



**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERÍA DE COMPUTADORES**

**Curso Académico 2022/2023**

**Trabajo Fin de Grado**

**App4Auctions – Subastas y pujas online**



**Autor:** Álvaro Martínez Quiroga

**Director:** Miguel Ángel Rodríguez García

# Resumen

El presente Trabajo de Fin de Grado desarrolla un sistema de subastas y pujas online a través de un navegador web, con actualizaciones a tiempo real. Los servicios cuentan con una capa de virtualización para su implementación. El sistema distribuido se compone de una serie de servicios, estos son: un sistema gestor de información, un servicio para gestionar las operaciones y lógica de negocio y finalmente un proceso capaz de servir la información de manera visual, todos ellos conectados en un sistema distribuido. También, se desarrolla una parte de investigación, comparando diferentes métodos para el procesado del lenguaje natural y las búsquedas de texto a través de diferentes técnicas de Machine Learning.

**Palabras clave:** sistema de gestión de subastas y pujas, tareas programadas, sistema de clasificación, contenedores Docker, Machine Learning.

# Índice del documento

## Contenido

1. Introducción .....	5
2. Estado del arte .....	6
3. Objetivos.....	7
4. Metodología de software empleada.....	8
5. Descripción informática.....	10
5.1 Diagrama de Gantt.....	10
5.2 Requisitos.....	12
5.3 Diseño .....	14
5.4 Implementación .....	15
5.4.1 Integración continua.....	16
5.4.2 Sistemas de gestión de memoria.....	19
5.4.3 Diseño de un esquema de almacenamiento de datos .....	22
5.4.4 Implementación de servicio para servir y procesar los datos guardados ...	25
5.4.5 Implementación de servicio para procesar tareas programadas .....	32
5.4.6 Implementación de servicio para servir datos en tiempo real .....	38
5.4.7 Implementación de envío de correos electrónicos.....	43
5.4.8 implementación de técnicas de búsqueda y filtrado de subastas .....	45
5.4.9 Desarrollo de vistas de la aplicación web .....	50
5.4.10 Arquitectura final cliente-servidor.....	53
5.5 Pruebas del sistema.....	53
5.5.1 Recopilación de datos .....	53
5.5.2 Resultados .....	55
6. Conclusiones generales.....	58
7. Referencias.....	60
8. Anexos .....	62
8.1 Manual de instalación .....	62
8.2 Manual de usuario .....	63
8.2.1 Creación de nuevo usuario .....	63
8.2.2 Creación de una subasta .....	66
8.2.3 Exploración de subastas .....	67
8.2.4 Cambio de datos de usuario.....	69

## Tabla de ilustraciones

### Contenido

Ilustración 1: metodología de cascada.....	10
Ilustración 2: diagrama de Gantt.....	11
Ilustración 3: arquitectura cliente-servidor.....	12
Ilustración 4: arquitectura del sistema.....	14
Ilustración 5: token de acceso a Docker Hub .....	17
Ilustración 6: imágenes Docker publicadas en Docker Hub.....	18
Ilustración 7: variables de entorno en Gitlab CI/CD.....	19
Ilustración 8: popularidad de bases de datos (2022) .....	22
Ilustración 9: modelo entidad relación .....	24
Ilustración 10: especificación Open API.....	28
Ilustración 11: endpoint de ejemplo.....	29
Ilustración 12: esquemas del API.....	30
Ilustración 13: respuesta del endpoint de salud del servicio backend.....	30
Ilustración 14: flujo de eventos al comenzar una subasta .....	35
Ilustración 15: flujo de eventos al finalizar una subasta.....	36
Ilustración 16: panel de control Flower.....	37
Ilustración 17: vista de tareas en Flower .....	37
Ilustración 18: ciclo de vida del protocolo WebSocket.....	39
Ilustración 19: websocket del mercado .....	42
Ilustración 20: websocket de subasta .....	43
Ilustración 21: fórmula TF-IDF .....	46
Ilustración 22: ejemplo de márgenes en SVM.....	48
Ilustración 23: arquitectura completa del sistema.....	53
Ilustración 24: creación de usuario.....	64
Ilustración 25: página de inicio.....	65
Ilustración 26: datos de usuario .....	65
Ilustración 27: página nueva subasta.....	66
Ilustración 28: lista de subastas de un usuario .....	67
Ilustración 29: mercado .....	68
Ilustración 30: listado de subastas con filtro.....	69

## Tabla de figuras

### Contenido

Tabla 1: requisitos funcionales .....	13
Tabla 2: requisitos no funcionales.....	13
Tabla 3: ejemplo de mediciones de algoritmos.....	56
Tabla 4: resultados de búsquedas finales.....	57

## 1. Introducción

Las subastas han existido antes de la era online actual, como un medio de intercambio de objetos por una cantidad de dinero al mejor postor. El término ‘subasta’ apareció por primera vez derivado del latín, de las palabras sub (bajo) y hasta (lanza). Este término se asemeja a una expresión que quiere decir “vender bienes a viva voz”. En esta época durante los eventos de las subastas se realizaban ofertas sobre un bien expuesto, las cuales se iban incrementando, y cuya venta se la asociaba el mejor postor [1].

Gracias al desarrollo digital y los sistemas online se proporciona un entorno que puede llegar a una serie mayor de posibles potenciales postores, sin necesidad de acudir personalmente al evento donde la subasta está teniendo lugar. También, las búsquedas en las subastas hacen más sencillo que los compradores encuentren de manera más sencilla lo que buscan para realizar ofertas. Estas herramientas de búsqueda benefician la visibilidad de las subastas a lo largo de Internet [2].

Una subasta puede ser de varios tipos, llegándose a categorizar por el tipo de compra o venta, la cantidad de bienes subastados, las reglas de finalización o incluso por las reglas de apuestas. Las subastas de carácter online tienden a orientarse a aquellas denominadas ‘subastas ascendentes’ o ‘subastas inglesas’. Como el propio nombre indica, el vendedor puede establecer un precio inicial o en ocasiones se comienza desde cero y los posibles compradores pujan en la subasta incrementando el precio hasta que se llega a la puja final, donde ninguna otra persona sigue pujando en la subasta o finaliza el tiempo para realizar pujas y se cierra la subasta, siendo el creador de la mayor puja el ganador de la subasta. Cabe recalcar que la subasta aparece en aquellos entornos que presentan dos características para su desarrollo, estas son: la existencia de personas que puedan aportar un número de posibles compradores y vendedores y la existencia de una divisa que sea capaz de dar valor a las pujas realizadas [18].

Este Trabajo de Fin de Grado intenta proporcionar una solución diferente a la manera en la que se subastan y venden artículos de manera online. Se trata de una plataforma web donde cada persona pueda mostrar al resto del mundo sus artículos únicos, además de realizar pujas en otros exclusivos productos expuestos en la web. Se presenta una interfaz web interactiva para los usuarios del sistema y un rápido servicio para atender las diferentes peticiones provenientes de dichos usuarios. La web tiene la misión de proveer con una interfaz inmersiva al usuario final con estilos web modernos, colores

claros y actualizaciones a tiempo real, haciendo posible la exploración de un largo número de subastas e incluso llegando a compras y pujas competitivas por los usuarios. El problema abordado con este Trabajo de Fin de Grado se centra en la creación de un sistema distribuido con diferentes componentes que interactúan entre sí para atender a diferentes usuarios que se conectan a través de Internet y para gestionar los procesos en marcha de la página web.

## 2. Estado del arte

Gracias al desarrollo de las tecnologías las páginas de subastas online han experimentado notables cambios en los últimos años, aprovechando también la creciente demanda del comercio electrónico por la facilidad que permite a las personas para vender sus artículos desde la comodidad de sus hogares.

En la actualidad existen una serie de plataformas online con este fin, algunas de ellas muy conocidas mundialmente. Algunos ejemplos de estas páginas web son las siguientes:

- *eBay*: plataforma multinacional orientada al comercio electrónico. Opera como intermediario para realizar las diferentes transacciones solicitadas por los clientes de la página web. Existen dos tipos de operaciones en esta web, las ventas a un precio fijo o las ventas en forma de subastas donde se incrementa el precio a medida que las diferentes personas crean o actualizan sus pujas. La plataforma cobra un impuesto cuando se publica un producto y en el momento de su venta. Cabe mencionar que el porcentaje puede variar dependiendo de muchos factores, como, por ejemplo, el tipo de cuenta del vendedor, el tipo de proceso, a precio fijo o variable, o incluso la categoría del objeto a vender.
- *Catawiki*: similar al sistema anterior, esta web se centra en la venta de artículos raros, únicos y de alto valor. Cabe mencionar que el sistema también carga a los usuarios que venden sus objetos un porcentaje en el precio final de las ventas que se realizan en la plataforma, cuanto más alto sea el precio, más alta será dicha tasa. Puede variar desde el 12,5% del precio final hasta el 15%.

Estas plataformas son de carácter profesional, también podemos relacionar el presente Trabajo de Fin de Grado con otros artículos científicos publicados por otros autores. Aquí podemos destacar diferentes proyectos: aquellos destinados al desarrollo de qué

es una subasta y sus tipos, otros dedicados a aplicaciones web orientadas a las subastas online y, por último, trabajos encaminados a la comparativa de métodos de búsqueda y/o filtrado de textos en el lenguaje natural.

Para el primer tipo, ya ha sido mencionado el trabajo por el autor Sergio Andreu, con referencia número [4] dentro de esta memoria, donde explica qué son las subastas y cuáles son sus diferentes tipos. En segundo lugar, el autor Carlos Álvarez, con referencia número [1] dentro de esta memoria, presenta una aplicación web orientada a una plataforma web de subastas con el uso de Smart Contracts. Por último, existen multitud de trabajos dedicados al procesamiento del lenguaje natural y filtrado y/o clasificación de textos. Quizás uno de los puntos más desafiantes a implementar por la poca experiencia en este ámbito. Cabe destacar el trabajo realizado por el autor Juan José Martín, con referencia número [19], para comprender mejor como aplicar la técnica de los *Support Vector Machines*.

### 3. Objetivos

El objetivo principal del sistema es desarrollar los diferentes servicios necesarios para atender peticiones de múltiples usuarios, servir los cambios en los datos guardados de una manera inmersiva en la página web y presentar de manera transparente las operaciones realizadas por los consumidores del servicio. El objetivo principal es dividido en diferentes subobjetivos:

- Obj1. Realizar un estudio del estado del arte actual, en referencia a aplicaciones similares desarrolladas por otros autores, de carácter profesional y académico.
- Obj2. Crear un sistema de manejo de usuarios y del inventario de operaciones existentes para gestionar la creación y actualización de subastas y pujas.
- Obj3. Implementar un sistema de pujas transparente, donde todo usuario sea capaz de visualizar una tabla histórica de las pujas en cada subasta.
- Obj4. Diseñar e implementar un módulo de búsqueda y comparar diferentes técnicas.

- Obj5. Desarrollar diferentes vistas, fáciles de usar para los usuarios, con información relevante sobre los procesos de la web y con actualizaciones a tiempo real.

## 4. Metodología de software empleada

La metodología de software hace referencia a aquellas técnicas o prácticas que se aplican a los sistemas de software y a su desarrollo. Su objetivo principal es asegurar que los proyectos se estiman, abordan y finalizan de una manera eficiente, a tiempo de entrega y dentro de un presupuesto estimado. Es una tarea que no se debe de subestimar, ya que, en muchos proyectos del ámbito profesional existe una persona o varias exclusivamente dedicadas a la gestión del propio proyecto, aplicando alguna de las metodologías de gestión de proyectos de software existentes. Estas metodologías siguen un proceso o una serie de actividades para el desarrollo y logro de los objetivos, por lo general son los siguientes: recopilación de requisitos del proyecto, diseños y pruebas de conceptos, implementación en código, pruebas (ya sean de integración, unitarias...), despliegue y mantenimiento. Cabe mencionar que no todas las metodologías del software han de seguir esta serie de actividades de una manera fija. Existen diferentes metodologías que se pueden aplicar, clasificadas dentro de dos tipos: las metodologías de gestión de proyectos de software tradicionales y las metodologías de gestión de proyectos de software ágiles. Dentro del primer tipo, podemos destacar:

- Metodología de cascada o *waterfall*: organizada de arriba abajo y basada en la idea de un progreso lineal, de comienzo a fin de proyecto, donde cada fase del desarrollo tiene que estar completa para comenzar la siguiente.
- Metodología de prototipado: se presenta un prototipo de una aplicación a un grupo de usuarios para que puedan probarlo, reportar errores y sugerir mejoras. Se basa en el método de prueba y error para mejor comprender los requerimientos del producto.
- Metodología espiral: es una combinación de las dos anteriores, incluyendo un análisis del riesgo después de la fase de planificación.

En segundo lugar, tenemos las metodologías ágiles, sin duda las más utilizadas hoy en día por sus etapas iterativas. Algunas de ellas son:

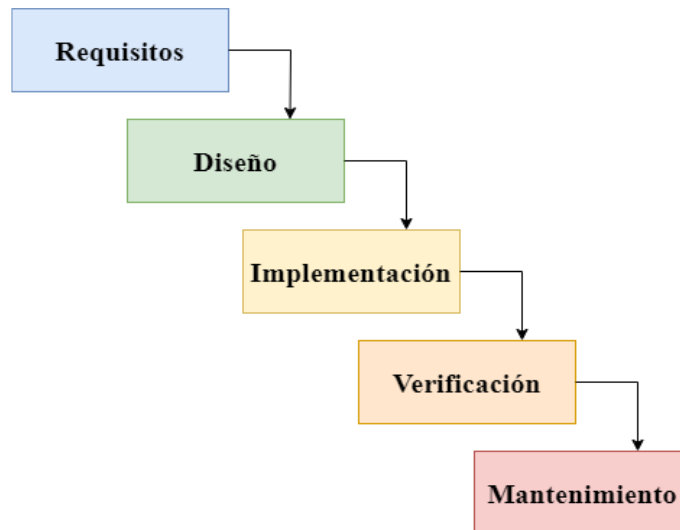


- Metodología *Scrum*: quizás la más popular en los últimos años, gracias a la posibilidad de un desarrollo iterativo e incremental. Tiene una serie de fases denominadas ciclos o mejor conocido como *Sprints*. Dentro de estos ciclos iterativos se realizan acciones con el fin de un desarrollo incremental del producto, como, por ejemplo: plan del *Sprint* y división de objetivos, ejecución de estos objetivos, reuniones diarias (*Daily meetings*) para realizar un seguimiento del proceso y demostración de los objetivos completados y no completados (*Sprint review*) al final del *Sprint*.
- Metodología *Lean*: esta metodología es normalmente aplicadas a proyectos que cuentan con una solución productiva, en los que hay que realizar cambios incrementales o mejoras.

Vistos estos ejemplos y sus descripciones, el proyecto se ha desarrollado siguiendo una metodología de cascada (*waterfall*), por los siguientes motivos: el proyecto tiene una estructura rígida, los ciclos de desarrollo son largos, el objetivo es estable y sin cambios desde el comienzo del desarrollo y las frases del desarrollo son secuenciales, es decir, hasta que no se ha terminado un servicio no se comienza el siguiente. Principalmente, este punto parte de la base de que el servicio siguiente tiene una alta dependencia del anterior.

Vistos estos puntos, la metodología en cascada se divide en las siguientes iteraciones:

1. Recopilación de requisitos de la aplicación.
2. Diseño, normalmente esquema de datos (si precisa), boceto de arquitectura y diagrama de clases (entre otros).
3. Implementación, proceso de desarrollo de los diferentes servicios.
4. Verificación, ejecutando los test necesarios (unitarios, de integración...) en los servicios, además de ejecutar y evidenciar que las búsquedas devuelven resultados coherentes.
5. Mantenimiento, normalmente llevado a cabo de la puesta en marcha en producción del sistema.



*Ilustración 1: metodología de cascada*

## 5. Descripción informática

### 5.1 Diagrama de Gantt

Los diagramas de Gantt se relacionan con la gestión de un proyecto, concretamente en su estimación de planificación a lo largo del tiempo de vida propuesto para el desarrollo del proyecto o sistema. Los ejes de este tipo de gráficas hacen referencia a diferentes elementos, por ejemplo; el eje vertical indica la lista de tareas a realizar, mientras que, el eje horizontal indica la línea temporal del proyecto. Las líneas de tareas dentro del propio diagrama indican el periodo de tiempo estimado para su resolución. También tenemos hitos o, en inglés, *milestones*, dentro de estos esquemas. Los hitos se podrían determinar como aquellas fases de mayor valor que se descomponen en otras fases. Las diferentes fases han sido estimadas en el siguiente diagrama de Gantt (*Ilustración 2*).

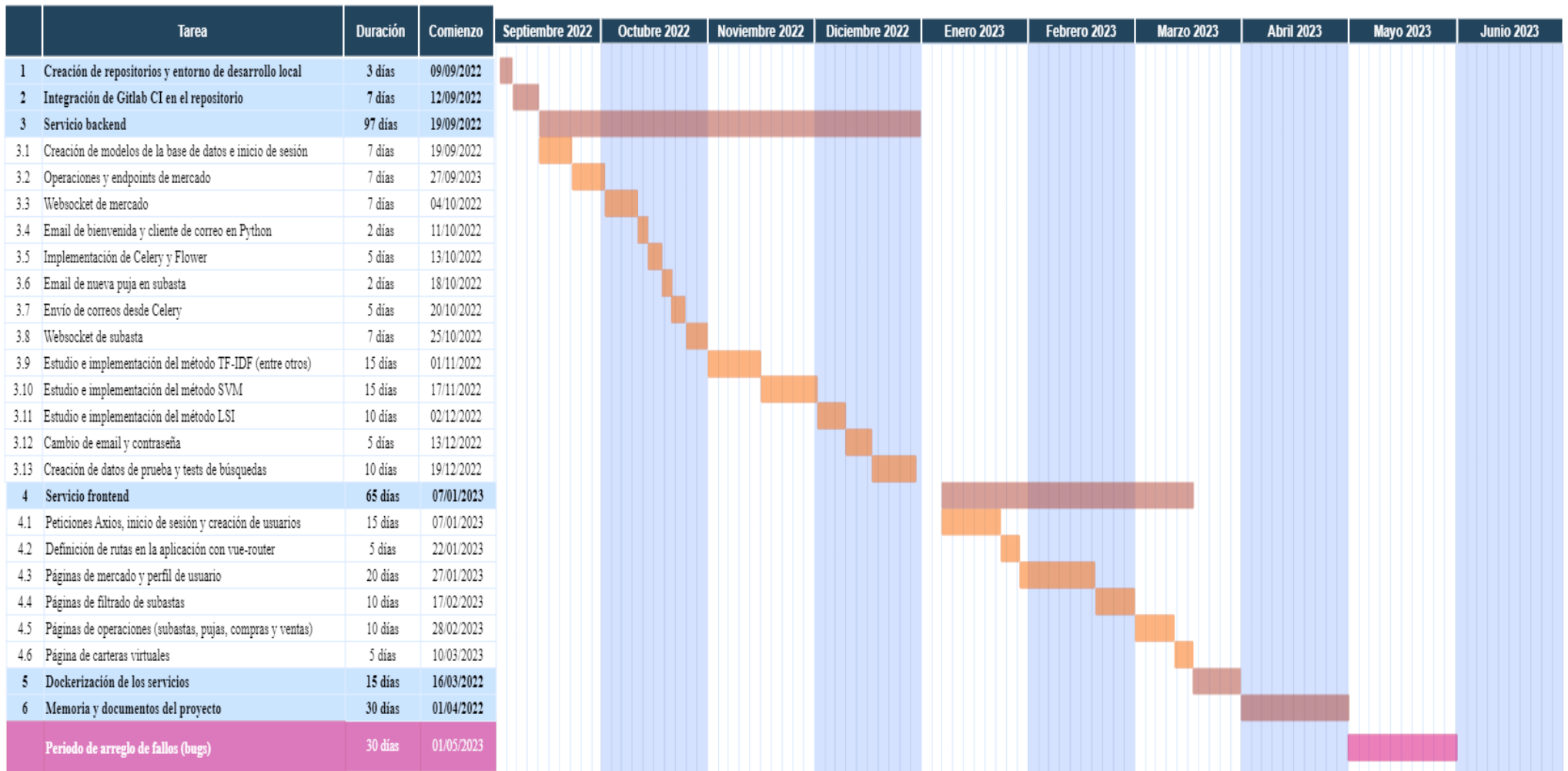
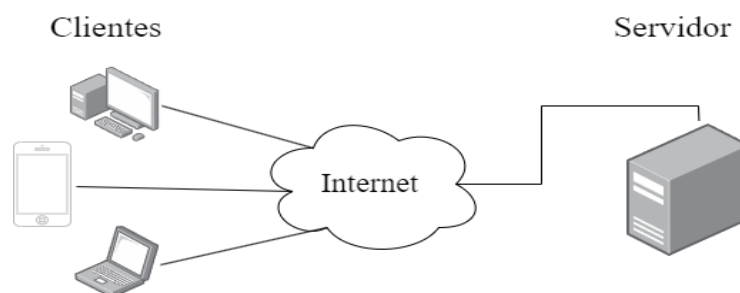


Ilustración 2: diagrama de Gantt

Como podemos observar en la imagen anterior (*Ilustración 2*), a la izquierda del esquema se encuentran una serie de filas que hacen referencia a las fases principales del proyecto con sus estimaciones en número de días y la fecha esperada del comienzo de la actividad. Estas fases se encuentran recalcadas en color rojo, mientras que las subtarefas de estas son de color naranja. También, al final del esquema observamos un periodo de tiempo reservado para la resolución de posibles fallos del sistema. Cabe recalcar que esto es una estimación organizadora del proyecto y no el mapa de desarrollo que se ha conseguido seguir al final. Este punto se podría extender con un seguimiento del proceso a medida que este va avanzando. De hecho, es una opción típica en este tipo de diagramas y bastaría con actualizar el diagrama con una línea vertical, indicando el punto de tiempo actual donde se encuentra el desarrollo del proyecto.

## 5.2 Requisitos

La primera fase de la metodología seleccionada envuelve la recopilación, documentación y entendimiento de los propios requisitos de la aplicación. Se trata de una fase, donde sin entrar mucho en detalle o a muy bajo nivel, se describen los requerimientos de la aplicación de una manera abstracta y entendible por las posibles personas no técnicas. Con estos puntos en mente, definiremos la aplicación como un servicio cliente-servidor (*Ilustración 3*).



*Ilustración 3: arquitectura cliente-servidor*

La aplicación tiene una serie de requisitos funcionales o RF y requisitos no funcionales o RNF. No hay que confundirlos como dos términos opuestos; los RF hacen referencia a comportamientos, especificaciones y características que el sistema necesita tener implementados para que los usuarios completen sus tareas u objetivos dentro de la

aplicación. Mientras que, los RNF, describen como el sistema debería comportarse o el rendimiento que debería presentar. Son relacionados los RF con los objetivos de la aplicación de la siguiente manera (*Tabla 1*).

Requisitos funcionales	Objetivos
Creación y autenticación de usuarios, el sistema debe proveer a los clientes suscribirse e iniciar sesión.	Objetivo 2.
Procesado de datos cuando comiencen y acaben las subastas.	Objetivo 2.
Validación de datos cuando los usuarios realizan operaciones.	Objetivo 2.
Creación de subastas y pujas, los usuarios deben ser capaces de crear y actualizar estas dos entidades.	Objetivo 3.
Búsquedas de subastas a partir de textos ingresados por los usuarios.	Objetivo 4.
Servir a los usuarios diferentes listados históricos de eventos como subastas y pujas creadas, compras o ventas.	Objetivo 5.
Actualizaciones en tiempo real, donde los usuarios reciben nueva información de pujas y estado de las subastas.	Objetivo 5.
Tiempo restante de las subastas, indicando cuando la subasta finaliza.	Objetivo 5.

*Tabla 1: requisitos funcionales*

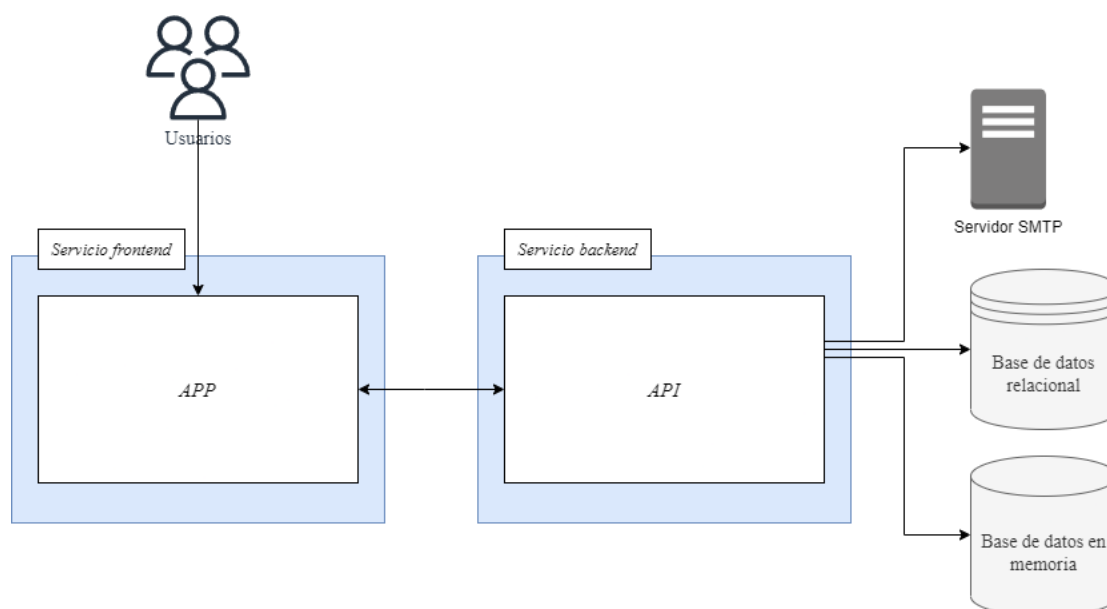
Por otro lado, los RNF se relacionan con los objetivos de la siguiente manera (*Tabla 2*).

Requisitos no funcionales	Objetivos
El sistema tiene que ser capaz de soportar un número de clientes conectados y proporcionar respuestas lo más rápido posible.	Objetivo 2.
El sistema debe proporcionar seguridad, guardar los campos sensibles cifrados en los gestores de memoria y autenticar cada operación realizada.	Objetivo 2.
Los usuarios deben de recibir actualizaciones de sus operaciones y actualizaciones de subastas por correo electrónico.	Objetivo 2.
La interfaz de usuario tiene que ser intuitiva y fácil de usar.	Objetivo 5.
El sistema debe de informar a los usuarios cuando ha ocurrido un error, con mensajes de error útiles y entendibles.	Objetivo 5.

*Tabla 2: requisitos no funcionales*

### 5.3 Diseño

La segunda fase implica dividir todo lo posible los diferentes componentes del sistema, es decir, tendremos los siguientes servicios interactuando entre sí: base de datos relacional, base de datos en memoria, *servicio backend* (en ocasiones mencionado como API, del inglés ‘*Application Programming Interface*’) y *servicio frontend* (en ocasiones mencionado como APP, acrónimo del inglés ‘*Application*’), todo ello virtualizado dentro de contenedores Docker. Además de estos puntos, cabe mencionar que el *servicio backend* se comunica a su vez con un servidor externo de correo electrónico, otro punto para tener en cuenta en este diagrama. Podemos observar en la siguiente imagen (*Ilustración 4*) el diseño de la aplicación distribuida.



*Ilustración 4: arquitectura del sistema*

Las bases de datos son las encargadas de guardar aquella información relevante para los usuarios. El *servicio backend* es donde la lógica de negocio reside en la aplicación y el encargado de crear, actualizar o borrar información guardada en las bases de datos. Además de esto, el *servicio backend* es el encargado también de mandar información sobre nuevas subastas o pujas creadas en la web. Evidentemente, será a su vez el responsable de verificar que las peticiones recibidas son posibles de ejecutar a través de métodos de autenticación. Finalmente, el *servicio frontend* es el encargado de servir

la vista al usuario y actualizarla cuando recibe nuevos datos, realizar las diferentes peticiones al *servicio backend* y gestionar las cookies de sesión de los usuarios.

Antes de comenzar la fase implementación, es de vital importancia decidir, al menos, los lenguajes de programación con los que se van a desarrollar el *servicio backend* y el *servicio frontend*, para así, encaminar la selección de paquetes software escogidos en las próximas secciones. En la sección del *servicio backend*, existen una serie de lenguajes de programación con los que es posible desarrollar este tipo de servicios, estos son: Python, utilizado a nivel mundial por su versatilidad y número de librerías disponibles de manera gratuita; JavaScript, lenguaje que sirve tanto para la creación de interfaces web como para el desarrollo de servidores backend; y finalmente, Java, altamente escalable y usado a menudo en aplicaciones empresariales y bancos. Existen otra serie de lenguajes como C#, Go o incluso PHP para desarrollar este tipo de servicios, sin embargo, la memoria se simplificará a los tres mencionados por su popularidad. El *servicio backend* está basado en Python, por los siguientes motivos: simplicidad y buen entendimiento general del lenguaje, productividad de desarrollo, prototipado rápido y el largo número de librerías disponibles. En segundo lugar, en la sección del *servicio frontend* también existen una serie de lenguajes disponibles para su implementación, estos son: JavaScript, código capaz de ejecutarse en navegadores web, además de la posibilidad de modificar las vistas de páginas web a tiempo real; y TypeScript, el cual hereda del primero, pero introduce la posibilidad de incluir tipos de datos a los objetos y variables definidos en el código, además de alguna otra funcionalidad como las interfaces, los enumerados o la inferencia de tipos. Por estos últimos motivos, el *servicio frontend* está basado en TypeScript, además de por los puntos ya expuestos, porque también permite un mejor control sobre el código gracias al tipado de datos, lo que lleva a una detección de errores temprana antes de la ejecución.

## 5.4 Implementación

La implementación se compone del desarrollo técnico de los objetivos propuestos para el sistema. En este apartado, no se explican los servicios uno a uno por el hecho de que algunos objetivos relacionan más de un servicio al mismo tiempo. Por esta razón,

comenzaremos explicando los objetivos de manera estructurada, es decir, de la capa de servicios más baja a la capa de servicios más alta.

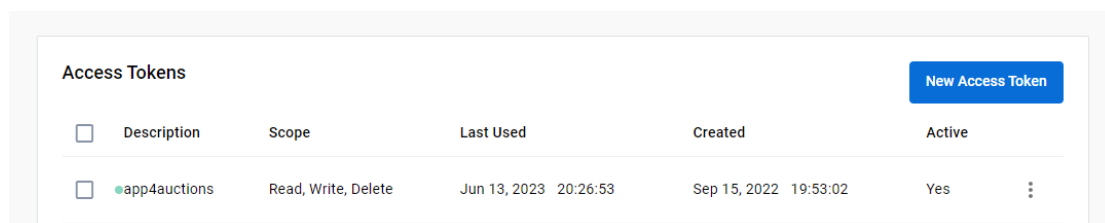
#### 5.4.1 Integración continua

Como primer punto de la implementación cabe destacar la integración continua, en inglés, *Continuous Integration*, o con sus siglas, *CI*. Se trata de un apartado un tanto especial, ya que no entra dentro del propio desarrollo de la aplicación. Es usado para realizar pruebas de integración o unitarias. Es decir, típicamente se usa en un sistema de control de versiones para verificar que los cambios implementados no han averiado funcionalidades ya implementadas, o regresiones, en el sistema. Existe también una técnica llamada despliegue o entrega continuos, en inglés *Continuous Delivery*, o con sus siglas, *CD*; el cual hace referencia al proceso de, una vez ya se han validado los cambios y ejecutado los test, desplegar el servicio de forma automática. De estos dos términos aparecen las siglas *CICD* o *CI/CD*, que indican una unión de los dos procesos, es decir, primero la ejecución de los test y más tarde el despliegue o la entrega. En el pasado, estos dos procesos estaban típicamente separados, llegando a tener que ejecutarlos de manera manual en el computador. Sin embargo, hoy en día muchos proveedores de servicios de control de versiones ofrecen estas funcionalidades de manera gratuita a los usuarios. Algunas de las herramientas más conocidas para realizar estos procesos son: Jenkins, servidor de automatización donde procesos de integración continua toman lugar, donde también es posible automatizar despliegues de software; CircleCI, similar al ejemplo anterior, presenta características de automatización de construcción de imágenes, test y despliegues; TeamCity, desarrollado por JetBrains, servidor de integración continua y despliegue. Sin duda, estos servicios ayudan a mantener una serie de estándares en el código, ejecutar comprobaciones a través de los test y desplegar las diferentes imágenes, aunque, son muchas las plataformas de control de versiones las que implementan estas tecnologías dentro de la propia plataforma y los repositorios de código. Algunos de estos ejemplos son GitHub con GitHub Actions y Gitlab con Gitlab *CICD*. Ambas soluciones son parecidas y tienen el mismo objetivo final, ejecutar *pipelines* de *CICD* en los repositorios de código. Sin embargo, este Trabajo de Fin de Grado está guardado en el sistema de control de versión Gitlab y hace uso de su servicio de *CICD* por los siguientes motivos: tiene un muy buen soporte e



integración con Docker y la configuración de los pipelines es sencilla. En muchos casos la decisión de este servicio se basa en la mera preferencia de un equipo de desarrollo o por conocimientos de la plataforma.

El siguiente y último punto de este apartado, es donde guardar las imágenes Docker una vez son construidas por los diferentes *pipelines*. Como es el caso, ya que un registro de imágenes provisionado por algún proveedor en la nube como Amazon Web Services, Azure o Google Cloud Platform tiene costes, la plataforma Docker Hub ofrece esta funcionalidad, pudiendo crear un repositorio de código privado por cada suscripción (o cuenta). El primer paso para publicar imágenes en un repositorio privado en Docker Hub es disponer de una cuenta. El servicio hace uso dicha cuenta y un repositorio privado de imágenes en la web (puede ser tanto privado como público). Más tarde, la publicación de imágenes a este repositorio se realiza utilizando un nuevo token de acceso, como contraseña, asociado a la cuenta de Docker Hub (ver *Ilustración 5*).



<input type="checkbox"/>	Description	Scope	Last Used	Created	Active
<input type="checkbox"/>	app4auctions	Read, Write, Delete	Jun 13, 2023 20:26:53	Sep 15, 2022 19:53:02	Yes

*Ilustración 5: token de acceso a Docker Hub*

Gitlab precisa de un fichero, con el nombre de `‘.gitlab-ci.yml’`, en la carpeta raíz del repositorio para ejecutar los *pipelines*. Este fichero tiene una serie de pasos o *steps*, los cuales cada *pipeline* ejecuta cuando se suben cambios al repositorio de código. Se ejecutan los siguientes pasos:











- Construcción de las imágenes de bases de datos:
  - Procesado únicamente en las ramas ``main`` y ``development``, además de generar únicamente una imagen en ambas ramas, bajo el nombre `“IMAGE_POSTGRES”` para Postgres y `“IMAGE_REDIS”` para Redis.
  - Tanto la construcción de la imagen Postgres como la imagen Redis precisan de una serie de variables de entorno, guardadas en los ajustes de Gitlab, para ejecutar las imágenes en tiempo de ejecución.
- Construcción de imagen del *servicio backend*:
  - Procesado únicamente en las ramas ``main`` y ``development``.

- Es el ocupado de ejecutar el fichero Dockerfile bajo la carpeta del proyecto ‘/api/Dockerfile’. El proceso crea dos imágenes, una con la etiqueta del número de *pipeline* y otras con las etiquetas “*IMAGE\_FASTAPI\_DEV*” y “*IMAGE\_FASTAPI\_PRO*”, estas últimas dependiendo de en qué rama se haya ejecutado el pipeline, ``development`` y ``main`` respectivamente.
- Construcción de imagen del *servicio frontend*:
  - Procesado únicamente en las ramas ``main`` y ``development``.
  - Es el ocupado de ejecutar el fichero Dockerfile bajo la carpeta del proyecto ‘/app/Dockerfile’. El proceso crea dos imágenes, una con la etiqueta del número de pipeline y otras con las etiquetas “*IMAGE\_VUE\_DEV*” y “*IMAGE\_VUE\_PRO*”, estas últimas dependiendo de en qué rama se haya ejecutado el pipeline, ``development`` y ``main`` respectivamente.

Cabe mencionar que la construcción de imágenes del *servicio backend* no tiene todas las variables de entorno presentes en la ejecución de los pipelines, ya que algunos de los valores son de carácter personal, como el cliente y contraseña de correo electrónico. A continuación, podemos observar las imágenes publicadas en Docker Hub (*Ilustración 6*).

## Tags

This repository contains 6 tag(s).

Tag	OS	Type	Pulled	Pushed
 vue_dev		Image	---	8 minutes ago
 fastapi_dev		Image	---	9 minutes ago
 postgres		Image	---	12 minutes ago
 redis		Image	10 minutes ago	12 minutes ago
 vue_pro		Image	---	15 days ago

[See all](#)

[Go to Advanced Image Management](#)

*Ilustración 6: imágenes Docker publicadas en Docker Hub*

Para conseguir realizar la conexión desde Gitlab a Docker Hub es necesario proveer a los *pipelines* las diferentes variables de entorno para iniciar sesión y realizar las publicaciones de las nuevas imágenes. Además de esto, dentro de estas variables también encontramos aquellas dedicadas a la construcción (*build*) de los diferentes servicios durante las ejecuciones de los *pipelines*. Dentro de todo proyecto en Gitlab, bajo la sección “*Settings > CI/CD > Variables*”, podemos agregar nuevas variables de entorno que estarán disponibles dentro de la ejecución de los *pipelines* (véase *Ilustración 7*).

**Variables** Collapse

Variables store information, like passwords and secret keys, that you can use in job scripts. Each project can define a maximum of 8000 variables. [Learn more](#).

Variables can have several attributes. [Learn more](#).

- Protected: Only exposed to protected branches or protected tags.
- Masked: Hidden in job logs. Must match masking requirements.
- Expanded: Variables with `$` will be treated as the start of a reference to another variable.

↑ Key	Value	Attributes	Environments
CL_GITLAB_NAMESPACE	*****	Masked Expanded	All (default)
CL_REGISTRY_NAMESPA CE	*****	Masked Expanded	All (default)
CL_REGISTRY_TOKEN	*****	Masked Expanded	All (default)
CL_REGISTRY_USER	*****	Masked Expanded	All (default)
POSTGRES_DB	*****	Protected Expanded	All (default)
POSTGRES_PASSWORD	*****	Protected Expanded	All (default)
POSTGRES_USER	*****	Protected Expanded	All (default)
REDIS_PASSWORD	*****	Protected Expanded	All (default)

[Add variable](#) [Reveal values](#)

*Ilustración 7: variables de entorno en Gitlab CI/CD*

### 5.4.2 Sistemas de gestión de memoria

La aplicación precisa de dos bases de datos diferentes para operar, estas son, una base de datos en memoria y una base de datos relacional con sus diferentes finalidades. La base de datos en memoria tiene como objetivo proporcionar un rápido acceso a las tareas programadas por el *servicio backend*, que han de ejecutarse en ciertos momentos

a lo largo del tiempo. Esta base de datos viene determinada por aquellas soportadas por el paquete [Celery](#), un programador y ejecutor de tareas. Las diferentes opciones son las siguientes: [RabbitMQ](#), un bróker diseñado para arquitecturas distribuidas que implementa una cola de mensajes a través de un protocolo conocido en inglés como *Advanced Message Queuing Protocol* (AMQP); [Redis](#), una base de datos en memoria basada en el almacenamiento de datos por pares clave-valor usada típicamente con Celery como bróker; y [Amazon SQS](#), un servicio de cola simple de mensajes distribuida. Con estas diferentes posibilidades para la implementación de la base de datos en memoria (o bróker de mensajes), se ha escogido Redis por los siguientes motivos:

- Altamente versátil: puede actuar como base de datos, bróker de mensajes o caché (en este caso se usará como bróker).
- Tipos de datos: soporta *strings*, hashes, listas y sets, entre otros.
- Permite que los datos sean guardados en disco.
- Lecturas y escrituras rápidas y eficientes.

Como segundo punto, en el apartado de las bases de datos, tenemos la base de datos de tipo relacional. Estas bases de datos se clasifican en los siguientes grupos:

- Relacionales: es el tipo de base de datos más usado. Guardan los datos de manera tabular, es decir, en tablas con filas y columnas. Cada fila tiene la información de una entrada, mientras que las columnas son los valores de sus atributos. Este tipo de base de datos utiliza un lenguaje denominado SQL (*Structured Query Language*) para realizar consultas, recuperar datos guardados, crearlos, actualizarlos o borrarlos. Algunos sistemas que presentan este tipo de base de datos son: [MySQL](#), [PostgreSQL](#) o [Microsoft SQL Server](#).
- No relaciones (o *NoSQL*): *NoSQL* no significa que no haga uso del lenguaje SQL, si no que quiere decir, en inglés, “*Not only SQL*”, o “*No solo SQL*”. Se trata de bases de datos capaces de realizar la gestión de información no estructurada o con una estructura mínima. Permiten crear modelos de datos de manera flexible y proporcionan una alta escalabilidad. Se usan normalmente en escenarios donde esta escalabilidad, la disponibilidad y una baja latencia son necesarias. Algunos ejemplos de este tipo de bases de datos son [MongoDB](#), [Cassandra](#), [CouchDB](#).

- Orientada a grafos (*Graph Databases*): diseñadas para guardar las relaciones entre entidades de datos. Son características por el uso de estructuras con nodos, aristas y propiedades para expresar las conexiones entre los diferentes nodos. Son extremadamente útiles en aplicaciones como redes sociales, motores de recomendación e incluso en sistemas de detección de fraude. Algunos ejemplos de este tipo de bases de datos son [Neo4j](#) y [ArangoDB](#).
- Datos históricos: optimizadas exclusivamente para guardar y servir, como el propio nombre indica, datos históricos, por ejemplo, mediciones de temperatura en procesos de laboratorio, mediciones de sensores o datos financieros. Ejemplos de este tipo de bases de datos son [InfluxDB](#) y [TimescaleDB](#).

Aunque existen una serie de otro tipo de bases de datos, la enumeración anterior contiene aquellas más usadas en los proyectos informáticos actuales.

Para este punto se ha tomado la decisión de implementar la base de datos relacional con *Postgres* (o *PostgreSQL*). *Postgres* es un sistema de gestión de base de datos relacional de objetos, o en inglés, “*Object-Relational Database Management System*” (también puede ser presentado con las siglas *ORDBMS*). Posee una serie de beneficios como, por ejemplo:

- Licencia de código abierto, cualquier persona puede desplegar esta base de datos con relativa facilidad.
- Alto rendimiento y capacidad de gestionar largas cantidades de datos.
- Integridad de los datos.
- Capacidad de ser usado tanto por aplicaciones web como largos sistemas de almacenamiento de datos.
- Soporte para tipos de datos primitivos como *integer*, *float* o *strings*, y hasta tipos de datos como *arrays* y documentos en formato JSON.

Además de todos estos puntos anteriores, *PostgreSQL* se ha convertido en uno de los sistemas gestores de bases de datos más populares hoy en día, llegando al nivel de MySQL en el pasado año 2022 y con expectativas de superarlo en 2023 (Véase *Ilustración 8*).

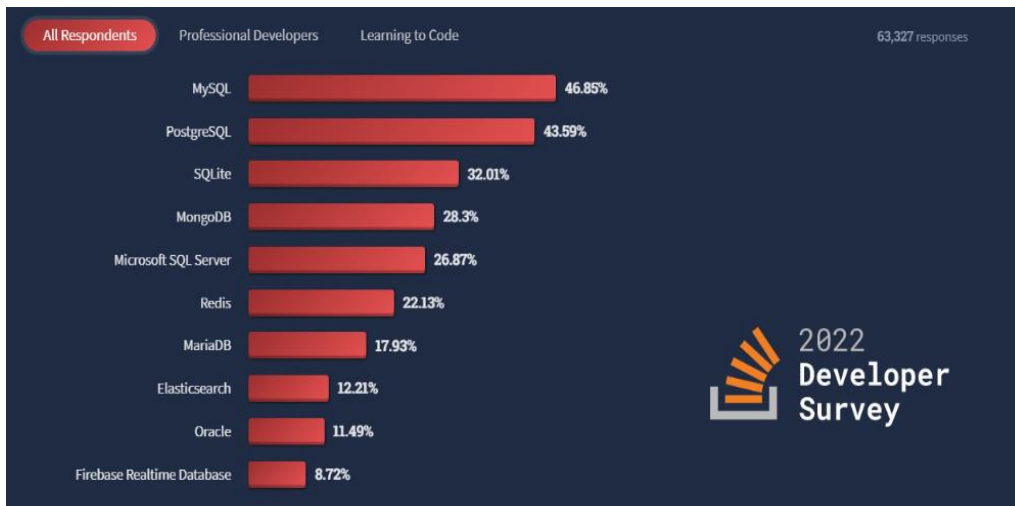


Ilustración 8: popularidad de bases de datos (2022)

### 5.4.3 Diseño de un esquema de almacenamiento de datos

el *servicio backend* será el encargado de, una vez se ponga en marcha la base de datos relacional, realizar los siguientes pasos:

1. Comprobar que los modelos de la base de datos se encuentran actualizados y en su versión más reciente.
2. En caso contrario, realizar los cambios necesarios en el esquema de la base de datos, sin perder la información existente en el sistema.

Para conseguir estos objetivos, se pueden utilizar diferentes paquetes Python que aportan estas funcionalidades. En este caso, se ha seleccionado el paquete *Alembic* para realizar esta tarea. *Alembic* es una herramienta orientada a la creación e implementación de migraciones a bases de datos, específicamente realizada para *SQLAlchemy*, otra librería popular dirigida al mapeo de relaciones de objetos. *SQLAlchemy* se define, en inglés, como *Object-Relational Mapping* (o con sus siglas, ORM). Es decir, se definen objetos en el código fuente que representan los modelos de datos. Como resultado, cada tabla de la base de datos corresponderá a diferentes objetos. Por lo tanto, definiendo un modelo en una clase Python (con el paquete *SQLAlchemy*) y haciendo uso de *Alembic* para registrar el modelo, se crean diferentes versiones de este en ficheros Python (denominados migraciones) donde se definen las diferentes columnas, el tipo de datos de cada columna, las claves foráneas y demás datos para generar el esquema entidad-relación. Una vez se ha creado y ha sido validado este fichero, se puede aplicar la migración contra la base de datos. Dentro del repositorio de código encontramos en la

carpeta `\api/alembic/versions\` las diferentes migraciones (o versiones) que se han realizado a lo largo del tiempo de desarrollo para los modelos de la base de datos. El nombre de los ficheros contiene, primero un numero hash, autogenerado por el paquete *Alembic*, seguido del nombre descriptivo que el desarrollador ha dado a esa migración en concreto. Finalmente, para aplicar estas migraciones a la base de datos, se ejecutan una serie de comandos, estos son:

- El comando `\alembic revision --autogenerate -m "objetivo de la migración"` crea el fichero con código Python de la migración.
  - La opción `--autogenerate` especifica si generar o no el script (código Python) a partir de la base de datos.
  - La opción `-m "objetivo de la migración"` especifica el mensaje que será incluido en el nombre del fichero al generar la migración.
- El comando `\alembic upgrade head` crea la transacción necesaria para crear o aplicar los cambios al modelo existente (si lo hay) en la base de datos.

Vistos estos puntos y descritos los dos paquetes relevantes para su ejecución, el *servicio backend* creara los siguientes modelos de datos: usuario, subasta, puja, cartera virtual, proceso de compra, proceso celery y versión de Alembic. Todos los modelos, excepto el último son definidos en el código fuente. El objeto versión de Alembic, es autogenerado por el propio paquete, para que así este sea capaz de verificar el identificador de la última migración aplicada a la base de datos. El siguiente esquema es generado tras aplicar las migraciones (*Ilustración 9*).



Ilustración 9: modelo entidad relación

Podemos observar en la imagen anterior (*Ilustración 9*), los diferentes modelos mencionados previamente.

- Versión Alembic (*'alembic\_version'*): modelo usado por el gestor de versionado de la base de datos.
- Usuario (*'auth\_base\_user'*): modelo que representa a un usuario. Es el modelo de mayor importancia de la aplicación, a partir de este, se generarán claves foráneas con todos los siguientes.
- Cartera virtual (*'virtual\_wallet'*): modelo que representa una cartera necesaria para realizar y recibir pujas. Las carteras virtuales tienen un número decimal representando una cantidad de dinero simulado.



- Proceso de compra (*'market\_buy\_process'*): modelo que representa una compra realizada en la plataforma. Este modelo es generado cuando una subasta ha terminado y tiene al menos una puja, esto significa que al menos un usuario se ha interesado por dicha subasta.
- Subasta (*'market\_auction'*): modelo que representa una subasta.
- Puja (*'market\_bid'*): modelo que representa una puja.
- Proceso Celery (*'celery\_process'*): modelo usado para referenciar procesos de Celery a sus respectivas subastas. Este modelo es necesario debido a las subastas, ya que estas se encuentran relacionadas con un proceso en la cola de tareas y si se actualiza la subasta, habrá que también actualizar dicha cola de procesos.

Cada uno de los nombres de los modelos, a exclusión de la versión de Alembic, contienen un prefijo referente a la parte de la base de datos con la que se relaciona: mercado (*market\_*), usuarios (*auth\_*) y Celery (*celery\_*).

Los paquetes más relevantes de este apartado son los siguientes: [SQLAlchemy](#), en su versión 1.4.41: mapeador de objetos relacionales en código para SQL; y [Pydantic](#), en su versión 1.10.2: validación y migración de datos a través de esquemas.

#### 5.4.4 Implementación de servicio para servir y procesar los datos guardados

El proceso encargado de servir los datos a los diferentes sistemas de la aplicación es el *servicio backend*. El *servicio backend* utiliza el lenguaje Python, sin embargo, existen a su vez lo que se denominan *frameworks* para el desarrollo de estos sistemas. Un *framework* es un paquete, grupo de paquetes o librerías que proporcionan la base necesaria, con componentes reutilizables, para construir servicios o aplicaciones. Dentro de Python, existen una serie de *frameworks* disponibles, los que más renombre tienen son los siguientes: [Flask](#), prioriza la simplicidad y el minimalismo. Proporciona las herramientas necesarias para crear un servicio de manera rápida; [FastAPI](#), conocido por su rapidez, simplicidad y escalabilidad. Proporciona un servicio de procesos y mensajes asíncronos para un eficiente desarrollo y aplicaciones rápidas; y finalmente, [DjangoREST](#), también conocido por las siglas *DRF*, construido por encima del paquete Django, proporciona una amplia cantidad de herramientas, características y utilidades para elaborar servicios REST. En este caso, se ha seleccionado FastAPI para el

desarrollo del *servicio backend* por los siguientes motivos: es rápido y eficiente, genera automáticamente la documentación del API, conocida como *Open API* (en ocasiones también es llamado *Swagger*), simplicidad de uso, integrable con la infinidad de paquetes existentes para el lenguaje, soporte tanto de código síncrono como asíncrono y la rápida integración que tiene con el paquete Celery.

La documentación o especificación *Open API* es un estándar de la industria del software para documentar *APIs* REST, con una lista de los diferentes *endpoints* que ofrecen, accesibles a través de un navegador web. A su vez, un *endpoint* es una ruta dedicada a una operación dentro de un *API*. Los *endpoints* aceptan, por lo general, peticiones HTTP de tipo GET, POST, PUT, DELETE o PATCH para así ejecutar código relacionado con en la lógica de negocio y realizar cambios en los recursos. Estos *endpoints* normalmente reciben los datos en la propia URL de la llamada o en el cuerpo de la petición.

El *servicio backend* precisa de una serie de variables de entorno para operar y ejecutar el código. Para un desarrollo local, estos valores por lo general se obtienen de un fichero llamado “.env”, donde se guardan las variables en un formato clave-valor. En el proyecto, este fichero lo encontramos bajo la ruta dentro ‘*/api/conf/.env.template*’. Dentro de este fichero visualizamos una serie de valores imprescindibles para la correcta ejecución del *servicio backend*, algunos de ellos ya definidos con un valor fijo, otros definidos como ‘*\_\_CHANGE\_ME\_\_*’ (o en español, “*cámbiame de valor*”) y otros sin valor alguno. Los diferentes campos son los siguientes:

- *SECRET\_KEY*: uno de los valores más importantes del *servicio backend*. Se usa para codificar los *tokens* de sesión cuando los usuarios se conectan a la página web (proceso de inicio de sesión).
- *ALGORITHM*: algoritmo usado para cifrar las contraseñas de los usuarios en la base de datos. En este caso usamos HS265.
- *ACCESS\_TOKEN\_EXPIRE\_MINUTES*: número de minutos antes de que un token de sesión sea invalidado por el *servicio backend*.
- *POSTGRES\_\**: todo valor con este prefijo estará relacionado con la conexión a la base de datos. Habrá un total de cinco valores con este prefijo, estos son: el usuario, la contraseña, el nombre del servidor o servicio que contiene la base de datos, el número de puerto por el que escucha dicho servidor y el nombre de la base de datos contra la que trabajar.

- *DEBUG*: modo desarrollo. Puede tener el valor ``True`` o ``False``.
- *MAILER\_\**: ajustes de correo electrónico. Para el desarrollo de este proyecto se ha usado un cliente de correo personal, por lo tanto, estos valores no serán incluidos ni en la memoria ni en el código fuente.
- *REDIS\_\**: variables de entorno relacionadas con la base de datos en memoria.
- *ACTIVATE\_ACC\_URL*: ruta del *servicio frontend* donde se activan las cuentas de usuarios. Principalmente, esta variable es usada para incluirla en los emails que reciben los usuarios al subscribirse al sistema.
- *CELERY\_\**: valores relacionados con el proceso Celery.

Todos los datos que son proporcionados al *servicio backend* requieren de un mecanismo de filtrado y validación, para así verificar que la información que recibe el *servicio backend* es útil y correcta. Uno de los paquetes Python más relevantes usados para este fin es *Pydantic*. Se trata de una librería de Python que proporciona validación y analizado de información, con foco en la serialización y deserialización de los datos. Con esta librería, es sencillo definir un esquema y validar una serie de datos entrantes contra ese esquema, además de transformar los datos obtenidos a un objeto Python (partiendo de un esquema definido).

Para este fin, será necesario crear diferentes clases en Python utilizando este paquete, y posteriormente, anotar los datos requeridos por cada endpoint con las clases creadas (los diferentes esquemas).


La especificación *Open API* nos proporciona una interfaz para consumir los *endpoints* del *servicio backend* de manera directa, sin necesidad de hacer uso de herramientas como *Postman* o *Testfully* (entre otras). Se trata de una interfaz de carácter público disponible para los desarrolladores de software.

Añadiendo un *endpoint* en el código, este automáticamente es incluido en la especificación *Open API*. Existe también la posibilidad de no agregar ciertos endpoints a esta especificación, simplemente añadiendo la opción `“include_in_schema=False”` en el router al definir cualquier *endpoint*. Todos los *endpoints* pueden ser explicados con una pequeña descripción del servicio que un *endpoint* ofrece, la lista, si precisa, de los parámetros necesarios para consumirlo y qué se puede esperar que sea devuelto como respuesta. Dentro de la aplicación encontramos la siguiente especificación *Open API* (*Ilustración 10*).

## App 4 auctions API 0.0.0 OAS3

/api/v1

Services and endpoints for APP4AUCTIONS project.

Authorize 

API status		^
GET	/system/status/	Check if the service api is online
User routes		^
GET	/auth/user/	Get your user data
PUT	/auth/user/	Update your user fields
POST	/auth/user/	Create a new user
DELETE	/auth/user/	Delete your user
GET	/auth/user/activate/{user_uuid}/	Activate your user
POST	/auth/user/login	Login with credentials: email & password
POST	/auth/user/email-change/	Change a user email
POST	/auth/user/password-change/	Change a user password
POST	/auth/user/password-reset/	Send an email to change your password based on a code

Ilustración 10: especificación Open API

Este es tan solo un ejemplo de algunos de los *endpoints* que contiene el *servicio backend*. En total, contiene un total de 30 *endpoints* destinados a diferentes servicios y operaciones.

Como ejemplo vamos tomaremos el *endpoint* `/auth/user/`, el cual tiene que ser llamado a través de una operación HTTP POST, ya que este es el primer *endpoint* por el cual un usuario nuevo de la web accedería, con el fin de darse de alta y comenzar a operar en la plataforma. Abriendo en detalle el *endpoint* dentro de la especificación, podemos observar los diferentes campos de los que precisa recibir, además de los códigos HTTP que puede devolver en cada llamada (*Ilustración 11*).

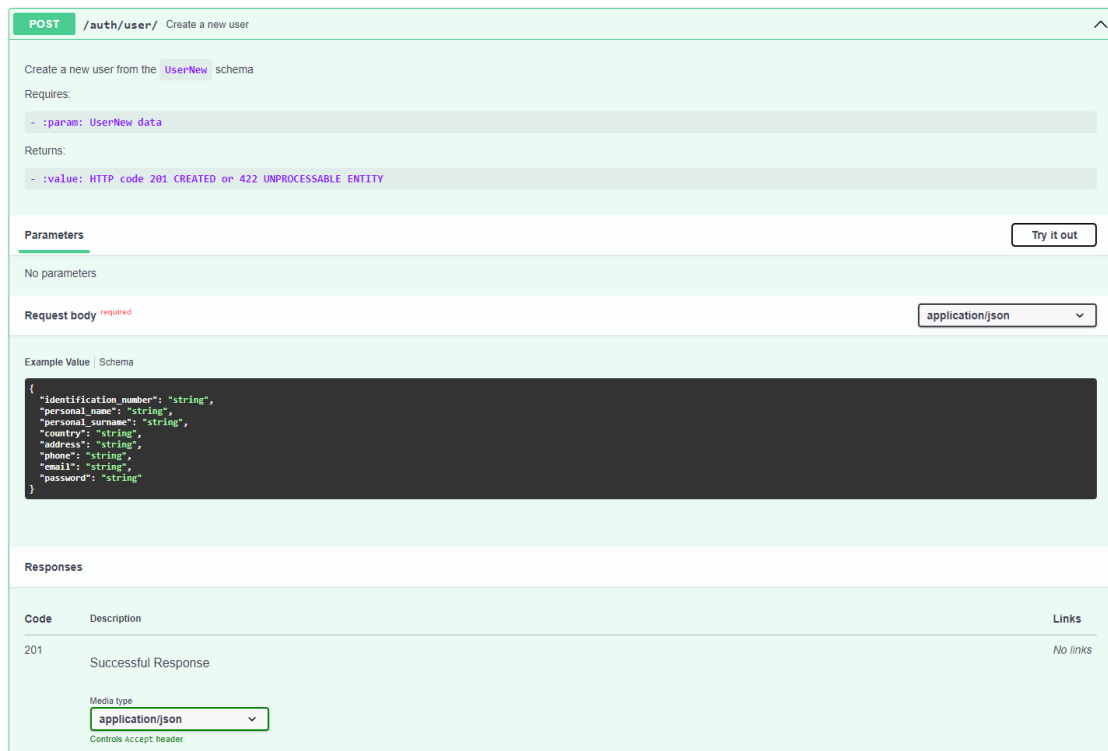


Ilustración 11: endpoint de ejemplo

Los *endpoints* del *servicio backend* se dividen en las siguientes categorías, relacionándose así en la mayoría de los casos, con los diferentes modelos o entidades de la base de datos relacional:

- Sistema o estado del servicio: especialmente empleado para comprobar si el servicio se encuentra levantado o no. Devuelve un sencillo mensaje con la hora cuando se ha realizado la petición y un pequeño mensaje.
- Usuarios: operaciones relacionadas con los usuarios de la web, ya sean peticiones para crear usuarios, iniciar sesión, cambiar los datos de la cuenta, etc.
- Mercado: creación de subastas y pujas, además de filtros de subastas y datos de las operaciones realizadas por los diferentes usuarios. Además de listados de procesos de venta y de compra de cada usuario.
- Carteras virtuales: la web simula una serie de carteras virtuales que cada usuario necesita para pujar y recibir ofertas en las subastas.

También, a través de la especificación *Open API*, es posible visualizar en detalle los esquemas de los que hace uso el *servicio backend*, al final de la lista de *endpoints* (Ilustración 12).

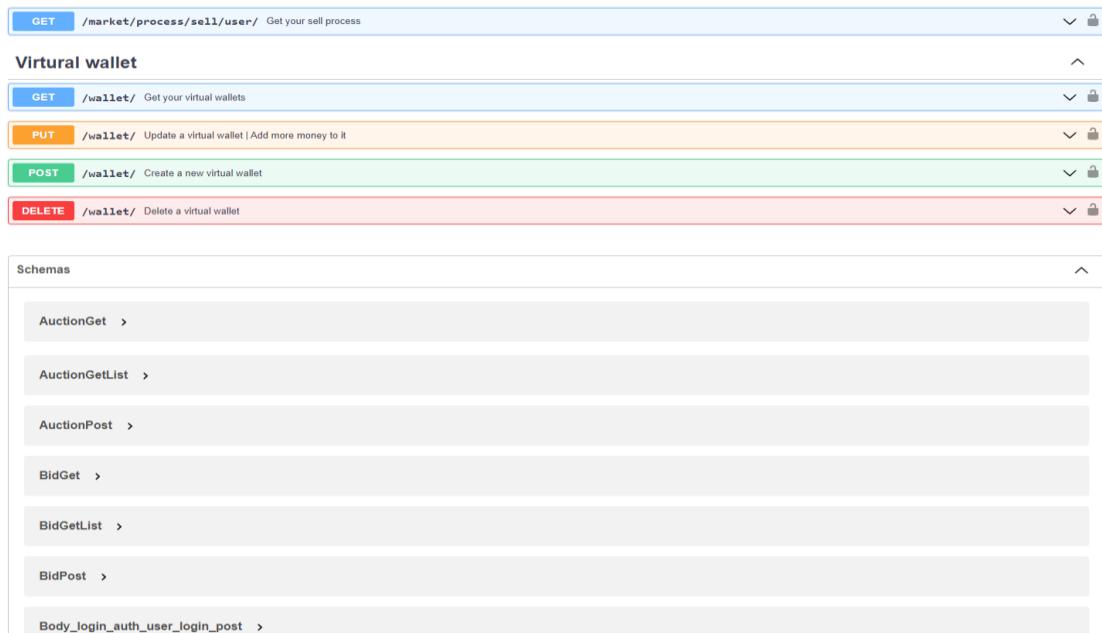


Ilustración 12: esquemas del API

Una llamada de ejemplo, tomando como modelo el *endpoint* para comprobar si el *servicio backend* se encuentra levantado o no, quedaría como la ilustración que vemos a continuación (*Ilustración 13*).

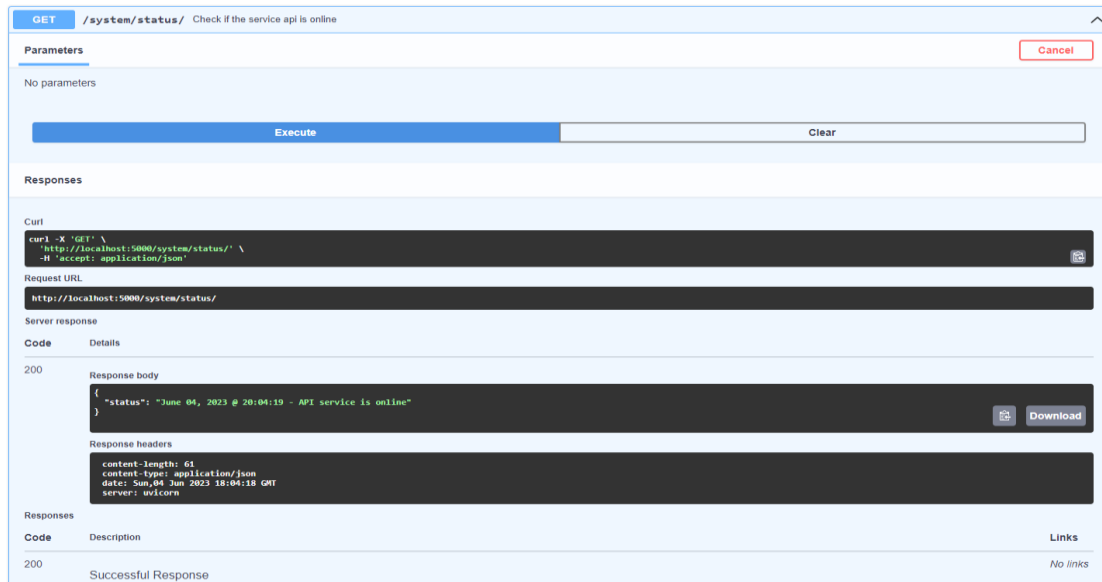


Ilustración 13: respuesta del endpoint de salud del servicio backend

Observamos diferentes campos en la imagen (*Ilustración 13*), como, por ejemplo: URL de la llamada con su cabecera y los datos y el código de respuesta.

Cabe destacar en este apartado una clase de carácter un tanto peculiar en el *servicio backend*; esta clase es usada principalmente para gestionar los modelos de la base de datos, es decir, actúa como interfaz conectora entre el *servicio backend* y la base de datos relacional. Para este fin, dentro de la carpeta que contiene el código del proyecto, en el fichero bajo la siguiente ruta `‘/api/utills/database_context.py’`, podemos encontrar una clase definida en Python con el nombre de “*Session*” que será la que se ocupe de registrar, actualizar y enviar los datos a la base de datos. En el momento que se declara un objeto de este tipo, son realizadas una serie de operaciones antes de devolver el objeto `‘db’` instanciado como variable dentro de la misma clase, como, por ejemplo:

- Crear el motor que trabajara contra la base de datos pasada como parámetro, o en su defecto, contra aquella definida en las diferentes variables de entorno precedidas con el prefijo “*POSTGRES\_*”.
- Definir una variable de carácter sesión, que será la ocupada de hacer las conexiones contra el motor de base de datos definido en el paso anterior.

Existe una larga variedad de ejemplos en el código donde se hace uso de esta clase, un ejemplo se puede encontrar dentro del fichero bajo la ruta `‘/api/app_auth/encrypt.py’`, en el método `‘get_current_user(...)’`, donde se realiza una transacción a la base de datos para recuperar los datos de un usuario a partir de un *token* de sesión.

Una vez creado un objeto de este tipo en el código, este contiene una serie de métodos recalables para realizar las transacciones contra la base de datos relacional, los más importantes son los siguientes:

- `‘db.query().filter().all()’`:
  - Al método `‘query()’` se le proporciona una clase de un modelo definido en la base de datos. Por ejemplo, en lenguaje informal, “Esta petición trabajará contra X modelo”.
  - Al método `‘filter()’` se le proporciona una sentencia que será más tarde transformada a lenguaje SQL. Por ejemplo, en lenguaje informal, “Filtra y devuelve todos los modelos de tipo X donde el campo id de X sea igual a Y”, siendo Y, por ejemplo, una variable de tipo entero.
  - El método `‘all()’` devuelve todas las entradas de la base de datos que coincidan con la consulta.
- `‘db.add()’`:

- Al método ``add()`` se le proporciona un objeto definido como alguno de los posibles modelos de la base de datos.
- ``db.commit()``:
  - El método ``commit()`` realizara la transacción de la base de datos, ya sea para registrar, actualizar o borrar una entrada de algún modelo de la base de datos.

Estos son algunos ejemplos sencillos de lo que una clase de sesión puede llegar a realizar, para una descripción más detallada de estos y muchos otros métodos véase [conceptos básicos de la sesión](#) de la documentación oficial del paquete *SQLAlchemy*. Los paquetes más relevantes de este apartado son los siguientes: [FastAPI](#) en su versión 0.85.0: framework para desplegar el *servicio backend*. Requiere una versión de Python igual o mayor que Python3.7; [Uvicorn](#) en su versión 0.18.3: servidor ASGI (del inglés, Asynchronous Server Gateway Interface), interfaz estándar entre servidores web y aplicaciones Python; [Starlette](#) en su versión 0.20.4: framework ASGI para servicios web asíncronos en Python, FastAPI está basado internamente en este paquete; [bcrypt](#) en su versión 4.0.1: hashing de contraseñas o cadenas de texto, [loguru](#) en su versión 0.6.0: logging simple y sencillo para procesos y aplicaciones Python; y finalmente [psycopg2-binary](#) en su versión 2.9.3: adaptador y conector Python para bases de datos PostgreSQL.

#### 5.4.5 Implementación de servicio para procesar tareas programadas

Para las tareas programadas se utiliza la librería Celery. Se trata de un paquete escrito en Python, que tiene el objetivo de implantarse en diferentes ámbitos como, por ejemplo: programación asíncrona, programación de tareas o gestionar tareas en segundo plano y distribuidas a lo largo de diferentes sistemas.

Es especialmente útil para ejecutar tareas en paralelo y no bloquear la ejecución principal. En este caso es usado para el punto número dos, como programador y ejecutor de tareas. En este momento nos podríamos hacer la siguiente pregunta, *¿por qué hacer uso de un sistema como Celery en la aplicación?* Muy sencillo, y poniendo como ejemplo la creación y/o finalización de una subasta:

- Durante la creación de una subasta, esta no empezará hasta la fecha programada por el usuario, lo que significa que el sistema:



- No devolverá la subasta hasta que la fecha actual sea mayor que la fecha de comienzo de la subasta.
- Cuando comience la subasta, será notificada desde el proceso Celery al *servicio backend*, el cual enviará a todos los clientes conectados, si es que hay alguno, a través del protocolo WebSocket, la nueva subasta para informar que ha comenzado y que se pueden realizar pujas en ella.
- Durante la finalización de una subasta, se determina quien ha ganado, si es que hay al menos una puja, informe al creador de la subasta si ha tenido algún usuario que haya pujado en su subasta y a aquella persona que puja, para informar que ha ganado la subasta. También, creará un proceso de compra y retirará la cantidad de dinero pujado por el artículo de la subasta de la persona que puja, el comprador, y lo ingresará en la cartera virtual de la persona que ha creado la subasta, el vendedor. En un entorno productivo real, esta cantidad de dinero debería de ser retenida en la web hasta que el usuario que puja y gana la subasta, el comprador, recibe el artículo de manera física o virtual.

Por lo tanto, la lista de tareas definidas dentro del proceso Celery son las siguientes: notificar sobre una subasta que acaba de comenzar y, terminar y procesar los datos finales de una subasta.

Celery necesita una serie de requisitos para trabajar, estos son:

- Interfaz en Python. Se instala como un paquete Python, [paquete Celery](#), dentro de un entorno virtual para trabajar y conectar los procesos Celery con los sistemas descritos a continuación.
- Un bróker para gestionar la comunicación entre procesos productores de tareas y los trabajadores. Hay diferentes tipos de bróker disponibles, pero en este caso y como se ha visto anteriormente, se usará *Redis*.
- Uno o varios procesos trabajadores (del inglés ‘*workers*’). Serán los encargados de ejecutar las tareas programadas en segundo plano. En este caso existe un único proceso de tipo trabajador para ejecutar las tareas programadas.
- Configuración para que el proceso Celery sea capaz de conectarse la base de datos en memoria y a la base de datos relacional.
- Definición en código de las diferentes tareas de Celery, que son funciones Python decoradas con la anotación “`@app.task()`”. “`@app`” hace referencia al

propio proceso trabajador de Celery, digamos que, es una manera de registrar los diferentes tipos de tareas.

Dentro de la carpeta que contiene el código del proyecto, en el fichero bajo la ruta `‘/api/executor.py’` se encuentra, en código Python, las tareas mencionadas anteriormente y la propia aplicación Celery.

Un objeto de tipo Celery requiere, como mínimo, de los siguientes parámetros:

- `‘main’`: nombre del módulo principal.
- La dirección del bróker, precedida en este caso del conector a la base de datos en memoria `‘redis://usuario:contraseña@servidor:puerto’`.

Necesitará también disponer de las variables de entorno precisas para acceder a la base de datos en memoria y a la base de datos relacional, igual que el *servicio backend*.

Una vez determinadas las tareas, son puestas en la cola de ejecución desde el *servicio backend*. Dentro de la carpeta que contiene el código del proyecto, en el fichero bajo la siguiente ruta `‘/api/app_marketplace/celery_creator.py’` podemos encontrar el código para programar tareas dentro de Celery.

Cabe mencionar que para el funcionamiento de Celery dentro del sistema, se ejecutará un solo nodo trabajador, mientras que el encargado de suscribir tareas a la cola será el propio *servicio backend*.

El proceso Celery desconoce del ciclo de ejecución del *servicio backend*, por lo tanto, este último dispone de un *endpoint* dedicado exclusivamente a recibir datos desde el ejecutor de la cola de tareas programadas. Para evitar que se haga un uso fraudulento o malintencionado de dicho *endpoint*, se encuentra protegido a través de un *API key*, conocido exclusivamente por Celery y, evidentemente, el *servicio backend*. Además, este endpoint no se incluye en la especificación *Open API*, ya que no interesa desde un punto de vista de seguridad que los usuarios u otras personas puedan conocerlo y así intentar explotarlo malintencionadamente. Un *API key* es un tipo de objeto, usado típicamente para autenticar y/o autorizar acceso a un *API* o servicio, desde otro completamente diferente, ya sea una aplicación web, otro *API* e incluso un script. Estas claves de acceso normalmente son enviadas en las cabeceras de las peticiones HTTP por el cliente que consume el servicio. Es de vital importancia mantener este tipo de claves seguras, de no hacerlo, cualquier persona con acceso a la clave podría autenticarse como el usuario original de la clave y realizar las peticiones en su nombre. En el supuesto caso de que se filtrase esta clave de acceso en concreto en un entorno

productivo, cualquier persona podría hacer uso de los *endpoints*, si los conoce, dedicados a la actualización de subastas y el paso de información desde el proceso Celery al *servicio backend*. Por lo general, un proveedor de un servicio que hace uso de esta tecnología presenta a los clientes una detallada lista con información, con los pasos a seguir para conseguir este tipo de clave de acceso. Sin embargo, en este caso, los clientes no podrán solicitarla, ya que no tendría sentido, la clave estará guardada en variables de entorno conocidas únicamente por el *servicio backend* y el proceso Celery. Al crear una nueva subasta se llevan a cabo los siguientes pasos de mensajes entre el *servicio frontend*, el *servicio backend* y el proceso Celery (*Ilustración 14*).

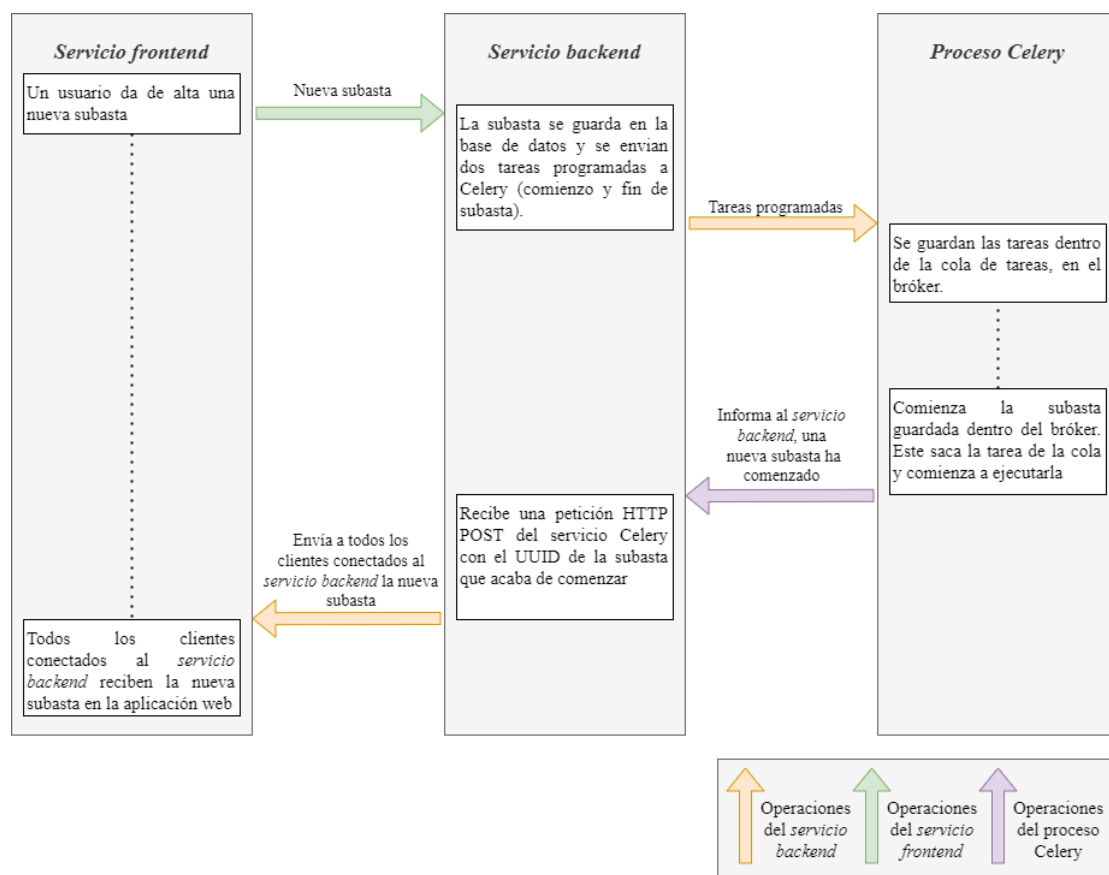


Ilustración 14: flujo de eventos al comenzar una subasta

En la imagen anterior (*Ilustración 14*), podemos visualizar como es el *servicio backend* el responsable de añadir las tareas de inicio de nuevas subastas a la cola de procesos, mientras que es el proceso Celery el encargado de sacarlas de la cola y procesarlas. Sin embargo, en el momento en el cual finaliza una subasta, se llevan a cabo los siguientes pasos de mensajes entre el proceso Celery, la base de datos relacional y el servidor externo SMTP (*Ilustración 15*).

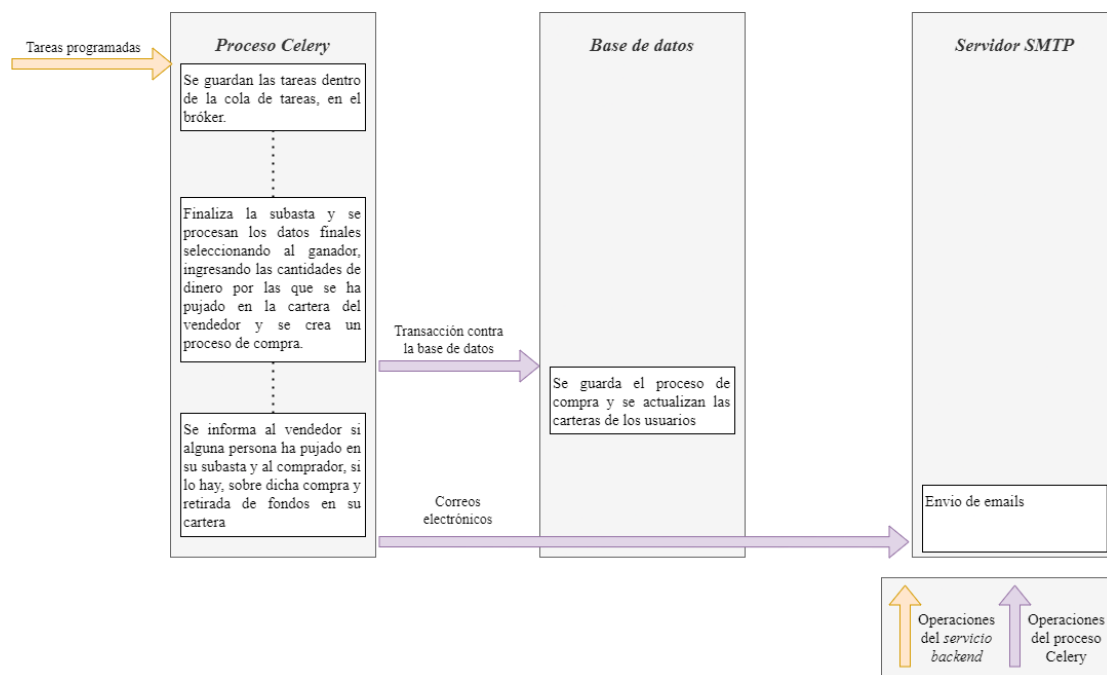


Ilustración 15: flujo de eventos al finalizar una subasta

El paquete Celery ofrece a su vez una herramienta de administración y monitorización a tiempo real basada en una aplicación web. Esta herramienta tiene el nombre de Flower. Con Flower se pueden realizar tareas como la monitorización en tiempo real del progreso de las tareas programadas, en curso y finalizadas, obtener una vista detallada de las tareas, con información como el nombre de las tareas, los argumentos que ha recibido al programar una tarea en concreto o el tiempo de ejecución y opcionalmente gráficas y estadísticas. Además de estas características, también se pueden comenzar y terminar tareas de una manera manual dentro de este panel de control, por lo tanto, es de vital importancia gestionar el acceso a esta herramienta. La siguiente imagen muestra un ejemplo de este panel de control (*Ilustración 16*).

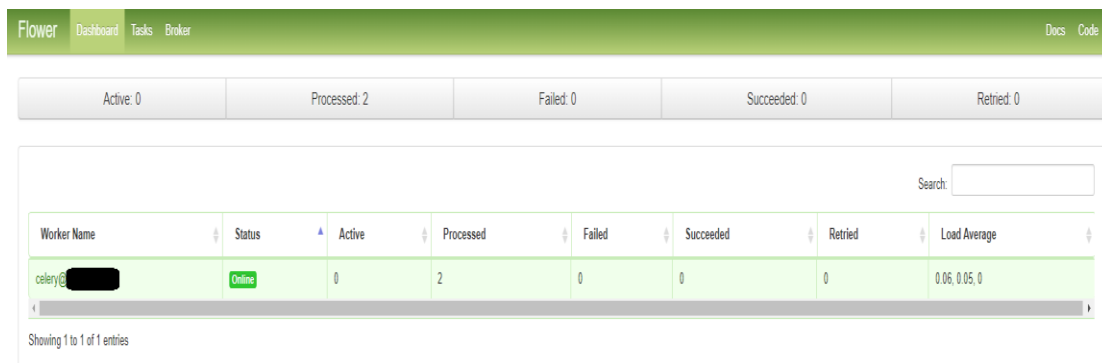


Ilustración 16: panel de control Flower

Como podemos observar en la imagen anterior (*Ilustración 16*), visualizamos un nodo de ejecución, con dos tareas en estado ‘*Processed*’, es decir, han llegado a la cola de tareas, pero aún no han sido ejecutadas. Accediendo en el panel al nodo de ejecución (*Worker Name*) y más tarde a tareas (*Tasks*), se pueden visualizar todas las tareas activas, programadas, reservadas y canceladas. Se dividen de la siguiente manera:

- Activa: la tarea se encuentra se encuentra actualmente en ejecución.
- Programada: la tarea se encuentra en espera a que llegue el momento de su ejecución o cuenta atrás.
- Reservada: la tarea ha llegado a la cola de tareas para ser ejecutada, pero ningún nodo de ejecución ha sido capaz de ponerla en marcha, esto se debe a una alta demanda de los nodos.
- Cancelada: la tarea no será ejecutada.



Ilustración 17: vista de tareas en Flower

En la imagen anterior (*Ilustración 17*) se observan dos tareas diferentes, correspondientes en este caso a una única subasta, con su identificador único de la tarea y los argumentos pasados a la misma en el momento que fue programada.

Cuando una subasta es actualizada, es decir, el usuario cambia su fecha de comienzo o de finalización, todas las tareas programadas respectivas a esa subasta en concreto tendrán que ser actualizadas en la cola de tareas. Por ejemplo:

- Un usuario crea una subasta que comienza el día 01/01/2024 a las 12:00 horas y termina el día 02/01/2024 a las 13:00 horas.
- Ambas tareas, comienzo y fin de subasta, son enviadas a la cola de tareas para ser ejecutadas en sus respectivos momentos.
- El usuario se percata de un error y decide actualizar su subasta para que termine un día más tarde a la misma hora, es decir, el día 03/01/2024 a las 13:00 horas.
- Se cancelan ambas tareas programadas originalmente y se envían otras dos tareas nuevas con los datos actualizados.

Tras este proceso, dentro del panel de control Flower encontraríamos 4 tareas, las dos iniciales, en estado *'Scheduled'* y *'Revoked'*, y las dos últimas, en estado *'Scheduled'*. Esto se debe a que Celery no elimina directamente las tareas de la cola programada para su ejecución, aunque una tarea haya sido cancelada. En su lugar, las tareas canceladas seguirán en estado *'Scheduled'* y en el momento de su ejecución Celery verificará que también se encuentran en estado *'Revoked'*, entonces en el momento de ejecución serán directamente descartadas sin ejecutarse.

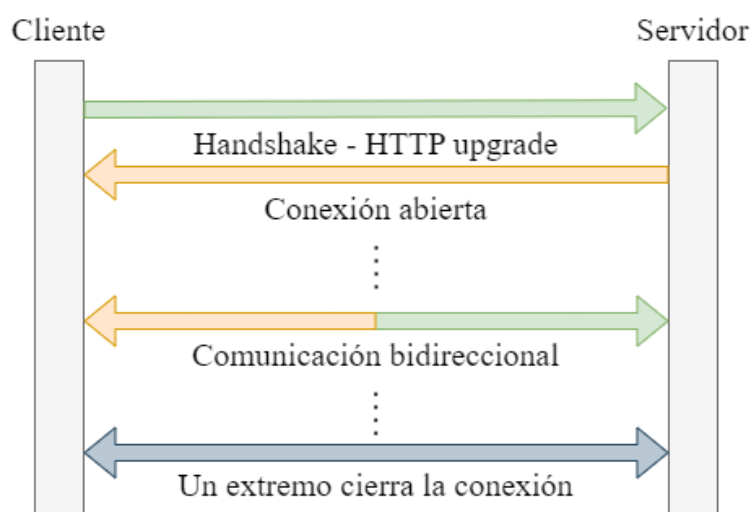
Los paquetes más relevantes de este apartado son los siguientes: [Celery](#) en su versión 5.2.7: cola de tareas distribuida; y [Flower](#) en su versión 1.2.0: interfaz web para visualizar colas de tareas y estado de ejecución de los diferentes procesadores de la cola.

#### 5.4.6 Implementación de servicio para servir datos en tiempo real

Con el desarrollo de las tecnologías, cada día son más páginas web las que proporcionan actualizaciones en tiempo real a los usuarios. Tiene una serie de beneficios, especialmente notables en la sección de experiencia de usuario y sistemas reactivos. Típicamente en procesos servidores, este fin suele aplicarse a través de un protocolo denominado WebSocket, el cual consiste en abrir un canal de comunicación

bidireccional entre un cliente y un servidor sobre un único socket TCP. Por lo general, el cliente es un navegador web. Este protocolo permite una transmisión de datos denominada como “*full-duplex*”, traducido como un tipo de comunicación bidireccional, que permite enviar y recibir mensajes de forma simultánea, es decir, mantener conexión persistente durante un periodo de tiempo hasta que una de las dos partes acaba con la conexión de la comunicación.

Dentro del *servicio backend*, se usa esta tecnología para facilitar la comunicación en tiempo real. El protocolo WebSocket atraviesa una serie de pasos para comenzar, mantener y cerrar las conexiones. Primero, comienza lo que denominamos “*Handshake*”, donde un cliente envía una petición HTTP a un servidor, pidiéndole al servidor actualizar, o en inglés, “*to upgrade*”, la conexión HTTP (conexión no segura) o HTTPS (conexión segura) a una de tipo WebSocket, WS (conexión no segura) o WSS (conexión segura). Si el servidor puede atender esta petición, responderá con un código HTTP 101 cambiando protocolos, o en inglés, “*Switching Protocols*”. Después, se inicia una conexión persistente entre el cliente y el servidor, hasta que una de las dos partes termine con la conexión. A lo largo de la vida de esta conexión, ambas partes aceptan una comunicación bidireccional, es decir, el cliente puede enviar datos al servidor y viceversa. Finalmente, cuando el cliente o servidor lo consideren necesario, se cierra la conexión establecida. Típicamente, es el cliente el que cierra esta conexión, por ejemplo, abandonando la página web en la que se encuentra el usuario. El navegador enviará finalmente un mensaje parecido a “*Please close connection*” o “*Connection closed*”. Véase *Ilustración 18*.



*Ilustración 18: ciclo de vida del protocolo WebSocket*

Algunas de las características del protocolo son las siguientes:

- Arquitectura orientada a eventos: el servidor es capaz de enviar datos al cliente sin este haberlos pedido explícitamente. Es una característica que permite interacciones y actualizaciones en tiempo real.
- La comunicación a través de este tipo de protocolo se realiza con paquetes llamados “*Data Frames*”. Cada uno de estos paquetes contiene a su vez un campo conocido como “*Payload*”, que es el propio mensaje o parte de él, que está siendo transmitido a través del paquete enviado. Si el “*Payload*” es demasiado grande, este puede ser dividido en diferentes “*Data Frames*” enviado y después reconstruido por el cliente.
- El protocolo WebSocket puede ser integrado con la lógica de aplicación de un servicio.

El *servicio backend* contiene dos endpoints que son capaces de crear, mantener y cerrar una conexión de tipo WebSocket. Dentro de la carpeta que contiene el código del proyecto, en el fichero bajo la siguiente ruta ‘*/api/app\_ws/marketplace\_ws.py*’ podemos encontrar las definiciones en código de los dos websockets creados por el *servicio backend*, estos tienen las siguientes rutas en el *servicio backend*:

- “*/marketplace/ws/*”: websocket del mercado.
- “*/marketplace/room/{auction\_uuid}/*”, siendo ‘*auction\_uuid*’ el identificador único de una subasta de la web: websocket de subasta.

El websocket del mercado, es usado por todo cliente al iniciar una conexión a la página web, sea una vista cualquiera, se abre de manera automática una conexión WebSocket contra el *servicio backend*. Esta conexión es utilizada para recibir actualizaciones de nuevas subastas que comienzan en la página, es decir, una subasta comienza en un punto en el tiempo más allá del momento actual y cuando este momento en el que comienza la subasta llega, el servicio backend se ocupa de actualizar a todos los clientes conectados con los datos de esa nueva subasta, para así incluirla en la página web de manera automática y ser visualizada por los usuarios sin necesidad de recargar el navegador.

Para visualizar un ejemplo completo de este proceso necesitamos como mínimo dos clientes conectados a la página web (podría ser uno solo, sin embargo, tomar el ejemplo



con dos clientes conectados al mismo tiempo es un escenario más real). El flujo de ejecución es el siguiente:

1. Dos clientes diferentes, cliente 1 y cliente 2, acceden al mercado de subastas de la página web.
2. Dos conexiones de tipo WebSocket son abiertas dentro de los navegadores de los clientes.
3. Una nueva tarea, de tipo inicio de subasta, comienza a ejecutarse y es eliminada de la cola de tareas por el proceso Celery.
4. El proceso Celery realiza una petición HTTP al *servicio backend* con el identificador único de la subasta que acaba de comenzar, aportando el *API key* en la cabecera de la petición.
5. El *servicio backend* toma como “*Payload*” de la petición HTTP el identificador de la subasta que comienza, la busca en la base de datos y su información es distribuida a todos los clientes conectados a través del websocket del mercado.
6. Los navegadores web del cliente 1 y el cliente 2 actualizan de manera automática la vista general de subastas incluyendo esta al comienzo.

Visto de manera visual, los clientes y servicios siguen el siguiente proceso de ejecución (*Ilustración 19*).

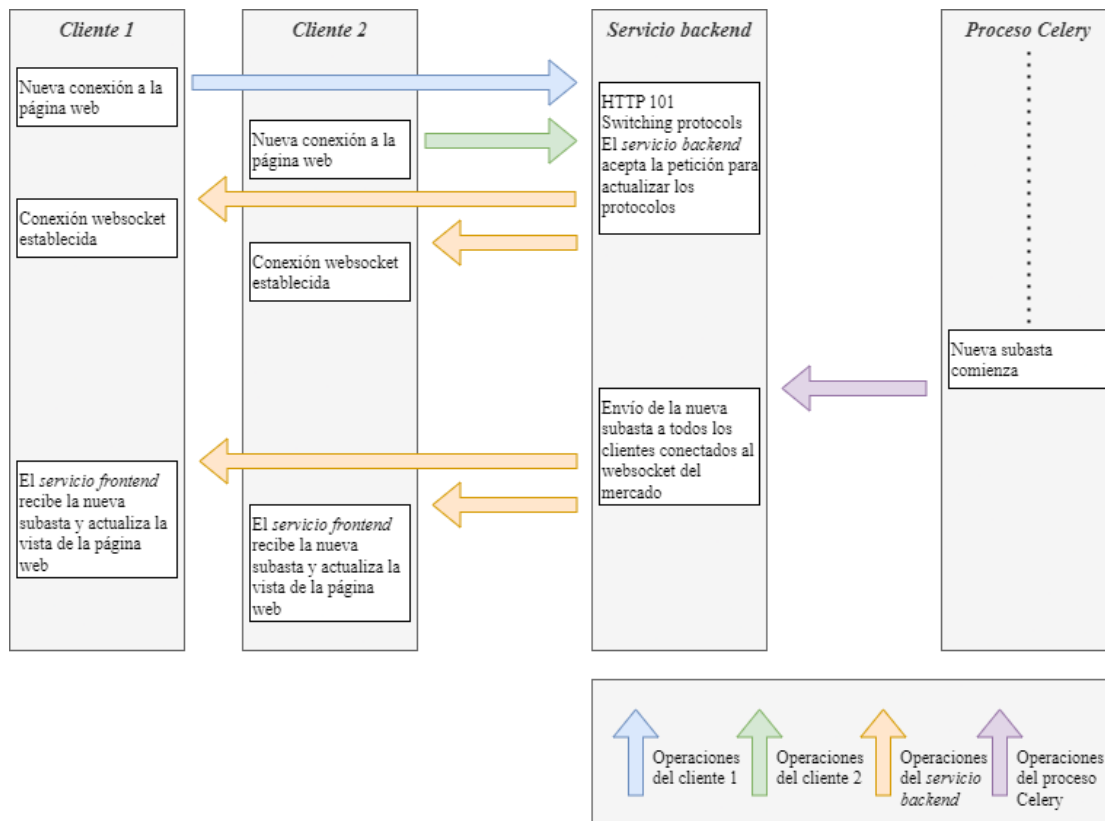


Ilustración 19: websocket del mercado

Por otra parte, dentro de la página web es posible visualizar subastas en detalle, es decir, una vista personalizada de cada subasta con el precio, título, descripción, horas de comienzo y fin, el precio inicial y la lista de pujas existentes en esa subasta. Cuando un usuario accede a esta vista, una conexión de tipo WebSocket es abierta para recibir notificaciones de las nuevas pujas creadas o actualizadas, en esa subasta en concreto. El flujo de ejecución podría verse de la siguiente manera, con dos clientes conectados a la página web:

1. Cliente 1 y cliente 2 acceden al detalle de la subasta con identificador único 1234 (este es un identificador de subasta de ejemplo, dentro de la aplicación los identificadores de subastas son *UUIDs*, del inglés *Universally Unique Identifier*, o en español, identificador único universal).
2. El cliente 2 decide pujar en la subasta con número 1234, por un valor de  $X$  euros, siendo  $X$  un numero decimal mayor que cero, mayor que el precio de salida de la subasta y mayor que el valor de la última puja de la subasta, si la hay.
3. Ambos clientes recibirán en sus navegadores el valor  $X$  de esta puja y el *servicio frontend* se ocupará de actualizar la vista de la página web. Si esa puja se ha

actualizado en lugar de crearse nueva, el servicio frontend se ocupará también de actualizar la tabla de pujas existente, reordenando todas las entradas.

Visto de manera visual, los clientes y servicios seguirán el siguiente proceso de ejecución (*Ilustración 20*).

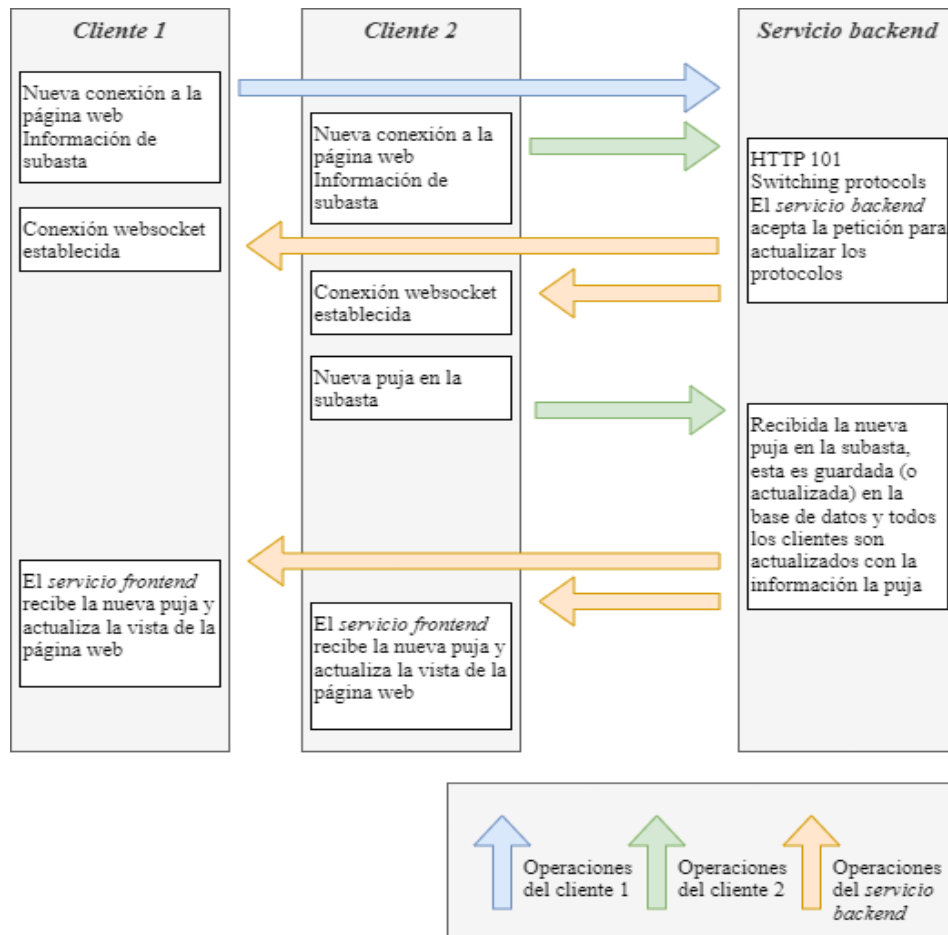


Ilustración 20: websocket de subasta

El paquete más relevante de este apartado es: [Websockets](#) en su versión 10.3: implementación del protocolo WebSocket (RFC 6455 y RFC 7692).

#### 5.4.7 Implementación de envío de correos electrónicos

Algunas de las diferentes operaciones que se ejecutan en la web tienen la posibilidad de enviar al usuario datos de la acción que ha realizado, por ejemplo, un correo electrónico de bienvenida cuando un usuario se da de alta en la página web. Para llevar

a cabo este proceso, precisamos de un servidor SMTP con credenciales, cuenta de usuario y contraseña, para enviar emails a través de internet.

Dentro de la carpeta que contiene el código del proyecto, en el fichero bajo la siguiente ruta `‘/api/app_mailer/sneder.py’` se encuentra la función genérica de la que hace uso el *servicio backend* para enviar correos electrónicos. En este fichero, comenzamos recuperando la plantilla HTML sobre la que se enviará el email. Para este fin, la función recibe el parámetro con el nombre `‘template’` de tipo texto (*str*), que hace referencia al nombre de la plantilla HTML. Si la plantilla que necesitamos se llama `‘welcome.html’`, el *servicio backend* busca dicho fichero, se renderiza el HTML con la función `‘.render()’` y la variable `‘email_data’`. Tomemos el ejemplo del documento HTML de bienvenida, el cual se encuentra dentro de la carpeta que contiene el código del proyecto, en el fichero bajo la siguiente ruta `‘/api/templates/welcome.html’`. Una llamada de ejemplo a la función `‘send_mail()’` contendría los siguientes valores, como ejemplo, dentro de la variable `‘email_data’`: `{ “email”: correo@ejemplo.com, “link”: link-de-activación }`. Observamos una correlación uno a uno dentro del fichero `‘welcome.html’` y la variable `‘email_data’` ya que en el documento `‘welcome.html’` encontramos varias variables rodeadas con los siguientes caracteres: `‘{{’` y `‘}}’`. Son precisamente, aquellas variables se encuentran también definidas dentro del diccionario `‘email_data’` que acabamos de mencionar. El resultado de aplicar la función `‘.render()’`, es la sustitución de estas variables dentro del fichero `‘welcome.html’` por aquellas definidas en los valores del diccionario `‘email_data’` en Python. Este es un ejemplo sencillo, sin embargo, se pueden llegar a aplicar clases CSS y otros elementos HTML para crear emails con más colores y/o estilos. En otro lugar, si se da el caso de no encontrar la plantilla `‘welcome.html’`, o cualquier otra, la ejecución del código terminará imprimiendo un error y evidentemente el correo electrónico no será enviado. Finalmente, el código continúa creando el email a enviar con: el asunto de email, el remitente, y el receptor. Haciendo uso de la librería [smtplib](#) y las variables de entorno definidas anteriormente se envía el email al receptor.

Los paquetes más relevantes de este apartado son: [email](#), contenido por defecto dentro de la librería estándar de Python. Su objetivo principal es generar objetos de tipo `‘text/*MIME’` y `‘multipart/*MIME’` donde se guardará el contenido del correo electrónico; [smtplib](#): contenido por defecto dentro de la librería estándar de Python. Su objetivo principal será enviar emails a través de internet; y [jinja2](#) en su versión 3.1.2, un motor

de plantillas, el cual permite de manera muy sencilla sustituir cadenas de texto dentro de, por ejemplo, documentos HTML, por otros valores.

#### 5.4.8 implementación de técnicas de búsqueda y filtrado de subastas

Toda subasta creada en la página web tiene dos campos descriptivos, un título y una descripción de lo que se está subastando. Puesto que tal vez el número de subastas sea elevado en la página web, el *servicio backend* ofrece diferentes técnicas de filtrado de subastas para que los usuarios puedan descartar aquellas que no necesiten o no estén interesados y así encontrar mejores resultados según sus criterios de búsqueda. El *servicio backend* dispone de cuatro tipos diferentes de filtros para las subastas, estos son los siguientes:

- Frecuencia de términos – Frecuencia inversa del documento o *TF-IDF*, en inglés, *Term Frequency – Inverse Document Frequency*.
- Indexación semántica latente o *LSI*, en inglés, *Latent Semantic Analysis*.
- Clasificadores Bayesianos o Naïve Bayes.
- Máquinas de vectores de soporte o *SVM*, en inglés, *Support Vector Machines*.

Ya que el título de una subasta puede llegar a ser una única palabra, las búsquedas se realizarán exclusivamente con la descripción de cada subasta. Cabe mencionar el término *Natural Language Processing*, también conocido como NLP. NLP es un campo de la lingüística, aplicada con técnicas de inteligencia artificial que permite a un sistema entender, interpretar, generar y clasificar el lenguaje humano. Presenta los siguientes objetivos:

- Entendimiento del texto.
- Traducción de un texto en lenguaje natural a un lenguaje entendido por una máquina.
- Extracción de información.
- Clasificación de textos o documentos.
- Generación de texto como respuesta.

Es una técnica muy conocida hoy en día, gracias a herramientas recientes como [ChatGPT](#), el cual emplea una combinación de pasos de preprocesado, entendimiento y generación de textos para proveer a los usuarios de conversaciones coherentes. Cabe mencionar que las técnicas empleadas para llevar a cabo técnicas del NLP se encuentran

hoy en día bajo procesos de mejora y optimización, ya que se podrían dar casos donde las respuestas generadas no fuesen del todo correctas, mal clasificadas o sin sentido alguno, debiéndose por lo general a los datos de entrenamiento de los modelos.

El primero de los métodos propuestos es TF-IDF. Se trata de una estadística numérica usada en técnicas como el minado de textos y la recuperación de información en un documento o serie de documentos, el cual evalúa la importancia de un fragmento de texto, auxiliar al documento o documentos. Es un método extremadamente útil para realizar tareas como el posicionamiento en buscadores, la clasificación de documentos y la recuperación de información. El propio método toma en cuenta dos partes diferentes, estas son:

- TF (*Term Frequency*): medición de un término dentro de un documento, calculando el número de veces que aparece ese término y normalizándolo contra el número total de términos. La idea es que aquellos términos con mayor frecuencia en el documento son más relevantes a ser más importantes que otros.
- IDF (*Inverse Document Frequency*): medición de la unicidad y raridad de un término en uno o varios documentos. Es calculado a partir de dividir el número total de documentos en la colección por el número de documentos que contienen el término. Esta parte del método penaliza aquellos términos que aparecen demasiadas veces en muchos documentos ya que proporcionan una cantidad de información menor.

$$w_{i,j} = tf_{i,j} \times \log \left( \frac{N}{df_i} \right)$$

$tf_{i,j}$  = number of occurrences of  $i$  in  $j$   
 $df_i$  = number of documents containing  $i$   
 $N$  = total number of documents

*Ilustración 21: fórmula TF-IDF*

Por cada búsqueda que se realice con este método, se devuelve una lista de subastas, conteniendo por cada una de ellas un número decimal, entre 0 y 1, que indica la relación entre la descripción de la subasta y el texto de búsqueda del usuario junto los datos de la propia subasta.

El segundo de los métodos es LSI. Este método es usado para analizar e indexar cadenas de texto o documentos basándose en su contenido semántico, midiendo la relación que existe entre las diferentes palabras y/o documentos. Se pueden realizar diferentes operaciones aplicando LSI como, por ejemplo, la medición de la similitud entre documentos, una agrupación de documentos parecidos o la recuperación de documentos similares partiendo de un texto inicial. LSI es extremadamente útil por su habilidad de extraer contenido de un cuerpo de un texto, estableciendo asociaciones entre aquellos términos que aparecen u ocurren en contextos similares. Usando una técnica matemática conocida como *Singular Value Decomposition* o SVD contra la matriz de documentos y términos, se consigue una similitud entre los diferentes textos. Durante la ejecución del método, se obtienen grupos de palabras que tienden a ocurrir con regularidad y tienen una relación semántica, y finalmente, se genera un índice que indica la relación con el documento original. Este índice varía entre los valores 0 y 1.

El tercero de los métodos es Naive Bayes. Se trata de uno de los métodos más antiguos jamás aplicados al campo del *Machine Learning*, inspirado por su creador Thomas Bayes. El teorema se basa en que no existe ninguna opción que sea correcta por completo, en su lugar, se seleccionará aquella con mayor probabilidad de que ocurra. El teorema de Bayes se utiliza para el cálculo de la probabilidad de un suceso, con información anterior sobre el mismo. Por ejemplo, partiendo de la existencia de dos sucesos A y B, trata de hacer inferencia del proceso B a partir de la información existente de A. También se busca la probabilidad condicionada de A sobre B. Los elementos del teorema son:

- $P(A)$ : probabilidad de suceso A.
- $P(B)$ : probabilidad de suceso B.
- $P(A|B)$ : probabilidad de A, teniendo en cuenta que sucede B.
- $P(B|A)$ : probabilidad de B, teniendo en cuenta que sucede A.

A su vez, existe la regla del producto, basada en el simple hecho de que, si los dos procesos A y B ocurren a la vez sin importar el orden, el producto puede escribirse como:

$$P(A*B) = P(A|B) * P(B)$$

$$P(B*A) = P(B|A) * P(A)$$

Con estas dos reglas, obtendríamos la fórmula del Teorema de Bayes:

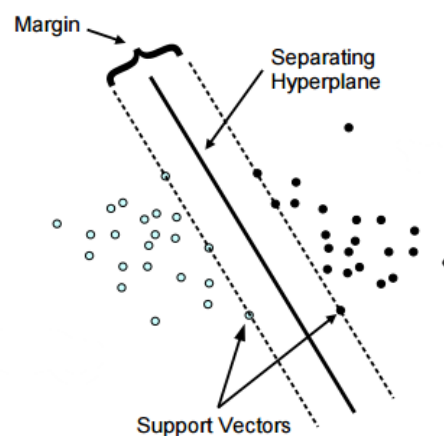
$$P(A|B) = (P(B|A) * P(A)) \div P(B)$$

Sin embargo, este es el teorema base, normalmente no solo se opera con únicamente dos sucesos. Por lo tanto, se define un conjunto  $X$ , que serán los datos que se desean clasificar, en este caso, el texto de búsqueda del usuario para el filtrado de subastas.

Finalmente, el ultimo método son los SVM. Se trata de un algoritmo del *Machine Learning* usado para tareas de clasificación y/o regresión. Basándose en la teoría del álgebra lineal, el método se centra en la idea de la separación de clases en una dimensión, hiperplano, dos dimensiones o un hiperplano de  $n-1$  cuando se tienen  $n$  dimensiones o categorías, con una línea recta. Esta técnica se emplea en varios campos, como pueden ser la clasificación de imágenes y de texto, en la bioinformática o en el mundo de las finanzas. Caben destacar los siguientes elementos cuando se trabaja con *Support Vector Machines*:

- El Hiperplano o *Hyperplane*: límites de decisión que separa un conjunto de puntos de datos con diferentes etiquetas.
- Los Vectores de Soporte o *Support Vectors*: son vectores formados por puntos de datos que se encuentran lo más cerca posible al Hiperplano separador.
- El Margen o *Margin*: es la separación entre las líneas creadas por los Vectores de Soporte.

En la siguiente imagen (*Ilustración 22*) podemos observar estos elementos.



*Ilustración 22: ejemplo de márgenes en SVM*

Además de los elementos anteriores, el método SVM es implementado usando un *Kernel* a través de una técnica denominada *Kernel Trick*. Se trata de una función capaz



de mapear la información de los datos en una dimensión donde estos sean separables. Además de esto, ayuda también a construir un clasificador más preciso. Existen una serie de *Kernels* para emplear, estos son:

- *Linear Kernel*: empleado cuando los datos pueden ser separados de manera directa usando una línea recta en el plano de clasificación.
- *Polynomial Kernel*: realiza una transformación de los datos a un espacio de dimensión superior usando funciones polinómicas. Es efectivo cuando los datos no tienen patrones lineales.
- *RBF Kernel*: del inglés, *Gaussian Radial Basis Function*. Lleva a cabo un mapeo de los datos a un espacio infinito dimensional usando una función Gaussiana. Es efectivo cuando los datos no son fácilmente separables y tienen relaciones complejas no lineales.
- *Sigmoid Kernel*: aplica la función sigmoide a los datos. Es útil para una clasificación binaria.
- *Custom Kernels*: existe la posibilidad de crear un Kernel propio para tratar cierto problema de clasificación. Depende de un problema específico, para cierto dominio en concreto.

Es importante para el ejemplo propuesto en este Trabajo de Fin de Grado, para clasificar un nuevo texto utilizando *Naive Bayes* o *Support Vector Machines*, los datos existentes han de encontrarse de alguna manera clasificados. Más tarde, el algoritmo recomienda una de estas etiquetas y devuelve todas las subastas que contienen dicha etiqueta en el campo “tags”.

Cabe mencionar el paquete [scikit-learn](#) en su versión 1.1.3, también conocido como *sklearn*, para el desarrollo de estos métodos. Se trata de un paquete ampliamente conocido a lo largo de la comunidad Python y en concreto aquella dedicada a la Inteligencia Artificial o el *Machine Learning*, ya que ofrece una amplia variedad de algoritmos y herramientas para llevar a cabo tareas como el procesado de datos, la extracción de características, selección de modelos y muchas más. También es importante la librería [NLTK](#) en su versión 3.8: composición de herramientas para el lenguaje natural.

#### 5.4.9 Desarrollo de vistas de la aplicación web

El proceso encargado de servir las vistas a los usuarios de la aplicación es el *servicio frontend*. El servicio utiliza el lenguaje TypeScript. Dentro de este lenguaje, existen al igual que en el *servicio backend*, una serie de frameworks para utilizar, estos son: [Angular](#), creado por Google. Está basado en TypeScript para el desarrollo de aplicaciones con impacto a gran escala con un largo número de componentes; [React](#), desarrollado por Facebook. Permite a los desarrolladores construir componentes reutilizables; y, finalmente, [Vue.js](#), en ocasiones mencionado como Vue. Denominado por sus creadores como “*The Progressive JavaScript Framework*”, se centra en la simplicidad y facilidades de uso. La curva de aprendizaje utilizando este framework es exponencial, provee al desarrollador con reactividad, es decir, la capacidad del propio framework de actualizar su capa de presentación en tiempo real cuando los datos de los componentes cambian. También, presenta un desarrollo basado en componentes reutilizables a lo largo de las diferentes vistas de la aplicación que puedan ser creadas. Vistas estas tres opciones, el *servicio frontend* hace uso del *framework* Vue.js en su versión Vue3 por los siguientes motivos: el buen soporte para código TypeScript, lo que implica mayor control en el código, es decir, las variables tienen un tipo de datos definido que una vez declarado no puede cambiar, la encapsulación de características de la aplicación en funciones, con la posibilidad de ser usadas a lo largo de toda la aplicación u otras aplicaciones y el buen sistema de reactividad entre componentes que presenta el *framework*. Además de esto, para los estilos de la web, se hace uso del *framework* de estilos [Quasar](#). Se trata de un *framework* de código abierto para construir SPAs (*Single Page Applications*), SSR (*Server Side Rendering*), PWAs (*Progressive Web Applications*) e incluso aplicaciones móviles. Se ha seleccionado esta tecnología por sus capacidades multiplataforma para construir tanto aplicaciones web como aplicaciones móviles, el largo set de diferentes componentes de estilos disponibles y la excelente documentación en su página web oficial.

Para que los usuarios accedan a la página se necesita un servidor web capaz de servir el *servicio frontend*. Con este fin, se hace uso de Nginx, un servidor web capaz de atender las peticiones de los clientes, el cual también puede actuar como proxy inverso o balanceador de carga. Este servidor es conocido por su rendimiento, alta escalabilidad y robustez. Nginx se carga a partir de un fichero de configuración para ejecutarse, donde se indican diferentes parámetros que determinan el comportamiento del servidor web.

Sigue una sintaxis específica, con aspectos como los contextos, que proveen niveles de configuración. Algunos de los elementos de configuración de Nginx más usados son: el tipo de recurso “*http*”, “*server*” o “*location*”; las directivas de control, que habilitan o deshabilitan funcionalidades además de configurar el permiso de hacer uso de conexiones SSL/TLS o WebSockets; y los bloques de servidores, los cuales permiten redireccionar tráfico a otras máquinas en el mismo u otro sistema, especificando la IP y puerto de estos sistemas subyacentes.

Al igual que el API, el *servicio frontend* requiere de una serie de variables de entorno para trabajar, estas son:

- *VITE\_BACKEND\_URL*: URL de destino donde el *servicio backend* se encuentra a la espera de peticiones. Si es desplegado en local, esta variable contendrá normalmente el valor <http://localhost:5000>.
- *VITE\_WS\_MARKETPLACE\_URL*: URL de destino donde el *servicio backend* se encuentra a la espera de nuevas conexiones de tipo WebSocket para que este envíe nuevas subastas cuando comienzan.
- *VITE\_WS\_AUCTION\_URL*: URL de destino donde el *servicio backend* se encuentra a la espera de nuevas conexiones de tipo WebSocket para que este envíe la creación o actualización de pujas en las subastas.
- *VITE\_WS\_ON*: campo booleano que indica si se deben abrir las conexiones WebSocket o no. Su principal objetivo es orientado a un entorno de desarrollo.

Para la gestión de sesiones de los usuarios se utilizan *cookies* de sesión. Estas *cookies* o *tokens* son normalmente una cadena de caracteres codificada, con información útil sobre el usuario que se encuentra con la sesión iniciada en un navegador. Cuando un usuario inicia sesión en la página web, es generado un *token* de sesión con diferentes parámetros de información. Al recibir este *token*, el *servicio frontend* lo guarda dentro del almacenamiento del navegador, con el nombre “*appforauctionsauth*” y tiene una vida útil de 60 minutos. Cada vez que una petición HTTP que requiera este *token* sea ejecutada contra el *servicio backend*, habrá que, evidentemente, enviar el *token* en la cabecera de la petición en el campo de autorización para verificar la identidad de cada usuario y el acceso a los datos.

Para gestionar las diferentes peticiones contra en *servicio backend* se ha usado el paquete [Axios](#). Se trata de una librería desarrollada en JavaScript que simplifica el proceso para realizar peticiones HTTP desde un navegador. Soporta una comunicación

HTTP asíncrona, algo que lo hace ideal para el proyecto. Tiene también la posibilidad de implementar una configuración personalizada, además de la inclusión de interceptores HTTP que pueden ejecutar código antes o después de enviar o recibir las diferentes peticiones.

El paquete [vue-router](#) permite crear rutas en la aplicación web, además de añadir la lógica de acceso a aquellas que requieren autenticación. Se trata de una librería para el *framework* Vue.js, que generara diferentes rutas asociadas a componentes de la propia web, típicamente llamadas vistas, o en inglés, *views*. Cuando un usuario navega a cierta ruta, esta es renderizada en el navegador y la URL es actualizada con aquella perteneciente a la ruta. La aplicación web tiene una serie de vistas para visualizar diferentes componentes. Se dividen dependiendo de, si la vista requiere que el usuario esté autenticado o no. Las pantallas que no requieren autenticación son las siguientes:

- Página de inicio o *Home Page*.
- Página de mercado.
- Página de inicio de sesión.
- Página para crear una nueva cuenta.
- Página para activar una cuenta.
- Página para visualizar una subasta.
- Página de filtro de subastas.
- Página de error, *404 Not Found*. En caso de que un usuario acceda a una ruta inexistente, es redireccionado automáticamente a esta página.

Por otra parte, aquellas rutas que sí que requieren autenticación, son las siguientes:

- Página de cierre de sesión.
- Página para visualizar los datos de la cuenta del usuario.
- Página para crear una nueva subasta.
- Página para actualizar una subasta existente.
- Páginas para visualizar el histórico de subastas, pujas y procesos de compra.
- Página para visualizar y/o crear carteras virtuales.

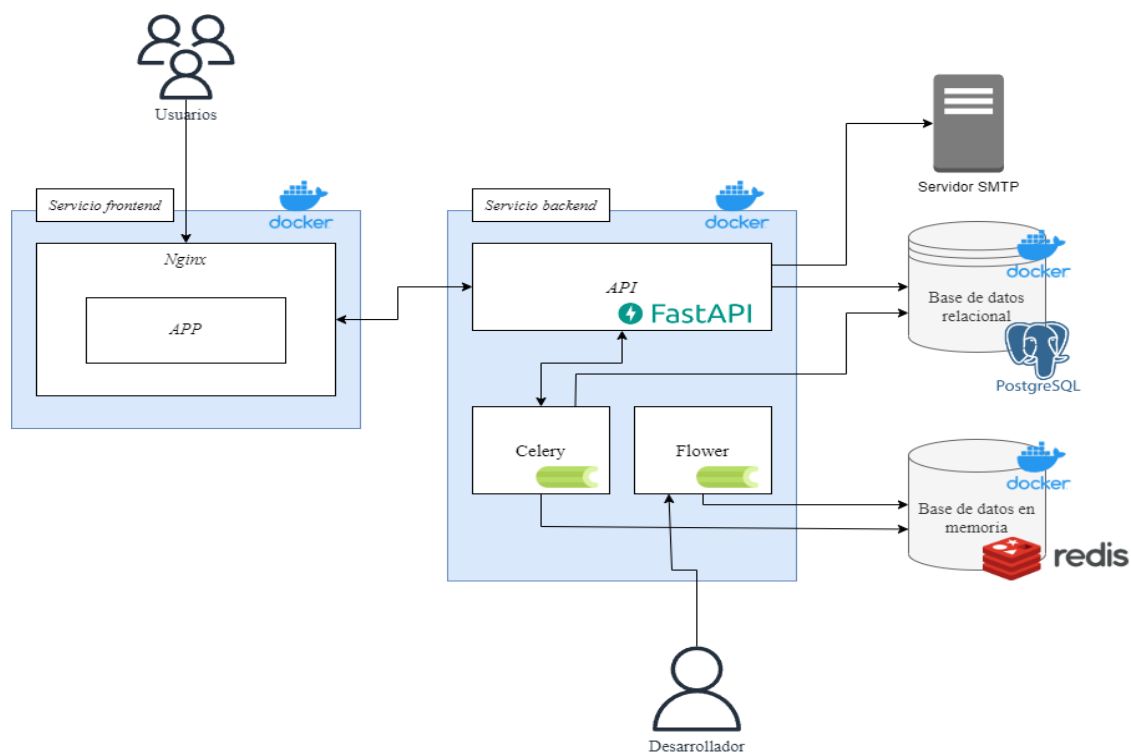
Evidentemente, un usuario autenticado podrá acceder a todas ellas.

Las librerías más relevantes de este apartado son las siguientes: [Vue](#) en su versión 3.0.0, *framework* basado en JavaScript para construir aplicaciones web interactivas; [Quasar](#) en su versión 2.6.0, *framework* de estilos para Vue y [Quasar-extras](#) en su versión 1.0.0, proporciona a Quasar una extensión de complementos como iconos, fuentes de texto y

animaciones CSS; [Pinia](#) en su versión 2.0.11, administrador de estado para aplicaciones Vue, donde guardar datos de carácter reactivos para reutilizarlos en múltiples vistas o funcionalidades de la aplicación web; [Vue-router](#) en su versión 4.0.0, permite navegación entre componentes de una aplicación web; y [Vuelidate](#) en su versión 0.7.7, validador de modelos de datos u objetos.

#### 5.4.10 Arquitectura final cliente-servidor

Implementados los anteriores puntos de desarrollo, visualizamos el sistema al completo y en detalle cómo podemos observar a continuación (*Ilustración 23*).



*Ilustración 23: arquitectura completa del sistema*

## 5.5 Pruebas del sistema

### 5.5.1 Recopilación de datos

Para el filtrado de las subastas, existe en el repositorio de código del Trabajo de Fin de Grado un fichero en formato *JSON* el cual contiene una serie de subastas con título,

descripción, precio inicial, etiquetas, fecha de comienzo y fecha de fin. Este fichero contiene alrededor de mil subastas de prueba y se encuentra bajo la ruta ``/api/sample_data/auctions.json``.

Dado que la cantidad de datos de las diferentes subastas es extremadamente grande para crearlos uno a uno, a través del método de *Web Scraping*, la gran mayoría de las subastas han sido obtenidas de la página web [www.liveauctioneers.com](http://www.liveauctioneers.com) con un fin exclusivamente de investigación.

El método de *Web Scraping* consiste en extraer información de una página web. Este método es usado para diferentes fines, como, por ejemplo: análisis de datos, investigación, agregación de contenidos o comparación de precios, entre muchos otros. Hay que seguir una serie de pasos previos antes de comenzar a usar esta técnica contra una página web. Estos pasos son los siguientes:

- Realizar un estudio y análisis de la página web objetivo, identificando los elementos que contienen la información buscada dentro del fichero HTML de la página.
- Seleccionar la librería para realizar la técnica de *Web Scraping*. En este caso usaremos [requests-html](#), una librería que permite renderizar código JavaScript y CSS dentro del fichero HTML. Además de la librería [json](#), provista por la librería estándar de Python.
- Escribir el código para realizar *Web Scraping* contra la página web, limpiar los datos obtenidos y guardarlos en un fichero o una base de datos.

Dentro de la carpeta que contiene el código del proyecto, bajo la siguiente ruta, `'/scraper'` podemos encontrar los diferentes ficheros para realizar estos pasos contra la página web objetivo. Cabe mencionar que podría darse el caso de que el código realizado para llevar a cabo la técnica de *Web Scraping* dejase de funcionar, ya que el código depende del propio HTML de la página web objetivo, es decir, si el código HTML cambia, habrá que cambiar o ajustar el código para recolectar datos.

El conjunto de datos utilizado para desarrollar los experimentos propuestos a continuación es una extensa lista de subastas. Existen un total de 57 diferentes etiquetas en las subastas existentes dentro de la base de datos. Algunas de estas etiquetas son las siguientes, en inglés: *“Plate”*, *“Rolex”*, *“Painting”*, *“Jewelry”*, *“Ring”*, *“Sword”* o *“Pokemon”*. No se van a enumerar todas las etiquetas ya que sería un listado demasiado extenso para incluir en la memoria.

Con estos datos, procedemos a realizar las búsquedas, utilizando los diferentes métodos propuestos. La siguiente enumeración de textos son ejemplos que podría introducir un usuario, junto con las etiquetas de mayor peso que se esperan que sean devueltas por las búsquedas. Todas estas búsquedas se realizarán exclusivamente en inglés, evidentemente sin las comillas:

- Frase 1: “*Pokemon Charizard old gold card*”. Su etiqueta esperada es: “pokemon”.
- Frase 2: “*Gold necklace eighteen karat gold*”. Su etiqueta esperada es: “jewelry collar”.
- Frase 3: “*Ferrari Gold car*”. Su etiqueta esperada es: “ferrari cars”.
- Frase 4: “*Gold medieval sword*”. Su etiqueta esperada es: “sword”.
- Frase 5: “*Old painting*”. Su etiqueta esperada es: “painting”.
- Frase 6: “*Porcelain decorative car*”. Su etiqueta esperada es: “car”.
- Frase 7: “*Basketball sticker card*”. Su etiqueta esperada es: “card”.
- Frase 8: “*Military style rolex watch*”. Su etiqueta esperada es: “rolex”.
- Frase 9: “*Porsche*”. Su etiqueta esperada es: “porsche cars”.
- Frase 10: “*Collector cards*”. Su etiqueta esperada es: “card”.
- Frase 11: “*Military badge*”. Su etiqueta esperada es: “world-war-2”.
- Frase 12: “*Vietnam propaganda poster*”. Su etiqueta esperada es: “vietnam”.
- Frase 13: “*Luke Skywalker*”. Su etiqueta esperada es: “star-wars”.
- Frase 14: “*Old porcelain decorative vase*”. Su etiqueta esperada es: “vase”.

### 5.5.2 Resultados

Una vez definidos los datos de prueba y los textos con los probar los métodos, procedemos a la ejecución y recopilación de los resultados. Durante las mediciones, se hará uso de diferentes métricas de evaluación para evaluar el comportamiento y rendimiento de los algoritmos propuestos.

El primero de los valores para medir, es lo que se denomina *Precision*, o en español exactitud. Se trata de una medición que evalúa cómo de precisa es la predicción del algoritmo, a partir de la siguiente fórmula:

$$Precision = \frac{Documentos\ relevantes\ recuperados}{Número\ total\ de\ documentos\ recuperados}$$

La segunda medición se conoce por el término *Recall*, o en español exhaustividad, también visto en ocasiones con el nombre de rellamada. Es un valor que indica la

proporción relevante de documentos recuperados, del total de documentos que son relevantes en el dataset. Si este valor es cercano a 1, esto indica que se han conseguido la mayoría de los documentos relevantes. Tiene la siguiente fórmula:

$$Recall = \frac{Documentos\ relevantes\ recuperados}{Documentos\ relevantes\ existentes\ en\ el\ dataset}$$

Finalmente, la última medición es el *F measure* o *F1-score*. Se trata de una media entre los dos últimos valores, *Precision* y *Recall*. Se podría definir también como la media armónica de ambos valores. El cálculo de este valor se obtiene a partir de la siguiente fórmula:

$$F1\ measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Todos los valores de las variables anteriores tomarán valores entre 0 y 1. Los resultados que se obtienen por cada frase utilizada son los siguientes, poniendo como ejemplo la Frase 1 (ver *Tabla 3*).

	TF-IDF	LSI	Naive Bayes	SVM
Precision	0,171	0,181	0,748	0,72
Recall	1,0	1,0	0,446	0,412
F1-score	0,291	0,307	0,434	0,397

*Tabla 3: ejemplo de mediciones de algoritmos*

Podemos observar diferentes comportamientos por parte de los algoritmos de búsqueda. Los dos primeros métodos, TF-IDF y LSI tienen una baja tasa de *Precision*, pero, sin embargo, una alta tasa de *Recall*. Esto se debe a que los algoritmos han devuelto todas las subastas que eran relevantes en la búsqueda, pero, devolviendo también un número de subastas no relevantes para el usuario, pero que se encuentran algo relacionadas con el texto de la búsqueda. Por otra parte, Naive Bayes y SVM tienen una mejor tasa en la variable *Precision*, pero, no han conseguido una buena tasa en la variable *Recall*, es decir, no son capaces de devolver todos los documentos relevantes con la búsqueda.



Partiendo de esta base, podemos obtener los valores para cada medición por frase y calcular las medias de los algoritmos para *Precision*, *Recall* y *F1-score* (Tabla 4).

	TF-IDF	LSI	Naive Bayes	SVM
Precision	0,378	0,385	0,725	0,769
Recall	0,769	0,701	0,444	0,481
F1-score	0,388	0,391	0,479	0,466

Tabla 4: resultados de búsquedas finales

Los valores obtenidos son parecidos en grupos dos a dos, es decir, los valores de TF-IDF son similares a aquellos obtenidos por LSI; mientras que los valores obtenidos por Naive Bayes son similares a los obtenidos por SVM. Observamos en esta figura (Tabla 4) que el primer grupo de algoritmos, TF-IDF y LSI, obtienen buenos resultados, por lo general, devolviendo aquellas subastas referentes con la búsqueda de los textos de prueba. Sin embargo, tienen una gran cantidad de otras subastas no referentes a este texto, ya que el valor de *Precision* es bajo, algo que podría afectar a la experiencia de usuario, devolviendo datos irrelevantes. Por otra parte, los algoritmos de Naive Bayes y SVM generan mejores valores para la variable *Precision*, lo que indica que devuelven información de mayor relevancia en comparación con los dos primeros.

Aunque los métodos presentan, por lo general, buenos resultados devolviendo un número de subastas razonable relacionadas con la búsqueda, debemos plantearnos finalmente la pregunta de qué algoritmo es más conveniente usar para el caso de uso presentado. La respuesta depende de dos factores, o cuestiones. Estas dudas son las siguientes; ¿el etiquetado de las subastas es fiable que lo hagan los usuarios? De ser así, incrementando el dataset y entrenando con nuevas subastas los modelos de Naive Bayes y SVM, ¿obtendríamos mejores resultados para estos métodos y podríamos usar las nuevas subastas creadas por los usuarios para alimentar el entrenamiento de los modelos?

Si la respuesta a la primera pregunta es no, entonces deberíamos de usar los dos primeros algoritmos, TF-IDF y LSI, ya que no podemos fiarnos de la correcta

clasificación de subastas por las etiquetas declaradas por los usuarios y, en este caso, no podríamos plantearnos la segunda pregunta. Por otra parte, si la respuesta a la primera pregunta es sí, habría que incluir en el dataset un número mayor de subastas para el entrenamiento de los modelos y así conseguir mejores predicciones, además de una retroalimentación de los modelos por parte de las nuevas subastas que se crearían en la web por los usuarios.

## 6. Conclusiones generales

El primer logro para destacar después de realizar la recolección de requisitos, implementación y pruebas, son los conocimientos adquiridos en materias desde la gestión de proyectos de software hasta la propia implementación y puesta en marcha del código en un entorno local. Cabe mencionar el aumento de conocimiento en los lenguajes de Python y TypeScript.

También, es notable el aprendizaje obtenido utilizando paquetes de terceros y su implementación en el código, en algunas ocasiones teniendo que leer extensiva documentación de estos.

Otro punto para tener en cuenta es la implementación de las técnicas de búsquedas empleadas, relacionadas con el *Machine Learning*, algo que ha llevado una gran cantidad de tiempo para ser teóricamente comprendidos e implementados.

Por último, hay que destacar los conocimientos adquiridos de procesos paralelos, trabajando entre sí para conseguir un bien común como el que se presenta en este Trabajo de Fin de Grado, que es, ni más ni menos, proporcionar una buena experiencia de usuario y una página web inmersiva con actualizaciones a tiempo real para los posibles clientes.

Como nuevas funcionalidades podría ser interesante los siguientes cambios: **I.** desde el punto de vista de rendimiento y arquitectura, separar el servicio de websockets a un nuevo proceso dedicado exclusivamente a ello, completamente aislado del *servicio backend*, para reducir la carga de trabajo en el servicio. **II.** También, se podría incluir una verificación en dos pasos de autenticación, por seguridad y al tratarse de una página web donde los usuarios operan con sus propias cantidades de dinero, es de vital importancia, si se lleva a un entorno productivo esta aplicación, el proporcionar una seguridad extra durante los inicios de sesión. Tal vez, este punto podría apoyarse en el

mismo proceso que presenta el cambio de email ya implementado en la aplicación, donde el usuario recibe un token para cambiar dicha dirección de correo electrónico.

**III.** Además de esto, podría ser interesante explorar los inicios de sesión con otras plataformas, como pueden ser Google o Facebook. **IV.** Desplegar la aplicación en algún proveedor *cloud* como son *Azure*, *AWS* o *GCP*, sin embargo y por los costes de estas plataformas, este punto no se ha llevado a cabo. **V.** En último lugar, también cabe la posibilidad de integrar la aplicación con pasarelas de pago como PayPal o Stripe, para realizar compras en la aplicación.

## 7. Referencias

En este punto se proporciona documentación extra para la evaluación de este Trabajo de Fin de Grado. El repositorio de código puede encontrarse en el siguiente [enlace](#) o a través de la siguiente url: <https://gitlab.com/AlvaroMartinezQ/app4auctions>.

[1] Carlos Álvarez López (2021). TronAuction: plataforma de subastas online utilizando Smart Contracts.

<https://gredos.usal.es/bitstream/handle/10366/149883/completa.%20ALVAREZ%20LOPEZ%2C%20CARLOS.pdf?sequence=1&isAllowed=y>

[2] Klein, Stefan y Keefe, Robert (1999). The Impact of the Web on Auctions: Some Empirical Evidence and Theoretical Considerations.

[3] Term Frequency – Inverse Document Frequency.

<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>

[4] Sergio Andreu Díez (2018/2019). Las subastas online.

<http://dspace.umh.es/bitstream/11000/7444/1/TFG-Andreu%20D%C3%ADez%2C%20Sergio.pdf>

[5] Eu Jin Lok (2017). TF-IDF para identificar la señal del sonido.

<https://mungingdata.wordpress.com/2017/11/25/episode-1-using-tf-idf-to-identify-the-signal-from-the-noise/>

[6] Múltiples autores. Latent Semantic Indexing.

[https://en.wikipedia.org/wiki/Latent\\_semantic\\_analysis](https://en.wikipedia.org/wiki/Latent_semantic_analysis)

[7] Elika Dizechi (2020). What is Latent Semantic Indexing and why does it matter for your SEO Strategy?

[https://blog.hubspot.com/marketing/what-is-latent-semantic-indexing-why-does-it-matter-for-your-seo-strategy#:~:text=Latent%20Semantic%20Indexing%2C%20also%20known,singular%20value%20decomposition%20\(SVD\).](https://blog.hubspot.com/marketing/what-is-latent-semantic-indexing-why-does-it-matter-for-your-seo-strategy#:~:text=Latent%20Semantic%20Indexing%2C%20also%20known,singular%20value%20decomposition%20(SVD).)

[8] Múltiples autores. Naive Bayes Classifier.

[https://en.wikipedia.org/wiki/Naive\\_Bayes\\_classifier](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

[9] Múltiples autores. Naive Bayes.

[https://scikit-learn.org/stable/modules/naive\\_bayes.html](https://scikit-learn.org/stable/modules/naive_bayes.html)

[10] Múltiples autores. Support Vector Machine.

[https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)

[11] Múltiples autores. Support Vector Machines.

<https://scikit-learn.org/stable/modules/svm.html>

[12] R. Kusumawati (2019). Comparison performance of Naive Bayes Classifier and Support Vector Machine Algorithms for Twitter's classification of Tokopedia Services.

<https://iopscience.iop.org/article/10.1088/1742-6596/1320/1/012016/pdf>

[13] Múltiples autores, Baeldung (2023). Comparing Naive Bayes and SVM for text classification.

<https://www.baeldung.com/cs/naive-bayes-vs-svm>

- [14] Rohit Gandhi (2018). Support Vector Machine – Introduction to Machine Learning Algorithms.  
<https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- [15] ShortHills Tech (2022). Django vs FastAPI: A Detailed Comparison.  
[https://medium.com/@ShortHills\\_Tech/django-vs-fast-api-a-detailed-comparison-df8d00f3c3b2](https://medium.com/@ShortHills_Tech/django-vs-fast-api-a-detailed-comparison-df8d00f3c3b2)
- [16] Charlie Gilman (2021). 7 Reasons why to migrate from Vuetify to Quasar Framework  
<https://medium.com/@charliegilmanuk/7-reasons-to-migrate-from-vuetify-to-quasar-framework-8ea47358262>
- [17] Santiago Lorenzo (2021). Configurando un pipeline con Gitlab CI/CD.  
<https://somospt.com/blog/232-configurando-un-pipeline-cicd-con-gitlab-ci-cid>
- [18] Martin Shubik (1983). *Auctions, bidding and contracting*.  
<https://web.stanford.edu/~milgrom/publishedarticles/Auctions,%20Bidding,%20and%20Contracting,%20Uses%20and%20Theory,%201983.pdf>
- [19] Juan José Martín Guareño (2018). Support vector regression: propiedades y aplicaciones.  
<https://idus.us.es/bitstream/handle/11441/43808/Mart%C3%ADn%20Guare%C3%B1o%20Juan%20Jos%C3%A9%20TFG.pdf?sequence=1&isAllowed=y>
- [20] José Manuel Medrano Martínez (2018). Aplicación de técnicas de Machine Learning para la predicción de resultados deportivos.  
[https://ebuah.uah.es/dspace/bitstream/handle/10017/33860/TFG\\_Medrano\\_Martinez\\_2018.pdf?sequence=1&isAllowed=y](https://ebuah.uah.es/dspace/bitstream/handle/10017/33860/TFG_Medrano_Martinez_2018.pdf?sequence=1&isAllowed=y)

## 8. Anexos

### 8.1 Manual de instalación

Para lanzar la aplicación en un entorno local y usar sus servicios, se requiere de los siguientes puntos:

- Conexión a Internet, ya que algunos paquetes actualizan dependencias a la hora de iniciar los servicios, y, por supuesto, los paquetes de cada servicio han de ser instalados desde sus respectivos repositorios.
- Que los puertos 3000, 5000, 5555, 5432 y 6379 del computador no estén vinculados a ningún otro servicio.
- Docker (es posible no usarlo, pero se tendría que desplegar cada servicio uno a uno).

Dentro de la carpeta del código del proyecto, en el fichero bajo la ruta `‘/README.md’` podemos encontrar las instrucciones de despliegue de la aplicación. Sin embargo, bastará con:

1. Iniciar Docker en el sistema.
2. Ejecutar el comando ``docker-compose up -d --build`` dentro de la raíz de la carpeta que contiene el código del proyecto, para descargar las imágenes base, construir las imágenes de los servicios y ejecutarlas en segundo plano (o devolver el flujo de ejecución a la línea de comandos desde la que se ejecuta la operación).

Después de realizar estos pasos, podremos acceder a los diferentes servicios:

- `http://localhost:3000` → Aplicación web.
- `http://localhost:5000` → Especificación OpenAPI.
- `http://localhost:5555` → Panel de control Flower.
- `http://localhost:5432` → Puerto de Postgres.
- `http://localhost:6379` → Puerto de Redis.

Cabe mencionar que el envío de correos electrónicos no está activo en este despliegue y precisa de una serie de variables de entorno para hacer uso de este. Estas variables son las siguientes:

- `MAILER = True`.

- *MAILER\_USER* = la dirección de correo electrónico desde donde se enviarán los emails.
- *MAILER\_PASSWORD* = la contraseña de la dirección de correo.
- *MAILER\_HOST* = proveedor del servicio de correo electrónico.
- *MAILER\_PORT* = puerto de escucha del proveedor del servicio de correo electrónico.

## 8.2 Manual de usuario

Algunas de las operaciones más comunes que un usuario puede realizar en el sistema son las siguientes:

- Creación de usuario.
- Creación de subasta.
- Exploración de subastas.
- Cambio de datos personales de un usuario.

Las describiremos a continuación en los siguientes apartados.

### 8.2.1 Creación de nuevo usuario

Para crear una nueva cuenta o nuevo usuario en la página web, habrá que navegar a la ruta del *servicio frontend* siguiente, suponiendo que el servidor se encuentra desplegado en un entorno local: `https://localhost:3000/account/new`. Una vez introducidos los campos, la vista de la página web quedará como se ve a continuación (*Ilustración 24*).

The screenshot shows a web form titled "Create a new account" for "APP4AUCTIONS". The form is contained within a white box with a light gray border. At the top left of the form is a small icon of a scale and the text "APP4AUCTIONS". The form fields are as follows:

- Name: Alvaro
- Surname(s): Martinez
- Email: [Redacted]
- Contact number: +34 652652652
- Identification number DNI/NIE/other: 50505050J
- Address: Non existent street number 19
- Country: Spain
- Password: [Redacted]
- Repeat password: [Redacted]
- Age:  18+ (checked)
- Date of birth: 1999/01/01

At the bottom right of the form is a button with a plus sign and the text "Create".

Below the form, there is a dark blue footer bar. On the left, it says "v0.0.1". On the right, there is a circular icon with a double arrow pointing right.

Ilustración 24: creación de usuario

Se ha eliminado el email de la imagen anterior (*Ilustración 24*) ya que se trata de un email personal, para más tarde mostrar el correo para activar la cuenta.

Una vez creado el usuario, se muestra un mensaje informando que en breves instantes recibirá un correo para activar la cuenta y solo después de haber realizado dicho proceso, podrá iniciar sesión. Este correo contiene una ruta de la página web, en caso de encontrarse el *servicio backend* con el servicio de envío de emails activado, de no ser así, imprimirá una ruta del propio *servicio backend* donde activar la cuenta del usuario.

Una vez activada la cuenta el usuario puede iniciar sesión, habrá que navegar a la ruta del *servicio frontend* siguiente, suponiendo que el servidor se encuentra desplegado en un entorno local: ``https://localhost:3000/login``. Introducidos y validados los datos, la aplicación recibe el *token* de sesión para el usuario, y este es redireccionado a la página de inicio, como se muestra a continuación (*Ilustración 25*)



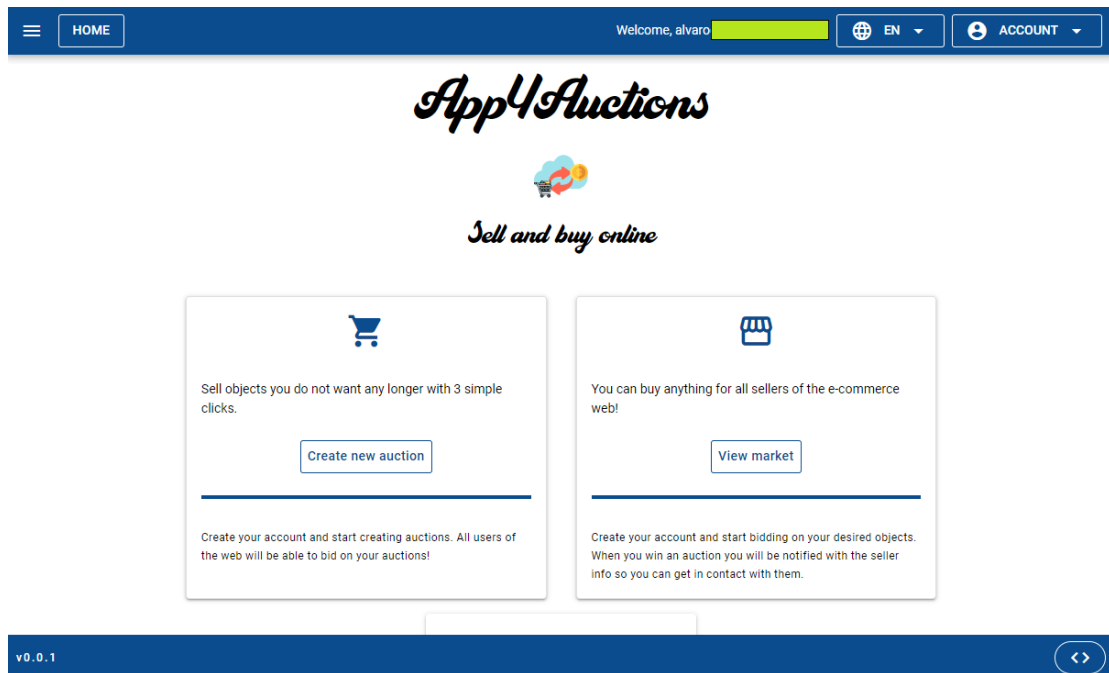


Ilustración 25: página de inicio

En este momento, el usuario es capaz de realizar operaciones como, por ejemplo, ver los datos de su perfil en la página web (Ilustración 26).

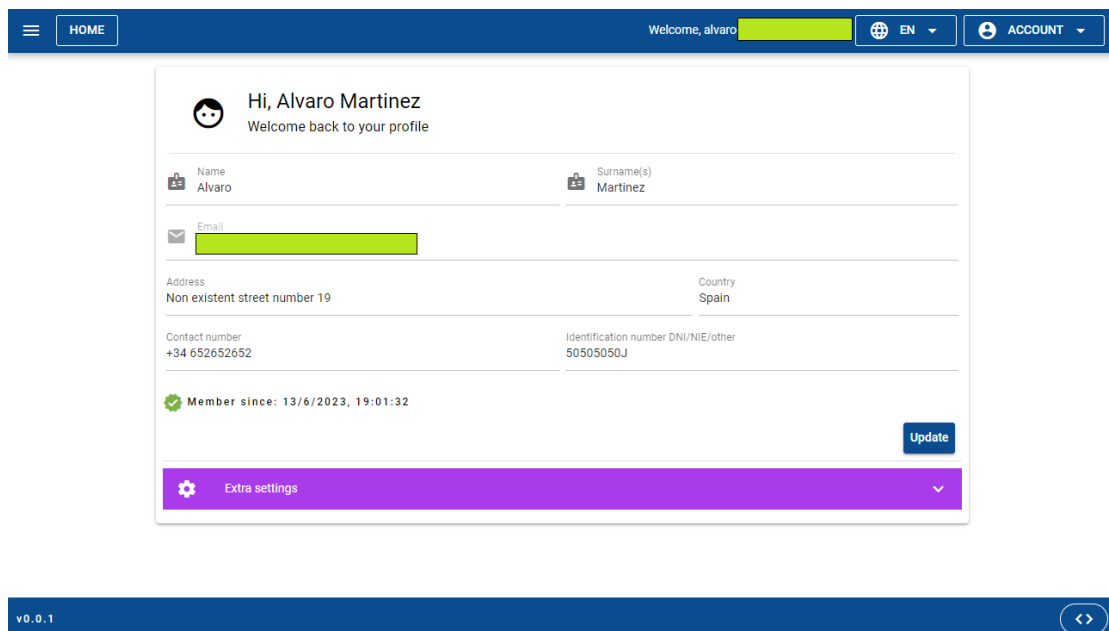


Ilustración 26: datos de usuario

## 8.2.2 Creación de una subasta

Para crear una nueva subasta, habrá que navegar a la ruta del *servicio frontend* siguiente, suponiendo que el servidor se encuentra desplegado en un entorno local: ``http://localhost:3000/auction/new`` (*Ilustración 27*).

The screenshot displays the 'Create a new auction' interface. The header is dark blue with a 'HOME' button, a user greeting 'Welcome, alvaro', a language selector 'EN', and an 'ACCOUNT' dropdown. The left sidebar, titled 'Quick links', contains several menu items with icons: Home (Start page), Market (Browse all active auctions), My auctions (View your auctions), My bids (View your bids), My wallets (View your wallets), My buy processes (View your buys on the web), My sell processes (View your sells on the web), and New auction (Create a new auction). The main content area is titled 'Create a new auction' and includes a title field, a description field, input fields for 'Initial price' and 'Price currency', 'Start time' and 'Finish time' with calendar icons, and a 'Tags' field. A green 'Create' button is at the bottom right. The footer shows 'v0.0.1' and navigation arrows.

*Ilustración 27: página nueva subasta*

También se puede observar en la imagen anterior (*Ilustración 27*), el menú lateral de las diferentes rutas de la aplicación disponibles para el usuario autenticado (este menú se compone de una lista de rutas reducida y algunos diferentes enlaces para un usuario no autenticado).

Una vez creada una subasta, el usuario puede visualizarla en la vista de *mis subastas*, además de modificarla en caso de que no haya llegado aún su fecha de inicio (*Ilustración 28*).

Quick links

- Home  
Start page
- Market  
Browse all active auctions
- My auctions  
View your auctions
- My bids  
View your bids
- My wallets  
View your wallets
- My buy processes  
View your buys on the web
- My sell processes  
View your sells on the web
- New auction  
Create a new auction

### Your auctions

Your can only edit an auction if it has not started yet

[+ New auction](#)

Auction	Initial price	Highest bid	Price currency	Start time	Finish time	Actions
Auction test	50	None	euro	31 Jul 2023 13:00	31 Aug 2023 13:00	

Records per page: 10 1-1 of 1

v0.0.1

Ilustración 28: lista de subastas de un usuario

### 8.2.3 Exploración de subastas

Para visualizar las subastas activas en la web, habrá que navegar a la ruta del *servicio frontend* siguiente, suponiendo que el servidor se encuentra desplegado en un entorno local: ``http://localhost:3000/auction/new`` (Ilustración 29).

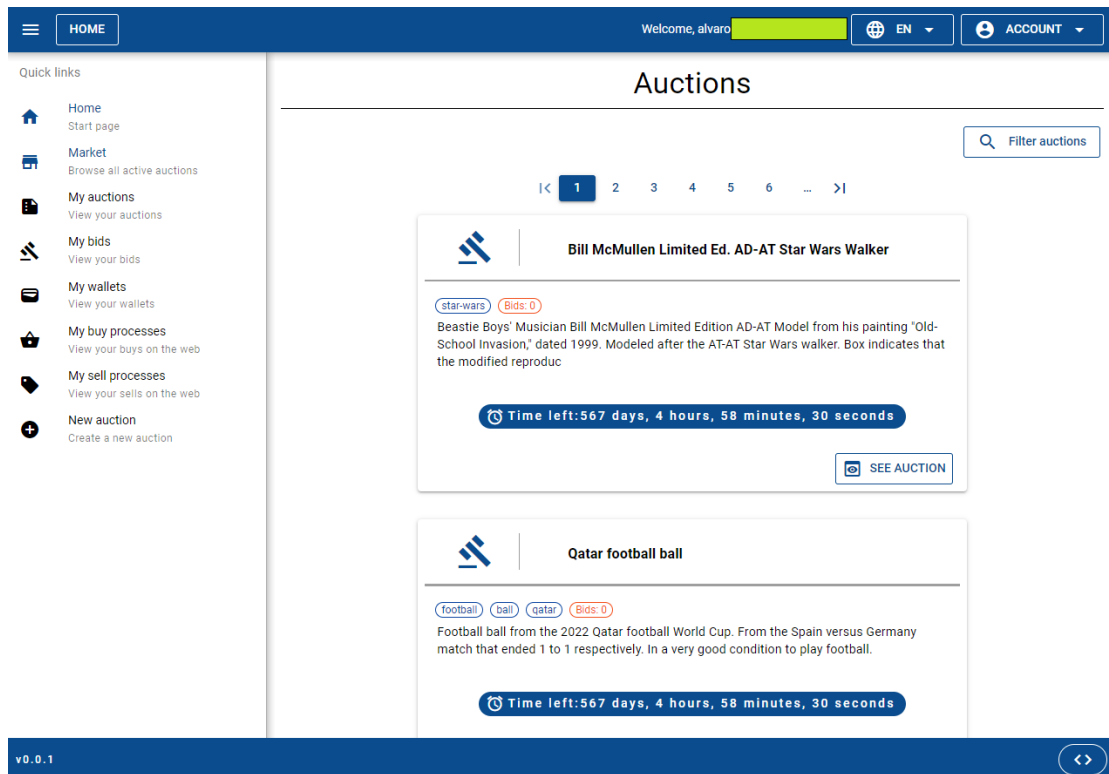


Ilustración 29: mercado

Para filtrar subastas en la web, habrá que navegar a la ruta del *servicio frontend* siguiente, suponiendo que el servidor se encuentra desplegado en un entorno local: `http://localhost:3000/auction/search` y seleccionar un método de búsqueda (TF-IDF, LSI, Naive Bayes o SVM) además de las palabras clave que se quieren buscar (*Ilustración 30*).

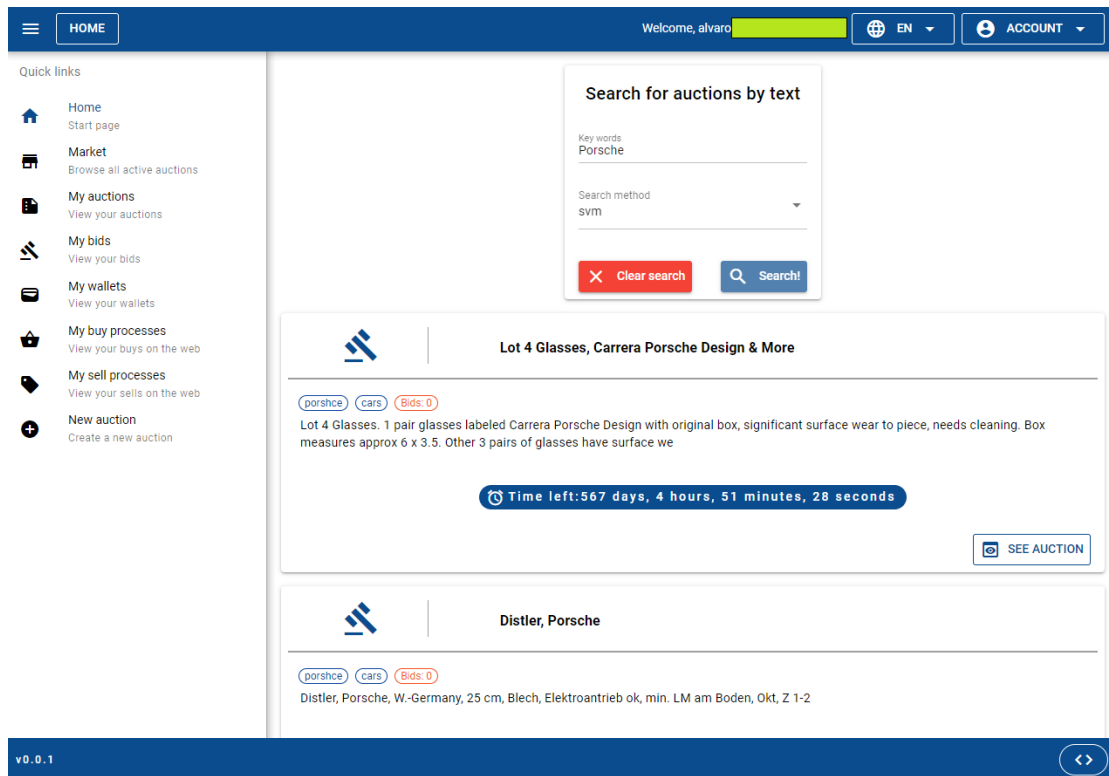


Ilustración 30: listado de subastas con filtro

Como podemos observar, se renderizan los resultados reutilizando la vista de la *Ilustración 29*, sin la paginación de subastas.

#### 8.2.4 Cambio de datos de usuario

Para actualizar los datos de un usuario, habrá que navegar a la ruta vista en la *Ilustración 26*. Algunos de los campos de usuario se pueden actualizar directamente en esta página, sin embargo, el email y la contraseña requieren un proceso dedicado para actualizar los datos.

Para cambiar la contraseña habrá que introducir la contraseña nueva (dos veces por seguridad) además de la antigua para validar la autenticidad de esta operación. Una vez cambiada, se cierra la sesión automáticamente y se redirecciona al usuario a la página de inicio de sesión.

En caso de desear cambiar el email, el usuario solicita este cambio en la misma pestaña. Tendrá que introducir su email como paso previo para solicitar el cambio, sin abandonar la página web cuando se realice este paso. Una vez introducido, se genera una cadena de texto aleatoria y esta se guarda dentro de una columna en el objeto usuario en la base

de datos, además de enviarle un email con la cadena de texto. En paralelo a este proceso, se solicita en la página web que el usuario introduzca la cadena de texto y la nueva dirección de email. En caso de introducir correctamente la cadena de texto (que concuerde con aquella guardada en la base de datos), se actualiza el email del usuario y se cierra su sesión, redireccionándole también a la página de inicio de sesión.

©2023 Álvaro Martínez Quiroga

Algunos derechos reservados

Este documento se distribuye bajo la licencia "Atribución 4.0 Internacional" de Creative Commons,

disponible en: <https://creativecommons.org/licenses/by/4.0/deed.es>