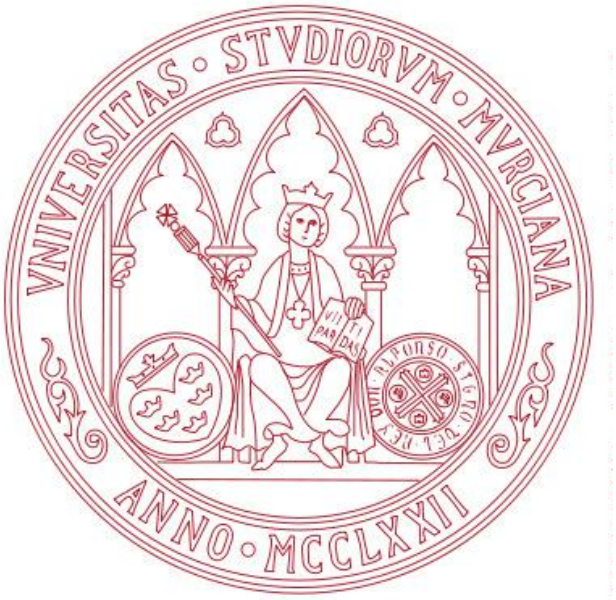

Universidad de Murcia

Facultad de Ingeniería Informática

Informe Proyecto LlamaChat

Sistemas Distribuidos

José Hurtado García - email: jose.hurtadog@um.es
Nerea Bastida Mateo - email: nerea.bastidam@um.es

Curso 2023/2024
Convocatoria Julio

Índice

Índice de figuras

1. Introducción

En primer lugar, el desarrollo de la práctica se ha realizado usando Git y GitHub para el control de versiones y trabajo en equipo. Hemos realizado dicha práctica en el siguiente repositorio: https://github.com/pepehurtado/ssdd-proyecto.

Este documento presenta una descripción del desarrollo realizado para la creación de una aplicación distribuida formada por un conjunto de servicios: gestión de usuarios, estadísticas de acceso y de petición de servicios, servicio de sesión de prompt LlamaChat y servicio de log de sesiones de prompt. Esta aplicación recibe el nombre de **LlamaChat**.

Para conseguir implementar el conjunto de servicios mencionados anteriormente, se hace uso de distintas tecnologías: REST, gPRC, Kafka y Docker (entre otras).

El modelo seguido, proporcionado por los profesores de la asignatura es el de la figura 1. En dicho modelo, se observa la estructura de la aplicación y los servicios implementados para cada capa de aplicación.

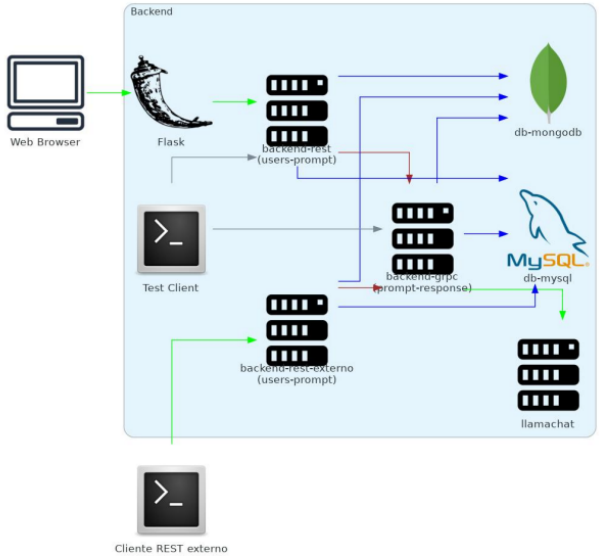

Figura 1: Modelo a seguir

En la parte de **frontend**, se ofrece servicio a través de un acceso web. Implementando tecnologías de cliente como *Bootstrap* y para la parte servidor, se utiliza Python Flask.

En la parte de **backend**, se incluyen los servidores relativos a la vista y los servidores que implementan los servicios REST y gPRC.

Para la capa de **datos** se utilizan una base de datos para almacenar los datos, en nuestro caso, **MongoDB**.

2. Servicios desplegados

2.1. Frontend

Frontend incluye una interfaz web para realizar diversas tareas, entre ellas: registro de usuarios; pantalla de login; iniciar sesión de prompt; administración de logs de conversaciones; estadísticas de acceso...

A continuación se describen las rutas implementadas en el código:

- **'/'**: se accede a la página de inicio, es decir, `index.html`.
- **'/signup'**: contiene el formulario de registro, utilizando `SignupForm`. La página web pide el nombre de usuario (id), nombre (name), correo electrónico (email) y contraseña (password). Estos datos se recogen en formato JSON y son enviados al servidor REST, concretamente a la clase `UsersEndpoint`. De esta función se obtiene un mensaje que determina si se ha creado con éxito el usuario o, de lo contrario, no ha sido creado.
- **'/login'**: contiene el formulario de login, utilizando `LoginForm`. Al igual que con `'/signup'`, se recogen los datos introducidos (correo electrónico y contraseña) y se envían a REST. Si los datos han sido correctos, se obtiene un usuario *User*.
- **'/profile'**: el usuario es capaz de acceder a su perfil y consultar los datos de sesión y estadísticas. En esta ruta, se le muestra al usuario la opción de eliminar usuario.
- **'/delete_user'**: se elimina un usuario de la base de datos. También utiliza REST, por lo que se envía un mensaje DELETE.
- **'/logout'**: se cierra la sesión del usuario actual.
- **'/dialogue'**: permite al usuario ver una lista de todos sus diálogos. Utiliza el formulario `DialogueForm` para enviar nuevos diálogos al servidor REST.
- **'/dialogue/<dialogueId>'**: permite al usuario ver los detalles de un diálogo específico identificado por `dialogueId`. Realiza una solicitud GET al servidor REST para obtener los datos del diálogo.
- **'/dialogue/<dialogueId>/update_dialogue'**: permite al usuario actualizar un diálogo específico. Se puede acceder mediante solicitudes GET, POST o PUT. La solicitud PUT envía los datos actualizados del diálogo al servidor REST.
- **'/dialogue/<dialogueId>/delete_dialogue'**: permite al usuario eliminar un diálogo específico mediante una solicitud POST, la cual envía un mensaje DELETE al servidor REST.
- **'/dialogue/<dialogueId>/chat'**: permite al usuario acceder a la interfaz de chat de un diálogo específico. Realiza una solicitud GET al servidor REST para obtener los mensajes del diálogo.
- **'/chat/<dialogueId>/send_message'**: permite al usuario enviar un mensaje en un diálogo específico. Realiza una solicitud POST al servidor REST con el nuevo mensaje.

2.2. Backend

2.2.1. REST

- **GET /username —— getUserInfo**: Obtiene la información del usuario identificado por `username`. Retorna un objeto UserDTO en formato JSON que representa la información del usuario encontrado por el email asociado al username. Si no se encuentra ningún usuario, retorna null.

- **POST /register —— addUser**: Registra un nuevo usuario utilizando los datos proporcionados en formato JSON. Convierte el objeto UserDTO recibido a un objeto User y lo guarda usando la lógica de la aplicación. Retorna un código de estado 201 (Created) junto con el objeto UserDTO del usuario creado si la operación es exitosa. En caso contrario, retorna un código de estado 400 (Bad Request) con un mensaje de error.

- **DELETE /username —— deleteUser**: Intenta eliminar al usuario identificado por el nombre de usuario `username`. Retorna un código de estado 204 (No Content) si el usuario se elimina correctamente. En caso contrario, retorna un código de estado 404 (Not Found) junto con un mensaje indicando que el usuario no fue encontrado.

- **POST /username/dialogue —— createDialogue**: Crea un nuevo diálogo asociado al usuario identificado por `username`. Recibe un objeto DialogueDTO en formato JSON que contiene el identificador del diálogo. Verifica que el identificador del diálogo no contenga espacios ni esté vacío. Si el diálogo se crea correctamente, retorna un código de estado 201 (Created) y la ubicación del recurso creado en la cabecera "Location". En caso de que el diálogo ya exista, retorna un código de estado 409 (Conflict). En caso de que el identificador del diálogo contenga espacios o esté vacío, retorna un código de estado 400 (Bad Request) con un mensaje explicativo.

- **GET /username/dialogue —— getUserDialogues**: Obtiene los diálogos asociados al usuario identificado por `username`. Retorna una lista de objetos Dialogue en formato JSON que representan los diálogos del usuario. Si el usuario no es encontrado, retorna un código de estado 404 (Not Found) con un mensaje indicando que el usuario no fue encontrado. En caso de error interno del servidor, retorna un código de estado 500 (Internal Server Error) con un mensaje de error.

- **PUT /username/dialogue/dialogueId —— updateDialogue**: Actualiza el diálogo identificado por `dialogueId` del usuario identificado por `username` con los datos proporcionados en formato JSON en el cuerpo de la solicitud. Retorna un código de estado 200 (OK) si el diálogo se actualiza correctamente. Si el diálogo no es encontrado, retorna un código de estado 404 (Not Found) con un mensaje indicando que el diálogo no fue encontrado. En caso de error interno del servidor, retorna un código de estado 500 (Internal Server Error) con un mensaje de error.

- **DELETE /username/dialogue/dialogueId —— deleteDialogue**: Elimina el diálogo identificado por `dialogueId` del usuario identificado por `username`. Retorna un código de estado 200 (OK) si el diálogo se elimina correctamente. Si el diálogo no es encontrado, retorna un código de estado 404 (Not Found) con un mensaje indicando

que el diálogo no fue encontrado. En caso de error interno del servidor, retorna un código de estado 500 (Internal Server Error) con un mensaje de error.

- **GET /username/dialogue/dialogueId —— getDialogue**: Obtiene el diálogo identificado por `dialogueId` del usuario identificado por `username`. Retorna un objeto Dialogue en formato JSON que representa el diálogo solicitado. Si el usuario no es encontrado, retorna un código de estado 404 (Not Found) con un mensaje indicando que el usuario no fue encontrado. Si el diálogo no es encontrado, retorna un código de estado 404 (Not Found) con un mensaje indicando que el diálogo no fue encontrado. En caso de error interno del servidor, retorna un código de estado 500 (Internal Server Error) con un mensaje de error.
- **POST /username/dialogue/dialogueId/nextUrl —— addPrompt**: Agrega un nuevo prompt al diálogo identificado por `dialogueId` del usuario identificado por `username`. Recibe un objeto PromptDTO en formato JSON que representa el prompt a agregar, junto con el parámetro adicional `nextUrl`. Si el prompt se agrega correctamente, retorna un código de estado 201 (Created) y un mensaje indicando que el prompt fue enviado. Si el diálogo no es encontrado, retorna un código de estado 404 (Not Found) con un mensaje indicando que el diálogo no fue encontrado. En caso de error interno del servidor, retorna un código de estado 500 (Internal Server Error) con un mensaje de error.

2.2.2. gRPC

El servicio gRPC permite la comunicación entre diferentes componentes y facilita el manejo de peticiones de respuestas a prompts. Este servicio está compuesto por dos proyectos: **GrpcService** y **GrpcServiceImpl**.

2.2.2.1 GrpcService

Contiene los stubs y skeletons necesarios para gRPC. El fichero *grpcservice.proto* define los tipos de datos, servicios y operaciones necesarias.

Ofrece dos operaciones: *Ping* y *SendPrompt*.

- **Ping**: define un método que toma como entrada *PingRequest* y devuelve un *PingResponse*. Ambos contienen un solo campo *"v"*. Se utiliza para enviar y recibir un valor entero, para verificar la conectividad y funcionalidad del servicio.
- **SendPrompt**: define un método que toma como entrada *PromptRequest* y devuelve un *PromptResponse*. *PromptRequest* contiene 4 campos:
  - **"prompt"**: representa el texto del prompt que se envía.
  - **"dialogueId"**: identifica el diálogo al que pertenece el prompt.
  - **"timestamp"**: marca el tiempo en el que se envió el prompt.
  - "**userId"**: identifica el usuario que envió el prompt.

  PromptResponse contiene un campo *"success"*, un booleano que indica si la operación de enviar el prompt fue exitosa o no.

2.2.2.2 GrpcServiceImpl

Implementa el servidor gRPC y utiliza un DAO para la gestión de datos. Se incluye la funcionalidad *Ping* y la gestión de peticiones de prompts mediante el uso de hilos.

De la siguiente forma, la clase **GrpcServiceImpl** maneja las peticiones de *Ping* y *SendPrompt*.

```
@Override
public void ping(PingRequest request, StreamObserver<PingResponse>
    responseObserver) {
    logger.info("Received␣PING␣request,␣value␣=␣" + request.getV());
    responseObserver.onNext(PingResponse.newBuilder().setV(request.getV()).
       build());
    responseObserver.onCompleted();
}

@Override
public void sendPrompt(PromptRequest request, StreamObserver<PromptResponse>
     responseObserver) {
    HilosConversaciones hiloChat = new HilosConversaciones(request,
       responseObserver, cont++, dao);
    hilosChat.add(hiloChat);
    hiloChat.start();
}
```

Código 1: Implementación de los métodos ping y sendPrompt en GrpcServiceImpl

La clase **HilosConversaciones** gestiona las peticiones de prompts en un hilo separado, permite manejar operaciones de manera asíncronica. Se encarga de procesar una solicitud de prompt, enviar el prompt a un servicio externo, recibir la respuesta y actualizar la base de datos.

El método *run* es el punto de entrada del hilo cuando éste se inicia.

```
    @Override
    public void run() {
        LocalDateTime timestamp = LocalDateTime.parse(request.getTimestamp()
           );
        Prompt promptMensaje = new Prompt(request.getPrompt(), "", timestamp
           );
        String token = enviarLlamaChat(promptMensaje.getPrompt());
        String respuesta = getLlamaChatResponse(token);
        promptMensaje.setAnswer(respuesta);
        dao.addPromptRespuesta(request.getUserId(), request.getDialogueId(),
            promptMensaje);
```

```
        dao.updateDialogueEstado(request.getUserId(), request.getDialogueId
            (), DialogueEstados.READY);
    }
```

Código 2: Método run

- Convierte la marca de tiempo (timestamp) en un objeto *LocalDateTime*.
- Crea un nuevo objeto *prompt*.
- Envía el prompt al servicio *LlamaChat* y obtiene token de seguimiento.
- Usa el token para obtener respuesta del servicio *LlamaChat*.
- Actualiza el objeto *prompt* con la respuesta.
- Añade la respuesta a la base de datos.
- Actualiza el diálogo a ***READY***.

Con el método *enviarLlamaChat*, se envía el prompt a *LlamaChat* y se obtiene el token de seguimiento.

Con el método *getLlamaChatResponse*, se obtiene la respuesta de *LlamaChat*. Intenta obtener la respuesta hasta 100 veces. Si la respuesta tiene un estado 200, se extrae la respuesta del cuerpo de la respuesta.

2.2.3. Nivel de datos

Finalmente, se implementa una única base datos utilizando **MongoDB**.
Se utiliza un patrón DAO, en la clase IUserDAO:

```
package es.um.sisdist.backend.dao.user;

import java.util.Optional;

import es.um.sisdist.backend.dao.models.Dialogue;
import es.um.sisdist.backend.dao.models.DialogueEstados;
import es.um.sisdist.backend.dao.models.Prompt;
import es.um.sisdist.backend.dao.models.User;

public interface IUserDAO {
    public Optional<User> getUserById(String id);

    public Optional<User> getUserByEmail(String email);
```

```
    boolean addUser(User user);

    boolean deleteUser(String username);

    boolean addVisits(String username);

    boolean createDialogue(String userId, Dialogue dialogue);

    boolean updateDialogue(String userId, String dialogueId, Dialogue
       dialogue);

    boolean deleteDialogue(String userId, String dialogueId);

    boolean addPrompt(String userId, String dialogueId, String nextUrl,
       Prompt prompt);

    boolean addPromptRespuesta(String userId, String dialogueId, Prompt
       prompt);

    boolean updateDialogueEstado(String userId, String dialogueId,
       DialogueEstados status);

    Dialogue getDialogue(String userId, String dialogueId);
}
```

Código 3: IUserDAO

En este código se realizan funciones para la gestión de usuarios, diálogos y prompts.

Queremos destacar la siguiente función la cual no hemos especificado antes:

- **Método 'addPromptRespuesta'**: Este método busca agregar una respuesta a un prompt dentro de un diálogo específico de un usuario en la base de datos. Primero, busca al usuario por su 'userId'. Si el usuario no existe, registra un mensaje de información y retorna 'false'. Luego, itera sobre los diálogos del usuario buscando el diálogo con el 'dialogueId' proporcionado. Compara el timestamp del prompt a agregar con los prompts existentes en el diálogo. Si encuentra un prompt con el mismo timestamp, actualiza su respuesta y marca el diálogo como listo. Guarda los cambios en la base de datos y retorna 'true' si se modifica algún diálogo. Si no encuentra el prompt con el mismo timestamp, registra un mensaje y retorna 'false'. Si no encuentra el diálogo con el 'dialogueId', registra un mensaje y retorna 'false'. Maneja cualquier excepción durante el proceso y registra un mensaje de error.

2.2.4. Servicio externo REST

El servidor REST externo está disponible en el puerto 8180. Este servidor permite al cliente externo, no hacer uso del frontend. Por ello, es necesaria la autenticación del usuario

para asegurar la integridad y seguridad de las interacciones.

Para realizar la autenticación de las peticiones al servicios se requieren tres cabeceras: *User*, *Date* y *Auth-Token*. Para generar el *Auth-Token* se concatena la url de la petición, la fecha en formado ISO8601 y el token privado del usuario.

A continuación, se explica el código 4, está implementado en la clase **UsersEndpoint** y se comprueba en todas las funciones de la misma para saber si el usuario accede desde el servicio externo o no.

```
    private boolean authenticateUser(UriInfo uriInfo, HttpHeaders headers) {
        Logger logger = Logger.getLogger(UsersEndpoint.class.getName());
        String ServerAux = System.getenv("AUX_SERVER");
        logger.info("ServerAux: " + ServerAux);
        if (ServerAux == null || "false".equals(ServerAux)) {
            return true;
        }
        String user = headers.getHeaderString("User");
        String date = headers.getHeaderString("Date");
        String authToken = headers.getHeaderString("Auth-Token");
        if (user == null || date == null || authToken == null) {
            logger.severe("Faltan cabeceras:\n" + "User: " + user + "\nDate:
                " + date + "\nAuth-Token: " + authToken);
            return false;
        }
        User u = impl.getUserById(user).orElse(null);
        if (u == null) {
            logger.severe("Usuario no encontrado: " + user);
            return false;
        }
        try {
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd'T'
                HH:mm:ss.SSS'Z'");
            dateFormat.setLenient(false);
            dateFormat.parse(date);
        } catch (Exception e) {
            logger.severe("Formato fecha invalida");
            return false;
        }
        logger.info("URL: " + uriInfo.getRequestUri() + "\nDATE: " + date +
            "\nTOKEN: " + u.getToken());
        String expectedToken = UserUtils.md5pass(uriInfo.getRequestUri().
            toString() + date + u.getToken());
        logger.info("expectedToken: " + expectedToken + " authtoken: " +
            authToken);
```

```
        if (!expectedToken.equals(authToken)) {
            logger.severe("Token␣Auth␣invalido␣del:␣" + user + ".␣Esperado:␣"
                + expectedToken + ",␣recibido:␣" + authToken);
            return false;
        }
        logger.info("User␣" + user + "␣autenticado␣en␣el␣servidor␣REST␣
           externo");
        return true;
    }
```

Código 4: authenticateUser

Este método está diseñado para autenticar usuarios en el servidor REST. Se utilizan dos parámetros: *UriInfo* y *HttpHeaders*, que se utilizan para extraer la información necesaria para la autenticación.

El código realiza una serie de comprobaciones como la variable entorno *AUX_SERVER*; los encabezados *User*, *Date* y *Auth-Token*; obtener un usuario por su id; el formato del encabezado de fecha.

Tras realizar las comprobaciones, si en todas se obtiene un resultado correcto. Se crea el token (explicado anteriormente) y se compara con el token esperado. Si coincide, el usuario ha sido autenticado correctamente.

3. Manual de Usuario

Para comenzar, el usuario se encuentra en la página de inicio (figura 2). Desde dicha página, tiene acceso a *Login!* y *Sign up*.

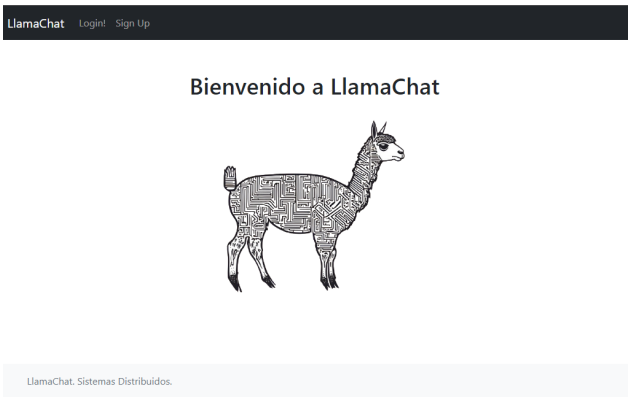

Figura 2: Página de inicio

Para realizar el registro de usuario, se accede a *Sign up* (figura 3), donde se pide: username, name, email y password.

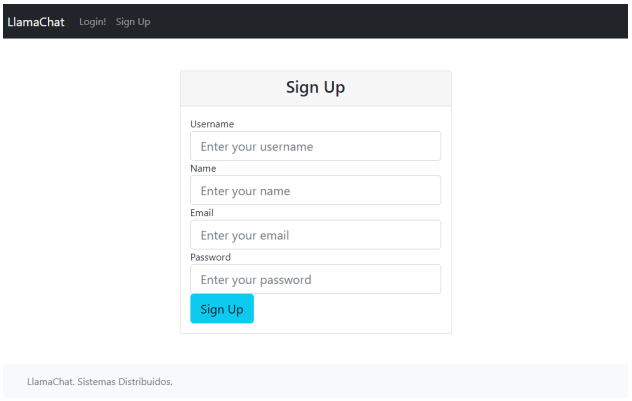

Figura 3: Sign up

Una vez registrado, el usuario es redirigido a *Login!*, donde debe introducir el *email* y *password* previamente introducidos (figura 4).

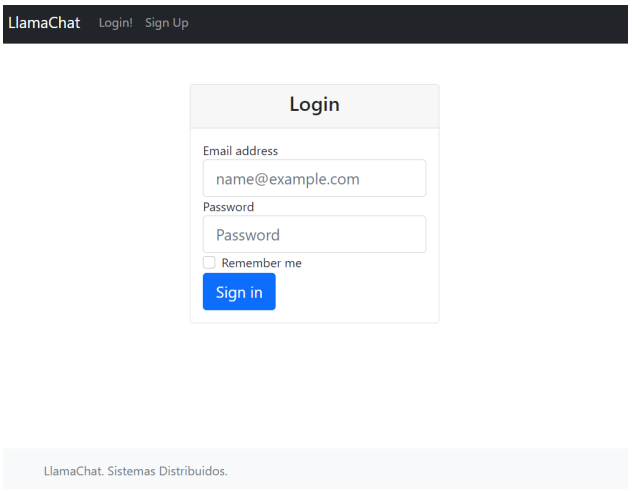

Figura 4: Login

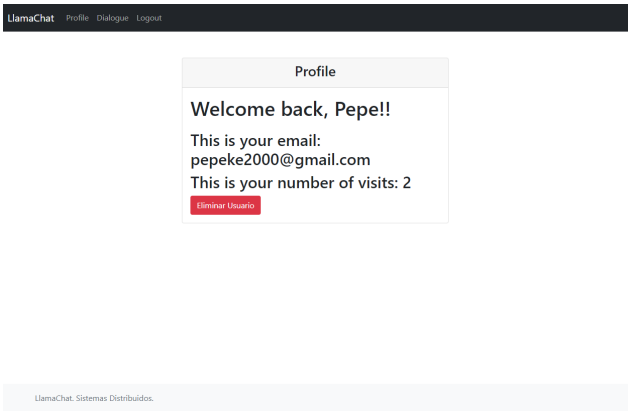

Figura 5: Profile

A continuación, el usuario tiene acceso a cierta información en *Profile*, además de la opción de eliminar el usuario (figura 5). Al estar autenticado, también tiene acceso a *Dialogue* y *Logout*. Este último se utiliza para cerrar la sesión.

En la página de *Dialogue* se pueden crear nuevos diálogos (6). Una vez creado un diálogo, aparecen cuatro opciones en el mismo: *Ver detalles*, *Modificar*, *Eliminar*, *Chatear* (figura 7).

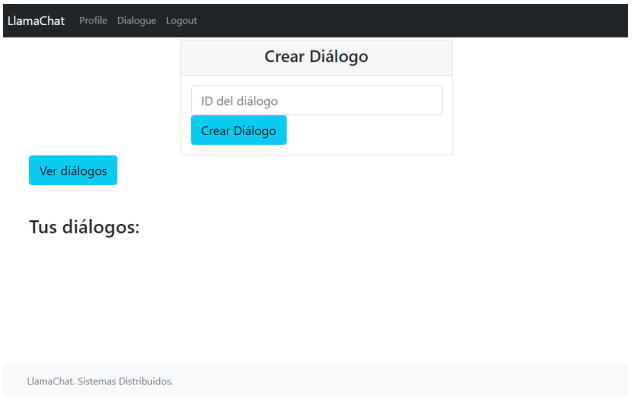

Figura 6: Dialogue

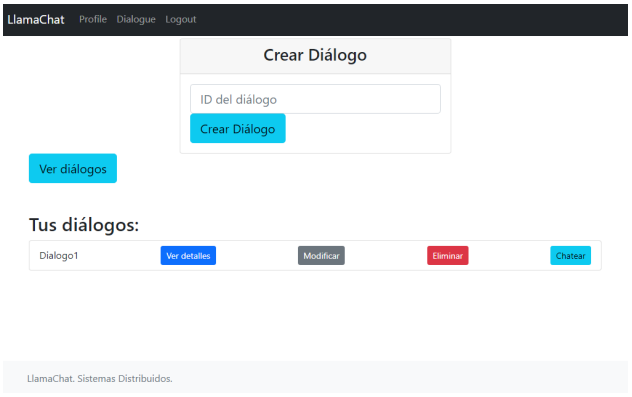

Figura 7: Opciones del diálogo

En *Ver detalles*, se obtiene la siguiente información: *status*, *next URL*, *end URL* y *prompts*. Desde esa página, también aparecen las opciones de *Modificar* y *Eliminar* (figura 8).

En *Modificar*, aparece un campo de texto para introducir el nuevo nombre del diálogo (figura 9).

Finalmente, en *Chatear*, el usuario tiene acceso a un chat donde puede realizar cuestiones que serán respondidas por LlamaChat (figura 10).

4. TestClient

Para la realización de *tests* en REST interno, se modifica la clase **TestClient** para realizar las siguientes pruebas:

Figura 8: Detalles del diálogo

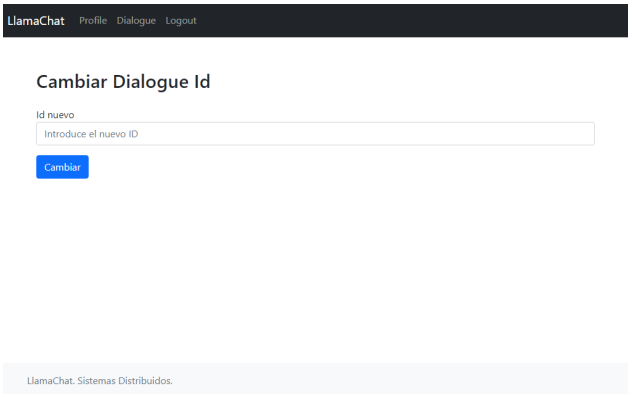

Figura 9: Modificar ID del diálogo

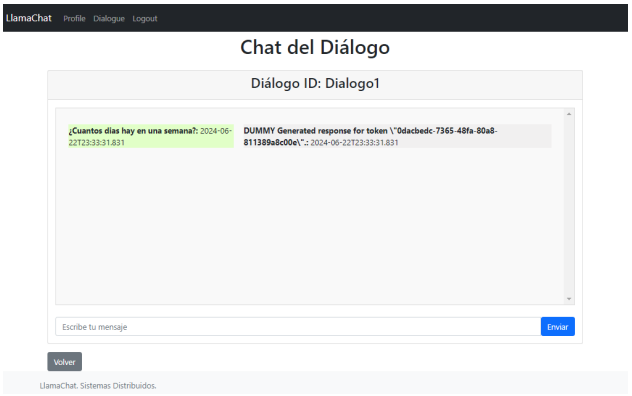

Figura 10: Conversación con LlamaChat

1. **Eliminar el usuario 'usuario1'**:
   - **Método HTTP**: DELETE
   - **Descripción**: Elimina el usuario 'usuario1' si existe en el sistema.
2. **Creación del usuario 'usuario1'**:
   - **Método HTTP**: POST

- **Descripción**: Crea un nuevo usuario 'usuario1'.

3. **Login de usuario 'usuario1'**:
   - **Método HTTP**: POST
   - **Descripción**: Inicia sesión con el usuario 'usuario1' utilizando el servicio `checkLogin`.
4. **Consulta el usuario 'usuario1'**:
   - **Método HTTP**: GET
   - **Descripción**: Obtiene la información del usuario 'usuario1'.
5. **Creación diálogo 'test'**:
   - **Método HTTP**: POST
   - **Descripción**: Crea un nuevo diálogo con ID 'test' asociado al usuario 'usuario1'.
6. **Consulta 'test'**:
   - **Método HTTP**: GET
   - **Descripción**: Obtiene información del diálogo 'test' asociado al usuario 'usuario1'.
7. **Modificar 'test' a 'dialogoClient'**:
   - **Método HTTP**: PUT
   - **Descripción**: Modifica el ID del diálogo 'test' a 'dialogoClient'.
8. **Consulta 'dialogoClient'**:
   - **Método HTTP**: GET
   - **Descripción**: Obtiene información del diálogo 'dialogoClient'.
9. **Prompt en 'dialogoClient'**:
   - **Método HTTP**: POST
   - **Descripción**: Realiza un prompt en el diálogo 'dialogoClient'.
10. **Consulta 'dialogoClient'**:
    - **Método HTTP**: GET
    - **Descripción**: Obtiene información actualizada del diálogo 'dialogoClient'.
11. **Consulta 'dialogoClient' después de 5 segundos para comprobar estado**:
    - **Método HTTP**: GET
    - **Descripción**: Obtiene información actualizada del diálogo 'dialogoClient'.
12. **Consulta diálogos de 'usuario1'**:

- **Método HTTP**: GET
- **Descripción**: Obtiene todos los diálogos asociados al usuario 'usuario1'.

13. **Eliminar 'dialogoClient'**:
    - **Método HTTP**: DELETE
    - **Descripción**: Elimina el diálogo 'dialogoClient'.
14. **Consulta diálogos de 'usuario1'**:
    - **Método HTTP**: GET
    - **Descripción**: Obtiene todos los diálogos asociados al usuario 'usuario1' después de eliminar 'dialogoClient'.
15. **Eliminar el usuario 'usuario1'**:
    - **Método HTTP**: DELETE
    - **Descripción**: Elimina completamente al usuario 'usuario1' del sistema.
16. **Consulta el usuario 'usuario1'**:
    - **Método HTTP**: GET
    - **Descripción**: Intenta obtener información del usuario 'usuario1'.