**Proyecto API-CHAT**

**DIGID - CIDESO**

Mayo 2022

**ÍNDICE:**

1. [Características generales del proyecto API-CHAT](#Vision_general)
2. [Definición de APIs](#Definición_APIs)
   1. [Solicitud de conversaciones posibles para un usuario logueado](#Conversacion_usuario)
   2. [Solicitud de los mensajes de una conversación dada](#Mensaje_conversacion)
   3. [Solicitud de creación de nuevo mensaje de tipo TEXTO](#Creacion_mje_texto)
   4. [Solicitud de creación de nuevo mensaje de tipo FILE](#Creacion_mje_file)
   5. [Solicitud de creación de nuevo mensaje de tipo POSITION](#Creacion_mje_position)
   6. [Solicitud de creación de nuev](#Creacion_position_usuario)a posición de un usuario
   7. [Solicitud de las posiciones de los contactos de un usuario](#posiciones_contactos_usuario)
3. [Información relevante para instalación](#Infor_relevante_instalacion)
4. ***Características generales del proyecto API-CHAT:***

* 1. Para el desarrollo se utilizó **PHP** + **MYSQL** + **Laravel** (framework PHP)
  2. Implementación de ***HTTPS*** *-* Hypertext Transfer Protocol Secure. Protocolo que permite establecer una conexión segura entre dispositivos de usuarios y los sitios web.
  3. *Seguridad de acceso a las APIs a través de rutas protegidas … (Analizar qué se utilizará)*
  4. ***Buenas prácticas*** para garantizar que el desarrollo de la aplicación minimice los riesgos de seguridad a través del desarrollo de código seguro.
  5. Implementación del estándar ***CORS*** (Cross-origin resource sharing)

1. **Definición de APIs**
2. *Solicitud de conversaciones posibles para un usuario logueado:*
3. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
4. Petición:
   * + **https://{{HOST}}/api/{user\_id}/conversations/**
     + Método: GET

(Debe enviarse el *token* dentro de los headers de la petición.

*Authorization*: Bearer ***token)***

1. ENTRADA: parámetros por URL
   * user\_id
2. SALIDA:
   * Para una petición correctamente ejecutada, se obtendrá el código de estado “**200 OK”** y la estructura de datos que devolverá el JSON es la siguiente:

{

"user\_origin": 1,

"conversations": [

{

"conversation\_id": 5,

"conversation\_name": "Grupo CIDESO",

"conversation\_members": [

{

"user\_id": 3,

"last\_read\_at": "2022-03-22 16:11:35"

},

{

"user\_id": 2,

"last\_read\_at": "2022-03-22 16:11:35"

},

{

"user\_id": 4,

"last\_read\_at": "2022-03-22 16:11:35"

},

{

"user\_id": 1,

"last\_read\_at": "2022-05-14 01:09:16"

}

],

"ammount\_no\_read": 0

},

{

"conversation\_id": 1,

"conversation\_name": **null**,

"conversation\_members": [

{

"user\_id": 2,

"name": "Javier"

}

],

"ammount\_no\_read": 0

}

]

}

* + - Donde:
* "*user\_origin*": es el ID del usuario logueado
* "*conversations*":
* "conversation\_id": es el ID de la conversación.
* "conversation\_name": Si la conversación es grupal, se enviará el nombre del grupo, si es individual se enviará ***null***,
* "conversation\_members" Array que contiene los ids de los usuarios que forman parte de la conversación. Si ésta es grupal, los ids corresponden a la totalidad de los usuarios del grupo. Si es individual, el array contiene el id del usuario destino.

Por cada usuario, también se envía la última fecha de visualización de los mensajes de dicha conversación por parte de ese usuario.

* "ammount\_no\_read": Cantidad de mensajes NO leídos por el usuario logueado, en dicha conversación.
  + - Las Conversaciones se devuelven ordenadas por las que tienen mensajes más recientes a las que tienen mensajes menos recientes.
  + Para una petición donde se envía como parámetro un ID de usuario que no existe, se obtendrá el código de estado “**500 Internal Server Error”** y se devolverá en un JSON, el detalle del error:

{

"status": 500,

"message": "No existe el usuario."

}

1. *Solicitud de los mensajes de una conversación dada:*
2. El *endpoint* devolverá un JSON con la información de los mensajes de una determinada conversación, con paginación de a 10.

El ***ID*** de la conversación que se debe pasar por URL, fue obtenido previamente con el endpoint de Solicitud de conversaciones para el usuario logueado.

1. Petición:

* **https://{{HOST}}/api/{user\_id}/conversations/{conversation\_id}**
* Método: GET

(Debe enviarse el *token* dentro de los headers de la petición.

*Authorization*: Bearer ***token)***

1. ENTRADA: parámetros por URL
   * user\_id
   * ***ID*** de la conversación.
2. SALIDA:

* Para una petición correctamente ejecutada, se obtendrá el código de estado “**200 OK”** y la estructura de datos que devolverá el JSON es la siguiente:

{

"user\_origin": 1,

"messages": {

"current\_page": 1,

"data": [

{

"conversation\_id": 1,

"sender\_id": 2,

"message\_type": "App\\Models\\PositionMessage",

"message\_id": 6,

"created\_at": "2022-05-17T13:55:25.000000Z",

"display\_type": "PositionMessage",

"message": {

"id": 6,

"lat": "-37.588967",

"lon": "-38.588967",

"alt": "-15.198195"

}

},

{

"conversation\_id": 1,

"sender\_id": 2,

"message\_type": "App\\Models\\TextMessage",

"message\_id": 25,

"created\_at": "2022-05-17T13:55:16.000000Z",

"display\_type": "TextMessage",

"message": {

"id": 25,

"text": "Hola Valei! Soy Javi 17/5/22 mje 2"

}

},

{

"conversation\_id": 1,

"sender\_id": 2,

"message\_type": "App\\Models\\TextMessage",

"message\_id": 24,

"created\_at": "2022-05-17T13:55:10.000000Z",

"display\_type": "TextMessage",

"message": {

"id": 24,

"text": "Hola Valei! Soy Javi 17/5/22 mje 1"

}

}

],

"first\_page\_url": "http://127.0.0.1:8000/api/1/conversations/1?page=1",

"from": 1,

"last\_page": 9,

"last\_page\_url": "http://127.0.0.1:8000/api/1/conversations/1?page=9",

"next\_page\_url": "http://127.0.0.1:8000/api/1/conversations/1?page=2",

"path": "http://127.0.0.1:8000/api/1/conversations/1",

"per\_page": 3,

"prev\_page\_url": **null**,

"to": 3,

"total": 27

}

}

**NOTA**:

* Para el ejemplo anterior se utilizó una paginación de 3 ítems. Realmente se utiliza una paginación de 10 ítems.
  + Para una petición donde se envía como parámetro un ID de usuario o de conversación que no existe, se obtendrá el código de estado “**500 Internal Server Error”** y se devolverá en un JSON, el detalle del error:

{

"status": 500,

"message": "No existe el usuario."

}

o

{

"status": 500,

"message": "No existe la conversación."

}

1. *Solicitud de creación de nuevo mensaje de tipo TEXTO:*
2. El *endpoint* creará un nuevo mensaje de TEXTO para una conversación dada.
3. Devolverá un JSON con la información del mensaje creado y el estado de respuesta a la petición.
4. Petición:
   * [**https://{{HOST}}/api/**](https://emersis.casya.com.ar/api/v1/messages)**textMessage**
   * Método: POST

(Debe enviarse el *token* dentro de los headers de la petición.

*Authorization*: Bearer ***token)***

1. ENTRADA:

* **Form-data:** 
  + - 'user\_id' => ['required','integer', 'exists:users,id'],
    - 'conversation \_id' => ['required','integer'],
    - 'message' => ['required','string', 'max:255'],

1. SALIDA:
   * Si el usuario logueado NO pertenece a la conversación a la que se desea enviar un mensaje, se obtendrá el código de estado **“500 Internal Server Error”** y se devolverá en un JSON, el detalle del error:

{

"status": 500,

"message": "El usuario no forma parte de la conversación a la que desea enviar el mensaje."

}

* + Para una petición correctamente ejecutada, se obtendrá el código de estado “**200 OK”** y la estructura de datos que devolverá el JSON es la siguiente:

{

"status": 200,

"message": "Creación del mensaje de TEXTO realizada con éxito",

"conversation\_id": 8,

"sender\_id": 6,

"message\_created": "Hola Gabriel! Soy Brian"

}

1. *Solicitud de creación de nuevo mensaje de tipo FILE:*
2. El *endpoint* creará un o varios mensaje/s de tipo FILE para una conversación dada.
3. Devolverá un JSON con la información del o los mensaje/s creado/s y el estado de respuesta a la petición. Para cada archivo que se envíe, se creará un mensaje del tipo file.
4. Para que los archivos enviados puedan ser accedidos, se debe crear un symbolic link desde **"public/storage"** a **"storage/app/public"**. Con Artisan se puede hacer desde la terminal corriendo el siguiente comando: *php artisan storage:link*
5. Petición:
   * **https://{{HOST}}/api/fileMessage**
   * Método: POST

(Debe enviarse el *token* dentro de los headers de la petición.

*Authorization*: Bearer ***token)***

1. ENTRADA:

* **Form-data:** 
  + - 'user\_id' => ['required','integer', 'exists:users,id'],
    - 'conversation \_id' => ['required','integer'],
    - 'file' => ['required', 'array'],
    - 'file.\*' => ['file','required', 'mimes:doc,pdf,docx,txt,zip,jpeg,png,bmp,xls,xlsx,mov,qt,mp4,mp3,m4a' ,'max:10240'],
    - 'description' => ['sometimes', 'array'],
    - 'description.\*' => ['nullable', 'string'],

**NOTA**:

* Los nombres de los archivos que se guardan en el servidor tienen el siguiente formato: ***nombreOriginal\_timestamp***.extensión

1. SALIDA:
   * Si el usuario logueado NO pertenece a la conversación a la que se desea enviar un mensaje, se obtendrá el código de estado **“500 Internal Server Error”** y se devolverá en un JSON, el detalle del error:

{

"status": 500,

"message": "El usuario no forma parte de la conversación a la que desea enviar el mensaje."

}

* + Los archivos se envían en un array.

Si algún archivo NO pasa las validaciones, se devuelve qué archivo no pasó / por qué y se crean los mensajes correspondientes a los archivos OK.

* + Si los parámetros "user\_id" y "conversation\_id", NO pasan las validaciones, se obtendrá el código de estado “**422 – Unprocessable Entity”** y lo que devolverá el JSON es lo siguiente, según corresponda:

{

"errors": {

"user\_id": [

"The user id must be an integer."

],

"user\_id": [

"The selected user id is invalid."

],

"conversation\_id": [

"The selected conversation id is invalid."

]

}

}

* + Si los parámetros "user\_id" y "conversation\_id", pasan las validaciones, se procede a validar los archivos enviados.
  + Si dentro de los enviados todos los archivos son válidos o hay algunos válidos y otros no, se obtendrá el código de estado “**200 OK”** y la estructura de datos que devolverá el JSON es la siguiente:

{

"status": 200,

"message": "Creación de mensaje de tipo FILE realizada con éxito",

"conversation\_id": 6,

"sender\_id": 2,

"message\_created": [

{

"file": "files/Norma\_DC\_001\_14\_0\_1654527765.pdf",

"original\_file": "Norma\_DC\_001\_14.pdf",

"description": "Norma\_DC\_001\_14.pdf"

}

],

"messages\_with\_error": [

{

"index": 1,

"original\_file": "Captura de pantalla de 2021-09-15 12-29-40.png",

"text\_error": "Los archivos sólo pueden ser doc,pdf,docx,txt,zip,jpeg,png,bmp,xls,xlsx,mov,qt,mp4,mp3,m4a"

}

]

}

Si dentro de los enviados todos los archivos son NO válidos, se obtendrá el código de estado “**200 OK”** y la estructura de datos que devolverá el JSON es la siguiente:

{

"message": "No se pudo crear el mensaje.",

"messages\_with\_error": [

{

"index": 1,

"original\_file": "Captura de pantalla de 2021-09-15 12-29-40.png",

"text\_error": "Los archivos sólo pueden ser doc,pdf,docx,txt,zip,jpeg,png,bmp,xls,xlsx,mov,qt,mp4,mp3,m4a"

}

]

}

1. *Solicitud de creación de nuevo mensaje de tipo POSITION:*
2. El *endpoint* creará un nuevo mensaje de *POSITION* para una conversación dada.
3. Devolverá un JSON con la información del mensaje creado y el estado de respuesta a la petición.
4. Petición:
   * **https://{{HOST}}/api/positionMessage**
   * Método: POST

(Debe enviarse el *token* dentro de los headers de la petición.

*Authorization*: Bearer ***token)***

1. ENTRADA:

* **Form-data:** 
  + - 'user\_id' => ['required','integer', 'exists:users,id'],
    - 'conversation \_id' => ['required','integer'],
    - 'lat' => ['required','numeric'],
    - 'lon' => ['required','numeric'],
    - 'alt' => ['required','numeric'],

1. SALIDA:
   * Si el usuario logueado NO pertenece a la conversación a la que se desea enviar un mensaje, se obtendrá el código de estado **“500 Internal Server Error”** y se devolverá en un JSON, el detalle del error:

{

"status": 500,

"message": "El usuario no forma parte de la conversación a la que desea enviar el mensaje."

}

* + Para una petición correctamente ejecutada, se obtendrá el código de estado “**200 OK”** y la estructura de datos que devolverá el JSON es la siguiente:

{

"status": 200,

"message": "Creación del mensaje de POSICIÓN realizada con éxito",

"conversation\_id": 1,

"sender\_id": 1,

"message\_created": {

"lat": "-37.588967",

"lon": "-38.588967",

"alt": "15.198195"

}

}

1. *Solicitud de creación de nueva posición de un usuario:*

1. El *endpoint* creará una nueva *POSITION* para el usuario logueado.
2. Devolverá un JSON con la información de la posición creada y el estado de respuesta a la petición.
3. Petición:
   * **https://{{HOST}}/api/position/user\_position**
   * Método: POST

(Debe enviarse el *token* dentro de los headers de la petición.

*Authorization*: Bearer ***token)***

1. ENTRADA:

* **Form-data:** 
  + - 'user\_id' => ['required','integer', 'exists:users,id'],
    - 'lat' => ['required','numeric'],
    - 'lon' => ['required','numeric'],
    - 'alt' => ['required','numeric']

1. SALIDA:
   * Para una petición correctamente ejecutada, se obtendrá el código de estado “**200 OK”** y la estructura de datos que devolverá el JSON es la siguiente:

{

"status": 200,

"message": "Creación de la posición del usuario, realizada con éxito",

"user\_id": 6,

"lat": "-55.12345",

"lon": "-58.09876",

"alt": "13"

}

1. *Solicitud de las posiciones de los contactos de un usuario:*
2. El *endpoint* devolverá un JSON con la información de las posiciones de los contactos del usuario logueado.
3. **Deberá definirse la cantidad de posiciones a devolver por cada contacto, actualmente se devuelven todas.**
4. Las posiciones se devuelven en forma decreciente en base a su fecha de creación.
5. Petición:

* **https://{{HOST}}/api/{user\_id}/position/user\_contacts\_positions**
* Método: GET

(Debe enviarse el *token* dentro de los headers de la petición.

*Authorization*: Bearer ***token)***

1. ENTRADA: parámetros por URL
   * user\_id

1. SALIDA:

* Para una petición correctamente ejecutada, se obtendrá el código de estado “**200 OK”** y la estructura de datos que devolverá el JSON es la siguiente:

{

"user": 5,

"user\_positions": [

{

"user\_id": 5,

"lat": "-55.123450",

"lon": "-58.098760",

"alt": "13.000000",

"created\_at": "2022-06-13T13:49:42.000000Z"

}

],

"user\_contacts\_positions": [

[

{

"user\_id": 3,

"lat": "45.666000",

"lon": "11.222000",

"alt": "1.000000",

"created\_at": "2022-03-28T15:33:18.000000Z"

}

],

[

{

"user\_id": 6,

"lat": "-56.123450",

"lon": "-59.098760",

"alt": "13.000000",

"created\_at": "2022-06-13T14:18:57.000000Z"

},

{

"user\_id": 6,

"lat": "-55.123450",

"lon": "-58.098760",

"alt": "13.000000",

"created\_at": "2022-06-13T14:17:21.000000Z"

}

]

]

}

1. ***Información relevante para instalación:***
2. En la terminal correr el siguiente comando que instalará los paquetes utilizados:
   1. *composer install*
3. Base de Datos:
4. Crear la BD con el nombre ***api-chat***
5. Poblar la BD con el archivo ***api-chat\_2022-06-13.sql*** ubicado en la carpeta Documentacion.

**NOTA**: El usuario y password usado de la BD se deberán especificar en el archivo de configuración (.env) que se detalla en el punto siguiente.

1. Variables de entorno:
2. Copiar el archivo **.env.example**
3. Cambiar el nombre a **.env**
4. En el .env modificar los valores de las siguientes variables:
   * + APP\_KEY=***base64:56/Otjz1sQoVascmscV5dqnpFSuKWCfSZ8GIVcR9xek=*** (poner este valor)
     + APP\_URL= *URL del back*
     + DB\_USERNAME= *usuario de la BD*
     + DB\_PASSWORD=*password de la BD*
     + BROADCAST\_DRIVER=***pusher*** (poner este valor)
     + PUSHER\_APP\_ID=*1****2345*** (poner este valor)
     + PUSHER\_APP\_KEY=***ASDASD2121*** (poner este valor)
     + PUSHER\_APP\_SECRET=***ASDASD123123*** (poner este valor)
     + PUSHER\_APP\_CLUSTER=***mt1*** (poner este valor)