**Proyecto CHAT-EA-WEB**

**DIGID – CIDESO**

2020/2021

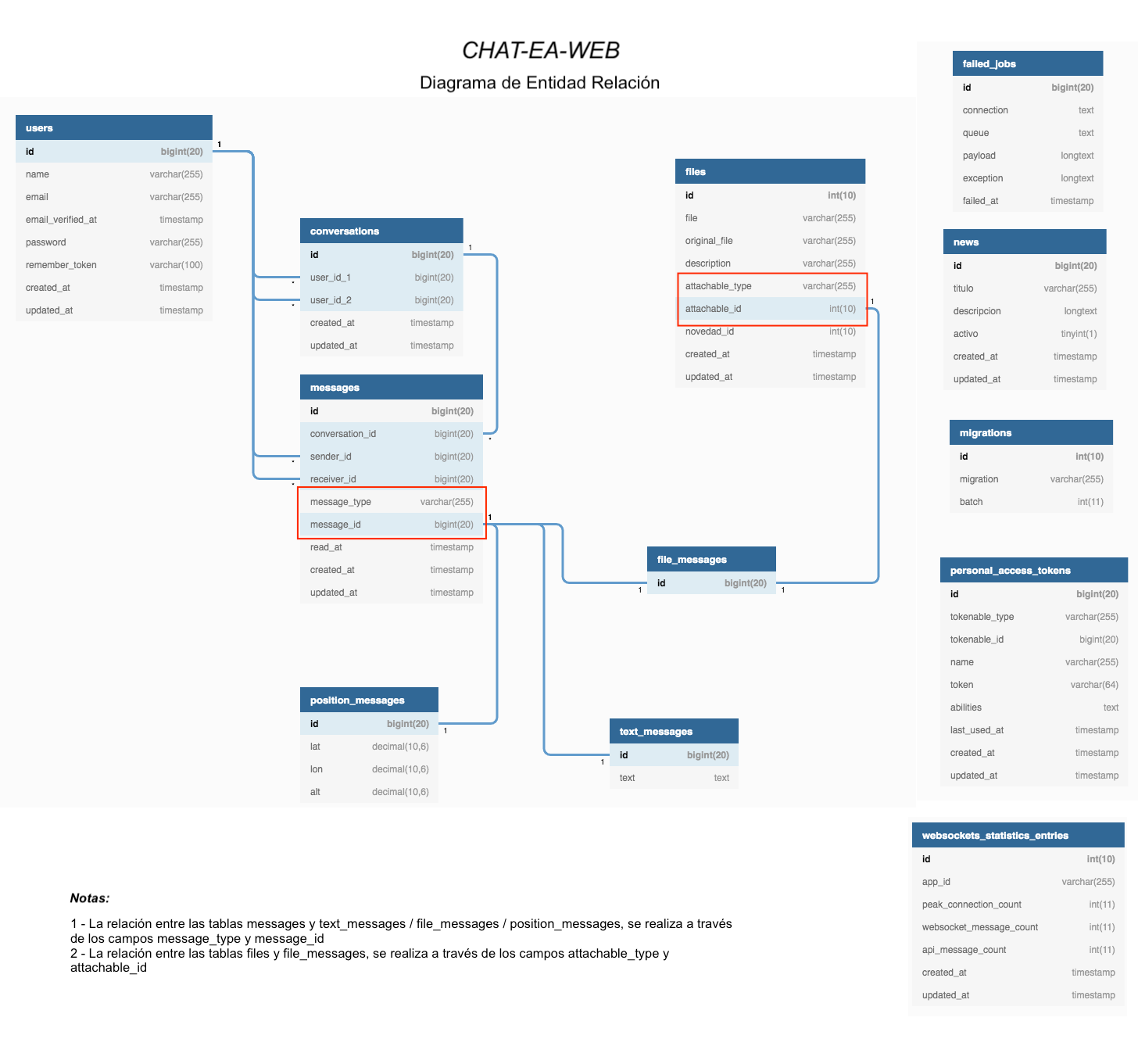
**ÍNDICE:**

1. [Características generales del proyecto CHAT-EA-WEB](#Vision_general)
2. [DER – Diagrama de Entidad Relación](#DER)
3. [Definición de APIs](#Definición_APIs)
   1. [Solicitud de login](#Loguin)
   2. [Solicitud de logout](#Logout)
   3. [Solicitud de conversaciones posibles para un usuario logueado](#Conversacion_usuario)
   4. [Solicitud de los mensajes de una conversación dada](#Mensaje_conversacion)
   5. [Solicitud de creación de nuevo mensaje de tipo TEXTO](#Creacion_mje_texto)
   6. [Solicitud de creación de nuevo mensaje de tipo FILE](#Creacion_mje_file)
   7. [Solicitud de creación de nuevo mensaje de tipo POSITION](#Creacion_mje_position)
4. ***Características generales del proyecto CHAT-EA-WEB:***
   1. CHAT-EA-WEB se desarrolló siguiendo un modelo de programación por capas. El objetivo fue el **desacoplamiento** de las capas de presentación (Frontend) y la lógica de negocio + la de datos (Backend).
   2. Ambas capas están alojadas en el web hosting **Neolo**, cada una como un subdominio diferente.
   3. La comunicación entre el Frontend y el Backend se realizó a través de APIs.
   4. Para el desarrollo del FRONT se utilizó **HTML** + **JavaScript** + **CSS** + **Vue** (framework de JavaScript)
   5. Para el desarrollo del BACK se utilizó **PHP** + **MYSQL** + **Laravel** (framework PHP)
   6. El envío de *datos en tiempo real*, se resolvío de dos maneras diferentes, a raíz de las limitaciones del servidor:
      1. **SSE** (Server Sent Events), se implementó en el servidor.
      2. **Websockets**, se implementó sólo en forma local, ya que en el servidor no se tenía permisos para correr el comando que inicia el servidor websockets.

***Aspectos de seguridad:***

* 1. Implementación de ***HTTPS*** *-* Hypertext Transfer Protocol Secure. Protocolo que permite establecer una conexión segura entre dispositivos de usuarios y los sitios web.
  2. ***Autenticación*** *y accesos protegidos* a través del uso de Tokens*.*
  3. ***Buenas prácticas*** para garantizar que el desarrollo de la aplicación minimice los riesgos de seguridad a través del desarrollo de código seguro.
  4. Implementación del estándar ***CORS*** (Cross-origin resource sharing)

1. **DER – Diagrama de Entidad Relación**



1. **Definición de APIs**
2. *Solicitud de login:*
3. Petición:
   * **https://emersis.casya.com.ar/api/v1/auth/login**
   * Método: POST
4. ENTRADA:

* **Form-data:**

'email' => ['required','email'],

'password' => 'required',

1. SALIDA:
   * Si la petición se intenta realizar sin un usuario autorizado, se obtendrá el código de estado “**401 – Unauthorized”**
   * 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": "Valeria",

        "email": "vale@test.com",

        "email\_verified\_at": null,

        "created\_at": null,

        "updated\_at": null

     },

"token": " 142|AVm0ShyZtTTQIGYK3VK46szeT9VOr7k8Iqu6VdJz

",

}

1. *Solicitud de logout:*
2. Petición:
   * **https://emersis.casya.com.ar/api/v1/auth/logout**
   * Método: POST

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:

{

"message": "Successfully logged out"

}

1. *Solicitud de conversaciones posibles para un usuario logueado:*
2. El *endpoint* devolverá un JSON con la información de las conversaciones activas más la información de los usuarios con los que todavía no se han iniciado conversaciones.
3. Petición:
   * + **https://emersis.casya.com.ar/api/v1/messages**
     + 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": 2,

    "conversations": [

        {

            "id": 3,

            "user\_dest": {

                "id": 3,

                "name": "Paul",

                "email": "paul@test.com"

            },

            "ammount\_no\_read": 2

        },

        {

            "id": 1,

            "user\_dest": {

                "id": 1,

                "name": "Valeria",

                "email": "vale@test.com"

            },

            "ammount\_no\_read": 6

        },

        {

            "id": 0,

            "user\_dest": {

                "id": 5,

                "name": "Daniel",

                "email": "dani@test.com"

            },

            "ammount\_no\_read": 0

        },

        {

            "id": 0,

            "user\_dest": {

                "id": 4,

                "name": "Hernán",

                "email": "herpa@test.com"

            },

            "ammount\_no\_read": 0

        }

    ]

}

* + - Donde:
* "*user\_origin*": es el ID del usuario logueado
* "*conversations*":
* "id": es el ID de la conversación (si es distinto de ***0*** es porque la conversación ya existe, de lo contrario quiere decir que todavía no existe una conversación iniciada con el usuario especificado en el campo "user\_dest").
* "user\_dest": información del usuario destinatario de la conversación.
* "ammount\_no\_read": Cantidad de mensajes NO leídos por el usuario logueado, en dicha conversación.
  + - Las Conversaciones se devuelven ordenadas de la siguiente manera:
      * Primero los usuarios con los que ya existe una conversación y se muestran, desde los que tienen mensajes más recientes a los que tienen mensajes menos recientes.
      * El resto de los usuarios con quienes no se ha iniciado una conversación, se muestran ordenados alfabéticamente.

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.
3. Petición:

* **https://emersis.casya.com.ar/api/v1/messages/{*ID\_conversacion* }**
* Método: GET

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

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

1. ENTRADA:

* ***URL: ID*** de la conversación. El ID de la conversación es ‘0’, si todavía no se inició la conversación (No hay mensajes dentro de esa conversación)

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\_origin": 1,

"messages": {

"current\_page": 1,

"data": [

{

"id": 10,

"conversation\_id": 2,

"sender\_id": 1,

"receiver\_id": 2,

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

"message\_id": 7,

"read\_at": null,

"created\_at": "2020-08-24T20:27:48.000000Z",

"updated\_at": "2020-08-24T20:27:48.000000Z",

"message": {

"id": 7,

"text": "Hoy es 24/08/20 17:26"

}

},

{

"id": 9,

"conversation\_id": 2,

"sender\_id": 1,

"receiver\_id": 2,

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

"message\_id": 6,

"read\_at": null,

"created\_at": "2020-08-24T20:26:32.000000Z",

"updated\_at": "2020-08-24T20:26:32.000000Z",

"message": {

"id": 6,

"text": "Hoy es 21/08/20 18:06"

}

},

{

"id": 7,

"conversation\_id": 2,

"sender\_id": 1,

"receiver\_id": 2,

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

"message\_id": 25,

"read\_at": null,

"created\_at": "2020-08-24T20:25:58.000000Z",

"updated\_at": "2020-08-24T20:25:58.000000Z",

"message": {

"id": 25,

"files": [

{

"id": 5,

"file": "files/BFX3FNYRHg95H5wbyCiryTTrhat7nekiYkZ3aGCU.png",

"original\_file": "iconos\_gral.png",

"description": "iconos\_gral.png",

"attachable\_type": "App\\Models\\FileMessage",

"attachable\_id": 25,

"novedad\_id": null,

"created\_at": "2020-08-24T20:25:58.000000Z",

"updated\_at": "2020-08-24T20:25:58.000000Z"

}

]

}

}

],

"first\_page\_url": "http://emersis\_back.test/api/v1/messages/2?page=1",

"from": 1,

"last\_page": 1,

"last\_page\_url": "http://emersis\_back.test/api/v1/messages/2?page=1",

"next\_page\_url": null,

"path": "http://emersis\_back.test/api/v1/messages/2",

"per\_page": 10,

"prev\_page\_url": null,

"to": 3,

"total": 3

}

}

1. *Solicitud de creación de nuevo mensaje de tipo TEXTO:*
2. El *endpoint* creará un nuevo mensaje de TEXTO para una conversación dada o para iniciar una nueva conversación.
3. Devolverá un JSON con la información del mensaje creado y el estado de respuesta a la petición.
4. Petición:
   * [**https://emersis.casya.com.ar/api/v1/messages**](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:** 
  + - 'message' => ['required','string', 'max:255'],
    - 'receiver\_id' => ['required','integer', 'exists:users,id'],

1. SALIDA:
   * Si el ***receiver\_id*** es el mismo id que el del usuario logueado, se obtendrá el código de estado “**500 - No se puede enviar mensaje al usuario destino”**.
   * Si el ***receiver\_id*** NO pertenece a un usuario existente, se obtendrá el código de estado “**500 - The given data was invalid”.**
   * 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 realizada con éxito",

    "conversation\_id": 1,

    "message\_id": 2,

    "sender\_id": 1,

    "receiver\_id": 4,

"read\_at": **null**,

**"message\_created": {**

**"id": 2,**

**"text": "Hoy es 21/08/20 17:27"**

**}**

}

1. *Solicitud de creación de nuevo mensaje de tipo FILE:*
2. El *endpoint* creará un nuevo mensaje de tipo FILE para una conversación dada o para iniciar una nueva conversación.
3. Devolverá un JSON con la información del mensaje creado y el estado de respuesta a la petición.
4. Petición:
   * **https://emersis.casya.com.ar/api/v1/messages/fileMessage**
   * Método: POST

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

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

1. ENTRADA:

* **Form-data:** 
  + - 'receiver\_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 ***receiver\_id*** es el mismo id que el del usuario logueado, se obtendrá el código de estado “**500 - No se puede enviar mensaje al usuario destino”**.
   * Si el ***receiver\_id*** NO pertenece a un usuario existente, se obtendrá el código de estado “**500 - The given data was invalid”.**
   * Los archivos se envían en un array, si algún archivo del array pasa las validaciones, se crea el mensaje con los archivos válidos, si alguno no lo fuera, se lo ignora.
   * Si todos los archicos del array 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": {

"file.0": [

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

"Cada archivo no puede ser mayor a 10MB"

]

}

}

* + 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 FILE realizada con éxito",

"conversation\_id": 1,

"message\_id": 6,

"sender\_id": 1,

"receiver\_id": 4,

"read\_at": **null**,

"message\_created": {

"id": 24,

"files": [

{

"id": 3,

"file": "files/ildFxLBGjBzXrwWzWvYP53N0CTNGnAppd6HZPZy2.png",

"original\_file": "iconos\_gral.png",

"description": "iconos\_gral.png",

"attachable\_type": "App\\Models\\FileMessage",

"attachable\_id": 24,

"novedad\_id": null,

"created\_at": "2020-08-24T20:19:12.000000Z",

"updated\_at": "2020-08-24T20:19:12.000000Z"

},

{

"id": 4,

"file": "files/7mNlTh2qzxlzWaYFpGo2eVAPPYLPVJzuFZz6sQ2M.jpeg",

"original\_file": "maquina\_3.jpeg",

"description": "maquina\_3.jpeg",

"attachable\_type": "App\\Models\\FileMessage",

"attachable\_id": 24,

"novedad\_id": null,

"created\_at": "2020-08-24T20:19:12.000000Z",

"updated\_at": "2020-08-24T20:19:12.000000Z"

}

]

}

}

1. *Solicitud de creación de nuevo mensaje de tipo POSITION:*
2. El *endpoint* creará un nuevo mensaje de *POSITION* para una conversación dada o para iniciar una nueva conversación.
3. Devolverá un JSON con la información del mensaje creado y el estado de respuesta a la petición.
4. Petición:
   * **https://emersis.casya.com.ar/api/v1/messages/positionMessage**
   * 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'],
    - 'receiver\_id' => ['required','integer','exists:users,id']

1. SALIDA:
   * Si el ***receiver\_id*** es el mismo id que el del usuario logueado, se obtendrá el código de estado “**500 - No se puede enviar mensaje al usuario destino”**.
   * Si el ***receiver\_id*** NO pertenece a un usuario existente, se obtendrá el código de estado “**500 - The given data was invalid”.**
   * 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": 4,

"message\_id": 104,

"sender\_id": 1,

"receiver\_id": 2,

"read\_at": **null**,

"message\_created": {

"id": 4,

"lat": "-34.588967",

"lon": "-34.588967",

"alt": "15.198195"

}

}