**Proyecto API-CHAT**

**DIGID - CIDESO**

Noviembre 2022

**ÍNDICE:**

1. [Características generales del proyecto API-CHAT](#Vision_general)
2. APIs del CHAT
   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. [APIs de Autogestión de usuarios](#APIs_Autogestion_usuarios)
   1. [Solicitud de creación de usuario](#Creación_usuario)
   2. [Solicitud de información del usuario logueado](#Info_usuario_logueado)
   3. [Solicitud de actualización del usuario logueado](#actualizacion_usuario_logueado)
4. [APIs de Administración de usuarios](#APIs_Admin_usuarios)
   1. [Solicitud de información de un determinado usuario](#informacion_determinado_usuario)
   2. [Solicitud de actualización de un determinado usuario](#actualizacion_determinado_usuario)
   3. [Solicitud de listado de usuarios filtrado por location](#listado_usuarios_por_location)
5. [APIs de Administración de LOCATIONS](#APIs_Admin_locations)
   1. [Solicitud de creación de location](#Creación_location)
   2. [Solicitud de información de todas las locations](#informacion_todas_locations)
   3. [Solicitud de información de una determinada location](#informacion_determinada_location)
   4. [Solicitud de actualización de una determinada location](#actualizacion_determinada_location)
6. [APIs de Administración de GROUPS](#APIs_Admin_groups)
   1. [Solicitud de creación de group](#Creación_group)
   2. [Solicitud de información de todos los groups](#informacion_todos_groups)
   3. [Solicitud de información de un determinado group](#informacion_determinado_group)
   4. [Solicitud de actualización de un determinado group](#actualizacion_determinado_group)
7. [Información relevante para instalación](#Infor_relevante_instalacion)
8. Información funcional
9. ***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 utilizando **JWT.**
  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. **APIs** **del CHAT**
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/conversations/**
     + Método: GET

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

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

1. ENTRADA: *ninguna*
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.

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/conversations/{conversation\_id}**
* Método: GET

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

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

1. ENTRADA: parámetros por URL
   * ***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 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 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:** 
  + - '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."

}

* + Si "conversation\_id" o 'message', 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": {

"conversation\_id": [

"The selected conversation id is invalid."

],

"conversation\_id": [

"The conversation id must be an integer."

],

"message": [

"The message field is required."

],

}

}

* + 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 uno 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:** 
  + - '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 "conversation\_id", NO pasa las validaciones, se obtendrá el código de estado “**422 – Unprocessable Entity”** y lo que devolverá el JSON es lo siguiente:

{

"errors": {

"conversation\_id": [

"The selected conversation id is invalid."

]

}

}

* + Si "conversation\_id", pasa las validaciones, se procede a validar los archivos enviados.
  + Si dentro de los archivos enviados todos 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 archivos enviados NINGUNO es válido, 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:** 
  + - '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."

}

* + Si "conversation\_id" , 'lat', ‘lon’ o ‘alt’, 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": {

"conversation\_id": [

"The selected conversation id is invalid."

],

"conversation\_id": [

"The conversation id must be an integer."

],

"lat": [

"The lat field is required."

],

"lon": [

"The lon field is required."

],

"alt": [

"The alt field is required."

],

"lat": [

"The alt must be a number."

],

"lon": [

"The alt must be a number."

],

"alt": [

"The alt must be a number."

]

}

}

* + 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:** 
  + - '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"

}

* + Si 'lat', ‘lon’ o ‘alt’, 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": {

"lat": [

"The lat field is required."

],

"lon": [

"The lon field is required."

],

"alt": [

"The alt field is required."

],

"lat": [

"The alt must be a number."

],

"lon": [

"The alt must be a number."

],

"alt": [

"The alt must be a number."

]

}

}

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/position/user\_contacts\_positions**
* Método: GET

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

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

1. ENTRADA: ninguna

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. **APIs de Autogestión de usuarios**
2. *Solicitud de creación de usuario:*
3. El *endpoint* creará un nuevo usuario en la base de datos, tomando la información contenida en el token. Por defecto los usuarios creados **NO son administrador**, luego un administrador podrá actualizar el perfil.
4. Debe enviarse el token dentro de los headers de la petición.

Authorization: *Bearer token*

1. El token DEBE contener el *user\_name* del usuario que desea crearse.
2. Devolverá un JSON con la información del usuario creado y el estado de respuesta a la petición.
3. Petición:
   * + **https://{{HOST}}/api/user**
     + Método: POST
4. ENTRADA:
   * **Form-data:**

'name' => ['required','string', 'max:255'],

'surname' => ['required','string', 'max:255'],

'grade' => ['nullable','string'],

'dni' => ['required','integer'],

'location\_id' => ['required','integer', 'exists:locations,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": {

"user\_name": "usurio2",

"name": "NombreUsuario2",

"surname": "ApellidoUsuario2",

"grade": **null**,

"dni": "9090876",

"location\_id": "1",

"updated\_at": "2022-11-02T14:29:08.000000Z",

"created\_at": "2022-11-02T14:29:08.000000Z",

"id": 12

}

}

* Si el **user\_name** enviado por JWT corresponde a un usuario existente, devuelve un error 422, con el siguiente mensaje:

{

"error": "Ya existe el usuario con el user\_name: vblanco"

}

* Si el JWT no contiene el valor de user\_name, devuelve un error 422, con el siguiente mensaje:

{

"error": "El token NO contiene el user\_name."

}

1. *Solicitud de información del usuario logueado:*
2. El *endpoint* devolverá un JSON con la información del usuario logueado.
3. *Esta API también se llama después del login contra el portal, si EXISTE el usuario -> devuelve el objeto usuario, SI NO existe -> devuelve 404*
4. Debe enviarse el token dentro de los headers de la petición.

Authorization: *Bearer token*

1. El token DEBE traer el user\_name cuando la API se llama inmediatamente después del login (todavía el front NO sabe el user\_id)
2. El token DEBE traer el user\_id cuando la API se llama para que un usuario se autoadministre/visualice su cuenta.
3. Petición:
   * + **https://{{HOST}}/api/user**
     + Método: GET
4. ENTRADA: *ninguna*
5. 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": {

"id": 9,

"name": "Usuario",

"surname": "aaa",

"grade": **null**,

"dni": "222222221",

"location\_id": 2,

"user\_name": "usuario@cideso.com.ar",

"admin": 0,

"created\_at": "2022-10-21T16:12:23.000000Z",

"updated\_at": "2022-10-21T16:12:23.000000Z"

}

}

1. *Solicitud de actualización del usuario logueado:*
   1. El *endpoint* devolverá un JSON con la información del usuario actualizado.
   2. Debe enviarse el token dentro de los headers de la petición.

Authorization: *Bearer token*

* 1. El token DEBE traer el user\_id , la API se llama para que un usuario se autoadministre/actualice su cuenta.
  2. Petición:
     + **https://{{HOST}}/api/user**
     + Método: PUT
  3. ENTRADA:
* **x-www-form-urlencoded (Postman):**

'name' => ['sometimes','string', 'max:255'],

'surname' => ['sometimes','string', 'max:255'],

'grade' => ['sometimes','nullable','string'],

'dni' => ['sometimes','integer'],

'location\_id' => ['sometimes','integer', 'exists:locations,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": {

"user\_name": "usurio2",

"name": "NombreUsuario2",

"surname": "ApellidoUsuario2",

"grade": **null**,

"dni": "9090876",

"location\_id": "1",

"updated\_at": "2022-11-02T14:29:08.000000Z",

"created\_at": "2022-11-02T14:29:08.000000Z",

"id": 12

}

}

* Si el **user\_id** enviado por JWT no corresponde a un usuario existente, devuelve un error 404, con el siguiente mensaje:

{

"error": "No existe el usuario."

}

* Si el JWT no contiene el valor de user\_id, devuelve un error 422, con el siguiente mensaje:

{

"error": "El token NO contiene el user\_id."

}

1. **APIs de Administración de usuarios**
2. *Solicitud de información de un determinado usuario:*
3. El *endpoint* devolverá un JSON con la información del usuario solicitado.
4. Debe enviarse el token dentro de los headers de la petición.

Authorization: *Bearer token*

1. El token DEBE traer el user\_id del ADMIN que se validará si es un usuario Administrador.
2. En la URL se envía el ID del usuario del cual se está pidiendo información.
3. Petición:
   * + **https://{{HOST}}/api/admin/user/{user\_id}**
     + Método: GET
4. ENTRADA: parámetros por URL
   * ***ID*** del usuario.
5. 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": {

"id": 1,

"name": "Usuario",

"surname": "Apellido",

"grade": "Docente civil",

"dni": "111333",

"location\_id": 2,

"user\_name": "usuario2",

"admin": 0,

"created\_at": **null**,

"updated\_at": "2022-11-02T12:37:39.000000Z"

}

}

* Si el **user\_id** enviado por URL no corresponde a un usuario existente, devuelve un error 404, con el siguiente mensaje:

{

"error": "No existe el usuario."

}

* Si el usuario logueado NO es administrador, devuelve un error 422, con el siguiente mensaje:

{

"error": "El Usuario debe se ADMINISTRADOR."

}

1. *Solicitud de actualización de un determinado usuario:*
2. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
3. Petición:
   * + **https://{{HOST}}/api/conversations/**
     + Método: GET

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

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

1. ENTRADA:

* **Por parámetros** por URL: ***ID*** del usuario.
* **x-www-form-urlencoded (Postman):**

'name' => ['sometimes','string', 'max:255'],

'surname' => ['sometimes','string', 'max:255'],

'grade' => ['sometimes','nullable','string'],

'dni' => ['sometimes','integer'],

'location\_id' => ['sometimes','integer', 'exists:locations,id'],

'admin' => ['sometimes','boolean'],

'contacts' => ['sometimes', 'array'],

'contacts.\*' => ['integer', 'exists:users,id'],

'groups' => ['sometimes', 'array'],

'groups.\*' => ['integer', 'exists:groups,id'],

NOTA:

* Si los arrays de contacs y groups contienen valores, esos son los valores actuales, cualquier otro se borra.
* Si los arrays de contacs y groups no son enviados, se asume que NO tiene asociado ningún contacto / grupo.

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": {

"id": 1,

"name": "Usuario",

"surname": "Apellido",

"grade": "Docente civil",

"dni": "111333",

"location\_id": 2,

"user\_name": "usuario2",

"admin": 0,

"created\_at": **null**,

"updated\_at": "2022-11-02T12:37:39.000000Z"

}

}

* Si el **user\_id** enviado por URL no corresponde a un usuario existente, devuelve un error 404, con el siguiente mensaje:

{

"error": "No existe el usuario."

}

* Si el usuario logueado NO es administrador, devuelve un error 422, con el siguiente mensaje:

{

"error": "El Usuario debe se ADMINISTRADOR."

}

1. *Solicitud de listado de usuarios filtrado por location*
2. El *endpoint* devolverá un JSON con el listado de los usuarios filtrados por location.
3. Debe enviarse el token dentro de los headers de la petición.

Authorization: *Bearer token*

1. El token DEBE traer el user\_id del ADMIN que se validará si es un usuario Administrador.
2. En la URL se puede enviar o no el ID de una location. Si se envía se utiliza como filtro, sino se devuelven todos los usuarios.
3. Petición:
   * + **https://{{HOST}}/api/admin/users/{location\_id}**
     + Método: GET
4. ENTRADA: parámetros por URL
   * ***ID*** de location (nada).
5. 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:

{

"users": [

{

"id": 10,

"name": "Usuario10",

"surname": "Apellido10",

"grade": **null**,

"dni": "45000999",

"location\_id": 1,

"user\_name": "usuario10@cideso.com.ar",

"admin": 0,

"created\_at": "2022-11-02T12:19:10.000000Z",

"updated\_at": "2022-11-02T12:19:10.000000Z"

},

{

"id": 12,

"name": "Usuario11",

"surname": "Apellido11",

"grade": **null**,

"dni": "45000888",

"location\_id": 1,

"user\_name": "usuario11@cideso.com.ar",

"admin": 0,

"created\_at": "2022-11-02T14:29:08.000000Z",

"updated\_at": "2022-11-02T14:29:08.000000Z"

}

]

}

**NOTA:** Es opcional enviar el ID de lacation por URL, si se lo envía sólo se devolverán los usuarios relacionados con la location seleccionada, si no se especifica un ID de location, se devolverán todos los usuarios. En ambos casos ordenados ascendentemente por user\_name

1. **APIs de Administración de LOCATIONS**
2. *Solicitud de creación de location:*
3. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
4. Petición:
   * + **https://{{HOST}}/api/conversations/**
     + Método: GET

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

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

1. ENTRADA: *ninguna*
2. SALIDA:
3. *Solicitud de información de todas las locations:*
4. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
5. Petición:
   * + **https://{{HOST}}/api/conversations/**
     + Método: GET

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

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

1. ENTRADA: *ninguna*
2. SALIDA:
3. *Solicitud de información de una determinada location:*
4. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
5. Petición:
   * + **https://{{HOST}}/api/conversations/**
     + Método: GET

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

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

1. ENTRADA: *ninguna*
2. SALIDA:
3. *Solicitud de actualización de una determinada location:*
4. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
5. Petición:
   * + **https://{{HOST}}/api/conversations/**
     + Método: GET

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

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

1. ENTRADA: *ninguna*
2. SALIDA:
3. **APIs de Administración de GROUPS**
4. *Solicitud de creación de group:*
5. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
6. Petición:
   * + **https://{{HOST}}/api/conversations/**
     + Método: GET

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

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

1. ENTRADA: *ninguna*
2. SALIDA:
3. *Solicitud de información de todos los groups:*
4. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
5. Petición:
   * + **https://{{HOST}}/api/conversations/**
     + Método: GET

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

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

1. ENTRADA: *ninguna*
2. SALIDA:
3. *Solicitud de información de un determinado group:*
4. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
5. Petición:
   * + **https://{{HOST}}/api/conversations/**
     + Método: GET

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

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

1. ENTRADA: *ninguna*
2. SALIDA:
3. *Solicitud de actualización de un determinado group:*
4. El *endpoint* devolverá un JSON con la información de las conversaciones del usuario logueado.
5. Petición:
   * + **https://{{HOST}}/api/conversations/**
     + Método: GET

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

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

1. ENTRADA: *ninguna*
2. SALIDA:
3. **Información funcional:**
   1. **Proceso de login y creación de usuario:**
      1. El usuario se loguea en el Portal e ingresa al Chat.
      2. **Request:** *get('/user', 'UserController@getUserLoggedJWT');*
         1. Si el usuario ya existe → devuelve el objeto Usuario
         2. Si el usuario NO existe → devuelve 404
            1. **Request:** *Route::post('/newUser', 'UserController@createUserJWT');*

Si el **user\_name** enviado por JWT NO corresponde a un usuario existente, devuelve el objeto Usuario recién creado.

Si el **user\_name** enviado por JWT corresponde a un usuario existente, devuelve un error 422, con el siguiente mensaje:

{

"error": "Ya existe el usuario con el user\_name: vblanco"

}

* 1. **Proceso de actualización de usuario por parte de un ADMIN – Designación de un usuario como Administrador:**
     1. El usuario ingresa al Chat, se valida que el usuario sea Administrador.
     2. **Request:** *get('/admin/user/{user\_id}', 'UserController@getUserManagedJWT');*
        1. Se obtiene la información del usuario solicitado.
     3. ***Request:*** *post('/admin/user/{user\_id}', 'UserController@updateUserManagedJWT');*
        1. El Administrador podrá actualizar los datos personales de un usuario (NO podrá actualizar el user\_name, ya que el mismo fue tomado del JWT cuando éste se creó) y también designarlo como Administrador o viceversa.
  2. **Proceso de actualización de usuario por parte del mismo:**
     1. El usuario ingresa al Chat con un usuario existente.
     2. **Request:** *get('/user', 'UserController@getUserLoggedJWT');* 
        1. Se obtiene la información del usuario logueado.
     3. ***Request:*** *post('/admin/user/{user\_id}',* [*'UserController@updateUserManagedJWT*](mailto:'UserController@updateUserManagedJWT)*');*
        1. El usuario podrá actualizar sus datos personales.
        2. NO podrá actualizar el user\_name, ya que el mismo fue tomado del JWT cuando éste se creó)
        3. NO podrá cambiar el dato de si es o no Administrador.

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)
     + CLIENT\_2\_SECRET="client\_2" //se modificará según lo acordado