Portal para comunidades de propietarios

José Luis Zorita Gutiérrez
Grado de Ingeniería Informática
Trabajo Final de Grado - Java EE

Consultor: Antoni Oller Arcas

26/06/2023

FICHA DEL TRABAJO FINAL

Título del trabajo:	ADMINH Portal para comunidades de propietarios
Nombre del autor:	José Luis Zorita Gutiérrez
Nombre del consultor:	Antoni Oller Arcas
Fecha de entrega (mm/aaaa):	06/2023
Área del Trabajo Final:	Java EE
Titulación:	*Grado de Ingeniería Informática*

Resumen del Trabajo (máximo 250 palabras):

En el presente Trabajo de Final de Grado he creado la aplicación ADMINH, una plataforma para consultar la información relativa a una comunidad de propietarios y habilitar la comunicación entre sus vecinos y el administrador.

Con el uso de la aplicación, los propietarios pueden consultar y modificar sus datos personales, visualizar el estado contable o enviar mensajes al administrador, entre otras acciones. Si el propietario es presidente, también puede autorizar el pago de facturas. Asimismo, cuando se producen ciertos eventos, los propietarios reciben una notificación.

Por su parte, el administrador puede realizar acciones como crear nuevas publicaciones en el tablón, dar de alta facturas o responder las consultas de los propietarios.

La parte *backend* del proyecto ha sido creada siguiendo una arquitectura basada en microservicios (Java / Spring Boot), debido a la facilidad que ofrece el desarrollo modular y su configuración en contenedores Docker.

Los microservicios proveen de información, mediante un API REST, a la aplicación web con la que interactúa el usuario para visualizan los contenidos. Para conseguir desacoplamiento entre los microservicios y la aplicación web, se han utilizado diferentes tecnologías. Tras deliberar al respecto, finalmente se ha elegido PHP para esta última, debido a la gran cantidad de documentación existente y su flexibilidad.

El desarrollo del proyecto ha tenido una duración de cuatro meses. Se ha elaborado siguiendo una planificación en etapas según lo estipulado en el Plan de Trabajo inicial, lo que ha dado como resultado el trabajo que se muestra en el presente documento.

Abstract (in English, 250 words or less):

This project shows ADMINH, a software that allows users to consult the information related to a community of owners and that enables communication between owners and administrators.

By using the application, the community owners can check their personal data and modify it, review the accounting or send messages to the administrator, among other actions. If the owner is the president, he can also authorize the payment of bills. In addition, owners receive a notification when certain events occur.

Furthermore, administrators can perform some actions, like creating new publications on the community board, registering invoices or replying to the owners' queries.

The backend part of the project has been developed following a microservices architecture based on the combination of tools that include Java, Spring Boot Kafka or Adminer. The choice of Microservices is due to the ease that modular development brings and its configuration using Docker containers.

Microservices provide information to a web application which user interacts with to access its content through an API REST. The technologies involved in the web app development are different from those used for the microservices, in order to achieve decoupling between the two platforms. After some deliberation, PHP was finally chosen for the web app due to the large amount of documentation available and its flexibility.

The development of the project has lasted four months, and has been carried out following the stages stipulated in the initial Work Plan. The end result is the work shown in this document.

Palabras clave (entre 4 y 8):

microservicios java php ajax api rest web

Índice

1. Introducción 1
1.1 Contexto y justificación del Trabajo 1
1.2 Objetivos del Trabajo 1
1.3 Enfoque y método seguido 2
1.4 Planificación del Trabajo 3
1.5 Breve sumario de productos obtenidos 5
1.6 Breve descripción de los otros capítulos de la memoria 5
2. Requerimientos, análisis y diseño 6
2.1 Modelo de casos de uso y actores 6
2.2 Fichas de casos de uso 8
2.3 Modelo de pantallas (prototipo) 17
2.4 Diagrama de arquitectura 24
2.5 Microservicios 25
2.6 Operaciones y consultas del sistema 28
2.7 Métodos de los servicios API REST 29
2.8 Control de sesión 41
2.9 Arquitectura hexagonal 42
2.10 Etapa 1: Esqueleto de la aplicación 45
3. Implementación 47
3.1 Entorno de desarrollo 47
3.2 Cambios en el alcance del proyecto 47
3.3 Implementación: cambios respecto a lo planificado inicialmente 48
3.4 Etapa 2: Creación infraestructura 49
3.5 Etapa 3: API REST y WEB propietarios 55
3.6 Etapa 4: API REST y WEB administrador 62
3.7 Etapa 5: Seguridad 64
4. Conclusiones 66
5. Glosario 68
6. Bibliografía 70
7. Anexos 72
7.1 Localización proyecto 72
7.2 Instalación y ejecución 72
7.3 Pruebas servicios del API REST con Postman 76

Lista de figuras

Fig 1 Diagrama realizado para la PEC1 con la previsión de la estructura del proyecto 2
Fig 2 Etapas del desarrollo del proyecto 3
Fig 3 Diagrama de GANTT 4
Fig 4 Diagrama de casos de uso 8
Fig 5 Login 18
Fig 6 Logout 18
Fig 7 Estado contable 19
Fig 8 Tablón de anuncios 19
Fig 9 Tablón de anuncios admininstrador 19
Fig 10 Nueva publicación en el tablón 20
Fig 11 Datos personales 20
Fig 12 Mensaje al administrador 20
Fig 13 Respuesta del administrador 21
Fig 14 Generar factura 21
Fig 15 Generación de nueva factura 21
Fig 16 Generación de nueva cuota 22
Fig 17 Autorización pago facturas 22
Fig 18 Consultar recibos pendientes 22
Fig 19 Consulta notificaciones 23
Fig 20 Subir fichero 23
Fig 21 Selección comunidad 23
Fig 22 Arquitectura proyecto 24
Fig 23 XAMPP 25
Fig 24 OpenJDK 25
Fig 25 Maven 25
Fig 26 Spring 25
Fig 27 PostgreSQL 25
Fig 28 Kafka 25
Fig 29 Spring Boot 25
Fig 30 Project Lombok 25
Fig 31 Hibernate 26
Fig 32 Entidades microservicios 26
Fig 33 Clases sesion y Kafka 27
Fig 34 Descomposición en subdominios 27
Fig 35 Permisos servicios REST 41
Fig 36 Arquitectura hexagonal microservicio adminh-user 42
Fig 37 Arquitectura hexagonal microservicio adminh-crm 43
Fig 38 Arquitectura microservicio adminh-core 44
Fig 39 Diagrama que muestra los nueve servicios desplegados con docker-compose 49
Fig 40 Captura Docker Desktop con despliegue servicios 50
Fig 41 Resultado método GET /presupuesto/partidas/{partidaId} en Postman 51
Fig 42 Esquema de la base de datos del microservicio adminh-crm visto desde Adminer 54
Fig 43 Ficheros web propietario 58

Fig 44 Al seleccionar una publicación del tablón en Figma vemos su código CSS en la parte derecha 61
Fig 45 Publicaciones en la versión final de la web 62
Fig 46 Ficheros web administrador 62
Fig 47 Publicación en el tablón con TinyMCE 63

1. Introducción

1.1 Contexto y justificación del Trabajo

En la actualidad existen multitud de administradores de fincas que ofrecen aplicativos a los vecinos de las comunidades que administran para que consulten datos relativos a la finca.

Normalmente, estas aplicaciones muestran información muy limitada, como el estado contable o mensajes creados en el sistema ERP utilizado por su administración, no permitiendo realizar otras acciones que involucren la inserción, la modificación o el eliminado de datos. Asimismo, se trata de aplicaciones que están pensadas para que sean utilizadas únicamente por los propietarios y no por el administrador, quién opera desde el mencionado sistema ERP de su empresa.

Esto provoca que, en la mayoría de los casos, la aplicación sea poco utilizada y que las gestiones se continúen realizando personalmente o con medios convencionales, como la comunicación telefónica o mediante el envío de correo postal.

En primer lugar, el propósito de ADMINH es ser una herramienta que pueda ser utilizada también por el propio administrador, de tal forma que esté presente en el mismo ecosistema que utilizan los vecinos. Desde la aplicación podrá realizar acciones como contestar los mensajes enviados por los clientes, crear facturas o recibos.

En segundo lugar, ADMINH tiene el objetivo de ser una plataforma desde la cual los vecinos puedan realizar gestiones fácilmente sin tener que desplazarse hasta su administrador o utilizar otros medios. Entre estas acciones se incluye el envío de consultas al administrador, la actualización de sus datos personales o la autorización del pago de facturas.

1.2 Objetivos del Trabajo

Podemos listar los siguientes objetivos:

1- Crear una plataforma digital que dé respuesta a las necesidades que hemos indicado en el punto anterior. Para ello, se pretende ofrecer al usuario un aplicativo con las siguientes funcionalidades:

- **Ficha con los datos personales del vecino**, con la posibilidad de que éste los modifique (cambiar la dirección de correo electrónico, añadir otro teléfono, modificar la cuenta bancaria, etc.).
- **Detalle de la contabilidad** de la comunidad de propietarios: relación de gastos e ingresos del ejercicio en curso y saldo con el que cuenta la comunidad.
- **Relación de cuotas pendientes** del vecino.
- Un **tablón virtual** en el que realizar notificaciones de forma global a todos los propietarios (por ejemplo, una convocatoria de reunión).
- **Canal de comunicación** para que los vecinos puedan remitir incidencias al administrador.
- Solicitar al presidente la **autorización para el pago de facturas** de trabajos imprevistos.
- **Notificar la generación de nuevas cuotas** y, si el vecino no tiene los pagos domiciliados, informarle adjuntando el número de **cuenta bancaria para que puedan realizar una transferencia**.

2- Montar una infraestructura modular basada en microservicios, que será la encargada de suplir la información necesaria al cliente mediante llamadas a los servicios API REST.

3- Crear una página web que consuma los servicios del API REST y que actúe como interfaz (*frontend)* para que el usuario pueda visualizar los distintos contenidos y realizar las acciones dependiendo del rango de usuario que tenga.

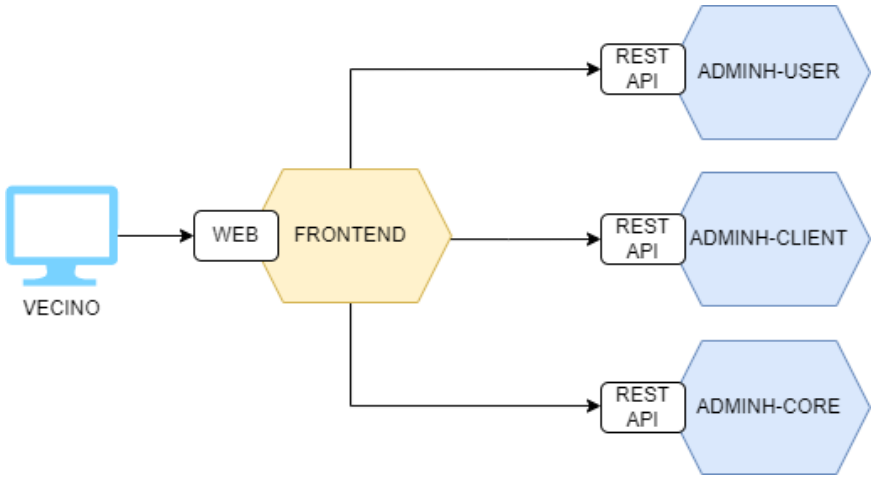


Fig 1 Diagrama realizado para la PEC1 con la previsión de la estructura del proyecto

1.3 Enfoque y método seguido

La estrategia a seguir para el desarrollo de la plataforma será la de producir un producto nuevo aplicando los conocimientos adquiridos en las asignaturas del itinerario cursadas durante el grado.

Concretamente, los microservicios serán realizados siguiendo las mismas tecnologías empleadas en las asignaturas de *Ingeniería de Programación de Componentes y Sistemas Distribuidos* y *Proyecto de Desarrollo de Software* (Java Spring Boot como lenguaje y Kafka para la comunicación entre microservicios), así como docker-compose para configurar y automatizar el arranque de todos los servicios necesarios.

Por otra parte, debido a que tengo únicamente conocimientos básicos en desarrollo de páginas web, se requiere planificar una etapa de investigación y aprendizaje para poder aplicar una solución concreta en la parte *frontend*.

En lo que respecta a la metodología de trabajo, ésta será del tipo ágil con un desarrollo iterativo, tal y como contempla el plan de trabajo descrito en el siguiente punto.

Finalmente, se han creado cinco repositorios en github para almacenar la infraestructura, los tres microservicios y la aplicación web.

1.4 Planificación del Trabajo

Como se ha indicado, para el desarrollo del proyecto se ha seguido una metodología ágil realizando diferentes iteraciones. En concreto, se han diseñado un total de ocho con una duración de una semana cada una durante el tiempo estipulado para la PEC 3.

Una etapa del desarrollo corresponde a la suma de dos iteraciones, que junto al esqueleto realizado previamente definen el total del desarrollo a ejecutar.

Se han definido las siguientes etapas a modo de *roadmap* para el proyecto:

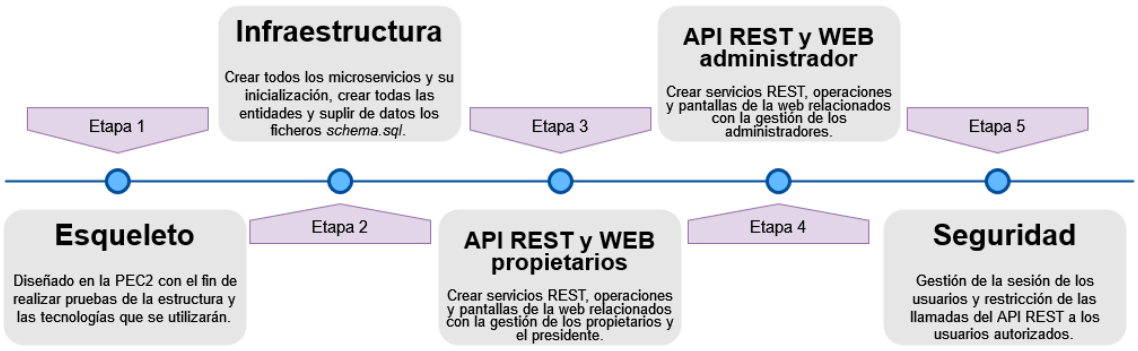


Fig 2 Etapas del desarrollo del proyecto

A continuación se adjunta el calendario con diagrama de Gantt que incluye todas las tareas del proyecto.

En este calendario se han señalado los hitos (que corresponden a las entregas, el fin del tribunal, el cierre del proyecto) y también las etapas que hemos descrito:

	Nombre	Duracion	Inicio	Terminado
1	**TFG JAVA EE**	**138 days**	**1/03/23**	**16/07/23**
2	**PEC1 - Plan de trabajo**	**14 days**	**1/03/23**	**14/03/23**
3	Definir 1ª propuesta requerimientos aplicación	6 days	1/03/23	6/03/23
4	Definir 1ª propuesta herramientas a utilizar para backend	3 days	7/03/23	9/03/23
5	Plan de trabajo	5 days	10/03/23	14/03/23
6	Entrega PEC1	1 day	14/03/23	14/03/23
7	**PEC 2 - Modelo y especificación proyecto**	**31 days**	**15/03/23**	**14/04/23**
8	Estudio software a utilizar y frameworks	31 days	15/03/23	14/04/23
9	Modelo de casos de uso y actores	7 days	15/03/23	21/03/23
10	Fichas de casos de uso	7 days	15/03/23	21/03/23
11	Modelo de pantallas (prototipo)	8 days	22/03/23	29/03/23
12	Diseño relacional de la base de datos	8 days	30/03/23	6/04/23
13	Diagrama de clases principales	8 days	30/03/23	6/04/23
14	Diagrama de arquitectura	7 days	7/04/23	13/04/23
15	Entrega PEC2	1 day	14/04/23	14/04/23
16	**PEC 3 - Implementación base de la aplicación**	**52 days**	**15/04/23**	**5/06/23**
17	Caso de uso 1 - Iteración 1 - Validación arquitectura	6 days	15/04/23	20/04/23
18	Caso de uso 1 - Iteración 2 - Usabilidad y gestión de err..	4 days	21/04/23	24/04/23
19	Entrega parcial Caso de uso 1	1 day	25/04/23	25/04/23
20	Caso de uso 1 - Iteración 3 - Completar caso de uso	3 days	26/04/23	28/04/23
21	Iteración 4	7 days	29/04/23	5/05/23
22	Iteración 5	7 days	6/05/23	12/05/23
23	Entrega parcial iteraciones 4 y 5	1 day	13/05/23	13/05/23
24	Iteración 6	7 days	14/05/23	20/05/23
25	Iteración 7	7 days	21/05/23	27/05/23
26	Entrega parcial iteraciones 6 y 7	1 day	28/05/23	28/05/23
27	Iteración 8	7 days	29/05/23	4/06/23
28	Entrega PEC 3	1 day	5/06/23	5/06/23
29	**Memoria y presentación**	**21 days**	**6/06/23**	**26/06/23**
30	Desarrollo documento memoria	15 days	6/06/23	20/06/23
31	Preparación producto final para su entrega	15 days	6/06/23	20/06/23
32	Desarrollo presentación virtual	5 days	21/06/23	25/06/23
33	Entrega Memoria y presentación	1 day	26/06/23	26/06/23
34	**Tribunal**	**7 days**	**3/07/23**	**9/07/23**
35	Responder preguntas tribunal	7 days	3/07/23	9/07/23
36	Calificación final - cierre TFG	1 day	16/07/23	16/07/23

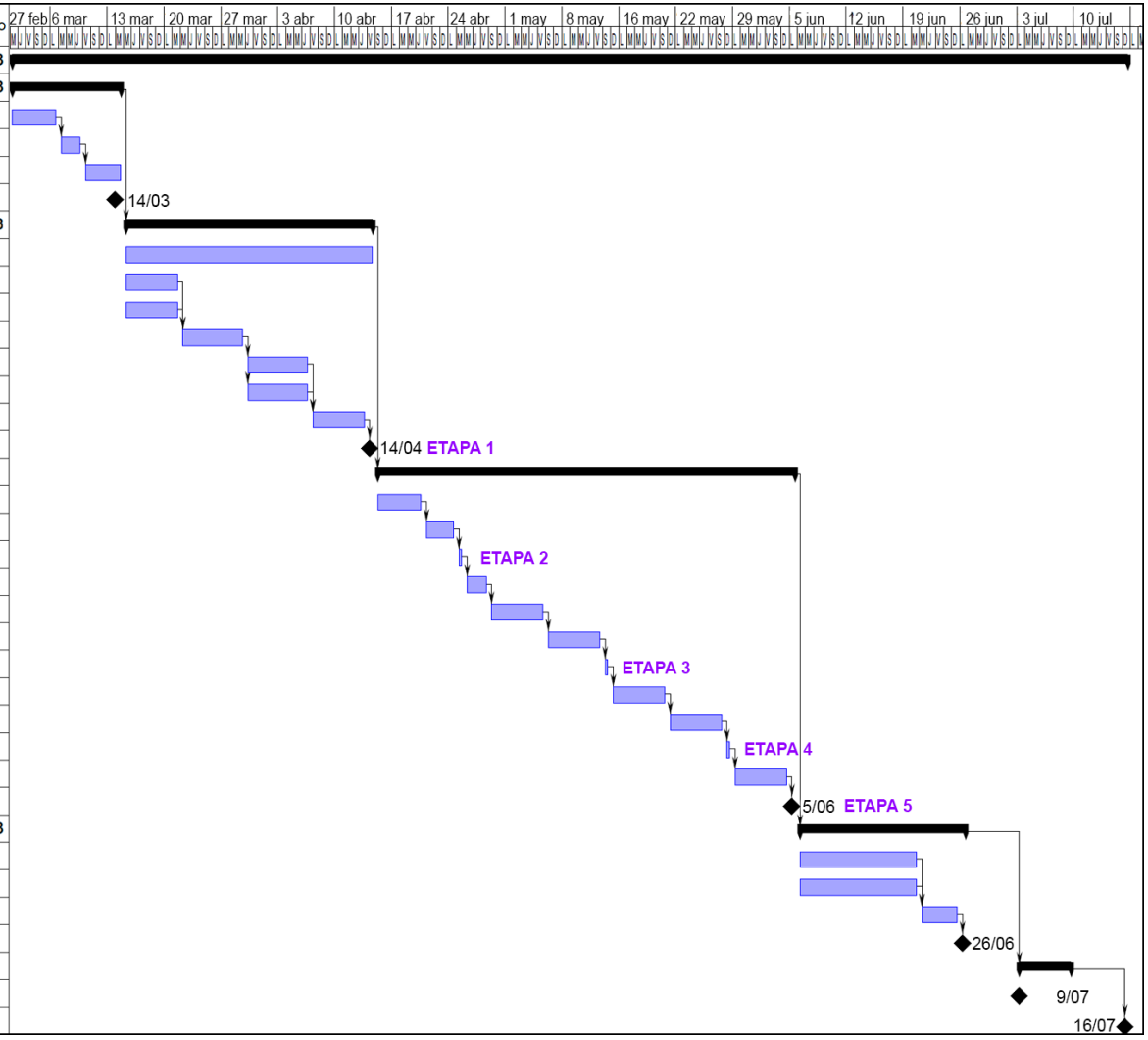


Fig 3 Diagrama de GANTT

1.5 Breve sumario de productos obtenidos

El resultado final del trabajo da como resultado lo siguiente:

- Documento con Memoria del proyecto
- Enlace a repositorio de github con fichero `docker-compose.yml` para el despliegue de los microservicios
- Enlace a los repositorios de github que contienen el código fuente de los microservicios *adminh-user*, *adminh-crm* y *adminh-core*
- Enlace al repositorio de github que contiene el código fuente de la aplicación web
- Documento con Presentación Virtual
- Video tipo *screencast* con presentación del proyecto

Asimismo, en el anexo se encuentran los enlaces a los repositorios y las instrucciones de instalación.

1.6 Breve descripción de los otros capítulos de la memoria

La memoria del trabajo está conformada por los siguientes puntos:

1- **Introducción**

En este punto hemos visto el contexto que hace viable el desarrollo de la aplicación, sus objetivos, el público al que va dirigido y finalmente la planificación realizada.

2- **Requerimientos, análisis y diseño**

En el segundo punto se muestra el análisis realizado, tanto a nivel de análisis de los requerimientos que da como resultado la elaboración de un prototipo, como a nivel de estudio y selección de las herramientas de software a utilizar para su implementación.

Se debe tener en cuenta que este análisis se realizó para la entrega de la PEC2. Debido a factores como la metodología ágil empleada o la falta de experiencia en el uso de varias tecnologías, el diseño final ha sufrido variaciones que se han ido documentando en la propia memoria.

3- **Implementación producto**

Punto que incluye detalles acerca de la implementación final del producto, las características incorporadas y su alcance.

4- **Conclusiones**
5- **Bibliografía**
6- **Glosario**
7- **Anexo**

En el anexo se adjuntan los enlaces para la descarga de los repositorios, las instrucciones de instalación y capturas de pantalla de Postman que muestran la ejecución satisfactoria de los métodos de los servicios del API REST.

2. Requerimientos, análisis y diseño

2.1 Modelo de casos de uso y actores

Podemos identificar los siguientes actores:

Nombre	Descripción
Propietario	Propietario de una entidad (vivienda, local, etc.) de una comunidad de propietarios.
Presidente	Presidente de una comunidad de propietarios.
Administrador	Administrador de fincas encargado de la gestión de la comunidad.
Sistema	Realiza acciones que no son iniciadas por el usuario, como las notificaciones a los propietarios.

Partimos de las historias de usuario que se pueden identificar como características o necesidades que tienen los usuarios y que hace factible el desarrollo de la aplicación:

- **Como** propietario / presidente **quiero** una aplicación **para** poder consultar la información de mi comunidad desde casa.
- **Como** administrador **quiero** poder enviar comunicados online a los vecinos **para** no tener que desplazarme hasta la comunidad para colgar un papel.
- **Como** presidente **quiero** poder enviar circulares **para** informar a los vecinos de incidencias que se produzcan.
- **Como** propietario / presidente **quiero** poder enviar mensajes al administrador por internet **para** poder tratar un problema de forma rápida.
- **Como** propietario / presidente **quiero** poder enviar comunicados por internet **para** poderlos añadir a mi calendario de Gmail.
- **Como** administrador **quiero** poder comunicarme con los vecinos por internet **para** que se guarde un registro de las incidencias acaecidas en la comunidad.
- **Como** administrador **quiero** poder crear recibos online **para** que los vecinos sean notificados al instante de la nueva cuota a pagar.
- **Como** propietario / presidente **quiero** poder consultar mis recibos pendientes **para** poder realizar el pago de lo que deba.
- **Como** propietario / presidente **quiero** recibir notificaciones **para** enterarme rápidamente de las comunicaciones del administrador.
- **Como** presidente **quiero** autorizar el pago de facturas online **para** no tener que desplazarme a la gestoría a firmar y llevar un mejor control.
- **Como** propietario / presidente **quiero** consultar el estado contable y el saldo online **para** no tener que esperar a una reunión para conocerlo.
- **Como** propietario /presidente **quiero** poder modificar mis datos de contacto por internet **para** que mi información esté siempre actualizada.

- **Como** administrador **quiero** gestionar facturas y recibos desde una aplicación **para** que los vecinos puedan consultar la información contable actualizada.

A raíz de los roles y las historias de usuario identificadas se pueden crear los siguientes casos de uso:

Id	Actor	Caso de uso
CU01.1	Administrador Propietario Presidente	Iniciar sesión en la aplicación
CU01.2		Cerrar sesión en la aplicación
CU02.1	Administrador Presidente Propietario	Consultar las publicaciones de una comunidad
CU02.2	Administrador	Crear una publicación en el tablón
CU02.3	Administrador	Eliminar una publicación del tablón
CU02.4	Administrador	Modificar una publicación del tablón
CU03.1	Administrador Presidente Propietario	Consultar las facturas de una comunidad
CU03.2	Administrador	Crear una factura de una comunidad
CU03.3	Administrador	Eliminar una factura de una comunidad
CU03.4	Administrador	Modificar una factura de una comunidad
CU03.4	Administrador	Anexar fichero pdf con factura **(funcionalidad incorporada durante la implementación)**
CU04	Propietario Presidente	Enviar mensaje al administrador
CU05	Administrador	Responder un mensaje de un propietario
CU06	Administrador	Crear recibos de una comunidad
CU07	Presidente	Autorizar pago factura
CU08	Propietario Presidente	Consultar los recibos pendientes
CU09.1	Propietario Presidente	Consultar datos personales
CU09.2		Modificar datos personales
CU10	Administrador Propietario Presidente	Consultar la contabilidad de la comunidad
CU11	Propietario Presidente	Añadir evento a Google Calendar
CU12.1	Propietario Presidente	Consultar notificaciones

CU12.2	Sistema	Enviar notificación
CU12.3	Propietario Presidente	Eliminar notificación
CU13	Administrador	Cambiar de comunidad **(funcionalidad incorporada durante la implementación)**

El diagrama de casos de uso que se desprende es el siguiente:

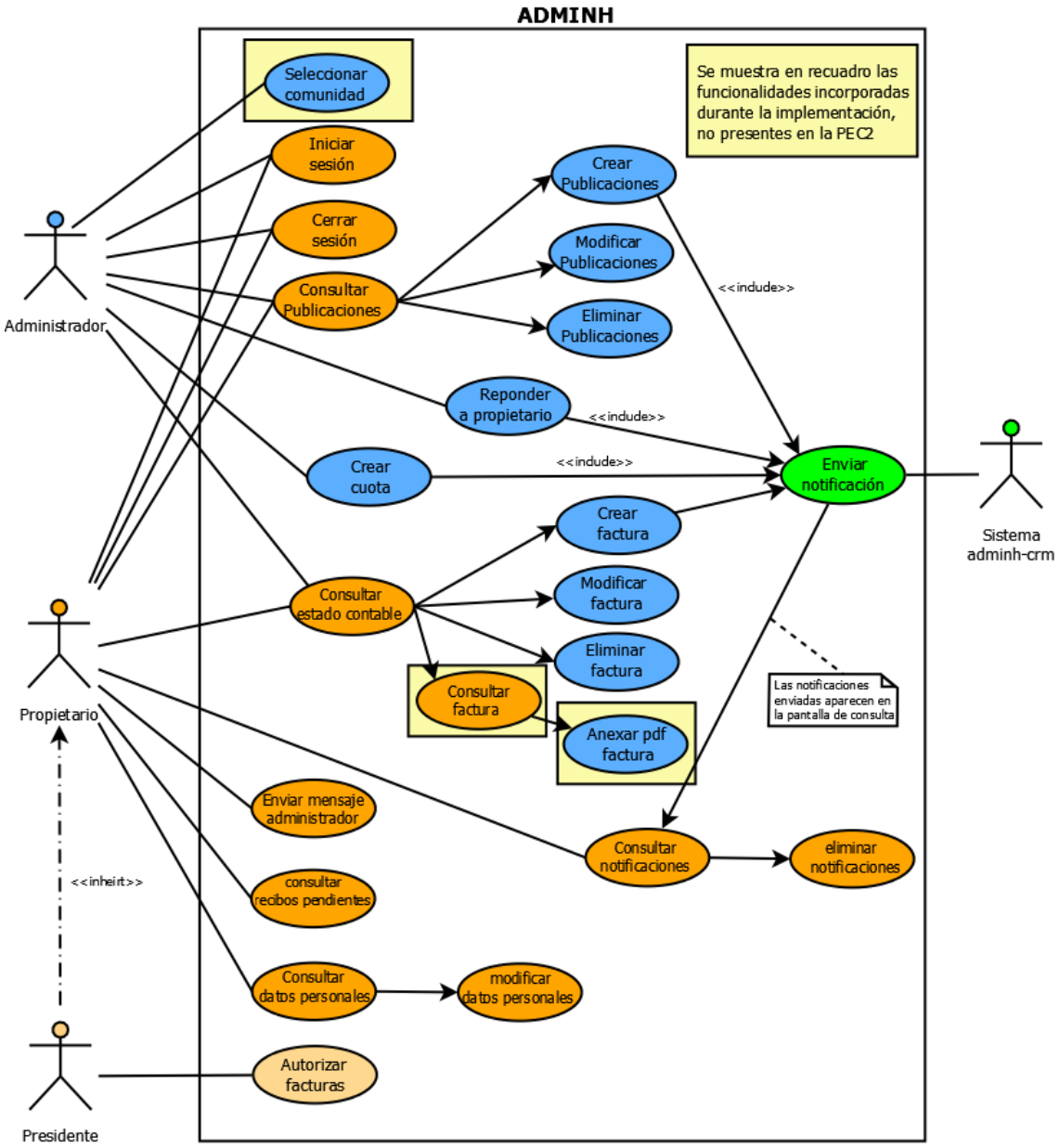


Fig 4 Diagrama de casos de uso

2.2 Fichas de casos de uso

Las fichas individualizadas para estos casos de uso se detallan a continuación. Para aquellos con acciones múltiples (consulta, alta, etc.) se detalla solo uno

de ellos. Asimismo, se subrayan y sombrean aquellas características que se han modificado una vez iniciada la implementación:

Id	CU01.1
Actores	Administrador, Propietario, Presidente
Caso de uso	Iniciar sesión en la aplicación
Descripción	Pantalla que muestra dos campos para que el usuario pueda introducir su nombre de usuario y contraseña y un botón que le permita iniciar sesión y acceder a la aplicación.
Precondiciones	El usuario tiene que estar previamente registrado en la aplicación.
Postcondiciones	Se crea un identificador para la sesión de usuario y se redirige a la pantalla principal de la aplicación, donde se mostrarán las opciones dependiendo del nivel que tenga el usuario.
Flujo principal	1. En la primera pantalla de la aplicación el usuario rellena los campos “Usuario” y “Contraseña” con sus datos de acceso. 2. Se hace clic en el botón “Acceder”. 3. El sistema redirige a la pantalla principal que muestra todas las opciones que el usuario tiene disponibles.
Flujo alternativo	1. Si el nombre de usuario o la contraseña son incorrectos, al clicar en el botón “Acceder” se permanecerá en la pantalla de *login* y se informará del error.
Prototipo	Captura 1: “*Login* usuario”

Id	CU01.2
Actores	Administrador, Propietario, Presidente
Caso de uso	Cerrar sesión en la aplicación
Descripción	En la pantalla principal de la aplicación, al lado del icono y el nombre de usuario, se muestra un botón de *logout* para que el usuario pueda cerrar su sesión.
Precondiciones	El usuario tiene que tener una sesión activa en la aplicación.
Postcondiciones	Se redirigirá a la pantalla de *login*.
Flujo principal	1. Se hace clic en el botón de *[logout]* situado en la pantalla principal. 2. Se cierra sesión y la aplicación redirige a la pantalla de *login*.
Flujo alternativo	-
Prototipo	Captura 2: *logout* en rectángulo rojo de la parte superior derecha.

Id	CU02
Actores	Administrador, Propietario, Presidente

Caso de uso	Gestionar una publicación en el tablón
Descripción	El usuario puede crear mensajes en el tablón que pueden ser leídos por los usuarios cuyos clientes pertenezcan a la comunidad de propietarios asociado al mismo.
Precondiciones	El usuario debe tener una sesión activa en la aplicación.
Postcondiciones	Alta (administrador): El sistema redirigirá a la pantalla que muestra todas las publicaciones de la comunidad. Se envía una notificación a los propietarios de la comunidad.
Flujo principal	Flujo para un alta (administrador): 1. En la pantalla principal, se clica en la opción “Tablón de anuncios”. 2. En esta pantalla se clica en el botón “Nueva publicación” y se introducen los siguientes datos: a. Título b. Mensaje c. Fecha inicio d. Fecha fin e. Fecha evento 3. Al hacer clic en el botón “Crear publicación”, el sistema redirige de nuevo a la lista de publicaciones de la comunidad seleccionada.
Flujo alternativo	Si se introduce un dato incorrecto (título o mensaje vacíos / fechas de inicio, fin o evento ya pasadas), al hacer clic en el botón “Publicar” no se redirigirá a ninguna otra pantalla y se informará del error detectado.
Prototipo	Captura 4: Tablón de anuncios en la pantalla de un propietario / presidente Captura 5: Tablón de anuncios en la pantalla del administrador, donde se encuentran los botones “Nueva publicación” y “Eliminar publicación”. Captura 6: Pantalla del administrador para añadir una nueva publicación.

Id	CU03
Actores	Administrador, Presidente, Propietario
Caso de uso	Gestionar una factura de una comunidad
Descripción	El administrador puede generar una factura de una comunidad, así como modificar sus datos o eliminarla.
Precondiciones	El usuario tiene que tener una sesión activa en la aplicación.
Postcondiciones	Alta (administrador): Una vez creada se redirigirá a la pantalla que muestra el listado de facturas pendientes de pago de la comunidad.
Flujo principal	Flujo para un alta (administrador): 1. En el menú principal se clica en la opción “Estado contable”.

	2. Se selecciona el presupuesto en el que queremos dar de alta la factura. 3. Se clica en el botón “Generar nueva factura” y se introducen los siguientes datos: a. Partida b. Proveedor c. Fecha Factura d. Número factura e. Descripción f. Importe 4. Al hacer clic en el botón “Generar factura”, el sistema redirige de nuevo al estado contable de la comunidad seleccionada, donde aparecerá la factura que hemos creado en el listado de “Facturas pendientes de pago”.
Flujo alternativo	Si se introduce un dato incorrecto (los campos fecha factura, descripción o importe se dejan vacíos), al hacer clic en el botón “Generar factura” no se redirigirá a ninguna otra pantalla y se informará del error detectado.
Prototipo	Captura 10: Pantalla de estado contable del administrador, donde se encuentra el botón “Generar nueva factura”. Captura 11: Pantalla “Generación de nueva factura”

Id	CU03.4 Funcionalidad añadida en la implementación
Actores	Administrador
Caso de uso	Anexar una factura de una comunidad
Descripción	El administrador puede anexar un fichero pdf en las facturas
Precondiciones	El usuario tiene que tener una sesión activa en la aplicación con el rango administrador La factura no puede tener un fichero anexado previamente.
Postcondiciones	Se refresca la página con el detalle de la factura mostrando un icono que enlaza al fichero pdf que se acaba de subir.
Flujo principal	Flujo para un alta: 1. En el menú principal se clica en la opción “Estado contable”. 2. Se clica en la descripción de una factura para visualizar su detalle. 3. Se clica en el botón “Examinar…” y se elige el fichero que queremos subir. 4. Se clica en el botón “Subir factura”.
Flujo alternativo	-
Prototipo	Captura 16: Pantalla que muestra el detalle de la factura con el botón “Subir factura”.

Id	CU04
Actores	Propietario Presidente
Caso de uso	Enviar mensaje al administrador
Descripción	Los propietarios y el presidente de una comunidad pueden enviar un mensaje al administrador. El mensaje quedará abierto hasta que se reciba la réplica del administrador.
Precondiciones	El usuario tiene que tener una sesión activa en la aplicación con nivel de presidente o propietario. No puede haber mensajes del mismo usuario pendientes de contestar.
Postcondiciones	Se muestra el listado de mensajes enviados al administrador.
Flujo principal	1. En el menú principal se clica en la opción “Consulta administrador”. 2. Dentro de la pantalla “Consulta administrador” se muestran los mensajes enviados previamente y la opción “Nueva consulta.”, que contiene los campos: a. Título b. Mensaje 3. Al hacer clic en el botón “Enviar mensaje” se actualiza el listado de mensajes, donde se muestra primero el mensaje que se acaba de enviar.
Flujo alternativo	Si se deja el campo del título o del mensaje vacíos se muestra un mensaje indicando el error.
Prototipo	Captura 8: Pantalla “Consulta administrador”

Id	CU05
Actores	Administrador
Caso de uso	Responder un mensaje de un propietario
Descripción	El administrador dispone de un panel en el que leer los mensajes que recibe de los propietarios y la opción de dar su réplica.
Precondiciones	El usuario tiene que tener una sesión activa con el nivel de administrador. Debe haber mensajes pendientes de respuesta.
Postcondiciones	Se muestra el listado de mensajes pendientes de contestar. Se envía una notificación al propietario conforme tiene una nueva respuesta pendiente de leer.
Flujo principal	1. En el menú principal se clica en la opción “Consulta administrador”. 2. Dentro de la pantalla “Consulta administrador” se muestran los mensajes pendientes de contestar, junto a un campo de texto “Respuesta” y un botón “Enviar respuesta” en cada u de ellos.

	3. Al rellenar el campo de texto “Respuesta” y clicar al botón “Enviar respuesta” se actualiza la página y ya no se muestra el mensaje. 4. Se envía notificación al propietario.
Flujo alternativo	Si se intenta contestar sin haber rellenado ningún texto en el campo “Respuesta” se informa del error.
Prototipo	Captura 9: Pantalla “Consulta administrador”

Id	CU06
Actores	Administrador
Caso de uso	Crear recibos de una comunidad
Descripción	El administrador tiene la posibilidad de crear recibos de los propietarios de las comunidades.
Precondiciones	El usuario tiene que tener una sesión activa con el nivel de administrador.
Postcondiciones	Se envía notificación a los propietarios de la comunidad para indicarles que tienen una cuota pendiente nueva.
Flujo principal	1. En el menú principal se clica en la opción “Generación cuota”. 2. Dentro de la pantalla “Generar cuota se introducen los siguientes datos: a. Concepto b. Fecha recibo c. Forma de pago (coeficiente / partes iguales) d. Importe total e. Importe por entidad (se rellena automáticamente al elegir un importe total y se puede modificar individualmente). 3. Al hacer clic en el botón “Generar recibos” se genera un recibo pendiente para cada vecino de la comunidad y se muestra un mensaje en pantalla indicando que se han realizado correctamente. 4. Se envía notificación al usuario para indicarle que tiene una nueva cuota.
Flujo alternativo	Si se dejan los campos de texto de “concepto” o “fecha recibo” vacíos se informará del error al hacer clic en el botón “Generar”. Debe haber al menos un importe en los campos de “Importes por entidad”.
Prototipo	Captura 12: pantalla “Generación cuota”

Id	CU07
Actores	Presidente
Caso de uso	Autorizar pago factura
Descripción	Una factura que pertenece a una partida del presupuesto que requiere autorización, únicamente se podrá pagar si se cuenta con la indicada autorización.
Precondiciones	El usuario tiene que tener una sesión activa con el nivel de presidente.

	La comunidad tiene que tener facturas pendientes de autorización.
Postcondiciones	La factura pendiente desaparece del listado de facturas pendientes de autorización del administrador.
Flujo principal	1. Desde el menú principal, el usuario entra en el menú “Autorizar facturas”. 2. Se muestra una lista de facturas pendientes de autorización con los datos, concretamente: a. Descripción b. Proveedor c. Número de factura d. Fecha factura e. Partida f. Importe g. Fichero PDF con documento de la factura. 3. Al hacer clic en el botón “Autorizar” se marcará como autorizada y se informará en pantalla.
Flujo alternativo	-
Prototipo	Captura 13: Pantalla “Autorizar facturas”

Id	CU08
Actores	Propietario Presidente
Caso de uso	Consultar los recibos pendientes
Descripción	Los propietarios de las comunidades pueden consultar los recibos pendientes desde el menú principal.
Precondiciones	El usuario tiene que tener una sesión activa con el nivel de presidente o propietario.
Postcondiciones	-
Flujo principal	1. En el menú principal se clica en la opción “Recibos pendientes”. 2. Se muestra un listado con los recibos pendientes de la entidad de las que sea propietario el usuario, mostrando los campos: a. Número de recibo b. Concepto c. Fecha d. Importe 3. Asimismo, se muestra el importe total y el número de cuenta de la comunidad para realizar el ingreso.
Flujo alternativo	Si no hubiera recibos pendientes, se informará con un mensaje con el texto “No hay recibos pendientes”
Prototipo	Captura 14: Pantalla “Recibos pendientes”

Id	CU09
Actores	Propietario

	Presidente
Caso de uso	Gestionar datos personales
Descripción	Los propietarios de las comunidades pueden consultar y modificar datos personales como su dirección, cuenta bancaria y datos de contacto.
Precondiciones	El usuario tiene que tener una sesión activa con el nivel de presidente o propietario.
Postcondiciones	Modificación datos: Se muestran los datos actualizados.
Flujo principal	Flujo modificación datos: 1. En el menú principal se clica en la opción “Datos personales”. 2. Se muestra detalle de los datos personales del cliente en campos de texto con la posibilidad de escribir en los datos modificables: a. Nombre b. NIF c. Dirección (MODIFICABLE) d. CP (MODIFICABLE) e. Municipio (MODIFICABLE) f. Provincia (MODIFICABLE) g. IBAN h. Teléfono (MODIFICABLE) i. Email (MODIFICABLE) 3. Si se clica en el botón “Guardar cambios” se actualizan los datos y se informa en pantalla.
Flujo alternativo	Si se deja vacío el campo Dirección, CP, Municipio o email se avisa del error.
Prototipo	Captura 7: Pantalla “Datos personales”

Id	CU10
Actores	Propietario Presidente
Caso de uso	Consultar la contabilidad de la comunidad
Descripción	Los propietarios pueden consultar el detalle de la contabilidad de la comunidad: saldo, gastos e ingresos.
Precondiciones	El usuario tiene que tener una sesión activa con el nivel de presidente o propietario.
Postcondiciones	-
Flujo principal	1. Al hacer *login* se accede al menú principal y se muestra por defecto la contabilidad de la comunidad. Si se está en otra pantalla, se puede clicar en el botón “Estado contable” para acceder. 2. Se muestran en pantalla los siguientes elementos: a. Presupuesto actual i. Nombre ii. Fecha inicio iii. Fecha final b. Saldo inicial

	c. Total ingresos d. Total gastos e. Saldo actual f. Relación de facturas pagadas por partida g. Relación de facturas pendientes de pago 3. Si se clica en la descripción de una factura se accede a su detalle.
Flujo alternativo	-
Prototipo	Captura 3: Pantalla “Estado contable”

Id	CU11
Actores	Propietario Presidente
Caso de uso	Añadir evento a Google Calendar
Descripción	En el tablón de anuncios de la comunidad el propietario de la finca puede clicar en una publicación para añadir el evento asociado a su calendario de Google.
Precondiciones	El usuario tiene que tener una sesión activa con el nivel de presidente o propietario. Debe haber publicaciones en el tablón.
Postcondiciones	Se redirige a cuenta personal de Google para añadir el evento.
Flujo principal	1. Desde el menú principal se acceder a la opción “Tablón de anuncios”. 2. Se elige un evento y se hace clic en el icono que indica que se agenda un evento. 3. Al hacer clic en el mencionado icono se redirige a la página de Google.
Flujo alternativo	-
Prototipo	Captura 5: Icono en la parte inferior derecha junto a la fecha del evento.

Id	CU12
Actores	Propietario Presidente
Caso de uso	Consultar notificaciones
Descripción	Cuando un usuario tiene notificaciones no leídas se muestra el icono de una campana en la parte superior de la pantalla, el cual permite acceder a su listado.
Precondiciones	El usuario tiene que tener una sesión activa con el nivel de presidente o propietario. Debe haber al menos una notificación.
Postcondiciones	Si se borra una notificación y pasan a haber 0 notificaciones, desparece el aviso superior que indica que se tienen notificaciones
Flujo principal	1. Se hace clic en el botón “Tienes notificaciones” de la parte superior. 2. Se listan las notificaciones con los siguientes elementos:

	a. Número de notificación b. Mensaje c. Botón “eliminar” 3. Si se hace clic en el botón “eliminar” se borra la notificación y desaparece de la lista.
Flujo alternativo	-
Prototipo	Captura 15: Icono de “Nuevas notificaciones” y pantalla con listado.

Id	CU13 Funcionalidad añadida en implementación
Actores	Administrador
Caso de uso	Cambiar de comunidad
Descripción	El administrador puede cambiar la comunidad sobre la cual se visualizan los datos.
Precondiciones	El usuario tiene que tener una sesión activa con el nivel de administrador.
Postcondiciones	Al elegir una comunidad se redirige a las publicaciones del tablón de la misma.
Flujo principal	1. Se clica en el *select* de la parte superior que por defecto tiene la comunidad actual. 2. Se selecciona la comunidad de la que se quieran visualizar los datos.
Flujo alternativo	-
Prototipo	Captura 17: Selector en la parte superior, dentro del recuadro rojo.

2.3 Modelo de pantallas (prototipo)

Se ha realizado un prototipo de la aplicación utilizando el software Figma [22].

Para su diseño se han tenido en cuenta los principios básicos de la interacción persona-ordenador, entre los cuales se pueden destacar los siguientes:

- Uso de **metáforas** [5] (iconos al lado de las opciones del menú de la izquierda y aviso de notificaciones, entre otros) para representar gráficamente las diferentes acciones que puede llevar a cabo el usuario.

- ***Affordance*** [6]: Diseño material en varios elementos como los botones, a los que se les ha dotado de efecto de profundidad para recalcar su funcionalidad como elemento que se puede clicar.

- El diseño ha sido pensado teniendo en cuenta el principio de **retroacción visual** [7], ya que cuando el usuario realiza ciertas acciones, como por ejemplo el eliminar una notificación (captura 15), automáticamente se actualiza el listado se actualiza para mostrar los nuevos cambios.

- **Restricción** [8], ya que los elementos que no son modificables (por ejemplo, el nombre o el NIF en los datos del cliente) se muestran con un sombreado para representar que no se pueden editar.

- Finalmente, también se ha tenido en cuenta el **modelo mental** [9], agrupando los elementos en tres regiones bien diferenciadas: la parte superior con detalles de la sesión, la parte izquierda con el menú de opciones, y el resto de pantalla en la que se visualizan los elementos de la opción escogida.

Se adjuntan imágenes de las diferentes pantallas:

Captura 1
Login

Fig 5 Login

Captura 2
Logout

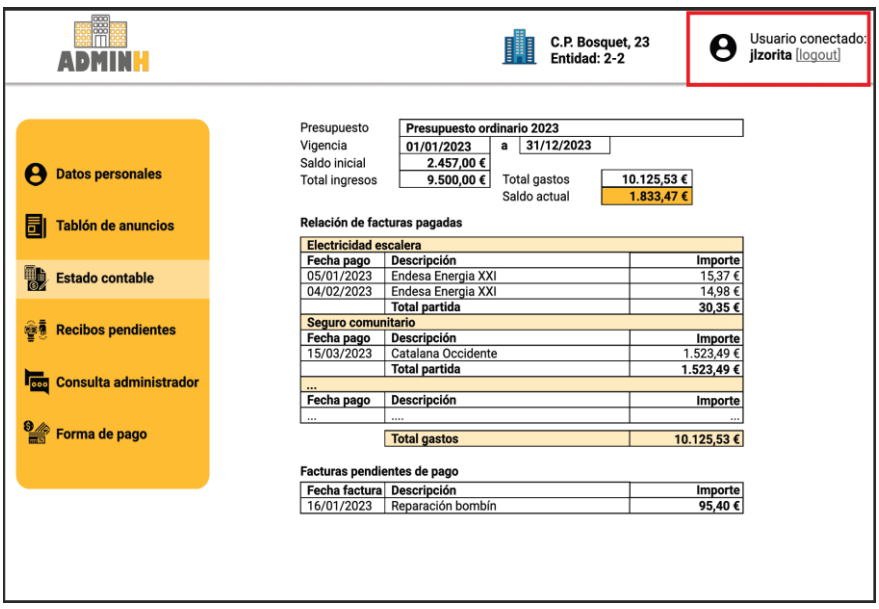


Fig 6 Logout

Captura 3
Estado contable

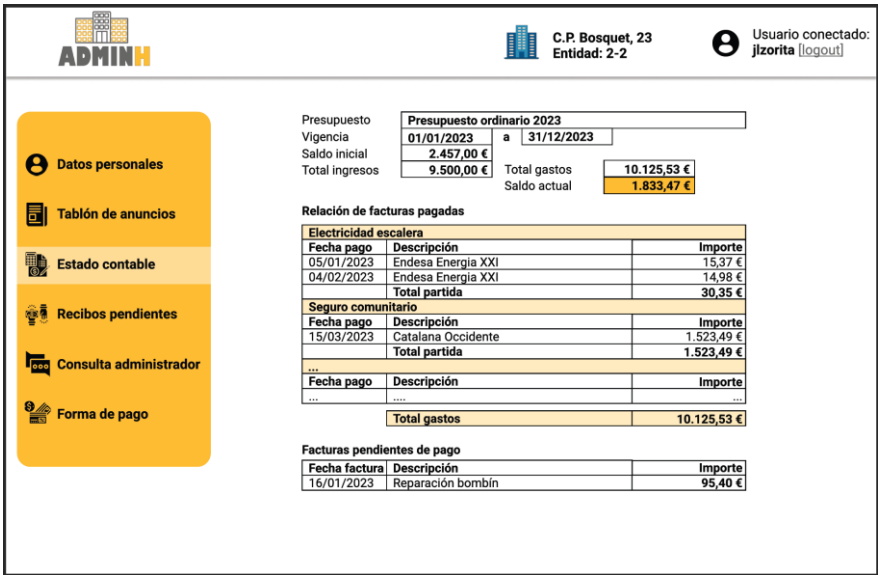


Fig 7 Estado contable

Captura 4
Tablón de anuncios

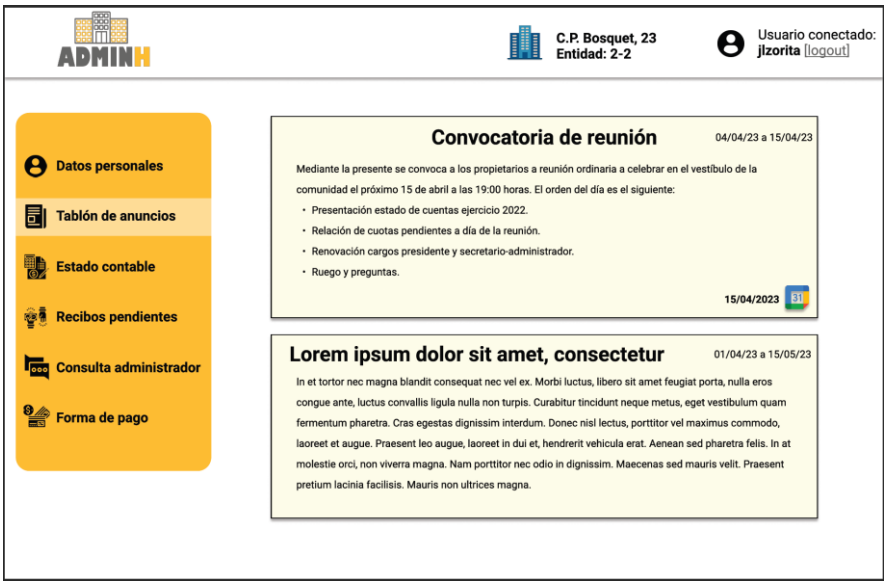


Fig 8 Tablón de anuncios

Captura 5
Tablón de anuncios administrador

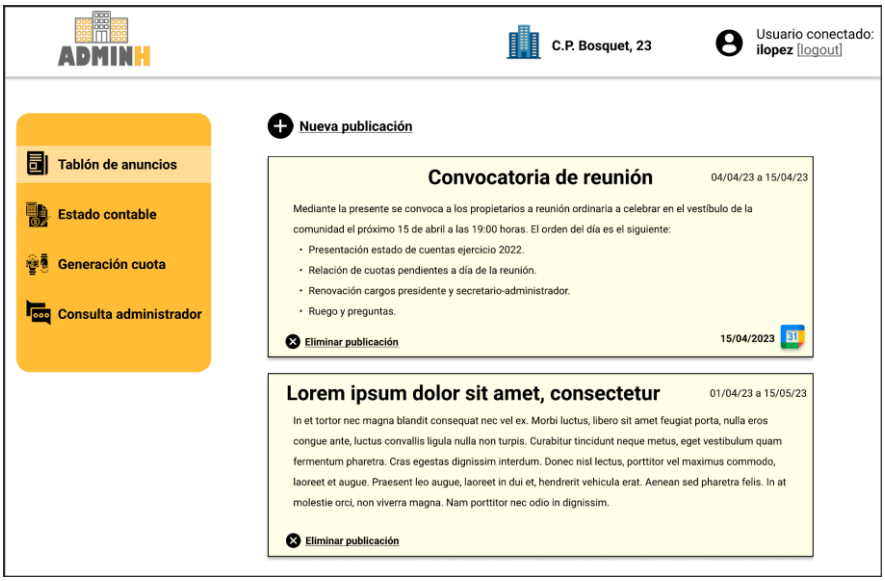


Fig 9 Tablón de anuncios admininstrador

Captura 6
Nueva publicación en el tablón

Fig 10 Nueva publicación en el tablón

Captura 7
Datos personales

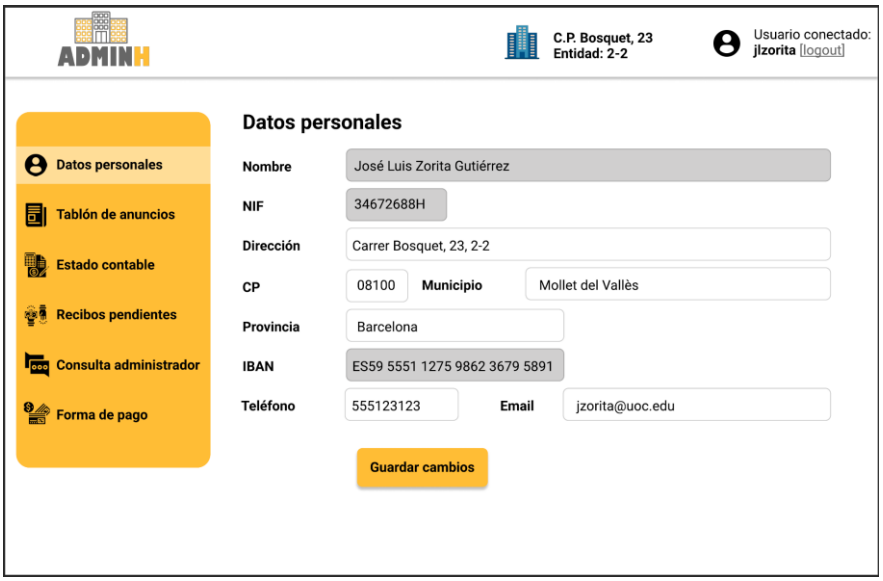


Fig 11 Datos personales

Captura 8
Mensaje al administrador

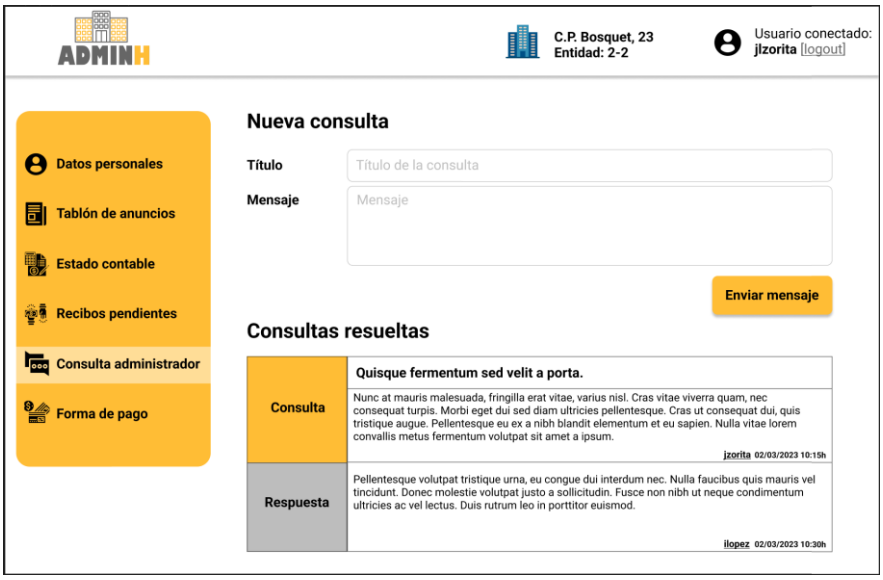


Fig 12 Mensaje al administrador

Captura 9
Respuesta del administrador

Fig 13 Respuesta del administrador

Captura 10
Opción para generar nueva factura

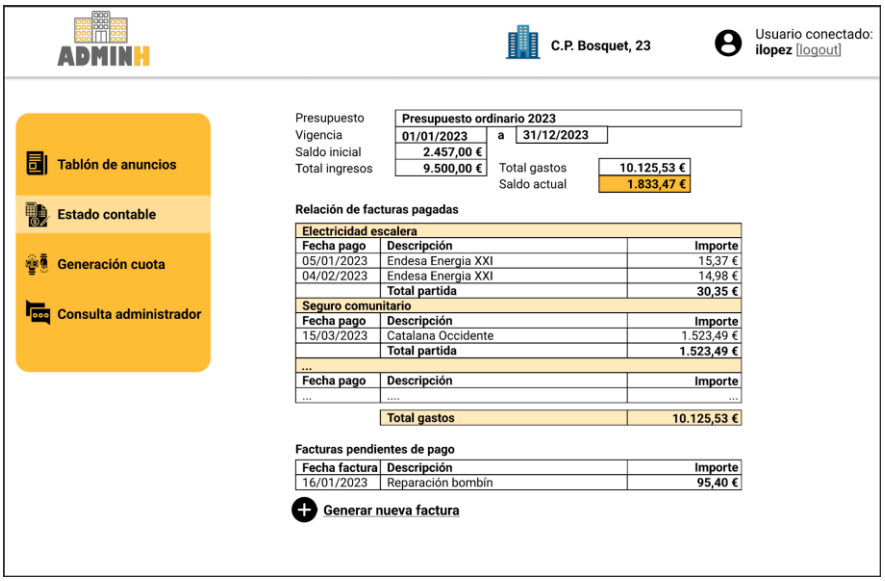


Fig 14 Generar factura

Captura 11
Generación de nueva factura

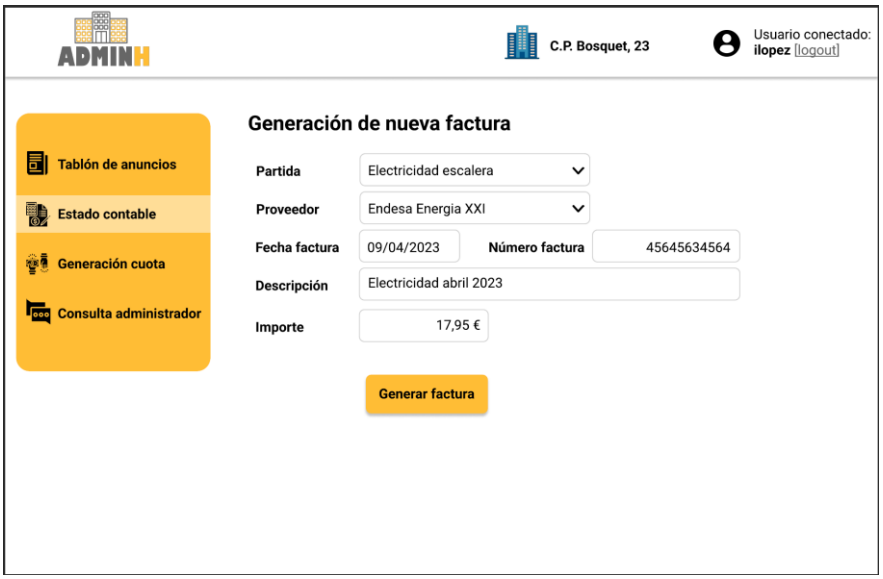


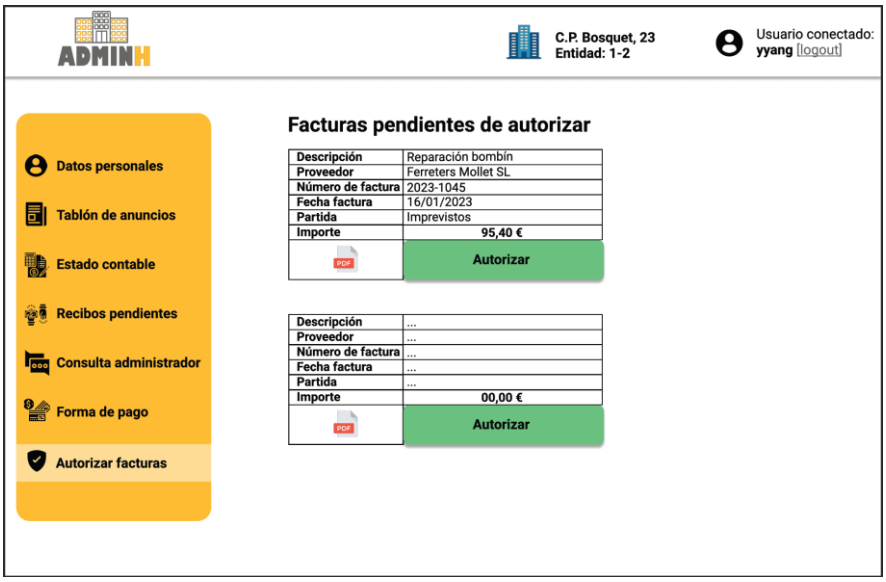
Fig 15 Generación de nueva factura

Captura 12
Generación de nueva cuota

Fig 16 Generación de nueva cuota

Captura 13
Autorización pago facturas

Fig 17 Autorización pago facturas



Captura 14
Consultar recibos pendientes

Fig 18 Consultar recibos pendientes

Captura 15
Consulta notificaciones

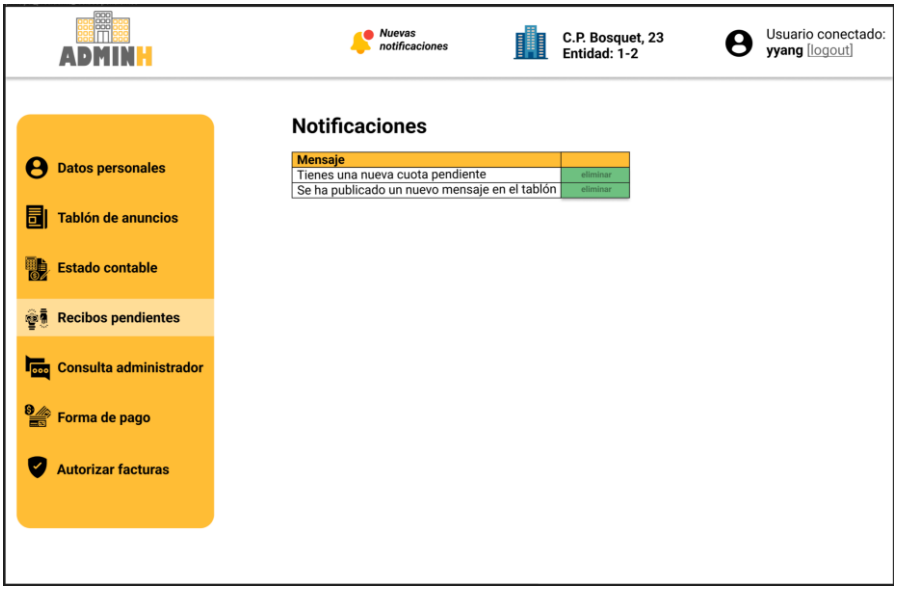


Fig 19 Consulta notificaciones

Captura 16
Subir fichero

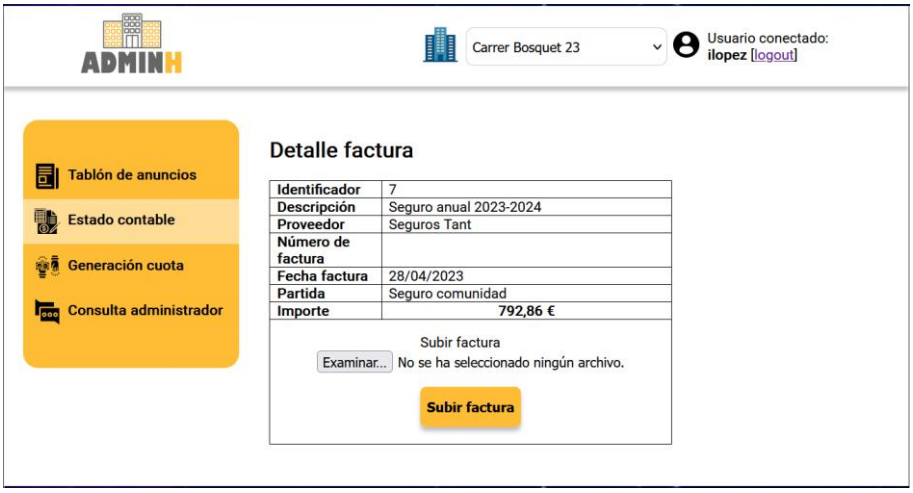


Fig 20 Subir fichero

Captura 17
Selección Comunidad

Fig 21 Selección comunidad

2.4 Diagrama de arquitectura

De modo general, la arquitectura de la aplicación seguirá el siguiente esquema:

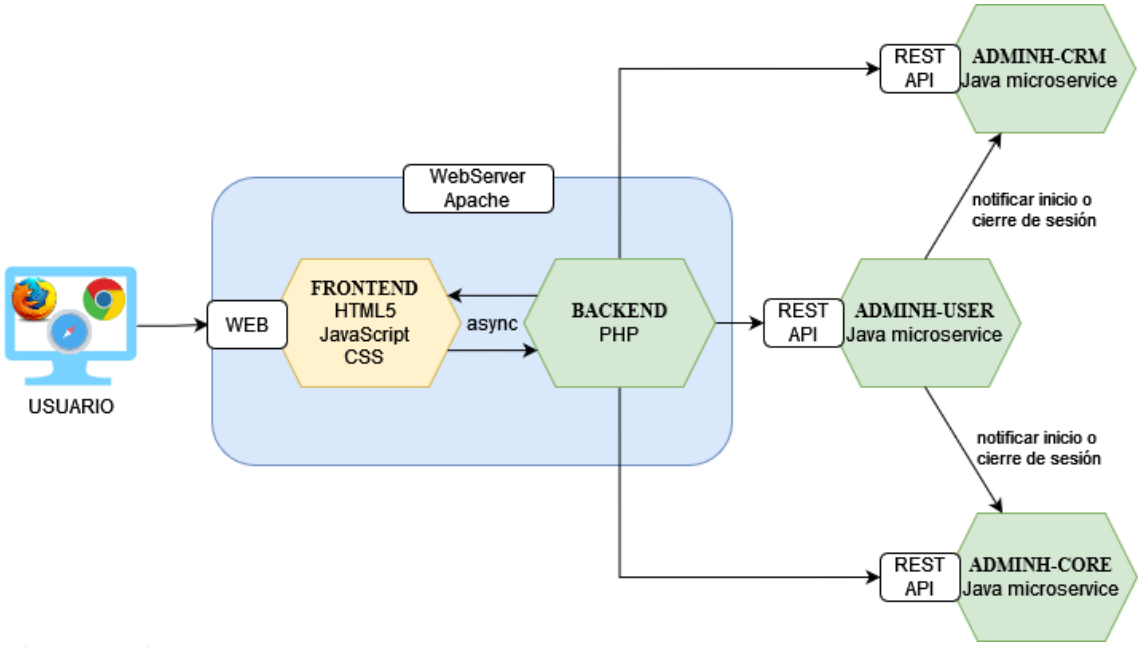


Fig 22 Arquitectura proyecto

Esta arquitectura la podemos dividir en dos partes:

- **Microservicios con servicios API REST**. Para lanzar los microservicios y el resto de servicios necesarios para el correcto funcionamiento de la aplicación se utilizará Docker Compose [11], que permitirá ejecutar varios contenedores de forma centralizada mediante su definición en un único fichero.

  En concreto, se utilizará un fichero YAML (`docker-compose.yml`) en el que se definirán los siguientes servicios:

  - **adminh-user**: Microservicio de usuarios.
  - **db-user**: Base de datos del microservicio de usuarios.
  - **adminh-crm**: Microservicio de clientes.
  - **db-crm**: Base de datos del microservicio de clientes.
  - **adminh-core**: Microservicio con lógica de negocio.
  - **db-core**: Base de datos del microservicio de lógica de negocio.
  - **kafka**: Será necesario un *Producer* en el microservicio adminh-user para poder notificar los datos de sesión del usuario en su inicio y cierre a los otros dos microservicios.
  - **zookeeper**: Gestión *brokers* de Kafka.
  - **adminer**: Gestión web de las bases de datos de los microservicios.

- **Aplicación web**. Con el fin de que no haya dependencia entre los servicios API REST ofrecidos por los microservicios y la aplicación Web que actuará como cliente (y así conseguir desacoplamiento entre ambos), se ha optado por desarrollar la web de la aplicación en el entorno XAMPP, que lanzará el servidor web *Apache* con el lenguaje php 8 habilitado.

Fig 23 XAMPP

  La comunicación entre la parte *backend* de la aplicación (php) y la parte *frontend* (html, CSS y Javascript) permitirá el desarrollo de una aplicación web que podrá consumir los servicios de los API REST y ofrecer información de forma rápida y asíncrona.

  Actualización: También se ha utilizado la librería **cURL** de php para las operaciones de transferencia de datos de los métodos POST del API REST.

2.5 Microservicios

Si entramos en detalle en los microservicios, estos utilizarán las siguientes tecnologías:

Lenguaje / kit de desarrollo	Java / OpenJDK 11	OpenJDK *Fig 24 OpenJDK*
Herramienta de gestión del proyecto	Apache Maven	Maven *Fig 25 Maven*
Persistencia /manipulación de datos	Java Persistence API / Spring JPA repository	spring *Fig 26 Spring*
Base de datos	postgresql	PostgreSQL *Fig 27 PostgreSQL*
Productor / consumidor mensajes	Kafka	APACHE kafka *Fig 28 Kafka*
Automatización en configuración del proyecto	Spring Boot	spring boot *Fig 29 Spring Boot*
***Template* para uso de Kafka en Java**	Spring para Apache Kafka	spring APACHE kafka
Librería para uso de anotaciones	lombok	Project Lombok *Fig 30 Project Lombok*

Inicialización base de datos	Hibernate	HIBERNATE *Fig 31 Hibernate*

Las tablas que conformarán las bases de datos de los diferentes microservicios estarán definidas por las entidades JPA [21] que se incluirán en cada uno de ellos, y que serán las siguientes:

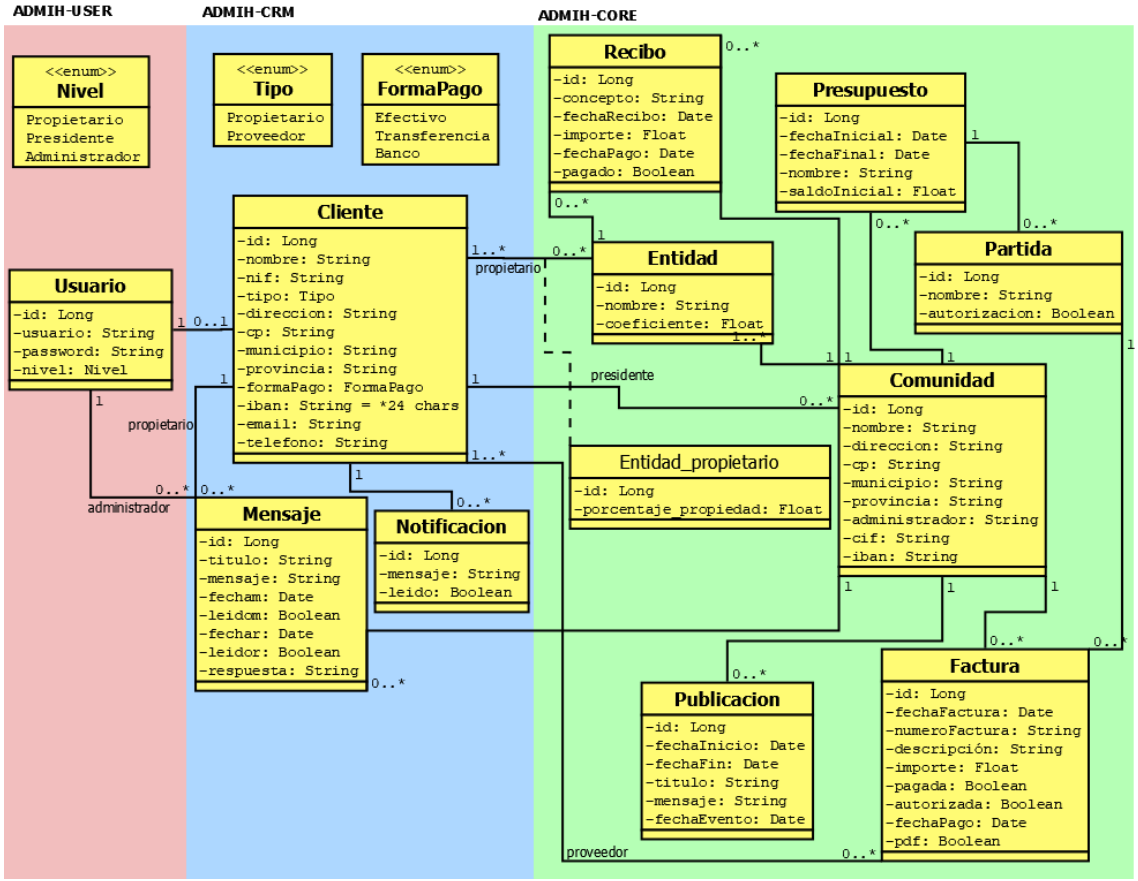


Fig 32 Entidades microservicios

Nota: Se trata de la versión definitiva, y que ha sufrido cambios respecto a la mostrada en la PEC2. La novedad más destacada es la inclusión de la tabla `Entidad_propietario` que tiene la función de relacionar los clientes con las entidades de las que son propietarios y asignarles un porcentaje de propiedad.

El esquema se completa con las clases utilizada para la gestión de las sesiones de usuario. Para conseguir tratar los datos de sesión de manera efectiva, se ha creado la clase `SesionData` en todos los microservicios. Esta clase tiene el objetivo de que los tres microservicios puedan tratar, en un único objeto enviado mediante el *kafka Template*, toda la información necesaria para manipular la sesión. Mediante la variable `alta` se identifica si lo que queremos es iniciar o cerrar una sesión.

El microservicio adminh-user es el productor. Cuando se efectua un login envía los datos a los dos consumidores de mensajes, los microservicios adminh-crm y adminh-core, para que añadan los datos de la sesión.

El esquema que sigue es el siguiente:

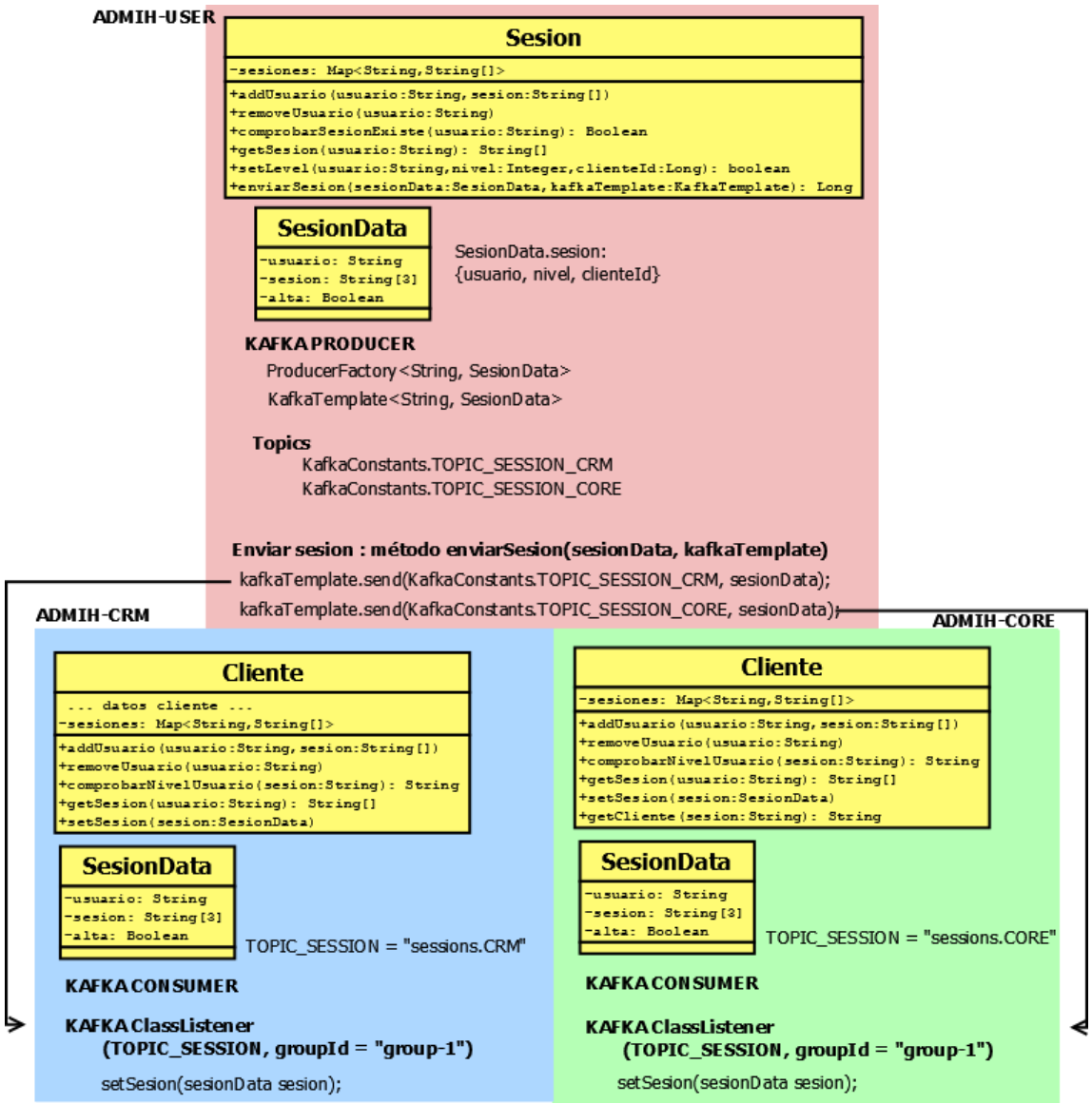


Fig 33 Clases sesion y Kafka

Desde el punto de vista de la descomposición en un patrón de sub-dominios [19], la relación sería la siguiente:

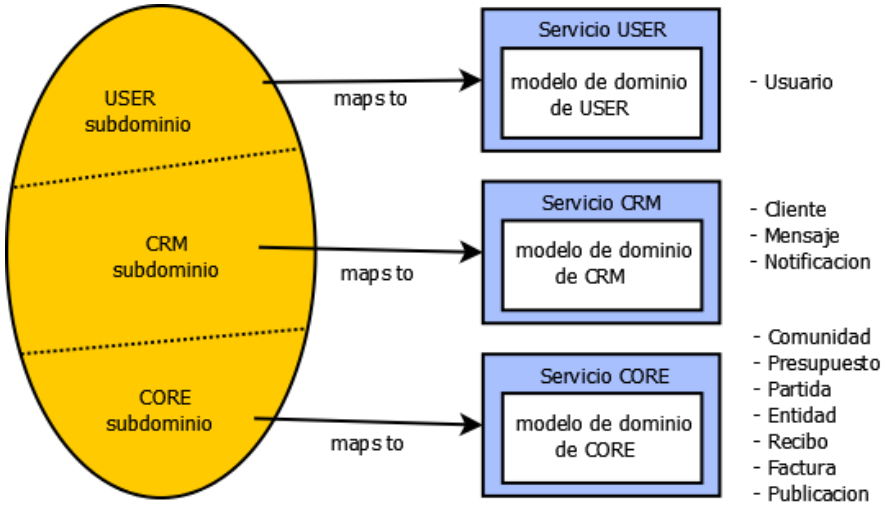


Fig 34 Descomposición en subdominios

Cuando se inicie sesión en la aplicación, el identificador de la sesión será guardado en un objeto del tipo `Map<K,V>` (donde la clave será el nombre de usuario y el valor los datos de sesión indicados en la figura 24) de la clase `Sesion` o `Cliente` dentro de todos los microservicios, de tal forma que cuando se ejecute un método REST se pueda verificar que el usuario tiene permisos para ello.

El código de sesión será creado en php [`session_start()`] y enviado al microservicio adminh-user con la operación POST *login*. Después será propagado al resto mediante su productor de Kafka.

2.6 Operaciones y consultas del sistema

Las operaciones y consultas identificadas para cada microservicio son las que se listan a continuación. Se remarcan aquellas que no habían sido contempladas inicialmente:

Servicio	Operaciones y consultas
USER *UserService*	**login**(LoginRequest loginRequest) : int **logout**(Stringusuario) : Boolean **buscaUsuario**(String usuario) : Optional<Usuario>
CRM *CrmService*	**modificarDatosCliente**(UpdateClienteRequest clienteRequest) : Boolean **enviarMensaje**(MensajeRequest mensajeRequest) : Boolean **responderMensaje**(RespuestaRequest respuestaRequest) : Boolean **eliminarNotificacion**(Long id) : Boolean **crearNotificacion**(NotificacionRequest notificacionRequest) : Boolean **buscaClientePorId**(Long id) : Optional<Cliente> **buscaClienteporUsuario**(String usuario) : Optional<Cliente> **buscaDatosCliente**() **buscaMensajesNoLeidosPorUsuario**(String usuario) : List<Mensaje> **buscaMensajesNoLeidosPorComunidad**(Long comunidad) : List<Mensaje> **buscaNotificacion**(Long id) : Optional<NotificacionEntity> **buscaMensajesContestadosPorUsuario**(String usuario) : List<Mensaje> **buscaProveedores**() : List<Cliente> **buscaNotificacionesPorUsuario**(String usuario) : List<Notificacion>
CORE *CoreService*	**crearPublicacion**(PublicacionRequest publicacionRequest) : Boolean **eliminarPublicacion**(Long id) : Boolean **crearRecibo**(ReciboRequest reciboRequest) : Boolean **crearFactura**(FacturaRequest facturaRequest, Boolean autorizada) : Boolean **autorizarFactura**(Long id) : Boolean **subirFactura**(Long id) : Boolean **buscaComunidad**(Long comunidad) : Optional<Comunidad> **buscaEntidadesPorCliente**(Long cliente) : List<EntidadPropietario>

	buscaRecibosPagadosCom(Long comunidadId, String anualidad) : List<Recibo> **buscaEntidadesComunidad**(Long comunidad) : List<Entidad> **buscaPublicacionesCom**(Long comunidadId) : List<Publicacion> **buscaFacturasPorPartida**(Long partidaId) : List<Factura> **buscaFacturasPagadas**(Long partidaId) : List<Factura> **buscaFacturasPendientes**(Long comunidadId) : List<Factura> **facturasPendientesAutCom**(Long comunidadId) : List<Factura> **buscaRecibosPendientes**(Long comunidadId) : List<Recibo> **buscaRecibosPendEntidad**(Long entidadId) : List<Recibo> **buscaDatosPresupuesto**(Long comunidadId) : List<Presupuesto> **buscaPartidasPresupuesto**(Long presupuestoId) : List<Partida> **buscaPartidaReqAut**(Long partidaId) : Boolean **buscaComunidadPorEntidadId**(Long entidadId) : Long **buscaComunidadesAdministradas**(String administrador) : List<Comunidad> **buscaClientesComunidad**(Long comunidadId) : List<EntidadPropietario> **esVecino**(Long clienteId, Long comunidadId) : Boolean **buscaComunidadPartida**(Long partidaId) : Long **buscaComunidadFactura**(Long id) : Long **esPresidente**(Long clienteId, Long comunidadId) : Boolean **esPropietario**(Long clienteId, Long entidadId) : Boolean **buscaComunidadPresupuesto**(Long presupuestoId) : Long **buscaDetalleFactura**(Long facturaId) : Factura
CORE *SubirFichero Service*	**guardar**(MultipartFile fichero, String nombre) : Boolean **cargar**(String nombreFichero) : Resource

2.7 Métodos de los servicios API REST

Los métodos API REST que estarán disponibles en los microservicios son las que se listan a continuación.

De forma general, se recibirá una respuesta 200 OK cuando la información enviada sea correcta, 401 UNAUTHORIZED cuando los datos de sesión o bien sean incorrectos o bien el usuario no tenga permiso para ejecutar la orden y 500 INTERNAL SERVER ERROR cuando el error sea de otra naturaleza (por ejemplo, que el cuerpo del mensaje no tenga el formato correcto).

Actualización: Se obtiene el código 404 NOT FOUND cuando la petición recibe como respuesta un conjunto vacío de datos y 400 BAD REQUEST cuando falta algún dato en la petición.

Micoservicio: adminh-user

Descripción	**1. Inicio de sesión.** Método para que el usuario pueda iniciar sesión en la aplicación. El código de sesión creado por PHP es enviado al microservicio adminh-user, que a su vez lo envía al resto mediante Kafka.
Tipo	**POST**

Ruta	/login
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	Body: {"usuarioData": {usuario, password}, sesion}
Parámetros respuesta	- Nivel

Descripción	**2. Cierre de sesión.** Método para que el usuario pueda cerrar sesión en la aplicación. Al igual que con el *login*, también existe comunicación en Kafka para que los microservicios eliminen el código de sesión.
Tipo	**POST**
Ruta	**/logout/{usuario}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	usuario
Parámetros respuesta	-

Micoservicio: adminh-crm

Descripción	**3. Consulta datos personales.** Devuelve los datos del cliente a partir de su usuario.
Tipo	**GET**
Ruta	**/usuario/{usuario}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	usuario, sesion
Parámetros respuesta	Body: {id, nombre, nif, tipo, direccion, cp, municipio, provincia, formaPago, iban, email, teléfono}

Descripción	**4. Modificar datos personales.** Método PUT para modificar datos personales del cliente.
Tipo	**PUT**
Ruta	**/usuario/actualizar**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	Body: {id, direccion, cp, municipio, provincia, teléfono, email}

	- Sesión
Parámetros respuesta	-

Descripción	**5. Enviar mensaje al administrador.** Método POST con el que el usuario puede enviarle mensajes al administrador. Por defecto quedarán como pendientes de respuesta.
	Método
Tipo	**POST**
Ruta	**/mensaje**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 AUTHORIZED, 500 INTERNAL SERVER ERROR

Parámetros entrada	`Body:`
	`{titulo, mensaje, clienteId, comunidadId, administrador}`
	- sesion

Parámetros respuesta	-

Descripción	**6. Responder mensaje de propietario**. Método PUT que permite al administrador dar respuesta a los mensajes pendientes, actualizando los campos del mismo.
Tipo	**PUT**
Ruta	**/respuesta**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR

Parámetros entrada	`Body:`
	`{respuesta, mensajeId}`
	- sesion

Parámetros respuesta	-

Descripción	**7. Consultar mensajes no respondidos por usuario.** Devuelve la lista de mensajes de un usuario que todavía no han sido respondidos por el administrador.
Tipo	**GET**
Ruta	**/mensaje/usuario/{usuario}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	usuario, tipo {NOLEIDO / CONTESTADOS}, sesion

Parámetros respuesta	`Body:`
	`{`
	`{id, titulo, mensaje, fechaM, leidoM, respuesta, fechaR,leidoR,`

	administrador, comunidadId, cliente} ,{…} }

Descripción	**8. Consultar mensajes no respondidos por comunidad**. Permite al administrador visualizar todos los mensajes pendientes de contestar de una comunidad concreta.
Tipo	**GET**
Ruta	**/mensaje/comunidad/{comunidad}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	comunidadId, sesion

Parámetros respuesta	Body:
	{
	{id, titulo, mensaje, fechaM, leidoM, respuesta, fechaR,leidoR, administrador, comunidadId, cliente}
	,{…}
	}

Descripción	**9. Consultar notificaciones usuario.** Método con el que el usuario puede listar todas sus notificaciones.
Tipo	**GET**
Ruta	**/notificacion/{usuario}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	usuario, sesion
Parámetros respuesta	Body: {{id, mensaje, leido, cliente}, {…}}

Descripción	**10. Crear notificación – Incorporada PEC3.** Utilizado para crear una nueva notificación asociada a un cliente.
Tipo	**POST**
Ruta	**/notificacion/crear**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	Body: {mensaje, cliented}
Parámetros respuesta	-

Descripción	**11. Eliminar notificación**. Método para suprimir una notificación.
Tipo	**DELETE**

Ruta	**/notificacion/eliminar/{id}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	id, sesion
Parámetros respuesta	-

Descripción	**12. Obtener datos cliente – Incorporada PEC3.** Permite obtener la información de cualquier tipo de cliente, ya sea propietario o proveedor.
Tipo	**GET**
Ruta	**/cliente/id/{id}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	clienteId, sesion
Parámetros respuesta	`Body:` `{id, nombre, nif, tipo, direccion, cp, municipio, provincia, formaPago, iban, email, telefono}`

Descripción	**13. Obtener proveedores – Incorporada PEC3.** Devuelve la lista completa de clientes del tipo proveedor.
Tipo	**GET**
Ruta	**/proveedores**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	sesion

Parámetros respuesta	`Body:`
	`{`
	`{id, nombre, nif, tipo, direccion, cp, municipio, provincia, formaPago, iban, email, telefono}`
	`,{…}`
	`}`

Micoservicio: adminh-core

Descripción	**14. Consultar entidades / comunidad cliente.** Devuelve una lista de entidades que tiene un cliente de todas las comunidades, así como el porcentaje de titularidad.
Tipo	**GET**
Ruta	**/comunidades/{clienteId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED

	404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	clienteId, sesion

Parámetros respuesta	Body:
	{
	{id, clienteId, entidad, porcentajePropiedad}
	,{…}
	}

Descripción	**15. Consultar publicaciones por comunidad.** Devuelve todas las publicaciones vigentes de una comunidad. Es decir, que la fecha del evento todavía no haya pasado.
Tipo	**GET**
Ruta	**/publicacion/{comunidadId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOTFOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	comunidadId, sesion

Parámetros respuesta	Body:
	{
	{id, fechaInicio, fechaFin, titulo, mensaje, fechaEvento}
	,{…}
	}

Descripción	**16. Crear publicación.** Utilizado por el administrador para crear nuevas publicaciones en el tablón.
Tipo	**POST**
Ruta	**/publicacion/crear**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	Body: {titulo, mensaje, fechaInicio, fechaFin, fechaEvento, comunidadId}
Parámetros respuesta	-

Descripción	**17. Eliminar publicación.** Utilizado por el administrador para eliminar publicaciones vigentes del tablón.
Tipo	**POST**
Ruta	**/publicacion/eliminar/{id}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR

Parámetros entrada	id, sesion
Parámetros respuesta	-

Descripción	**18. Consultar facturas por partida presupuesto.** Devuelve las facturas de una partida. Con el atributo pendientes {“si”,”no”} se puede especificar si se quiere que se incluyan las facturas pendientes o no respectivamente.
Tipo	**GET**
Ruta	**/factura/{partidaId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	partidaId, pendientes {“si”, “no”}, sesion

Parámetros respuesta	`Body:`
	`{`
	`{id, fechaFactura, numeroFactura, descripcion, importe, pagada, autorizada, fechaPago, pdf, proveedorId}`
	`,{…}`
	`}`

Descripción	**19. Consultar facturas pendientes por comunidad.** Devuelve todas las facturas pendientes de pago de una comunidad.
Tipo	**GET**
Ruta	**/factura/pendientes/{comunidadId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	comunidadId, sesion

Parámetros respuesta	`Body:`
	`{`
	`{id, fechaFactura, numeroFactura, descripcion, importe, pagada, autorizada, fechaPago, pdf, proveedorId}`
	`,{…}`
	`}`

Descripción	**20. Crear factura comunidad.** Método utilizado por el admin para generar una nueva factura de una comunidad y partida de presupuesto.
Tipo	**POST**
Ruta	**/factura/crear**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR

Parámetros entrada	Body: {fechaFactura, numeroFactura, descripcion, importe, proveedorId, comunidadId, partidaId} - sesion
Parámetros respuesta	-

Descripción	**21. Crear recibo entidad comunidad.** Método utilizado por el administrador para generar un nuevo recibo de una entidad de una comunidad.
Tipo	**POST**
Ruta	**/recibo/crear**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	Body: {fechaRecibo, concepto, importe, entidadId, comunidadId} - sesion
Parámetros respuesta	-

Descripción	**22. Autorizar pago factura.** Permite al presidente de una comunidad marcar una factura como autorizada para su pago.
Tipo	**PUT**
Ruta	**/factura/autorizar/{id}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	facturaId, sesion
Parámetros respuesta	-

Descripción	**23. Consulta recibos pendientes por entidad.** Devuelve la lista de recibos pendientes de una entidad de una comunidad.
Tipo	**GET**
Ruta	**/recibo/pendientes/entidad{entidadId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	entidadId, sesion

Parámetros respuesta	Body:
	{
	{id, concepto, fechaRecibo, importe, fechaPago, pagado, comunidad, entidad}
	,{…}
	}

Descripción	**24. Consultar presupuestos.** Devuelve una lista de presupuestos y sus datos de una comunidad.
Tipo	**GET**
Ruta	**/presupuesto/{comunidadId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	comunidadId, sesion

Parámetros respuesta	`Body:`
	`{`
	`{id, fechaInicial, fechaFinal, nombre, saldoInicial}`
	`,{…}`
	`}`

Descripción	**25. Consultar partidas presupuesto.** Devuelve la lista de partidas de un presupuesto y si sus facturas necesitan autorización.
Tipo	**GET**
Ruta	**/presupuesto/partidas/{presupuestoId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	presupuestoId, sesion

Parámetros respuesta	`Body:`
	`{`
	`{id, nombre, autorizacion}`
	`,{…}`
	`}`

Descripción	**26. Consultar facturas pendientes de autorizar por comunidad.** Devuelve la lista de facturas de una comunidad que están pendientes de recibir autorización por parte del presidente.
Tipo	**GET**
Ruta	**/factura/autorizar/pendientes/{comunidadId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	comunidadId, sesion

Parámetros respuesta	`Body:`
	`{`
	`{id, fechaFactura, numeroFactura, descripcion, importe, pagada, autorizada, fechaPago, pdf, proveedorId}`

	,{…} }

Descripción	**27. Consultar comunidades de un administrador – Incorporada PEC3.** Devuelve la lista de comunidades que son administradas por un administrador.
Tipo	**GET**
Ruta	**/comunidades/administrador/{administrador}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	administrador, sesion

Parámetros respuesta	Body:
	{
	{id, nombre, direccion, cif, cp, municipio, provincia, iban, presidenteId, administrador}
	,{…}
	}

Descripción	**28. Consultar entidades de una comunidad – Incorporada PEC3.** Devuelve una lista con las entidades de una comunidad.
Tipo	**GET**
Ruta	**/comunidad/entidades/{comunidadId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	comunidadId, sesion

Parámetros respuesta	Body:
	{
	{id, nombre, coeficiente}
	,{…}
	}

Descripción	**29. Consultar datos de una comunidad – Incorporada PEC3.** Devuelve toda la información relativa a una comunidad.
Tipo	**GET**
Ruta	**/comunidades/{id}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	id, sesion
Parámetros respuesta	Body: {id, nombre, direccion, cif, cp, municipio, provincia, iban,

	presidenteId, administrador}

Descripción	**30. Consultar id de comunidad a partir de la entidad – Incorporada PEC3.** Dado el identificador de una entidad, devuelve el identificador de su comunidad.
Tipo	**GET**
Ruta	**/comunidadEntidad/{entidadId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	entidadId, sesion
Parámetros respuesta	Long

Descripción	**31. Consultar detalle de factura – Incorporada PEC3.** Devuelve los datos de una factura.
Tipo	**GET**
Ruta	**/factura/detalle/{facturaId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	facturaId, comunidadId, sesion
Parámetros respuesta	Body: {id, fechaFactura, numeroFactura, descripcion, importe, pagada, autorizada, fechaPago, pdf, proveedorId}

Descripción	**32. Consultar recibos pagados por año – Incorporada PEC3.** Devuelve los recibos pagados en un año concreto en una comunidad.
Tipo	**GET**
Ruta	**/recibo/{comunidadId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	comunidadId, anualidad, sesion

Parámetros respuesta	Body:
	{
	{id, concepto, fechaRecibo, importe, fechaPago, pagado, comunidad, entidad}
	,{…}
	}

Descripción	**33. Consultar recibos pendientes por comunidad – Incorporada PEC3.** Devuelve la lista de recibos pendientes de una comunidad.
Tipo	**GET**
Ruta	**/recibo/pendientes/{comunidadId}**

Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	comunidadId, sesion

Parámetros respuesta	Body:
	{
	{id, concepto, fechaRecibo, importe, fechaPago, pagado, comunidad, entidad}
	,{…}
	}

Descripción	**34. Consultar clientes de una comunidad – Incorporada PEC3.** Método que devuelve la lista de clientes que son propietarios de una entidad de la comunidad.
Tipo	**GET**
Ruta	**/comunidad/clientes/{comunidadId}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED 404 NOT FOUND, 500 INTERNAL SERVER ERROR
Parámetros entrada	comunidadId, sesion

Parámetros respuesta	Body:
	{
	{id, clienteId, entidad, porcentajePropiedad}
	,{…}
	}

Descripción	**35. Visualizar pdf factura – Incorporada PEC3**
Tipo	**GET**
Ruta	**/factura/ver/{filename:.+}**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	filename, comunidadId, sesion
Parámetros respuesta	Resource (fichero .pdf)

Descripción	**36. Subir pdf factura – Incorporada PEC3**
Tipo	**GET**
Ruta	**/factura/subir**
Respuesta estado	200 OK
Respuesta estado error	400 BAD REQUEST, 401 UNAUTHORIZED, 500 INTERNAL SERVER ERROR
Parámetros entrada	(MultipartFile) factura, id, sesion
Parámetros respuesta	Si se sube correctamente se devuelve el texto “fichero subido correctamente”

2.8 Control de sesión

Las operaciones del API REST se han diseñado para que solo puedan ser ejecutadas por los usuarios que tengan los permisos necesarios.

Estos permisos son gestionados gracias a los datos de sesión almacenados en cada microservicio cuando se inicia sesión en la aplicación.

El diagrama siguiente muestra los permisos desglosados según el grupo de usuarios que pueden ejecutar cada método del API REST, siguiendo la numeración establecida en el punto anterior:

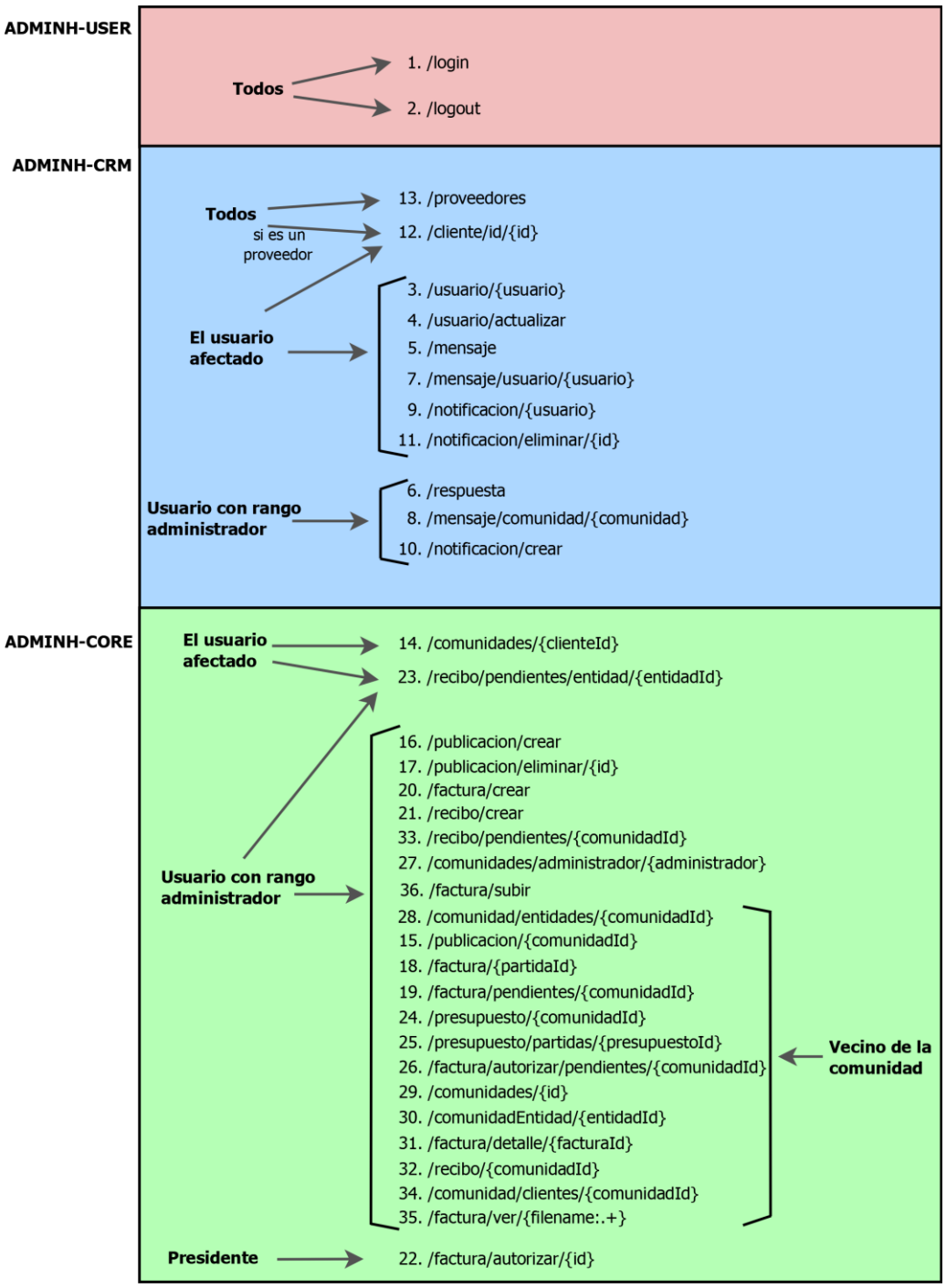


Fig 35 Permisos servicios REST

2.9 Arquitectura hexagonal

Una vez identificados los actores, las entidades que conforman la base de datos, la comunicación entre microservicios mediante Kafka y los métodos del API REST de cada microservicio, podemos definir cómo será la arquitectura hexagonal de cada uno de ellos.

Se adjunta diagrama con el diseño de la arquitectura hexagonal de todos los microservicios:

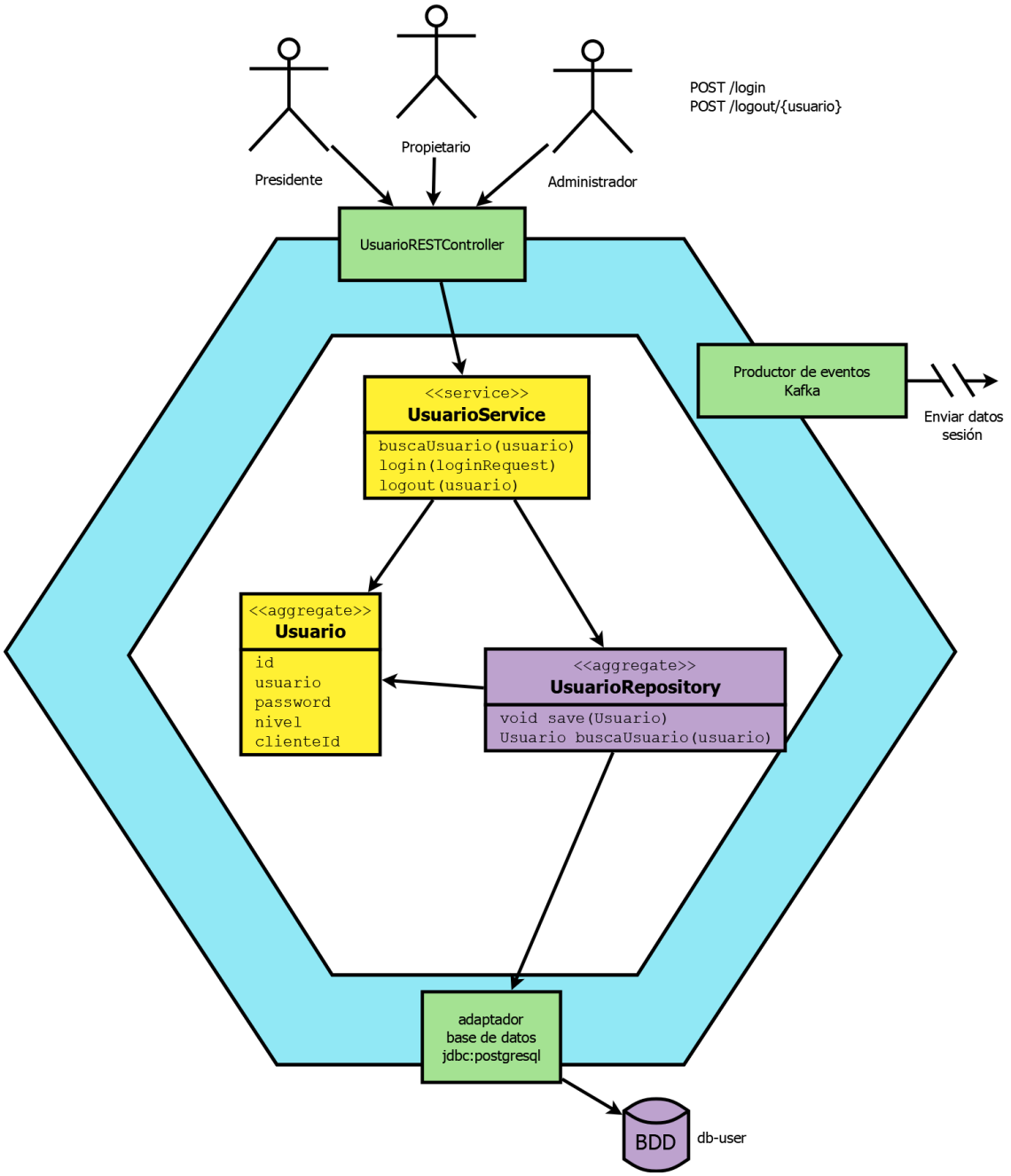


Fig 36 Arquitectura hexagonal microservicio adminh-user

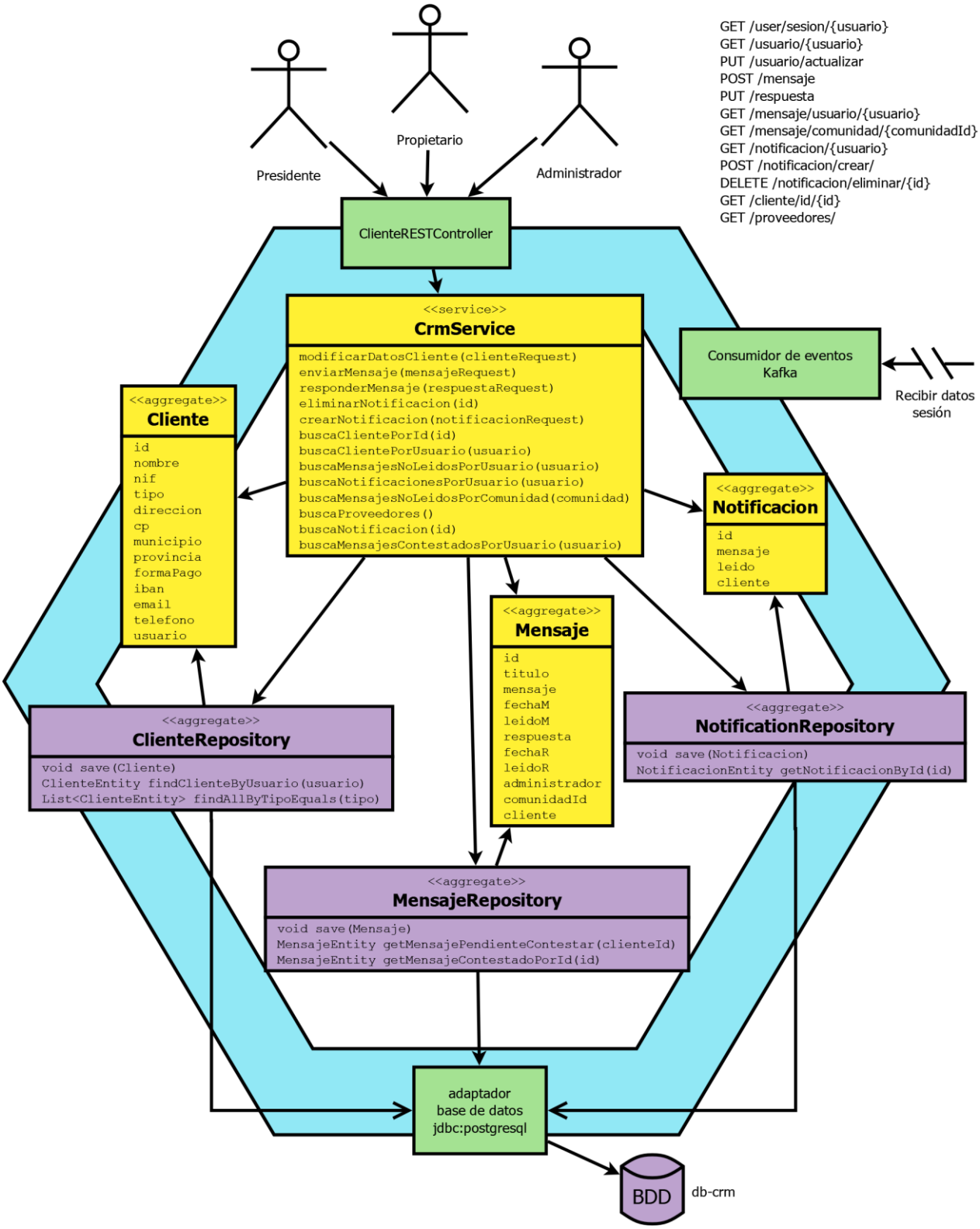


Fig 37 Arquitectura hexagonal microservicio adminh-crm

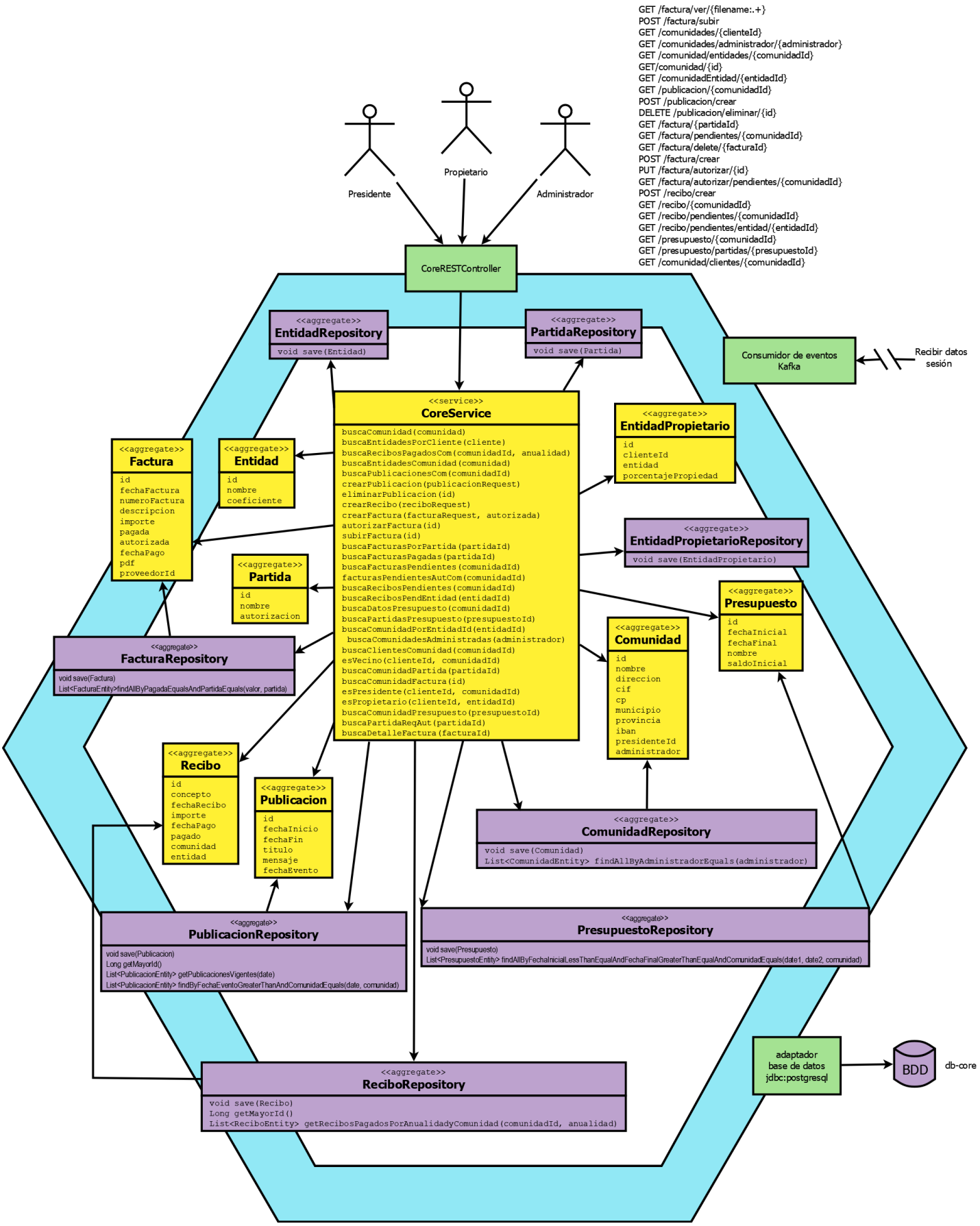
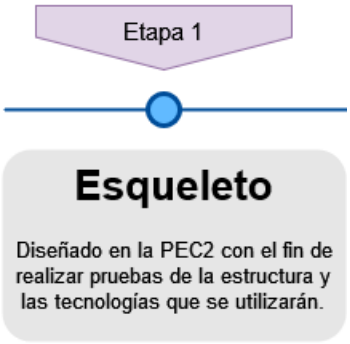


Fig 38 Arquitectura microservicio adminh-core

2.10 Etapa 1: Esqueleto de la aplicación



Como punto de partida se ha diseñado un esqueleto de la aplicación para realizar una configuración inicial del entorno y tener una primera toma de contacto con las herramientas que se utilizarán durante el posterior desarrollo, y que se han comentado en los puntos anteriores.

Concretamente, se ha creado una versión inicial de los microservicios adminh-user y adminh-crm y una pantalla de *login.* Esta pantalla permite conectar con el microservicio de usuarios para que éste, mediante Kafka, pueda transmitir un flujo de datos con el usuario y el código de sesión creado por php para que sea consumido por el microservicio adminh-crm, el cual también almacenará dicha información.

A tal efecto se han creado dos servicios REST de prueba en ambos microservicios en la `ruta /user/sesion/{usuario}`, de tal forma que se pueda verificar mediante la aplicación Postman que el código de sesión ha sido almacenado correctamente.

Asimismo, la web de *login* ha sido creada utilizando php, html y la técnica AJAX, con el objetivo de que esta pantalla trabaje de forma asíncrona. Esto nos permitirá que se informe si los datos de usuario son incorrectos sin necesidad de refrescar la página.

Para lanzar la aplicación se ha creado un fichero YAML (`docker-compose.yml`) en el que se han definido ambos microservicios, sus bases de datos, un broker Kafka ligado a un servicio zookeeper y un servicio Adminer para poder consultar las bases de datos vía web.

Por su parte, la aplicación web ha sido lanzada utilizando XAMPP.

Ejecución del fichero multicontenedor con la orden:

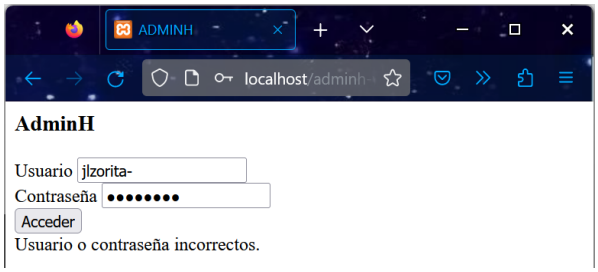
```
docker-compose up --build -d
```

```
[+] Running 7/7
 - Container tfg-db-user-1      Healthy
 - Container tfg-zookeeper-1    Started
 - Container tfg-adminer-1      Started
 - Container tfg-db-crm-1       Started
 - Container tfg-kafka-1        Healthy
 - Container tfg-adminh-user-1  Started
 - Container tfg-adminh-crm-1   Started
PS C:\TFG>
```

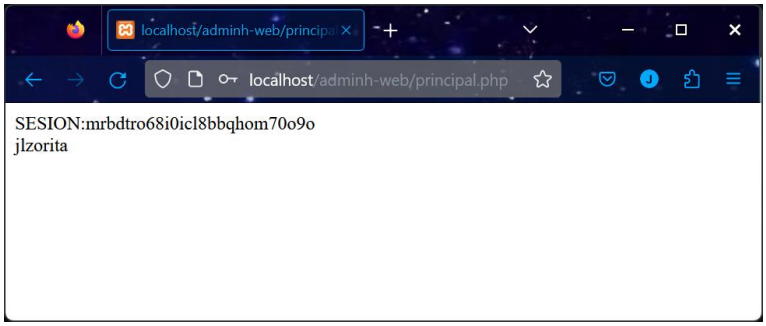
Se han creado tres usuarios con tres niveles diferentes para inicializar la base de datos una vez se lanzan los microservicios:

```
INSERT INTO usuario (usuario, password, nivel)
VALUES ('jlzorita', 'pass', 0)
       ,('jlzoritaP', 'pass', 1)
       ,('jlzoritaA', 'pass', 2);
```

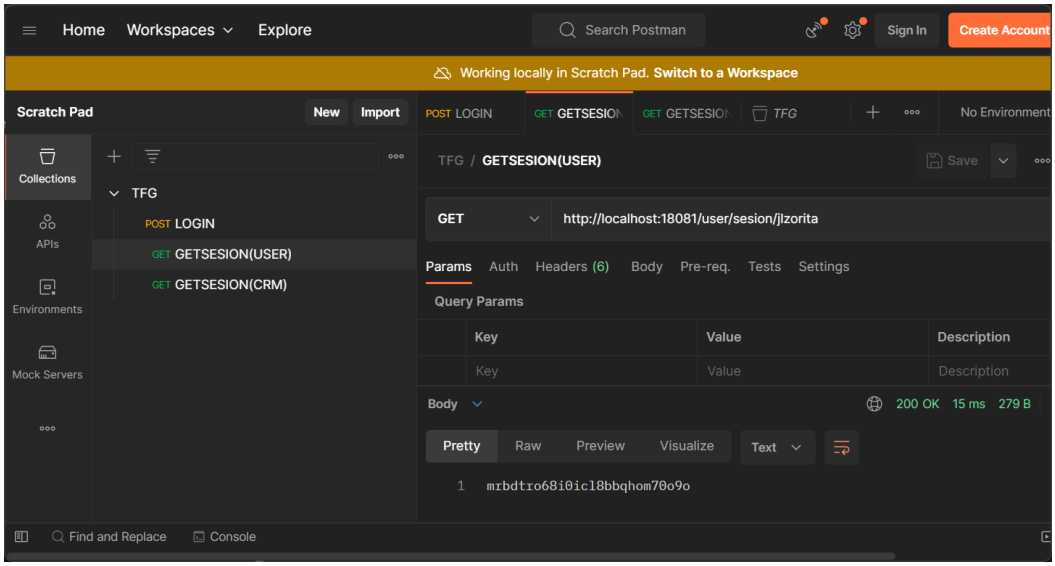
Si introducimos datos de usuario incorrectos, al clicar en el botón acceder se informa de ello:

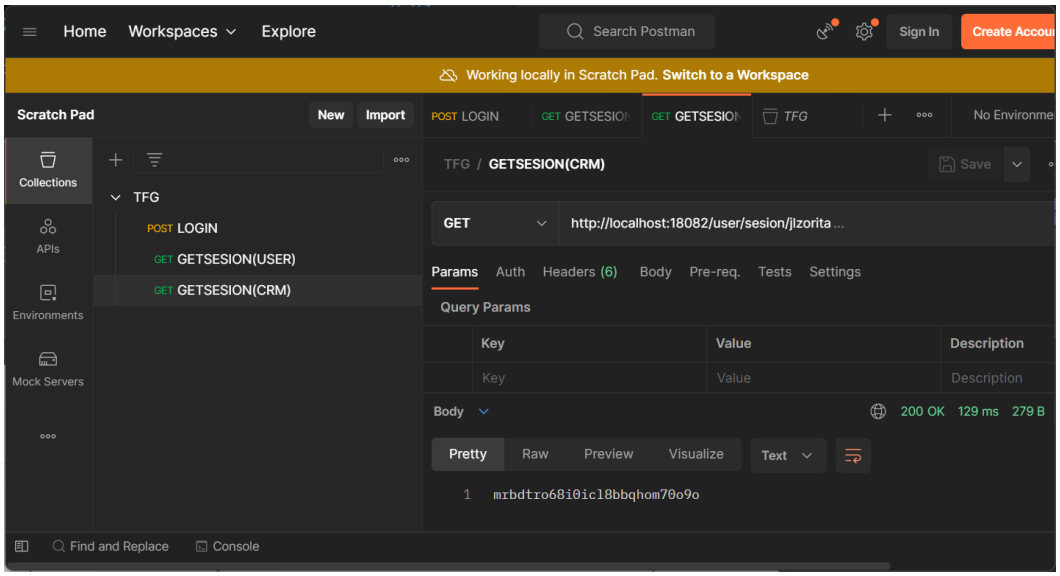


Al entrar a la página principal se informa del código de sesión creado por php y de nuestro nombre de usuario:



Podemos comprobar mediante los métodos GET `/user/sesion/{usuario}`, que el código de sesión se introduce correctamente en microservicio adminh-user y que es consumido mediante Kafka por el microservicio adminh-crm:





De este modo, una vez hemos elegido las herramientas a utilizar y hemos verificado que la plataforma funciona, ya tenemos todo preparado para iniciar la implementación del proyecto.

3. Implementación

3.1 Entorno de desarrollo

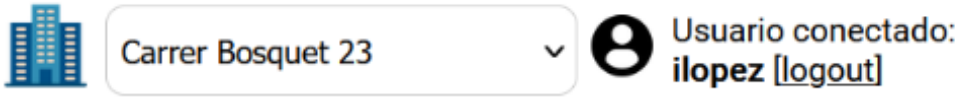
Se han utilizado dos entornos de desarrollo (IDE) para la implementación del proyecto:

Aplicación	Tipos de fichero / lenguajes	IDE
Aplicación web	- HTML - CSS - Javascript - PHP	Visual Studio Code 
Contenedores Docker	- Dockerfile - YAML	
Microservicios	- Java - Sql - XML	IntelliJ IDEA Ultimate 

3.2 Cambios en el alcance del proyecto

Se han incorporado las siguientes funcionalidades respecto a las listadas en la PEC2:

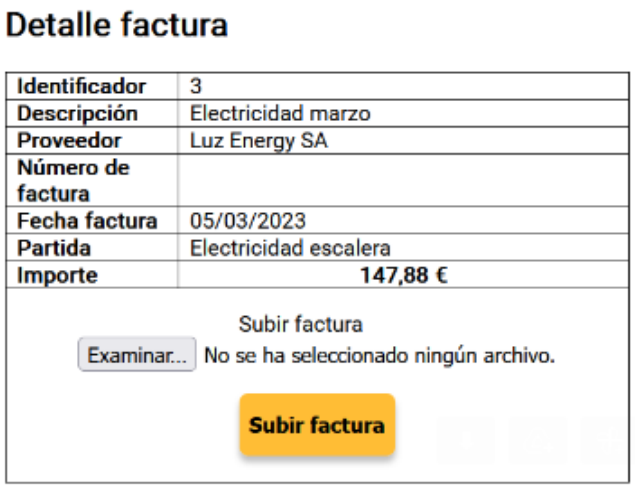
- El administrador puede cambiar la comunidad que administra desde la parte superior de la página:



- Se puede consultar el detalle de las facturas desde el estado contable al hacer clic en la descripción:

Gastos Presupuesto ordinario		
Electricidad escalera		
Fecha pago	**Descripción**	**Importe**
05/03/2023	Electricidad marzo	147,88 €
05/04/2023	Electricidad abril	163,11 €

En el detalle el administrador puede subir el documento pdf con la factura:



3.3 Implementación: cambios respecto a lo planificado inicialmente

La arquitectura y las tecnologías utilizadas para la implementación del proyecto son las mismas que se han definido en los puntos 2.4 y 2.5, con la incorporación de la librería **cURL** en PHP para la transferencia de datos mediante los métodos POST del API REST.

Los métodos de los servicios API REST han sufrido cambios para poder cumplir con los requisitos que se indicaron en los casos de uso, y también para dar respuesta a las nuevas funcionalidades introducidas.

- Algunas operaciones se han eliminado por no ser finalmente necesarias y se han añadido otras.
- Lo mismo ocurre con las llamadas al API REST. En este caso, también se han modificado datos como la repuesta obtenida (en el caso de

devolución de conjuntos vacíos la respuesta obtenida es el código 404 NOT FOUND) o los parámetros de entrada o salida.

Se ha actualizado la información en el punto 2 del documento respecto a la PEC2 para dar cabida a los cambios realizados.

3.4 Etapa 2: Creación infraestructura

Tras el montaje del esqueleto, el siguiente paso es acabar de configurar el resto de la infraestructura necesaria para desplegar todos los servicios que requiere el proyecto.

Para ello, se ha inicializado el microservicio adminh-core, y se ha terminado de editar el fichero `docker-compose.yml` para dar cabida a todos los microservicios y los servicios auxiliares que estos requieren (adminer, Kafka y zookeeper).

La inicialización del microservicio adminh-core también ha supuesto la configuración de su consumidor de mensajes de Kafka, con el fin de que pueda recibir los datos de sesión desde el microservicio adminh-user, al igual que sucede con el microservicio adminh-crm.

El esquema de los contenedores configurados en el fichero `docker-compose.yml` es el siguiente:

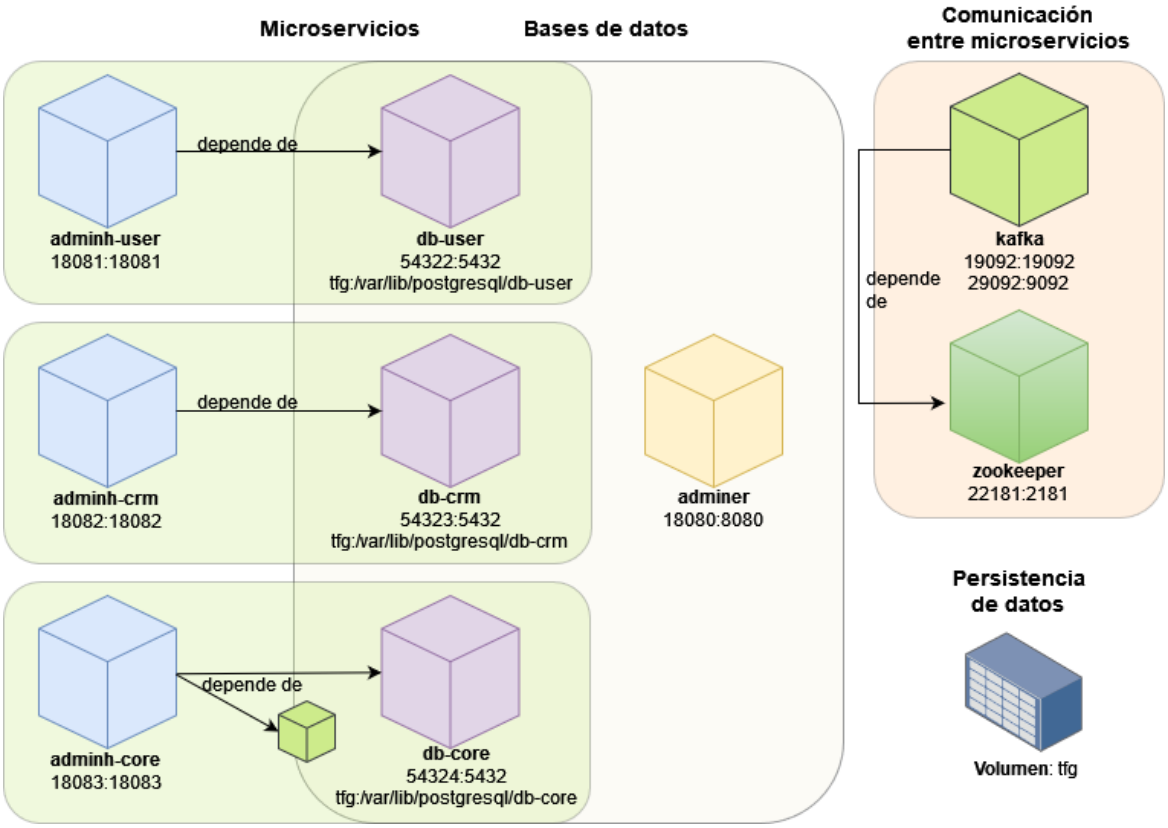


Fig 39 Diagrama que muestra los nueve servicios desplegados con docker-compose

Una vez desplegada la infraestructura con el comando:

```
docker-compose up --build -d
```

Desde *Docker Desktop* se muestra el correcto despliegue de los nueve servicios:

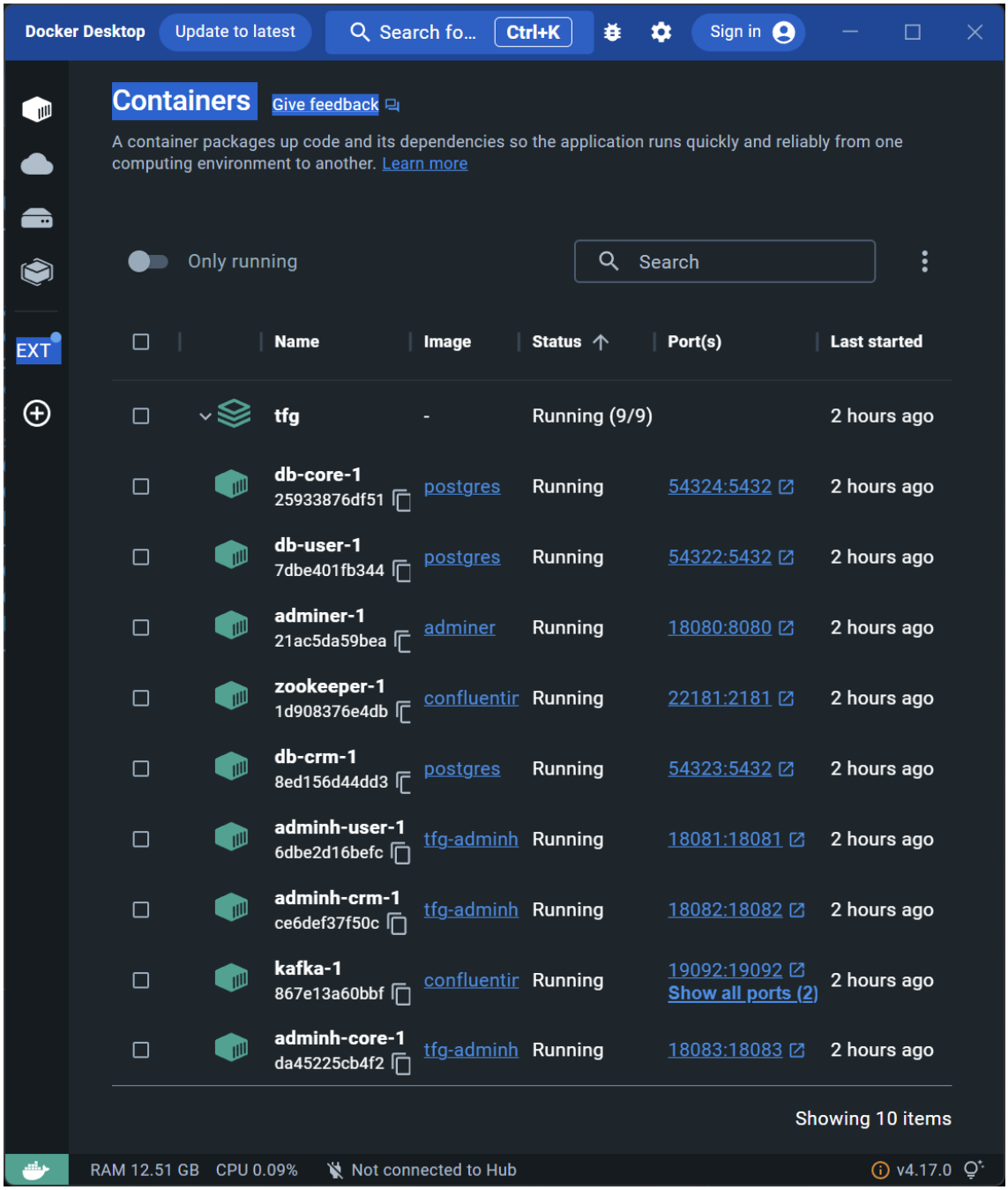


Fig 40 Captura Docker Desktop con despliegue servicios

Asimismo, dentro de esta etapa también se ha efectuado la creación de las entidades dentro de cada microservicio.

Para ello se han seguido los siguientes pasos:

Dentro del directorio `src\main\java\edu\uoc\tfg\core\domain` se han creado las clases que representan todos los objetos que vamos a utilizar en la aplicación. Por ejemplo, el contenido de la clase `Partida.java` del microservicio adminh-crm es el siguiente:

```java
@Getter
@Setter
@Builder
@EqualsAndHashCode
@NoArgsConstructor
@AllArgsConstructor
public class Partida {

    private Long id;
    private String nombre;
    private Boolean autorizacion;

    @Builder.Default
    @JsonIgnore
    private Set<Factura> facturas = new HashSet<>();

    @NotNull
    @JsonIgnore
    private Presupuesto presupuesto;
}
```

La clase tiene los atributos `id`, `nombre`, `autorizacion` y `presupuesto`, así como el conjunto de facturas `Set<Factura>` que están dentro de la partida.

Los atributos `presupuesto` y `Set<Factura>` tienen la anotación `@JsonIgnore`. De esta forma, cuando se devuelve el conjunto de resultados que contenga este tipo de datos con un método GET, no se visualizarán:

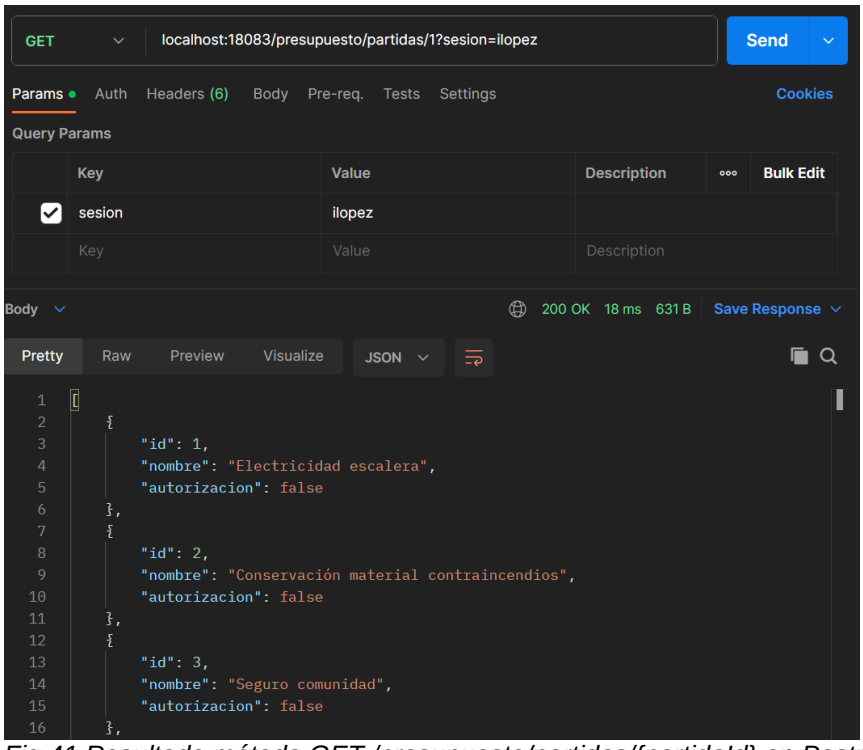


Fig 41 Resultado método GET /presupuesto/partidas/{partidaId} en Postman

Asimismo, se ha realizado la definición de las entidades dentro del directorio `src\main\java\edu\uoc\tfg\core\infrastructure\repository\jpa`. De forma análoga al ejemplo anterior, se muestra el contenido del fichero `PartidaEntity.java`:

```
@Entity
@Getter
@Setter
@AllArgsConstructor
@EqualsAndHashCode(exclude={"facturas", "presupuesto"})
@Builder
@NoArgsConstructor
@Table(name = "partida")
public class PartidaEntity implements DomainTranslatable<Partida> {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nombre", nullable = false)
    private String nombre;

    @Column(name = "autorizacion", nullable = false)
    private Boolean autorizacion;

    @OneToMany(mappedBy="id")
    private Set<FacturaEntity> facturas = new HashSet<>();

    @ManyToOne
    @JoinColumn(name="presupuesto_id", referencedColumnName = "id")
    private PresupuestoEntity presupuesto;

    public static PartidaEntity fromDomain(Partida partida) {
        if (partida == null) {
            return null;
        }

        return PartidaEntity.builder()
                .id(partida.getId())
                .nombre(partida.getNombre())
                .autorizacion(partida.getAutorizacion())
                .facturas(partida.getFacturas().stream().map(FacturaEn
tity::fromDomain).collect(Collectors.toSet()))
                .presupuesto(PresupuestoEntity.fromDomain(partida.getP
resupuesto()))
                .build();
    }
    @Override
    public Partida toDomain() {
        return Partida.builder()
                .id(this.getId())
                .nombre(this.getNombre())
                .autorizacion(this.getAutorizacion())
//.facturas((this.getFacturas().stream().map(FacturaEntity::toDomain).
collect(Collectors.toSet())))
                //.presupuesto(this.getPresupuesto().toDomain())
                .build();
    }
}
```

Este tipo de clases están declaradas con la anotación `@Entity` para indicar que se tratan de objetos a almacenar en la base de datos.

Cada variable de la clase tiene, o bien la anotación `@Column` para indicar que se trata de un atributo a incluir en la tabla de la base de datos, o bien la anotación `@ManyToOne / @OneToMany` para indicar el tipo de relación que tiene con otra entidad / tabla de la base de datos.

En el primer caso, con las opciones `name` y `nullable` indicamos el nombre que tendrá el campo en la base de datos y si el campo se puede dejar vacío (valor `null`) respectivamente.

En el caso de una relación *many to one*, se utiliza la anotación `@JoinColumn` para indicar el nombre del campo con el identificador de la tabla que vamos a unir (opción `referencedColumnName)` y el nombre que le vamos a dar en la presente tabla (opción `name`).

La excepción a la hora de tratar los campos es el que tiene el identificador de la tabla, que tiene las anotaciones `@Id` para indicar que es la clave y `@GeneratedValue` para que el valor se genere automáticamente con autoincremento.

Finalmente tenemos los métodos `fromDomain` y `toDomain`. Gracias al uso de la anotación de lombok `@Builder` que se había introducido en la clase del objeto, estas clases nos permiten automatizar la creación y obtención de objetos de la entidad mediante el método `builder()`.

Como puede comprobarse en el ejemplo indicado arriba de la clase `PartidaEntity`, en el método `toDomain` están comentadas las líneas referentes a la obtención de los datos de las facturas y el presupuesto. Esto se debe a que si no se comentaba la aplicación devolvía el siguiente error:

```
2023-06-17 19:39:44.631 ERROR 28520 --- [io-18082-exec-6]
o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servlet.service() for
servlet [dispatcherServlet] in context with path [] threw exception
[Handler dispatch failed; nested exception is
java.lang.StackOverflowError] with root cause

java.lang.StackOverflowError: null at
java.base/java.util.Spliterators$IteratorSpliterator.estimateSize(Spli
terators.java:1941) ~[na:na] at
java.base/java.util.Spliterator.getExactSizeIfKnown(Spliterator.java:4
14) ~[na:na]      at
java.base/java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.
java:508) ~[na:na] at
java.base/java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPi
peline.java:499) ~[na:na] at
java.base/java.util.stream.ReduceOps$ReduceOp.evaluateSequential(Reduc
eOps.java:921) ~[na:na] at
java.base/java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.
java:234) ~[na:na] at
java.base/java.util.stream.ReferencePipeline.collect(ReferencePipeline
.java:682) ~[na:na] at
```

```
edu.uoc.tfg.core.infrastructure.repository.jpa.ComunidadEntity.toDomai
n(ComunidadEntity.java:93) ~[classes/:na] at
edu.uoc.tfg.core.infrastructure.repository.jpa.ReciboEntity.toDomain(R
eciboEntity.java:70) ~[classes/:na]
```

Tras divagar por la red para hallar solución a este problema, este parece producirse [1] debido a un bucle infinito que se da al obtener los datos de una entidad. Hay que tener en cuenta que, por ejemplo, la clase `Partida` devuelve los datos de la clase `Presupuesto` al que pertenece, mientras que éste último también devuelve la lista de partidas. Por lo tanto, si no se limita la información que se devuelve, se produce un bucle que provoca el error.

Finalmente, el último paso realizado en esta etapa ha sido el de dotar al archivo `src\main\resources\schema.sql` de información para que cuando se inicien los microservicios estos ya cuenten con datos con los que trabajar.

Ejemplo de inicialización de los datos de las tablas `Comunidad` y `Presupuesto`:

```
INSERT INTO comunidad(nombre, direccion, cif, cp ,municipio, provincia,
presidente_id, iban, administrador)
VALUES ('Carrer Bosquet 23', 'Carrer Bosquet, 23', 'B71231823',
'08100','Mollet del Vallès','Barcelona',9,'ES26 5434 6552 4123 1236
7655', 'ilopez')
,('Avinguda Mar 112', 'Avinguda Mar 112', 'B16732345', '08100','Mollet
del Vallès','Barcelona',1,'ES34 3422 5811 0031 8728 1852', 'ilopez');

INSERT INTO presupuesto(fecha_inicial, fecha_final, nombre,
saldo_inicial, comunidad_id)
VALUES  ('2023/01/01','2023/12/31','Presupuesto ordinario','4375.23',1),
       ('2023/01/01','2023/12/31','Presupuesto ordinario','1225.80',2);
```

Una vez finalizada esta etapa, ya podemos consultar todas las tablas que se han creado y los datos almacenados mediante Adminer:

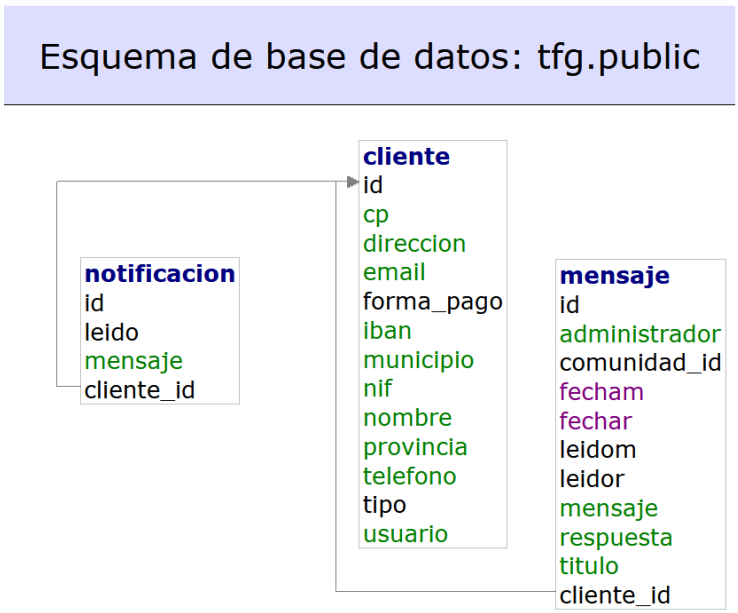


Fig 42 Esquema de la base de datos del microservicio adminh-crm visto desde Adminer

3.5 Etapa 3: API REST y WEB propietarios

Esta etapa está centrada en la creación de los métodos del API REST, sus operaciones asociadas en cada microservicio y las pantallas del *frontend* que permitan realizar las acciones necesarias a los usuarios con rango propietario o presidente.

En cada microservicio se han creado los métodos del API REST dentro de los ficheros `UsuarioRESTController`, `ClienteRESTController` y `CoreRESTController`, así como las operaciones utilizadas en los servicios `UsuarioService`, `CrmService` y `CoreService`, tal y como se define en los diagramas de la arquitectura hexagonal del punto 2.9.

Ejemplo de métodos REST del microservicio adminh-user:

```
@Log4j2
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
@CrossOrigin
@RestController
public class UsuarioRESTController {

    private final UsuarioService usuarioService;

    @PostMapping("/login")
    @ResponseStatus(HttpStatus.OK)
    public ResponseEntity<Integer> login(@RequestBody LoginRequest
loginRequest){

        log.trace("login usuario");
        int level =  usuarioService.login(loginRequest);

        if(level >= 0) return ResponseEntity.ok().body(level);
        return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
    }

    @PostMapping("/logout/{usuario}")
    @ResponseStatus(HttpStatus.OK)
    public ResponseEntity logout(@PathVariable String usuario){

        if(usuarioService.logout(usuario)) return new
ResponseEntity<>(HttpStatus.OK);
        else return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
    }
```

La clase posee la anotación de *Spring* `@RestController` para indicar que se trata de un Servicio REST.

Asimismo, también se ha introducido la anotación de lombok `@CrossOrigin` para permitir el acceso a los recursos desde diferentes orígenes (CORS).

Respecto a los métodos, estos están definidos con anotaciones de Spring dependiendo del tipo que sea, como `@GetMapping("url")` para un método del tipo GET o `@PostMapping("url")`, para un método del tipo POST. En este tipo de anotación, `"url"` hace referencia a la dirección para acceder al recurso.

Cada método tiene también la anotación de Spring `@ResponseStatus` para establecer por defecto la respuesta `200 OK`.

Respecto a los parámetros de entrada, en algunos casos se ha creado un objeto del tipo *Request* para establecer unos campos concretos dentro del cuerpo del mensaje dependiendo del tipo de objeto que queramos tratar. Este tipo de objetos están localizados dentro del directorio `src\main\java\edu\uoc\tfg\user\application\request`.

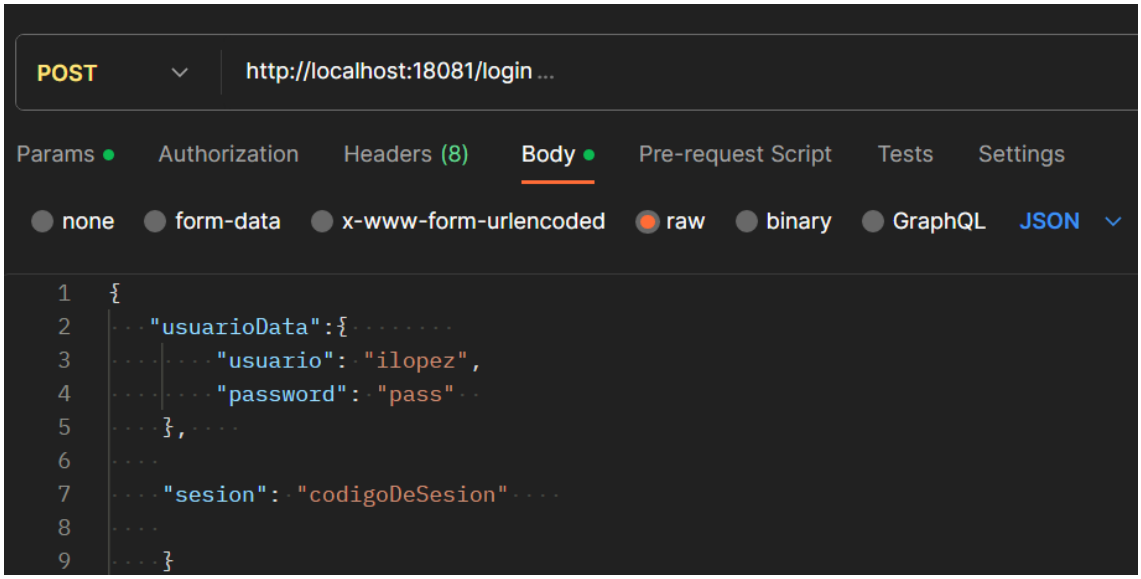
En el caso del método para realizar el *login* se trabaja con el objeto `LoginRequest`, cuyo contenido es el siguiente. Estas clases incluyen la anotación `@JsonCreator` para la deserialización de los datos que llegan del JSON y poder asignarlos a los argumentos del constructor:

```
@Getter
@NotNull
private final Usuario usuario;
@Getter
@NotNull
private final String[] sesion = new String[2];

@JsonCreator
public LoginRequest(@JsonProperty("usuarioData") @NotNull final
Usuario usuario, String sesion) {
    this.usuario = usuario;
    this.sesion[0] = sesion;

    // Crear / reiniciar sesión
    Sesion.removeUsuario(usuario.getUsuario());
    Sesion.addUsuario(usuario.getUsuario(), this.sesion);
}
```

La petición JSON tiene el nombre `usuarioData`, tal y como se ha definido con la anotación `@JsonProperty`.

Los datos que se espera recibir en el cuerpo del mensaje son los de la clase `Usuario` (`usuario` y `password`) y también un código de sesión. Asimismo, el propio constructor ya realiza las llamadas a los métodos que introducen la sesión de usuario dentro del sistema.

Desde el punto de vista de la ejecución del *login*, el contenido del cuerpo del JSON sería el siguiente:



En el directorio `src\main\java\edu\uoc\tfg\user\domain\service` se encuentran las operaciones de cada servicio.

Siguiendo con el ejemplo del *login*, en el fichero `UsuarioServiceImpl` tenemos el método *login*, que realiza la tarea de gestionar los datos de sesión recibidos, comprobar que son correctos y, en caso afirmativo, enviarlo mediante el productor de Kafka al resto de microsevicios.

Finalmente y en lo que respecta a la edición de los microservicios, en la ruta `src\main\java\edu\uoc\tfg\user\infrastructure\repository\jpa` de cada uno de ellos se encuentran los ficheros para la edición del repositorio de cada entidad.

Dentro de los ficheros con denominación `SpringData{Clase}Repository` se encuentran un conjunto de operaciones a realizar sobre la base de datos.

Estas operaciones pueden ser realizadas de dos formas.

- Mediante una consulta sql:

```
@Query("from UsuarioEntity u where u.usuario = ?1")
Optional<UsuarioEntity> findUsuario(String usuario);
```

- Mediante los métodos ofrecidos por el API de Spring Data JPA, con el uso de palabras clave como `findAllBy`, `Equals` o `And`:

```
List<FacturaEntity> findAllByPagadaEqualsAndPartidaEquals(Boolean
valor, PartidaEntity partida);
```

Dentro de los ficheros con la denominación `{Clase}RepositoryImpl` se encuentra la implementación de la lógica para la realización de las diferentes operaciones. Dentro de estas clases se realiza la manipulación de las

entidades con su tratamiento mediante objetos gracias a las posibilidades que ofrece Spring Boot.

En este ejemplo que muestra el método `responderMensaje`, vemos como a partir del identificador de un mensaje éste se recupera de la base de datos, modificamos los campos necesarios y volemos a guardarlo mediante el método `save()` de JPA:

```
@Override
public Boolean responderMensaje(RespuestaRequest respuestaRequest) {
    Long mensajeId = respuestaRequest.getMensajeId();
    String respuesta = respuestaRequest.getRespuesta();

    if(jpaRepository.getMensajeContestadoPorId(mensajeId).isPresent())
        return false;
    else{
        MensajeEntity me = jpaRepository.findById(mensajeId).get();
        me.setRespuesta(respuesta);
        me.setLeidom(true);
        Date date = new Date();
        me.setFechar(date);
        jpaRepository.save(me);
        return true;
    }
}
```

En esta etapa también se ha completado la parte de *frontend* relativa a la página web. Los ficheros creados hasta el momento dentro del proyecto adminh-web han sido los que se muestran en la siguiente captura:

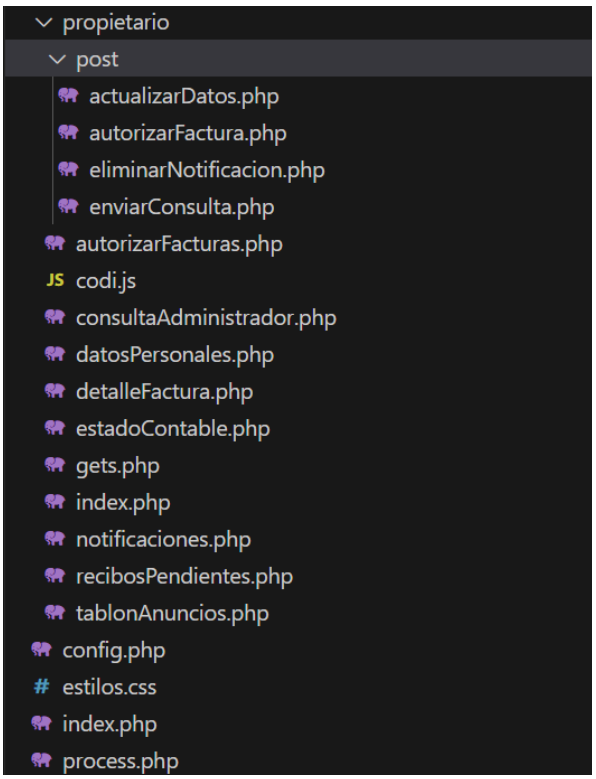


Fig 43 Ficheros web propietario

Dentro del directorio `propietario` se encuentran todos los ficheros utilizados para las pantallas del propietario y del presidente.

En el fichero `gets.php` se encuentran todas las funciones que realizan llamadas del tipo GET al API REST de los microservicios, devolviendo el resultado de este. Por ejemplo, las dos siguientes funciones devuelven el número de notificaciones de un usuario y sus datos:

```
function numNotificaciones(){
   $sessionId = session_id();
   $url =
"http://$GLOBALS[host]:$GLOBALS[puertoCrm]/notificacion/$_SESSION[usuario
]?sesion=$sessionId";
   $data = json_decode(file_get_contents($url), true);
   if($data != null) return sizeof($data);
   else return 0;
}

 function datosUsuario(){
   $sessionId = session_id();
   $url =
"http://$GLOBALS[host]:$GLOBALS[puertoCrm]/usuario/$_SESSION[usuario]?ses
ion=$sessionId";
   return json_decode(file_get_contents($url), true);
 }
```

Como se puede comprobar en el código, se ha utilizado la sesión (`$_SESSION`) de php para almacenar variables como el nombre de usuario, el identificador de la entidad o el de la comunidad, de tal forma que no sea necesario tener que pasar esta información como parámetros en las funciones.

En el fichero `codi.js` se encuentran funciones codificadas en javascript. Principalmente podemos distinguir dos tipos de funciones:

- Funciones que utilizan AJAX para actualizar la información sin refrescar la página. El siguiente código, por ejemplo, actualiza el contenido del `<div>` con identificador `#contenido` con el contenido del fichero `estadoContable.php`:

```
function estadoContable(){
    $.ajax({
    url: 'estadoContable.php',
    success: function(result) {
        $('#contenido').html(result);
    }
});
}
```

- Funciones que utilizan AJAX para tratar los datos de un formulario de forma asíncrona (ejecución de métodos POST del API REST) y poder mostrar una respuesta sin refrescar la página.

  El siguiente método `enviarConsulta()` recoge los datos del formulario con identificador `form1` y los envía serializados a la página `post/enviarConsulta.php`. En el caso de que la ejecución sea satisfactoria [`success: function(respuesta)`] se mostrará un mensaje según la respuesta que se haya devuelto:

```
function enviarConsulta(){
    $.ajax({
        type: 'POST',
        url: 'post/enviarConsulta.php',
        data: $('#form1').serialize(),

        success: function(respuesta) {
            document.getElementById("retorno").style.color = "red";
            document.getElementById("retorno").style.fontWeight = "bold";

            if(respuesta == "500"){
                document.getElementById("retorno").innerHTML = "<b>No puedes
enviar mensajes si hay pendientes de contestar.</b>";
            }else if(respuesta == "noTitulo"){
                document.getElementById("retorno").innerHTML = "<b>Debes
introducir un título.</b>";
            }else if(respuesta == "noMensaje"){
                document.getElementById("retorno").innerHTML = "<b>Debes
introducir un mensaje.</b>";
            }else{
                document.getElementById("retorno").style.color = "green";
                document.getElementById("retorno").innerHTML = "<b>Mensaje
enviado</b>";
                document.getElementById("pendiente").innerHTML =
respuesta;
            }
        }
        })
```

Dentro del directorio `post` están los ficheros php encargados de realizar las operaciones POST del API REST. Si tomamos como ejemplo el fichero `post/autorizarFactura.php`, la operativa seguida para la ejecución de los métodos POST es la siguiente:

```
$id = $_POST["id"];
//Realizar POST
$url =
"http://$GLOBALS[host]:$GLOBALS[puertoCore]/factura/autorizar/$id?s
esion=$sesionId";
```

```php
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_URL, $url);
curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PUT");
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

$headers = array(
    "Accept: application/json",
    "Content-Type: application/json",
);
curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
$resp = curl_exec($curl);

$httpCode = curl_getinfo($curl, CURLINFO_HTTP_CODE);
curl_close($curl);

if($httpCode == "200"){
    echo $httpCode;
}
?>
```

En primer lugar, se deserializan los datos remitidos desde el formulario para obtener el valor de cada variable con `$_POST`. Posteriormente, se utiliza la librería curl para la configuración y la ejecución del método POST indicado en la variable `$url`, así como para la obtención del código de respuesta.

Para finalizar este apartado, se debe resaltar lo sencillo que ha sido trasladar el diseño del prototipo que se había realizado en Figma gracias a la posibilidad de poder exportar el código css de los elementos:

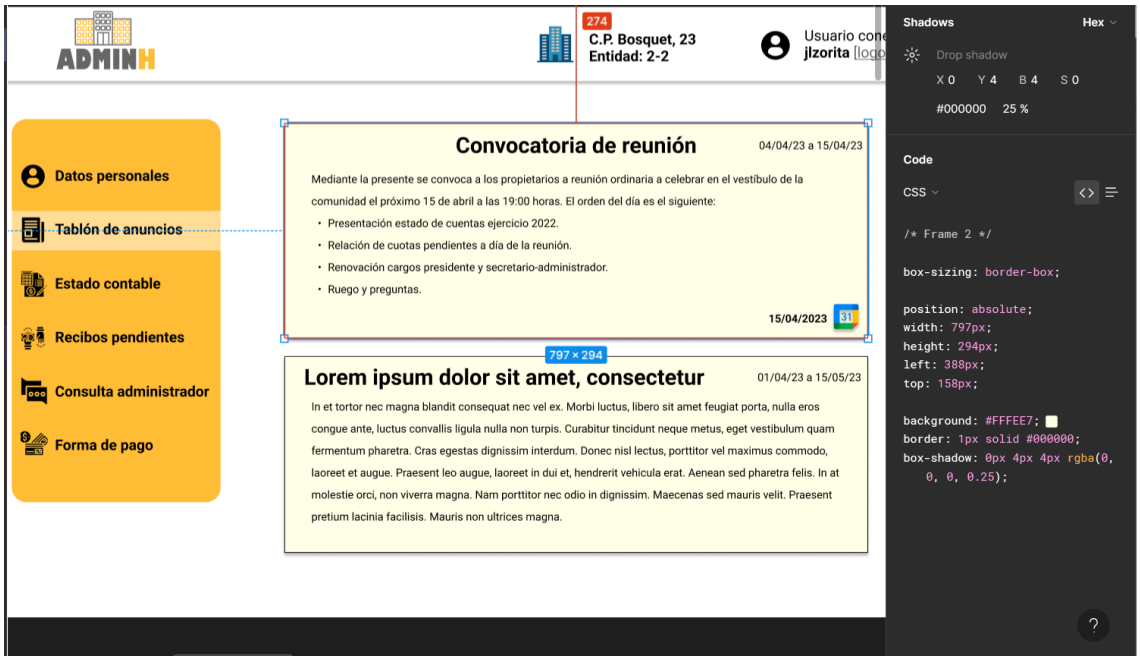


Fig 44 Al seleccionar una publicación del tablón en Figma vemos su código CSS en la parte derecha

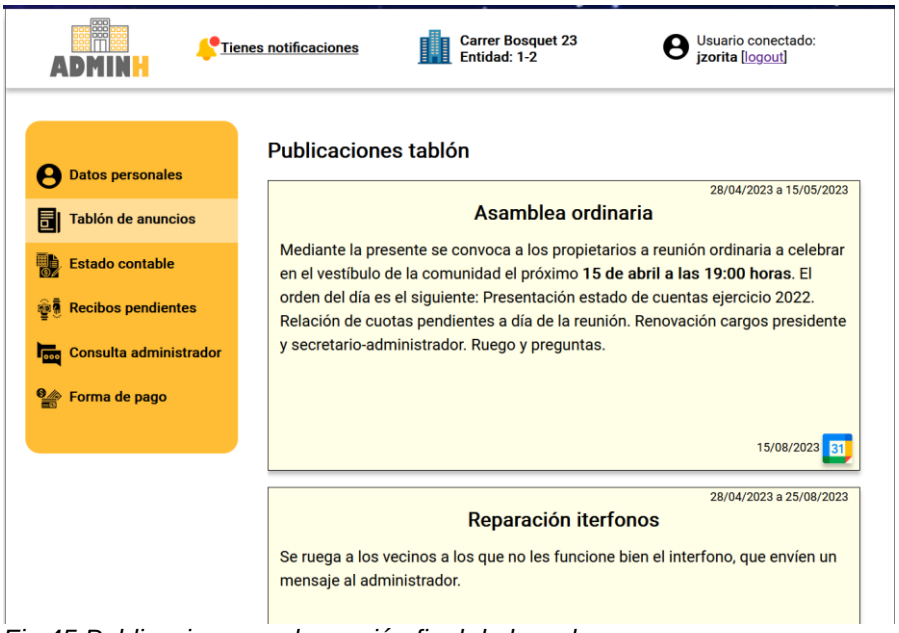


Fig 45 Publicaciones en la versión final de la web

3.6 Etapa 4: API REST y WEB administrador

El proceso seguido para crear las características necesarias para la página del administrador y las pantallas web ha sido similar a lo visto en el punto anterior.

En la página web se han desarrollado los métodos en el directorio `administrador` siguiendo la misma estructura que lo visto para el propietario.

La estructura de ficheros de la parte administrador es la siguiente:

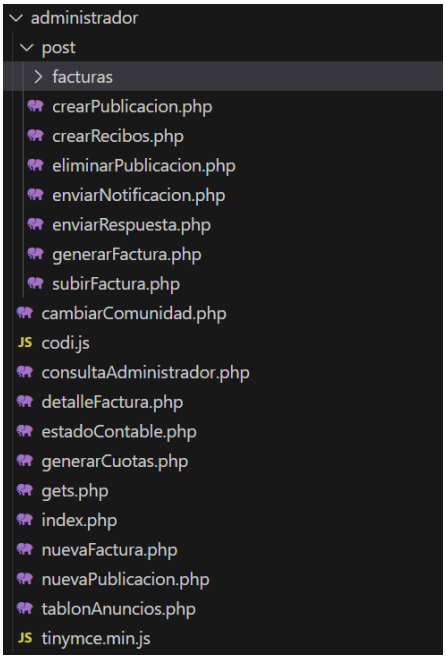


Fig 46 Ficheros web administrador

En este caso se ha incorporado el fichero `tinymce.min.js`. Este fichero es la API del editor de texto enriquecido TinyMCE [2], que tiene licencia LGPL [3] y que se ha utilizado para la creación de publicaciones en el tablón:

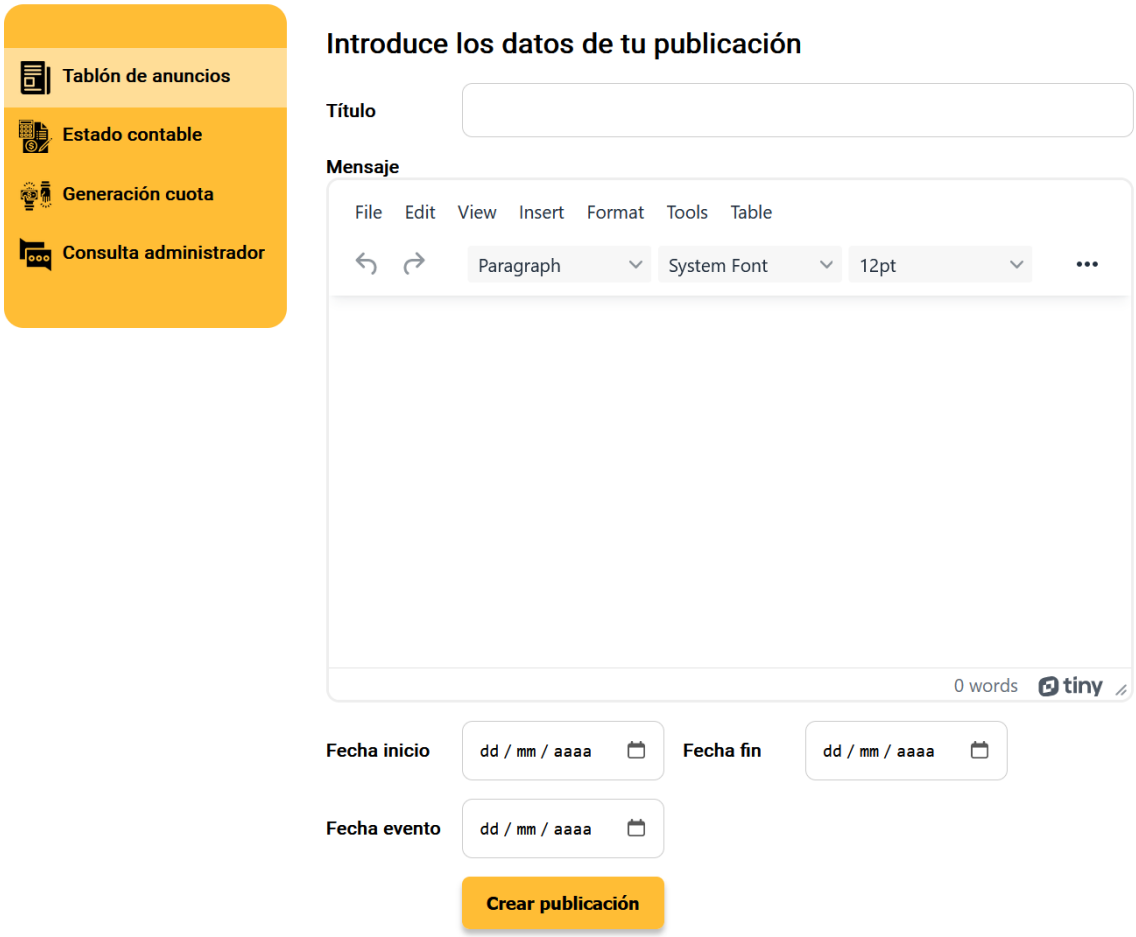


Fig 47 Publicación en el tablón con TinyMCE

Una nueva funcionalidad que se ha añadido a la aplicación es la posibilidad de que el administrador suba facturas. Se ha tomado como modelo para su implementación la guía de la página spring.io [4]. El directorio `post/facturas` tiene la utilidad de almacenar, de forma temporal, las facturas que sube el administrador.

Un error que me encontré en la web a la hora de desplegar el proyecto, es que se devolvía un error a la hora de subir ficheros. Hay que tener en cuenta que si se realiza el despliegue en un sistema con una distribución Linux se debe otorgar permisos de escritura al mencionado directorio.

El portal del administrador también permite cambiar de comunidad administrada. En la parte superior existe un *select* en el que aparecen todas sus comunidades:

```
<select onChange="cambiaComunidad();" name="comunidad"
style="width:200px;">
```

Al seleccionar otra comunidad lo único que se realiza es cambiar la variable almacenada en la sesión:

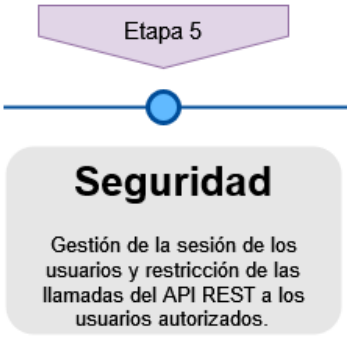
```php
<?php
require("../config.php");
include("gets.php");
$_SESSION["comunidadId"] = $_POST["comunidad"];
?>
```

Un error con el que me he encontrado a la hora de ejecutar el método POST para la generación de recibos, es que no se generaban correctamente porque el microservicio intentaba asignarles un identificador comenzando en 1, ignorando los recibos que se habían inicializado al arrancar el microservicio.

El problema se ha solucionado al eliminar el identificador en la instrucción INSERT INTO del `schema.sql` y dejar que éste se genere automáticamente, puesto que la entidad tiene la anotación `@GeneratedValue(strategy = GenerationType.IDENTITY)`:

```sql
INSERT INTO recibo(concepto, fecha_recibo, importe, fecha_pago, pagado, entidad_id, comunidad_id)
VALUES ('Cuota ordinaria anual', '2023/01/01', 250, '2023/01/01', true, 1, 1),
       ('Cuota ordinaria mensual', '2023/01/01', 45, '2023/01/01', true, 2, 1),
       ('Cuota ordinaria mensual', '2023/02/01', 45, '2023/02/01', true, 2, 1),
       ('Cuota ordinaria mensual', '2023/03/01', 45, '2023/03/01', true, 2, 1),
       ('Cuota ordinaria mensual', '2023/04/01', 45, '2023/04/01', true, 2, 1),
```

3.7 Etapa 5: Seguridad



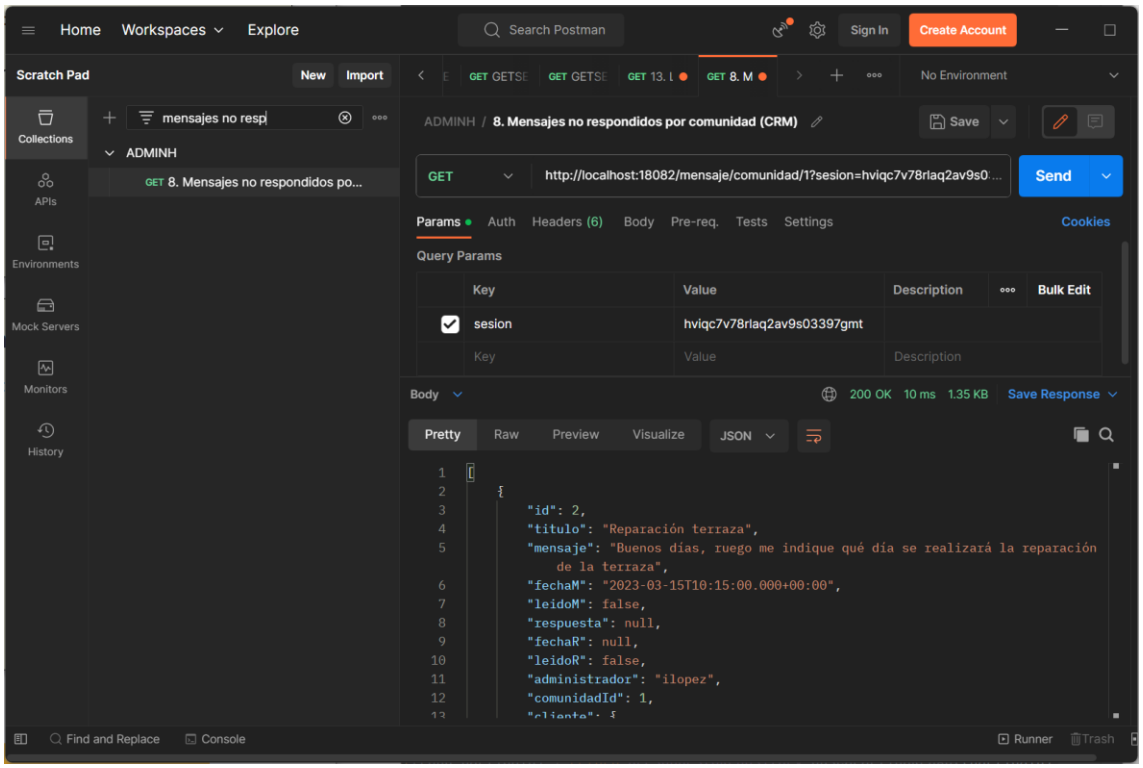
Una vez implementadas las funcionalidades para todos los usuarios, la última etapa del proyecto ha consistido en la modificación de todas las operaciones del API REST para limitar su ejecución a los usuarios que tengan los permisos necesarios.

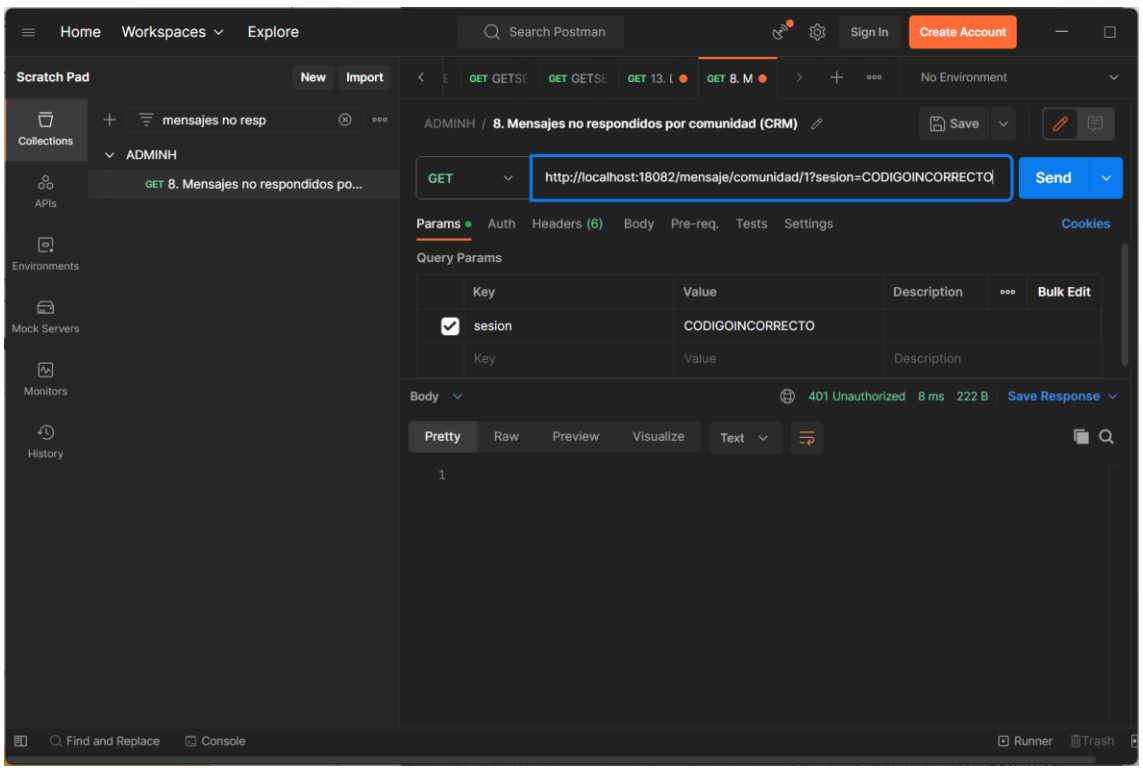
Estos permisos son los que están definidos en el punto 2.8 del documento.

Para ello, se ha añadido el parámetro de entrada `sesion` en cada método. Este parámetro es gestionado internamente para comprobar que el usuario tenga permisos necesarios, devolviendo el código de repuesta 401 UNAUTHORIZED en caso contrario.

Para realizar estas verificaciones se han tenido que añadir nuevas operaciones en los servicios. Por ejemplo, en el microservicio adminh-core se han añadido operaciones como `esVecino()`, `esPresidente()` o `esPropietario()`.

Las siguientes capturas muestran la ejecución de los métodos para mostrar la lista de mensajes enviados a un administrador en una comunidad, con un usuario autorizado primero y con un código erróneo posteriormente, mostrando los códigos 200 OK y 401 UNAUTHORIZED respectivamente:


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
Collections
APIs
Environments
Mock Servers
Monitors
History
mensajes no resp
ADMINH
GET 8. Mensajes no respondidos po...
No Environment
ADMINH / 8. Mensajes no respondidos por comunidad (CRM)
Save
GET
http://localhost:18082/mensaje/comunidad/1?sesion=hviqc7v78rlaq2av9s0...
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
hviqc7v78rlaq2av9s03397gmt
Body
200 OK
10 ms
1.35 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 2,
"titulo": "Reparación terraza",
"mensaje": "Buenos días, ruego me indique qué día se realizará la reparación de la terraza",
"fechaM": "2023-03-15T10:15:00.000+00:00",
"leidoM": false,
"respuesta": null,
"fechaR": null,
"leidoR": false,
"administrador": "ilopez",
"comunidadId": 1,
Find and Replace
Console
Runner
Trash


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
Collections
APIs
Environments
Mock Servers
Monitors
History
mensajes no resp
ADMINH
GET 8. Mensajes no respondidos po...
No Environment
ADMINH / 8. Mensajes no respondidos por comunidad (CRM)
Save
GET
http://localhost:18082/mensaje/comunidad/1?sesion=CODIGOINCORRECTO
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
CODIGOINCORRECTO
Body
401 Unauthorized
8 ms
222 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1
Find and Replace
Console
Runner
Trash

4. Conclusiones

De forma global, considero que las conclusiones que se pueden extraer del trabajo realizado son positivas, ya que se han podido completar las tareas que se habían planificado durante las primeras semanas y el diseño de la aplicación final se asemeja al prototipo realizado.

Para la etapa inicial del proyecto me ha ayudado en gran medida haber elegido una temática que me es familiar, ya que durante mi carrera profesional he trabajado durante muchos años dentro del ámbito de la administración de fincas. Esta experiencia me ha facilitado las tareas de identificar los actores que interactúan con la aplicación y de reconocer las necesidades que los usuarios van a tener en forma de historias de usuario.

Otro elemento importante para el correcto desarrollo del proyecto ha sido el montaje de un esqueleto en la PEC2. Poder realizar de forma previa a la fase de implantación la configuración de la infraestructura básica y validar su funcionamiento me ha resultado positivo para afrontar la indicada fase con garantías.

Respecto a la implantación, finalmente se han añadido funcionalidades no planificadas inicialmente con el fin de dotar de coherencia al resultado final. Tareas como poder consultar el detalle de una factura, adjuntarles un fichero o que el administrador pueda cambiar la comunidad administrada, las he considerado básicas para la consistencia del ecosistema de la plataforma, razón por la cual se han acabado implementando.

El desarrollo de las tareas para implementar las funcionalidades propias del *backend* me ha permitido profundizar en el conocimiento de la programación de microservicios con Java y Spring Boot, más allá de lo visto en las asignaturas propias del itinerario de Ingeniería del Software, así como la manipulación de bases de datos con postresql o el intercambio de mensajes entre productores y consumidores con Kafka. Durante la ejecución me he encontrado con las dificultades expuestas en la memoria, las cuales se han podido solventar adecuadamente.

Otro factor que considero importante es haber dotado al API REST de seguridad para que únicamente un tipo de usuario determinado pudiera utilizar los servicios. Lo he considerado prioritario ya que en asignaturas como Proyecto de Desarrollo del Software, donde se había realizado un trabajo en grupo con las mismas tecnologías para el *backend*, no se había tenido en cuenta esta cuestión.

Por otro lado, la investigación y el estudio realizados para encarar la parte *frontend* de la aplicación me ha permitido, no únicamente superar satisfactoriamente el reto de realizar esta parte, sino también adquirir nuevas habilidades en lo que respecto a la programación web (html + css + javascript) y php, que me pueden ser de gran ayuda para el desarrollo de aplicaciones en el futuro.

Sin dejar de lado el *frontend* del proyecto, me gustaría destacar lo positivo que que resulta utilizar aplicaciones como Figma para la generación de un prototipo. Esta herramienta me ha permitido, por un lado, realizar de forma muy sencilla un boceto del aspecto visual de las pantallas y, por otro lado, trasladar esta información al proyecto gracias a la posibilidad de inspeccionar el código CSS de los elementos.

Esto ha dado como resultado que el aspecto de la página web sea prácticamente calcado al que se había prototipado. Este hecho lo considero muy importante sobre todo porque al principio del proyecto no estaba entre mis prioridades cuidar la parte visual del *frontend* debido a que el desconocimiento que tenía sobre este tema me hacía pensar que debería dedicarle demasiado tiempo.

Finalmente, del desarrollo del trabajo podemos extraer también algunas conclusiones negativas o aspectos a mejorar:

- Se han producido bastantes variaciones en las operaciones de los servicios y métodos del API REST que se han acabado implementando respecto a las que se habían planificado inicialmente. Considero que la causa principal ha sido la falta de experiencia en el desarrollo de este tipo de aplicaciones desde sus etapas más tempranas, lo que me ha dificultado afinar a la hora de establecer unos requisitos.

- La aplicación no cuenta con pruebas para verificar el correcto funcionamiento del código. Un error que considero que he cometido ha sido el de definir una gran cantidad de funcionalidades. El tiempo necesario para el desarrollo de la aplicación ha sido muy elevado, aspecto que he notado sobre todo a la hora de desarrollar el *frontend*. Esto ha provocado que no pudiese dedicar mucho tiempo a realizar pruebas, más allá de las efectuadas con Postman para comprobar que el API REST devolviera los resultados deseados.

- Hubiera sido deseable haber profundizado más en el desarrollo web y utilizar algún *framework* como Angular o Laravel, en lugar de utilizar lenguaje php puro.

Estos últimos puntos quedan como aspectos a mejorar en el desarrollo de futuros proyectos.

5. Glosario

AJAX. *Asynchronous JavaScript and XML*

API REST. Estilo de arquitectura centrado en el desarrollo de servicios web mediante un modelo cliente-servidor.

Backend. Parte del desarrollo centrada en la capa de negocio de la aplicación, es decir, la encargada de suplir de información al *frontend* para presentarla al usuario.

CSS. *Cascade Style Sheets*

Framework. Entorno de trabajo que sigue una estructura concreta y estándar (módulos, lenguaje de programación, librerías) y que tiene el fin de servir de base a los desarrolladores para la creación de software.

Frontend. Parte del desarrollo centrada en la capa con la que interactúa el usuario.

html. *HyperText Markup Language*

IDE. *Integrated Development Environment*

IntelliJ IDEA. IDE de la empresa JetBrains centrado en la programación en el lenguaje Java y otros derivados de éste como Kotlin.

Java. Lenguaje de programación orientado a objetos que se ejecuta en una máquina virtual denominada JVM (*Java Virtual Machine*).

Javascript. Lenguaje de programación de *scripting* utilizado en el desarrollo web para operar en el lado del cliente.

LGPL. *Lesser General Public License*

Metodología ágil. Se trata de una forma de desarrollar software mediante un modelo iterativo en el que se sigue el principio de la flexibilidad. De esta forma, el proyecto pueda adaptarse rápidamente a los cambios.

Microservicio. Estilo arquitectónico que permite desarrollar aplicaciones mediante su división en módulos independientes, lo que aporta beneficios como una mayor flexibilidad y escalabilidad.

PHP. Lenguaje de programación de *scripting* centrado en el desarrollo de aplicaciones web en el lado del servidor.

Postman. Software de la empresa Postman inc. que permite la realización de pruebas sobre un API REST.

REST. *REpresentational State Transfer*

Spring Boot. *Framework* para el lenguaje de programación Java cuyo propósito es la simplificación de tareas. Está extensamente utilizado dentro del ámbito del desarrollo de microservicios.

Visual Studio Code. Editor de Código de la empresa Microsoft que puede ser utilizado con una gran cantidad de lenguajes y al que se le pueden añadir más funcionalidades mediante extensiones.

6. Bibliografía

[1] STACKOVERFLOW. *Handler dispatch failed; nested exception is java.lang.StackOverflowError with root cause.* 2021 Consultado el 13/05/2023. Disponible en: https://stackoverflow.com/questions/68602821/handler-dispatch-failed-nested-exception-is-java-lang-stackoverflowerror-with-r

[2] TINY.CLOUD. *Get TinyMCE's free WYSIWYG HTML editor.* Consultado el 11/05/2023. Disponible en: https://www.tiny.cloud/get-tiny/

[3] GNU.ORG. *GNU Lesser General Public License*. 2007. Consultado el 12/05/2023. Disponible en: https://www.gnu.org/licenses/lgpl-3.0.html

[4] SPRING.IO *Uploading Files*. Consultado el 28/05/2023. Disponible en: https://spring.io/guides/gs/uploading-files/

[5] Design Toolkit. *Metáfora*. Consultado el 06/04/2023. Disponible en: http://design-toolkit.recursos.uoc.edu/es/metafora/

[6] Design Toolkit. *Affordance*. Consultado el 06/04/2023. Disponible en: http://design-toolkit.recursos.uoc.edu/es/affordance/

[7] Design Toolkit. *Retroacción*. Consultado el 06/04/2023. Disponible en: http://design-toolkit.recursos.uoc.edu/es/retroaccion/

[8] Design Toolkit. Restricción. Consultado el 07/04/2023. Disponible en: http://design-toolkit.recursos.uoc.edu/es/restriccion/

[9] Design Toolkit. *Modelo mental*. Consultado el 07/04/2023. Disponible en: http://design-toolkit.recursos.uoc.edu/es/modelo-mental/

[10] XAMPP. Consultado el 13/03/2023. Disponible en: https://www.apachefriends.org/es/index.html

[11] Docker Compose Overview. Consultado el 07/04/2023. Disponible en: https://docs.docker.com/compose/

[12] Open JDK 11. Consultado el 07/04/2023. Disponible en: https://openjdk.org/projects/jdk/11/

[13] Apache Maven Project: Consultado el 11/04/2023. Disponible en: https://maven.apache.org/

[14] Accessing data with JPA. Consultado el 11/04/2023. Disponible en: https://spring.io/guides/gs/accessing-data-jpa/

[15] PostgreSQL. Consultado el 12/04/2023. Disponible en: https://www.postgresql.org/

[16] Apache Kafka. Consultado el 12/04/2023. Disponible en: https://kafka.apache.org/

[17] Spring Boot. Consultado el 12/02/2023. Enlace: https://spring.io/

[18] Spring for Apache Kafka. Consultado el 12/04/2023. Disponible en: https://docs.spring.io/spring-kafka/reference/html/

[19] Project Lombok. Consultado el 14/04/2023. Disponible en: https://projectlombok.org

[20] Database initialization. Consultado el 14/04/2023. Disponible en: https://docs.spring.io/spring-boot/docs/2.1.x/reference/html/howto-database-initialization.html

[21] Vivek Balasubramaniam. 2022. BAELDING. *Defining JPA Entities*. Consultado el 14/04/2023. Disponible en: https://www.baeldung.com/jpa-entities

[22] Figma. Consultado el 14/04/2023. Disponible en: https://www.figma.com

[23] Pattern: Descompose by subdomain. Consultado el 12/04/2023. Disponible en: https://microservices.io/patterns/decomposition/decompose-by-subdomain.html

[24] Richardson, C. *Microservices Patterns*. 2019. Manning Publications Co.

[25] Carnell, C & Huaylupo Sánchez, I. *Spring Microservices in action.* 2021. Manning Publicacions Co.

Iconos utilizados

FLATICON. *Icono oficina*. Consultado el 12/03/2023. Disponible en: https://www.flaticon.es/icono-gratis/oficina_10019567

FLATICON. *EDIFICIO DE OFICINAS*. Consultado el 05/04/2023. Disponible en: https://www.flaticon.es/icono-gratis/edificio-de-oficinas_3633290

FLATICON. *PROFILE-USER*. Consultado el 05/04/2023. Disponible en: https://www.flaticon.com/free-icon/profile-user_64572

FLATICON. *NEWSPAPER*. Consultado el 07/04/2023. Disponible en: https://www.flaticon.com/free-icon/newspaper_4253340

FLATICON. *ACCOUNTING*. Consultado el 08/04/2023. Disponible en: https://www.flaticon.com/free-icon/accounting_1570887

FLATICON. *MONEDA-DE-DINERO*. Consultado el 08/04/2023. Disponible en: https://www.flaticon.es/icono-gratis/moneda-de-dinero_4140926

FLATICON. *CHAT*. Consultado el 08/04/2023. Disponible en: https://www.flaticon.com/free-icon/chat_10311452

FLATICON. *SAFE*. Consultado el 08/04/2023. Disponible en: https://www.flaticon.com/free-icon/safe_10146602

FLATICON. *CREDIT-CARDS*. Consultado el 08/04/2023. Disponible en: https://www.flaticon.com/free-icon/credit-cards_1558934

FLATICON. *GOOGLE-CALENDAR*. Consultado el 08/04/2023. Disponible en: https://www.flaticon.com/free-icon/google-calendar_5968499

FLATICON. *PLUS*. Consultado el 08/04/2023. Disponible en: https://www.flaticon.com/free-icon/plus_1828919

FLATICON. *DELETE*. Consultado el 09/04/2023. Disponible en: https://www.flaticon.com/free-icon/delete_1450571

FLATICON. *PDF*. Consultado el 09/04/2023. Disponible en: https://www.flaticon.com/free-icon/pdf_337946

FLATICON. *NOTIFICATION*. Consultado el 09/04/2023. Disponible en: https://www.flaticon.com/free-icon/notification_1827370

7. Anexos

7.1 Localización proyecto

El proyecto se encuentra disponible en los siguientes enlaces:

Fichero docker-compose.yml para el despliegue de los microservicios

https://github.com/jlzorita/adminh-infra

Microservicios

https://github.com/jlzorita/adminh-user
https://github.com/jlzorita/adminh-crm
https://github.com/jlzorita/adminh-core

Aplicación web

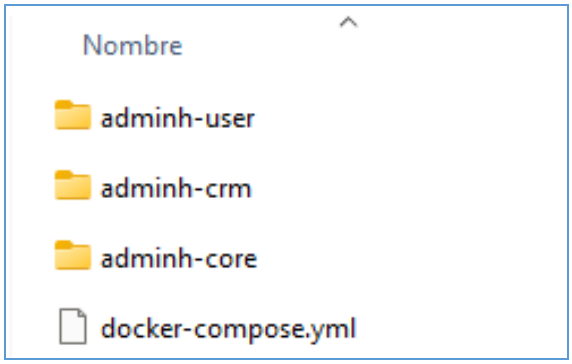
https://github.com/jlzorita/adminh-web

7.2 Instalación y ejecución

Se listan a continuación los pasos para poner en marcha la aplicación:

Microservicios

1. Clonar la infraestructura y los microservicios de tal forma que tengamos la siguiente estructura de ficheros y directorios:



2. Desde el directorio en el que se encuentra el fichero `docker-compose.yml` ejecutar la siguiente orden:

```
docker-compose up --build -d
```

3. Una vez ejecutado, las llamadas a API REST se realizan con los siguientes puertos:

Microservicio	Puerto
adminh-user	18081
adminh-crm	18082
admihn-core	18083

También se puede acceder a Adminer utilizando el puerto 18080.

Aplicación web

1. Se debe tener un servidor web con php y php-curl instalados:

   a. **Windows**: Se puede utilizar la última versión de XAMPP disponible en su página web: https://www.apachefriends.org

   Una vez lanzado el panel de control de XAMPP, únicamente es necesario iniciar el servidor web Apache. Se deben colocar los ficheros de la aplicación en el directorio `htdocs` dentro del directorio en el que se haya instalado XAMPP.

   Finalmente, para activar cURL se debe descomentar la siguiente línea en el fichero `xampp/php/php.ini`:

   ```
   ;extension=curl
   ```

   b. **GNU/Linux (ejemplo con Ubuntu)**: Es necesario instalar los siguientes componentes. Se indican las órdenes en el caso de que se use una distribución Ubuntu:

   Apache
   ```
   $ sudo apt install apache2
   ```

   PHP
   ```
   $ sudo apt install php
   $ sudo apt install libapache2-mod-php
   $ sudo apt install php-curl
   ```

   Por defecto, el directorio utilizado por Apache para almacenar los ficheros de la web es el siguiente:

   ```
   /var/www/html
   ```

   Una vez clonado el repositorio, es importante dar permisos de escritura al siguiente directorio, ya que es utilizado para almacenar temporalmente ficheros que sube el usuario:

   ```
   administrador/post/facturas/
   ```

2. Editar el fichero `config.php`:

En este fichero se encuentran diferentes variables globales, como los puertos utilizados por cada microservicio.

```php
<?php

if(session_status() !== PHP_SESSION_ACTIVE) session_start();
$GLOBALS["host"] = "localhost";
$GLOBALS["exHost"] = "localhost";
$GLOBALS["puertoUser"] = "18081";
$GLOBALS["puertoCrm"] = "18082";
$GLOBALS["puertoCore"] = "18083";

?>
```

Las dos variables a tener en cuenta para el despliegue son las siguientes:

- **$GLOBALS[“exHost”]**: Es la dirección del host desde el punto de vista del *frontend* / cliente. Cuando una operación REST se realice desde el extremo del usuario y su navegador, este deberá conectar con el servidor en el que están alojados los microservicios. Si por ejemplos estos están alojados en un host con nombre `jlztfg.ddns.net` sería:

  ```
  $GLOBALS[“exHost”] = “jlztfg.ddns.net”;
  ```

- **$GLOBALS[“host”]**: Es la dirección del host desde el punto de vista del servidor PHP. Hay que tener en cuenta que PHP trabaja en el lado del servidor, por lo que sí éste se encuentra funcionando en el mismo sistema en el que están ejecutándose los microservicios, las llamadas al API REST serán a `localhost`, a diferencia de las llamadas que se hagan desde el extremo cliente. En el caso descrito sería:

  ```
  $GLOBALS[“host”] = “localhost”;
  ```

3. Una vez en marcha, se puede acceder a la aplicación y hacer *login* con los datos de usuario siguientes:

Usuario	Contraseña	Tipo de usuario
jzorita	pass	Propietario
yyang	pass	Propietario presidente
ilopez	pass	Administrador

El despliegue de ambas plataformas se puede automatizar utilizando un *engine* de orquestación. A continuación, se muestra el contenido de un *playbook* de Ansible que se podría utilizar para este fin:

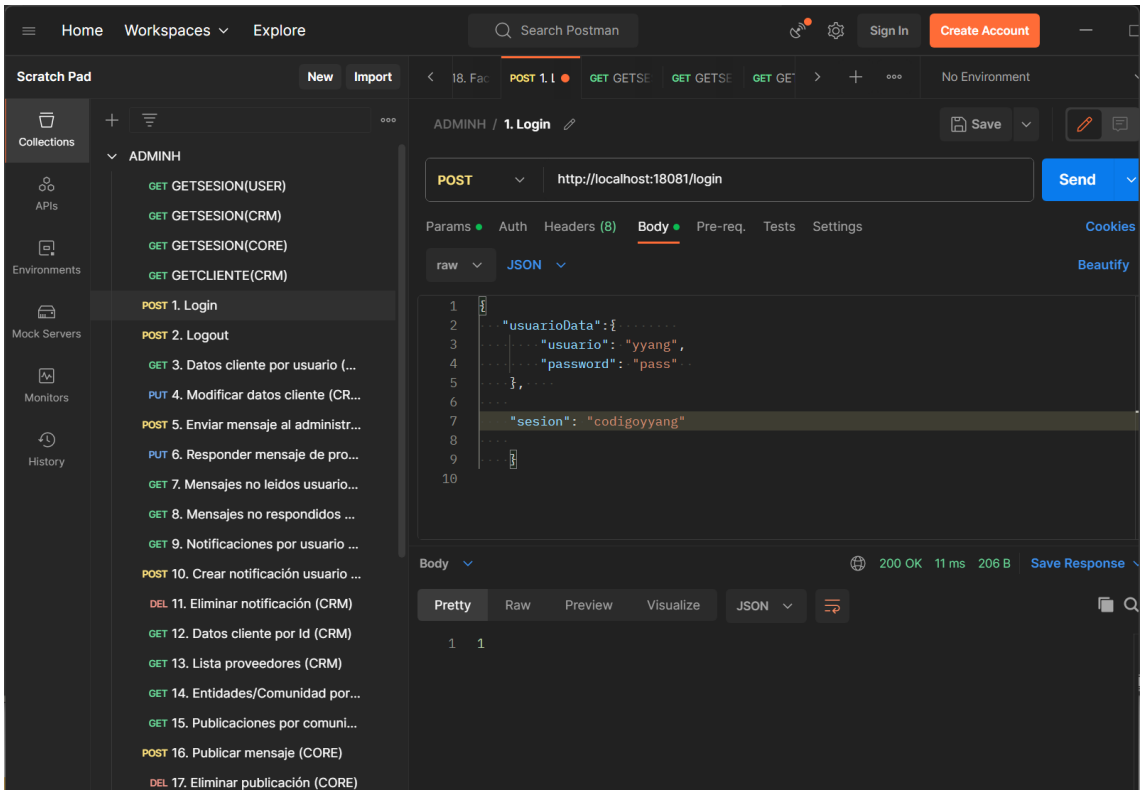
2: jzorita@jzorita-virtual-machine: ~/ansible

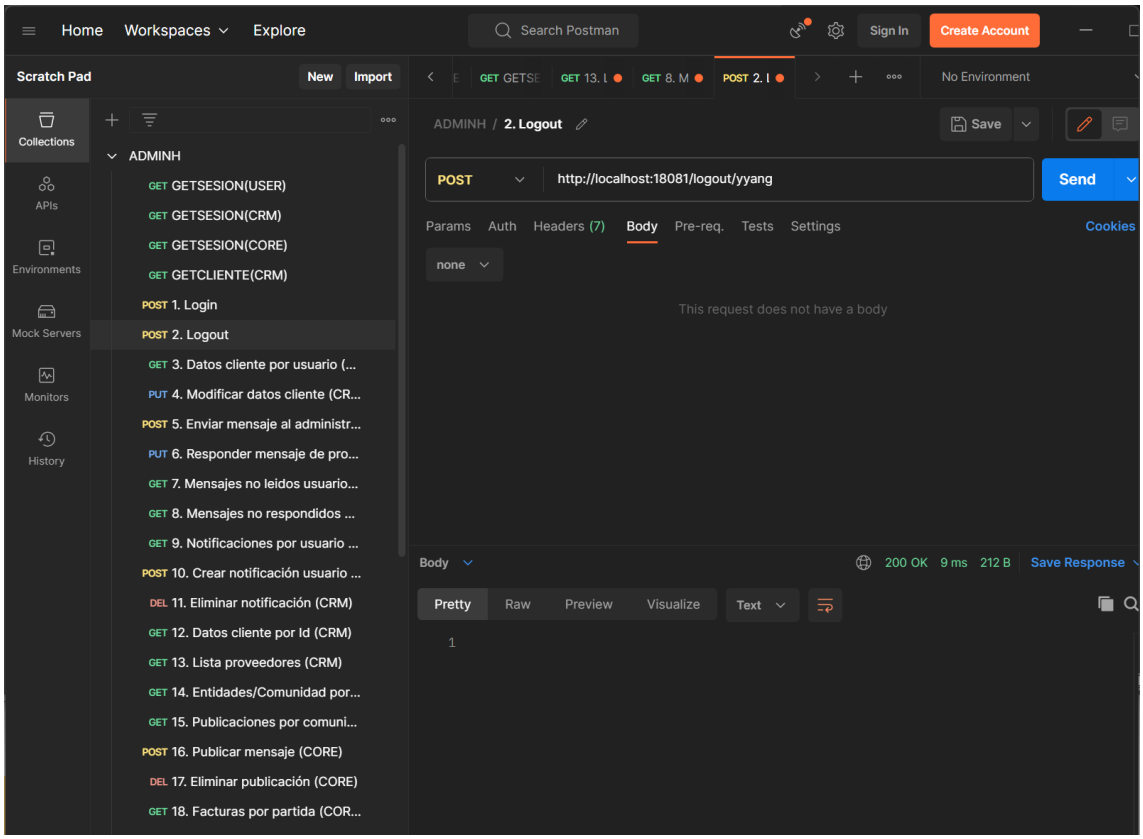
GNU nano 6.2 playbook.yml

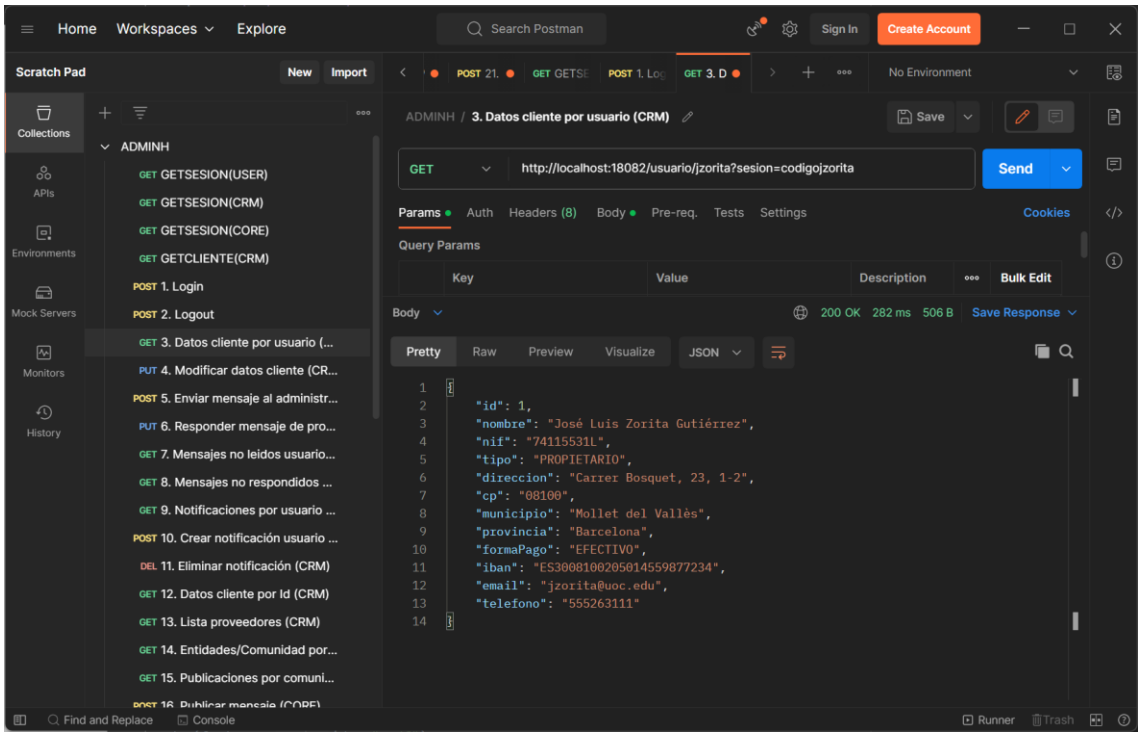
```
---
- hosts: all
  become: yes
  tasks:
  - name: Clonar repositorio con la aplicación web
    git:
      repo: https://github.com/jlzorita/adminh-web
      dest: /var/www/html
      clone: yes
      force: yes
      update: yes
      version: main

  - name: Clonar repositorio con YML de docker-compose
    git:
      repo: https://github.com/jlzorita/adminh-infra
      dest: /home/jzorita/ansible/dst
      clone: yes
      force: yes
      update: yes
      version: main

  - name: Clonar repositorio con microservicio adminh-user
    git:
      repo: https://github.com/jlzorita/adminh-user
      dest: /home/jzorita/ansible/dst/adminh-user
      clone: yes
      force: yes
      update: yes
      version: main

  - name: Clonar el repositorio con el microservicio adminh-crm
    git:
      repo: https://github.com/jlzorita/adminh-crm
      dest: /home/jzorita/ansible/dst/adminh-crm
      clone: yes
      force: yes
      update: yes
      version: main

  - name: Clonar el repositorio con el microservicio adminh-core
    git:
      repo: https://github.com/jlzorita/adminh-core
      dest: /home/jzorita/ansible/dst/adminh-core
      clone: yes
      force: yes
      update: yes
      version: main

  - name: Desplegar microservicios
    community.docker.docker_compose:
      project_src: /home/jzorita/ansible/dst/
      files:
      - docker-compose.yml
```

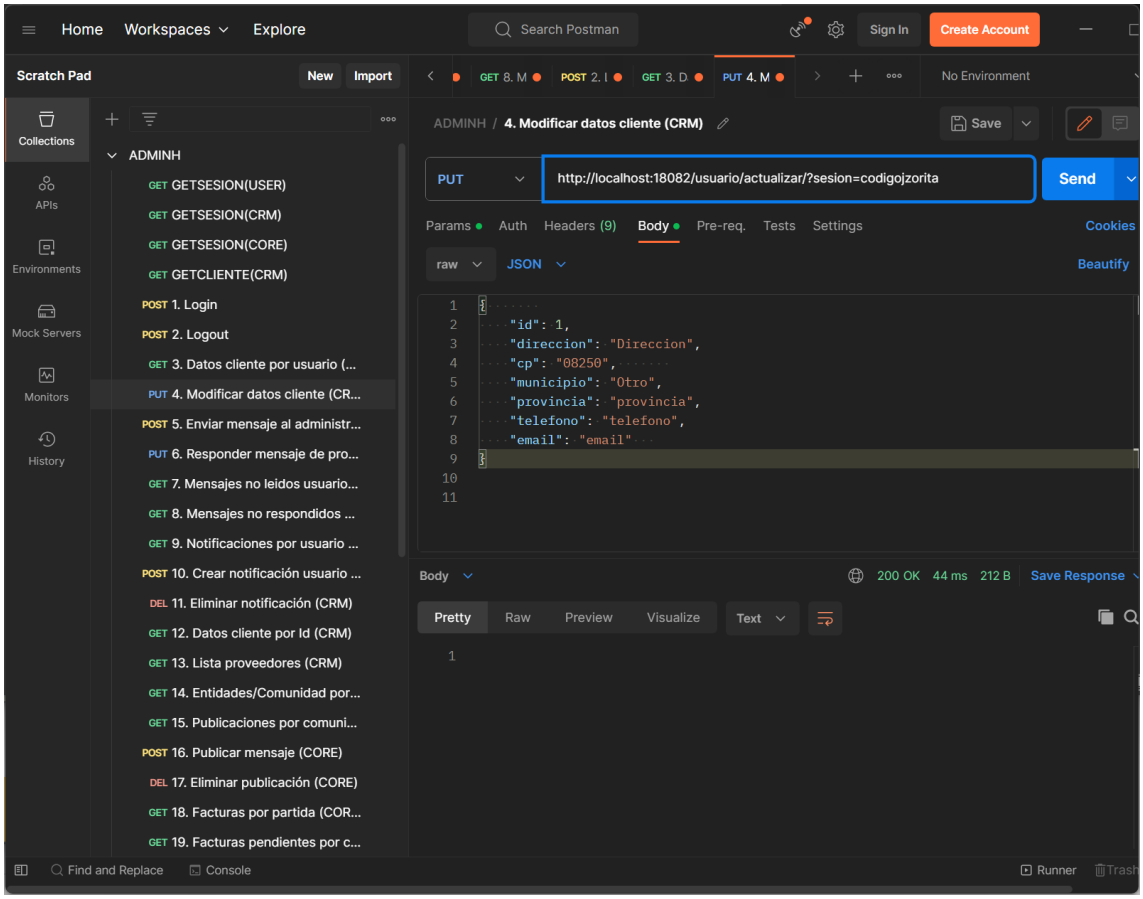
7.3 Pruebas servicios del API REST con Postman

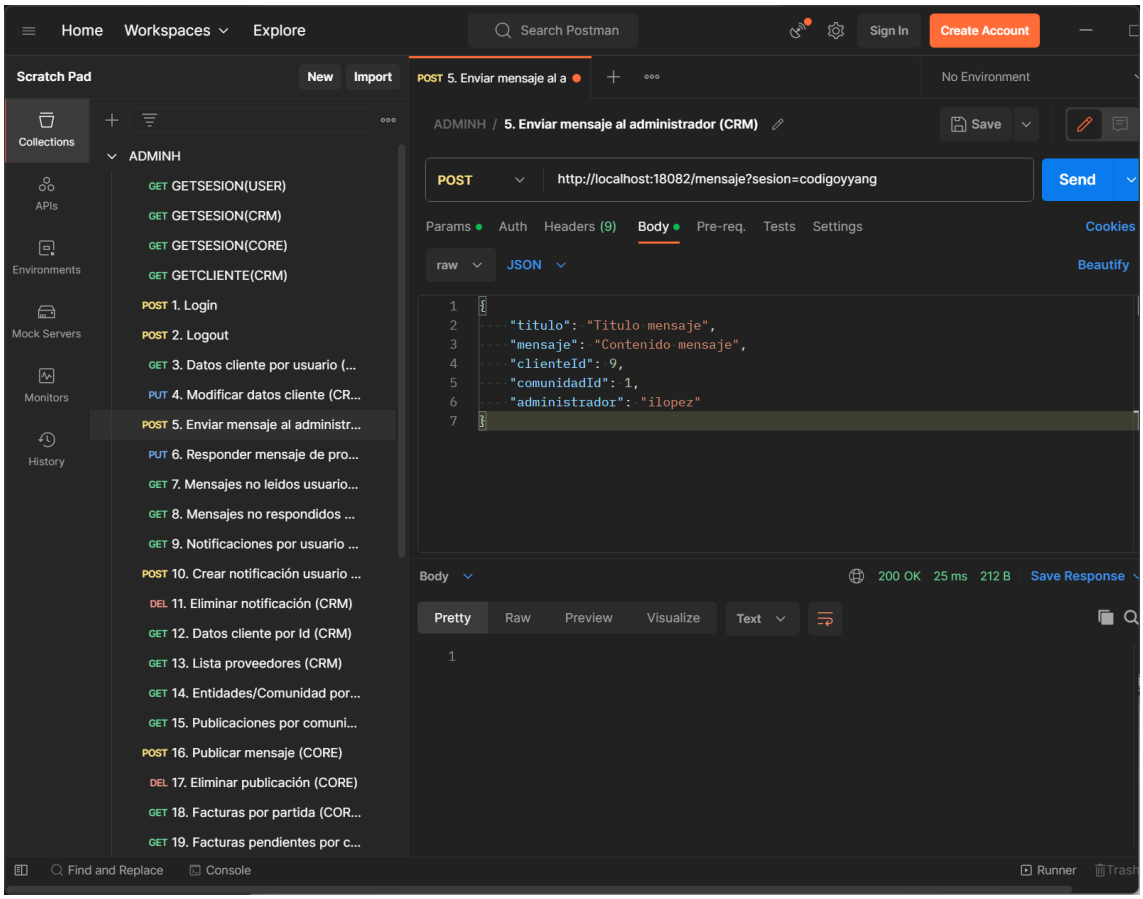
Captura de la ejecución de todos los métodos de los servicios API REST en Postman:

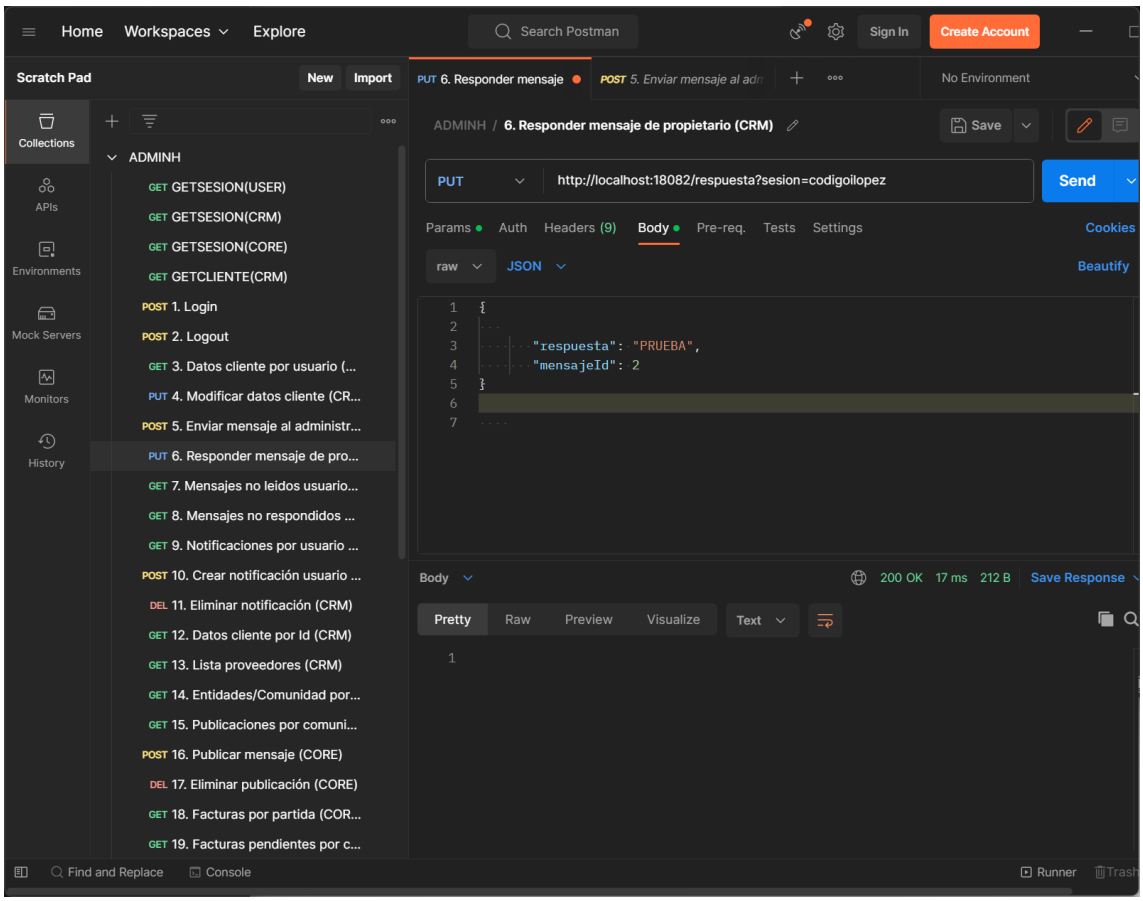


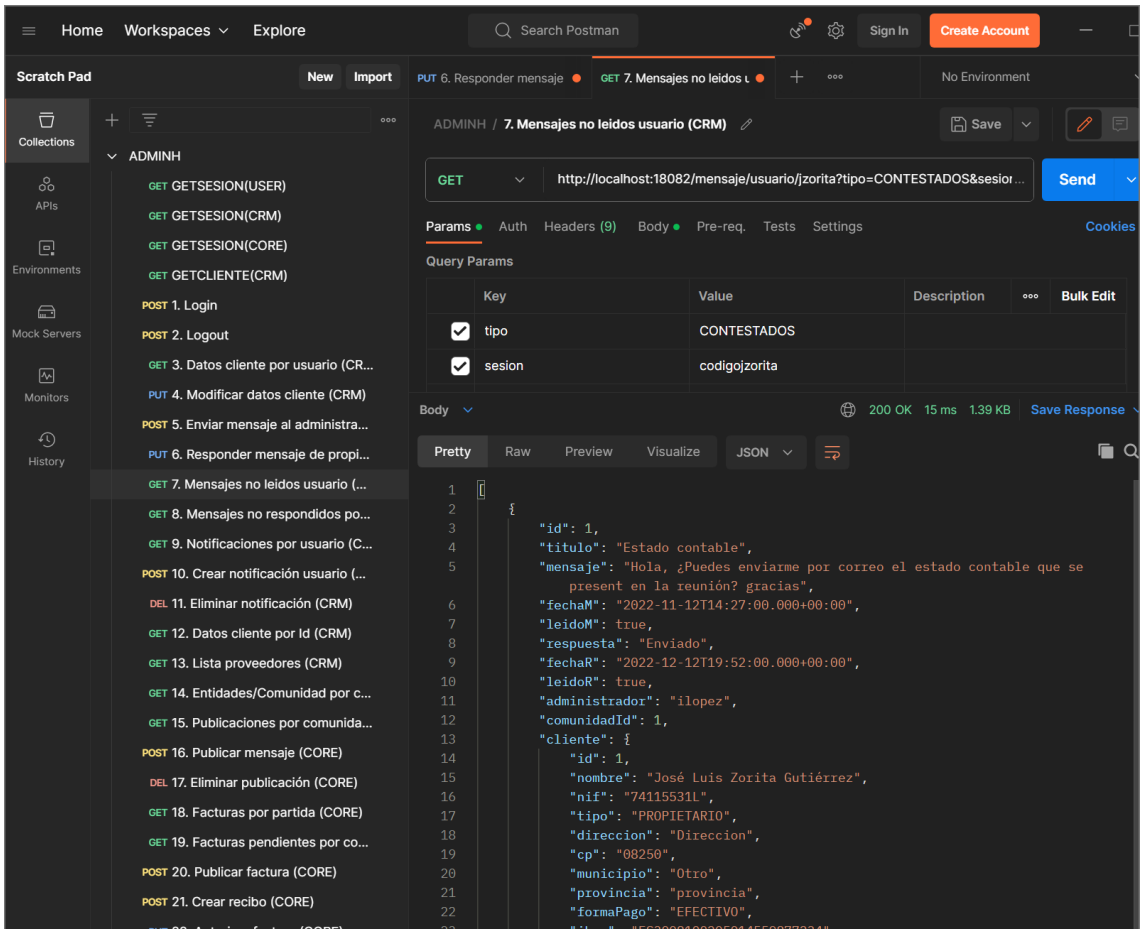


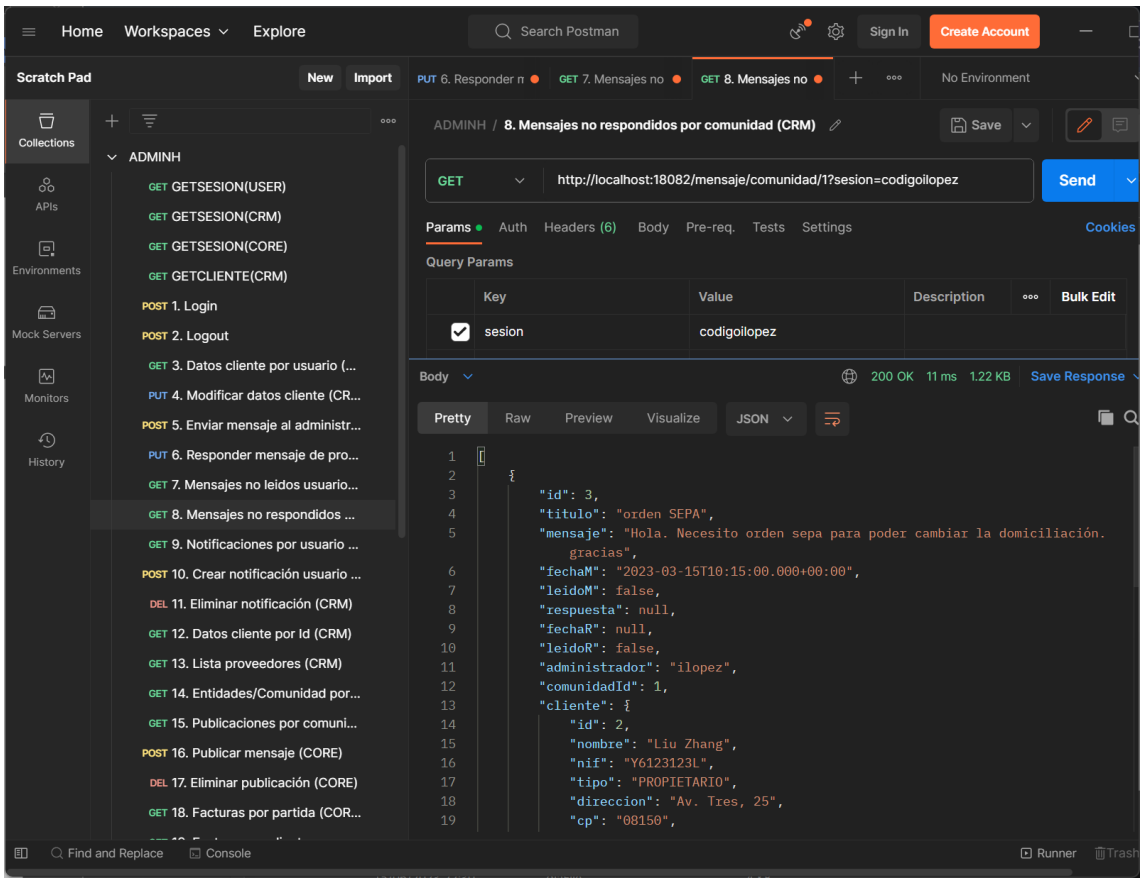

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH
GET GETSESION(USER)
GET GETSESION(CRM)
GET GETSESION(CORE)
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
ADMINH / 3. Datos cliente por usuario (CRM)
Save
GET
http://localhost:18082/usuario/jzorita?sesion=codigojzorita
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
200 OK
282 ms
506 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"nombre": "José Luis Zorita Gutiérrez",
"nif": "74115531L",
"tipo": "PROPIETARIO",
"direccion": "Carrer Bosquet, 23, 1-2",
"cp": "08100",
"municipio": "Mollet del Vallès",
"provincia": "Barcelona",
"formaPago": "EFECTIVO",
"iban": "ES3008100205014559877234",
"email": "jzorita@uoc.edu",
"telefono": "555263111"
}
Find and Replace
Console
Runner
Trash

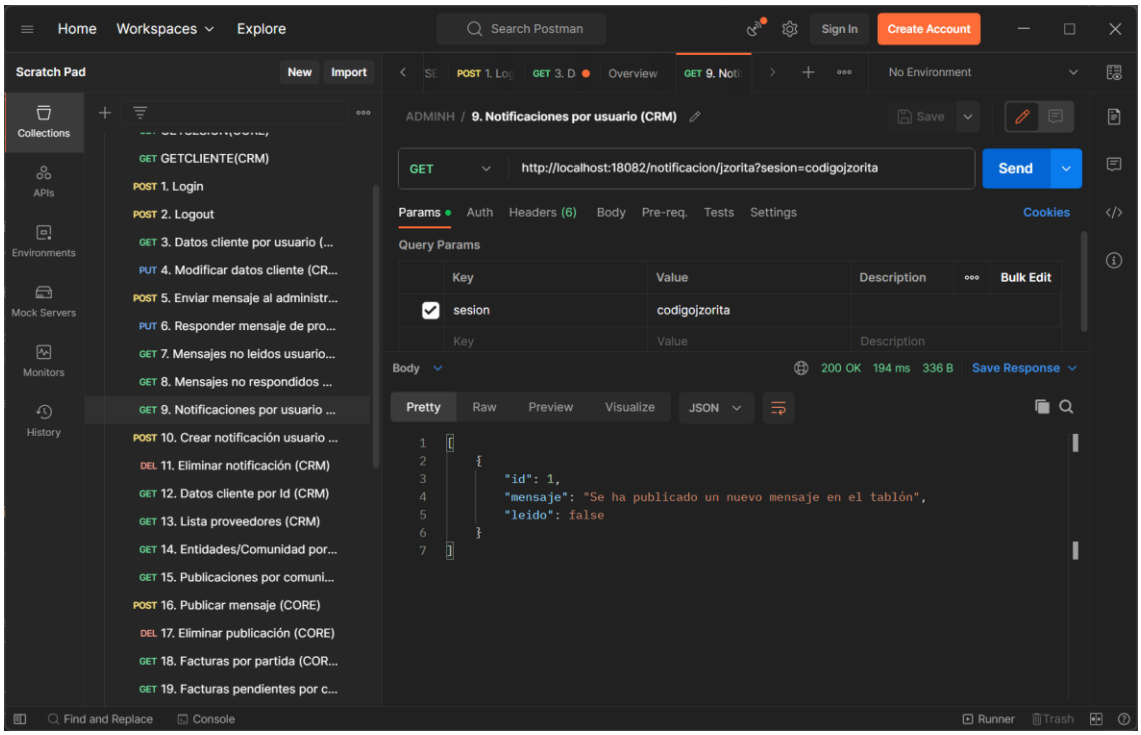

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH
GET GETSESION(USER)
GET GETSESION(CRM)
GET GETSESION(CORE)
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
ADMINH / 4. Modificar datos cliente (CRM)
Save
PUT
http://localhost:18082/usuario/actualizar/?sesion=codigojzorita
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"id": 1,
"direccion": "Direccion",
"cp": "08250",
"municipio": "Otro",
"provincia": "provincia",
"telefono": "telefono",
"email": "email"
}
Body
200 OK
44 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
Runner
Trash

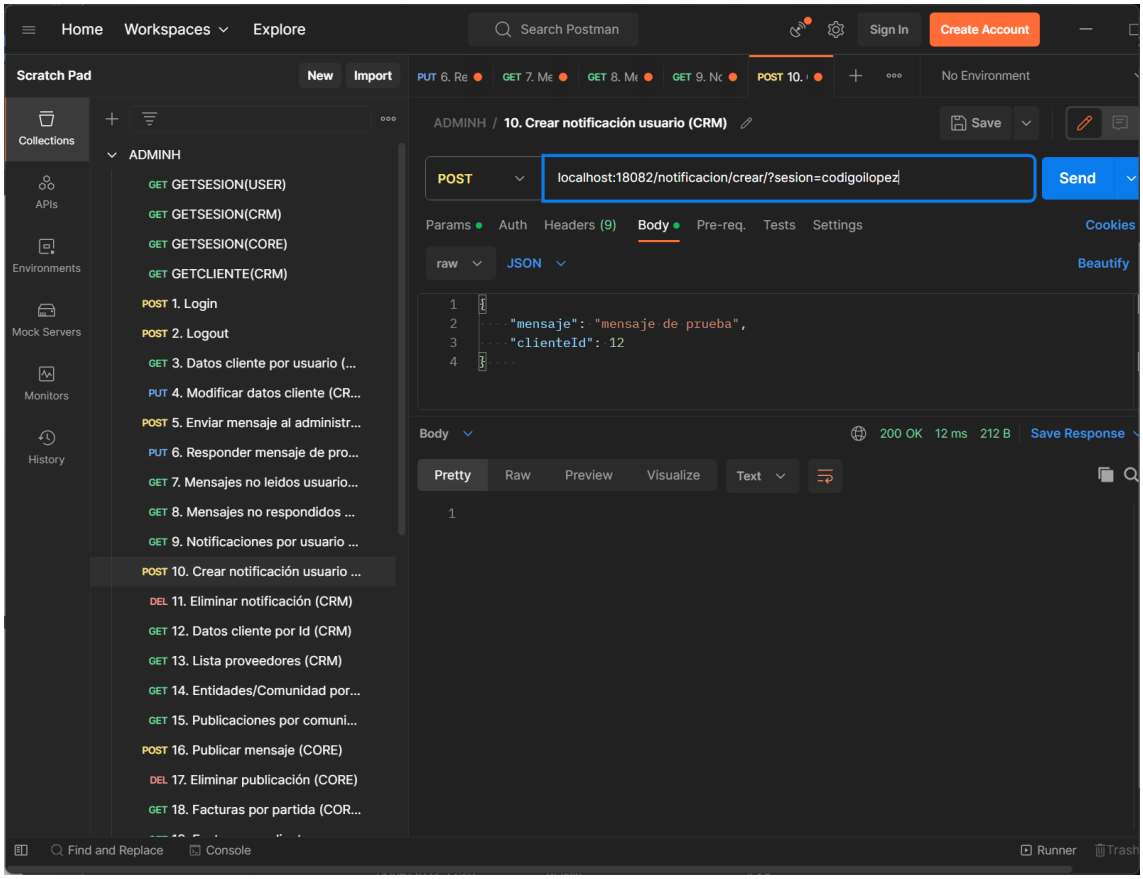

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
POST 5. Enviar mensaje al a
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH
GET GETSESION(USER)
GET GETSESION(CRM)
GET GETSESION(CORE)
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
ADMINH / 5. Enviar mensaje al administrador (CRM)
Save
POST
http://localhost:18082/mensaje?sesion=codigoyyang
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"titulo": "Titulo mensaje",
"mensaje": "Contenido mensaje",
"clienteId": 9,
"comunidadId": 1,
"administrador": "ilopez"
}
Body
200 OK
25 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
Runner

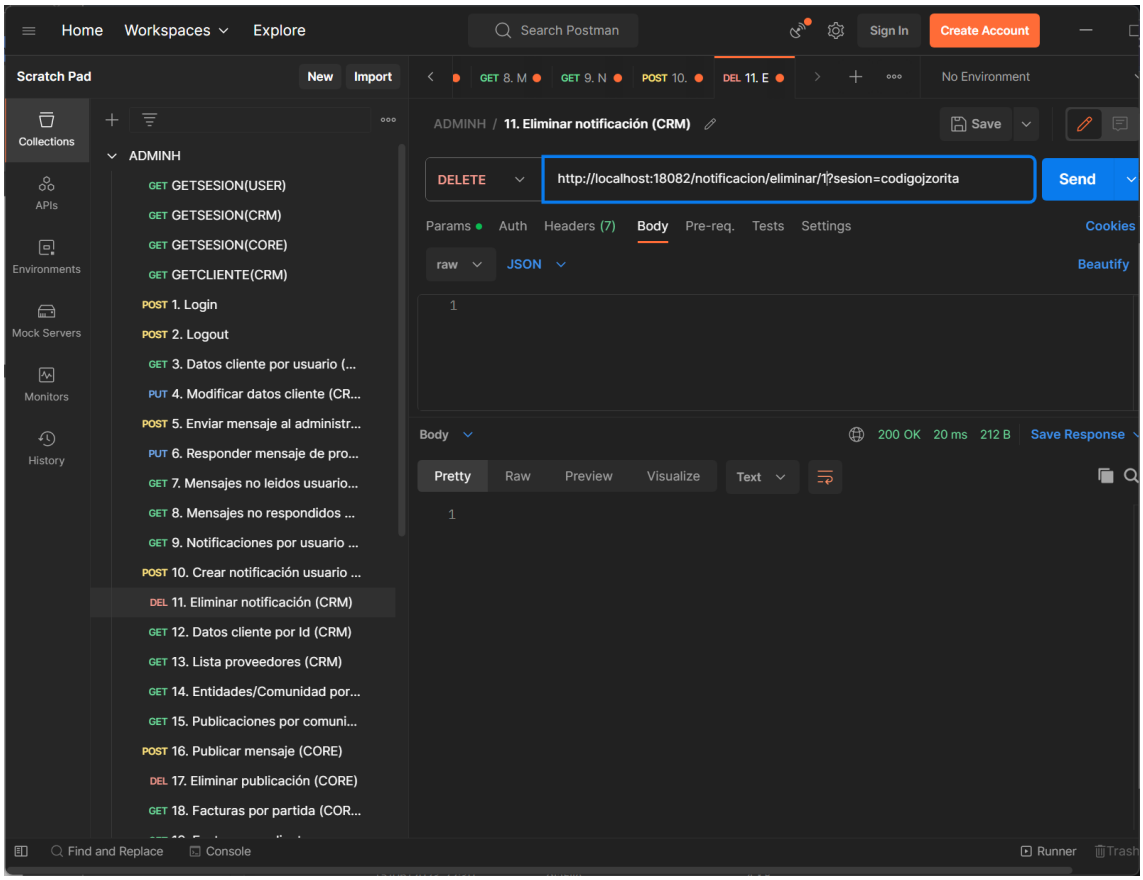

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
PUT 6. Responder mensaje
POST 5. Enviar mensaje al adr
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH
GET GETSESION(USER)
GET GETSESION(CRM)
GET GETSESION(CORE)
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
ADMINH / 6. Responder mensaje de propietario (CRM)
Save
PUT
http://localhost:18082/respuesta?sesion=codigoilopez
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"respuesta": "PRUEBA",
"mensajeId": 2
}
Body
200 OK
17 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
Runner

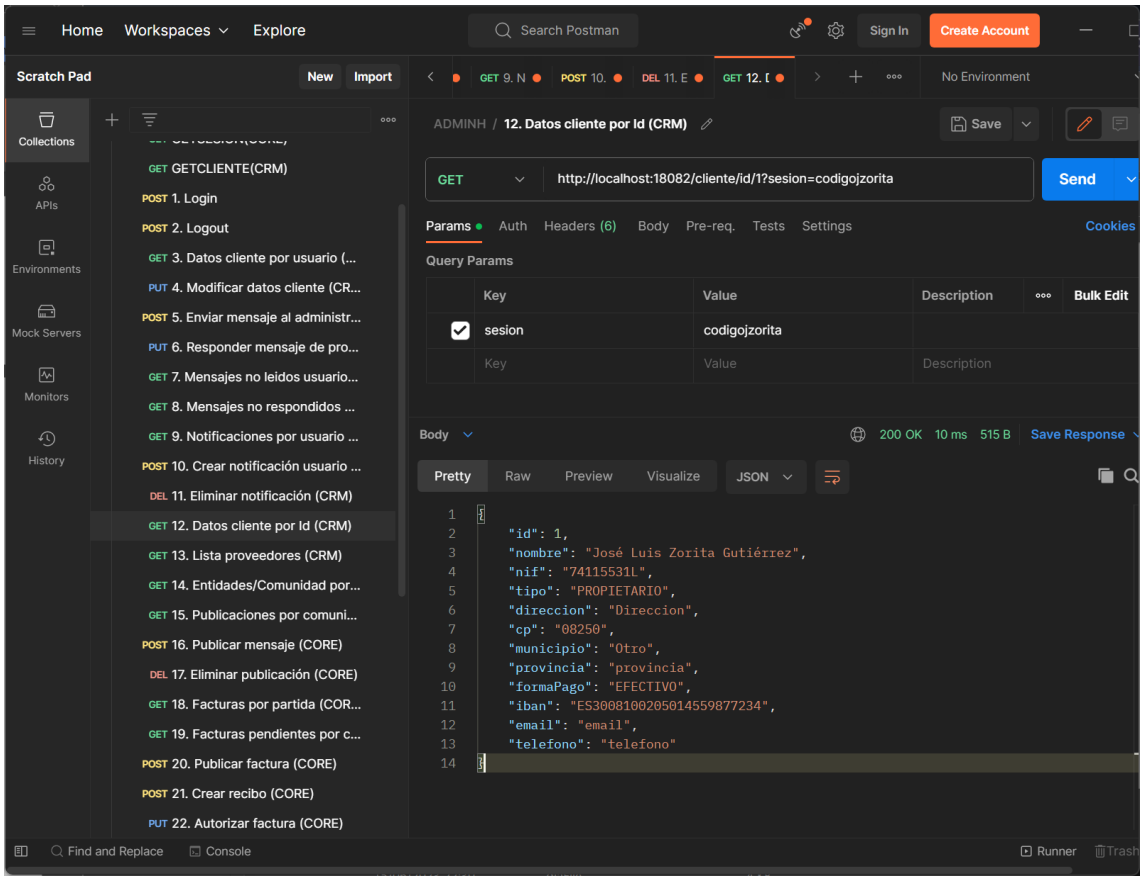

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
PUT 6. Responder mensaje
GET 7. Mensajes no leidos u
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH
GET GETSESION(USER)
GET GETSESION(CRM)
GET GETSESION(CORE)
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (CR...
PUT 4. Modificar datos cliente (CRM)
POST 5. Enviar mensaje al administra...
PUT 6. Responder mensaje de propi...
GET 7. Mensajes no leidos usuario (...
GET 8. Mensajes no respondidos po...
GET 9. Notificaciones por usuario (C...
POST 10. Crear notificación usuario (...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por c...
GET 15. Publicaciones por comunida...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (CORE)
GET 19. Facturas pendientes por co...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
ADMINH / 7. Mensajes no leidos usuario (CRM)
Save
GET
http://localhost:18082/mensaje/usuario/jzorita?tipo=CONTESTADOS&sesior...
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
tipo
CONTESTADOS
sesion
codigojzorita
Body
200 OK
15 ms
1.39 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"titulo": "Estado contable",
"mensaje": "Hola, ¿Puedes enviarme por correo el estado contable que se
present en la reunión? gracias",
"fechaM": "2022-11-12T14:27:00.000+00:00",
"leidoM": true,
"respuesta": "Enviado",
"fechaR": "2022-12-12T19:52:00.000+00:00",
"leidoR": true,
"administrador": "ilopez",
"comunidadId": 1,
"cliente": {
"id": 1,
"nombre": "José Luis Zorita Gutiérrez",
"nif": "74115531L",
"tipo": "PROPIETARIO",
"direccion": "Direccion",
"cp": "08250",
"municipio": "Otro",
"provincia": "provincia",
"formaPago": "EFECTIVO",

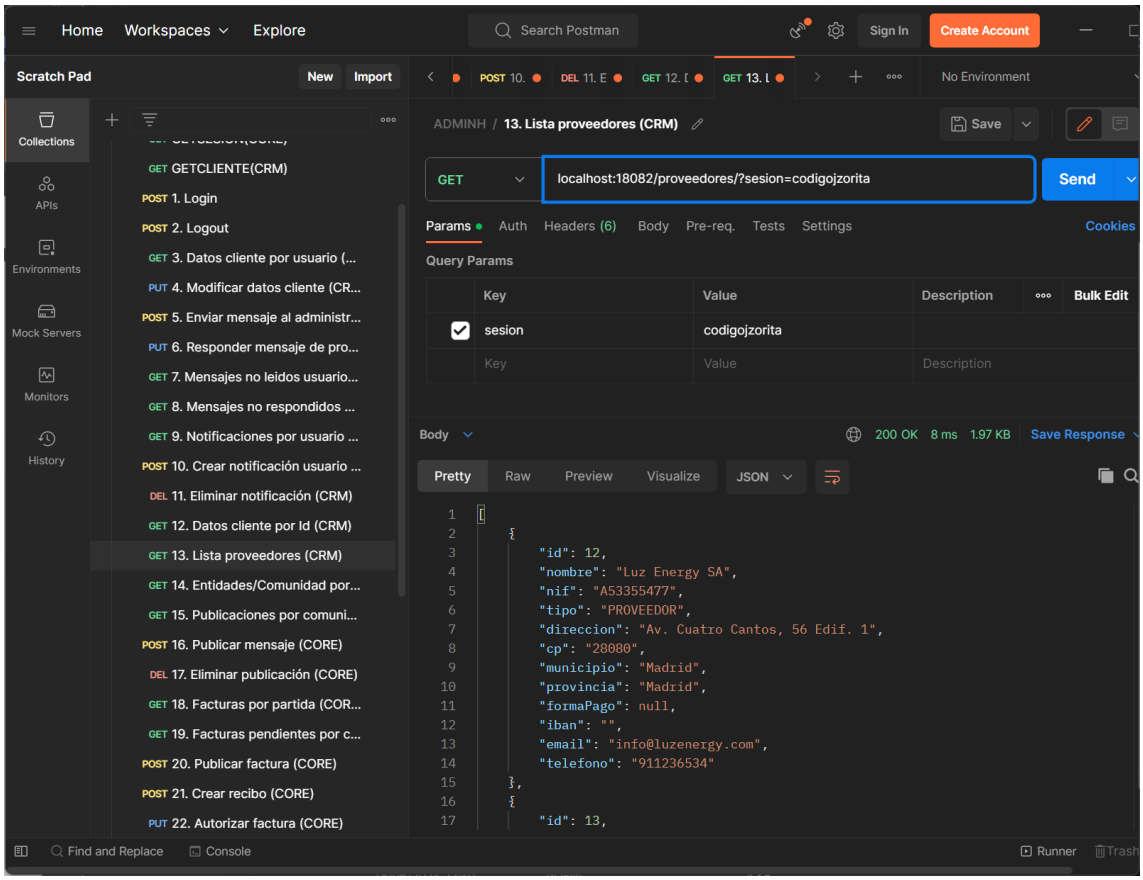

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
PUT 6. Responder n
GET 7. Mensajes no
GET 8. Mensajes no
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH
GET GETSESION(USER)
GET GETSESION(CRM)
GET GETSESION(CORE)
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
ADMINH / 8. Mensajes no respondidos por comunidad (CRM)
Save
GET
http://localhost:18082/mensaje/comunidad/1?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
11 ms
1.22 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 3,
"titulo": "orden SEPA",
"mensaje": "Hola. Necesito orden sepa para poder cambiar la domiciliación.
gracias",
"fechaM": "2023-03-15T10:15:00.000+00:00",
"leidoM": false,
"respuesta": null,
"fechaR": null,
"leidoR": false,
"administrador": "ilopez",
"comunidadId": 1,
"cliente": {
"id": 2,
"nombre": "Liu Zhang",
"nif": "Y6123123L",
"tipo": "PROPIETARIO",
"direccion": "Av. Tres, 25",
"cp": "08150",
Find and Replace
Console
Runner

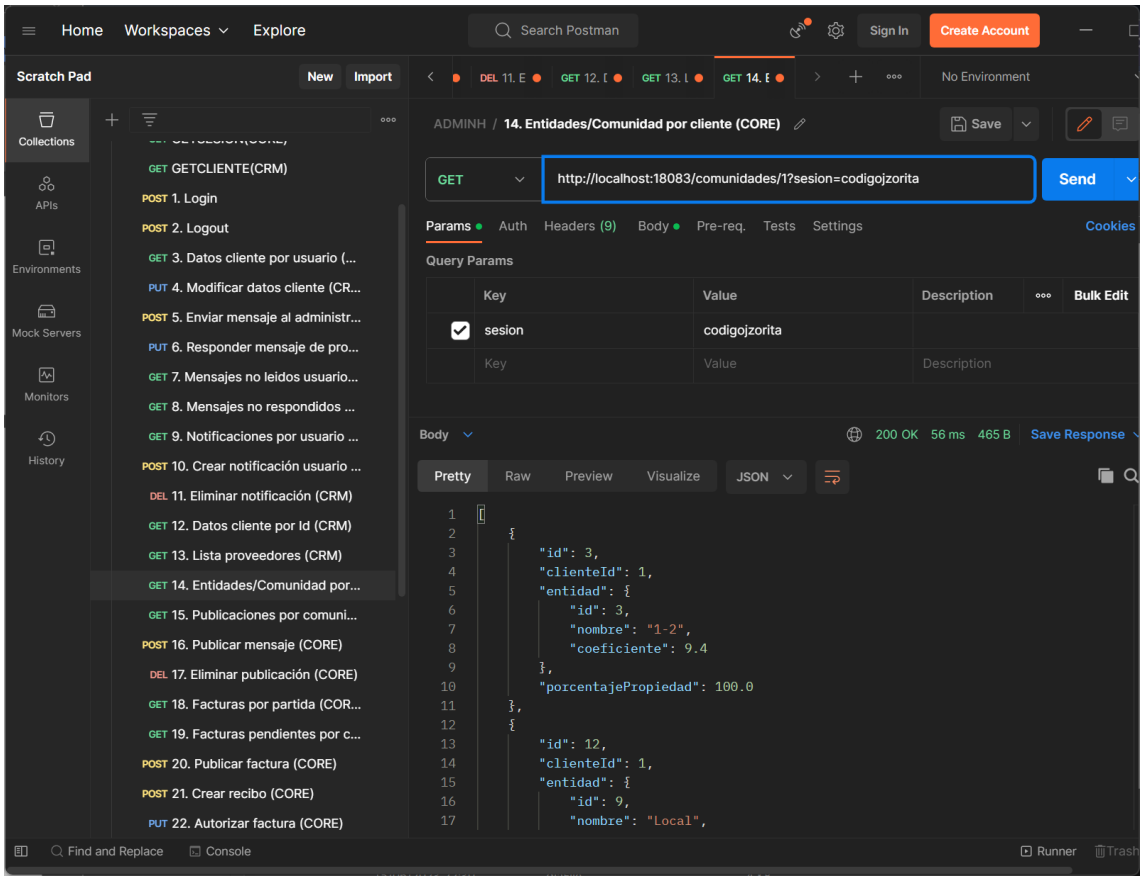

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
ADMINH / 9. Notificaciones por usuario (CRM)
Save
GET
http://localhost:18082/notificacion/jzorita?sesion=codigojzorita
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigojzorita
Body
200 OK
194 ms
336 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"mensaje": "Se ha publicado un nuevo mensaje en el tablón",
"leido": false
}
Find and Replace
Console
Runner
Trash

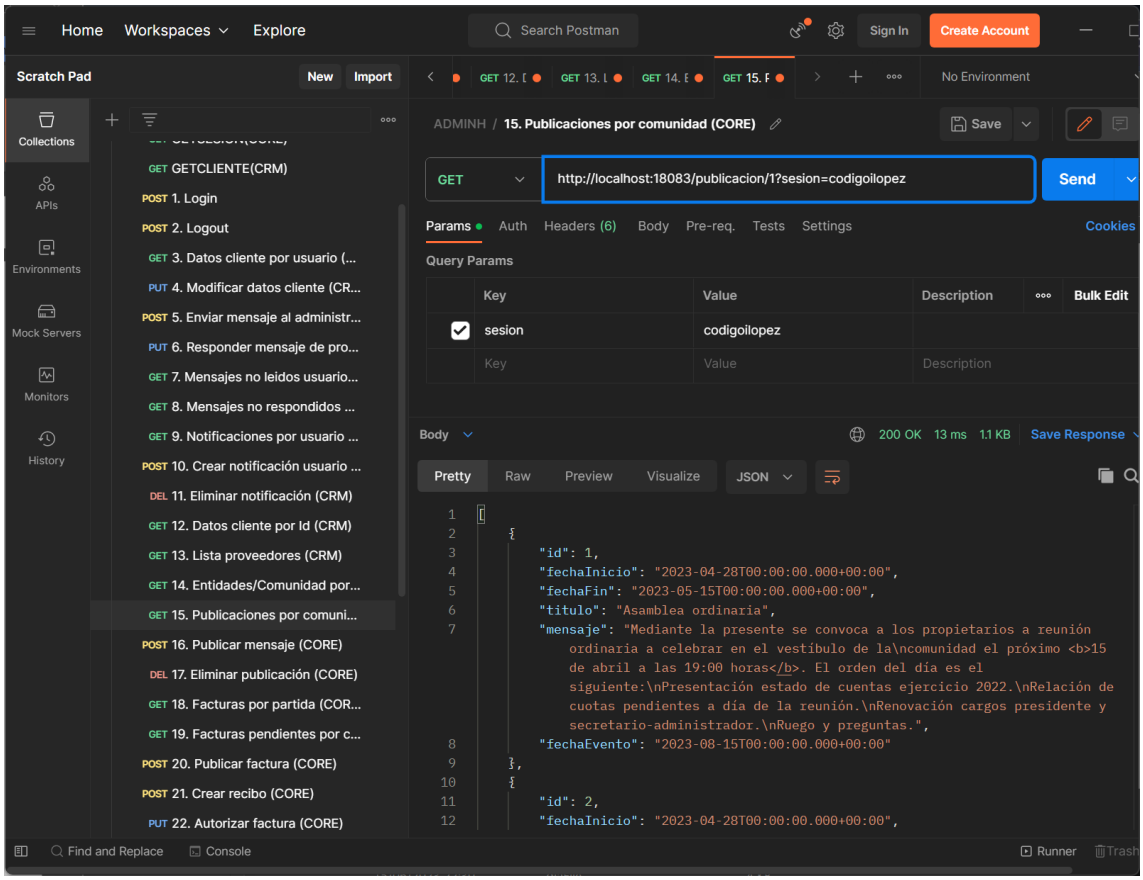

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH
GET GETSESION(USER)
GET GETSESION(CRM)
GET GETSESION(CORE)
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
ADMINH / 10. Crear notificación usuario (CRM)
Save
POST
localhost:18082/notificacion/crear/?sesion=codigoilopez
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"mensaje": "mensaje de prueba",
"clienteId": 12
}
Body
200 OK
12 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
Runner
Trash

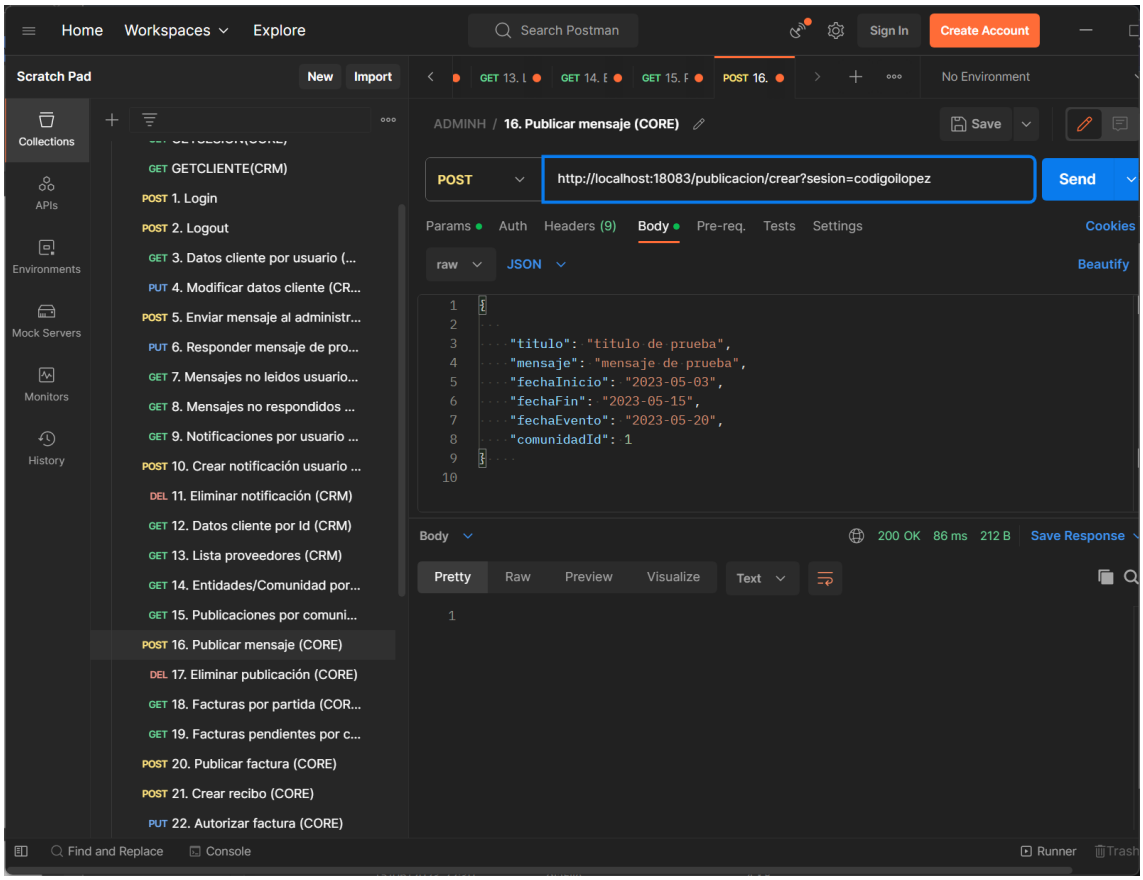

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH
GET GETSESION(USER)
GET GETSESION(CRM)
GET GETSESION(CORE)
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
ADMINH / 11. Eliminar notificación (CRM)
Save
DELETE
http://localhost:18082/notificacion/eliminar/1?sesion=codigojzorita
Send
Params
Auth
Headers (7)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
1
Body
200 OK
20 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1
Find and Replace
Console
Runner

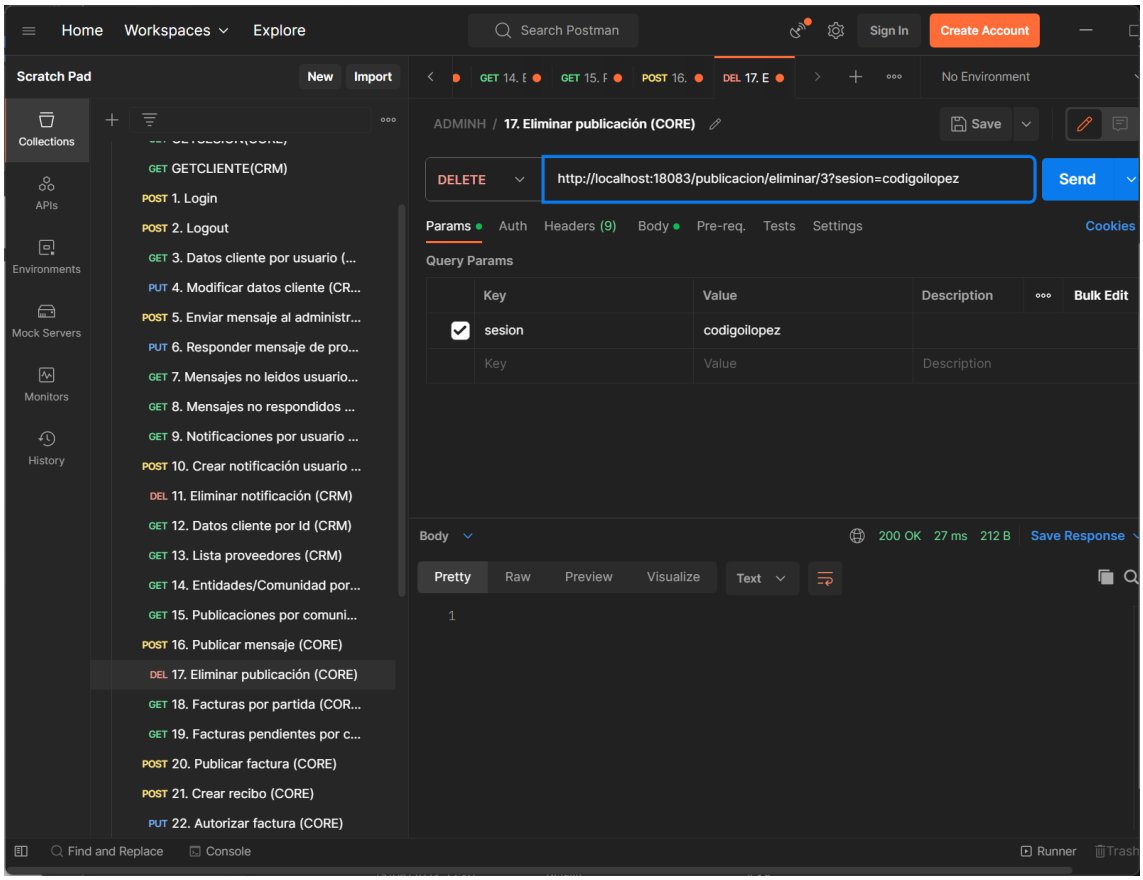

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
ADMINH / 12. Datos cliente por Id (CRM)
Save
GET
http://localhost:18082/cliente/id/1?sesion=codigojzorita
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigojzorita
Body
200 OK
10 ms
515 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"nombre": "José Luis Zorita Gutiérrez",
"nif": "74115531L",
"tipo": "PROPIETARIO",
"direccion": "Direccion",
"cp": "08250",
"municipio": "Otro",
"provincia": "provincia",
"formaPago": "EFECTIVO",
"iban": "ES3008100205014559877234",
"email": "email",
"telefono": "telefono"
}
Find and Replace
Console
Runner

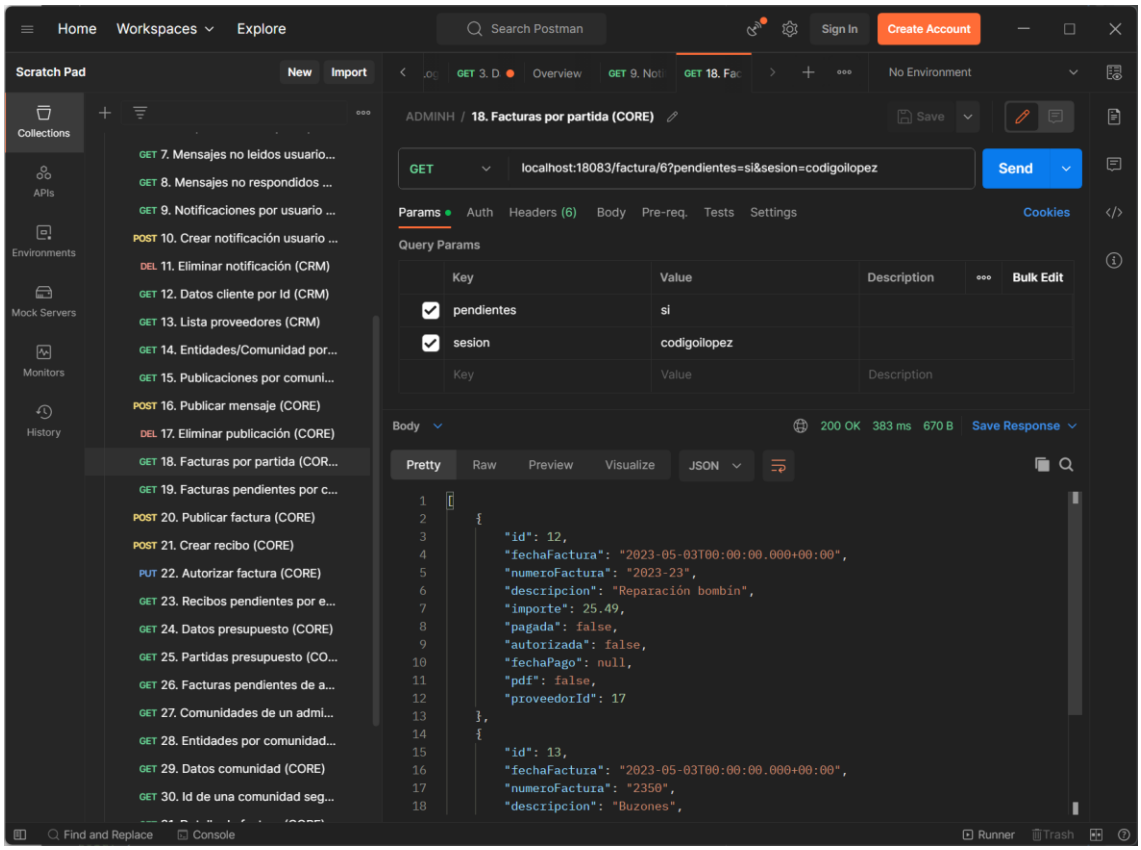

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH / 13. Lista proveedores (CRM)
Save
GET
localhost:18082/proveedores/?sesion=codigojzorita
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigojzorita
Body
200 OK
8 ms
1.97 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
{
"id": 12,
"nombre": "Luz Energy SA",
"nif": "A53355477",
"tipo": "PROVEEDOR",
"direccion": "Av. Cuatro Cantos, 56 Edif. 1",
"cp": "28080",
"municipio": "Madrid",
"provincia": "Madrid",
"formaPago": null,
"iban": "",
"email": "info@luzenergy.com",
"telefono": "911236534"
},
{
"id": 13,
Find and Replace
Console
Runner

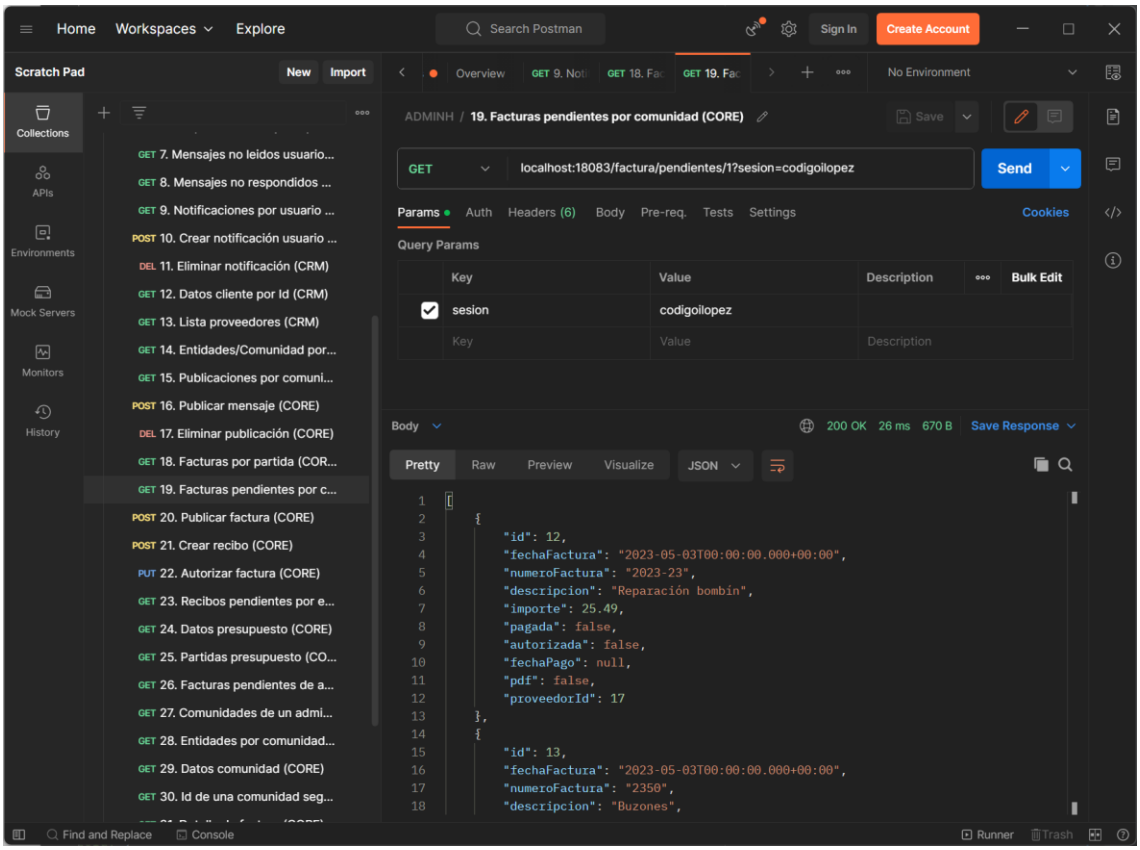

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH / 14. Entidades/Comunidad por cliente (CORE)
Save
GET
http://localhost:18083/comunidades/1?sesion=codigojzorita
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigojzorita
Body
200 OK
56 ms
465 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
{
"id": 3,
"clienteId": 1,
"entidad": {
"id": 3,
"nombre": "1-2",
"coeficiente": 9.4
},
"porcentajePropiedad": 100.0
},
{
"id": 12,
"clienteId": 1,
"entidad": {
"id": 9,
"nombre": "Local",
Find and Replace
Console
Runner

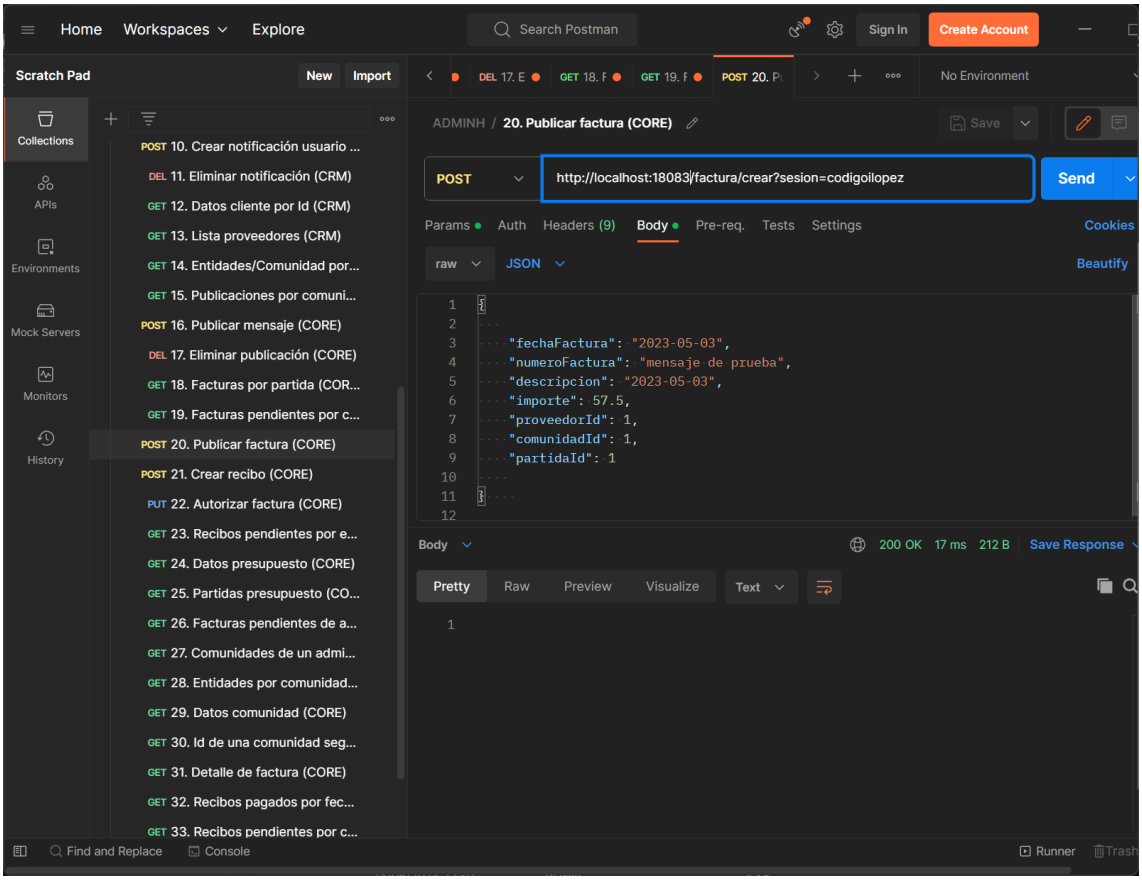

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
ADMINH / 15. Publicaciones por comunidad (CORE)
Save
GET
http://localhost:18083/publicacion/1?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
13 ms
1.1 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"fechaInicio": "2023-04-28T00:00:00.000+00:00",
"fechaFin": "2023-05-15T00:00:00.000+00:00",
"titulo": "Asamblea ordinaria",
"mensaje": "Mediante la presente se convoca a los propietarios a reunión ordinaria a celebrar en el vestíbulo de la\ncomunidad el próximo 15 de abril a las 19:00 horas. El orden del día es el siguiente:\nPresentación estado de cuentas ejercicio 2022.\nRelación de cuotas pendientes a día de la reunión.\nRenovación cargos presidente y secretario-administrador.\nRuego y preguntas.",
"fechaEvento": "2023-08-15T00:00:00.000+00:00"
},
{
"id": 2,
"fechaInicio": "2023-04-28T00:00:00.000+00:00",
Find and Replace
Console
Runner

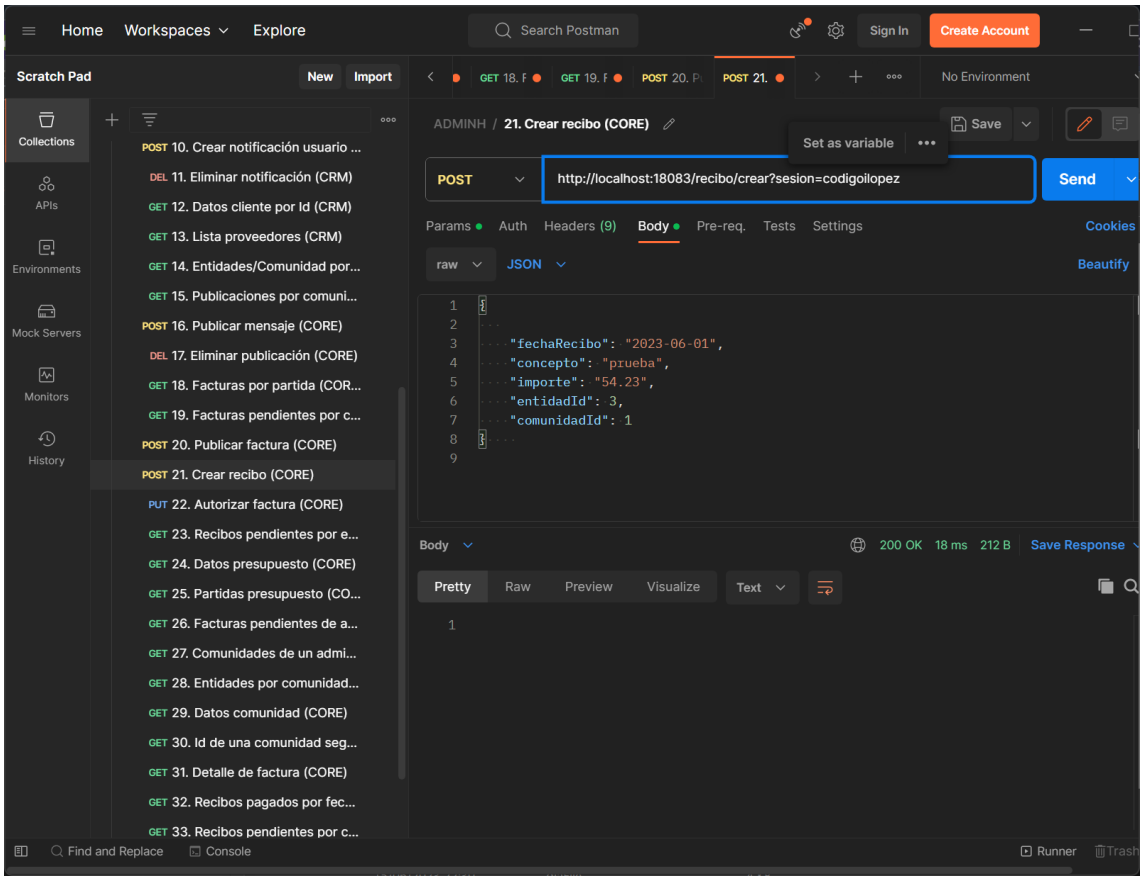

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
ADMINH / 16. Publicar mensaje (CORE)
Save
POST
http://localhost:18083/publicacion/crear?sesion=codigoilopez
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"titulo": "titulo de prueba",
"mensaje": "mensaje de prueba",
"fechaInicio": "2023-05-03",
"fechaFin": "2023-05-15",
"fechaEvento": "2023-05-20",
"comunidadId": 1
}
Body
200 OK
86 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
Runner

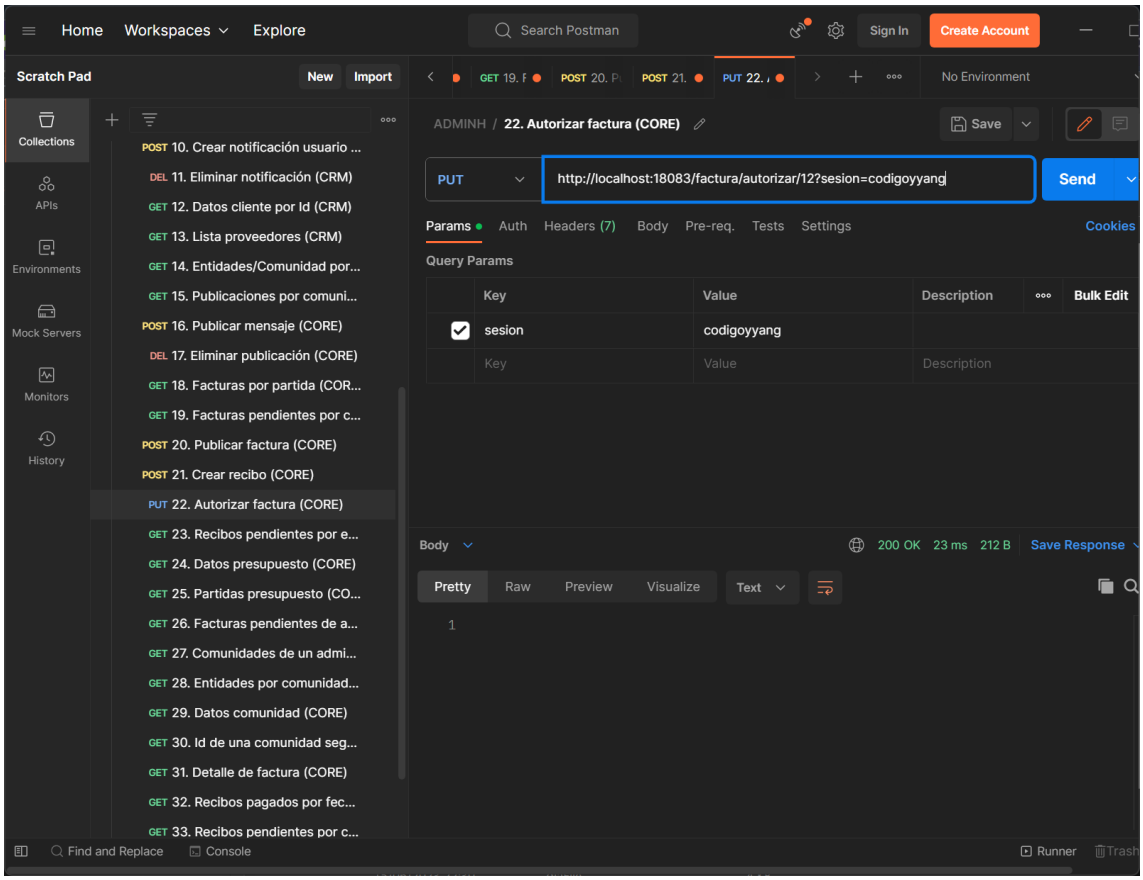

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET GETCLIENTE(CRM)
POST 1. Login
POST 2. Logout
GET 3. Datos cliente por usuario (...
PUT 4. Modificar datos cliente (CR...
POST 5. Enviar mensaje al administr...
PUT 6. Responder mensaje de pro...
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
ADMINH / 17. Eliminar publicación (CORE)
Save
DELETE
http://localhost:18083/publicacion/eliminar/3?sesion=codigoilopez
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
27 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1
Find and Replace
Console
Runner

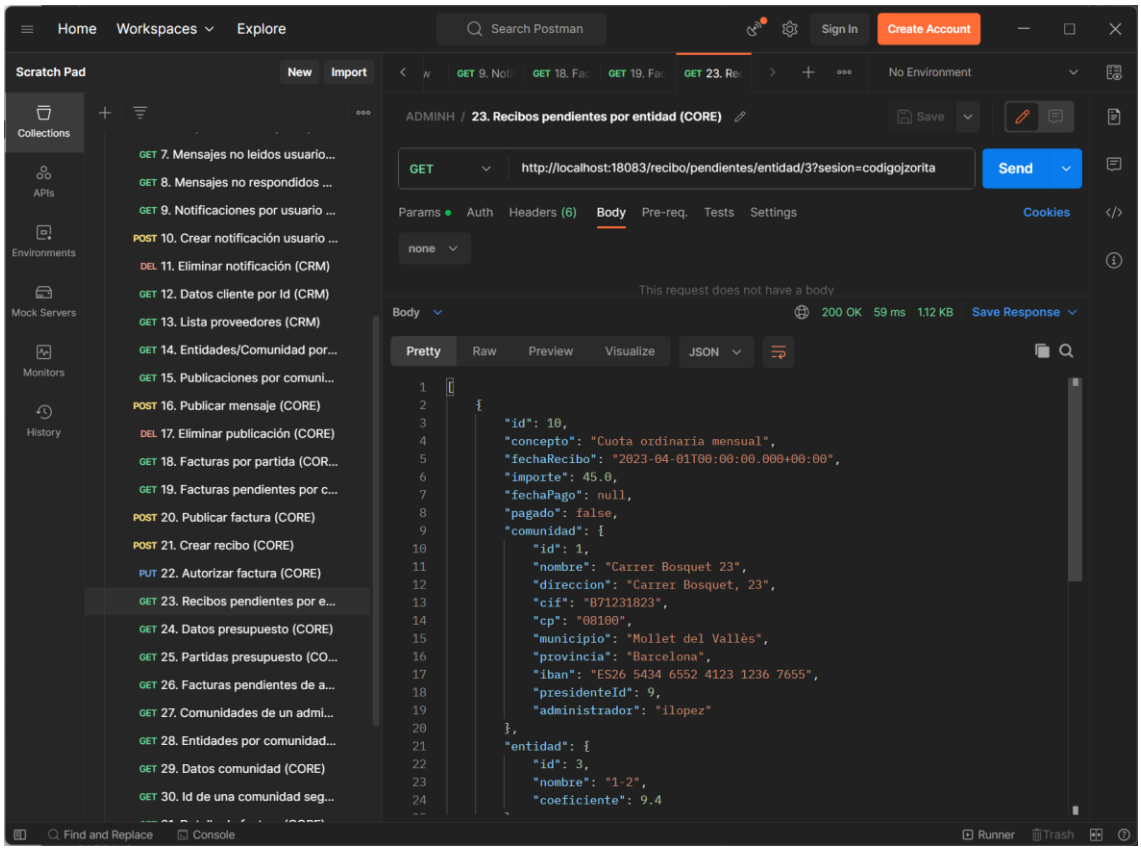

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
ADMINH / 18. Facturas por partida (CORE)
Save
GET
localhost:18083/factura/6?pendientes=si&sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
pendientes
si
sesion
codigoilopez
Body
200 OK
383 ms
670 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 12,
"fechaFactura": "2023-05-03T00:00:00.000+00:00",
"numeroFactura": "2023-23",
"descripcion": "Reparación bombín",
"importe": 25.49,
"pagada": false,
"autorizada": false,
"fechaPago": null,
"pdf": false,
"proveedorId": 17
},
{
"id": 13,
"fechaFactura": "2023-05-03T00:00:00.000+00:00",
"numeroFactura": "2350",
"descripcion": "Buzones",
Find and Replace
Console
Runner

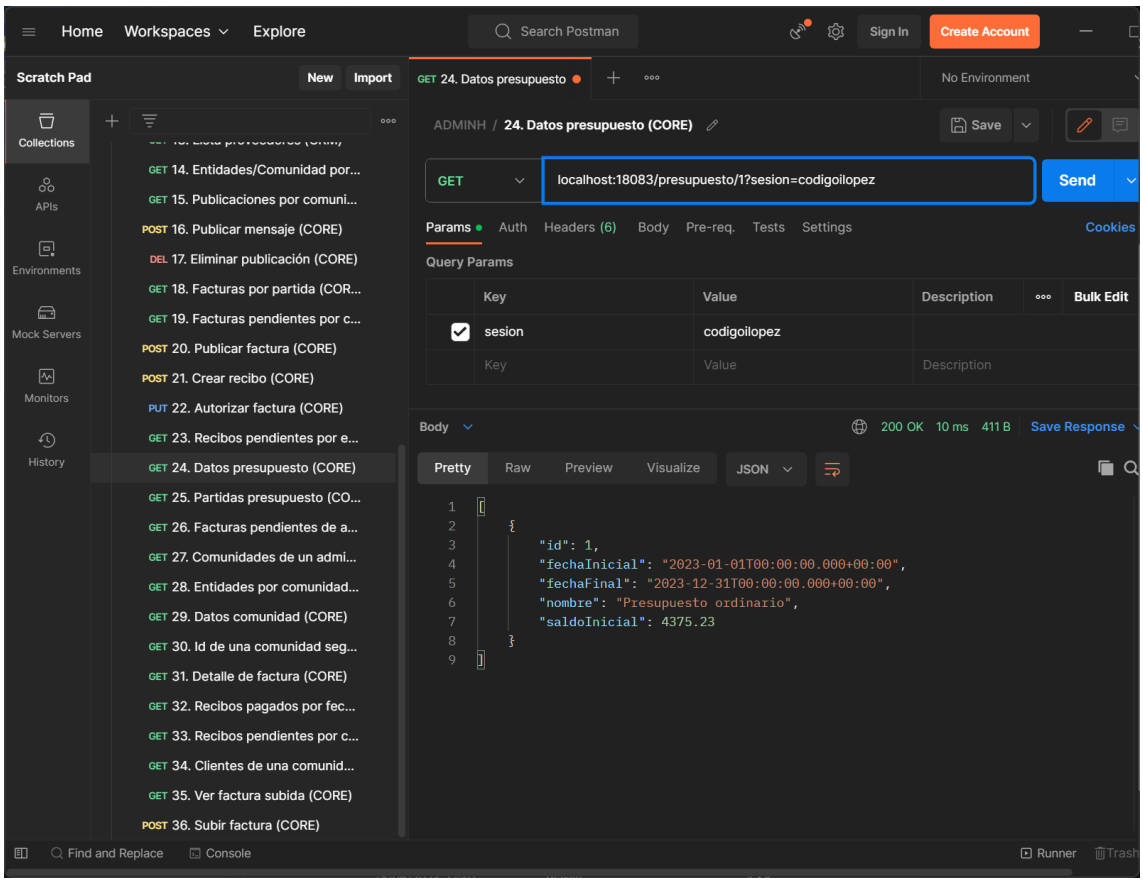

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
No Environment
ADMINH / 19. Facturas pendientes por comunidad (CORE)
Save
GET
localhost:18083/factura/pendientes/1?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
26 ms
670 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 12,
"fechaFactura": "2023-05-03T00:00:00.000+00:00",
"numeroFactura": "2023-23",
"descripcion": "Reparación bombín",
"importe": 25.49,
"pagada": false,
"autorizada": false,
"fechaPago": null,
"pdf": false,
"proveedorId": 17
},
{
"id": 13,
"fechaFactura": "2023-05-03T00:00:00.000+00:00",
"numeroFactura": "2350",
"descripcion": "Buzones",
Find and Replace
Console
Runner
Trash

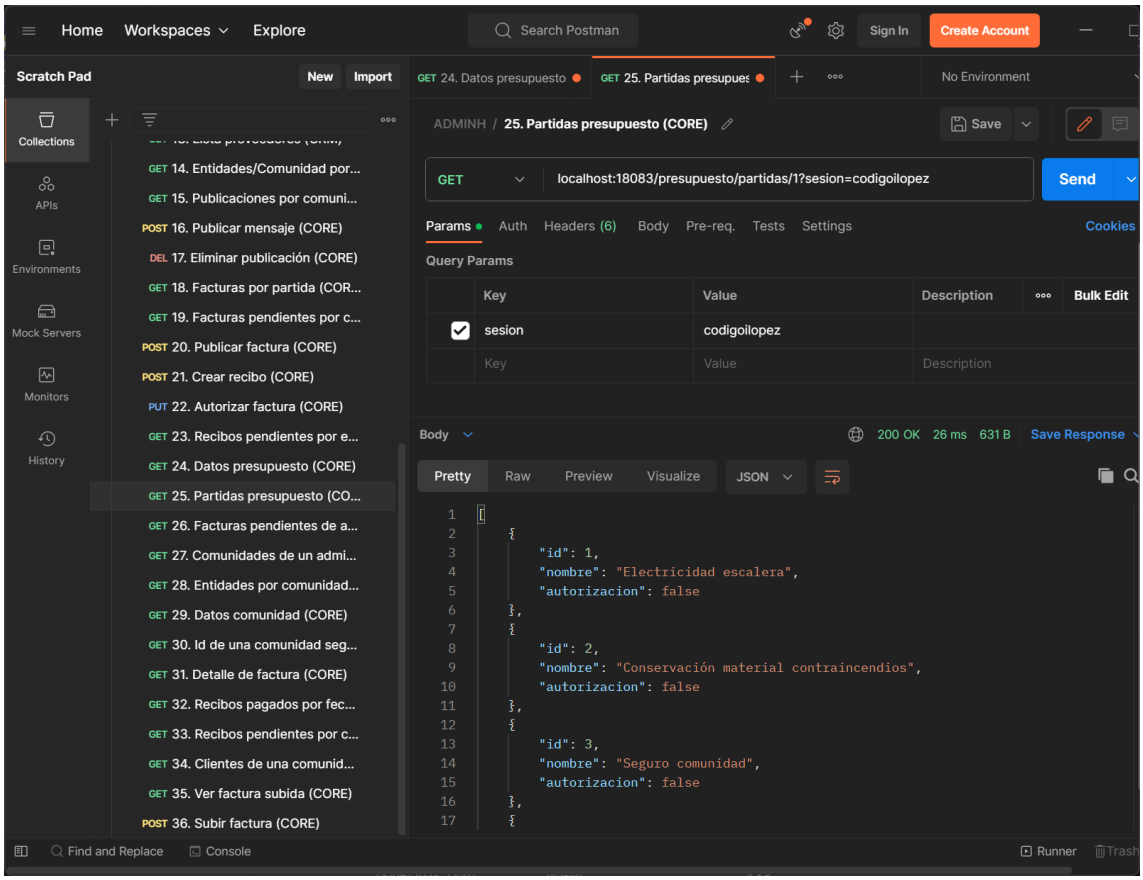

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
Collections
APIs
Environments
Mock Servers
Monitors
History
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
No Environment
ADMINH / 20. Publicar factura (CORE)
Save
POST
http://localhost:18083/factura/crear?sesion=codigoilopez
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"fechaFactura": "2023-05-03",
"numeroFactura": "mensaje de prueba",
"descripcion": "2023-05-03",
"importe": 57.5,
"proveedorId": 1,
"comunidadId": 1,
"partidaId": 1
}
Body
200 OK
17 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
Runner

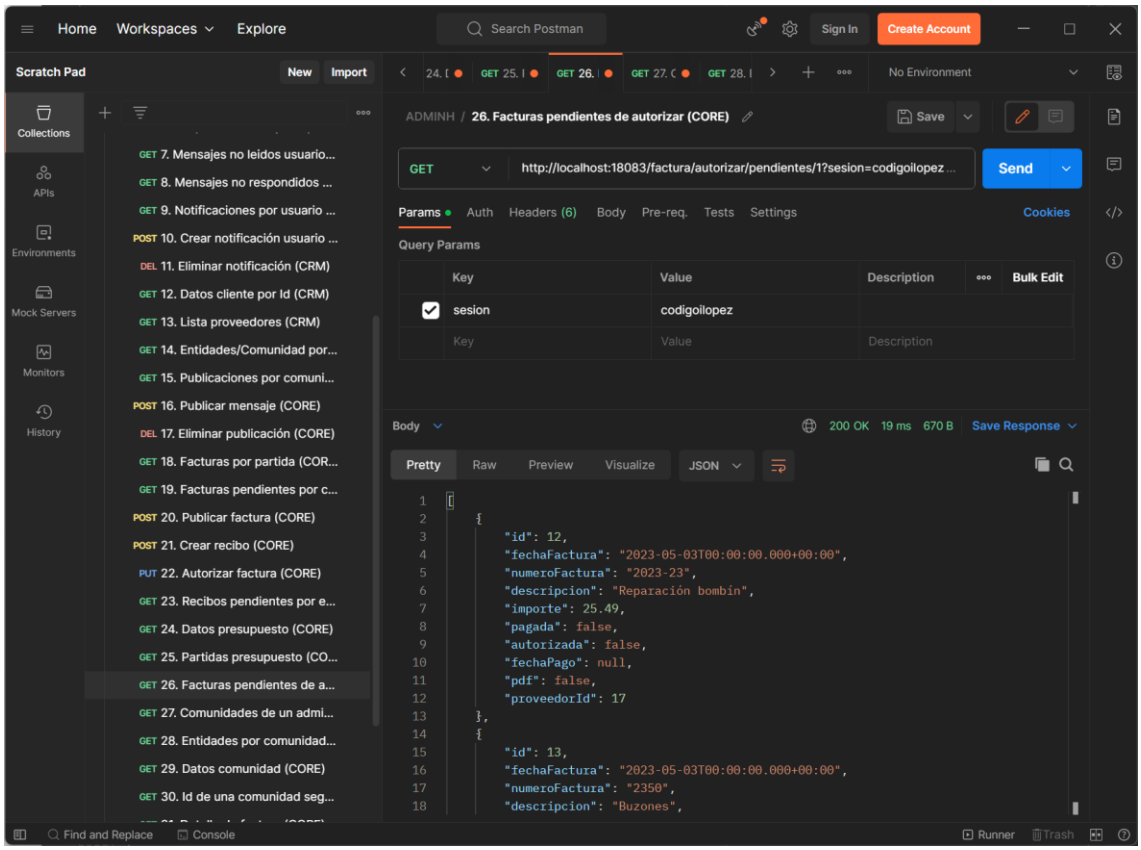

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
ADMINH / 21. Crear recibo (CORE)
Save
Set as variable
POST
http://localhost:18083/recibo/crear?sesion=codigoilopez
Send
Params
Auth
Headers (9)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"fechaRecibo": "2023-06-01",
"concepto": "prueba",
"importe": "54.23",
"entidadId": 3,
"comunidadId": 1
}
Body
200 OK
18 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
Runner

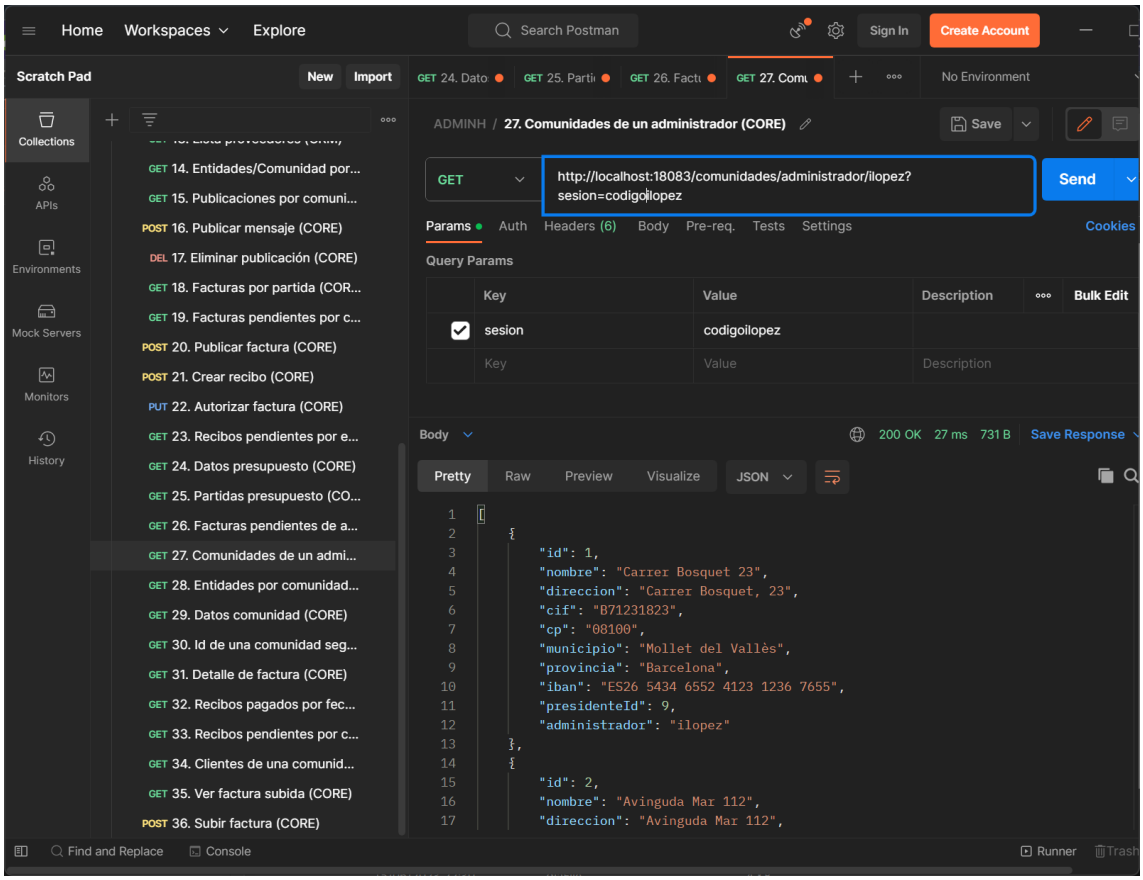

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
ADMINH / 22. Autorizar factura (CORE)
Save
PUT
http://localhost:18083/factura/autorizar/12?sesion=codigoyyang
Send
Params
Auth
Headers (7)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoyyang
Body
200 OK
23 ms
212 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
Runner

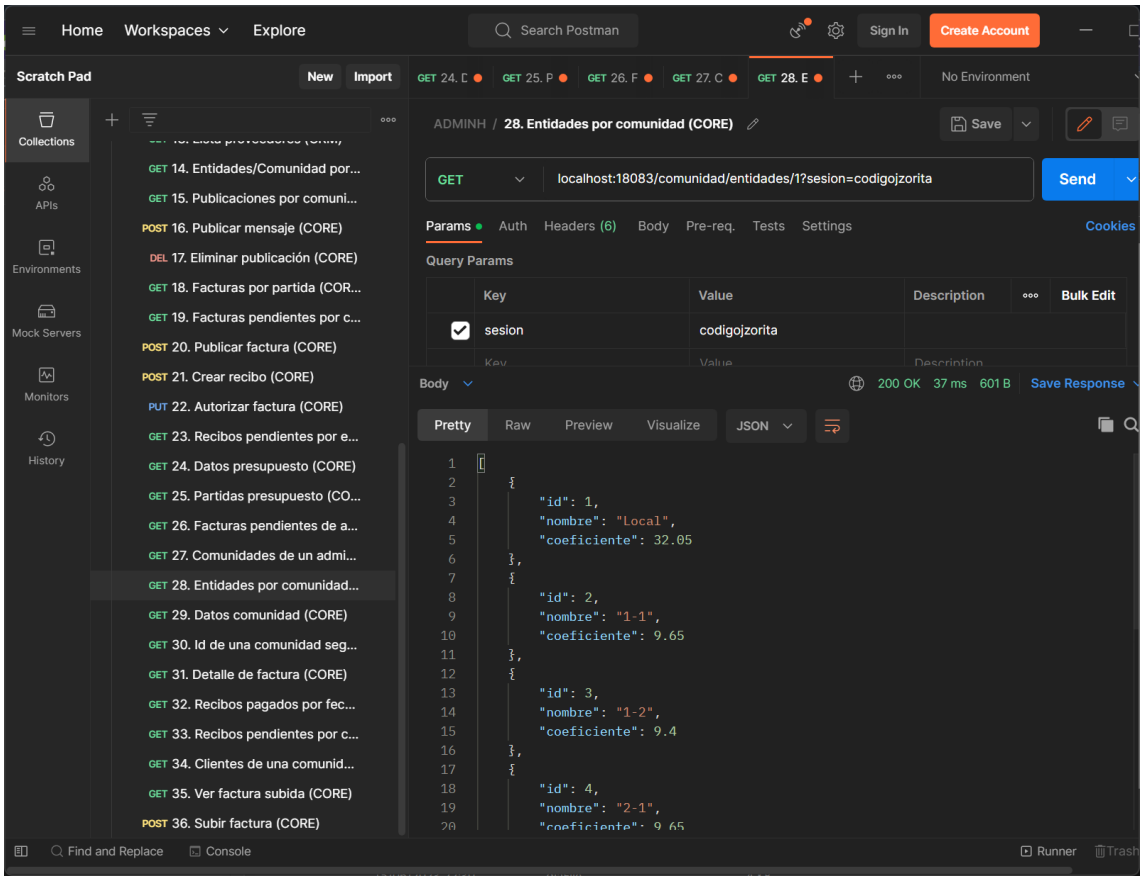

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH / 23. Recibos pendientes por entidad (CORE)
Save
GET
http://localhost:18083/recibo/pendientes/entidad/3?sesion=codigojzorita
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
none
This request does not have a body
Body
200 OK
59 ms
1.12 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
{
"id": 10,
"concepto": "Cuota ordinaria mensual",
"fechaRecibo": "2023-04-01T00:00:00.000+00:00",
"importe": 45.0,
"fechaPago": null,
"pagado": false,
"comunidad": {
"id": 1,
"nombre": "Carrer Bosquet 23",
"direccion": "Carrer Bosquet, 23",
"cif": "B71231823",
"cp": "08100",
"municipio": "Mollet del Vallès",
"provincia": "Barcelona",
"iban": "ES26 5434 6552 4123 1236 7655",
"presidenteId": 9,
"administrador": "ilopez"
},
"entidad": {
"id": 3,
"nombre": "1-2",
"coeficiente": 9.4
Find and Replace
Console
Runner
Trash

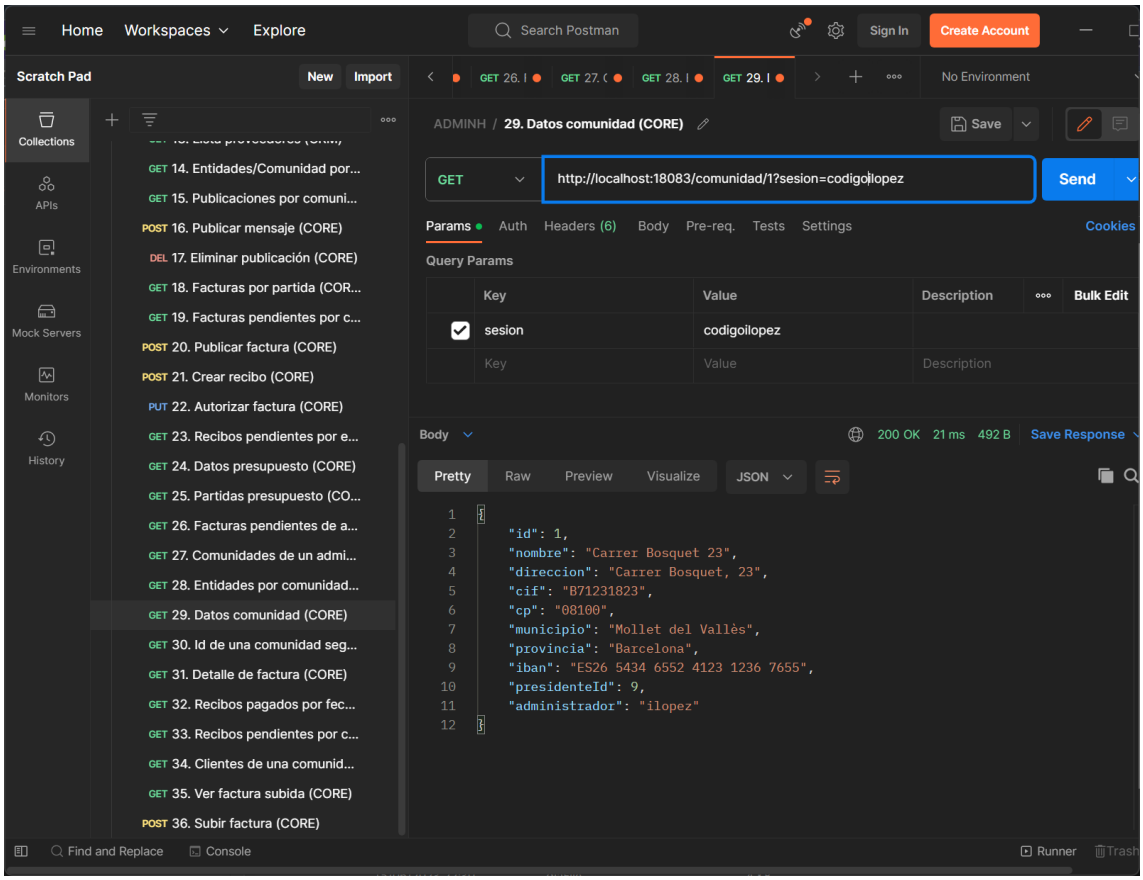

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
GET 24. Datos presupuesto
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH / 24. Datos presupuesto (CORE)
Save
GET
localhost:18083/presupuesto/1?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Key
Value
Description
Body
200 OK
10 ms
411 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
POST 36. Subir factura (CORE)
{
"id": 1,
"fechaInicial": "2023-01-01T00:00:00.000+00:00",
"fechaFinal": "2023-12-31T00:00:00.000+00:00",
"nombre": "Presupuesto ordinario",
"saldoInicial": 4375.23
}
Find and Replace
Console
Runner

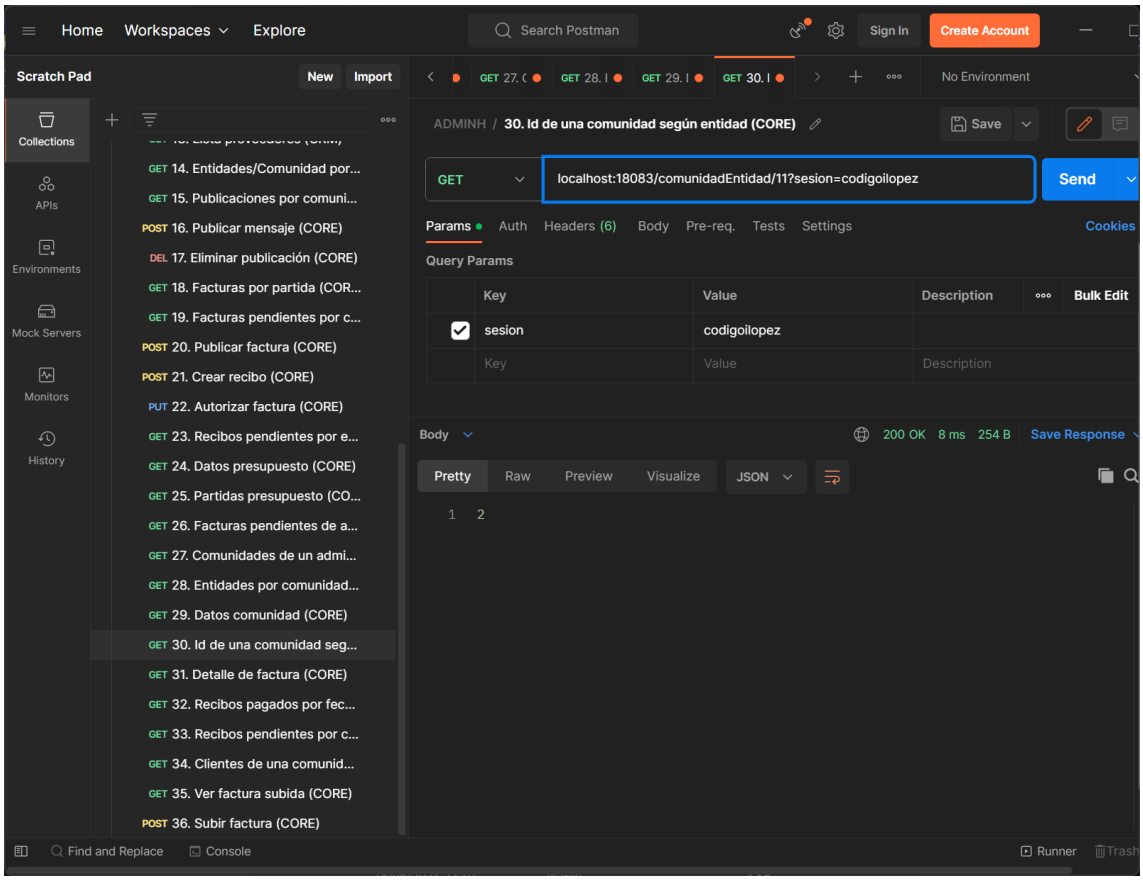

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
GET 24. Datos presupuesto
GET 25. Partidas presupues
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH / 25. Partidas presupuesto (CORE)
Save
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
POST 36. Subir factura (CORE)
GET
localhost:18083/presupuesto/partidas/1?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
26 ms
631 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"nombre": "Electricidad escalera",
"autorizacion": false
},
{
"id": 2,
"nombre": "Conservación material contraincendios",
"autorizacion": false
},
{
"id": 3,
"nombre": "Seguro comunidad",
"autorizacion": false
},
{
Find and Replace
Console
Runner
Trash

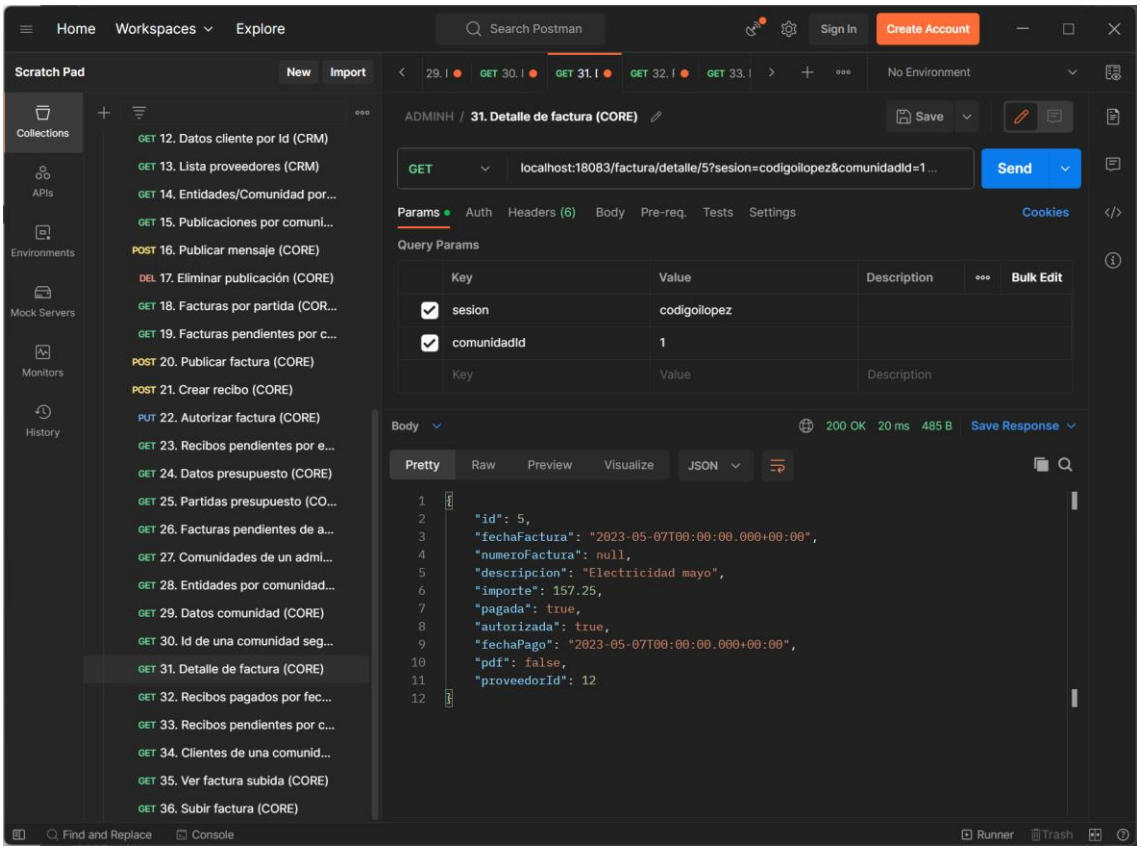

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
ADMINH / 26. Facturas pendientes de autorizar (CORE)
Save
GET 7. Mensajes no leidos usuario...
GET 8. Mensajes no respondidos ...
GET 9. Notificaciones por usuario ...
POST 10. Crear notificación usuario ...
DEL 11. Eliminar notificación (CRM)
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET
http://localhost:18083/factura/autorizar/pendientes/1?sesion=codigoilopez ...
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
19 ms
670 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 12,
"fechaFactura": "2023-05-03T00:00:00.000+00:00",
"numeroFactura": "2023-23",
"descripcion": "Reparación bombín",
"importe": 25.49,
"pagada": false,
"autorizada": false,
"fechaPago": null,
"pdf": false,
"proveedorId": 17
},
{
"id": 13,
"fechaFactura": "2023-05-03T00:00:00.000+00:00",
"numeroFactura": "2350",
"descripcion": "Buzones",
Find and Replace
Console
Runner
Trash

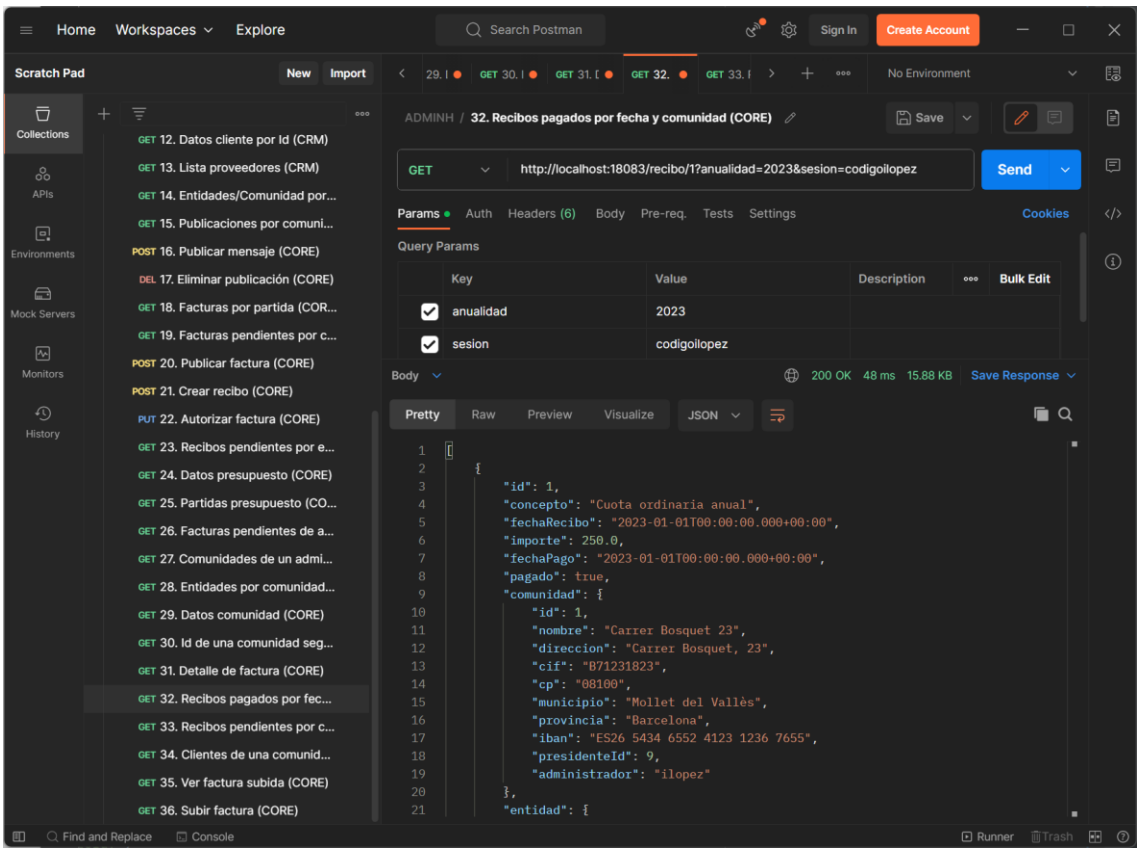

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
POST 36. Subir factura (CORE)
ADMINH / 27. Comunidades de un administrador (CORE)
Save
GET
http://localhost:18083/comunidades/administrador/ilopez?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
27 ms
731 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"nombre": "Carrer Bosquet 23",
"direccion": "Carrer Bosquet, 23",
"cif": "B71231823",
"cp": "08100",
"municipio": "Mollet del Vallès",
"provincia": "Barcelona",
"iban": "ES26 5434 6552 4123 1236 7655",
"presidenteId": 9,
"administrador": "ilopez"
},
{
"id": 2,
"nombre": "Avinguda Mar 112",
"direccion": "Avinguda Mar 112",
Find and Replace
Console
Runner

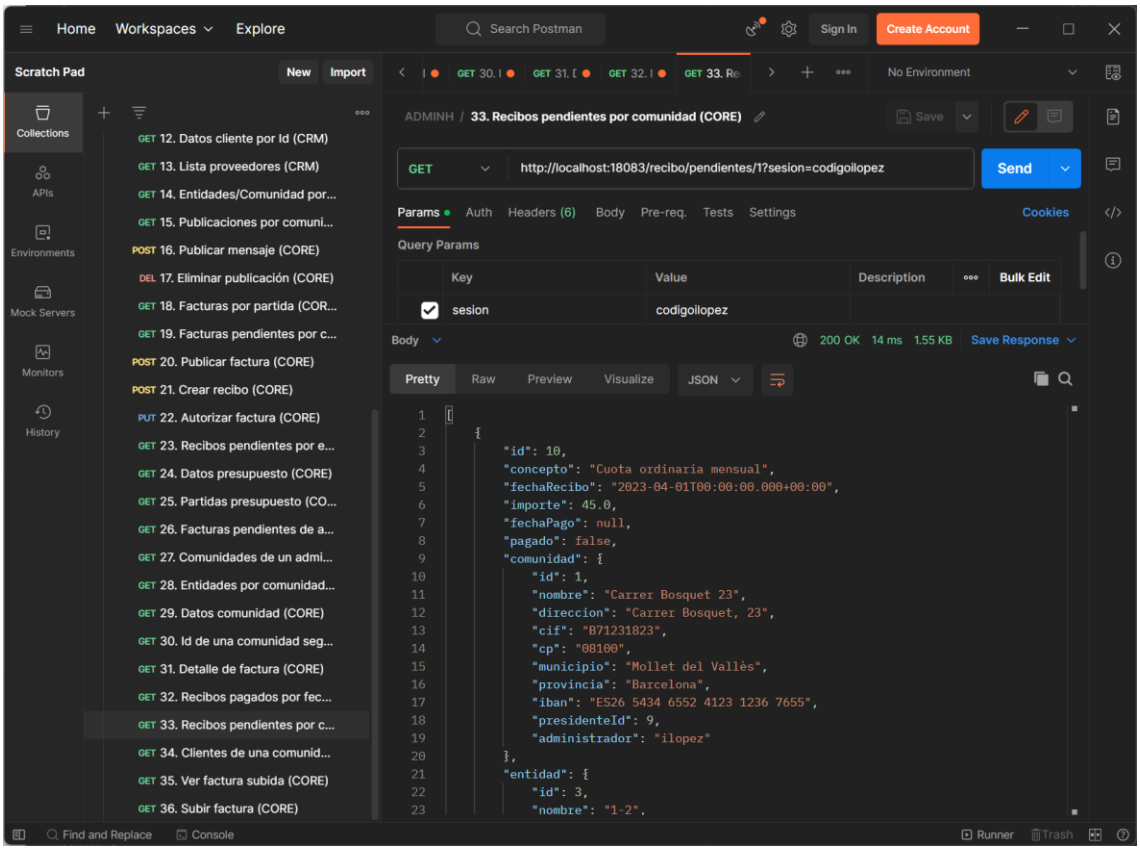

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
POST 36. Subir factura (CORE)
ADMINH / 28. Entidades por comunidad (CORE)
Save
GET
localhost:18083/comunidad/entidades/1?sesion=codigojzorita
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigojzorita
Body
200 OK
37 ms
601 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"nombre": "Local",
"coeficiente": 32.05
},
{
"id": 2,
"nombre": "1-1",
"coeficiente": 9.65
},
{
"id": 3,
"nombre": "1-2",
"coeficiente": 9.4
},
{
"id": 4,
"nombre": "2-1",
Find and Replace
Console
Runner

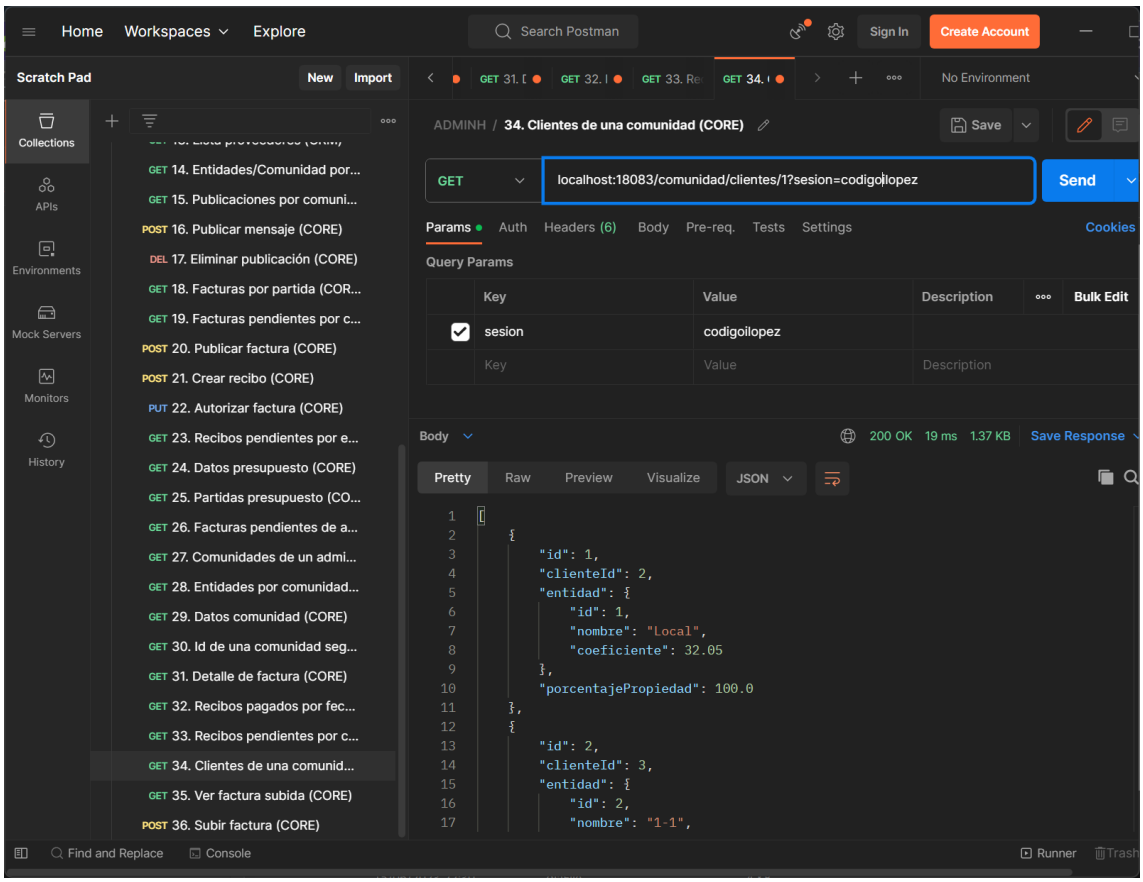

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
POST 36. Subir factura (CORE)
ADMINH / 29. Datos comunidad (CORE)
Save
GET
http://localhost:18083/comunidad/1?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
21 ms
492 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"nombre": "Carrer Bosquet 23",
"direccion": "Carrer Bosquet, 23",
"cif": "B71231823",
"cp": "08100",
"municipio": "Mollet del Vallès",
"provincia": "Barcelona",
"iban": "ES26 5434 6552 4123 1236 7655",
"presidenteId": 9,
"administrador": "ilopez"
}
Find and Replace
Console
Runner

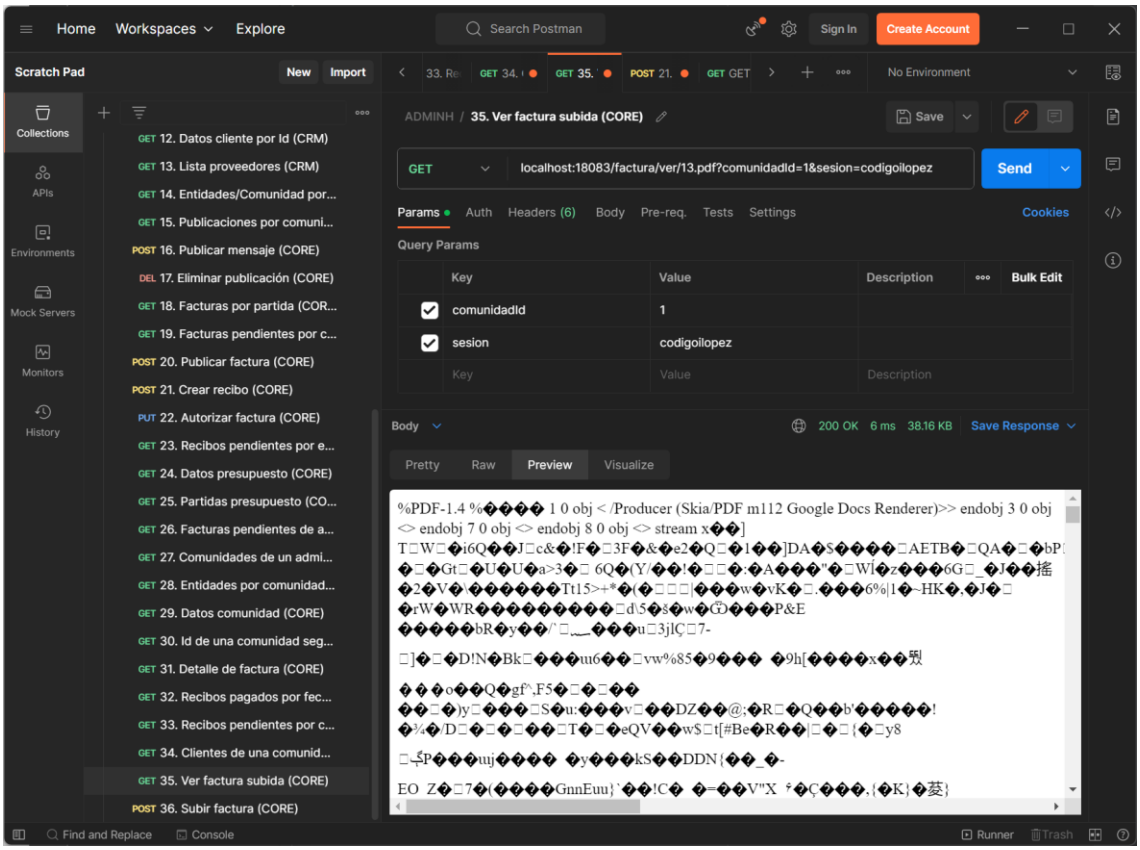

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
POST 36. Subir factura (CORE)
ADMINH / 30. Id de una comunidad según entidad (CORE)
Save
GET
localhost:18083/comunidadEntidad/11?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
8 ms
254 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 2
Find and Replace
Console
Runner

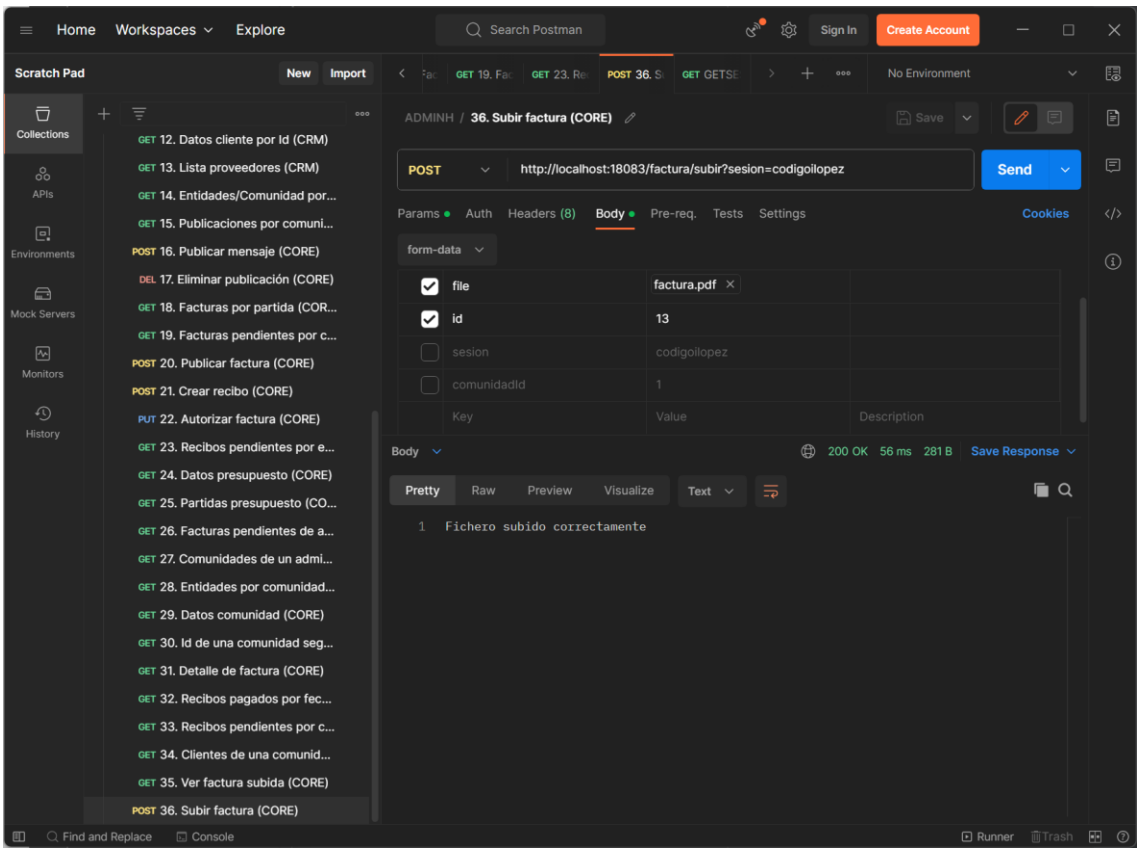

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
GET 36. Subir factura (CORE)
ADMINH / 31. Detalle de factura (CORE)
Save
GET
localhost:18083/factura/detalle/5?sesion=codigoilopez&comunidadId=1 ...
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
comunidadId
1
Body
200 OK
20 ms
485 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 5,
"fechaFactura": "2023-05-07T00:00:00.000+00:00",
"numeroFactura": null,
"descripcion": "Electricidad mayo",
"importe": 157.25,
"pagada": true,
"autorizada": true,
"fechaPago": "2023-05-07T00:00:00.000+00:00",
"pdf": false,
"proveedorId": 12
}
Find and Replace
Console
Runner
Trash


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
GET 36. Subir factura (CORE)
ADMINH / 32. Recibos pagados por fecha y comunidad (CORE)
Save
GET
http://localhost:18083/recibo/1?anualidad=2023&sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
anualidad
2023
sesion
codigoilopez
Body
200 OK
48 ms
15.88 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"concepto": "Cuota ordinaria anual",
"fechaRecibo": "2023-01-01T00:00:00.000+00:00",
"importe": 250.0,
"fechaPago": "2023-01-01T00:00:00.000+00:00",
"pagado": true,
"comunidad": {
"id": 1,
"nombre": "Carrer Bosquet 23",
"direccion": "Carrer Bosquet, 23",
"cif": "B71231823",
"cp": "08100",
"municipio": "Mollet del Vallès",
"provincia": "Barcelona",
"iban": "ES26 5434 6552 4123 1236 7655",
"presidenteId": 9,
"administrador": "ilopez"
},
"entidad": {
Find and Replace
Console
Runner
Trash


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
GET 36. Subir factura (CORE)
ADMINH / 33. Recibos pendientes por comunidad (CORE)
Save
GET
http://localhost:18083/recibo/pendientes/1?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
14 ms
1.55 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 10,
"concepto": "Cuota ordinaria mensual",
"fechaRecibo": "2023-04-01T00:00:00.000+00:00",
"importe": 45.0,
"fechaPago": null,
"pagado": false,
"comunidad": {
"id": 1,
"nombre": "Carrer Bosquet 23",
"direccion": "Carrer Bosquet, 23",
"cif": "B71231823",
"cp": "08100",
"municipio": "Mollet del Vallès",
"provincia": "Barcelona",
"iban": "ES26 5434 6552 4123 1236 7655",
"presidenteId": 9,
"administrador": "ilopez"
},
"entidad": {
"id": 3,
"nombre": "1-2",
Find and Replace
Console
Runner
Trash


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
POST 36. Subir factura (CORE)
ADMINH / 34. Clientes de una comunidad (CORE)
Save
GET
localhost:18083/comunidad/clientes/1?sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sesion
codigoilopez
Body
200 OK
19 ms
1.37 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"clienteId": 2,
"entidad": {
"id": 1,
"nombre": "Local",
"coeficiente": 32.05
},
"porcentajePropiedad": 100.0
},
{
"id": 2,
"clienteId": 3,
"entidad": {
"id": 2,
"nombre": "1-1",
Find and Replace
Console
Runner


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
POST 36. Subir factura (CORE)
ADMINH / 35. Ver factura subida (CORE)
Save
GET
localhost:18083/factura/ver/13.pdf?comunidadId=1&sesion=codigoilopez
Send
Params
Auth
Headers (6)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
comunidadId
1
sesion
codigoilopez
Body
200 OK
6 ms
38.16 KB
Save Response
Pretty
Raw
Preview
Visualize
%PDF-1.4 %
1 0 obj < /Producer (Skia/PDF m112 Google Docs Renderer)>> endobj 3 0 obj
<> endobj 7 0 obj <> endobj 8 0 obj <> stream x
T W i6Q J c& !F 3F & e2 Q 1]DA S AETB QA bP
Gt U U a>3 6Q (Y ! x
Tt15>+* {] w vK . 6%|1 ~HK , J
rW WR d\5 s w D P&E
bR y / u 3jlÇ 7-
]
D!N Bk u6 vw%85 9 9h[x
o Q gf^,F5
jy S u: v DZ @; R Q b'
/D T eQV wS t[#Be R | y8
P uj kS DDN{ _
EO Z 7{ GnnEuu}` !C = V"X ç , {K}
Find and Replace
Console
Runner
Trash


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
Scratch Pad
New
Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
GET 12. Datos cliente por Id (CRM)
GET 13. Lista proveedores (CRM)
GET 14. Entidades/Comunidad por...
GET 15. Publicaciones por comuni...
POST 16. Publicar mensaje (CORE)
DEL 17. Eliminar publicación (CORE)
GET 18. Facturas por partida (COR...
GET 19. Facturas pendientes por c...
POST 20. Publicar factura (CORE)
POST 21. Crear recibo (CORE)
PUT 22. Autorizar factura (CORE)
GET 23. Recibos pendientes por e...
GET 24. Datos presupuesto (CORE)
GET 25. Partidas presupuesto (CO...
GET 26. Facturas pendientes de a...
GET 27. Comunidades de un admi...
GET 28. Entidades por comunidad...
GET 29. Datos comunidad (CORE)
GET 30. Id de una comunidad seg...
GET 31. Detalle de factura (CORE)
GET 32. Recibos pagados por fec...
GET 33. Recibos pendientes por c...
GET 34. Clientes de una comunid...
GET 35. Ver factura subida (CORE)
POST 36. Subir factura (CORE)
ADMINH / 36. Subir factura (CORE)
Save
POST
http://localhost:18083/factura/subir?sesion=codigoilopez
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
form-data
file
factura.pdf
id
13
sesion
codigoilopez
comunidadId
1
Key
Value
Description
Body
200 OK
56 ms
281 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1 Fichero subido correctamente
Find and Replace
Console
Runner
Trash