Universidad de San Buenaventura

https://www.usbbog.edu.co/

Descripción breve

En este documento se describe como utilizar el microservicio de SignChat.

Manual de usuario

Plugin SignChat

Contenido

[Introducción 2](#_Toc149667174)

[¿Qué es SignChatMS? 2](#_Toc149667175)

[Requisitos 2](#_Toc149667176)

[Como empezar 2](#_Toc149667177)

[Descargando el plugin 2](#_Toc149667178)

[Usando el web service 2](#_Toc149667179)

[Teclado LSC 3](#_Toc149667180)

[Como llamar el teclado LSC 3](#_Toc149667181)

[Funciones 4](#_Toc149667182)

[Configuraciones en página web 5](#_Toc149667183)

[Página web con JavaScript 5](#_Toc149667184)

[Página web con Php 9](#_Toc149667185)

[Configuraciones en aplicación Móvil 10](#_Toc149667186)

[Mensajes 10](#_Toc149667187)

[Traducción de mensajes 10](#_Toc149667188)

[Traducción de texto a imágenes LSC 10](#_Toc149667189)

[Traducción de imágenes LSC a texto 11](#_Toc149667190)

[Comunicación en tiempo real 12](#_Toc149667191)

[SockJS 13](#_Toc149667192)

[StompJS 13](#_Toc149667193)

[Funcionalidades extra 14](#_Toc149667194)

[Lista de rutas de la API 14](#_Toc149667195)

[Base de datos 15](#_Toc149667196)

[Usuarios 16](#_Toc149667197)

[Mensajes 18](#_Toc149667198)

[Lista de contactos 19](#_Toc149667199)

[Plantillas HTML 21](#_Toc149667200)

# Introducción

## ¿Qué es SignChatMS?

SignChatMS es un plugin para implementar traducción de lenguaje de señas colombiano en texto español latino y viceversa. El objetivo es facilitar la comunicación entre la gente con discapacidad sordomuda y sus contactos. Este plugin puede ser implementando en cualquier aplicación utilizando ya sea el archivo JAR o la URL dada que se tiene.

## Requisitos

Si vas a usar el plugin necesitas:

* Java 17 o superior.
* Maven
* Algún IDE de desarrollo en Java (opcional, es para mejor experiencia en su uso).

Si vas a usar la URL, no se requiere de ningún programa externo. Solo usa tu editor de código de preferencia y te recomendamos guardar la URL de SignChatMS en tu código como una variable global.

# Como empezar

## Descargando el plugin

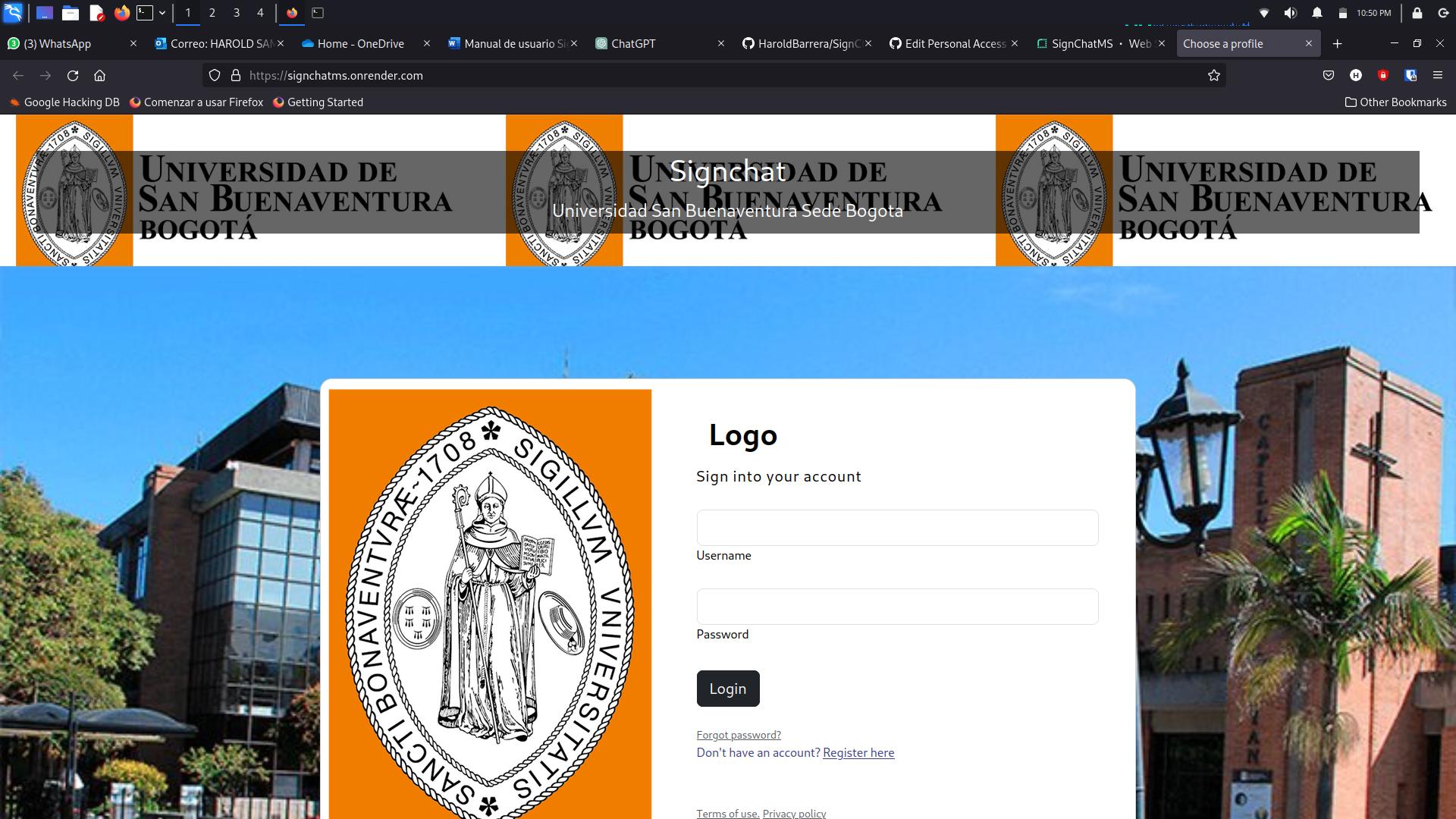
Puedes descargar el plugin visitando nuestro [repositorio de GitHub](https://github.com/HaroldBarrera/SignChatMS-Plugin). Descargarlo te dará un archivo JAR el cual debes importar dentro de las librerías de tu proyecto.

A partir de aquí, dependiendo del IDE o editor de código que utilices, deberías seguir una serie de pasos para poner el archivo JAR dentro de las librerías de tu aplicación. Aquí hay un [ejemplo](https://www.geeksforgeeks.org/how-to-add-external-jar-file-to-an-intellij-idea-project/) si usas el IDE de IntelliJ.

Una vez instalado, cada vez que ejecutes tu aplicación, tendrás acceso a todas las rutas desde un ambiente local en el puerto 8080, es decir, en [*www.localhost:8080/*](http://www.localhost:8080/).

## Usando el web service

Puedes acceder a todas las funcionalidades que tiene el plugin utilizando la siguiente URL <https://signchatms.onrender.com>. Es recomendable guardar esa URL en alguna variable dentro de tu código para acceder fácilmente a ella.

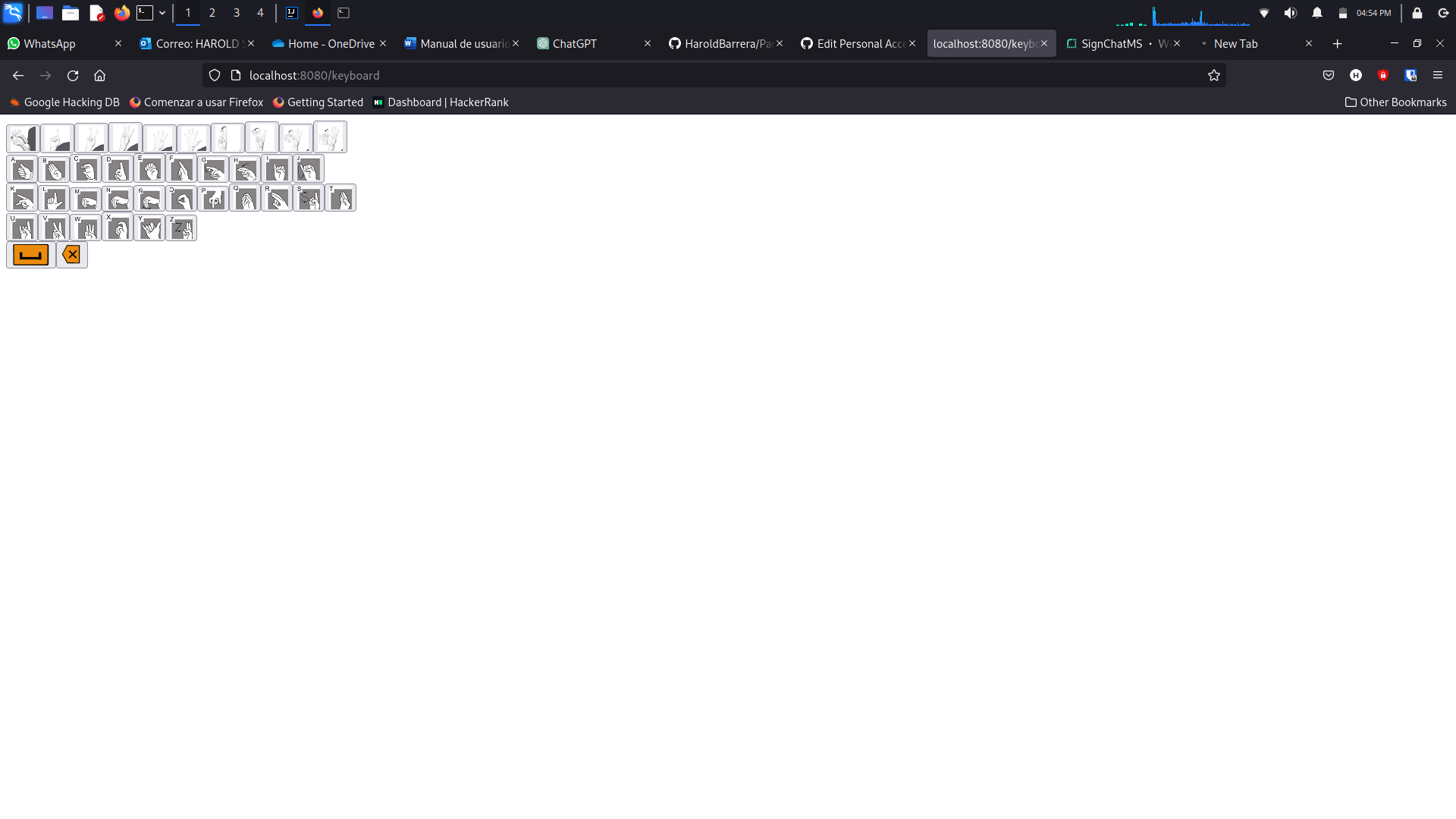


Recuerda también que, al usar nuestra URL, aunque no necesitaras descargar ninguna librería externa, si es importante recordar que la velocidad de funcionamiento se verá afectada por tu conexión a internet.

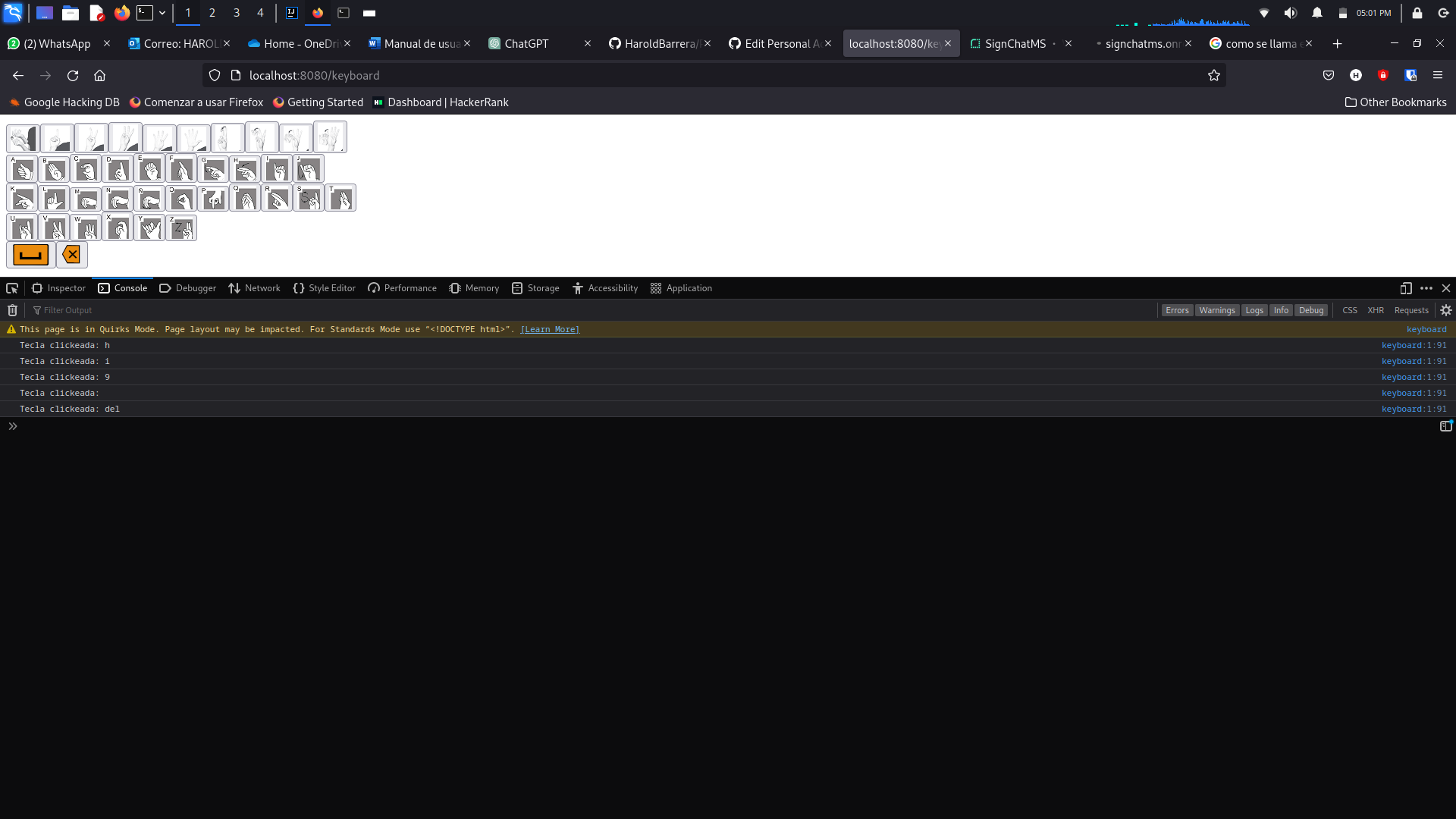
# Teclado LSC

## Como llamar el teclado LSC

Para llamar el teclado de lenguajes de señas colombiano (LSC), debes acceder a la ruta */keyboard*. Si visitas esa ruta, deberías ver lo siguiente:



A partir de aquí puedes probar haciendo clic en los diferentes botones. Puedes ver, desde el panel del navegador que puedes abrir al oprimir F12, que al hacer clic se imprime el valor de dicha tecla que oprimiste con su traducción.

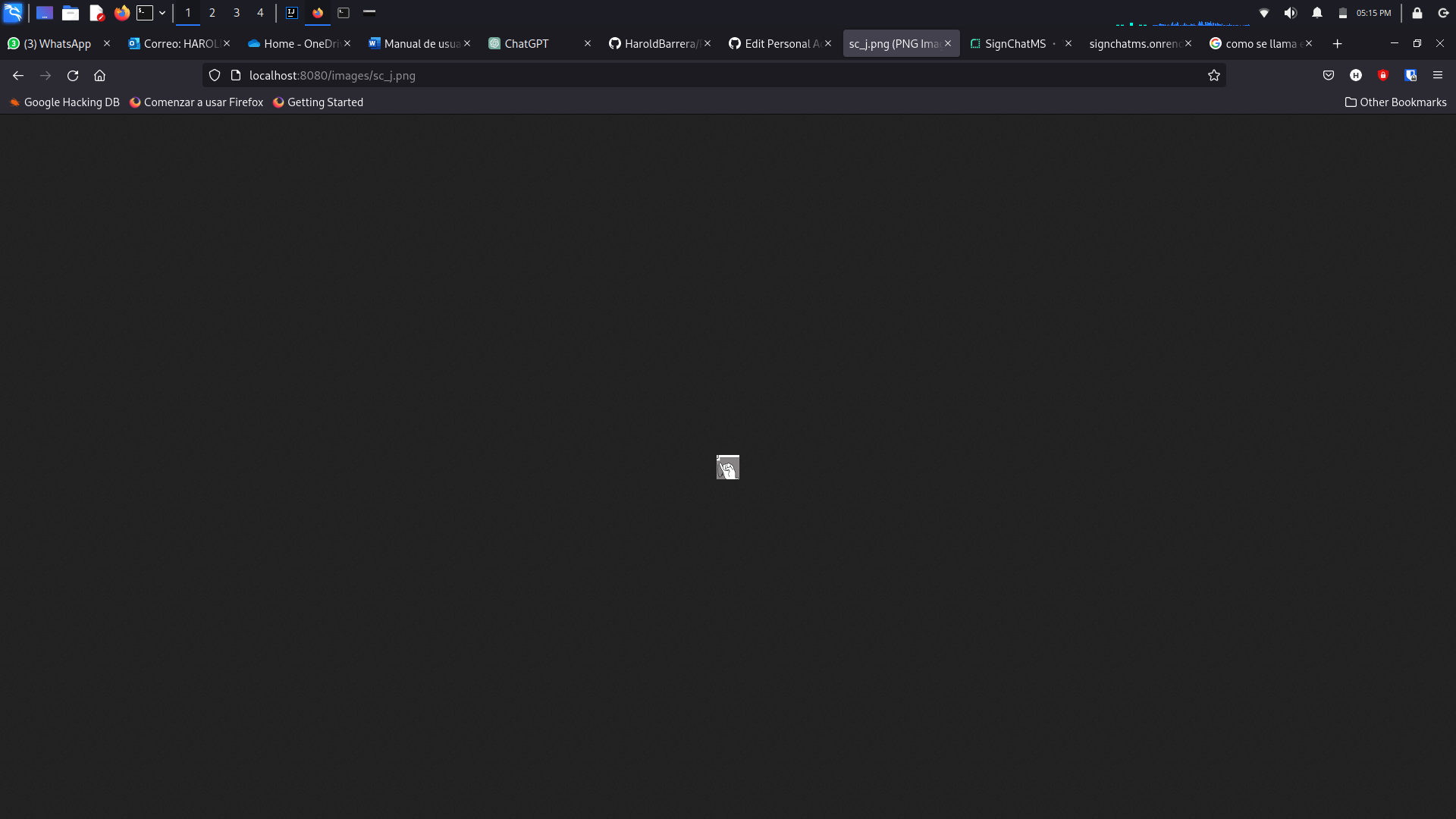


A partir de este punto, ya puede utilizar el teclado LSC en tu aplicación. Ten en cuenta que cuando lo llamas mediante la ruta */keyboard* estas llamando al elemento HTML del teclado, por lo tanto, también sería buena idea crear algún tipo de componente para el mismo dentro de tu aplicación, en caso de que planees usarlo en más de una página dentro de tu software.

## Funciones

Además de la ruta */keyboard*, el teclado también tiene otras rutas definidas para casos más específicos.

Una de sus funciones es poder llamar individualmente la imagen de cada letra del abecedario en LSC. Para lograr eso, debemos llamar la ruta */images**/[nombre de la imagen]* donde el nombre de la imagen debe ser un String que comience con “sc\_” y seguido de la letra o número que deseas ver con “.png” al final. Por ejemplo, si quieres ver la letra J en LSC, la ruta seria */images/sc\_j.png*. Aquí la demostración.



Estas imágenes individuales son pequeñas, pero pueden ser utiles para alguna simbologia que quieras darle a tu aplicación. Para algunas teclas especiales, los Strings son algo distintos, aquí su listado.

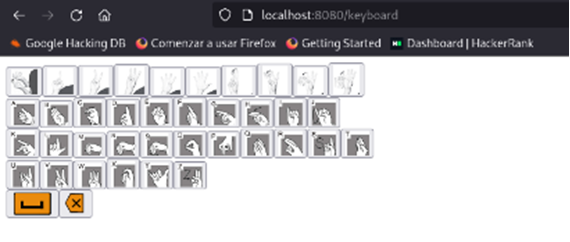
* DELETE -> *delete\_key.png*
* SPACE -> *space\_view.png*

Otras funciones que se tendrán a futuro son los teclados en LSC para diferentes frameworks web específicos. Por el momento, se cuenta con el teclado LSC para React, el cual ha demostrado funcionar no solo en React sino en otros frameworks de JavaScript como Angular, y para llamarlo se utiliza */keyboard/react*.

## Configuraciones en página web

### Página web con JavaScript

Para la Implementación del plugin en una página web utilizando JavaScript es bastante sencillo, para empezar en el código HTML de la página web se debe realizar la solicitud del teclado. Se realiza una petición a una dirección URL donde se ejecuta Spring Boot seguido de la extensión que diseña la estructura del teclado, un ejemplo de la URL es http://localhost:8080/keyboard (En esta dirección la ruta es /keyboard pues es la que dirige al teclado virtual). Lo que se obtiene de la dirección es la siguiente



La forma en que se integra el código en el backend en la página web es la siguiente. Primero se debe definir el espacio donde será publicado el teclado, por ejemplo, un DIV:

|  |
| --- |
| <div *class*="input-field" *id*="keyboard-container"></div> |

La función utilizada para cargar y publicar el teclado se llama cargarContenido() donde implementa otra función llamada connect() para establecer una conexión con la base de datos temporal creada por Spring Boot para poder manejar websockets

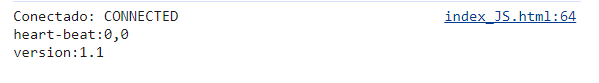
|  |
| --- |
| *function* cargarContenido() {  *// Realizar una solicitud HTTP GET a la URL deseada*            fetch('http://localhost:8080/keyboard')            then(*response* => response.text())             then(*data* => {            document.getElementById('keyboard-container').innerHTML = data;                  connect();              })              .catch(*error* => console.error('Error:', error));          } |

|  |
| --- |
| *var* stompClient = null;  *function* connect() {                  console.log(typeof SockJS);  *var* socket = new SockJS('http://localhost:8080/ws-chat'); *// Conecta al endpoint WebSocket*                  stompClient = Stomp.over(socket);                  stompClient.connect({}, *function* (*frame*) {                      console.log('Conectado: ' + frame);                  });  *// Suscribirse al canal después de la conexión exitosa*          stompClient.subscribe('/topic/message', *function* (*response*) {  *var* message = JSON.parse(response.body);  *// Manejar el mensaje recibido y mostrarlo en el chat-container*  *var* chatContainer = document.getElementById('chat-container');                chatContainer.innerHTML += '<p>' + message.message + '</p>';                  });              } |

Para establecer la conexión con el servidor para las pruebas que ofrece SpringBoot se recomienda utilizar las siguientes dependencias y poder implementar el procedimiento de web service creado para la comunicación en tiempo real utilizando la variable stompClient. Para manejar la conexión a websocket se crea un objeto StompJS bajo el nombre de socket para representar su conexión (valor que se le asigna a la variable de manejo).

|  |
| --- |
| // dependencias necesarias para utilizar SockJS y Stopm     <script *src*="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.6.1/sockjs.min.js" *integrity*="sha512-1QvjE7BtotQjkq8PxLeF6P46gEpBRXuskzIVgjFpekzFVF4yjRgrQvTG1MTOJ3yQgvTteKAcO7DSZI92+u/yZw==" *crossorigin*="anonymous" *referrerpolicy*="no-referrer"></script>      <script *src*="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.6.1/sockjs.js" *integrity*="sha512-aHuAnb+FZ/yM5zYvdiA8q0sxLdsBQ3PRwO8D/08qsenCLQ2gWsosB43Cn1X8esAz0IpaC7Ns5cAPyKIdJQ0hNw==" *crossorigin*="anonymous" *referrerpolicy*="no-referrer"></script>      <script *src*="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.min.js" *integrity*="sha512-iKDtgDyTHjAitUDdLljGhenhPwrbBfqTKWO1mkhSFH3A7blITC9MhYon6SjnMhp4o0rADGw9yAC6EW4t5a4K3g==" *crossorigin*="anonymous" *referrerpolicy*="no-referrer"></script>      <script *src*="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.js" *integrity*="sha512-tL4PIUsPy+Rks1go4kQG8M8/ItpRMvKnbBjQm4d2DQnFwgcBYRRN00QdyQnWSCwNMsoY/MfJY8nHp2CzlNdtZA==" *crossorigin*="anonymous" *referrerpolicy*="no-referrer"></script> |

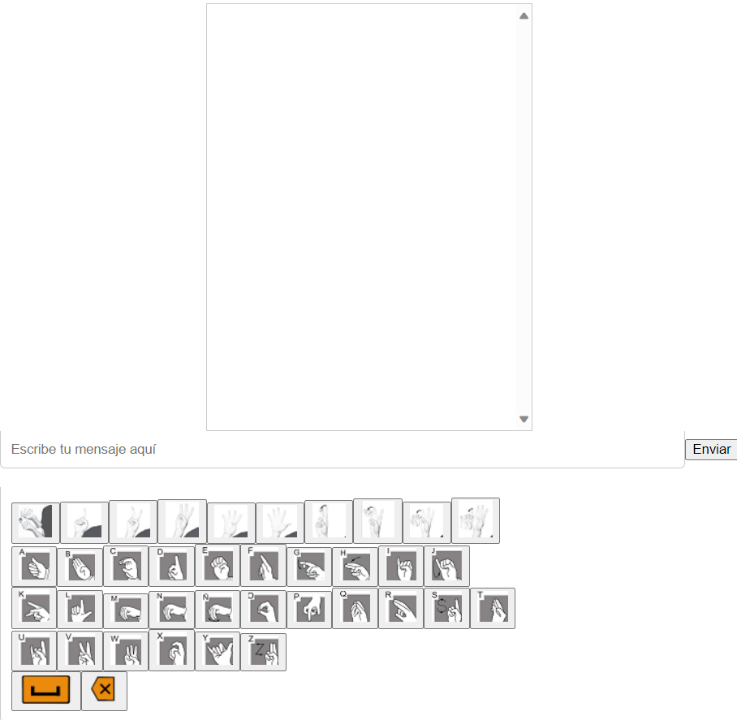
En la parte inferior del código de la función Connect() se hace referencia a una ruta llamada /topic/message la cual es utilizada para crear una suscripción al canal del servidor. El resultado es un mensaje de conexión en la página HTML como el siguiente.



En el teclado virtual cada botón se definió con una identidad y valor único dependiendo del valor que este representando, una vez mencionado esto cada botón tiene una función OnClick que ejecuta una función llamada LskKeyPressed que publica el valor del botón en un input que sea definido en la función. La función hace una distinción entre los botones de eliminar y crear espacios.

|  |
| --- |
| function LscKeyPressed(button) {          var dataKey = button.getAttribute('data-key');          var messageInput = document.getElementById('message-input');    // Verificar si el boton elegido es BORRAR se elimina el último carácter de la secuencia, en caso contrario se agrega en la sentencia              if (dataKey === "del") {                  var inputValue = messageInput.value;                  if (inputValue.length > 0) {                      // Eliminar el último carácter                      messageInput.value = inputValue.slice(0, -1);                  }              } else {                  // Agregar el valor de dataKey al input-field                  messageInput.value += dataKey;              }        } |

Esta es una imagen que muestra cómo se vería la página.



### Página web con Php

Para la Implementación del plugin en una página web utilizando PHP es diferente, pues php no es un lenguaje para interactuar con el usuario, sino que es utilizado para comunicarse con el servidor por lo cual las acciones de los botones deben realizarse utilizando otro lenguaje como JavaScript. Para ejemplificar el como se puede publicar el teclado virtual utilizando PHP se ha creado una pagina HTML de ejemplo donde primero se define un contenedor como DIV donde se va a albergar el teclado y dentro de este se abre PHP para hacer una solicitud a la dirección y la ruta del teclado virtual como lo puede ser http://localhost:8080/keyboard (En esta dirección la ruta es /keyboard es la que dirige al teclado virtual).

|  |
| --- |
| div class="input-field" id="keyboard-container">          <?php  //Obtener el contenido de http://localhost:8080/keyboard utilizando PHP     $contenido = file\_get\_contents('http://localhost:8080/keyboard');          // Enseñar el contenido dentro de la página          echo $contenido;          ?>   </div> |

Otra forma de solicitar el teclado virtual se puede realizar utilizando el elemento iframe de HTML convidando con la dirección del teclado virtual.

|  |
| --- |
| <iframe src="http://localhost:8080/keyboard" frameborder="0"></iframe> |

El resultado de ambas formas de solicitar el teclado da como resultado publicar el teclado en el espacio delimitado con el contenedor DIV.



## Configuraciones en aplicación Móvil

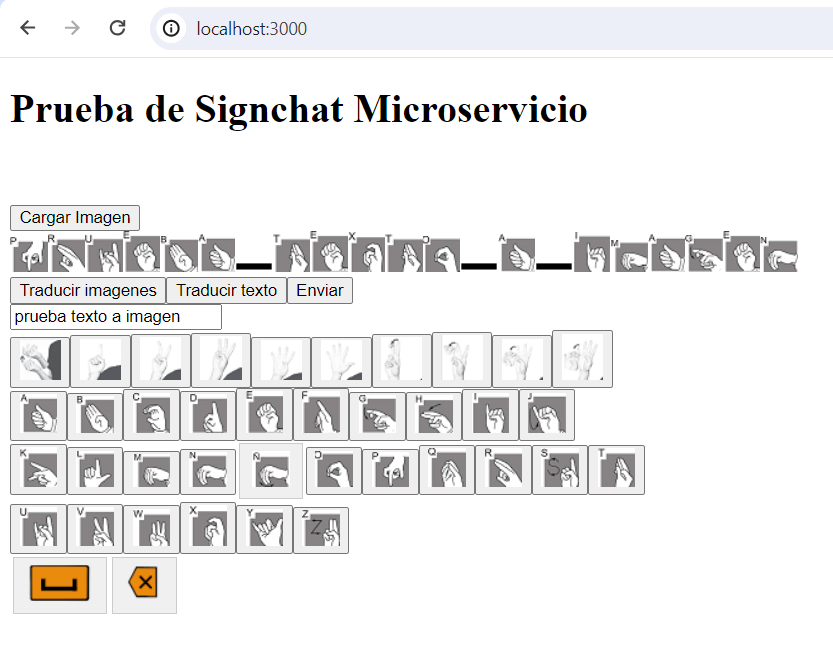
# Mensajes

## Traducción de mensajes

La traducción de mensajes es la parte fundamental de SignChat. Esta traducción es de español latino a LSC y viceversa. Se puede llamar sin la necesidad de utilizar el teclado, aunque lo recomendable es hacerlo, porque parte de la lógica viene de analizar las imágenes de la ruta */images/.*

### Traducción de texto a imágenes LSC

Para traducir de texto a imagen LSC, se utiliza la siguiente ruta */translate/txt-img*. A continuación, una demostración de la traducción de texto a imágenes utilizando una página de prueba desarrollada en REACT.



El código utilizado para realiza esta acción es el siguiente. Aquí en la función de ejemplo llamada “sendTextToBackend” concatena el valor del texto con la ruta definida para la traducción de imágenes a texto que es “*/translate/txt-img*“ y su publicación en la página.

|  |
| --- |
| *const* sendTextToBackend = *async* () => {      try {  *const* response = await fetch(API\_URLS.backend\_url + 'api/messages/translate/txt-img', {          method: 'POST',          headers: {            'Content-Type': 'application/json',          },          body: JSON.stringify({ text: inputValue }),        });          if (response.ok) {          console.log(response);  *const* translatedData = await response.json();          setInputImgValue(translatedData);          console.log('Mensaje enviado con éxito al backend.');        } else {          console.error('Error al enviar el mensaje al backend.');        }      } catch (error) {        console.error('Error al enviar el mensaje al backend:', error);      }    } |

### Traducción de imágenes LSC a texto

Para traducir de imagen LSC a texto, se utiliza la siguiente ruta */translate/img-txt*. A continuación, una demostración en una página de ejemplo desarrollada en REACT.

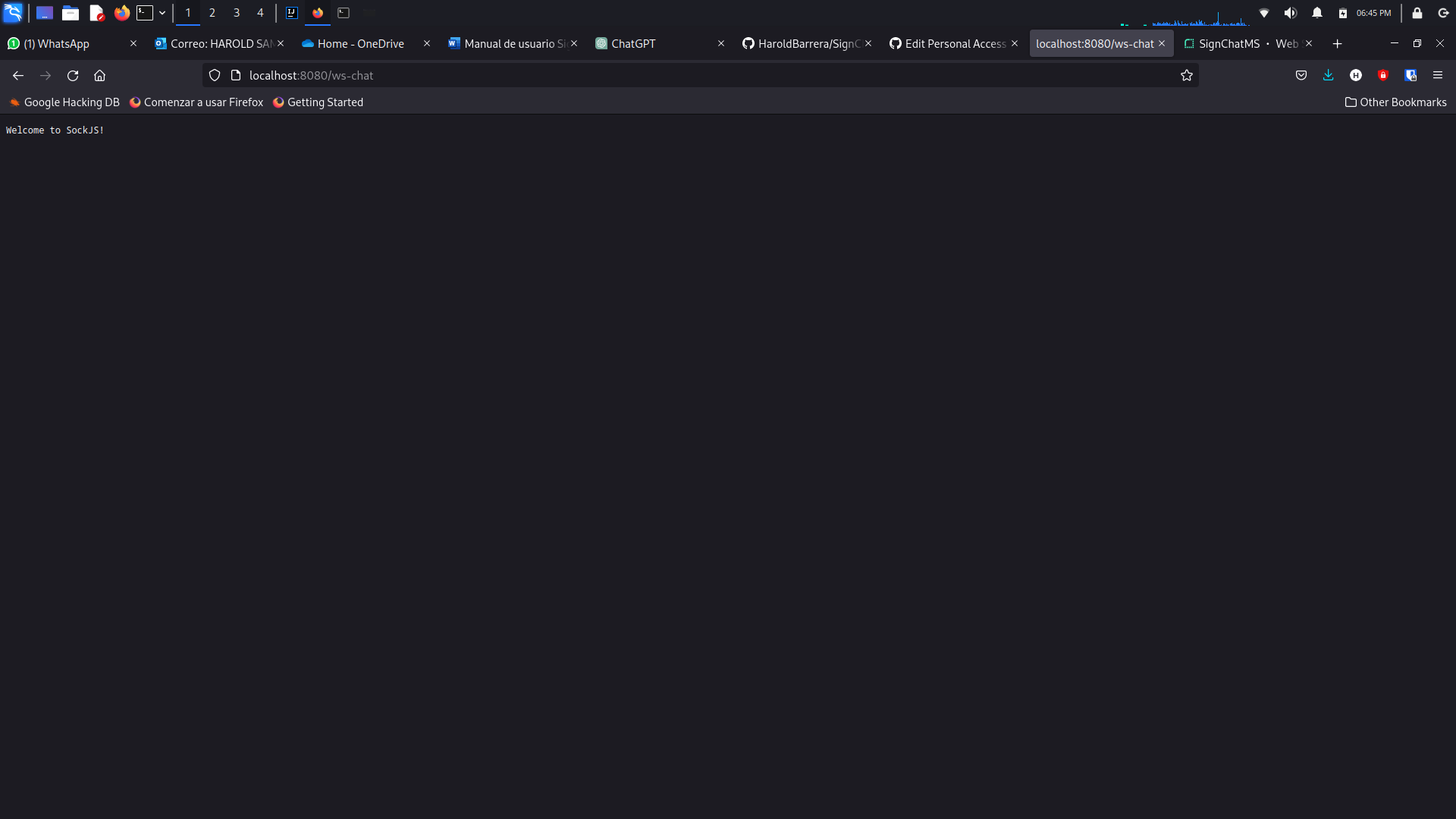


El código utilizado para realiza esta acción es el siguiente. Aquí en la función de ejemplo llamada “sendImagesToBackend” consigue el valor de las imágenes y las concatena con la ruta definida para la traducción de imágenes a texto que es “*/translate/img-txt* “.

|  |
| --- |
| *const* sendImagesToBackend = *async* () => {      console.log(API\_URLS.backend\_url + 'api/messages/translate/img-txt');      try {  *const* response = await fetch(API\_URLS.backend\_url + 'api/messages/translate/img-txt', {          method: 'POST',          headers: {            'Content-Type': 'application/json',          },          body: JSON.stringify({ images: inputImgValue }),        });          if (response.ok) {          console.log(response);  *const* translatedData = await response.text();          setInputValue(translatedData);          console.log('Imágenes enviadas con éxito al backend.');        } else {          console.error('Error al enviar imágenes al backend.');        }      } catch (error) {        console.error('Error al enviar imágenes al backend:', error);      }    } |

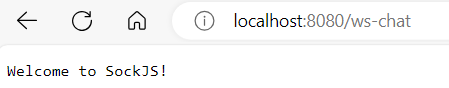
# Comunicación en tiempo real

Para la comunicación en tiempo real se implementó el uso de WebSockets SockJS. SockJS también usa para el envío de mensajes entre cliente servidor StompJS. Estas dos tecnologías de JavaScript permiten una comunicación en tiempo real. Puedes comprobar que el servidor WebSocket está funcionando si vistas la ruta */ws-chat/*.



## SockJS

Tecnología que será utilizada para facilitar la implementación de websockets junto con StompJS. El WebSocket es utilizado en nuestro proyecto para establecer una conexión con la base de datos para lo cual se implementan las dependencias de SockJS esta se destina para la implementación del Websocket, en caso de querer verificar el estado del websocket se dirige a la ruta /ws-chat la cual envía un mensaje de éxito de conexión de WebSocket

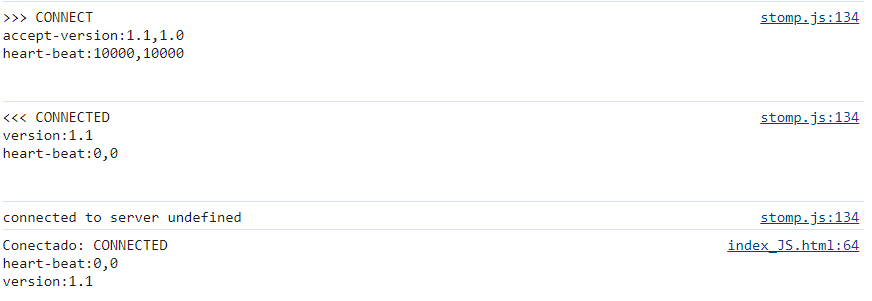


## StompJS

Tecnología que será utilizada para facilitar la implementación de websockets junto con SockJS. Las dependencias utilizadas en la página de prueba están destinadas para el buen funcionamiento de StompJS.

|  |
| --- |
| // dependencias necesarias para utilizar SockJS y Stopm     <script *src*="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.6.1/sockjs.min.js" *integrity*="sha512-1QvjE7BtotQjkq8PxLeF6P46gEpBRXuskzIVgjFpekzFVF4yjRgrQvTG1MTOJ3yQgvTteKAcO7DSZI92+u/yZw==" *crossorigin*="anonymous" *referrerpolicy*="no-referrer"></script>      <script *src*="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.6.1/sockjs.js" *integrity*="sha512-aHuAnb+FZ/yM5zYvdiA8q0sxLdsBQ3PRwO8D/08qsenCLQ2gWsosB43Cn1X8esAz0IpaC7Ns5cAPyKIdJQ0hNw==" *crossorigin*="anonymous" *referrerpolicy*="no-referrer"></script>      <script *src*="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.min.js" *integrity*="sha512-iKDtgDyTHjAitUDdLljGhenhPwrbBfqTKWO1mkhSFH3A7blITC9MhYon6SjnMhp4o0rADGw9yAC6EW4t5a4K3g==" *crossorigin*="anonymous" *referrerpolicy*="no-referrer"></script>      <script *src*="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.js" *integrity*="sha512-tL4PIUsPy+Rks1go4kQG8M8/ItpRMvKnbBjQm4d2DQnFwgcBYRRN00QdyQnWSCwNMsoY/MfJY8nHp2CzlNdtZA==" *crossorigin*="anonymous" *referrerpolicy*="no-referrer"></script> |

Estas dependencias son utilizadas para la mensajería en tiempo real, utilizar una versión reducida de stomp.js, utilizar versiones reducidas de Sockjs.min.js y ayudar a la conexión e implementación de los websockets siendo StompJS para él envió de mensajería al servidor. Para verificar la conectividad de StompJS se implementa en las páginas web de prueba las dependencias ya mencionadas para poder implementar la función de StompJS, luego de esto se publica una serie de mensajes que explica el estado de la conexión en la página web de prueba donde se está implementando.

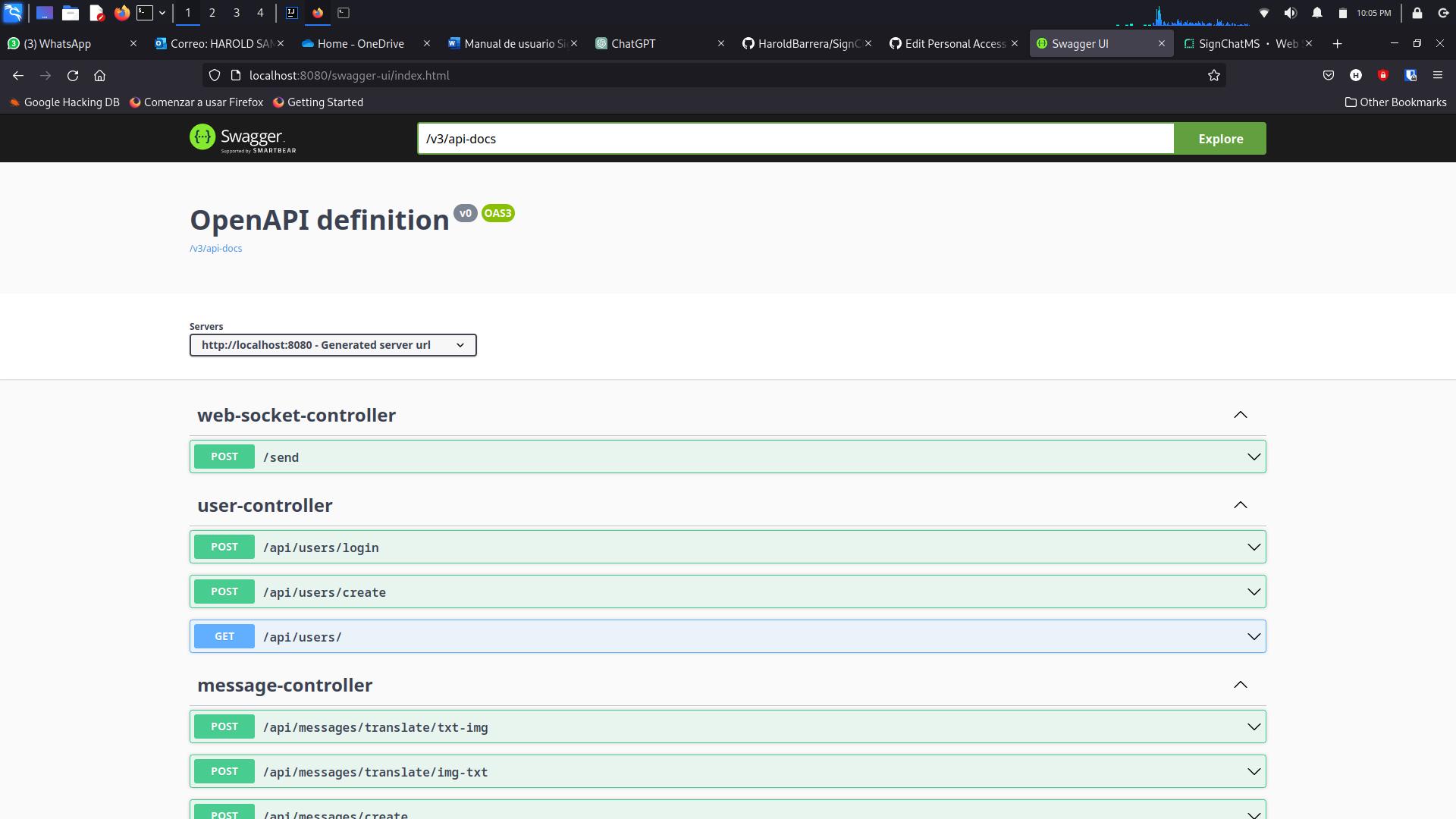


# Funcionalidades extra

Además de las funcionalidades mostradas hasta este punto, el plugin cuenta con más características, que, dependiendo de las circunstancias, pueden resultar ser útiles.

## Lista de rutas de la API

El plugin cuenta con Swagger UI, una forma de ver de manera resumida todas las rutas de la aplicación. Cuenta tambien con la forma de testear todas y cada una de estas funcionalidades, ademas de una breve descripción de como deberia usarse. Para ingresar a Swagger UI, toca ingresar a la ruta */swagger-ui/index.html*.



Las actividades se dividen en las clases que se crearon según su funcionalidad.

Web-Socket-Controller:

* /send: utilizado para el envío de mensaje mediante el websocket hacia la base de datos

User-Controller: funcionalidades creadas para realizar pruebas de funcionamiento y pueden ser modificadas o alteradas en base a las necesidades. Además, que las funciones se destinan a la creación de usuarios

* /api/users/login: envió de mensaje mediante el uso del websocket del api desarrollado
* /api/users/create: en una actividad de prueba para la creación de usuarios y almacenamiento en la base de datos de SpringBoot

Message-Controller:

* /api/messages/traslate/txt-img: Esta es la base principal del api pues esta función se destina a cumplir la función de transformar los textos ingresados por su visión en lenguaje de señas
* /api/messages/traslate/img-txt: Esta funcionen es el complemento de la función principal que se encarga de la traducción de los textos expresados en imágenes para convertirlos a texto
* /api/messages/create: función destinada a crear y guardar el texto desarrollado
* /api/messages/: esta función es de tipo GET con la finalidad de obtener los mensajes que se han almacenados

ContactList-Controller: funcionalidades creadas para realizar pruebas de funcionamiento y pueden ser modificadas o alteradas en base a las necesidades. Además, esta se encarga de listar la lista de contactos según el usuario.

* /api/contacts/create: se encarga de crear una lista de contactos
* /api/contacts: lista los contactos

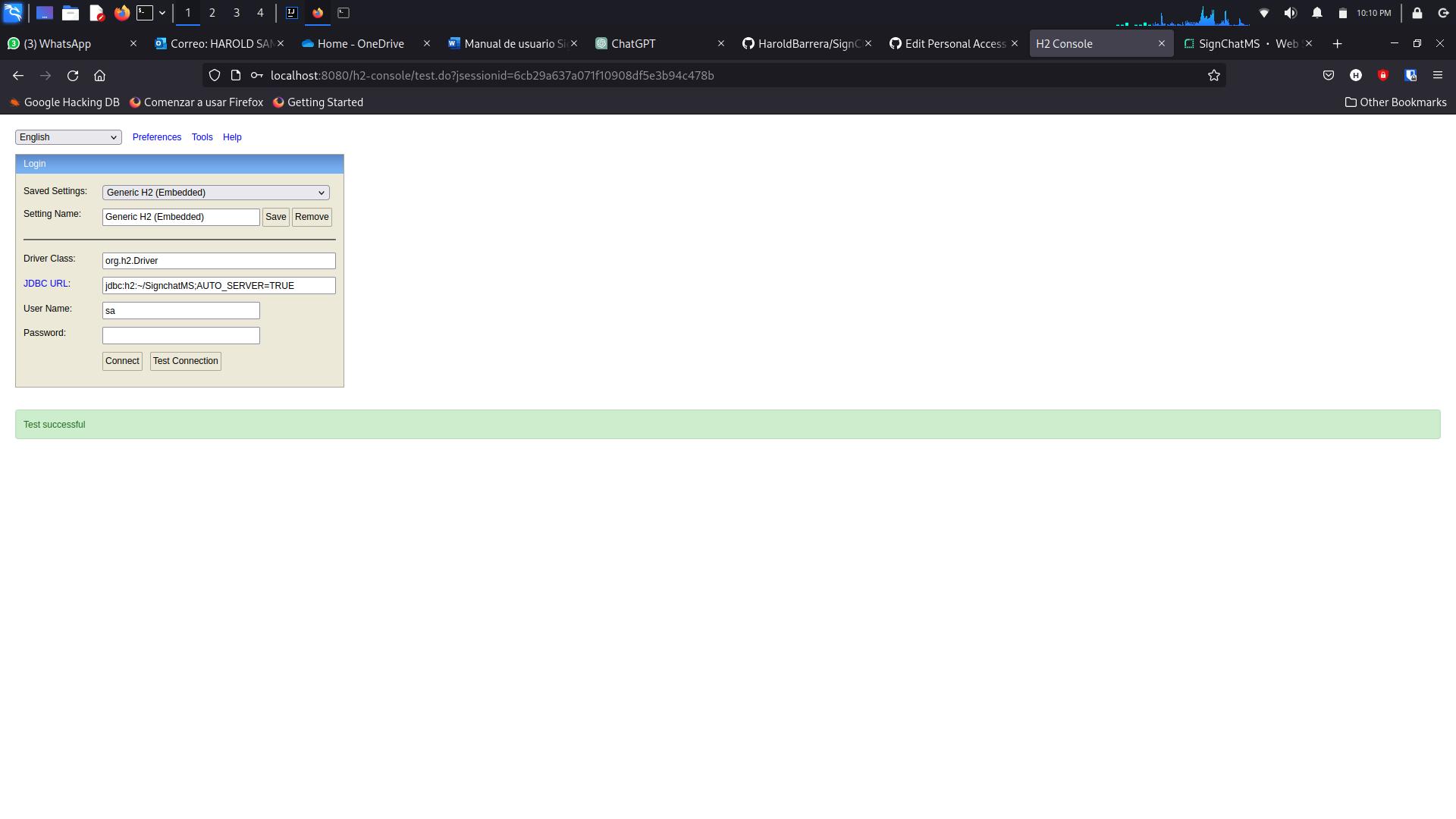
Keyboard-Controller:

* /keyboard: esta funcionalidad es utilizada para la solicitud e implementación del teclado virtual el cual publica imágenes individuales en los botones según la letra especificada esto mediante el uso de la funcionalidad /images/{imageName}.
* /keyboard/react: realiza la misma actividad de /keyboard pero esta funcionalidad destinada para ser utilizada en frameworks para desarrollo web avanzados
* /images/{imageName}: esta función es la encargada de buscar las imágenes individualmente que están almacenadas además de ser utilizad por la función /keyboard

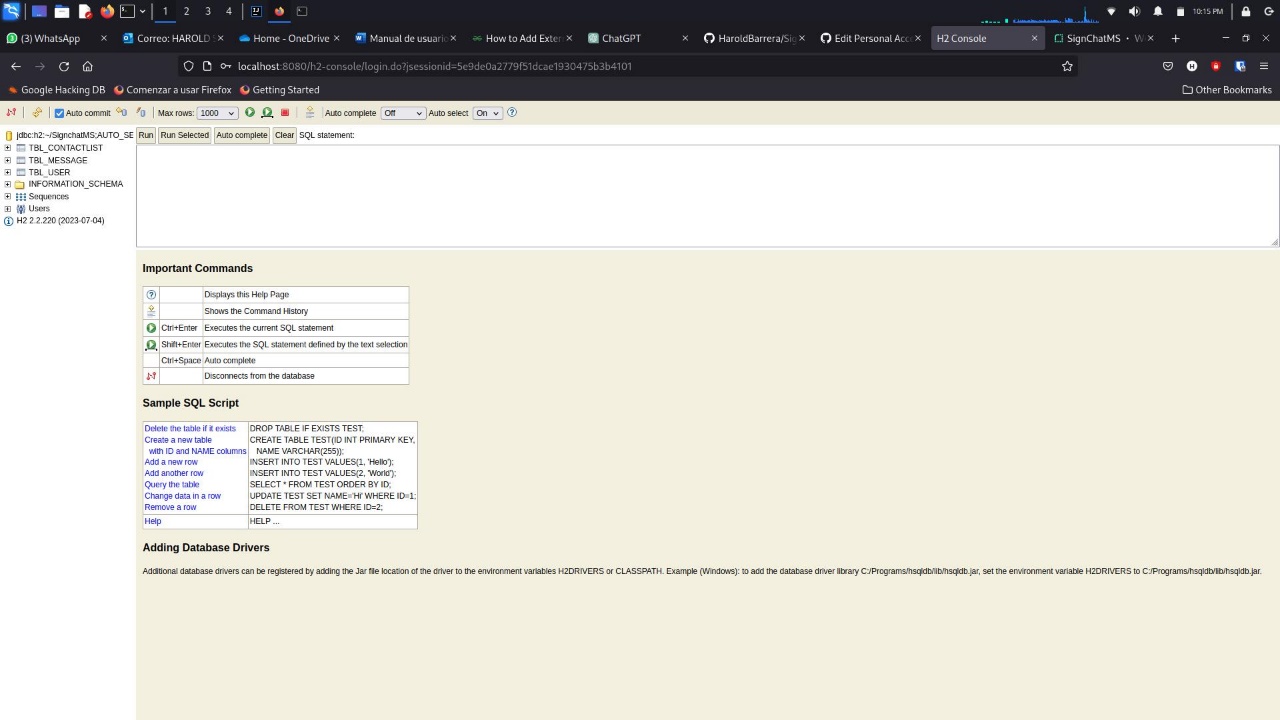
## Base de datos

El plugin cuenta con una base de datos embebida llamada H2. Esta base de datos utiliza SQL para la creación de sus tablas y demás propiedades que pueden ser vistas en su [documentacion](https://www.h2database.com/html/main.html).

Para visitar esta base de datos, debes dirigirte a la ruta */h2-console/*. Ahí te pedirá un usuario y contraseña. El usuario es *sa* y la contraseña es *password*. Prueba insertar estos valores y probar la conexión.



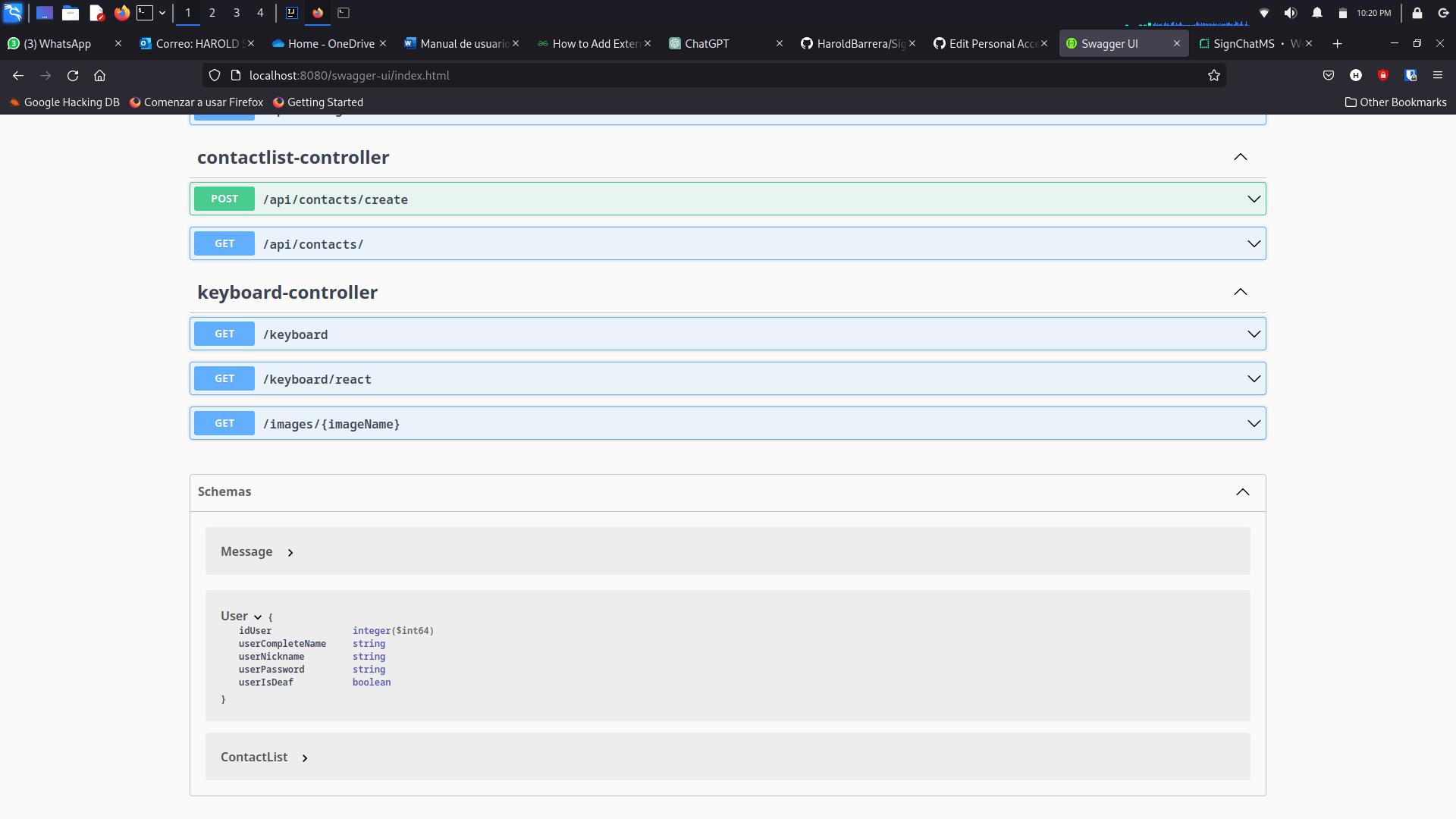
Si todo sale bien, puedes conectarte a la base de datos de H2 y empezar a utilizarla.



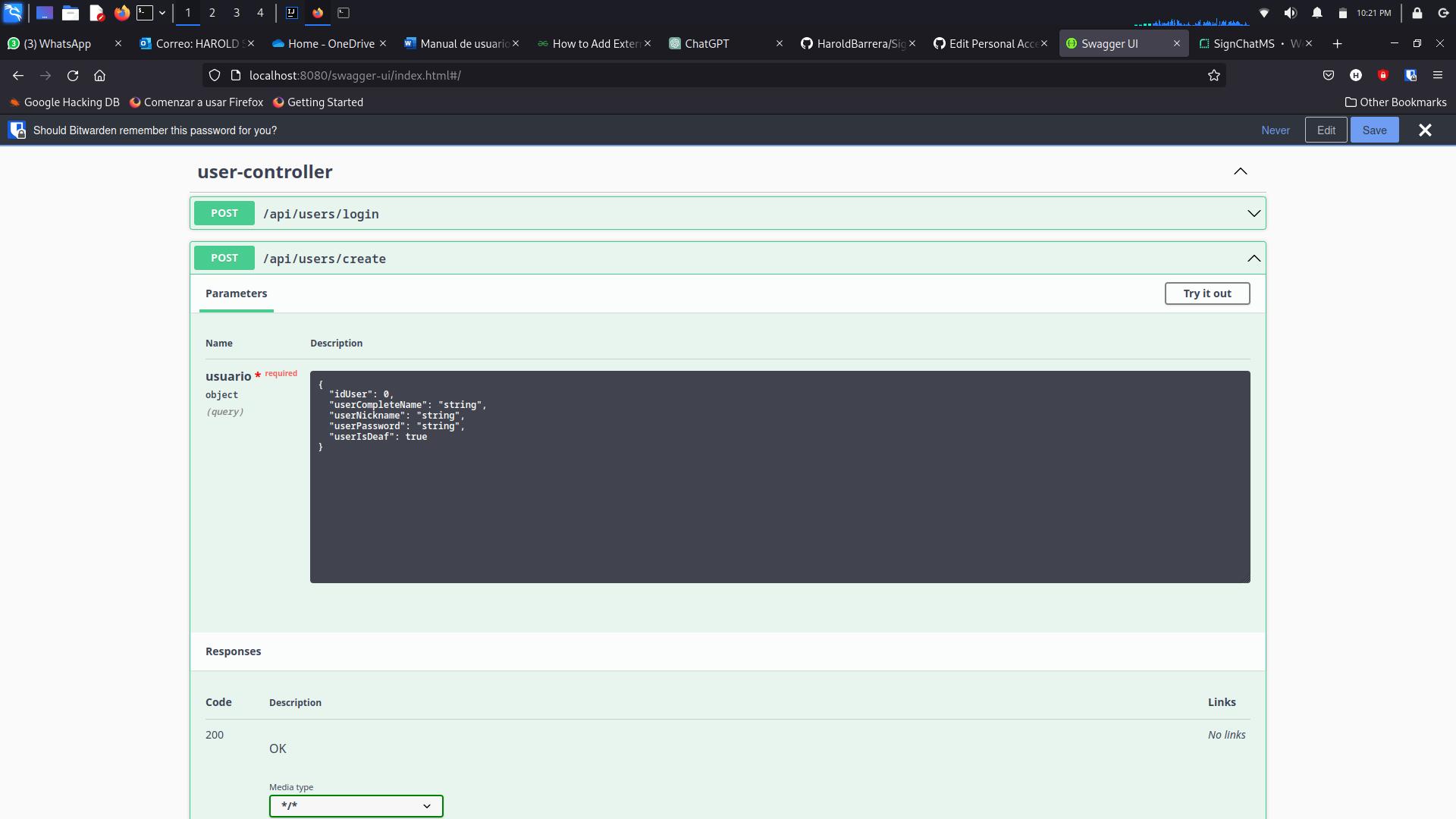
Recuerda que la base de datos H2 **no es recomendable para producto final**. Esto debido a que no es escalable. El objetivo de esta base de datos es para la realización de pruebas y su manejo durante su desarrollo.

### Usuarios

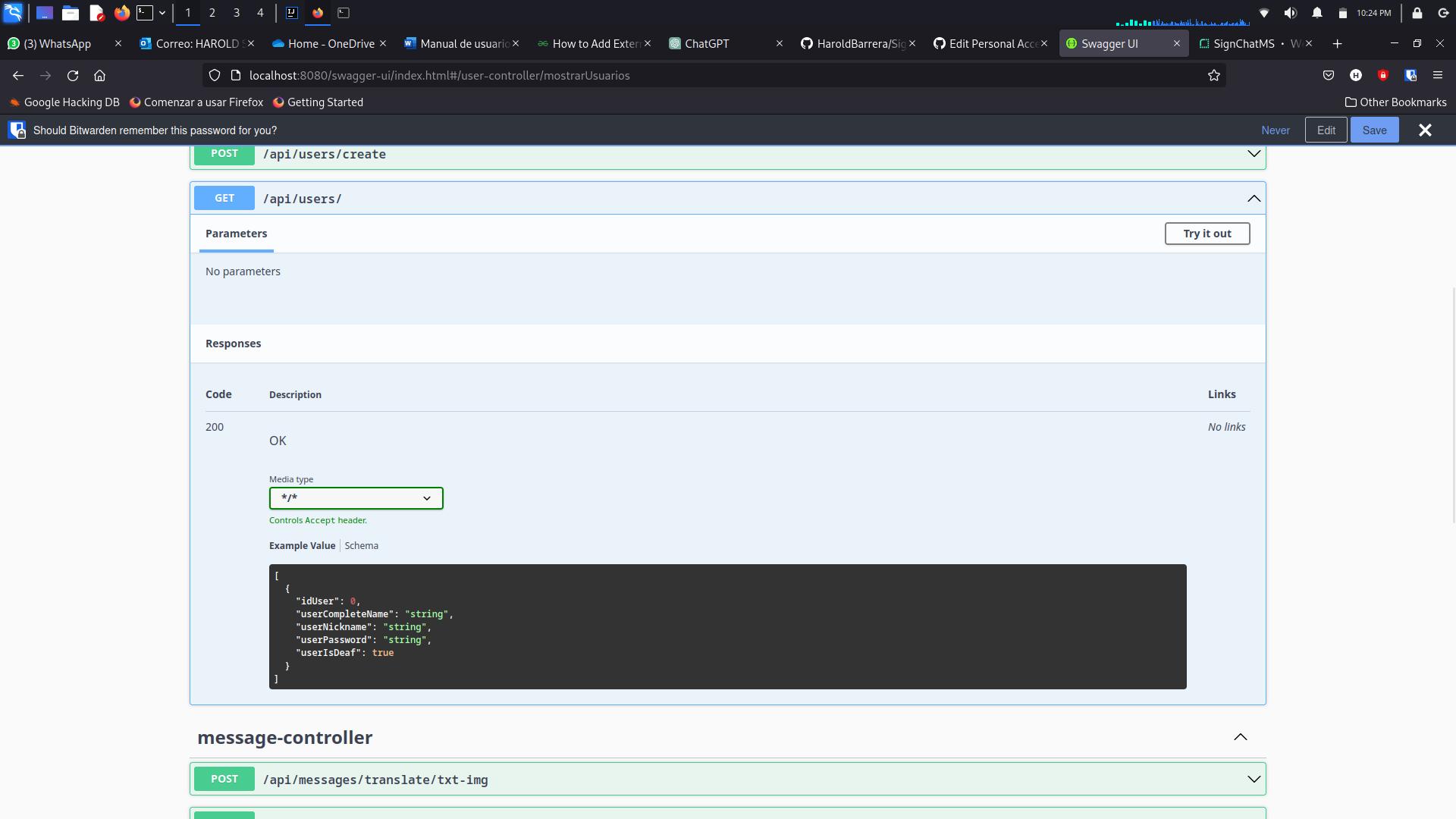
La base de datos cuenta con la capacidad de almacenar usuarios. Su estructura y métodos pueden ser encontrados en Swagger UI.



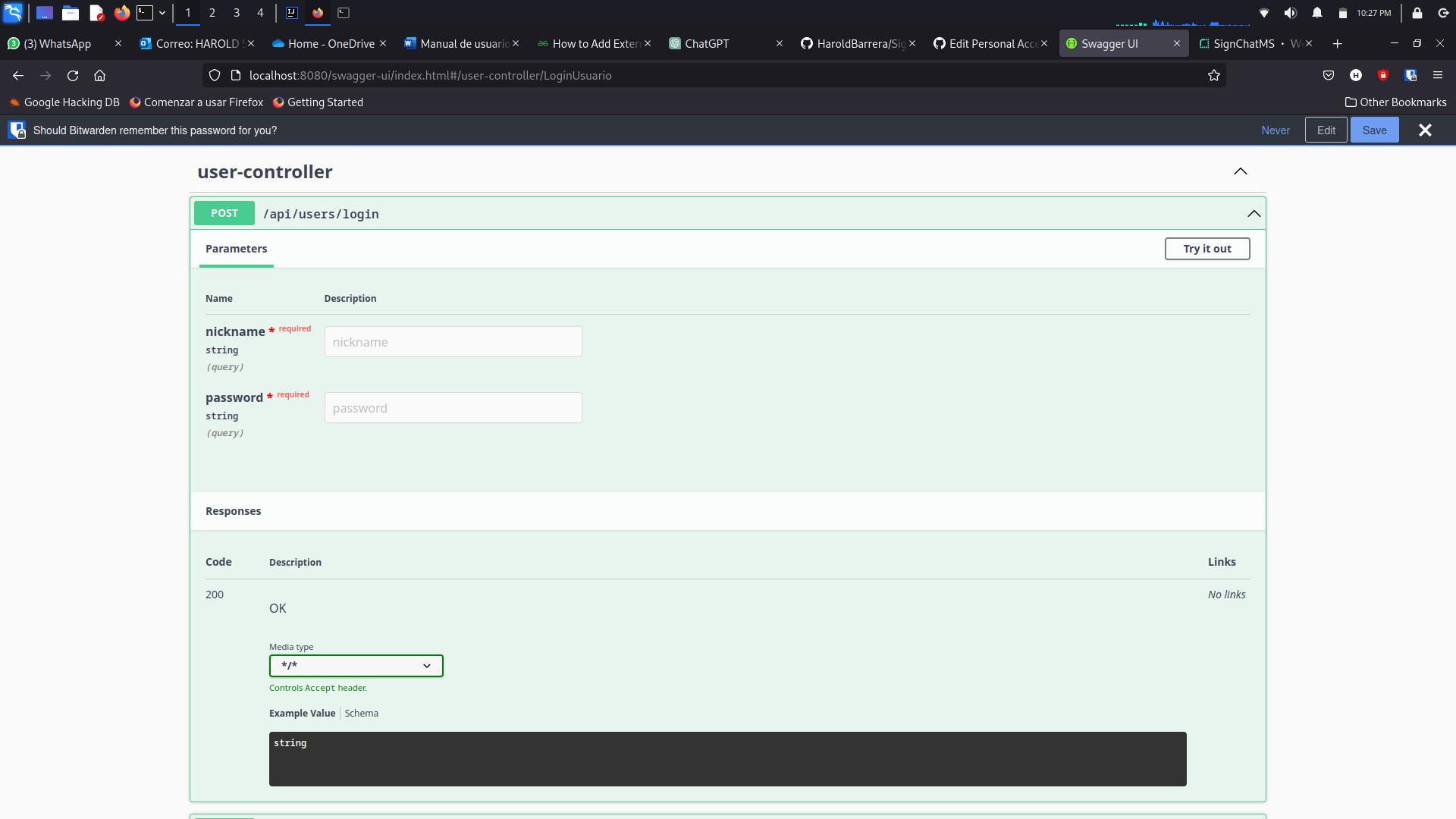
Los usuarios cuentan con la ruta */api/users/create* para crear usuarios y almacenarlos en la base de datos.



Puedes traer todos tus usuarios de la base de datos con la ruta */api/users*.

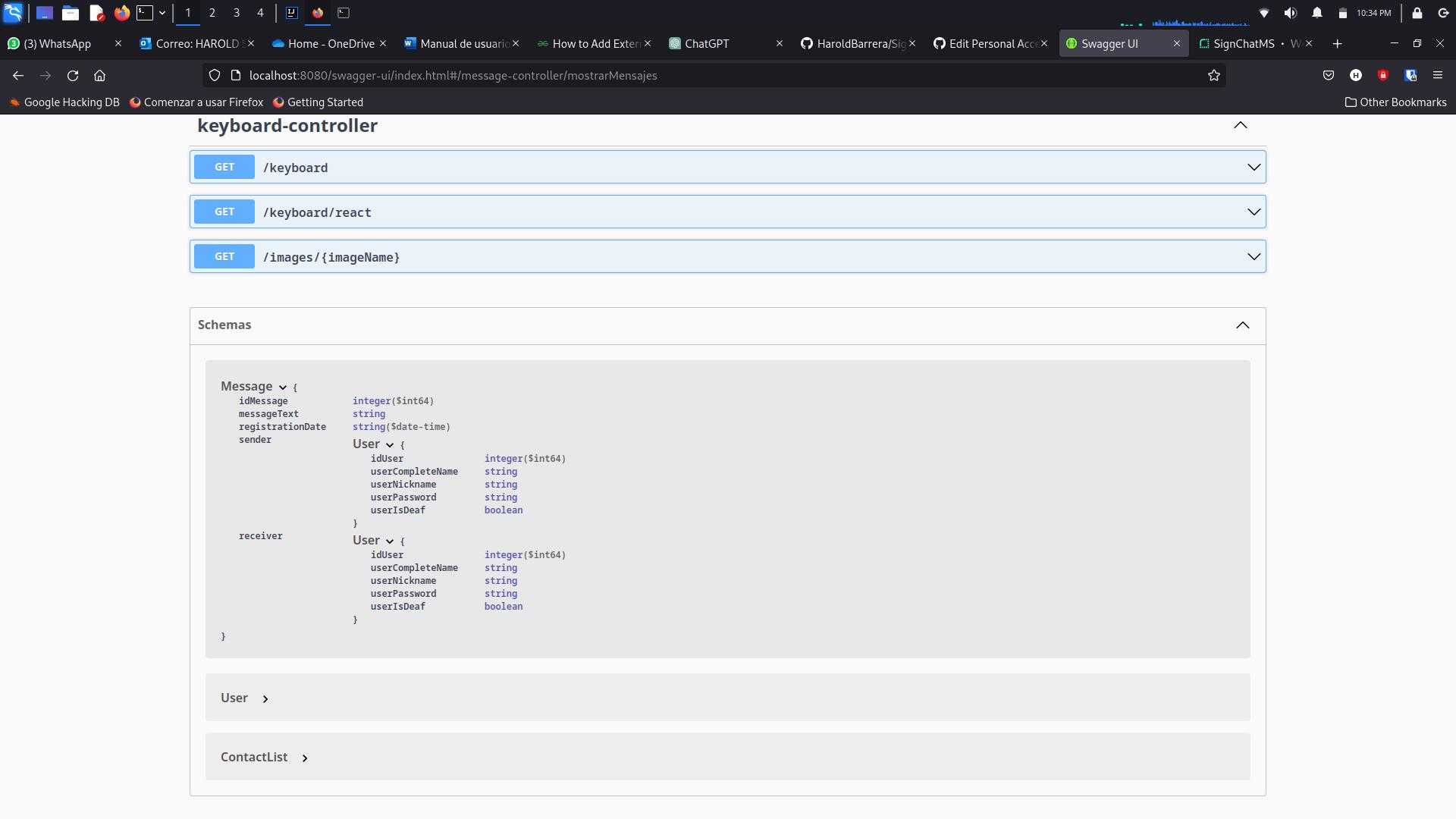


Por último, puedes hacer un inicio de sesión con tus usuarios con la ruta */api/users/login/*.

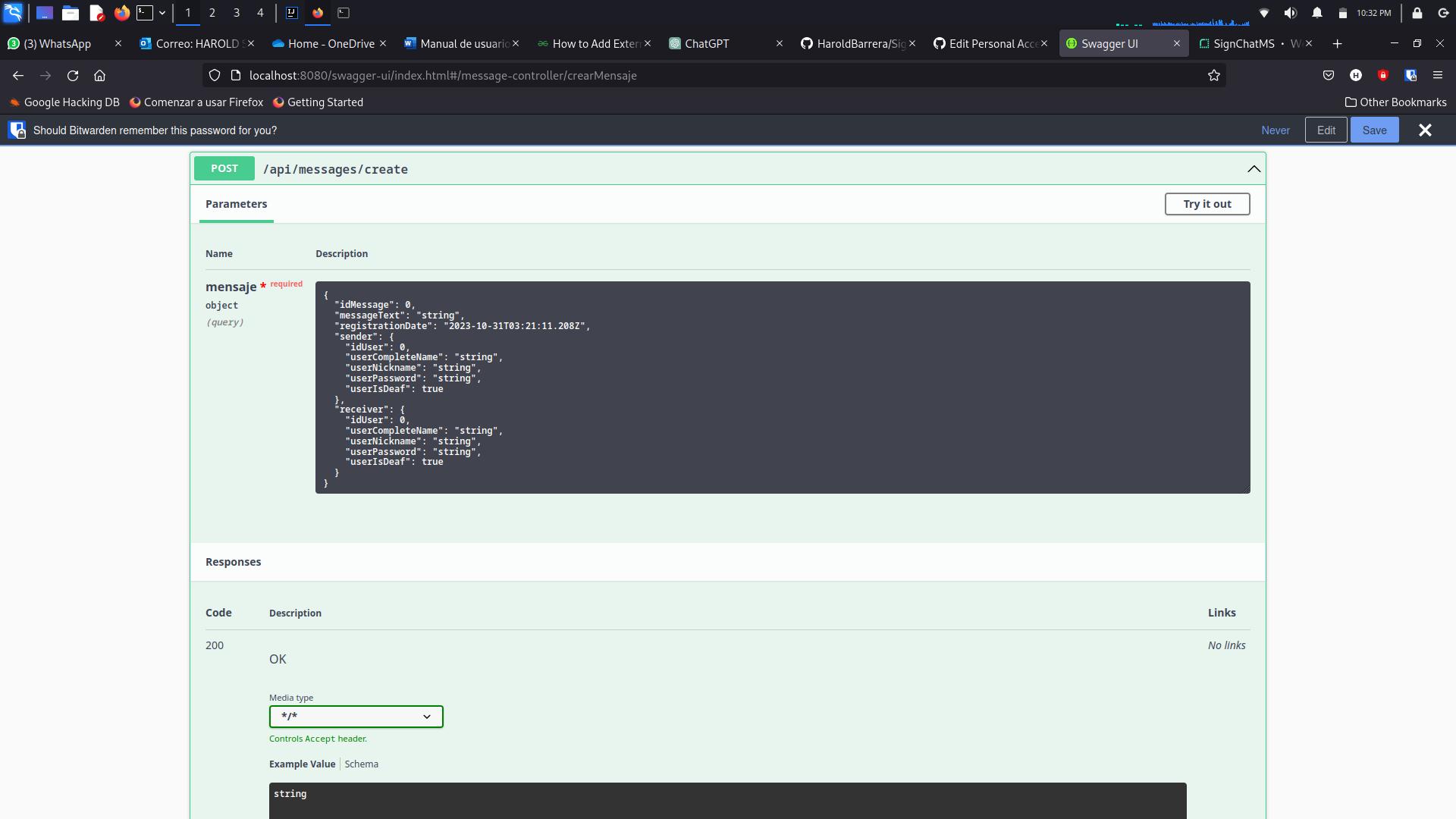


### Mensajes

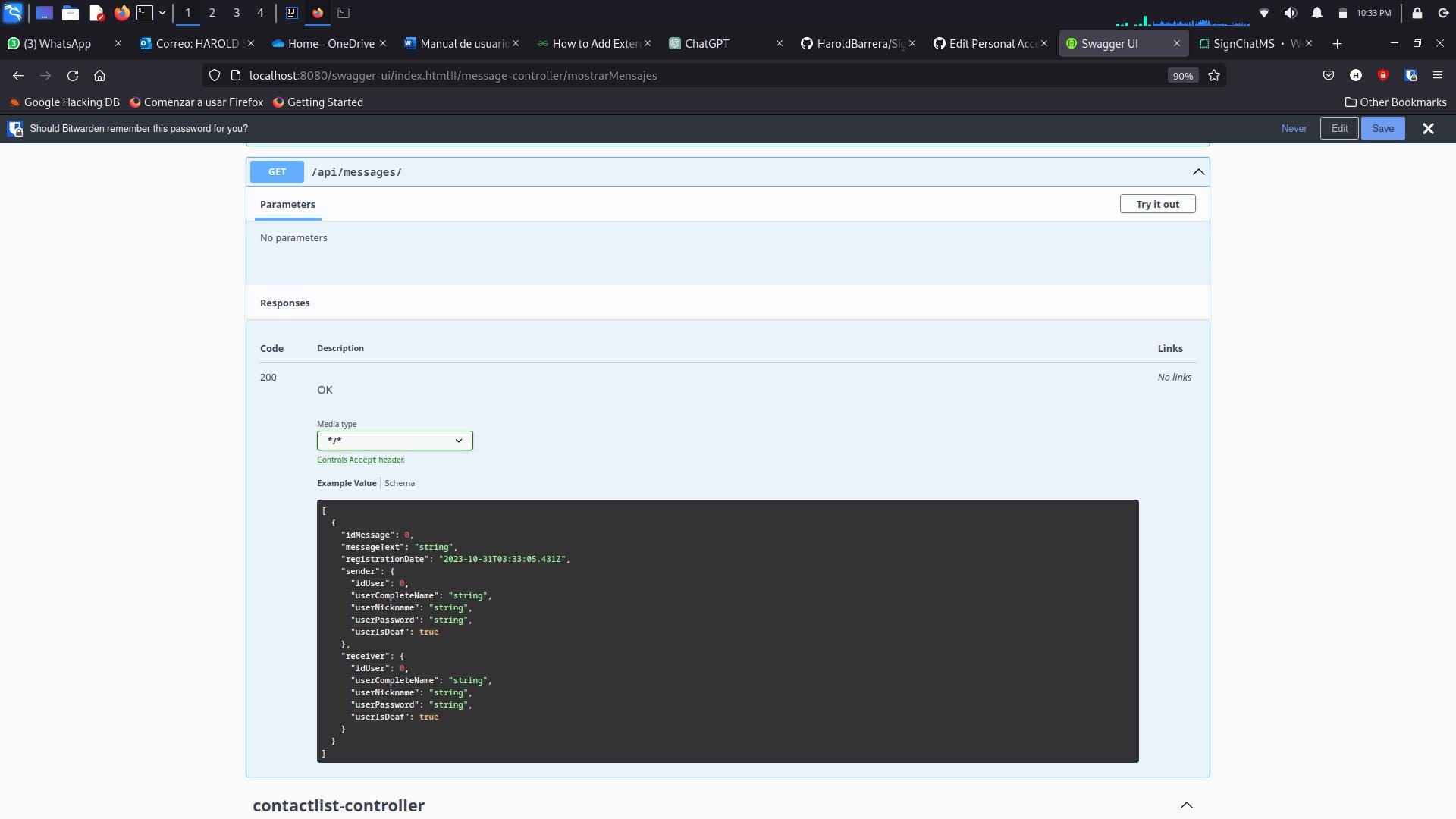
La base de datos también almacenara los mensajes enviados a la aplicación si así lo deseas. Además de la traducción de mensajes que ya se mostró como usar con anterioridad, también puedes crear y obtener los mensajes creados de tu base de datos.



Usa */api/messages/create* para crear un mensaje.

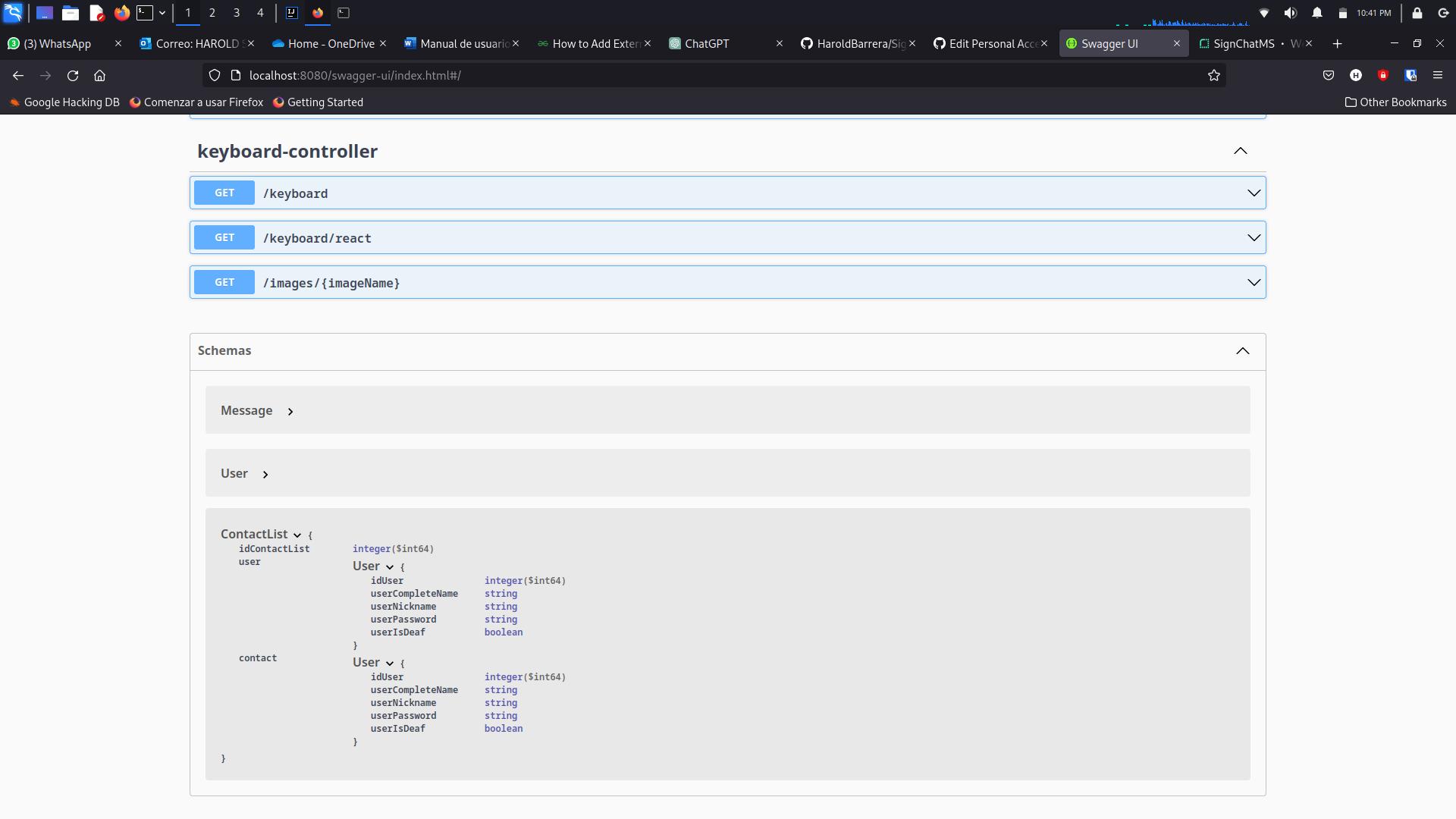


Usa */api/messages/* para obtener tus mensajes de la base de datos.

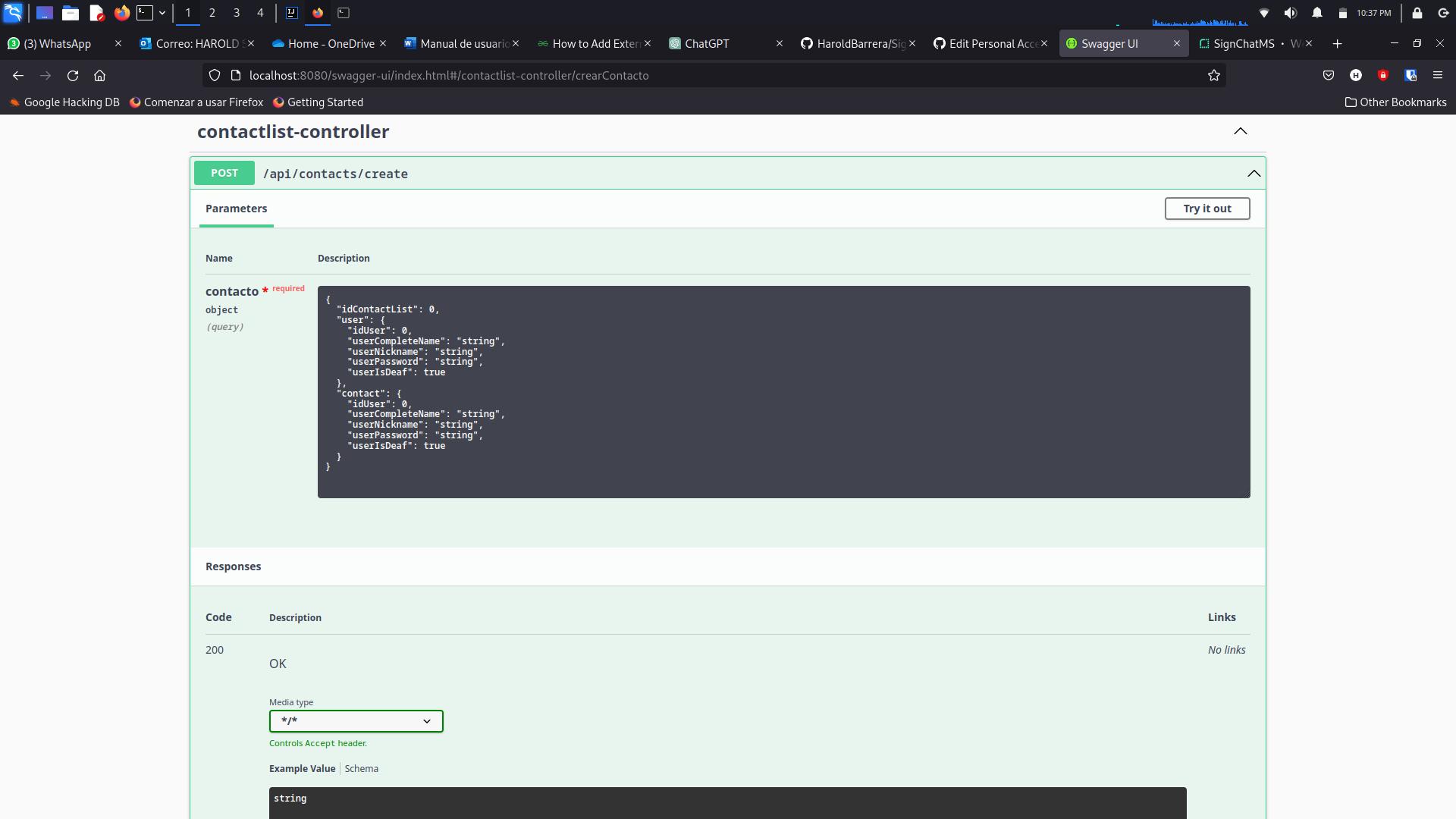


### Lista de contactos

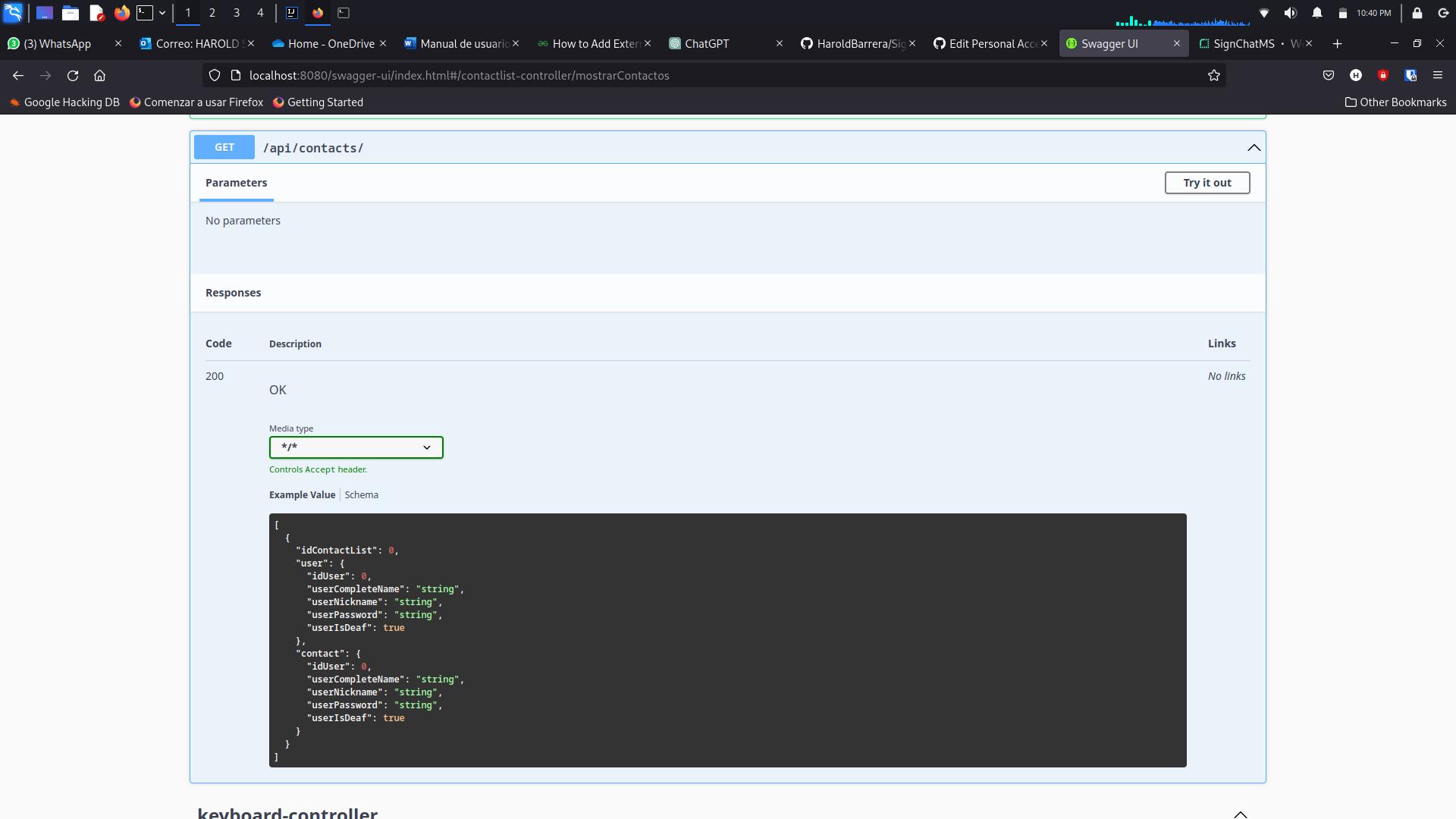
La base de datos también almacenara la lista de contactos de cada usuario. De igual manera, puedes crear un contacto como obtener los contactos de un usuario desde la base de datos.



Usa */api/contactlist/create* para crear un contacto.



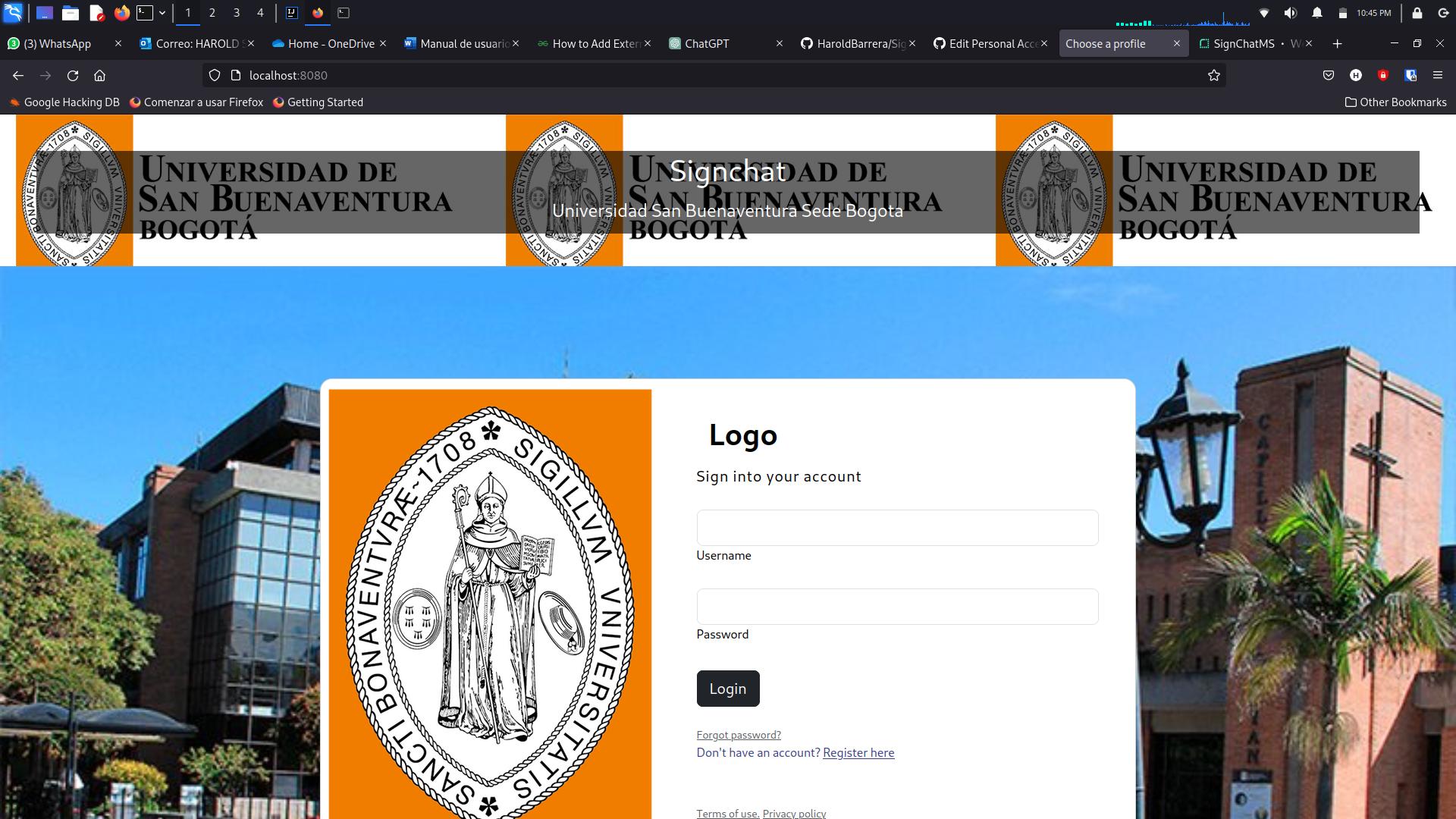
Usa */api/contactlist/* para obtener los contactos de un usuario.



## Plantillas HTML

El plugin cuenta por el momento con dos plantillas HTML que te pueden servir de guía cuando crees tu aplicación. Estas plantillas no tienen funcionalidades como tal, pero sirven para tener un diseño guía.

Esta es la plantilla para una página de inicio de sesión en la ruta /.



Aquí una plantilla de cómo se vería una página de mensajería en tiempo real en la ruta */signchat*.

