UNIVERSIDAD DEL VALLE

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

“TALKATIVE”

ESTUDIANTES:

Alejandro Marshel Ayala Orosco

Juan Pablo Menacho Castro

Patrick Fernando Martínez Moscoso

Geomara Leslie Castillo Cordero

DOCENTE: ING. JOSE ARROYO SANTA CRUZ

GRUPO: “A”

La Paz - Bolivia
2024

1 INTRODUCCIÓN

Con el paso de los años los sistemas informáticos empezaron a tener problemas por el gran volumen de datos que iba en aumento, esto provocaba latencia en la comunicación, falta de integridad de datos y falta de sincronicidad en los tiempos de registros.

El sistema de mensajería propuesto, abarcara las soluciones a los problemas ya mencionados, con la arquitectura de un sistema distribuido, mejorando así el rendimiento, disponibilidad e integridad de datos.

2 OBJETIVO

Desarrollar un sistema de mensajería distribuido con videollamada, que pueda abarcar con el aumento potencial de datos, sin que este falle.

3 REQUISITOS DEL SISTEMA

Los requisitos para poder utilizar el sistema son los siguientes:

REQUISITOS MÍNIMOS

- **CPU**: Intel Core i3 o AMD Ryzen 3.
- **Memoria (RAM)**: 4 GB de RAM.
- **Almacenamiento**: HDD de 500 GB o SSD de 128 GB.
- **Tarjeta Gráfica**: Gráfica integrada (Intel HD Graphics o AMD Radeon Vega).
- **Monitor**: Monitor HD (1366x768).
- **Sistema Operativo**: Windows 10 Home.
- **Conectividad**: Conexión a internet estable (al menos 10 Mbps).

REQUISITOS RECOMENDADOS

- **CPU**: Intel Core i5 o AMD Ryzen 5.
- **Memoria (RAM)**: 8 GB de RAM.
- **Almacenamiento**: SSD de 256 GB.
- **Tarjeta Gráfica**: Gráfica integrada o dedicada de gama baja (por ejemplo, NVIDIA GeForce MX150).
- **Monitor**: Monitor Full HD (1920x1080).
- **Sistema Operativo**: Windows 10 Pro o superior.
- **Conectividad**: Conexión a internet estable (al menos 20 Mbps).

4 BASE DE DATOS.

- **USUARIOS**

NOMBRE CAMPO	TIPO DATO	DESCRIPCIÓN
ID_USUARIO	OBJECTID	Identificar del usuario.
NOMBRE_DE_USUARIO	STRING	Nombre del usuario en el sistema
NOMBRE	STRING	Nombre
APELLIDO	STRING	Apellido del usuario
CORREO	STRING	Dirección de correo electrónico del usuario
CONTRASEÑA	STRING	Contraseña para la autenticación del usuario
FECHA DE REGISTRO	DATE	Fecha en la que el usuario se registro

5 TECNOLOGÍAS UTILIZADAS.

- **VISUAL STUDIO CODE:** Visual Studio Code es un entorno de desarrollo integrado (IDE) que se centra en la simplicidad y la productividad. Desarrollado por Microsoft, es conocido por su interfaz de usuario minimalista y altamente personalizable. Ofrece una amplia gama de características para mejorar el flujo de trabajo de desarrollo, como resaltado de sintaxis, finalización automática de código, navegación inteligente, integración con herramientas de control de versiones como Git, capacidad de depuración integrada y una extensa biblioteca de extensiones que permiten a los desarrolladores adaptar el IDE a sus necesidades específicas. Además, es multiplataforma, compatible con Windows, macOS y Linux, lo que lo hace accesible para una amplia variedad de desarrolladores.

- **LARAVEL:** Laravel es un framework de desarrollo web de código abierto y basado en PHP, conocido por su elegancia y facilidad de uso. Diseñado para permitir la creación rápida de aplicaciones web, Laravel ofrece una sintaxis expresiva y clara que facilita el desarrollo de aplicaciones robustas y escalables. Entre sus características destacadas se incluyen un sistema de enrutamiento intuitivo, un ORM (Object-Relational Mapping) llamado Eloquent para interactuar con la base de datos de forma sencilla, un potente sistema de plantillas llamado Blade, autenticación integrada, migraciones de base de datos, programación de tareas en segundo plano y una comunidad activa que proporciona una amplia variedad de complementos y recursos para facilitar el desarrollo. Con una curva de aprendizaje relativamente baja y una documentación completa, Laravel es una opción popular entre los desarrolladores para la creación rápida y eficiente de aplicaciones web modernas.

- **GIT:** Git es un sistema de control de versiones distribuido, diseñado para manejar proyectos de software de cualquier tamaño con eficiencia y facilidad. Desarrollado por Linus Torvalds en 2005, Git se ha convertido en uno de los sistemas de control de versiones más populares y ampliamente utilizados en la industria del desarrollo de software. Su funcionamiento se basa en la creación de una serie de instantáneas (snapshots) de un proyecto en diferentes puntos en el tiempo, lo que permite a los desarrolladores trabajar de forma colaborativa y controlar los cambios realizados en el código fuente. Además, Git ofrece características como ramificación (branching) y fusión (merging), que facilitan la gestión del desarrollo paralelo de características y la integración de cambios entre diferentes versiones del código. Una de las principales ventajas de Git es su naturaleza distribuida, lo que significa que cada desarrollador tiene una copia completa del repositorio en su propia máquina, lo que permite un desarrollo ágil y descentralizado. Además, Git es altamente flexible y se puede integrar fácilmente con una amplia gama de herramientas y servicios de desarrollo.

- **GIT HUB :** GitHub es una plataforma de desarrollo colaborativo que utiliza Git como sistema de control de versiones. Permite a los desarrolladores alojar y revisar código, gestionar proyectos, realizar seguimiento de problemas y colaborar en equipos de desarrollo de software. GitHub proporciona herramientas adicionales, como seguimiento de problemas, solicitudes de extracción y wikis, que facilitan la colaboración y la gestión de proyectos de software de manera eficiente. Es una herramienta esencial para la comunidad de desarrollo de software y fomenta la contribución y el trabajo en equipo en proyectos de código abierto y privados.

- **NODE.JS :** Node.js es un entorno de ejecución de JavaScript basado en el motor V8 de Chrome. Permite a los desarrolladores ejecutar código JavaScript fuera del navegador, lo que lo convierte en una opción popular para el desarrollo de aplicaciones del lado del servidor y herramientas de línea de comandos. Con Node.js, los desarrolladores pueden crear aplicaciones escalables y de alto rendimiento utilizando un lenguaje de programación unificado tanto en el cliente como en el servidor. Node.js cuenta con un vasto ecosistema de módulos y bibliotecas disponibles a través de npm (Node Package Manager), lo que facilita la construcción de aplicaciones complejas de manera eficiente. Es una tecnología ampliamente adoptada en la comunidad de desarrollo web y es una opción poderosa para la creación de aplicaciones modernas y rápidas.

- **CHATIFY:** Chatify de Munafio es un repositorio de código abierto enfocado al desarrollo de aplicaciones de mensajería instantánea para proyectos Laravel utilizando Pusher. Proporciona características como salas de chat públicas y privadas, mensajes directos, emojis, y opciones de personalización. Chatify se utiliza en una variedad de contextos, incluyendo aplicaciones de mensajería empresarial, plataformas de soporte al cliente y comunidades en línea. Es una herramienta versátil que facilita la comunicación y la colaboración entre individuos y grupos, ya sea para trabajar en proyectos, socializar o brindar soporte.

- **WEBSOCKETS:** Los WebSockets son una tecnología que proporciona comunicación bidireccional y en tiempo real entre un cliente y un servidor a través de una única conexión persistente. Permiten una interacción más dinámica en aplicaciones web al permitir que los datos se envíen y reciban de forma instantánea, sin la necesidad de recargar la página. Cuando se implementan en una aplicación, los WebSockets facilitan características como chats en tiempo real, actualizaciones en vivo, juegos multijugador y colaboración en tiempo real. Son útiles en situaciones donde se necesita una comunicación instantánea y continua entre cliente y servidor.

- **FIREBASE:** Firebase es una plataforma de desarrollo de aplicaciones móviles y web proporcionada por Google. Ofrece una variedad de herramientas y servicios que facilitan la creación y gestión de aplicaciones, como bases de datos en tiempo real, autenticación de usuarios, almacenamiento de archivos, alojamiento de aplicaciones web, análisis de uso y más. Firebase permite a los desarrolladores centrarse en la creación de la experiencia de usuario y la lógica de la aplicación, mientras que la infraestructura y los servicios de backend están gestionados por Firebase.

- **FASTAPI:** FastAPI es un framework web moderno y de alto rendimiento para construir APIs con Python. Es conocido por su rapidez y eficiencia, así como por su facilidad de uso y la capacidad de generar automáticamente documentación interactiva para las APIs. FastAPI aprovecha las características de tipo de datos de Python para realizar validaciones automáticas y proporciona soporte asíncrono, lo que lo hace ideal para desarrollar aplicaciones web rápidas y escalables.

- **API GATEWAY DE AMAZON WEB SERVICES:** Amazon API Gateway es un servicio completamente gestionado de Amazon Web Services (AWS) que facilita la creación, publicación, mantenimiento, monitoreo y protección de APIs a cualquier escala. Permite a los desarrolladores crear APIs RESTful y WebSocket para acceder a servicios de backend como aplicaciones ejecutadas en AWS Lambda, Amazon EC2, o cualquier aplicación web. API Gateway maneja tareas como la gestión del tráfico, el control de versiones de API, la autenticación y autorización, y la limitación de tasas, lo que permite a los desarrolladores enfocarse en la lógica de la aplicación y no en la infraestructura.

6 ARQUITECTURA

6.1.1 DESCRIPCIÓN GENERAL

El proyecto es una plataforma web que permite a los usuarios subir, visualizar y comentar películas. Utiliza una topología basada en API REST para la comunicación entre el frontend y el backend. Los usuarios pueden registrarse, autenticarse, subir videos, ver detalles de las películas y comentar sobre ellas. Los datos de las películas y los comentarios se almacenan en una base de datos MongoDB, y los archivos de video se almacenan en Firebase Storage.

6.1.2 COMPONENTES PRINCIPALES

7 FRONTEND.

- **Tecnología:** Laravel
- **Descripción:** La interfaz de usuario está construida con Laravel. Proporciona formularios para que los usuarios manden mensajes, vean detalles de los usuarios y puedan realizar la comunicación por medio de una videollamada. La interfaz consume una API Gateway proporcionada por el servicio de AWS para interactuar con los datos.

8 BACKEND.

- **Tecnología:** Node.js con Express
- **Descripción:** El servidor backend está construido con Node.js y el framework Express. Proporciona endpoints RESTful para manejar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre las mensajes y los usuarios. También maneja la autenticación de usuarios y la carga de archivos de imágenes al servico imgBB (servicio de almacenamiento de imágenes).

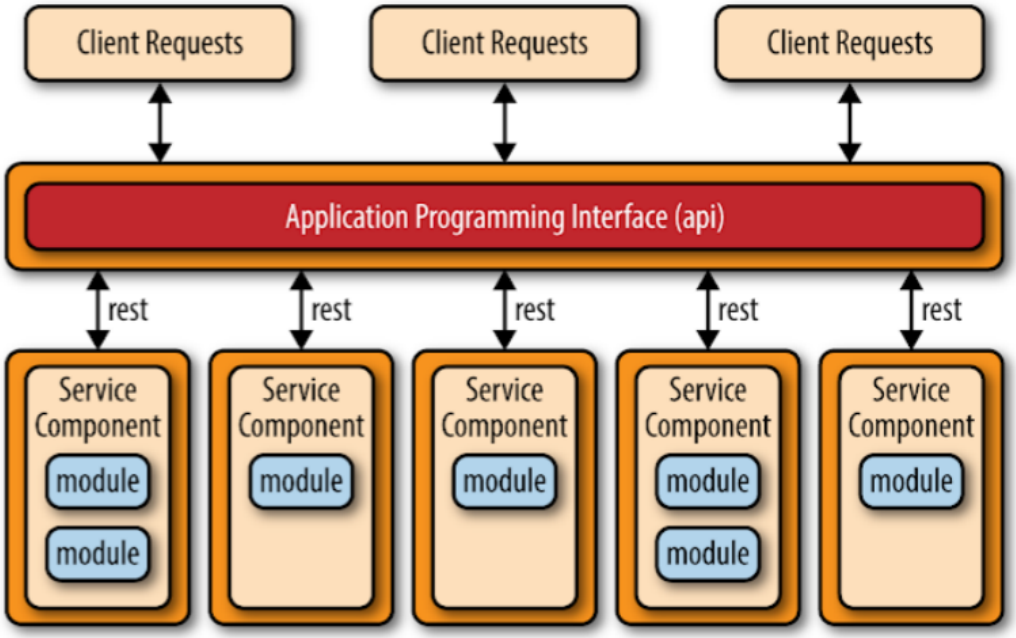
9 BASE DE DATOS.

- **Tecnología:** MongoDB
- **Descripción:** Los datos de las películas, los comentarios y la información de los usuarios se almacenan en MongoDB. La base de datos se comunica con el backend a través de Mongoose, una biblioteca de modelado de datos para MongoDB y Node.js.

10 ALMACENAMIENTO DE ARCHIVOS.

- **Tecnología:** Firebase Storage
- **Descripción:** Los archivos de imágenes subidos por los usuarios se almacenan en imgBB. El backend se encarga de subir los archivos a imgBB y de gestionar las URLs de las imágenes para su acceso desde el frontend.

10.1.1 TOPOLOGÍA



El proyecto tiene una topología basada en API REST. La API REST actúa como intermediaria entre el frontend y el backend, permitiendo una comunicación eficiente y estructurada. Los endpoints de la API siguen las convenciones RESTful, facilitando la escalabilidad y mantenibilidad del sistema.

11 COMPONENTES

El sistema distribuido se divide en tres componentes que se comunican entre sí para poder lograr toda la gestión de usuarios también la visualización de películas y por último la gestión de las películas y esos componentes son:

- **Componente de Gestión de usuarios y Logeo:** Este componente que está desarrollado en asp.net CORE MVC .net 6.0 es un componente muy importante para el sistema ya que en este se realiza toda la gestión de los usuarios y perfiles de estos, también se incluye la recuperación de contraseña y la modificación de esta y por último todo este componente se conecta con la parte de node.js mediante una API que hace un post de los datos de un perfil y mediante un endpoint esta los recupera en node.js.

- **Componente de visualización de películas:** Este componente que esta realizado en node.js con react tiene como objetivo mostrar las caratulas de las películas y su respectiva visualización al hacer clic en una película toda esta lógica de poder ver el listado y demás este hecho con apis para mostrar los datos de todas las películas.
- **Componente de gestión de películas:** En este apartado se tiene una interfaz para poder subir una película como archivo a la base de datos de firebase y esto es vital para poder subir contenido y después visualizarlo dentro de la visualización de las películas.

12 APIS.

- **OBTENER LISTAS**

  Método: GET

  Endpoint: /

  Descripción: Obtiene una lista de elementos del sistema, filtrados por tipo y género si se proporcionan.

  Parámetros:

  type (query): Tipo de contenido (opcional).

  genre (query): Género del contenido (opcional).

  Respuestas:

  200 OK: La solicitud se ha completado exitosamente. Retorna una lista de elementos.

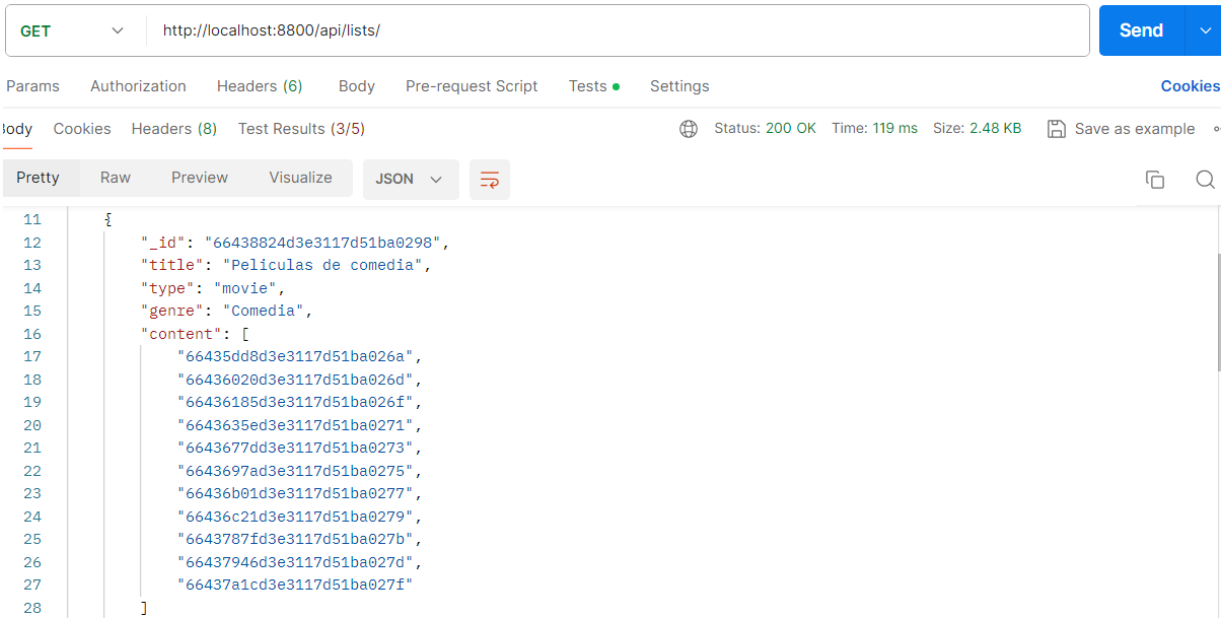
  500 Internal Server Error: Ocurrió un error interno al intentar obtener las listas.

  Notas:

  Si se proporciona type, la lista se filtrará por tipo.

  Si se proporciona genre, la lista se filtrará por género además del tipo.

  Respuesta exitosa.



13 MICROSERVICIOS

Se desarrollaron 3 microservicios:

API Gateway Express

https://6l3ago91kg.execute-api.sa-east-1.amazonaws.com/api/

Servidor express:

https://apirestnodejs-jev4.onrender.com/api-docs/

API Gateway Firebase

https://9j3jl8aftf.execute-api.sa-east-1.amazonaws.com/userconfig/

Servidor de Firebase expres:

https://nodeapifirebase.onrender.com/api-docs/

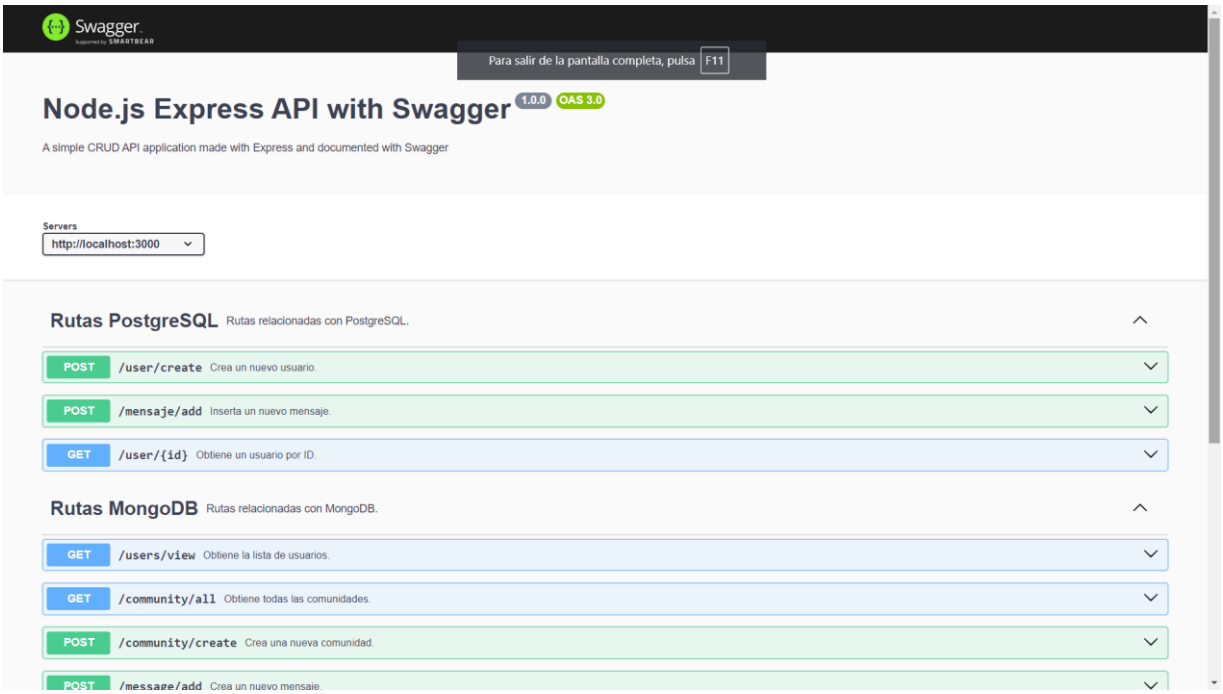
Servidor FastAPI

https://talkativefastapi.onrender.com/docs

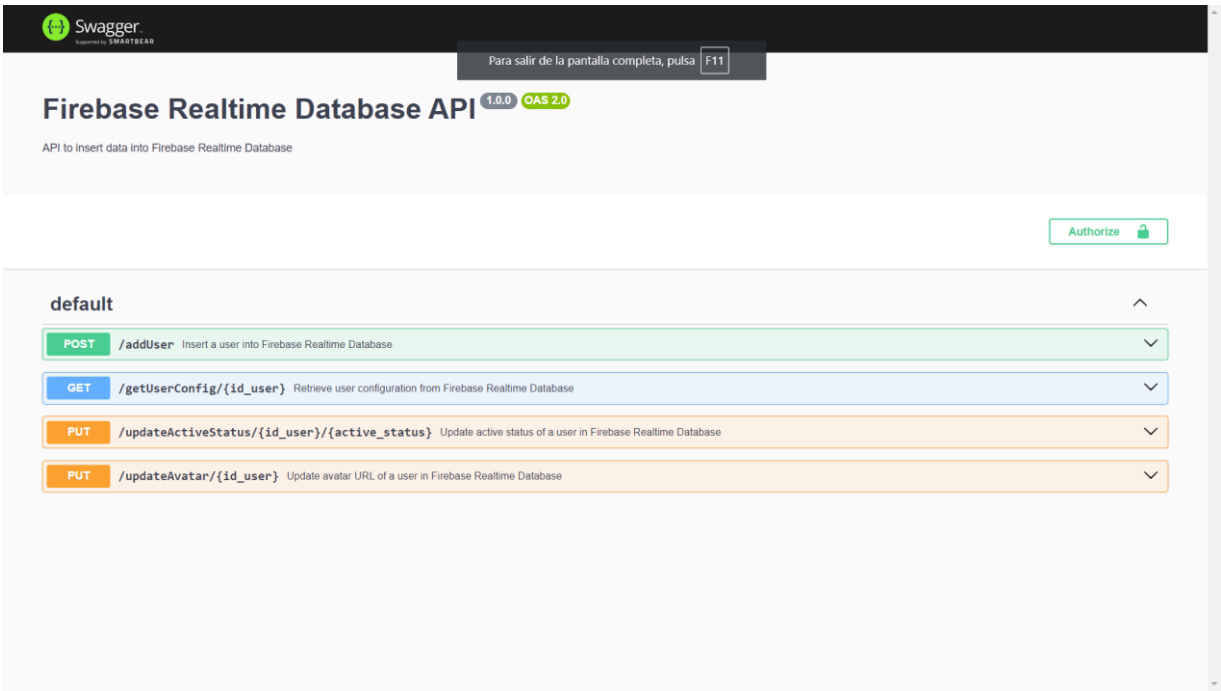
14 ENDPOINTS

A) MICROSERVICIO USUARIOS, MENSAJES Y COMUNIDADES

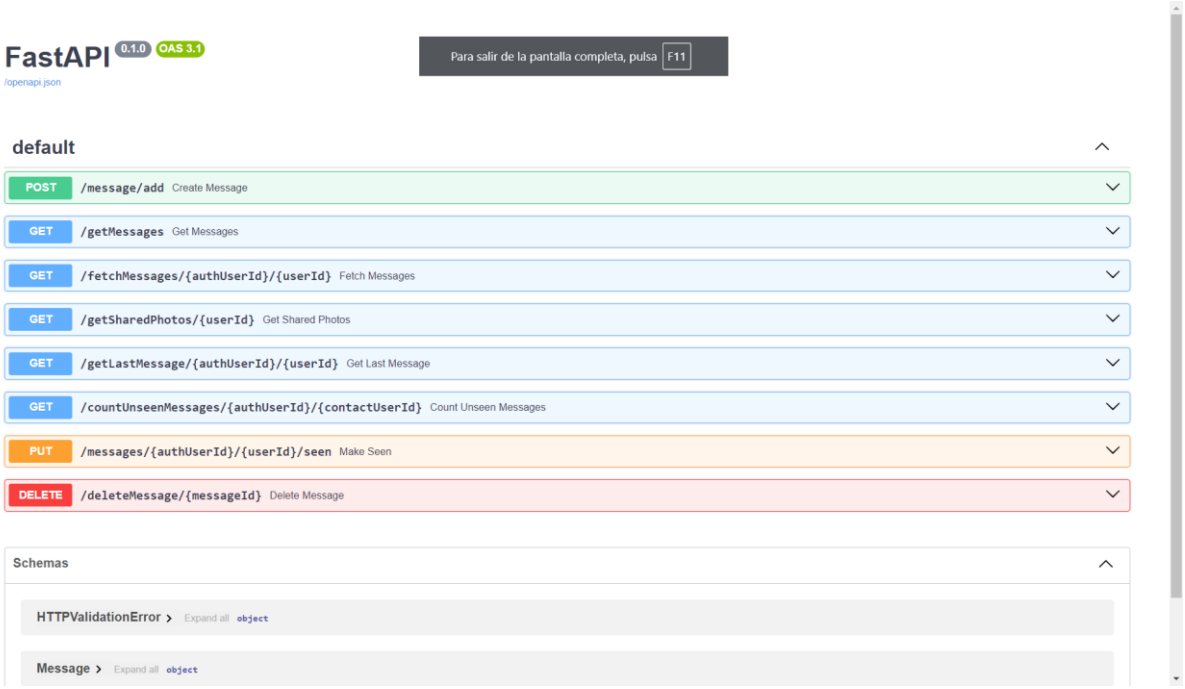
El algoritmo de



B) MICROSERVICIO CONFIGURACIÓN DE USUARIOS



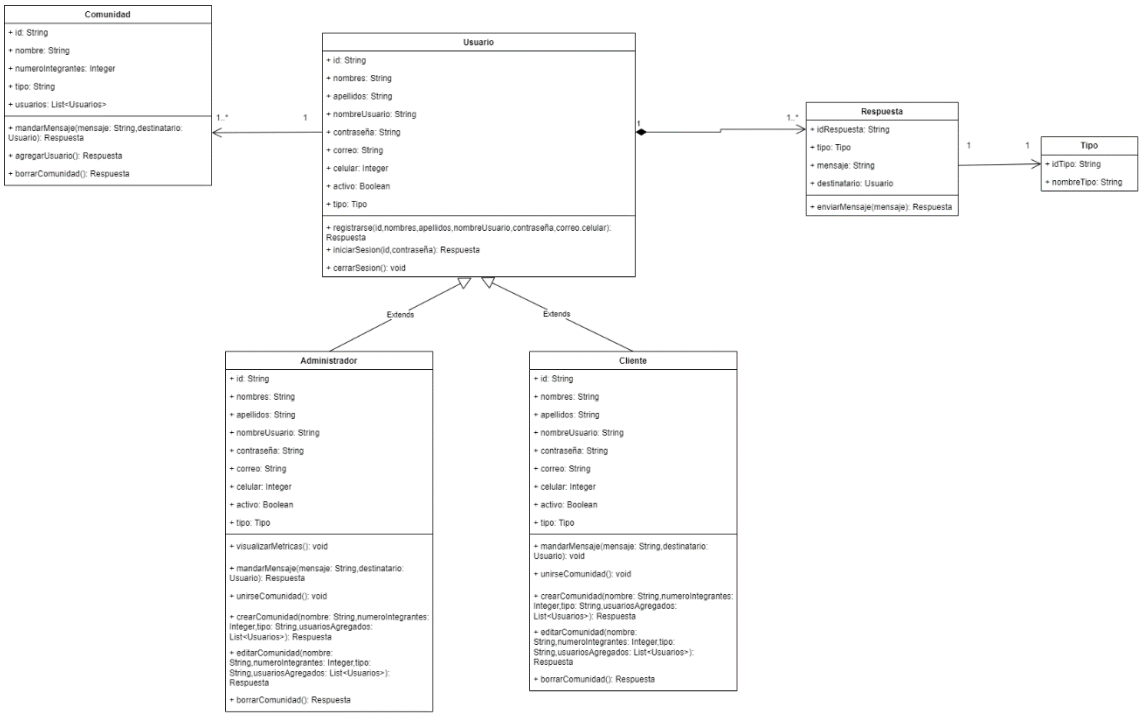
C) MICROSERVICIO MENSAJES



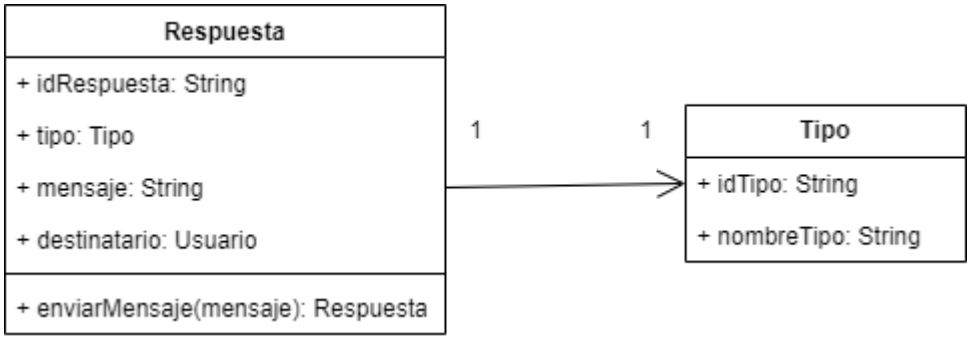
15 FLUJO Y FUNCIONAMIENTO DEL SISTEMA

1. DIAGRAMAS

1.1.DIAGRAMA BD RELACIONAL



1.2.DIAGRAMA BD NO RELACIONAL



2. SCRIPTS

2.1.SCRIPT BD RELACIONAL (POSTGRESQL 15)

```
CREATE DATABASE "BDRChat"

    WITH

    OWNER = postgres

    ENCODING = 'UTF8'

    LC_COLLATE = 'Spanish_Bolivia.1252'

    LC_CTYPE = 'Spanish_Bolivia.1252'
```

```
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;

-- Tabla usuarios
CREATE TABLE IF NOT EXISTS public.usuarios
(
    id character varying(20) COLLATE pg_catalog."default" NOT NULL,
    nombres character varying(100) COLLATE pg_catalog."default" NOT NULL,
    apellidos character varying(100) COLLATE pg_catalog."default" NOT NULL,
    nombre_usuario character varying(100) COLLATE pg_catalog."default" NOT NULL,
    clave character varying(30) COLLATE pg_catalog."default" NOT NULL,
    correo character varying(50) COLLATE pg_catalog."default",
    celular character varying(50) COLLATE pg_catalog."default",
    activo boolean,
    tipo character varying(50) COLLATE pg_catalog."default" NOT NULL,
    imagen_perfil character varying(200) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT usuarios_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.usuarios
    OWNER to postgres;

-- Tabla mensajes
CREATE TABLE IF NOT EXISTS public.mensaje
(
```

```
    id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1
MAXVALUE 2147483647 CACHE 1 ),
    mensaje character varying(500) COLLATE pg_catalog."default" NOT NULL,
    fecha timestamp with time zone NOT NULL,
    destinatario character varying(20) COLLATE pg_catalog."default" NOT NULL,
    estado character varying(50) COLLATE pg_catalog."default" NOT NULL,
    imagen character varying(200) COLLATE pg_catalog."default",
    CONSTRAINT mensaje_pkey PRIMARY KEY (id),
    CONSTRAINT destinatario_fk FOREIGN KEY (destinatario)
        REFERENCES public.usuarios (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.mensaje
    OWNER to postgres;

-- Tabla comunidad

CREATE TABLE IF NOT EXISTS public.comunidad
(
    id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1
MAXVALUE 2147483647 CACHE 1 ),
    nombre character varying(100) COLLATE pg_catalog."default" NOT NULL,
    numero_integrantes integer NOT NULL,
    tipo character varying(50) COLLATE pg_catalog."default" NOT NULL,
    usuario_id character varying(20) COLLATE pg_catalog."default",
    CONSTRAINT comunidad_pkey PRIMARY KEY (id),
```

```
    CONSTRAINT usuarios_fk FOREIGN KEY (usuario_id)
        REFERENCES public.usuarios (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.comunidad
    OWNER to postgres;

-- Tabla respuestas
CREATE TABLE IF NOT EXISTS public.respuesta
(
    id_respuesta integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1
MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
    tipo integer NOT NULL,
    mensaje character varying(500) COLLATE pg_catalog."default" NOT NULL,
    destinatario character varying(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT respuesta_pkey PRIMARY KEY (id_respuesta),
    CONSTRAINT destinatario_fk FOREIGN KEY (destinatario)
        REFERENCES public.usuarios (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
       CONSTRAINT tipo_fk FOREIGN KEY (tipo)
        REFERENCES public.tipo_respuesta (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

```
TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.respuesta

    OWNER to postgres;


-- Tabla tipo respuestas

CREATE TABLE IF NOT EXISTS public.tipo_respuesta

(

    id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1
MAXVALUE 2147483647 CACHE 1 ),

    nombre character varying(100) COLLATE pg_catalog."default" NOT NULL

)


TABLESPACE pg_default;


ALTER TABLE IF EXISTS public.tipo_respuesta

    OWNER to postgres;
```

2.2.SCRIPT BD NO RELACIONAL (MONGODB)

```
// Colección respuesta

db.createCollection("respuesta", {

  validator: {

    $jsonSchema: {

      bsonType: "object",

      required: ["tipo", "mensaje", "destinatario"],

      properties: {

        tipo: {

          bsonType: "int",
```

```
            description: "Debe ser un entero"
          },
          mensaje: {
            bsonType: "string",
            maxLength: 500,
            description: "Debe ser una cadena de texto de máximo 500 caracteres"
          },
          destinatario: {
            bsonType: "string",
            maxLength: 20,
            description: "Debe ser una cadena de texto de máximo 20 caracteres"
          }
        }
      }
    }
});

// Crear índices
db.respuesta.createIndex({ "destinatario": 1 });

// Colección tipo_respuesta
db.createCollection("tipo_respuesta", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nombre"],
      properties: {
        nombre: {
          bsonType: "string",
```

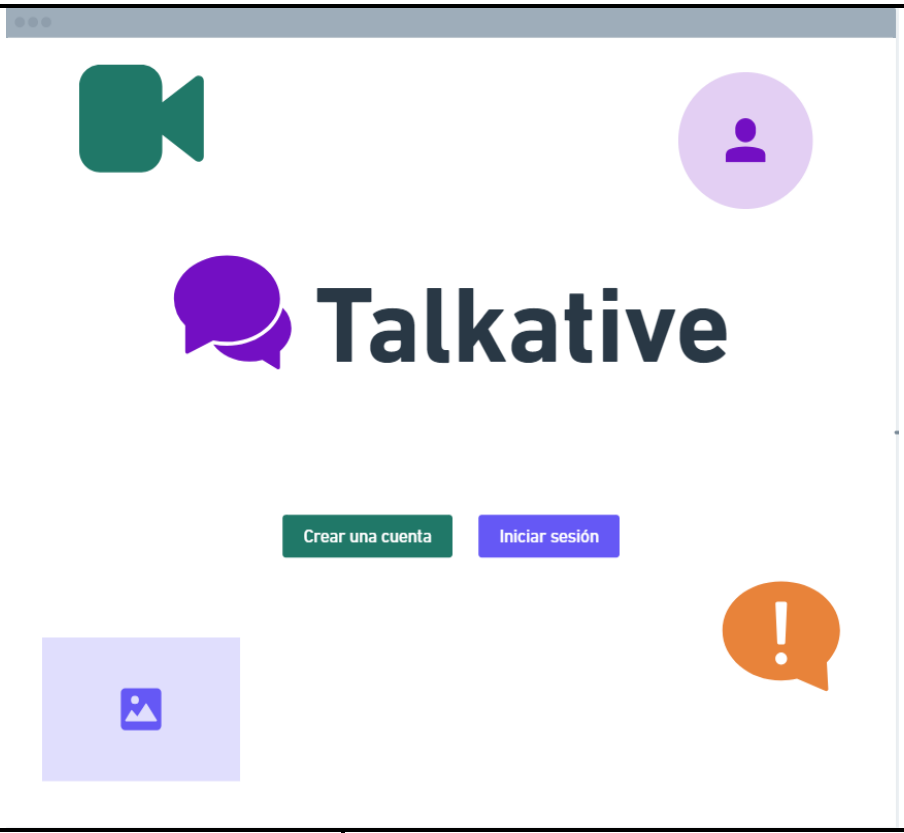
```
            maxLength: 100,
            description: "Debe ser una cadena de texto de máximo 100 caracteres"
          }
        }
      }
    }
});

// Crear índices
db.tipo_respuesta.createIndex({ "nombre": 1 });
```

3. HISTORIAS DE USUARIO POR SPRINT

3.1.SPRINT 1

Historia de Usuario		
Código: 01	**Usuario:** Cliente	**Orden:** 1
Nombre Historia: Creación del login		
Como: Cliente **Quiero:** Visualizar el login del sistema web **Para:** Iniciar sesión o crear una cuenta.		



Prioridad:	**Riesgo de Desarrollo:**
Alta	Baja

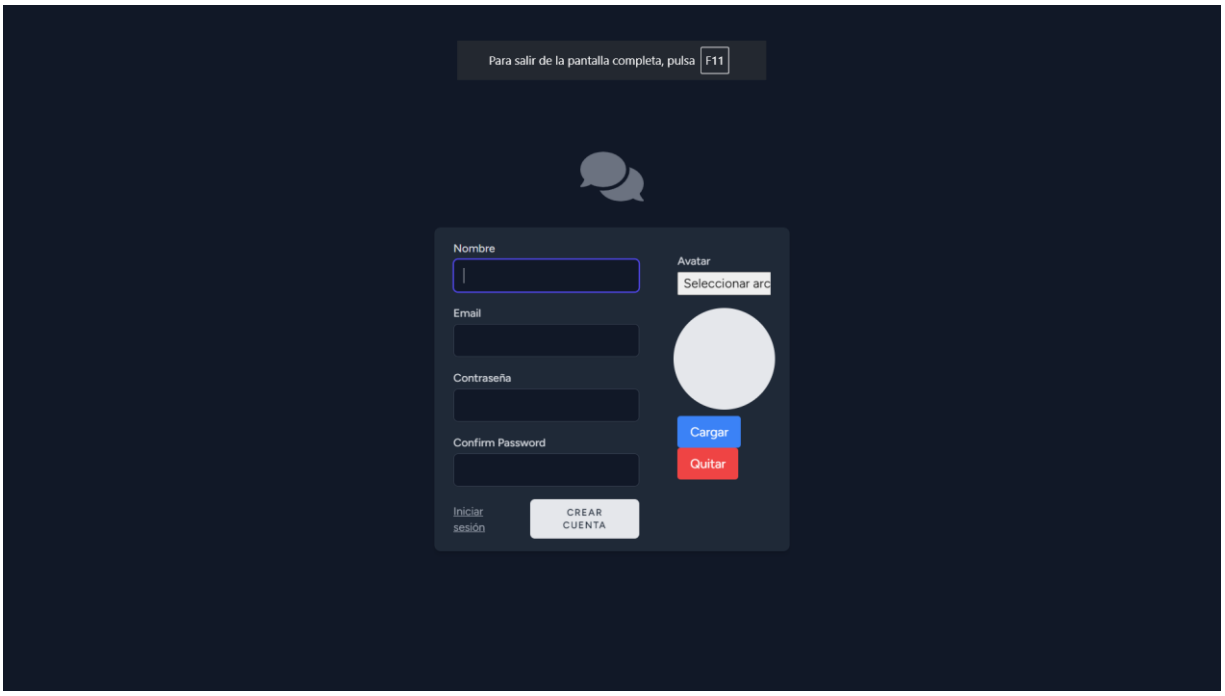
Programador Responsable:
Patrick Martínez Moscoso

Descripción:

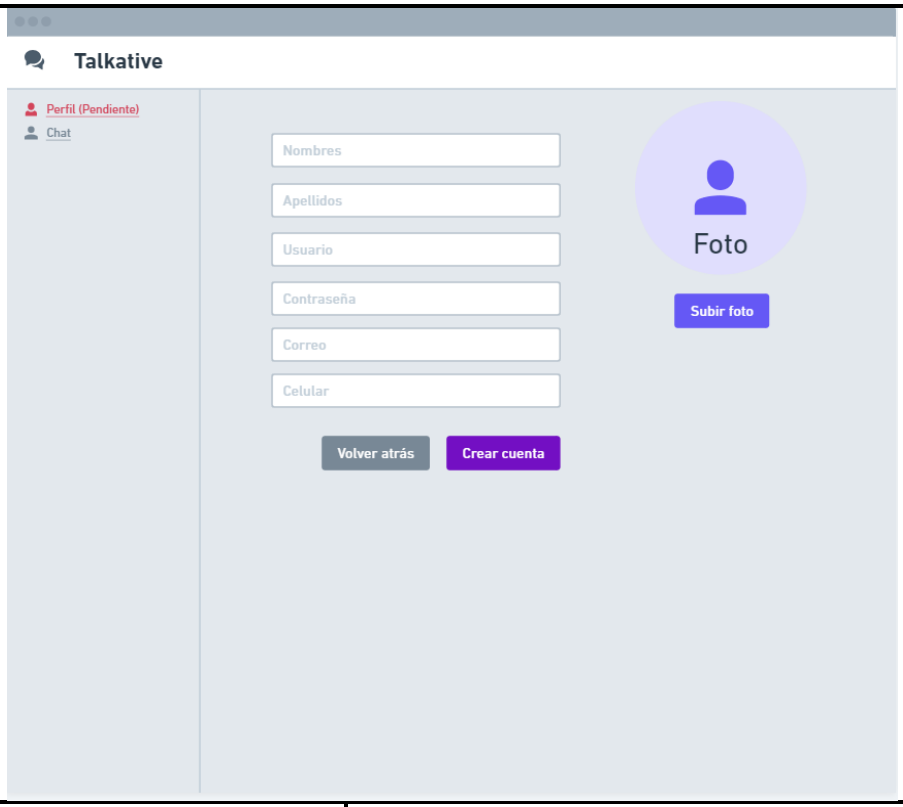
Poder iniciar sesión o crear una cuenta.

Validación:

1. El cliente puede crear una cuenta
2. El cliente puede iniciar sesión



Historia de Usuario		
Código: 02	**Usuario:** Cliente	**Orden:** 1
Nombre Historia: Creación del perfil		
Como: Cliente **Quiero:** Visualizar una interfaz para crear un perfil **Para:** Crear perfil		



Prioridad:	**Riesgo de Desarrollo:**
Alta	Baja

Programador Responsable:
Patrick Martínez Moscoso

Descripción:

Poder crear un perfil en el sistema de mensajería

Validación:

1. El cliente debe poder introducir solamente caracteres en el campo de nombre o apellido.
2. El cliente debe poder introducir un correo válido en el campo de correo.
3. El cliente debe poder introducir solamente caracteres numéricos en el campo de celular.
4. El cliente solo podrá subir una fotografía de máximo 300x300 pixeles con los siguientes formatos .jpg o .png

Historia de Usuario

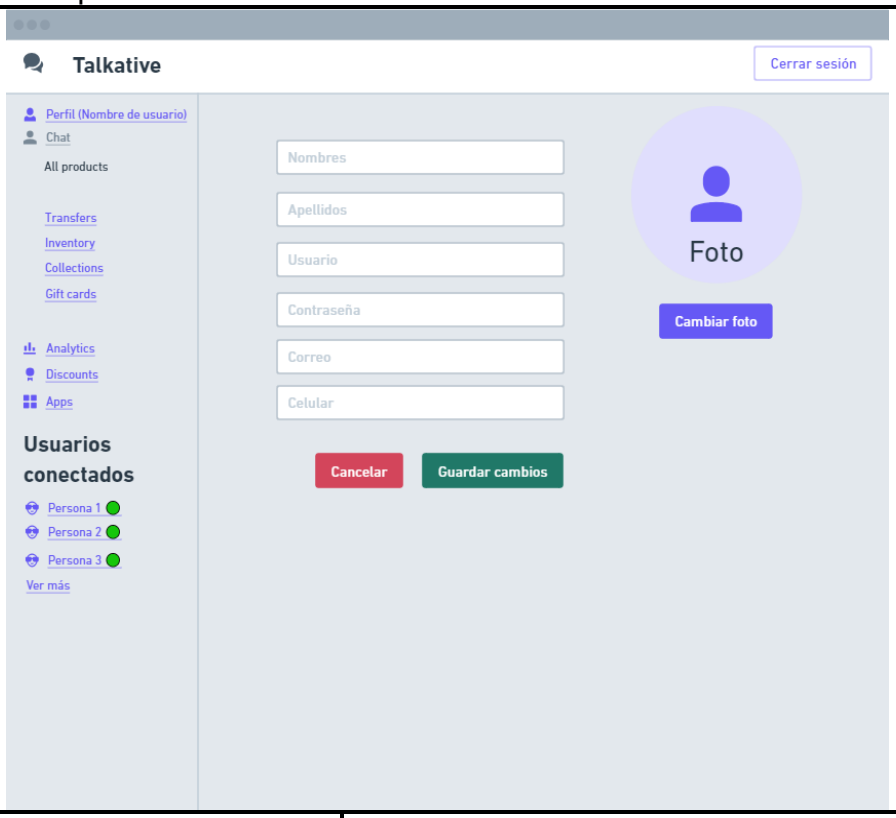
Código: 03	**Usuario:** Cliente	**Orden:** 1

Nombre Historia: Editar perfil

Como: Cliente

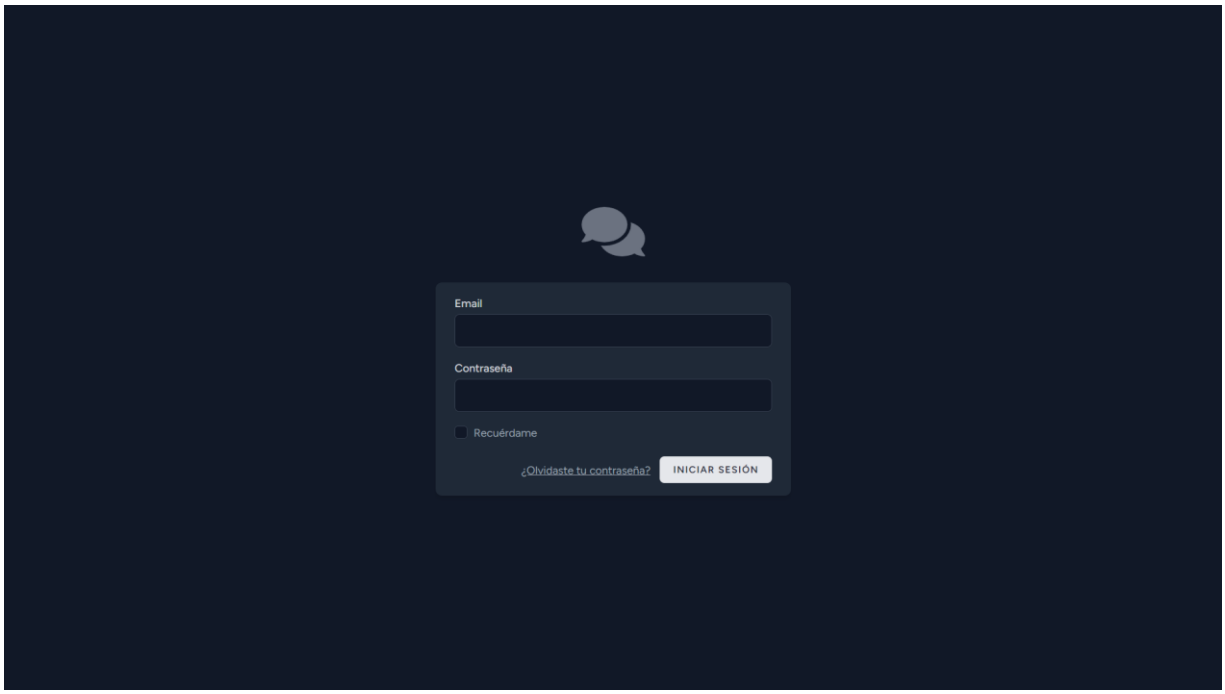
Quiero: Visualizar la edición del perfil

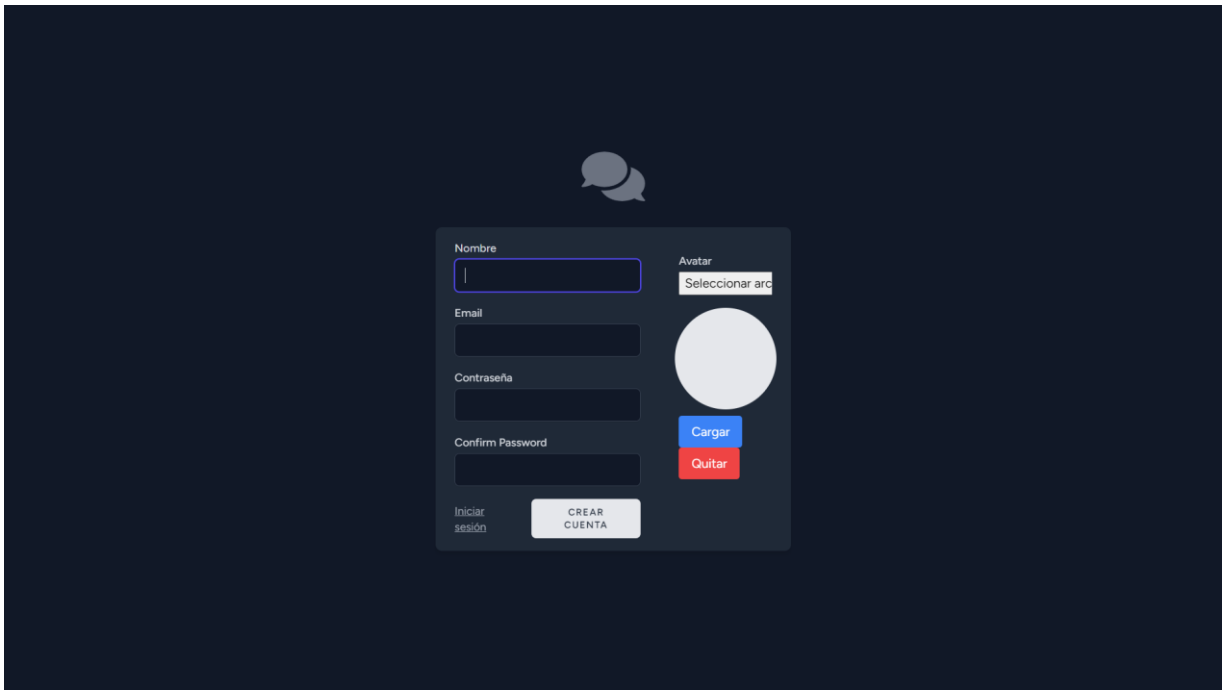
Para: Editar perfil



Prioridad: Alta	**Riesgo de Desarrollo:** Baja

Programador Responsable: Patrick Martínez Moscoso
Descripción: Poder editar un perfil de usuario.
Validación: 1. El cliente debe poder introducir solamente caracteres en el campo de nombre o apellido. 2. El cliente debe poder introducir un correo válido en el campo de correo. 3. El cliente debe poder introducir solamente caracteres numéricos en el campo de celular. 4. El cliente solo podrá subir una fotografía de máximo 300x300 pixeles con los siguientes formatos .jpg o .png





3.2.SPRINT 2

Historia de Usuario		
Código: 05	**Usuario:** Cliente	**Orden:** 2
Nombre Historia: Recibir mensaje		
Como: Cliente **Quiero:** Poder recibir mensajes a otros usuarios **Para:** Leer y responder a sus mensajes		

Prioridad: Alta	**Riesgo de Desarrollo:** Baja

Programador Responsable:
Juan Pablo Menacho Castro

Descripción:

Poder recibir mensajes de otros usuarios y visualizarlos en mi aplicación de chat.

Validación:

Los mensajes entrantes se muestran correctamente en la interfaz del usuario. Se notifica al usuario cuando recibe un nuevo mensaje.

Historia de Usuario		
Código: 06	**Usuario:** Cliente	**Orden:** 2
Nombre Historia: Envío de imágenes		

Como: Cliente

Quiero: Poder enviar imágenes a otros usuarios

Para: Compartir imágenes de forma visual en las conversaciones

Prioridad:	**Riesgo de Desarrollo:**
Alta	Baja

Programador Responsable:
Juan Pablo Menacho Castro

Descripción:

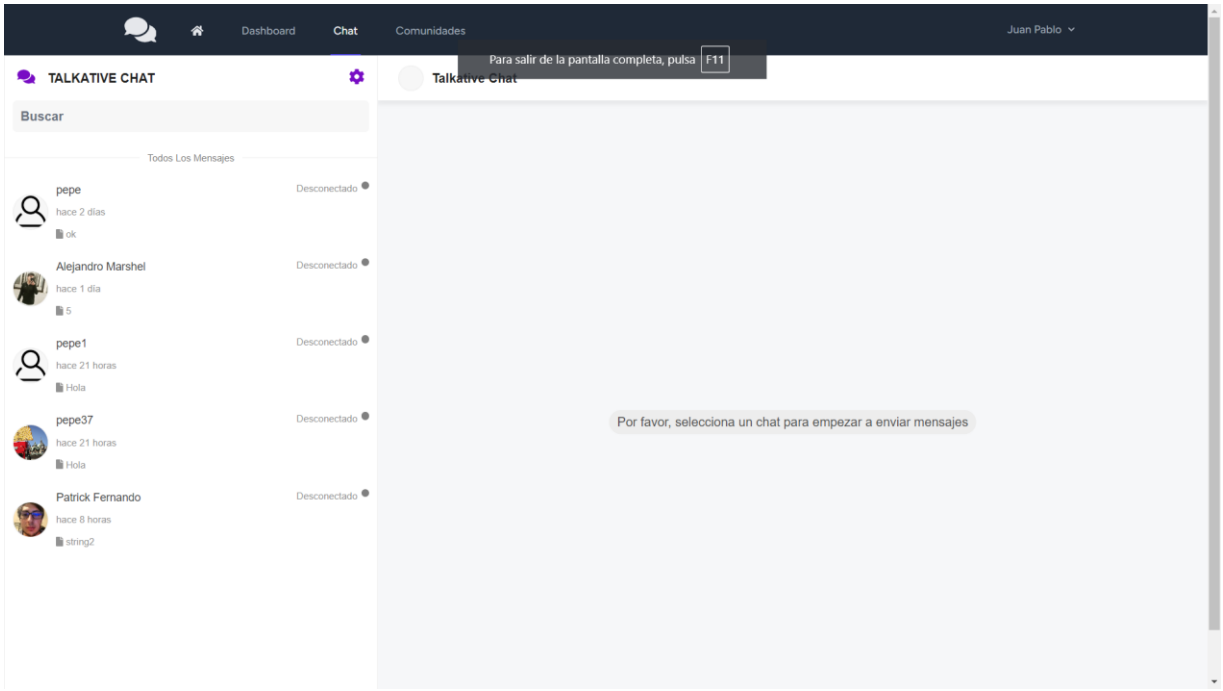
Porder adjuntar imágenes a los mensajes que envío para poder compartir imágenes de forma visual en las conversaciones con otros usuarios.

Validación:

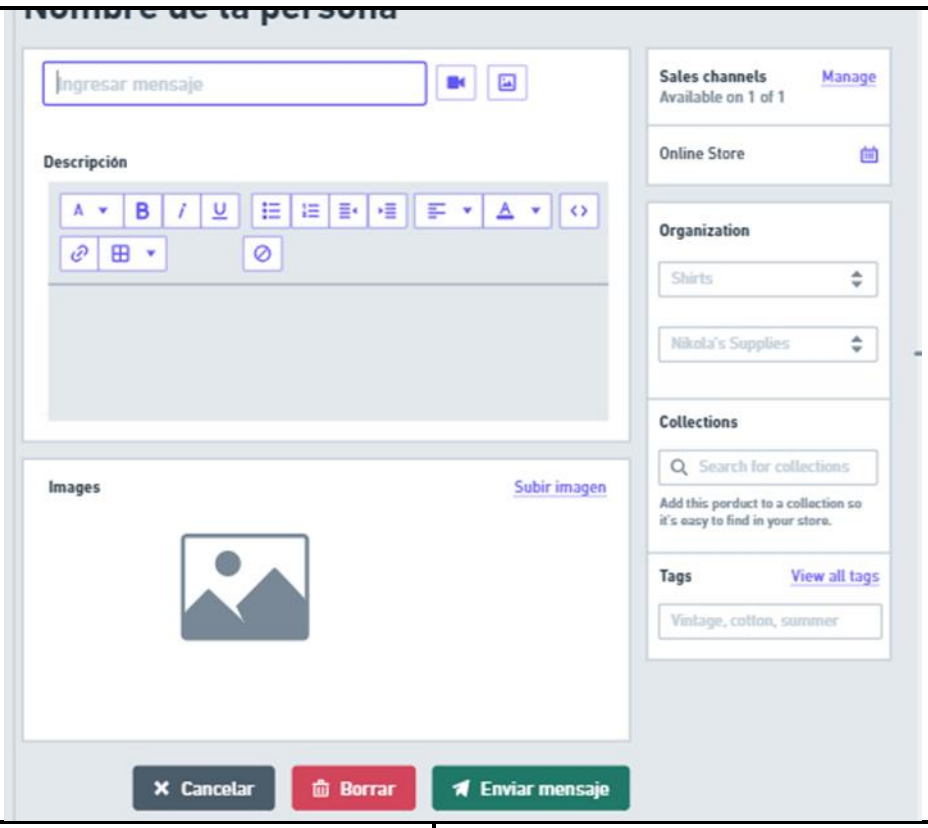
Se permite adjuntar imágenes a los mensajes enviados por el usuario. Las imágenes adjuntas se muestran correctamente en la conversación y pueden ser visualizadas en su tamaño original.

Historia de Usuario		
Cdigo: 07	**Usuario:** Cliente	**Orden:** 2
Nombre Historia: Verificar estado de conexión en línea		
Como: Cliente **Quiero:** Poder ver si un usuario está conectado en línea **Para:** Saber si puedo iniciar una conversación con ellos y esperar una respuesta rápida		
Prioridad: Alta	**Riesgo de Desarrollo:** Baja	
Programador Responsable: Juan Pablo Menacho Castro		
Descripción: Poder ver el estado de conexión en línea de un usuario para saber si puedo iniciar una conversación con ellos y esperar una respuesta rápida.		
Validación:		

Se muestra el estado de conexión en línea de un usuario en su perfil o junto a su nombre en la lista de contactos.
El estado de conexión se actualiza en tiempo real para reflejar cambios en la disponibilidad del usuario.



Historia de Usuario		
Código: 08	**Usuario:** Cliente	**Orden:** 2
Nombre Historia: Eliminar mensaje enviado		
Como: Cliente **Quiero:** Poder eliminar un mensaje que he enviado previamente **Para:** Retirar un mensaje que ya no deseo que aparezca en la conversación		



Prioridad:	**Riesgo de Desarrollo:**
Alta	Baja

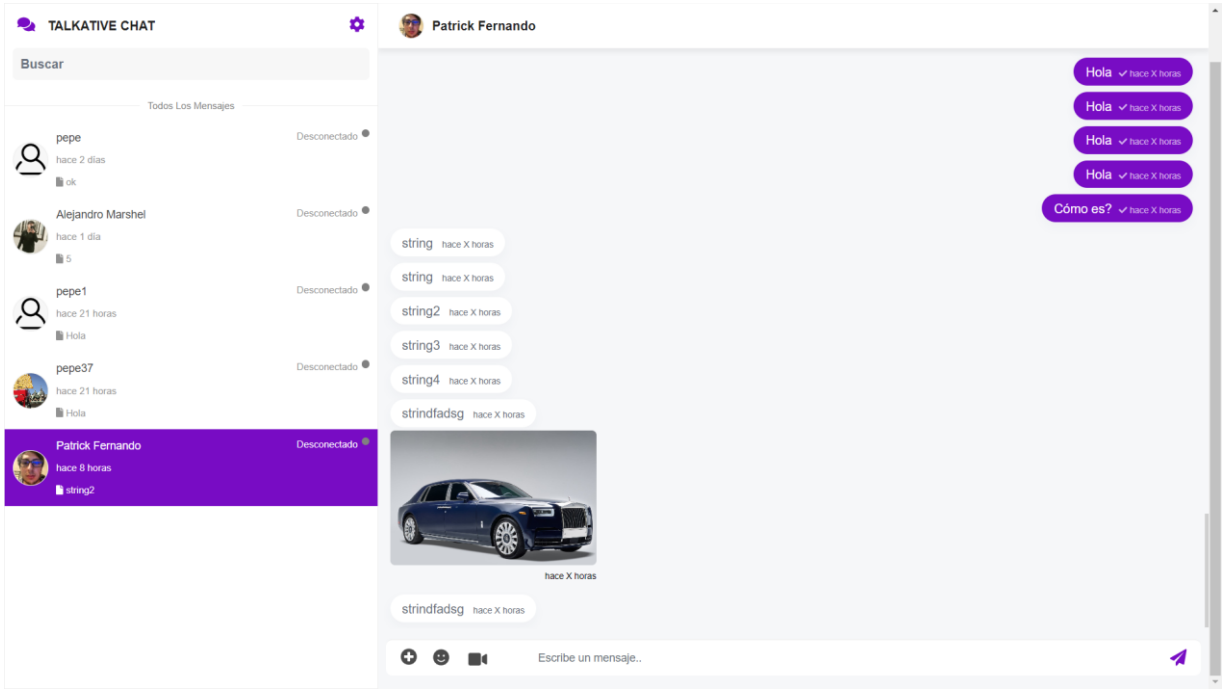
Programador Responsable:
Juan Pablo Menacho Castro

Descripción:

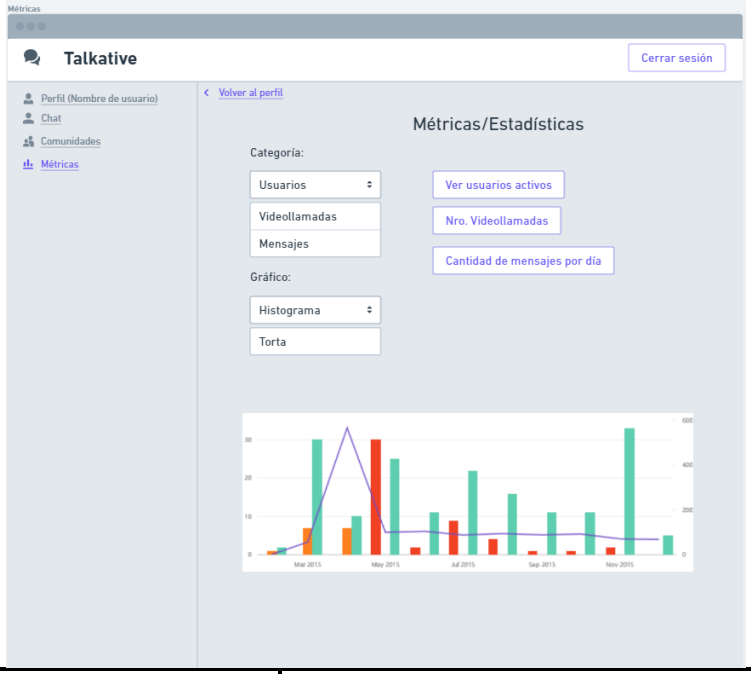
Poder eliminar un mensaje que he enviado previamente en caso de que desee retirarlo de la conversación.

Validación:

Se permite eliminar mensajes enviados anteriormente por el usuario.
El mensaje eliminado desaparece correctamente de la conversación.



Historia de Usuario		
Código: 09	**Usuario:** Administrador	**Orden: 2**
Nombre Historia: Visualizar métricas		
Como: Administrador **Quiero:** Ver estadísticas **Para:** Poder visualizar las estadísticas en cuanto a los usuarios, videollamadas y mensajes.		



Prioridad: Alta	**Riesgo de Desarrollo:** Media

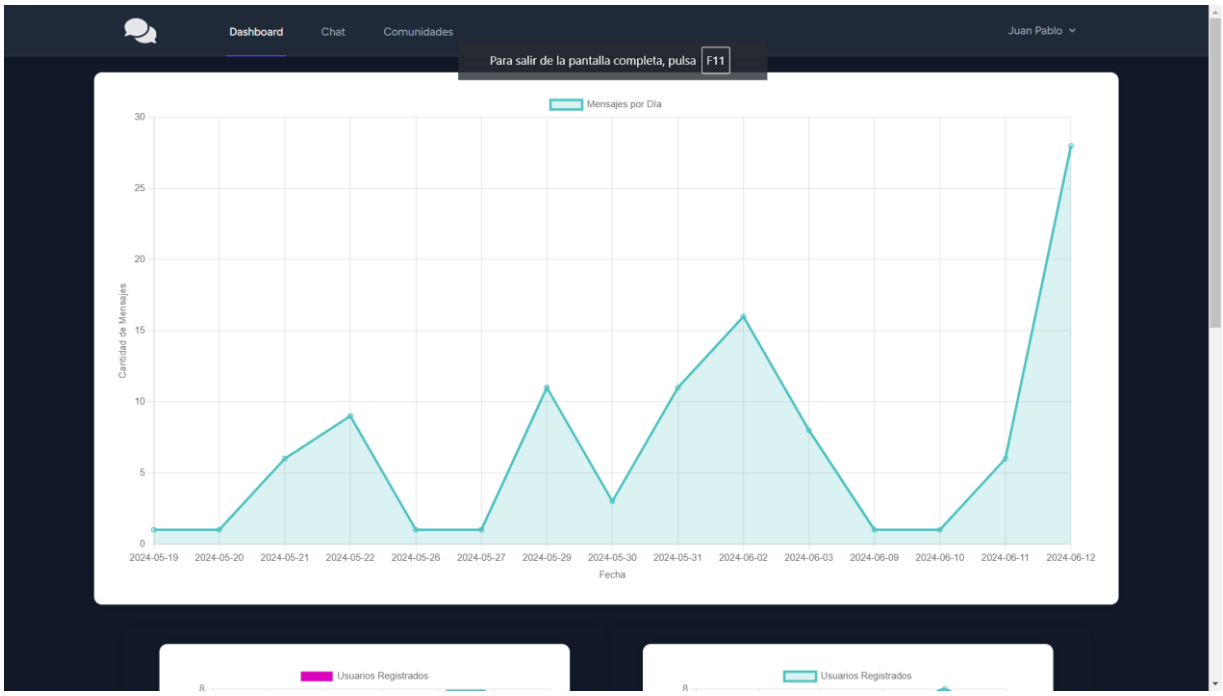
Programador Responsable:
Patrick Fernando Martínez

Descripción:

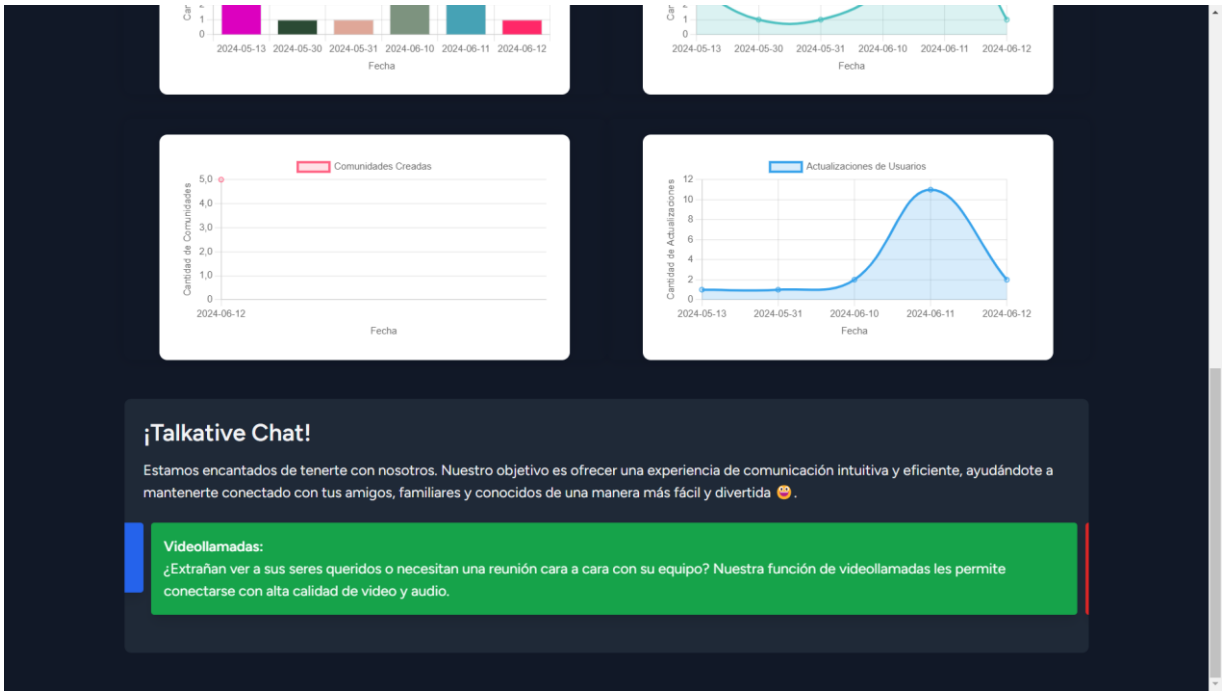
El usuario podrá visualizar las métricas.

Validación:

1. El usuario puede visualizar las métricas de acuerdo a una categoría y un tipo de gráfico.

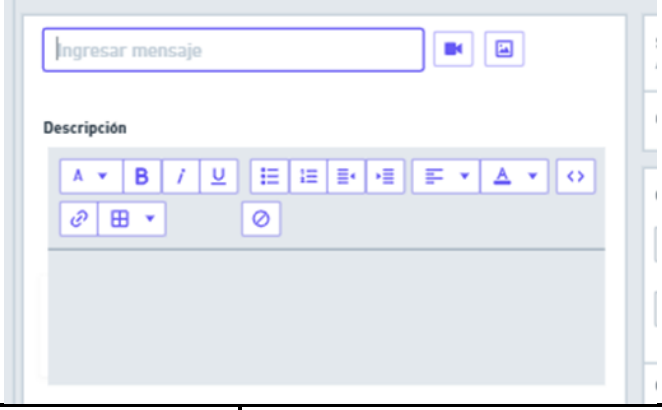

Dashboard
Chat
Comunidades
Juan Pablo
Para salir de la pantalla completa, pulsa F11
Mensajes por Día
Cantidad de Mensajes
Fecha
Usuarios Registrados

Usuarios Registrados
Cantidad de Usuarios
Fecha
Usuarios Registrados
Cantidad de Usuarios
Fecha
Comunidades Creadas
Cantidad de Comunidades
Fecha
Actualizaciones de Usuarios
Cantidad de Actualizaciones
Fecha
¡Talkative Chat!
Estamos encantados de tenerte con nosotros. Nuestro objetivo es ofrecer una experiencia de comunicación intuitiva y eficiente, ayudándote a mantenerte conectado con tus amigos, familiares y conocidos de una manera más fácil y divertida 😃.

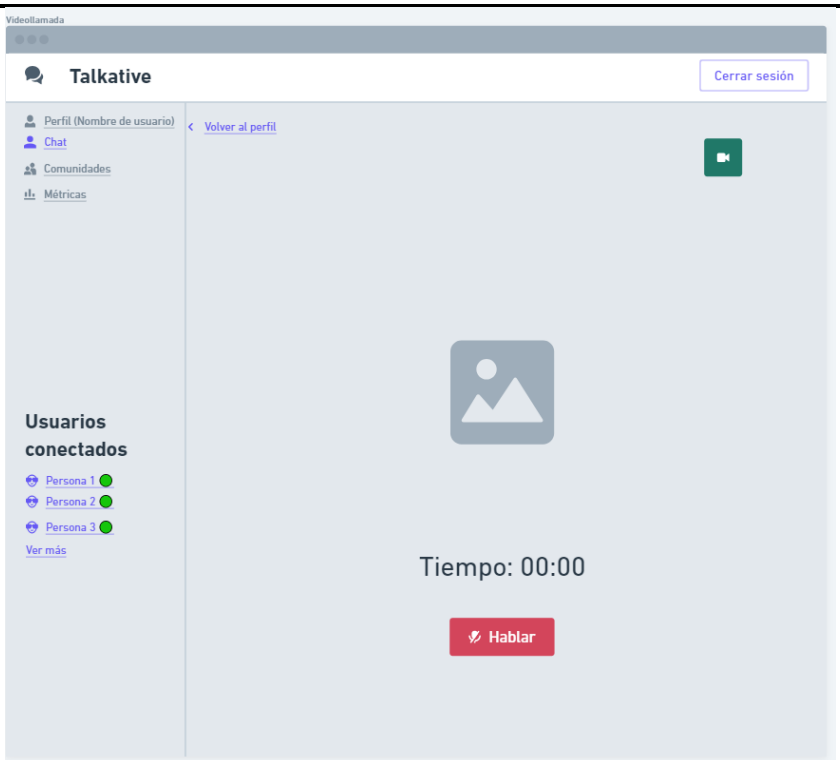


3.3.SPRINT 3

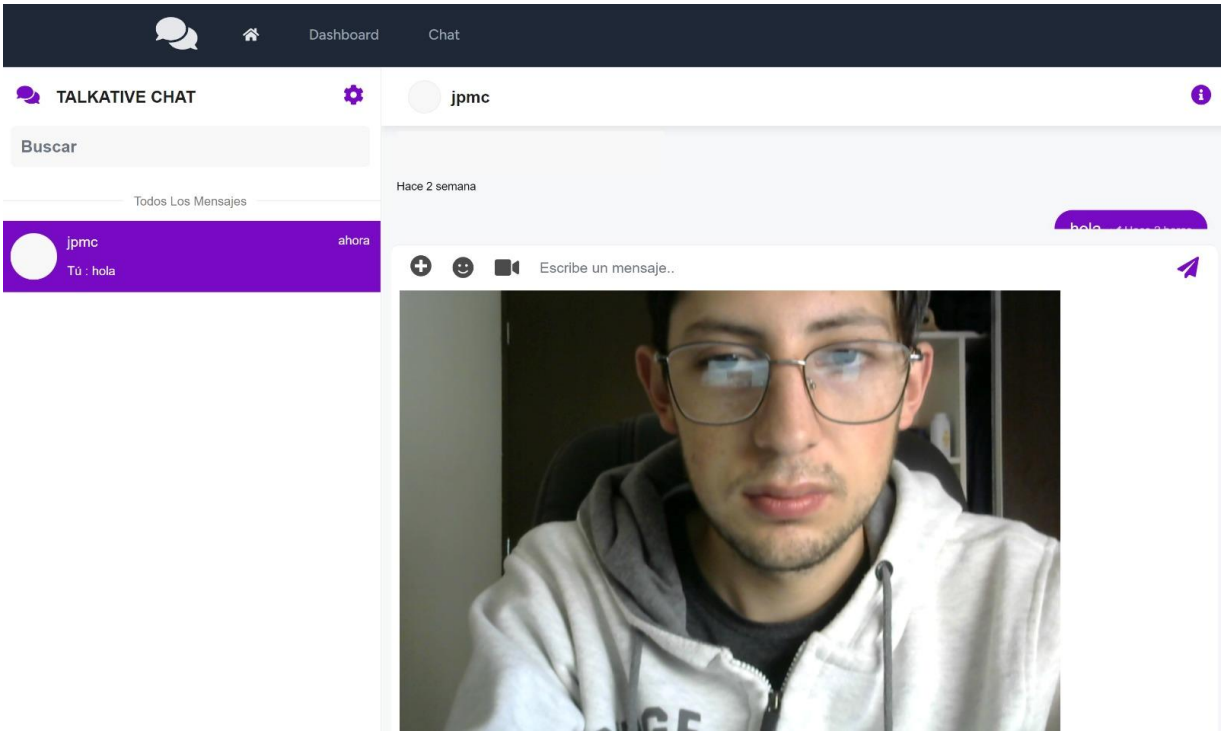
Historia de Usuario		
Código: 09	**Usuario:** Cliente	**Orden:** 3
Nombre Historia: Iniciar video llamada		
Como: Cliente **Quiero:** Comenzar una video llamada **Para:** Poder comunicarme através de una video llamada con otro usuario de mi chat		

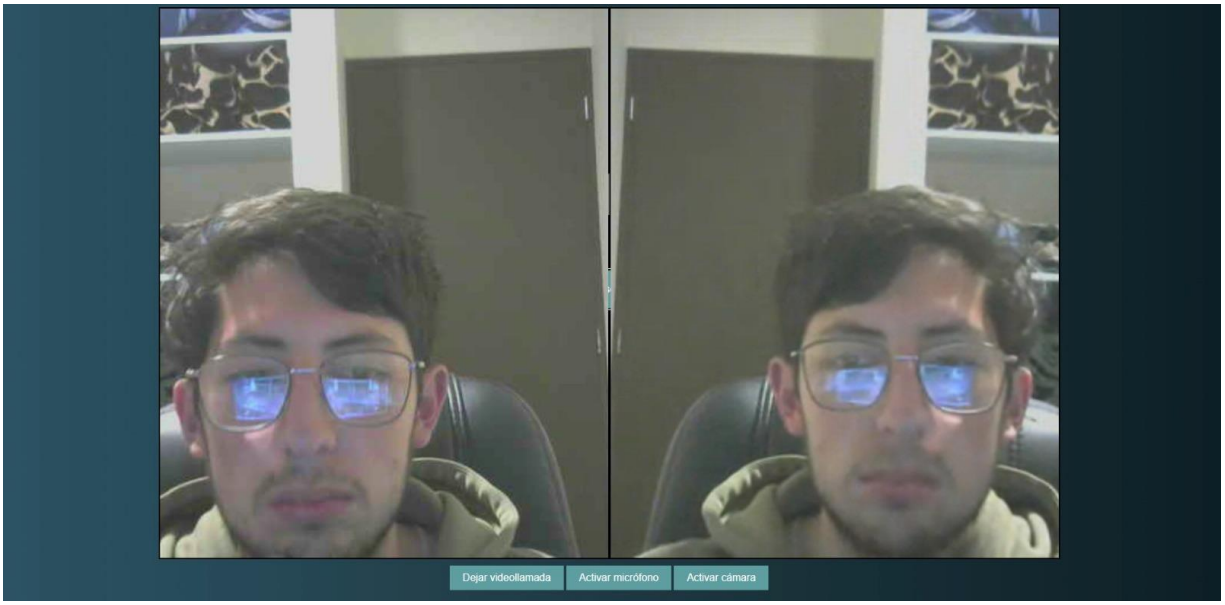


Prioridad: Alta	**Riesgo de Desarrollo:** Baja
Programador Responsable: Alejandro Marshel Ayala Orosco	
Descripción: El usuario desea poder iniciar una video llamada con otro usuario de su chat.	
Validación: 1. Existencia de un boton para poder comenzar la video llamada. 2. Que el usuario tenga añadadida a la persona en su chat.	

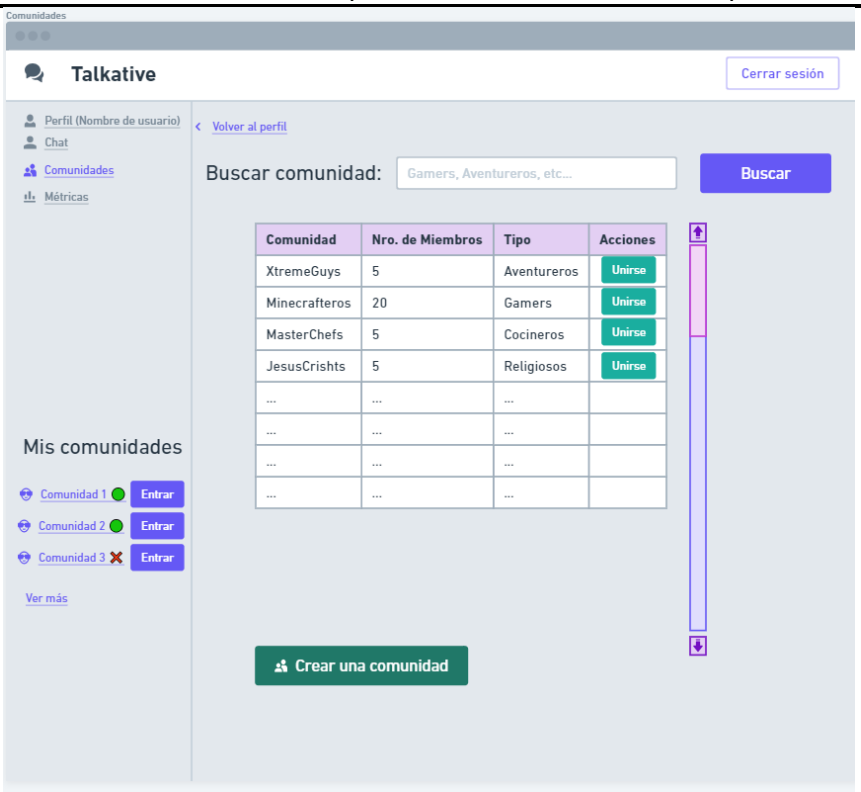
Historia de Usuario		
Código: 10	**Usuario:** Cliente	**Orden: 3**
Nombre Historia: Video llamada activa		
Como: Cliente **Quiero:** Realizar una video llamada **Para:** Comunicarme a través de este medio por la aplicación en vivo con otro usuario		
		
Prioridad: Alta	**Riesgo de Desarrollo:** Media	
Programador Responsable: Alejandro Marshel Ayala Orosco		
Descripción: El usuario desea poder realizar la comunicación por video llamada con otro usuario.		
Validación: 1. Los usuarios aceptan los terminos necesarios para la video llamada (cámara y microfono).		

2. El usuario receptor acepta la video llamada.
3. Se permite la video llamada con otro usuario a través de la aplicación satisfactoriamente.
4. Se mide el tiempo que tardo la video llamada.





Historia de Usuario		
Código: 11	**Usuario:** Cliente	**Orden: 3**
Nombre Historia: Comunidades		
Como: Cliente **Quiero:** Unirme a una comunidad **Para:** Tener una comunidad donde pueda interactuar con varias personas		



Prioridad:	**Riesgo de Desarrollo:**
Alta	Media

Programador Responsable:
Alejandro Marshel Ayala Orosco

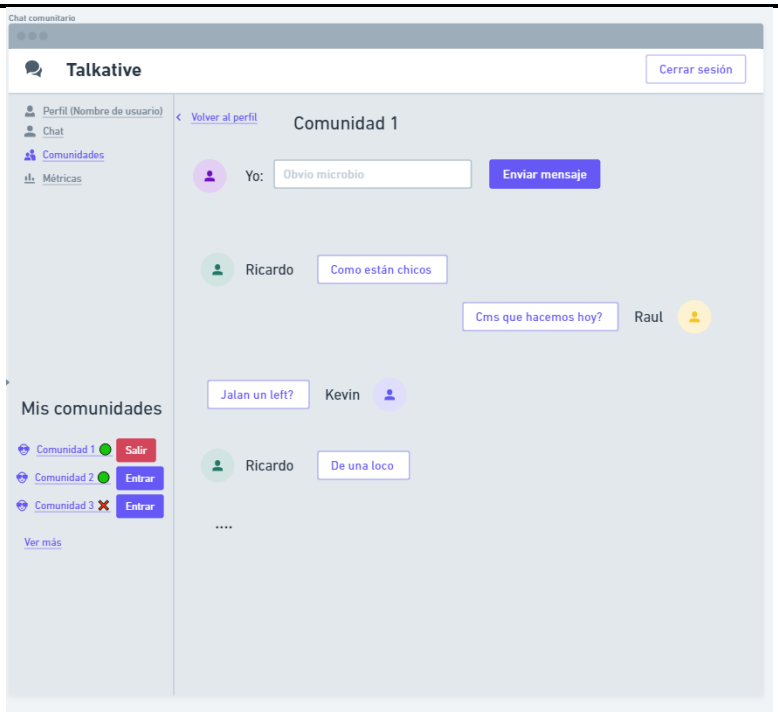
Descripción:

El usuario desea poder buscar, crear y elegir una comunidad

Validación:

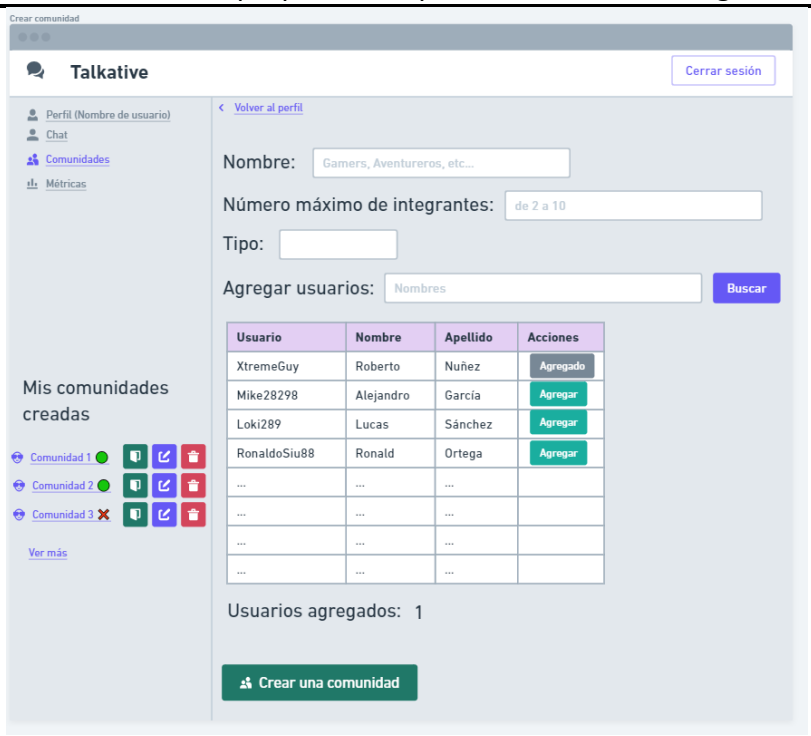
1. Los usuarios pueden buscar una comunidad de acuerdo a su tipo
2. Los usuarios pueden unirse a varias comunidades con un límite de hasta 10 comunidades.
3. Los usuarios pueden visualizar las comunidades a las que pertenece y entrar en dichas comunidades.

Los usuarios pueden crear una comunidad.

Historia de Usuario		
Código: 12	**Usuario:** Cliente	**Orden: 3**
Nombre Historia: Chat comunitario		
Como: Cliente **Quiero:** Chatear con varias personas **Para:** Comunicarme con gente que tenga gustos parecidos o realizar actividades en grupo.		
		
Prioridad: Alta	**Riesgo de Desarrollo:** Media	
Programador Responsable: Alejandro Marshel Ayala Orosco		
Descripción: El usuario desea poder enviar un mensaje y salir de la comunidad		

Validación:

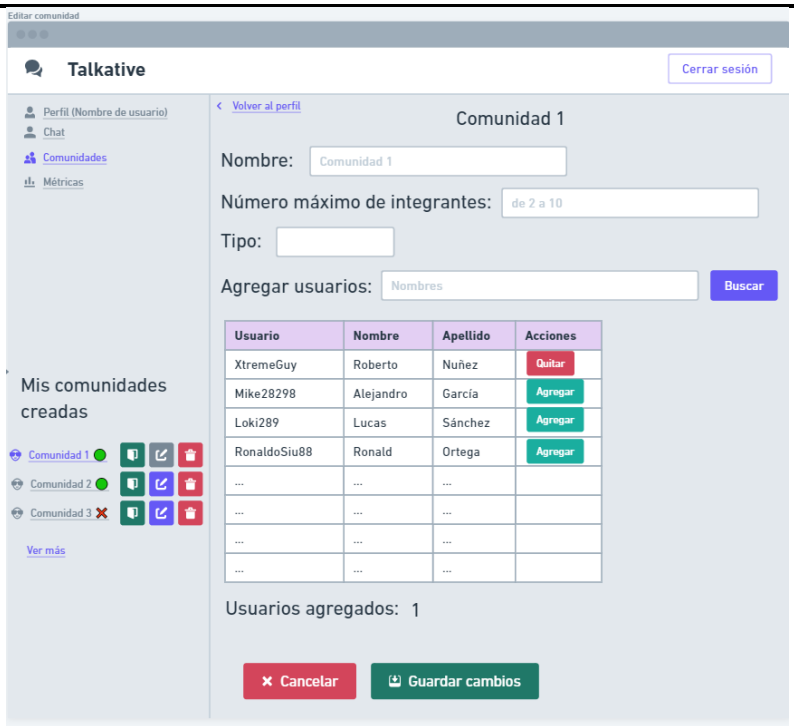
1. El usuario puede mandar un mensaje y visualizar varios mensajes
2. El usuarios podrá salir de la comunidad si lo desea.

Historia de Usuario		
Código: 13	**Usuario:** Cliente	**Orden: 3**
Nombre Historia: Crear comunidad		
Como: Cliente **Quiero:** Crear una comunidad **Para:** Tener una comunidad propia donde pueda chatear con amigos conocidos etc.		
		
Prioridad: Alta	**Riesgo de Desarrollo:** Media	
Programador Responsable: Alejandro Marshel Ayala Orosco		
Descripción: El usuario desea poder crear una comunidad con un nombre, número máximo de integrantes, el tipo y los usuarios agregados.		

Validación:

1. El usuario puede introducir un número máximo de integrantes comprendido entre 2 y 10 integrantes por comunidad.
2. El usuario podrá buscar y agregar usuarios mediante un listado.
3. El usuario podrá crear la comunidad y esta será mostrada en la parte izquierda.
4. Si la comunidad no tiene a nadie se mostrará un equis que significa que nadie esta conectado en la comunidad.
5. El usuario podrá borrar y editar una comunidad en la pestaña mis comunidades creadas.

Historia de Usuario		
Código: 14	**Usuario:** Cliente	**Orden: 3**
Nombre Historia: Editar comunidad		
Como: Cliente **Quiero:** Editar una comunidad **Para:** Poder editar una comunidad existente.		



Prioridad:	**Riesgo de Desarrollo:**
Alta	Media

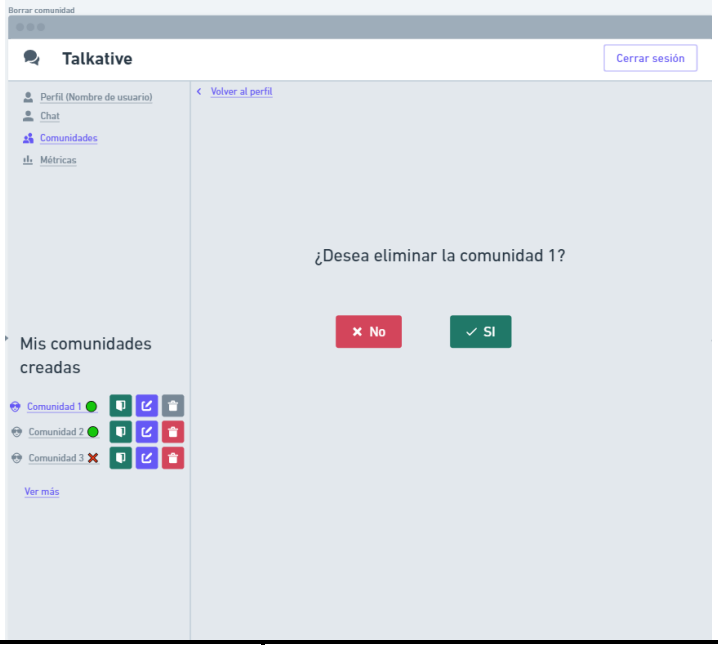
Programador Responsable:
Patrick Fernando Martínez

Descripción:

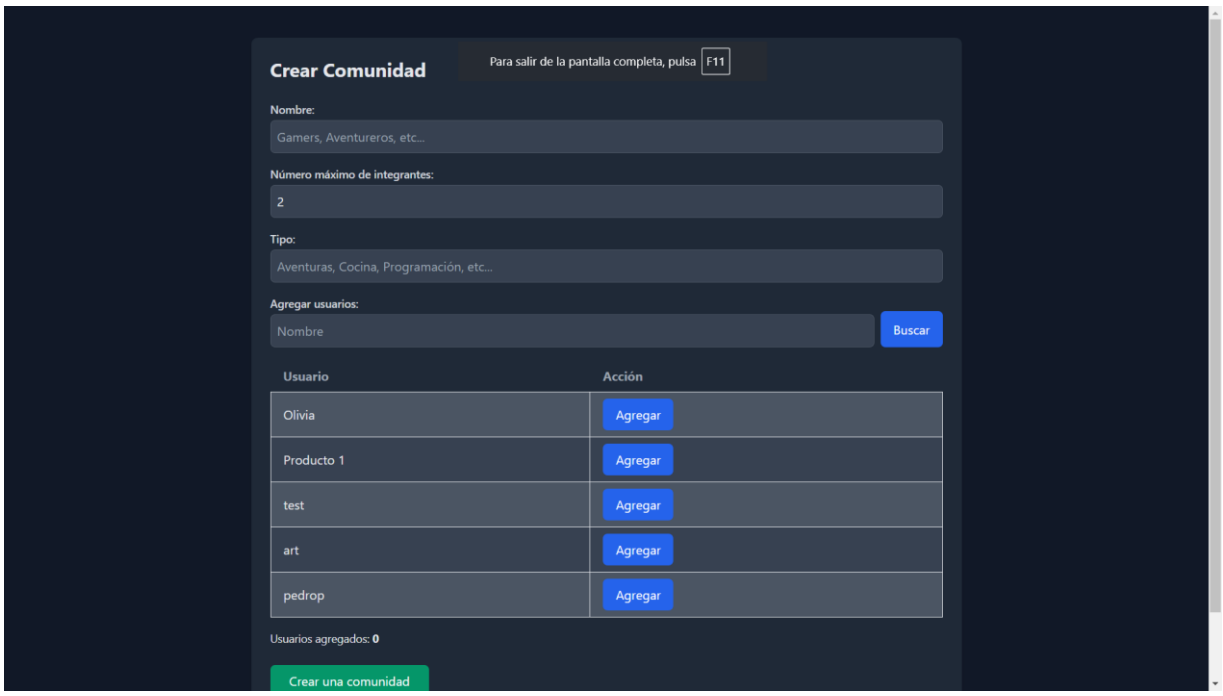
El usuario desea poder editar una comunidad con un nombre, número máximo de integrantes, el tipo y los usuarios agregados.

Validación:

1. El usuario puede introducir un número máximo de integrantes comprendido entre 2 y 10 integrantes por comunidad.
2. El usuario podrá buscar y quitar/agregar usuarios mediante un listado.
3. El usuario podrá crear la comunidad y esta será mostrada en la parte izquierda.
4. El usuario podrá cancelar o guardar los cambios de la comunidad.

Historia de Usuario		
Código: 15	**Usuario:** Cliente	**Orden: 3**
Nombre Historia: Borrar comunidad		
Como: Cliente **Quiero:** Borrar una comunidad **Para:** Poder quitar una comunidad existente.		
		
Prioridad: Alta	**Riesgo de Desarrollo:** Media	
Programador Responsable: Patrick Fernando Martínez		

Descripción: El usuario podrá borrar una comunidad.
Validación: 1. El usuario puede borrar una comunidad o no.



4. MODELO DEL SISTEMA

La arquitectura del sistema se basará en microservicios, lo que permitirá una mayor modularidad y escalabilidad. Además, se implementará un modelo de comunicación basado en mensajería/eventos, lo que facilitará la integración de nuevos servicios y la comunicación entre los diferentes componentes del sistema. La comunicación entre los microservicios se realizará principalmente a través de HTTP/REST, lo que garantizará una comunicación eficiente y fácil de implementar.

5. SEGURIDAD APLICADA

Se utilizará Bcrypt, que es una función de hashing de contraseñas y derivación de claves para contraseñas basada en el cifrado Blowfish que garantiza una alta seguridad y resistencia a los ataques de fuerza bruta. Esta función se utilizará para la encriptación de contraseñas y mensajes.

6. ALGORITMO DE SINCRONIZACIÓN

Para la sincronización de mensajes en tiempo real, se implementará el algoritmo de Cristian basado en UTC (Tiempo Universal Coordinado). Este algoritmo garantiza una sincronización precisa entre los diferentes componentes del sistema, lo que es fundamental para garantizar la consistencia y la integridad de los datos en un entorno distribuido. La combinación de estos elementos garantizará un sistema seguro, eficiente y altamente disponible.

16 DESCRIPCIÓN DE LOS ALGORITMOS

El algoritmo de Chandy-Lamport se utiliza para realizar una captura instantánea del estado de un sistema distribuido sin interrumpir su funcionamiento. Esto es útil para la detección de cortes lógicos en la comunicación entre procesos. Mientras que el algoritmo de Cristian se utiliza para sincronizar los relojes de los diferentes nodos en un sistema distribuido. Esto es importante para garantizar que los eventos se registren correctamente en el tiempo.

16.1 ALGORITMO DE CHANDY-LAMPORT

Primero se tiene la recuperación asíncrona para usuarios y mensajes:

```
234        async function fetchUsers() {
235            try {
236                const response = await fetch('http://localhost:3000/users/view');
237                if (!response.ok) {
238                    throw new Error('Network response was not ok');
239                }
240                const users = await response.json();
241                return users;
242            } catch (error) {
243                console.error('Error fetching users:', error);
244                return [];
245            }
246        }
247
248        function processUserData(users) {
249            const dailyCount = {};
250            users.forEach(user => {
251                const date = new Date(user.created_at).toISOString().split('T')[0];
252                dailyCount[date] = (dailyCount[date] || 0) + 1;
253            });
254            const labels = Object.keys(dailyCount).sort();
255            const data = labels.map(date => dailyCount[date]);
256            return { labels, data };
257        }
258
259        async function createLineChart() {
260            const users = await fetchUsers();
261            const { labels, data } = processUserData(users);
262
263            if (!labels.length) {
264                console.error('No data available to display.');
265                return;
266            }
267
```

Luego de recuperar los datos se aplica el algoritmo haciendo uso de los marcadores esto se hace para cada modelo, en este ejemplo se aplica para el modelo de comunidades:

```
function processUpdateData(users) {
    const dailyUpdates = {};
    users.forEach(user => {
        const date = new Date(user.updated_at).toISOString().split('T')[0];
        dailyUpdates[date] = (dailyUpdates[date] || 0) + 1;
    });
    const labels = Object.keys(dailyUpdates).sort();
    const data = labels.map(date => dailyUpdates[date]);
    return { labels, data };
}

async function createLineChart() {
    const users = await fetchUsers();
    const { labels, data } = processUpdateData(users);
```

16.1.1 PASOS DEL ALGORITMO DE CHANDY-LAMPOT

El algoritmo de Chandy-Lamport consta de dos fases principales:

1. **Fase de iniciación del marcador:** En esta fase, un proceso seleccionado aleatoriamente actúa como iniciador y envía un marcador a todos sus procesos vecinos. Cuando un proceso recibe un marcador por primera vez, registra su estado local y comienza a enviar marcadores a sus propios vecinos.
2. **Fase de propagación del marcador:** En esta fase, los marcadores se propagan a través de la red de procesos. Cada proceso, al recibir un marcador, registra su estado local y lo pasa a sus vecinos, asegurando que todos los procesos de la red reciban y registren el marcador.

Una vez completadas estas dos fases, cada proceso tiene una captura instantánea de su estado local y de la comunicación en la red en ese momento. Esta información se puede utilizar para detectar si ha ocurrido un corte lógico en la red distribuida.

16.1.2 APLICACIÓN EN EL PROYECTO

El algoritmo de Chandy-Lamport se utiliza para realizar una captura del estado de los mensajes enviados y recibidos en un momento dado. Esto permitiría detectar de manera precisa si ha ocurrido un corte en la comunicación entre los diferentes nodos del sistema.

16.2 ALGORITMO DE CRISTIAN

```
// ALGORITMO DE CRISTIAN
// Medir tiempo de respuesta
const sequelizeConnect = (() => {
  let cache = {};
  return (n) => {
    if (n in cache) {
      console.log(`Cache: sequelizeConnect(${n})`);
      return cache[n];
    } else {
      console.log(`Resultado: sequelizeConnect(${n})`);
      const startTime = new Date();
      sequelize.authenticate();
      const endTime = new Date();
      cache[n] = endTime - startTime;
      return endTime - startTime;
    }
  }
})();

const mongoConnect = (() => {
  let cache = {};
  return (n) => {
    if (n in cache) {
      console.log(`Cache: mongoConnect(${n})`);
      return cache[n];
    } else {
      console.log(`Resultado: mongoConnect(${n})`);
      const startTime = new Date();
      mongoose.connect(MONGO_DB_URI);
      const endTime = new Date();
      cache[n] = endTime - startTime;
      return endTime - startTime;
    }
  }
})();

// Medir la respuesta por cada conexión de servidor
console.log('~~~~~~~~~~~~~~~ TAREA 1 ~~~~~~~~~~~~~~~');
console.log(`PostgreSQL tiempo de conexión: ${sequelizeConnect(1)} ms`);
console.log(`MongoDB tiempo de conexión: ${mongoConnect(1)} ms`);

console.log('~~~~~~~~~~~~~~~ TAREA 2 ~~~~~~~~~~~~~~~');
console.log(`PostgreSQL tiempo de conexión: ${sequelizeConnect(2)} ms`);
console.log(`MongoDB tiempo de conexión: ${mongoConnect(2)} ms`);

// Calcular el promedio por cada respuesta
const numTasks = 2;
const sequelizeTotalTime = sequelizeConnect(1) + sequelizeConnect(2);
const mongoTotalTime = mongoConnect(1) + mongoConnect(2);
const AVG_TIME = sequelizeTotalTime+mongoTotalTime / numTasks;
console.log(AVG_TIME)
export {sequelize,mongoDB, pruebaConexion};
```

16.2.1 FUNCIONAMIENTO DEL ALGORITMO DE CRISTIAN

Se toma en cuenta cada conexión para cada servicio y con ayuda de una memoria en cache se almacenan los tiempos de inicio y finalización para cada conexión con el número de tareas que para el proyecto son solamente dos servicios uno para la base de datos de PostgreSQL y otro para MongoDB.

16.2.2 Pasos para EL ALGORITMO DE CRISTIAN

- Funciones de conexión a la base de datos:

`sequelizeConnect` y `mongoConnect` son funciones que intentan conectar a las bases de datos PostgreSQL y MongoDB respectivamente.

Estas funciones utilizan un caché para almacenar los tiempos de conexión previamente medidos, evitando así realizar la misma operación de conexión más de una vez si ya se ha medido antes.

- Medición del tiempo de conexión:

Se imprime el tiempo de conexión de cada base de datos para dos tareas diferentes (`TAREA 1` y `TAREA 2`) utilizando las funciones `sequelizeConnect` y `mongoConnect`.

- Cálculo del tiempo promedio de conexión:

Se calcula el tiempo promedio de conexión para ambas bases de datos, sumando los tiempos de conexión de las dos tareas y dividiendo por el número total de tareas (`numTasks`).

16.2.3 APLICACIÓN EN EL PROYECTO

El algoritmo de Cristian se utiliza en el proyecto para sincronizar los relojes de los diferentes nodos que envían y reciben mensajes. Esto garantizaría que los mensajes se registren con precisión en el momento en que fueron enviados y recibidos.

17 DESCRIPCIÓN DE MÓDULOS.

- **MODULO DE PELÍCULAS**

En este módulo el usuario puede visualizar las diferentes películas existentes en el sistema dependiendo al género que desea ver, además puede realizar una búsqueda con el nombre de la película escogida.