enviadas por un usuario.

Entrypoint:

http://localhost:8080/crud/webapi/permutoApp/:idusuario/permutas

URI	Crud/webapi/permutoApp/{idusuario}/permutas
Método	GET
Cadena de consulta	
Cuerpo de la petición	
Devuelve	200 y Application/JSON con las permutas enviadas por el usuario <pre>[{ "cuantia_final": 620000, "favoritos_idfavoritos": 4, "idinmueble_usuario": 3, "idpermuta": 3, "idusuario_pagador": 4, "inmueble1": { "barrio": "Goya", "ciudad": "Madrid", "dimension": 0, "direccion": "calle Goya", "estado": "buen estado", "exteriores_propiedad": "terraza", "garaje": 0, "hipoteca_asociada": 0, "idFavoritos": 0, "idinmueble": 3, "precio": 850000, "tipo": "piso", "usuario_idusuario": 3 }, "inmueble2": { "barrio": "chamartin", "ciudad": "Madrid", "dimension": 0, "direccion": "calle 14", "estado": "necesario reforma", "exteriores_propiedad": "", "garaje": 0, "hipoteca_asociada": 20000, "idFavoritos": 0, "idinmueble": 4, "precio": 250000, "tipo": "piso", "usuario_idusuario": 4 } }]</pre> 404 500

Tabla 18. Petición de visualización de permutas enviadas por un usuario.

18.**R18**: generar un contrato con los datos de la permuta.

Entrypoint:

http://localhost:8080/crud/webapi/permutApp/contrato/:idpermuta

URI	Crud/webapi/permutApp/contrato/{idpermuta}
Método	GET
Cadena de consulta	
Cuerpo de la petición	
Devuelve	200 y Application/JSON con el contrato { "cuantia_final": 500000, "inmueble1": { "barrio": "Goya", "ciudad": "Madrid", "dimension": 0, "direccion": "calle Goya", "estado": "buen estado", "exteriores_propiedad": "terraza", "garaje": 0, "hipoteca_asociada": 0, "idFavoritos": 0, "idinmueble": 3, "precio": 850000, "tipo": "piso", "usuario_idusuario": 3 }, "inmueble2": { "barrio": "malasaña", "ciudad": "Madrid", "dimension": 0, "direccion": "calle de la paz", "estado": "buen estado", "exteriores_propiedad": "", "garaje": 0, "hipoteca_asociada": 0, "idFavoritos": 0, "idinmueble": 5, "precio": 350000, "tipo": "ático", "usuario_idusuario": 5 }, "nombre_usuario_pagador": "Alvaro Lozano", "usuario1": { "apellidos": "Gil", "contraseña": "*****", "cuenta_bancaria": "ES8844 2500 2300 2806 9045", "dni": "24678933B", "domicilio": "calle 13", "email": "juangil@com", "idusuario": 3, "nombre": "Juan", "telefono": "679934756" }, "usuario2": { "apellidos": "Lozano", "contraseña": "*****", "cuenta_bancaria": "ES4334 5225 3443 2776 5665", "dni": "056443768G", "domicilio": "calle de la paz", "email": "alvarolozano99@com", } }

	<pre> {idusuario": 5, "nombre": "Alvaro", "telefono": "623457689" } 404 500 </pre>
--	--

Tabla 19. Petición de generación de un "contrato".

5 Pruebas

En este apartado se recogen las pruebas que verifican el funcionamiento de cada requisito especificado en el apartado 2.2. Para llevar a cabo las pruebas se ha utilizado la herramienta Postman.

1. R01: Registro de un usuario.

En primer lugar, será necesario realizar una petición a nuestra API con el usuario y el inmueble. Para poder registrar un usuario será obligatorio registrar un inmueble. Para ello se escribirán los datos en formato JSON, con todos los datos incluidos.

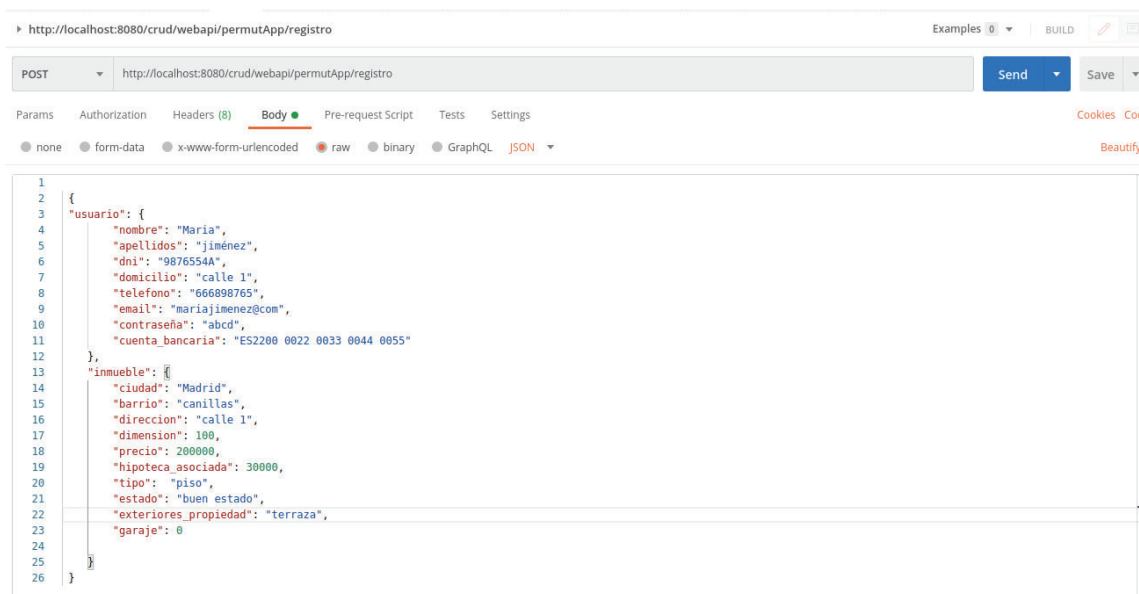


Ilustración 22. Prueba de registro.

La petición será un POST, ya que generará una nueva fila a la tabla usuario y a la tabla inmueble de la base de datos.

Si el usuario no estaba registrado en la base de datos, se consiguen los siguientes resultados en las tablas:

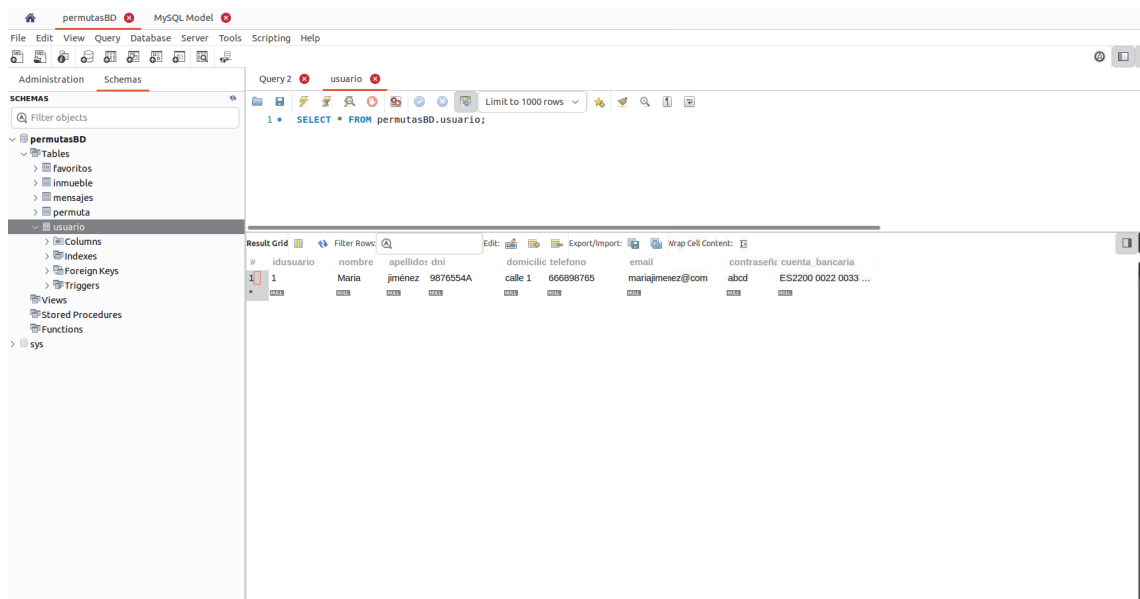


Ilustración 23. Tabla usuario tras registrar el primer usuario.

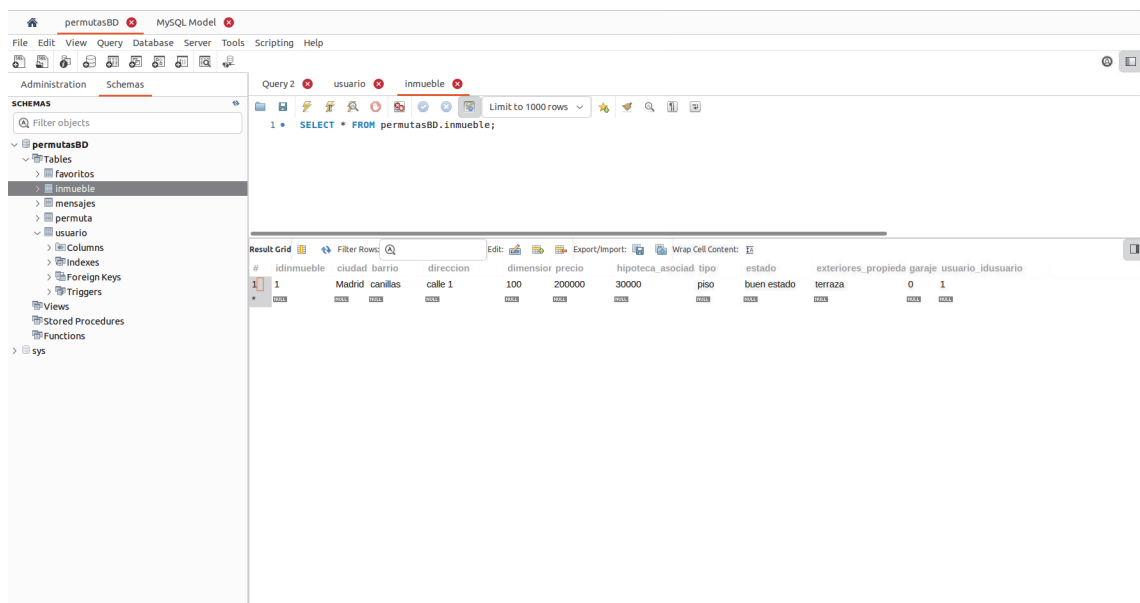


Ilustración 24. Tabla inmueble tras registrar el primer usuario.

Finalmente, tras varios registros de usuarios, se devuelven las tablas:

#	idusuario	nombre	apellidos	dni	domicilio	telefono	email	contraseña	cuenta_bancaria
1	1	Maria	jiménez	9876554A	calle 1	666898765	mariajimenez@com	abcd	ES2200 0022 0033 ...
2	2	Carlos	Gómez	2356554B	calle 22	675943545	carlosgomez@com	1234	ES3600 0023 2653 ...
3	3	Juan	Gil	24678933B	calle 13	679934756	juangil@com	0000	ES8844 2500 2300 ...
4	4	Mariana	García	05669873G	calle 14	688933755	marianagarcia@com	0220	ES8486 2520 3340 ...
5	5	Alvaro	Lozano	056443768G	calle de la paz	623457689	alvarolozano@com	1020	ES4334 5225 3443 ...

Tabla 20. Tabla usuario.

#	id_inmueble	ciudad	barrio	direccion	dimension	precio	hipoteca_asociada	tipo	estado	exteriores_propiedad	garaje	usuario_id_usuario
1	1	Madrid	canillas	calle 1	100	200000	30000	piso	buen estado	terrazza	0	1
2	2	Madrid	Arturo Soria	calle Arturo Soria	180	700000	70000	piso	buen estado	terrazza	1	2
3	3	Madrid	Goya	calle Goya	160	850000	0	piso	buen estado	terrazza	0	3
4	4	Madrid	chamartín	calle 14	80	250000	20000	piso	necesario ref...		0	4
5	5	Madrid	malasaña	calle de la paz	70	350000	0	ático	buen estado		0	5
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 25. Tabla inmueble.

Mientras que si el usuario ya estaba registrado devolverá un código http 500 con el siguiente mensaje:

The screenshot shows a REST client interface with the following details:

- URL:** http://localhost:8080/crud/webapi/permutApp/registro
- Method:** POST
- Body (JSON):**

```

{
  "usuario": {
    "nombre": "Alvaro",
    "apellidos": "Lozano",
    "dni": "0564437686",
    "domicilio": "calle de la paz",
    "telefono": "623457689",
    "email": "alvarolozano@com",
    "contraseña": "1020",
    "cuenta_bancaria": "ES4334 5225 3443 2776 5665"
  },
  "inmueble": {
    "ciudad": "Madrid",
    "barrio": "malasaña",
    "direccion": "calle de la paz",
    "dimension": 70,
    "precio": 350000,
    "hipoteca_asociada": 0,
    "tipo": "ático",
    "estado": "buen estado",
    "exteriores_propiedad": "",
    "garaje": 0
  }
}

```
- Status:** 500 Internal Server Error
- Response Body:**

```

1 no se ha podido realizar el registro con esos datos

```

Ilustración 26. Prueba errónea de registro.

2. R02: Login de un usuario.

La petición que deberá enviarse a la API será una petición POST que contenga como cuerpo un JSON con el email y la contraseña del usuario que desea iniciar sesión.

Si el email y la contraseña son correctos, se devolverá el identificador del usuario que ha iniciado sesión.

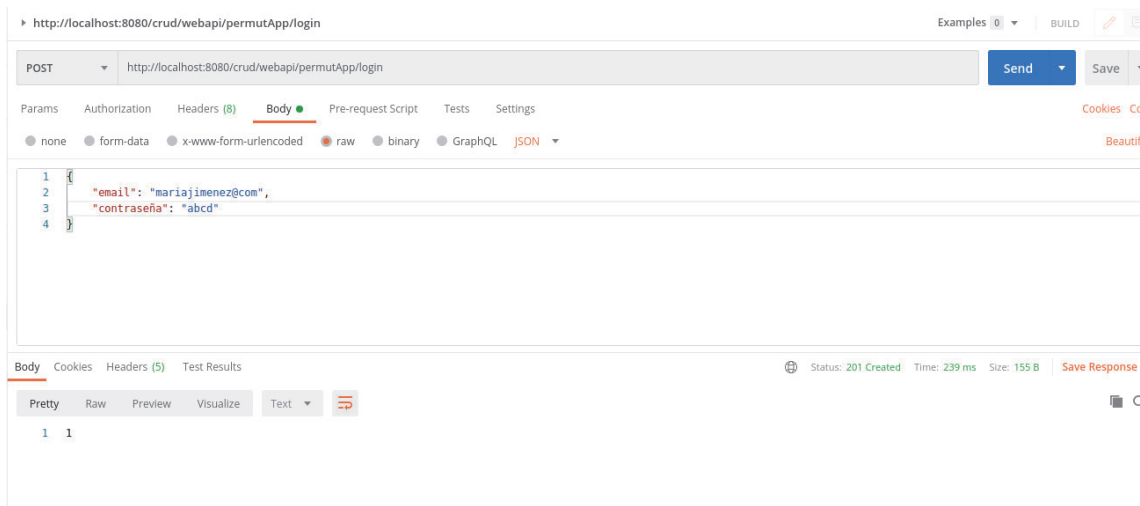


Ilustración 27. Prueba de login.

Por el contrario, si el email o la contraseña son incorrectos, es decir, no se encuentran en la tabla usuario o no pertenecen a la misma, se devolverá un código http 404 con el siguiente mensaje:

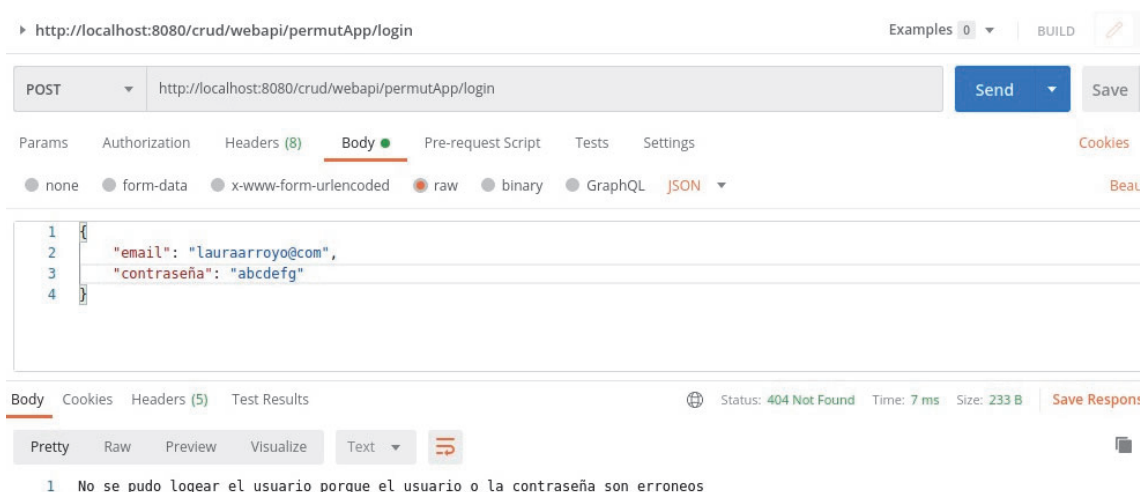


Ilustración 28. Prueba de login errónea.

3. R03: Consultar los datos de un usuario.

Se necesitará un petición GET con el identificador del usuario del que se quieren visualizar los datos. Esta petición devolverá un JSON en su cuerpo, con la información del usuario de la siguiente manera:

URL: `http://localhost:8080/crud/webapi/permutApp/idusuario/perfil`

Method: `GET`

Params: Authorization, Headers (6), Body, Pre-request Script, Tests, Settings

Query Params:

KEY	VALUE	DESCRIPTION
Idusuario	1	Description

Path Variables:

KEY	VALUE	DESCRIPTION
Idusuario	1	Description

Status: 200 OK Time: 15 ms Size: 371 B

Body (JSON):

```

1 {
2   "apellidos": "jiménez",
3   "contraseña": "*****",
4   "cuenta_bancaria": "ES2200 0022 0033 0044 0055",
5   "dni": "987654A",
6   "domicilio": "calle 1",
7   "email": "mariajimenez@com",
8   "idusuario": 1,
9   "nombre": "Maria",
10  "telefono": "666898765"
11 }

```

Ilustración 29. Prueba de consulta de datos de usuario.

Mientras que si el usuario no está en la base de datos devolverá un código 500 con el siguiente mensaje:

URL: `http://localhost:8080/crud/webapi/permutApp/idusuario/perfil`

Method: `GET`

Params: Authorization, Headers (6), Body, Pre-request Script, Tests, Settings

Query Params:

KEY	VALUE	DESCRIPTION
Idusuario	8	Description

Path Variables:

KEY	VALUE	DESCRIPTION
Idusuario	8	Description

Status: 500 Internal Server Error Time: 10 ms Size: 182 B

Body (JSON):

```

1 error al mostrar datos de usuario: 8

```

Ilustración 30. Prueba errónea de consulta de datos de usuario.

4. R04: Modificar los datos de un usuario.

Solo podrán modificarse el domicilio, teléfono, email, contraseña y cuenta bancaria. Para ello se realizará una petición POST con los datos del usuario que se deseen modificar. En el caso de modificar el email, no podrá introducirse un email ya registrado en la aplicación. Será necesario que el JSON que contiene los datos contenga el identificador del usuario. En caso de que la petición se envíe correctamente, devolverá el siguiente mensaje:

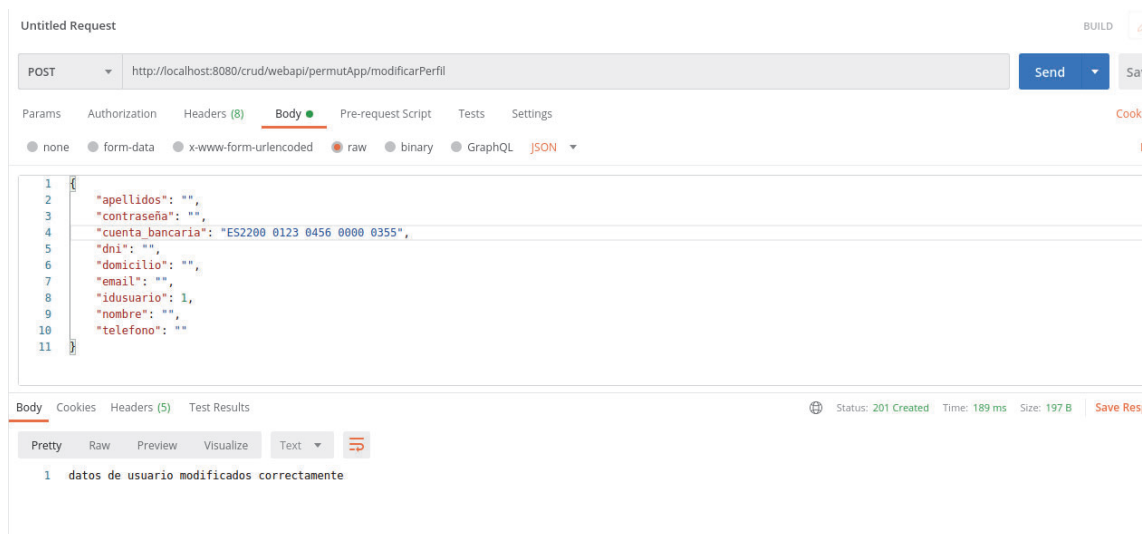


Ilustración 31. Prueba de modificación de datos de usuario.

#	idusuario	nombre	apellidos	dni	domicilio	telefono	email	contraseña	cuenta_bancaria
1	1	Maria	Jiménez	9876554A	calle 1	666898765	mariajimenez@com	abcd	ES2200 0123 0456 ...
2	2	Carlos	Gómez	2356554B	calle 22	675943545	carlosgomez@com	1234	ES3600 0023 2653 ...
3	3	Juan	Gil	24678933B	calle 13	679934756	juangil@com	0000	ES8844 2500 2300 ...
4	4	Mariana	García	05669873G	calle 14	688933755	marianagarcia@com	0220	ES8486 2520 3340 ...
5	5	Alvaro	Lozano	056443768G	calle de la paz	623457689	alvarolozano99@com	1020	ES4334 5225 3443 ...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 32. Tabla usuario tras la modificación del usuario 1.

En este caso el usuario ha decidido modificar la cuenta bancaria. Mientras que si los datos son incorrectos o el identificador del usuario no se encuentra, devolverá:

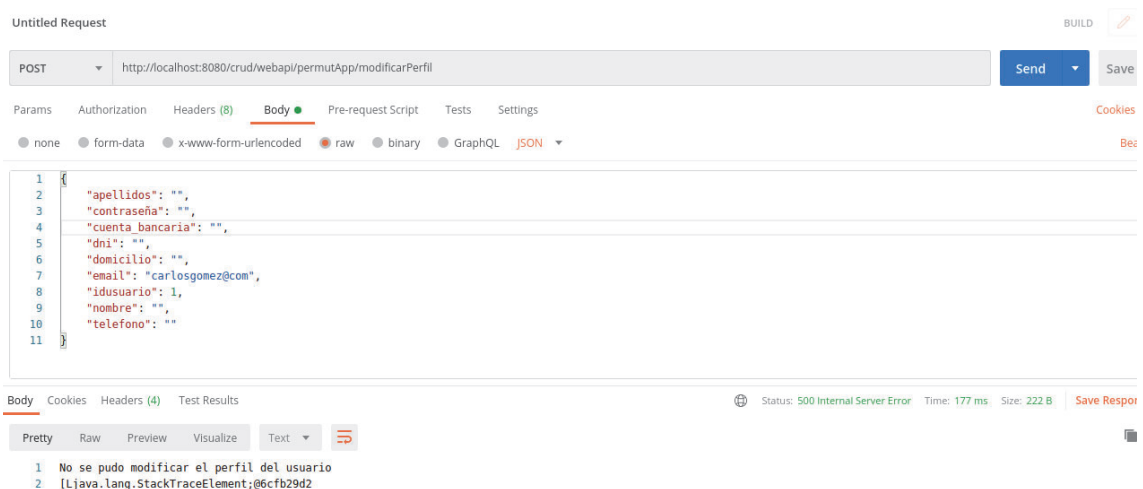


Ilustración 33. Prueba errónea de modificación de datos de usuario.

En este caso el usuario 1 ha intentado cambiar su email por uno ya registrado de otro usuario.

5. R05: Registrar una propiedad.

Será necesario una petición POST que incluya en su cuerpo un JSON con todos los datos necesarios para registrar un inmueble y un

parámetro con el identificador del usuario propietario. En caso de éxito devolverá el identificador el inmueble recién registrado.

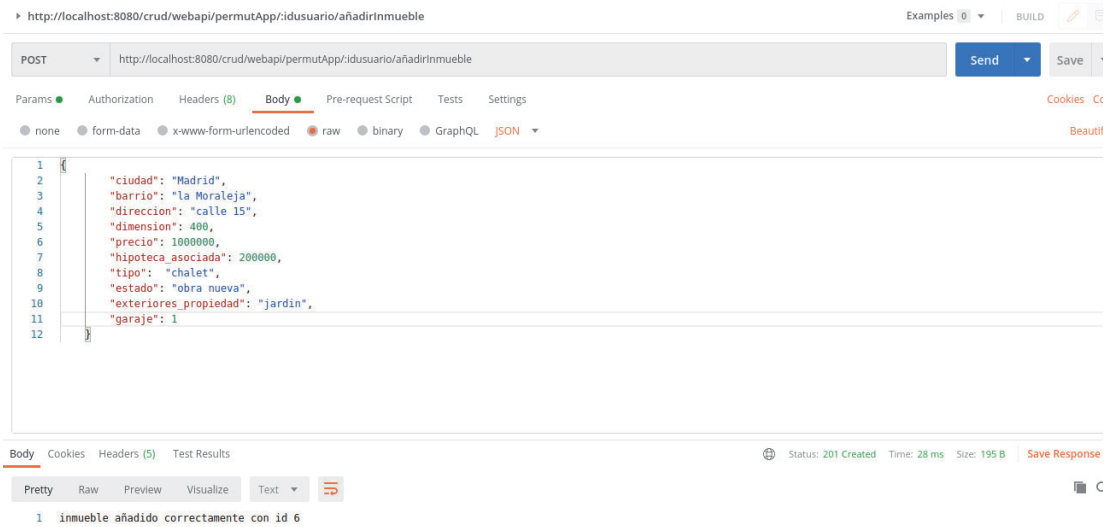


Ilustración 34. Prueba de registro de un inmueble.

#	idinmueble	ciudad	barrio	direccion	dimension	precio	hipoteca_asociad	tipo	estado	exteriores_propieda	garaje	usuario_idusuario
1	1	Madrid	canillas	calle 1	100	200000	30000	piso	buen estado	terrazza	0	1
2	2	Madrid	Arturo Soria	calle Arturo Soria	180	700000	70000	piso	buen estado	terrazza	1	2
3	3	Madrid	Goya	calle Goya	160	850000	0	piso	buen estado	terrazza	0	3
4	4	Madrid	chamartín	calle 14	80	250000	20000	piso	necesario ref...		0	4
5	5	Madrid	malasaña	calle de la paz	70	350000	0	ático	buen estado		0	5
6	6	Madrid	la Moraleja	calle 15	400	1000000	200000	chalet	obra nueva	jardin	1	1

Ilustración 35. Tabla inmueble con un nuevo inmueble.

Mientras que si el identificador del usuario no se encuentra en la base de datos o el cuerpo de la petición es incorrecto, devolverá un código 500 http.

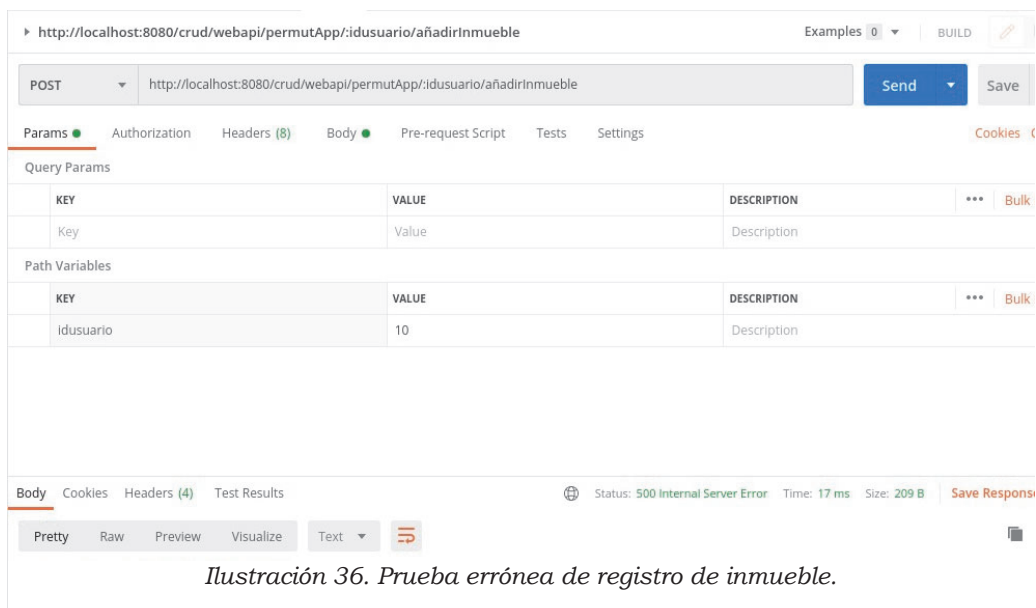
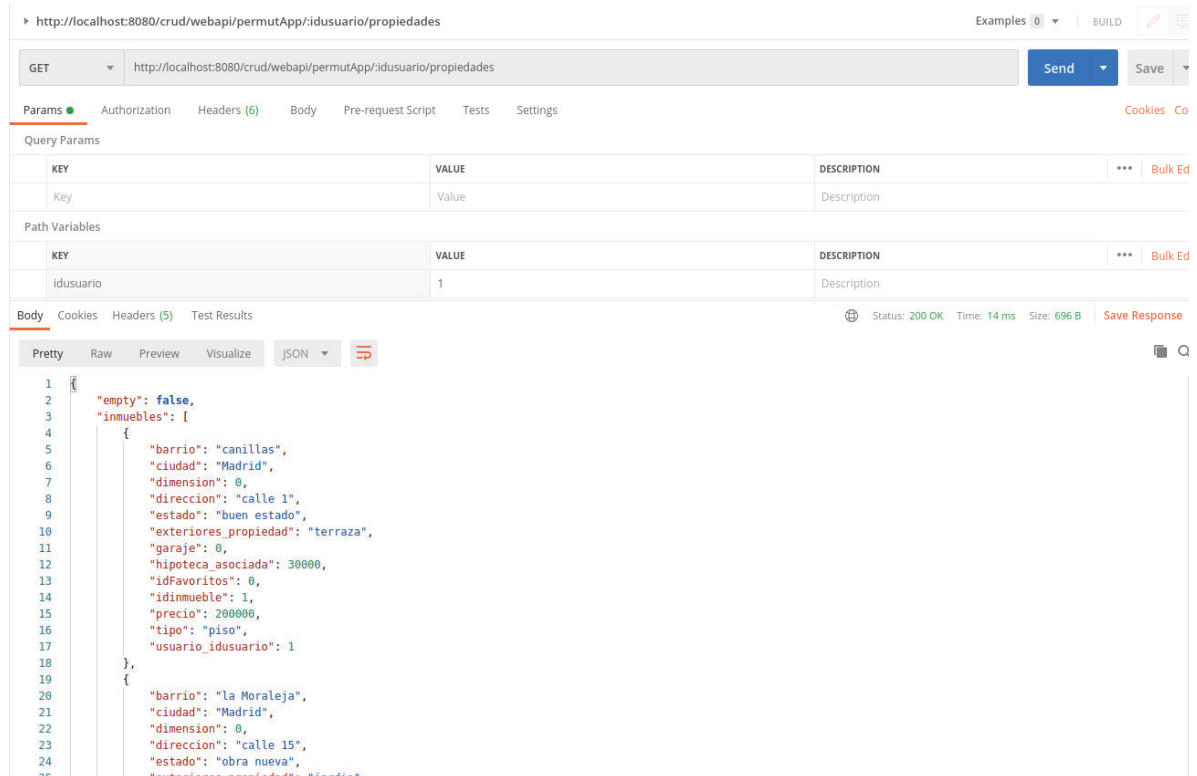


Ilustración 36. Prueba errónea de registro de inmueble.

6. R06: Consultar las propiedades de un usuario.

Para poder visualizar las propiedades será necesario una petición GET con el identificador del propietario de los inmuebles. Si la petición es exitosa devolverá:



URL: `http://localhost:8080/crud/webapi/permutApp/idusuario/propiedades`

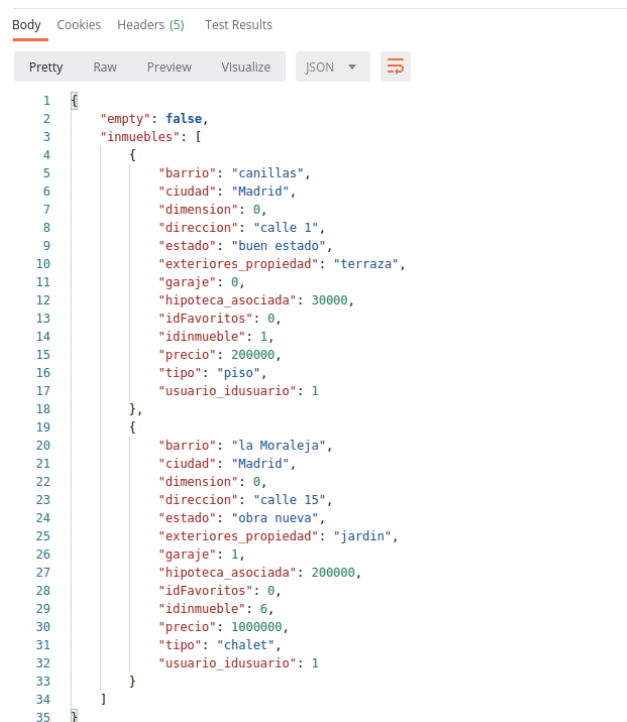
Method: `GET`

Path Variables:

KEY	VALUE	DESCRIPTION
idusuario	1	Description

Body:

```
{
  "empty": false,
  "inmuebles": [
    {
      "barrio": "canillas",
      "ciudad": "Madrid",
      "dimension": 0,
      "direccion": "calle 1",
      "estado": "buen estado",
      "exteriores_propiedad": "terrazza",
      "garaje": 0,
      "hipoteca_asociada": 30000,
      "idFavoritos": 0,
      "idinmuelle": 1,
      "precio": 200000,
      "tipo": "piso",
      "usuario_idusuario": 1
    },
    {
      "barrio": "la Moraleja",
      "ciudad": "Madrid",
      "dimension": 0,
      "direccion": "calle 15",
      "estado": "obra nueva",
      "exteriores_propiedad": "terrazza",
      "garaje": 0,
      "hipoteca_asociada": 30000,
      "idFavoritos": 0,
      "idinmuelle": 1,
      "precio": 200000,
      "tipo": "piso",
      "usuario_idusuario": 1
    }
  ]
}
```



Body:

```
{
  "empty": false,
  "inmuebles": [
    {
      "barrio": "canillas",
      "ciudad": "Madrid",
      "dimension": 0,
      "direccion": "calle 1",
      "estado": "buen estado",
      "exteriores_propiedad": "terrazza",
      "garaje": 0,
      "hipoteca_asociada": 30000,
      "idFavoritos": 0,
      "idinmuelle": 1,
      "precio": 200000,
      "tipo": "piso",
      "usuario_idusuario": 1
    },
    {
      "barrio": "la Moraleja",
      "ciudad": "Madrid",
      "dimension": 0,
      "direccion": "calle 15",
      "estado": "obra nueva",
      "exteriores_propiedad": "jardin",
      "garaje": 1,
      "hipoteca_asociada": 200000,
      "idFavoritos": 0,
      "idinmuelle": 6,
      "precio": 1000000,
      "tipo": "chalet",
      "usuario_idusuario": 1
    }
  ]
}
```

Ilustración 37. Prueba para visualizar las propiedades de un usuario.

En el caso de no existir el usuario devolverá una respuesta con un código 404 http.

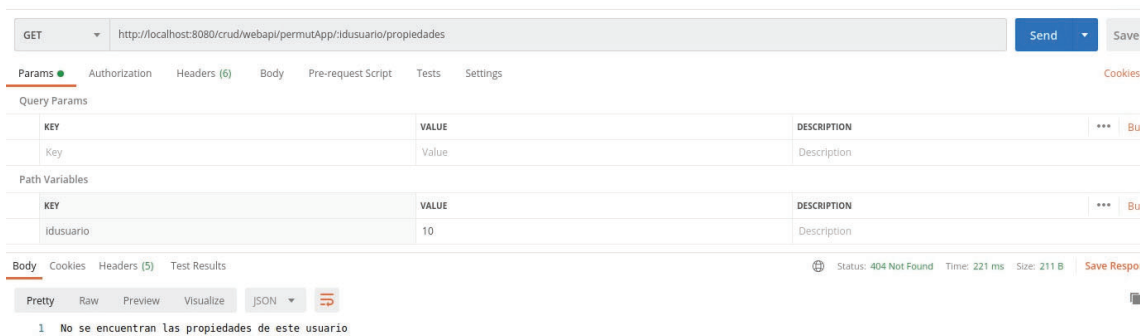
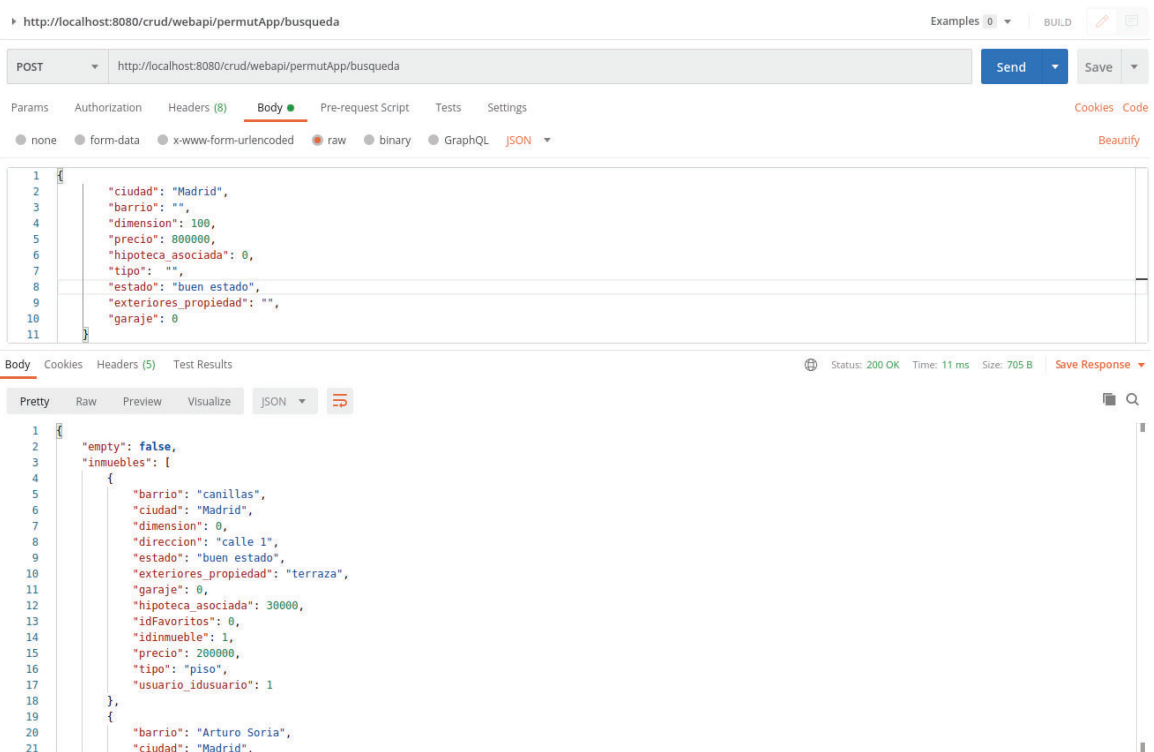


Ilustración 38. Prueba errónea para visualizar las propiedades de un usuario.

7. R07: Búsqueda de inmuebles.

Para consultar los inmuebles registrados en la aplicación se podrán aplicar una serie de filtros opcionales. Para ello se enviarán en el cuerpo de una petición POST los filtros especificados. Si la petición es exitosa, devolverá el listado con todos los inmuebles que cumplan las propiedades descritas en el JSON. En este ejemplo se buscan inmuebles en Madrid de más de 100 metros cuadrados y por un precio menor a 800.000 euros, que no tengan hipoteca asociada y que estén en buen estado.



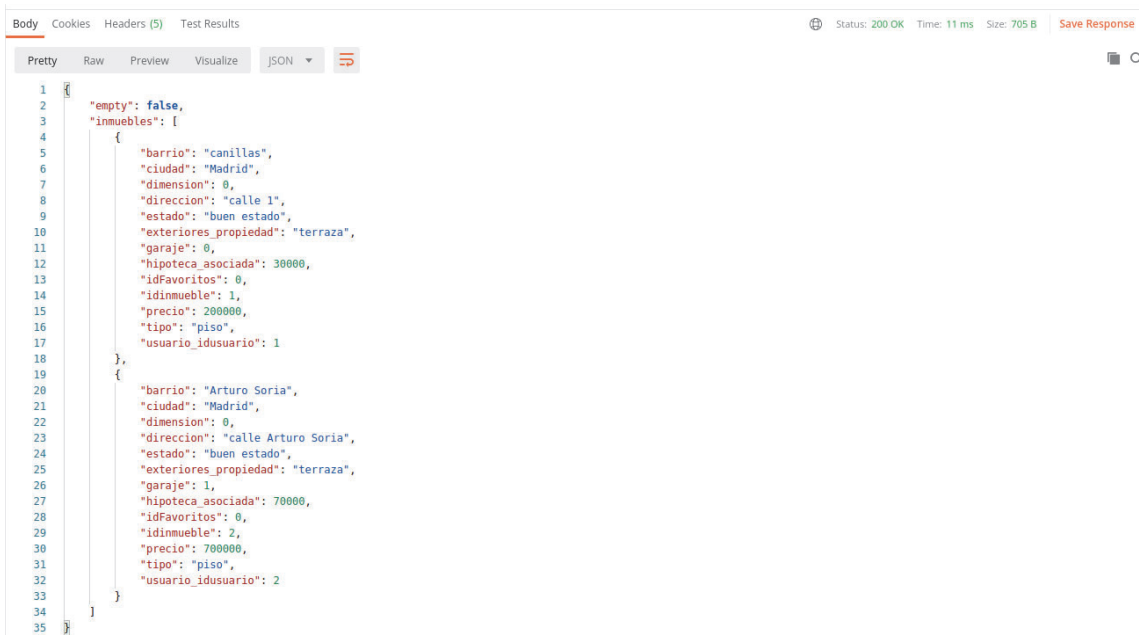


Ilustración 39. Prueba para realizar un búsqueda con filtro.

8. R08: Añadir un inmueble a la lista de favoritos de un usuario.

Para ello, es necesario una petición POST donde se indiquen el identificador del usuario que realiza la acción y el inmueble ajeno que quiere añadir a su lista de favoritos. La información se enviara como parámetros de la siguiente forma y devolverá el siguiente mensaje:

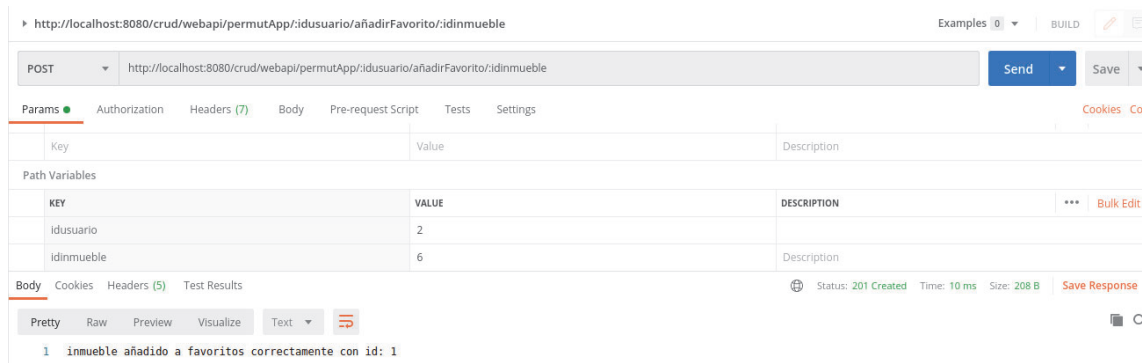


Ilustración 40. Prueba para añadir un inmueble a la lista de favoritos.

#	idfavoritos	inmueble_idinmueble	usuario_idusuario
1	1	6	2
2	2	5	2
3	3	4	1
*	NULL	NULL	NULL

Ilustración 41. Tabla favoritos tras añadir varios inmuebles a las listas.

En caso de que el inmueble no exista o el usuario esté intentando añadir un inmueble de su propiedad, devolverá:

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/crud/webapi/permApp/usuario/añadirFavorito/idinmuelle`
- Method:** POST
- Params:** A table with 3 columns: Key, Value, and Description. It contains two entries:

KEY	VALUE	DESCRIPTION
idusuario	2	
idinmuelle	9	Description
- Body:** The response is displayed in the "Text" view:


```
1 No se pudo añadir el inmueble a favoritos, porque el inmueble ya no existe o es de su propiedad
```
- Status:** 500 Internal Server Error, Time: 6 ms, Size: 236 B

Ilustración 42. Prueba errónea para añadir un inmueble a la lista de favoritos.

9. R09: Consultar la lista de favoritos.

Para consultar la lista de favoritos es necesario una petición GET que pase como parámetro el identificador del usuario del que se quiere consultar la lista de favoritos. Si la llamada es exitosa devolverá lo siguiente:

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/crud/webapi/permApp/usuario/favoritos`
- Method:** GET
- Params:** A table with 3 columns: Key, Value, and Description. It contains one entry:

KEY	VALUE	DESCRIPTION
idusuario	2	Description
- Body:** The response is displayed in the "JSON" view:


```
1 {
2   {
3     "barrio": "la Moraleja",
4     "ciudad": "Madrid",
5     "dimension": 0,
6     "direccion": "calle 15",
7     "estado": "obra nueva",
8     "exteriores_propiedad": "jardin",
9     "garaje": 1,
10    "hipoteca_asociada": 200000,
11    "idFavoritos": 1,
12    "idinmuelle": 6,
13    "precio": 1000000,
14    "tipo": "chalet",
15    "usuario_idusuario": 1
16  },
17  {
18    "barrio": "malasaña",
19    "ciudad": "Madrid",
20    "dimension": 0,
21    "direccion": "calle de la paz",
22    "estado": "buen estado",
23    "exteriores_propiedad": "",
24    "garaje": 0,
25  }
26 }
```
- Status:** 200 OK, Time: 13 ms, Size: 668 B

```

1  {
2    {
3      "barrio": "la Moraleja",
4      "ciudad": "Madrid",
5      "dimension": 0,
6      "direccion": "calle 15",
7      "estado": "obra nueva",
8      "exteriores_propiedad": "jardin",
9      "garaje": 1,
10     "hipoteca_asociada": 200000,
11     "idFavoritos": 1,
12     "idinmuble": 6,
13     "precio": 1000000,
14     "tipo": "chalet",
15     "usuario_idusuario": 1
16   },
17   {
18     "barrio": "malasaña",
19     "ciudad": "Madrid",
20     "dimension": 0,
21     "direccion": "calle de la paz",
22     "estado": "buen estado",
23     "exteriores_propiedad": "",
24     "garaje": 0,
25     "hipoteca_asociada": 0,
26     "idFavoritos": 2,
27     "idinmuble": 5,
28     "precio": 350000,
29     "tipo": "ático",
30     "usuario_idusuario": 5
31   }
32 }

```

Ilustración 43. Prueba de consulta de lista de favoritos de un usuario.

En caso contrario devolverá un código 500:

GET http://localhost:8080/crud/webapi/permutApp/idusuario/favoritos

Status: 500 Internal Server Error Time: 193 ms Size: 197 B

```

1  el usuario no existe o todavía no tiene favoritos

```

Ilustración 44. Prueba errónea de consulta de lista de favoritos de un usuario.

10. R10: Generar una permuta.

Para generar una solicitud de permuta tan solo será necesario realizar una petición POST con el identificador del inmueble de la lista de favoritos que desee el usuario y el identificador de una de sus propiedades con el objetivo de intercambiarlas. Si la petición es exitosa devolverá:

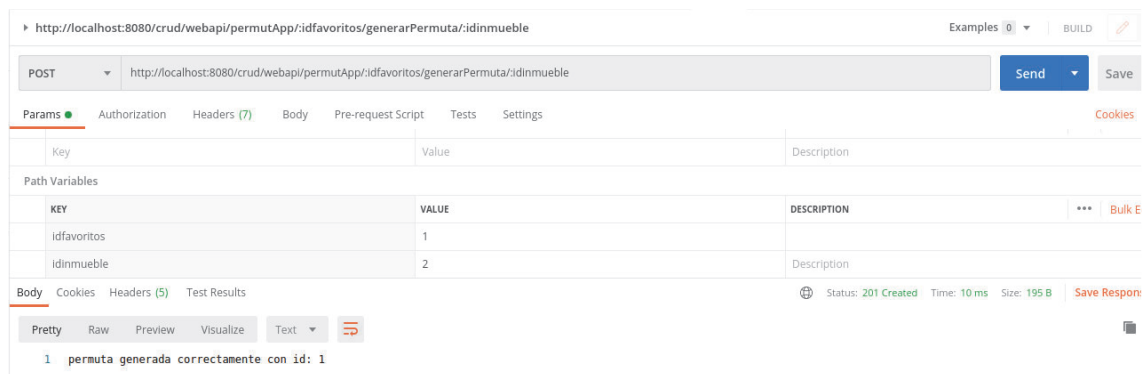


Ilustración 45. Prueba para generación de una permuta.

Y creará una nueva fila en la tabla permutas de modo que la tabla resultante de la llamada será:

1 • `SELECT * FROM permutasBD.permuta;`

#	idpermuta	favoritos_idfavoritos	cuantia_final	inmueble_idinmuelle	idusuario_pagado
1	1	1	170000	2	2
*	NULL	NULL	NULL	NULL	NULL

Ilustración 46. Tabla permuta tras generar una nueva permuta.

En caso de errores, ya sea porque el identificador del favorito o del inmueble no se encuentren en la base de datos, devolverá un código 500.

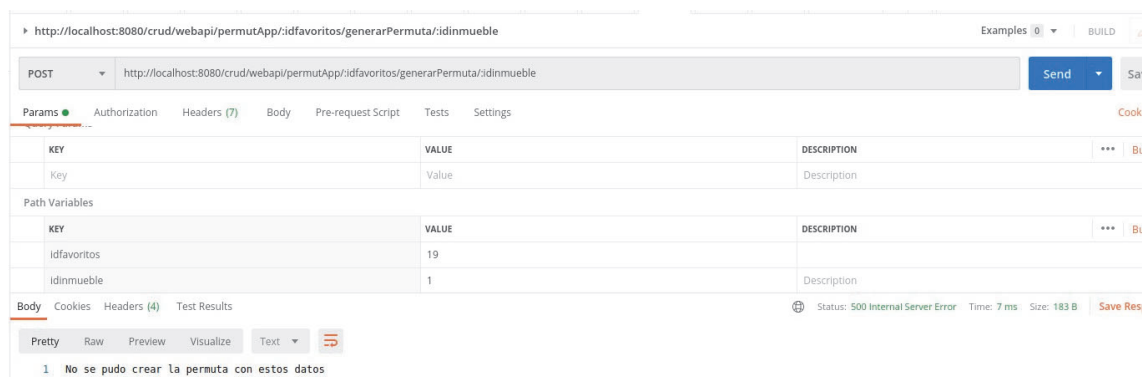


Ilustración 47. Prueba errónea de generación de permuta.

11. R11: Almacenar mensajes de una permuta.

Los usuarios podrán comunicarse entre sí a través de un canal particular por permuta. Para almacenar los mensajes, los usuarios deberán indicar el texto en cuestión y el identificador del emisor. Se realizará mediante una petición POST ya que creará una nueva fila en la tabla mensajes. Si la llamada es exitosa devolverá:

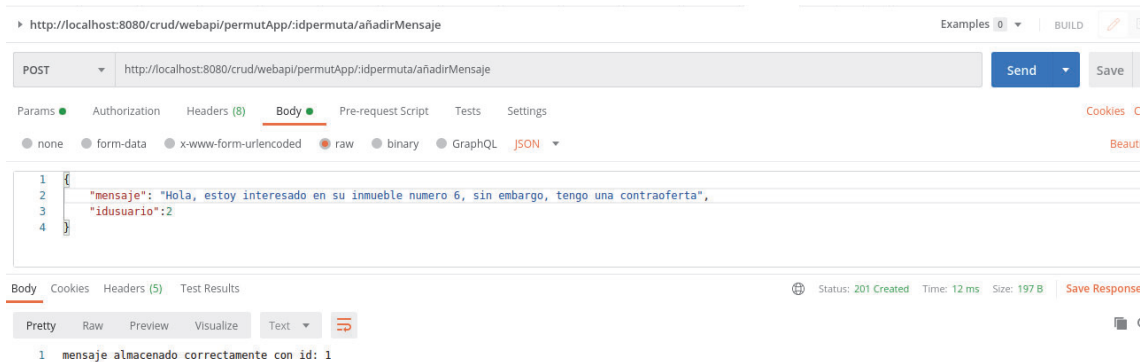


Ilustración 48. Prueba para almacenar mensajes de una permuta.

#	idmensajes	permuto_idpermuto	mensaje	idusuario
1	1	1	Hola, estoy interesado en su inmueble numero 6, sin embargo, tengo una cont...	2
2	2	1	De acuerdo, le escucho	1
3	3	1	Considero que deberíamos saldar nuestras hipotecas antes de realizar ningún...	2
4	4	1	Me parece bien, modificare mi hipoteca.	1
*	NONE	NONE	NONE	NONE

Ilustración 49. Tabla mensajes con los mensajes almacenados.

En caso contrario devolverá un código 404:

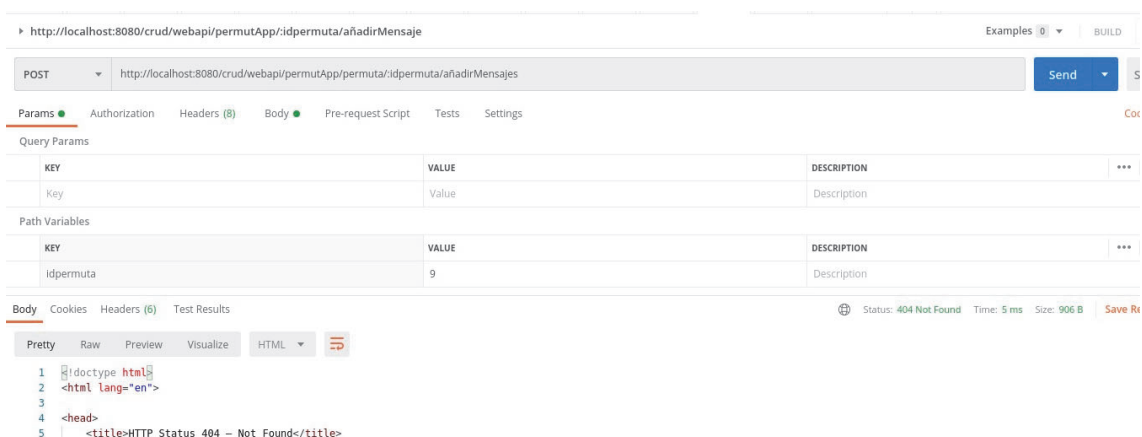


Ilustración 50. Prueba errónea para almacenar mensajes de una permuta.

12. R12: Consultar las permutas que hayan sido solicitadas a un usuario.

Para poder consultar las permutas recibidas será necesario enviar el identificador del usuario en una petición GET. Si la llamada es exitosa devolverá:

The screenshot shows a REST client interface with the following details:

- Request:** GET `http://localhost:8080/crud/webapi/permutApp/usuario/permutasSolicitadas`
- Path Variables:** `usuario` is set to `1`.
- Status:** 200 OK, Time: 252 ms, Size: 843 B.
- Response Body (JSON):**

```
{
  "permutas": [
    {
      "cuantia_final": 260000,
      "favoritos_idfavoritos": 1,
      "idinmueble_usuario": 2,
      "idpermuta": 1,
      "idusuario_pagador": 1,
      "inmueble1": {
        "barrio": "Arturo Soria",
        "ciudad": "Madrid",
        "dimension": 0,
        "direccion": "calle Arturo Soria",
        "estado": "buen estado",
        "exteriores_propiedad": "terrazza",
        "garaje": 1,
        "hipoteca_asociada": 30000,
        "idFavoritos": 0,
        "idinmueble": 2,
        "precio": 700000,
        "tipo": "piso",
        "usuario_idusuario": 2
      },
      "inmueble2": {
        "barrio": "la Moraleja",
        "ciudad": "Madrid",
        "dimension": 0,
        "direccion": "calle 15",
        "estado": "obra nueva",
        "exteriores_propiedad": "jardin",
        "garaje": 1,
        "hipoteca_asociada": 70000,
        "idFavoritos": 0,
        "idinmueble": 6,
        "precio": 1000000,
        "tipo": "chalet",
        "usuario_idusuario": 2
      }
    }
  ]
}
```

Ilustración 51. Prueba para visualizar las permutas solicitadas a un usuario.

En caso de que el usuario no existiera, devolverá un código 500:

The screenshot shows a REST client interface with the following details:

- Request:** GET `http://localhost:8080/crud/webapi/permutApp/usuario/permutasSolicitadas`
- Path Variables:** `usuario` is set to `20`.
- Status:** 500 Internal Server Error, Time: 120 ms, Size: 166 B.
- Response Body:** `1 No existe el usuario`

Ilustración 52. Prueba errónea para visualizar permutas solicitadas a un usuario.

13. R13: Visualizar los mensajes asociados a una permuta.

Los usuarios involucrados en una permuta podrán consultar sus conversaciones ordenadas por tiempo enviando una petición GET que incluya el identificador de la permuta. Si la permuta está en la base de datos y contiene mensajes, la petición devolverá:

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/crud/webapi/permutApp/permuta/idpermuta/mensajes`
- Method:** GET
- Path Variables:**

KEY	VALUE	DESCRIPTION
idpermuta	1	Description
- Status:** 200 OK, Time: 8 ms, Size: 643 B
- Body (JSON):**

```

1  {
2    "empty": false,
3    "mensajes": [
4      {
5        "idmensaje": 1,
6        "idpermuta": 1,
7        "idusuario": 2,
8        "mensaje": "Hola, estoy interesado en su inmueble numero 6, sin embargo, tengo una contraoferta"
9      },
10     {
11       "idmensaje": 2,
12       "idpermuta": 1,
13       "idusuario": 1,
14       "mensaje": "De acuerdo, le escucho"
15     },
16     {
17       "idmensaje": 3,
18       "idpermuta": 1,
19       "idusuario": 2,
20       "mensaje": "Considero que deberíamos saldar nuestras hipotecas antes de realizar ningún acuerdo."
21     },
22     {
23       "idmensaje": 4,
24       "idpermuta": 1,
25       "idusuario": 1,
26       "mensaje": "Me parece bien, modificare mi hipoteca."
27     }
28   ]
29 }
```

Ilustración 53. Prueba para visualizar los mensajes almacenados de una permuta.

En caso de error si no existe la permuta o la permuta no tiene mensajes asociados devolverá:

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/crud/webapi/permutApp/permuta/idpermuta/mensajes`
- Method:** GET
- Path Variables:**

KEY	VALUE	DESCRIPTION
idpermuta	10	Description
- Status:** 404 Not Found, Time: 7 ms, Size: 205 B
- Body (Text):**

```

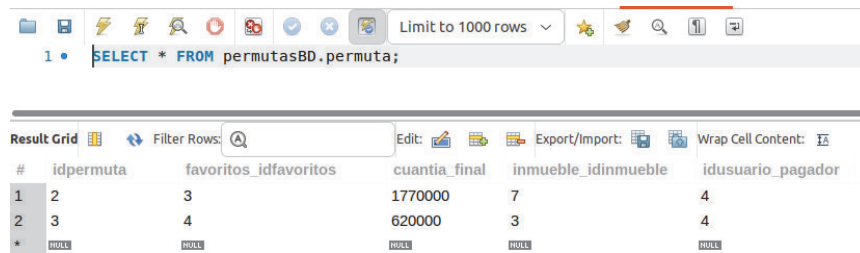
1  todavía no hay mensajes para esta permuta
```

Ilustración 54. Prueba errónea para visualizar los mensajes de una permuta.

14. R14: Eliminar una permuta.

Si el usuario solicitado no está interesado en la permuta ofertada, podrá rechazarla, provocando así la eliminación de la permuta de la base de datos.

Teniendo la siguiente tabla permuta:

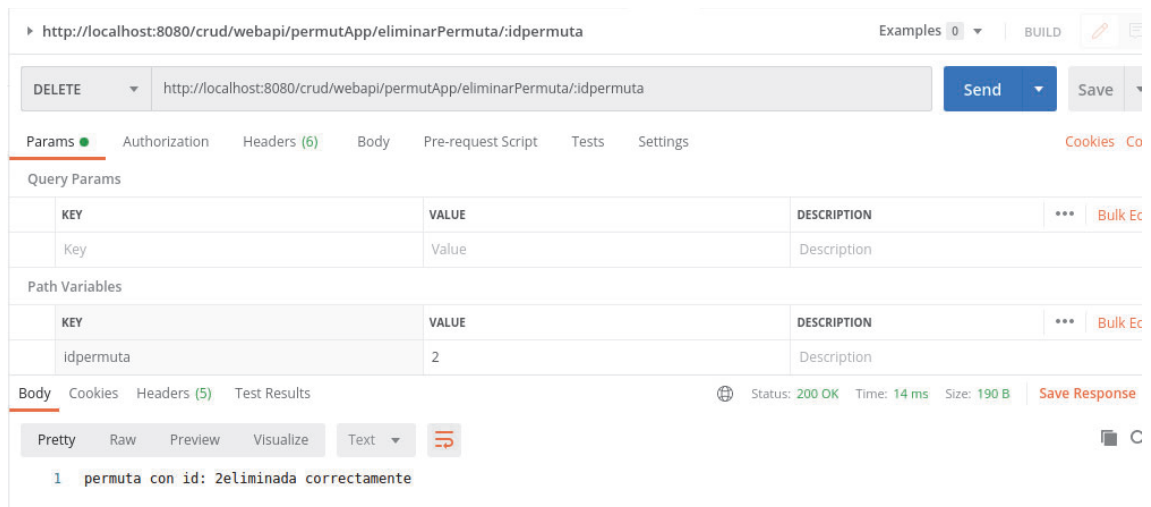


The screenshot shows a database query tool interface. At the top, there's a toolbar with various icons and a 'Limit to 1000 rows' dropdown. Below the toolbar, a SQL query is entered: `SELECT * FROM permutasBD.permuta;`. The results are displayed in a 'Result Grid' with columns: #, idpermuta, favoritos_idfavoritos, cuantia_final, inmueble_idinmueble, and idusuario_pagador. The grid contains three rows of data.

#	idpermuta	favoritos_idfavoritos	cuantia_final	inmueble_idinmueble	idusuario_pagador
1	2	3	1770000	7	4
2	3	4	620000	3	4
*	NULL	NULL	NULL	NULL	NULL

Ilustración 55. Tabla permuta.

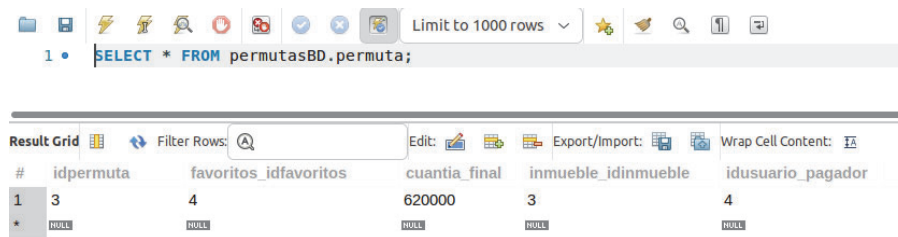
Se ha rechazado, y por lo tanto, se ha eliminado la permuta 2 la petición DELETE que ha devuelto el mensaje:



The screenshot shows a REST client interface. The URL bar displays `http://localhost:8080/crud/webapi/permutApp/eliminarPermuta/:idpermuta`. The method is set to 'DELETE'. The 'Send' button is highlighted. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is selected, showing a response message: `1 permuta con id: 2 eliminada correctamente`. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 14 ms', and 'Size: 190 B'.

Ilustración 56. Prueba para eliminar una permuta.

Y la tabla ha resultado:



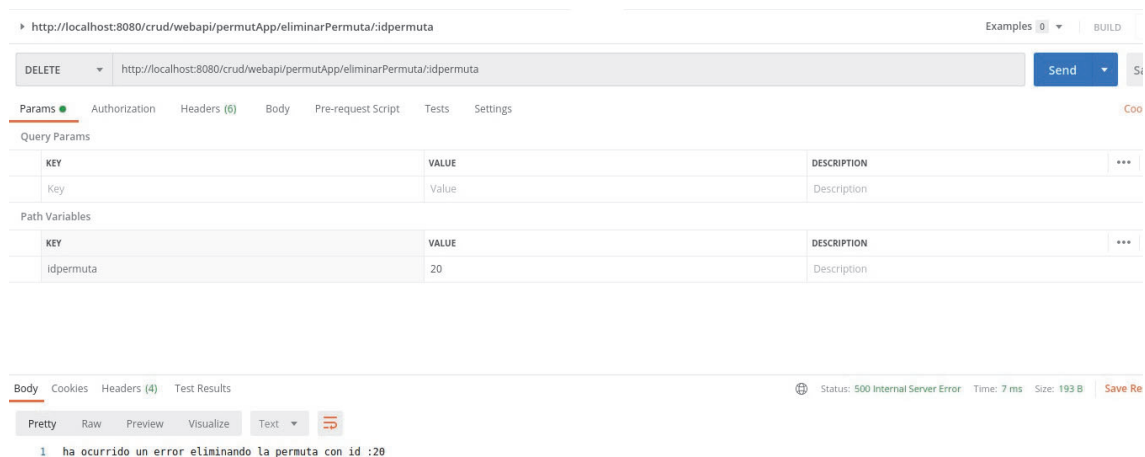
Limit to 1000 rows

```
1 • SELECT * FROM permutasBD.permuta;
```

#	idpermuta	favoritos_idfavoritos	cuantia_final	inmueble_idinmueble	idusuario_pagador
1	3	4	620000	3	4
*	NULL	NULL	NULL	NULL	NULL

Ilustración 57. Tabla permuta resultado.

En caso de error, si la permuta no existe, se indicará un código 500:



http://localhost:8080/crud/webapi/permutApp/eliminarPermuta/idpermuta

DELETE http://localhost:8080/crud/webapi/permutApp/eliminarPermuta/idpermuta

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Path Variables

KEY	VALUE	DESCRIPTION
idpermuta	20	Description

Body Cookies Headers (4) Test Results

Status: 500 Internal Server Error Time: 7 ms Size: 193 B Save Re

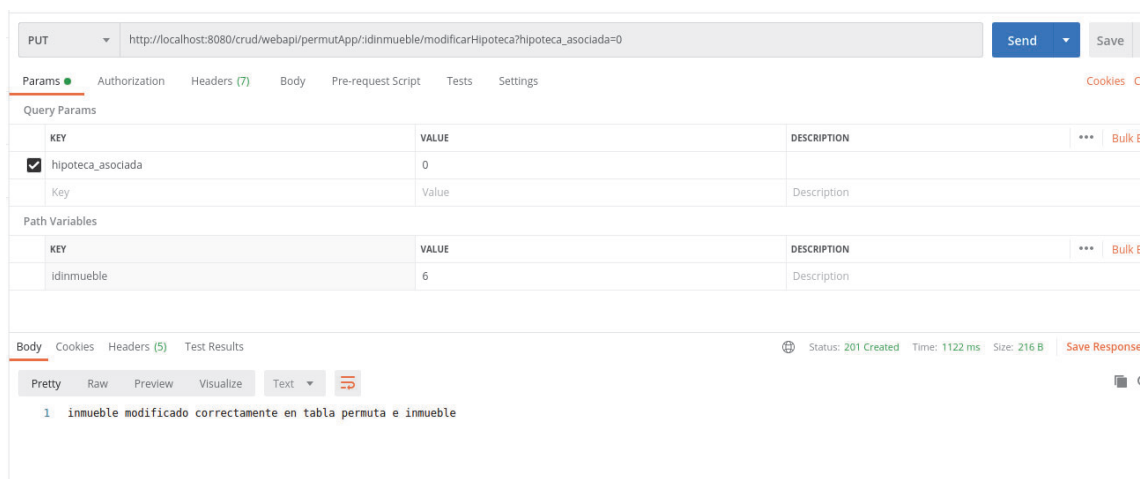
1 ha ocurrido un error eliminando la permuta con id :20

Ilustración 58. Prueba errónea eliminación de una permuta.

15. R15: Modificar precio e hipoteca.

Los usuarios podrán modificar el precio de sus inmuebles y la hipoteca asociada si así lo desean. Un posible caso podría ser el que ocurre en las conversaciones de la prueba 13. Ambos propietarios deben modificar el importe de sus hipotecas pendientes. Para ello, se debe realizar una petición PUT con el nuevo importe.

Siguiendo el ejemplo del punto 13, obtenemos los siguientes resultados:



PUT http://localhost:8080/crud/webapi/permutApp/idinmueble/modificarHipoteca?hipoteca_asociada=0

Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
☒ hipoteca_asociada	0	
Key	Value	Description

Path Variables

KEY	VALUE	DESCRIPTION
idinmueble	6	Description

Body Cookies Headers (5) Test Results

Status: 201 Created Time: 1122 ms Size: 216 B Save Response

1 inmueble modificado correctamente en tabla permuta e inmueble

Ilustración 59. Prueba para modificar la hipoteca asociada de un inmueble.

PUT http://localhost:8080/crud/webapi/permutApp/idinmuelle/modificarHipoteca?hipoteca_asociada=0 Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk
☒ hipoteca_asociada	0			
Key	Value	Description		

Path Variables

KEY	VALUE	DESCRIPTION	...	Bulk
idinmuelle	2	Description		

Body Cookies Headers (5) Test Results Status: 201 Created Time: 12 ms Size: 216 B Save Response

Pretty Raw Preview Visualize Text ...

1 inmueble modificado correctamente en tabla permuta e inmueble

Ilustración 60. Prueba para modificar la hipoteca asociada de un inmueble.

#	idinmuelle	ciudad	barrio	direccion	dimension	precio	hipoteca_asociad	tipo	estado	exteriores_propieda	garaje
1	1	Madrid	canillas	calle 1	100	200000	30000	piso	buen estado	terrazza	0
2	2	Madrid	Arturo Soria	calle Arturo Soria	180	700000	0	piso	buen estado	terrazza	1
3	3	Madrid	Goya	calle Goya	160	850000	0	piso	buen estado	terrazza	0
4	4	Madrid	chamartín	calle 14	80	250000	20000	piso	necesario ref...		0
5	5	Madrid	malasaña	calle de la paz	70	350000	0	ático	buen estado		0
6	6	Madrid	la Moraleja	calle 15	400	1000000	0	chalet	obra nueva	jardin	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 61. Tabla inmueble modificada.

#	idpermuta	favoritos_idfavoritos	cuantia_final	inmueble_idinmueble	idusuario_pagador
1	1	1	300000	2	2
*	NULL	NULL	NULL	NULL	NULL

Ilustración 62. Tabla permuta modificada.

En caso de error, devolverá un error 404:

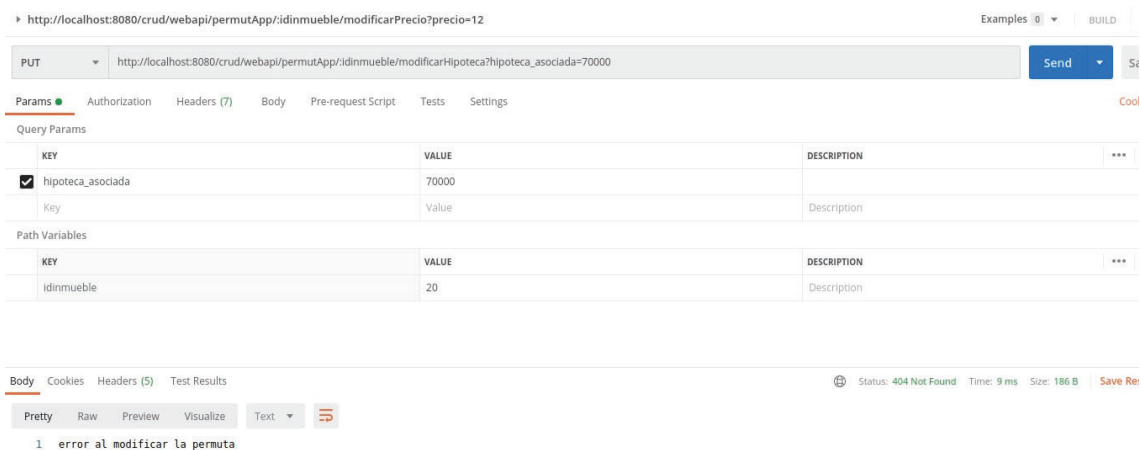


Ilustración 63. Prueba errónea para modificar el precio o la hipoteca de un inmueble.

16. R16: Eliminar un inmueble.

Si un usuario quiere eliminar de la aplicación una de sus propiedades deberá realizar una petición DELETE con el identificador del inmueble que quiere eliminar. Además, si este inmueble aparece en la lista de favoritos de algún usuario o interviene en alguna solicitud de permuta, se eliminarán ambos.

Teniendo la siguientes tabla favoritos:

Query 2 ✕ usuario ✕ inmueble ✕ usuario ✕ favoritos ✕ permuta ✕ mensajes ✕

Limit to 1000 rows

1 • `SELECT * FROM permutasBD.favoritos;`

#	idfavoritos	inmueble_idinmueble	usuario_idusuario
1	3	4	1
2	4	4	3
3	5	4	2
4	7	1	3
*	NULL	NULL	NULL

Ilustración 64. Tabla favoritos.

Si se elimina el inmueble con identificador 4, quedan los siguientes resultados:

Untitled Request BUILD

DELETE ▼ http://localhost:8080/crud/webapi/permutApp/eliminarInmueble/idInmueble Send Save

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk
Key	Value	Description		

Path Variables

KEY	VALUE	DESCRIPTION	...	Bulk
idInmueble	4	Description		

Body Cookies Headers (5) Test Results ⊕ Status: 200 OK Time: 10 ms Size: 192 B Save Response

Pretty Raw Preview Visualize Text ▼ ⌵

1 inmueble con id: 4 eliminado correctamente

Ilustración 65. Prueba para eliminar un inmueble.

1 • **SELECT * FROM** permutasBD.favoritos;

Result Grid ⌵ Filter Rows: 🔍 Edit: ✎ 📄 📄 Export/Import: 📄 📄 Wrap Cell Content: ⌵

#	idfavoritos	inmueble_idinmueble	usuario_idusuario
1	7	1	3
*	NULL	NULL	NULL

Ilustración 66. Nueva tabla favoritos.

En caso de error, devolverá un código http 500:

http://localhost:8080/crud/webapi/permutApp/eliminarInmueble/idInmueble Examples 0 BUILD

DELETE ▼ http://localhost:8080/crud/webapi/permutApp/eliminarInmueble/idInmueble Send Save

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk
Key	Value	Description		

Path Variables

KEY	VALUE	DESCRIPTION	...	Bulk
idInmueble	20	Description		

Body Cookies Headers (4) Test Results ⊕ Status: 500 Internal Server Error Time: 6 ms Size: 214 B Save Response

Pretty Raw Preview Visualize Text ▼ ⌵

1 ha ocurrido un error mientras se intentaba eliminar el inmueble con id: 20

Ilustración 67. Prueba errónea para eliminar inmuebles.

17. R17: Visualizar permutas enviadas.

Para visualizar las permutas que han sido enviadas por un usuario, tan solo es necesario realizar una petición GET con el identificador del usuario en cuestión, de modo que si la llamada es exitosa devolverá la lista con la información de las permutas de la siguiente manera:

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/crud/webapi/permutApp/usuario/permutas`
- Method:** GET
- Path Variables:** `usuario` is set to `1`.
- Status:** 200 OK
- Time:** 8 ms
- Size:** 834 B

The response body is displayed in JSON format:

```
1 {
2   {
3     "cuantia_final": 1860000,
4     "favoritos_idfavoritos": 8,
5     "idinmueble_usuario": 7,
6     "idpermuto": 7,
7     "idusuario_pagador": 2,
8     "inmueble": {
9       "barrio": "C6nde Orgaz",
10      "ciudad": "Madrid",
11      "dimension": 0,
12      "direccion": "Avenida de los Madroños",
13      "estado": "buen estado",
14      "exteriores_propiedad": "jardín",
15      "garaje": 1,
16      "hipoteca_asociada": 0,
17      "idFavoritos": 0,
18      "idinmueble": 7,
19      "precio": 2000000,
20      "tipo": "chalet",
21      "usuario_idusuario": 1
22    },
23    "inmueble2": {
24      "barrio": "Arturo Soria",
25      "ciudad": "Madrid",
26      "dimension": 0,
27      "direccion": "Calle de Arturo Soria",
28      "estado": "buen estado",
29      "exteriores_propiedad": "terrazza",
30      "garaje": 1,
31      "hipoteca_asociada": 60000,
32      "idFavoritos": 0,
33      "idinmueble": 9,
34      "precio": 200000,
35      "tipo": "piso",
36      "usuario_idusuario": 2
37    }
38  }
39 }
```

Ilustración 68. Prueba para visualizar las permutas enviadas por un usuario.

En caso de error devolverá un código 500:

▶ http://localhost:8080/crud/webapi/permutoApp/usuario/permuto

GET http://localhost:8080/crud/webapi/permutoApp/usuario/permuto

Params Authorization Headers (6) Body Pre-request Script Tests Settings

KEY	VALUE	DESCRIPTION
Key	Value	Description

Path Variables

KEY	VALUE	DESCRIPTION
usuario	20	Description

Body Cookies Headers (4) Test Results

Status: 500 Internal Server Error Time: 147 ms Size: 174 B Save

1 el usuario no tiene permuto

Ilustración 72. Prueba errónea para visualizar las permuto enviadas por un usuario.

18. R18: Generar un contrato.

Cuando ambas partes aceptan la permuto, se generará un contrato. Este contrato es un resumen que incluye todos los datos de la permuto. Además esta petición eliminará automáticamente los inmuebles involucrados de la base de datos. La petición devolverá:

Untitled Request

GET http://localhost:8080/crud/webapi/permutoApp/contrato/idpermuto

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Path Variables

KEY	VALUE	DESCRIPTION
idpermuto	1	Description

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results

Status: 201 Created Time: 22 ms Size: 1.2 KB Save Responses

```

1 {
2   "cantidad_final": 260000,
3   "inmueble1": {
4     "barrio": "Arturo Soria",
5     "ciudad": "Madrid",
6     "dimension": 0,
7     "direccion": "calle Arturo Soria",
8     "estado": "buen estado",
9     "exteriores_propiedad": "terrazza",
10    "garaje": 1,
11    "hipoteca_asociada": 30000,
12    "idFavoritos": 0,
13    "idinmueble": 2,
14    "precio": 700000,
15    "nombre_usuario_pagador": "Maria Jiménez",
16    "usuario1": {
17      "apellidos": "Gómez",
18      "contraseña": "*****",
19      "cuenta_bancaria": "ES3600 0023 2653 2844 9055",
20      "dni": "23565548",
21      "domicilio": "calle 22",
22      "email": "carlosgomez@com",
23      "idusuario": 2,
24      "nombre": "Carlos",
25      "telefono": "675943545"
26    },
27    "usuario2": {
28      "apellidos": "Jiménez",
29      "contraseña": "*****",
30      "cuenta_bancaria": "ES2200 0123 0456 0000 0355",
31      "dni": "9876554A",
32      "domicilio": "calle 1",
33      "email": "mariajimenez@com",
34      "idusuario": 1,
35      "nombre": "Maria",
36      "telefono": "666898765"
37    }
38  }
39 }

```

Ilustración 73. Prueba para generar un "contrato".

y la tabla inmueble resultante de la generación del contrato será:

#	idinmueble	ciudad	barrio	direccion	dimension	precio	hipoteca_asociad	tipo	estado	exteriores_propieda	garaje	usuario_idusuario
1	1	Madrid	canillas	calle 1	100	200000	30000	piso	buen estado	terrazza	0	1
2	3	Madrid	Goya	calle Goya	160	850000	0	piso	buen estado	terrazza	0	3
3	4	Madrid	chamartín	calle 14	80	250000	20000	piso	necesario ref...		0	4
4	5	Madrid	malasaña	calle de la paz	70	350000	0	ático	buen estado		0	5
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 74. Tabla inmueble resultante.

Del mismo modo que en el punto 17, al eliminar los inmuebles, se eliminarán los favoritos y permutas que les involucraban. También los mensajes asociados a la permuta en cuestión.

#	idpermuta	favoritos_idfavoritos	cuantia_final	inmueble_idinmueble	idusuario_pagador
*	NULL	NULL	NULL	NULL	NULL

Ilustración 76. Tabla permuta resultante.

#	idfavoritos	inmueble_idinmueble	usuario_idusuario
1	2	5	2
2	3	4	1
*	NULL	NULL	NULL

Ilustración 75. Tabla favoritos resultante.

En caso de error devolverá:

► <http://localhost:8080/crud/webapi/permutApp/contrato/idpermuta> Examples ▾

GET <http://localhost:8080/crud/webapi/permutApp/contrato/idpermuta> Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Path Variables

KEY	VALUE	DESCRIPTION
idpermuta	20	Description

Body Cookies Headers (4) Test Results Status: 500 Internal Server Error Time: 119 ms Size: 192 B

Pretty Raw Preview Visualize JSON Send

1 Error al imprimir contrato buscando la permuta

Ilustración 79. Prueba errónea al generar el contrato.

6 Conclusiones y trabajos futuros

6.1 Futuras ampliaciones

Por el momento PermutApp cuenta con los servicios descritos, pero con la opción de poder aportar otros en un futuro próximo. Algunas de las mejoras que se proponen para el futuro son las siguientes.

En primer lugar, la implementación de la funcionalidad para realizar permutas circulares que se ha descrito en el proyecto.

También, la aplicación podría añadir un servicio de verificación del registro de la propiedad y documentos requeridos para el intercambio de un inmueble.

En segundo lugar, podría añadir mecanismos de inteligencia artificial en el código para mejorar la generación de intercambios, pudiendo así mostrar a los usuarios en la pantalla principal de la aplicación una búsqueda personalizada y actualizada en tiempo real de inmuebles relacionados con sus búsquedas pasadas, sin necesidad de que el usuario inserte ningún filtro ni haga ninguna búsqueda como tal. Del mismo modo, se podrían crear alertas, para recibir notificaciones cada vez que se registre una propiedad con unas características específicas.

Por último, podría ser una nueva funcionalidad la de crear intercambios temporales. Estos consistirían en una alternativa a los alquileres. Esta funcionalidad estaría enfocada a aquellas personas que quieren mantener sus inmuebles en propiedad pero desean cambiar de hogar por una temporada. Para incluir esta propiedad, la aplicación solo tendría que incluir en el registro de la propiedad el alquiler mensual del inmueble. Así se podría calcular la diferencia entre los precios de alquiler que un usuario debiera pagar al otro.

6.2 Conclusiones

La realización de este trabajo ha supuesto un gran aprendizaje para mi carrera profesional, ya que el desarrollo de aplicaciones web es una de las áreas más recurridas en el sector de la informática. El auge de esta práctica se debe a la creación de aplicaciones para facilitar el trabajo manual que es necesario para llevar a cabo numerosas actividades que no han experimentado la digitalización hasta el momento. Un ejemplo de ello son las permutas, pero existen infinidad de actividades económicas que siguen realizándose de manera física, pudiendo desarrollar aplicaciones que agilicen su actividad.

Además, el hecho de crear una aplicación para un sector en el que no existían aplicaciones ha supuesto un reto, ya que no disponía de información para identificar las necesidades. Para encontrar dicha información pregunté a personas que habían realizado permutas y habían tenido que gestionar toda la operación por su cuenta. Gracias a ello aprendí bastante sobre el sector y pude especificar los requisitos de la aplicación.

Por último, cabe destacar que se han conseguido todos los objetivos propuestos en el apartado 1.2 y todas las pruebas han salido exitosas.

7 Análisis de Impacto

La realización de este trabajo me ha ayudado a desarrollar tanto mi conocimiento sobre tecnologías como mi capacidad empresarial. Esto se debe a que para el desarrollo de la aplicación PermutApp se han requerido conocimientos de tecnologías como Java, Maven o SQL. Pero también ha sido necesaria la visión empresarial para buscar un modelo de negocio atractivo e innovador. El conjunto de ambas ha llevado a conseguir completar los objetivos pautados.

Considero que a nivel económico, PermutApp puede tener éxito en el mercado si es acompañada de una campaña de publicidad que describa sus principales cualidades, ya que por su idea innovadora requiere de una explicación. Creo que PermutApp es una buena solución para miles de personas y cubre una necesidad no satisfecha hasta el momento.

Por otro lado, cabe destacar como el desarrollo de esta aplicación está alineado con el cumplimiento de los Objetivos de Desarrollo Sostenible de la Agenda de 2030. Al tratarse de un proceso de innovación en transformación digital contribuye positivamente al objetivo 8: “Empleo digno y crecimiento económico”, que promueve el emprendimiento en digitalización. También, su funcionalidad ayuda a personas que necesitan cambiar de hogar a conseguir su objetivo evitando en todo lo posible involucrar a los bancos, por lo que puede ser una oportunidad para reducir la desigualdad y con ello contribuir al objetivo 10. Por último, también contribuye al objetivo 12: “Consumo responsable”, ya que mediante la virtualización de la operación, la computación en la nube y las redes inteligentes se reduce el consumo de papel y otros materiales finitos.

Por todas estas razones, creo que PermutApp sería una buena idea de negocio a llevar a cabo.

8 Bibliografía

- [1] EUROPA PRESS, 7 diciembre 2021. [En línea]. Available: <https://www.europapress.es/economia/noticia-sector-inmobiliario-espanol-mejora-digitalizacion-28-cbre-20211207121408.html>. [Último acceso: mayo 2022].
- [2] PwC, «PwC,» 2021. [En línea]. Available: <https://www.pwc.es/es/real-estate/tendencias-mercado-inmobiliario-europa-2021.html>. [Último acceso: mayo 2022].
- [3] D. Bestard, «Por qué el 'proptech' es el mejor aliado de la inversión inmobiliaria,» *elEconomista*, 11 abril 2022.
- [4] B. Peláez, «Capterra. ¿Qué es PropTech? Descubre la revolución de la industria inmobiliaria,» 26 agosto 2020. [En línea]. Available: <https://www.capterra.es/blog/1694/que-es-proptech-descubre-la-revolucion-de-la-industria-inmobiliaria>. [Último acceso: mayo 2022].
- [5] «Permutas inmobiliarias: el trueque para simplificar la operación de cambiar de vivienda,» *ABC*, 22 julio 2021.
- [6] CBRE, «Madurez Digital del Sector Inmobiliario,» oct 2020.
- [7] Oi Realtor, «Real Estate News,» 4 septiembre 2021. [En línea]. Available: <https://www.oirealtor.com/noticias-inmobiliarias/la-evolucion-digital-inmobiliaria-y-su-avance/>. [Último acceso: 2022 mayo].
- [8] B. Peláez, «El 60% de los clientes usa una app móvil inmobiliaria en el proceso de compra o alquiler de propiedades,» 29 marzo 2022. [En línea]. Available: <https://www.capterra.es/blog/2611/estudio-comunicacion-tecnologia-en-el-sector-inmobiliario-espana>. [Último acceso: mayo 2022].
- [9] «Más del 70% de las empresas inmobiliarias en España están en proceso de digitalización,» *INMODIARIO*, 17 mayo 2021.
- [10] Oi REALTOR, «REAL ESTATE NEWS,» 2 diciembre 2021. [En línea]. Available: <https://www.oirealtor.com/noticias-inmobiliarias/permuta-de-inmuebles-de-que-se-trata-y-cuales-son-sus-ventajas/>. [Último acceso: mayo 2022].
- [11] «Statista,» 2019. [En línea]. Available: <https://es.statista.com/estadisticas/598603/porcentaje-de-propietarios-de-vivienda-con-hipoteca-union-europea-por-pais/>. [Último acceso: mayo 2022].
- [12] idealista, «idealista/news,» 11 marzo 2021. [En línea]. Available: <https://www.idealista.com/news/inmobiliario/blog-de-idealista/2021/03/11/789506-idealista-la-app-inmobiliaria-mas-usada>. [Último acceso: mayo 2022].
- [13] home exchange, [En línea]. Available: <https://www.homeexchange.com/es/blog/>. [Último acceso: mayo 2022].
- [14] IBM, «IBM Documentación JSON,» 27 febrero 2021. [En línea]. Available: <https://www.ibm.com/docs/es/baw/20.x?topic=formats-javascript-object-notation-json-format>. [Último acceso: mayo 2022].
- [15] Red Hat, «Red Hat: ¿Qué es una API de REST?,» 8 mayo 2020. [En línea]. Available: <https://www.redhat.com/es/topics/api/what-is-a-rest-api>. [Último acceso: mayo 2022].

- [16] Eclipse Foundation, «Eclipse,» [En línea]. Available: <https://www.eclipse.org/ide/>. [Último acceso: mayo 2022].
- [17] Maven, «Apache Maven project,» [En línea]. Available: <https://maven.apache.org/>. [Último acceso: mayo 2022].
- [18] «java,» [En línea]. Available: <https://www.java.com/es/download/help/techinfo.html>. [Último acceso: mayo 2022].
- [19] Eclipse, «Eclipse Jersey,» [En línea]. Available: <https://eclipse-ee4j.github.io/jersey/>. [Último acceso: mayo 2022].
- [20] Apache, «Apache Tomcat,» [En línea]. Available: <https://tomcat.apache.org/>. [Último acceso: mayo 2022].
- [21] «Postman,» [En línea]. Available: <https://www.postman.com/>. [Último acceso: mayo 2022].
- [22] MySQL, «MySQL Workbench,» [En línea]. Available: <https://www.mysql.com/products/workbench/>. [Último acceso: mayo 2022].
- [23] Digital Guide IONOS, «CRUD,» [En línea]. Available: <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/crud-las-principales-operaciones-de-bases-de-datos/>. [Último acceso: mayo 2022].
- [24] «El sector inmobiliario mejora en madurez digital por la pandemia pero todavía no llega al aprobado,» *elEconomista*, 7 12 2021.
- [25] idealista, «Idealista news,» 27 septiembre 2021. [En línea]. Available: <https://www.idealista.com/news/inmobiliario/vivienda/2021/09/27/792467-crece-la-permuta-de-viviendas-en-espana-en-que-consisten-los-trueques-inmobiliarios>. [Último acceso: mayo 2022].

Este documento esta firmado por



Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
Fecha/Hora	Thu Jun 02 22:36:54 CEST 2022
Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
Numero de Serie	561
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)