Universidad ORT Uruguay

Facultad de Ingeniería

# Diseño de aplicaciones 2
# Obligatorio 1:
# Evidencia del diseño y especificación de la API

Juan Pablo Poittevin(169766)
Joselen Cecilia(233552)

Entregado como requisito de la materia Diseño de aplicaciones 2

6 de mayo de 2021

Link al repositorio de GitHub
https://github.com/ORT-DA2/Poittevin-169766--Cecilia-233552-

# Declaraciones de autoría

Nosotros, Juan Pablo Poittevin y Joselen Cecilia, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos Diseño de Aplicaciones 2 ;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

# Índice general

# 1. Descripción de la API

## 1.1. Criterios REST

## 1.2. Descripción de los endpoint

Nuestra API se basa en REST, por lo cual, hemos definido un conjunto de resources (Recursos) en los cuales se basan nuestros endpoints.

Entendemos que para los requerimientos del sistema, con estos Resources es suficiente, algunas funcionalidades como por ejemplo agendar una meeting con un psicologo, están fuertemente relacionadas a los resources ya mencionados y por esto se manejan como endpoints especiales dentro de estos, un ejemplo es el ya mencionado agendar una meeting, el cual es parte del patient (POST /patient/schedule)

En las siguientes secciones se detallan los resources que existen en el sistema y el endpoint base de cada uno, para ver un detalle de todos los endpoints ir al Anexo.

### 1.2.1. PlayList

Representa una playlist, la cual es en si un conjunto de Audios, un conjunto de Categorias a las cuales pertenece la playlist, una descripcion, un nombre y una URL a la imagen que representa la playlist.

- **Endpoint base:** DOMAIN/api/playlist
- **Verbos HTTP:** GET - POST - DELETE - PUT
- **Headers:** Authorization (solo en el caso de POST, DELETE y PUT, asi validamos que sea administrador)

### 1.2.2. Audio

Representa un Audio reproducible, que esta compuesta por name, duration, AuthorName, UrlImage, UrlAudio, debemos aclarar que la duration es un string, el cual sera compuesto por un double y una "h", una "m", o una "s", de esta forma indicamos si el double esta en horas (h), minutos (m), segundos (s), ejemplo: "12h"seria un audio de 12 horas, "30s"seria un audio de 30 segundos, y "5m"seria un audio de 5 minutos, si se desea también se puede enviar por "30.4m"lo cual seria

30 minutos con 40 % de un minuto, es decir 24 minutos, esto mismo aplica con horas, en el caso de segundos se espera que sea un entero.

Cuando se obtiene un Audio (HTTP GET), siempre se convierte este numero en un double que representa las horas.

- **Endpoint base:** DOMAIN/api/audio
- **Verbos HTTP:** GET - POST - DELETE - PUT
- **Headers:** Authorization (solo en el caso de POST, DELETE y PUT, asi validamos que sea administrador)

### 1.2.3. Category

Este recurso representa una categoria, tanto de una playlist o de un audio, es importante saber que no se puede crear una categoria, las mismas vienen pre-definidas en el sistema, en el caso de los recursos de Playlist y Audio que tienen un conjunto de Categories entre sus datos, al enviar una Category, esta se asocia a la Playlist o Audio, pero no se crea, en caso de querer asociar una categoria que no existe en el sistema, se retornara un error. En el endpoint de category solo vamos a poder hacer la operacoin GET para obtener una categoria o un GET al /ID para obtener una especifica

- **Endpoint base:** DOMAIN/api/category
- **Verbos HTTP:** GET
- **Headers:** No Aplica

### 1.2.4. Problematic

Similar a las categorias, las problematic no se pueden crear, borrar o editar, simplemente estan predefinidas en el sistema y se asocian con otros recursos, los recursos con los cuales se pueden asociar lo veremos a continuacion. Similar a lo que sucede con category, solo tendremos los endpoints de GET

- **Endpoint base:** DOMAIN/api/problematic
- **Verbos HTTP:** GET
- **Headers:** No Aplica

### 1.2.5. Psychologist

Representa a un Psicólogo, los cuales tienen un Name, LastName, Address, WorksOnline, una lista de problematics, y una lista de meetings. Las problematics como ya se explico previamente, no pueden crearse al utilizar los endpoint de Psychologist, en las meeting pasa lo mismo,ya que para crear una meeting.

- **Endpoint base:** DOMAIN/api/psychologist
- **Verbos HTTP:** GET - POST - DELETE - PUT
- **Headers:** Authorization (solo en el caso de POST, DELETE y PUT, asi validamos que sea administrador)

### 1.2.6. Patient

Representa a un Paciente, los cuales tienen un Name, LastName, Cellphone, Birthday y una colección de meetings. El Birtday es un DateTime el cual debe enviarse utilizando el ISO 8601 https://en.wikipedia.org/wiki/ISO_8601

- **Endpoint base:** DOMAIN/api/patient
- **Verbos HTTP:** GET - POST - DELETE - PUT
- **Headers:** No Aplica

### 1.2.7. Administrator

El Administrator tiene Name, LastName, Email y Password, los mismos son los unicos que pueden hacer login en el sistema, asi mismo son los unicos que tienen acceso a algunas funcionalidades, como son el agregar Audio (POST /Audio), agregar Psychologist (POST /Psychologist), agregar otros Administradores y editar (PUT/PATCH) los resources de Audio, Psychologist y Administrador.

- **Endpoint base:** DOMAIN/api/administrator
- **Verbos HTTP:** GET - POST - DELETE - PUT
- **Headers:** Authorization (en todos los casos, asi validamos que sea administrador)

## 1.3. Descripción de códigos de éxito y error

Pasamos a detallar como se manejan las distintas respuestas HTTP en nuestra API. Esto se hizo siguiendo las buenas practicas detalladas en libro .ªPIgee Web API desing the missing link"

### 1.3.1. Handling 2XX

- **200**: expresa que "todo salio bien", normalmente un GET satifactorio es un ejemplo de esta respuesta
- **201**: expresa que .el objeto se creo correctamente", normalmente se usa durante en los metodos POST

### 1.3.2. Handling 4XX

- **401**: expresa que "no estas autorizado", lo cual significa que quisiste utilizar un endpoint el cual no tenes permisos basado en tu authentificacion, tambien se utiliza cuando hay un error de login.
- **404**: expresa que "no se encontró el objeto o endpoint", este error puede ocurrir en un GET, PUT, DELETE o PATCH
- **409**: expresa que "no se pudo procesar debido a un conflicto", un ejemplo de esto seria cuando se desea agregar una playlist con una categoria que no existe, al no poder crearse la categoria, se considera aun conflicto
- **422**: expresa que "la entidad no se puede procesar", esto son errores como un string superando el limite de caracteres, o un entero negativo donde no se permite para mencionar algunos ejemplos.

### 1.3.3. Handling 5XX

Se decidió controlar en un filtro todos las Excepciones y devolver siempre 500 y un mensaje genérico, de esta forma nos aseguramos no dejar posibles huecos de seguridad en los cuales se pueda obtener información sensible a través de una vulnerabilidad en el sistema. Idealmente esto se registraría en un log para poder ser analizado, pero escapa del alcance del proyecto

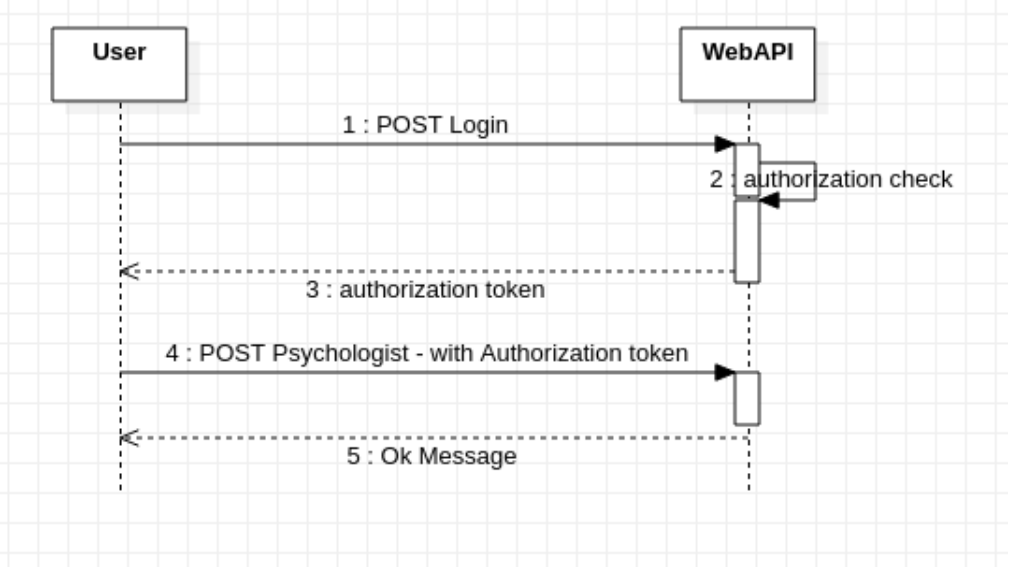
- **500**: Todo error no controlado por el sistema sera un 500.

## 1.4. Descripción proceso de Autenticación

Para el proceso de authentificacion se creo un endpoint especifico, este es: **/api/-login** El proceso consiste en enviar mediante HTTP POST una request de login, la cual en caso exitoso respondera un 200 con un token. Dicho token debera ser enviado en la header Authorization, siendo esta la forma en la cual el sistema reconocera si el administrador esta o no logeado.

Para simplificar el uso de la aplicacion y segun lo dado en clase, el token no tendra expiracion, en un caso ideal deberia darse un token de refresh y una expiracion, de esta forma evitamos el tener un token que valido de forma ïnfinita".

En caso de no poder authentificarse (datos incorrectos) se devolvera un error 401, en caso de poder autentificarlo pero no tener permiso para realizar la accion deseada, se muestra un 403.


User
WebAPI
1 : POST Login
2 : authorization check
3 : authorization token
4 : POST Psychologist - with Authorization token
5 : Ok Message