Universidad ORT Uruguay Facultad de Ingeniería

Diseño de Aplicaciones 2 Obligatorio 1

Evidencia del diseño y especificación de la API

Mauro Teixeira(211753) Rodrigo Hirigoyen (197396)

Docentes: Gabriel Piffaretti, Nicolas Fierro, Nicolas Hernandez

https://github.com/mauroteix/Teixeira-Hirigoyen

Entregado como requisito de la materia Diseño de Aplicaciones 2

6 de mayo de 2021

Índice

Criterios seguidos	2
Mecanismo de autenticación	2
Manejo del token	2
Encabezados para autenticación	3
Filtro de autenticación	3
Códigos de error	3
Controladores	4
CategoryController	4
PlaylistController	4
TrackController	4
PsychologistController	5
UserController	5
AdministratorController	5
SessionController	5
MedicalConditionController	5
Descripción de los resources	6
Administrator	6
Category	7
MedicalCondition	8
Playlist	9
Psychologist	12
Session	15
Track	15
User	18
Evidencia REST	19

Criterios seguidos

La aplicación fue diseñada considerando un conjunto de estándares establecidos por REST. Los mismos se listan a continuación:

- Los endpoint respetan las siguientes convenciones:
 - Se utilizan sustantivos entes que verbos. (ej. .../category)
 - Se busco que sea intuitiva y simple. Para esto los sustantivos utilizados son lo más nemotécnico en función de lo que se está solicitando.
 - · No se utilizan más de dos niveles
- Se utilizan los 4 verbos más comunes, estos son: GET, POST, PUT y DELETE.
- Se utilizan status codes de HTTP para el manejo de errores y excepciones donde la gran mayoría vienen acompañados de un mensaje que describe el error.

Mecanismo de autenticación

Para nuestra aplicación, utilizamos una autenticación basada en usuarios. Hoy en día se identifican dos tipos de usuarios:

- Administrador
- Usuario(Paciente)

El usuario(Paciente) puede realizar operaciones de ver las principales categorías y navegar entre su contenido y playlists, reproduciendo el contenido y agendar consultas con psicólogos en nuestro sistema. Para dichos endpoints, no se agregaron filtros de autenticación, dado que no requiere ningún nivel de seguridad.

Para los usuarios administradores, los cuales van a llevar el mantenimiento del sistema, se realizó un modelo el cual tiene un nombre, email, password y token de validación. Dicho token es único para cada usuario, el cual se va a utilizar en cada petición a los endpoints para validarse que el usuario tiene los permisos necesarios para realizar dicha operación.

Manejo del token

Nuestros administradores necesitan realizar un Login en nuestro sistema mediante email y password, y en caso de ser correctos, el mismo devuelve un token de autenticación, el cual debe ser usado en los Headers de las request, mediante el identificador 'Authentication'.

Encabezados para autenticación

```
[ServiceFilter(typeof(AuthorizationFilter))]
[HttpPost()]
5referencias @ 4/4 pasando | mauroteix, Hace 1 hora | 2 autores, 5 cambios
public IActionResult Add([FromBody] Administrator admin)
{
    try
    {
}
```

Como se muestra, cada endpoint que requieran de autenticación, se la anotación de ServiceFilter, con la clase que realiza dicha autorización.

Filtro de autenticación

Creamos una sección de Filtros en nuestro proyecto de webApi, dado que se va a utilizar en dicho proyecto, específicamente en los controllers. En dicha sección se va a crear nuestra clase de AuthorizationFilter, la cual se le inyecta nuestra interface que va a manejar todo el proceso de validación de token.

Códigos de error

Se utilizaron los siguientes códigos de errores:

Code	Description
200	Ok. Return the request object
401	Unauthorized. You do not have permissions to perform this action
404	NotFound. The requested object was not found
422	Unprocessable. Error in the semantics
501	InternalServerError. The server could not handle an exception in the system.

Nos permite indicar errores concretos, con mensajes para los usuarios a fin de comprender la situación actual de su petición. Los mismos se pueden observar en la documentación luego de realizar algún endpoint.

Controladores

De acuerdo a las funcionalidades solicitadas en la letra del problema planteado optamos por ocho controladores en nuestro sistema. Cada uno para las diferentes funcionalidades con cada recurso y serán presentados a continuación:



CategoryController

Vimos que era un controlador necesario ya que por letra los usuarios del sistema acceden a ver todas las categorías de forma pública.

- PlaylistController

Por letra y verificado en el foro, los administradores deben realizar el mantenimiento de las playlist por lo que nuestro sistema va a tener las funciones necesarias para realizar lo solicitado, y es por ello que se decidió implementarlo. A su vez el usuario puede navegar entre las playlist

- TrackController

También por letra indica que el administrador debe realizar alta y baja de un contenido reproducible de este modo es que decidimos incluir este controlador. También el usuario puede navegar entre el contenido reproducible.

- PsychologistController

En este caso también podemos ver que los psicólogos deben ser mantenidos en nuestro sistema, por lo que era sumamente necesario tener entre controlador así los administradores podrán mantener dicha entidad.

- UserController

El problema solicita agendar la consulta con un psicólogo ingresando los datos del usuario, por lo que vimos conveniente tener un controlador específico para realizar la agenda de su consulta. Y que ésta permita realizar esa operación.

- AdministratorController

Como la letra lo indica un administrador va realizar las tareas del mantenimiento del sistema.

- SessionController

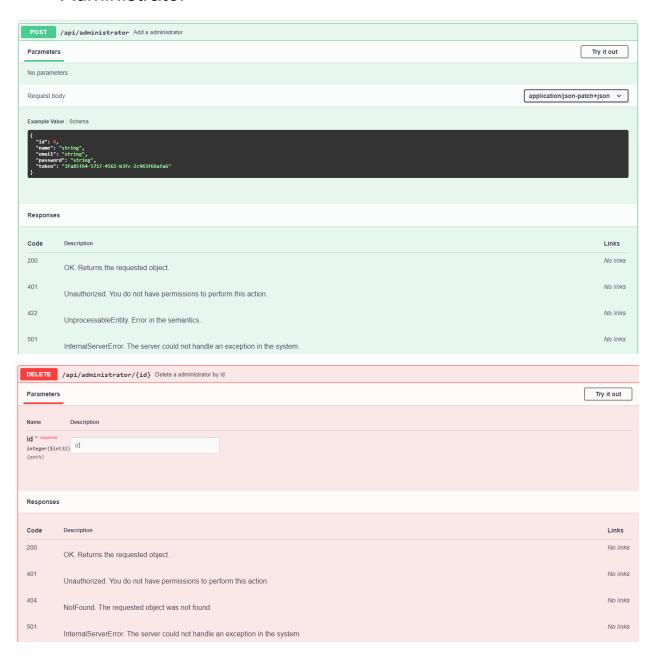
Debido a que el sistema solo permite realizar ciertas funcionalidades a los administradores es de aquí que se decide tener entre controlador para manejar la sesión.

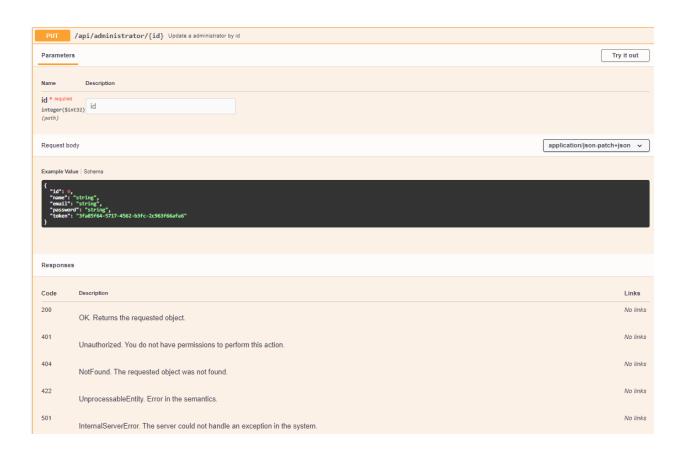
MedicalConditionController

Los usuarios del sistema deben poder ver las medical condition para luego poder hacer la agenda para esa problemática.

Descripción de los resources

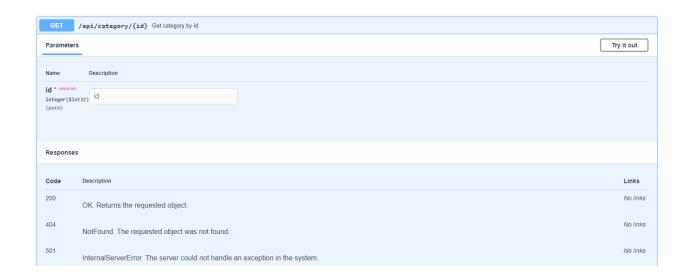
Administrator





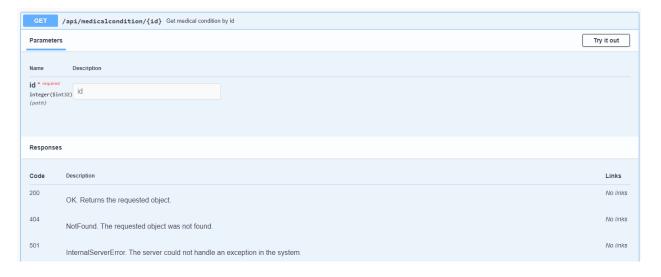
Category



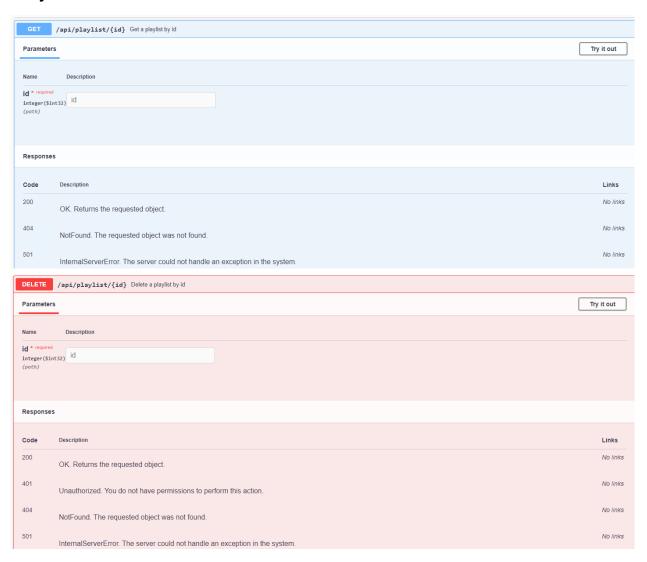


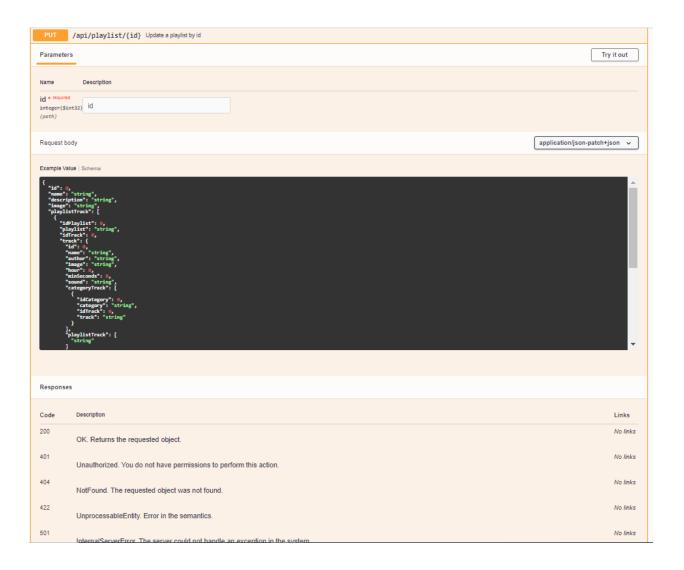
MedicalCondition

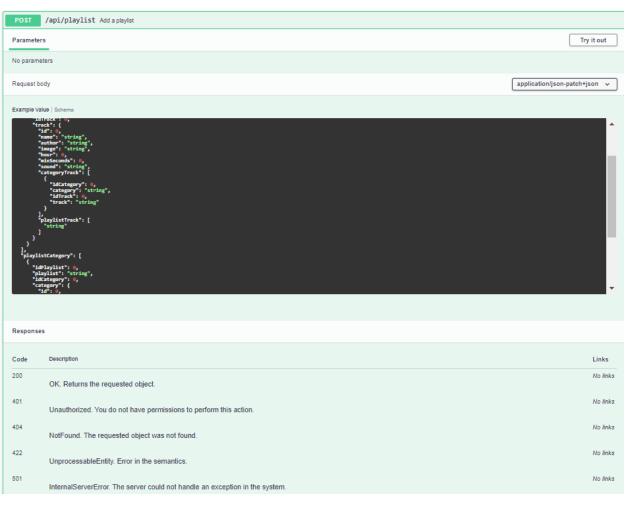




Playlist



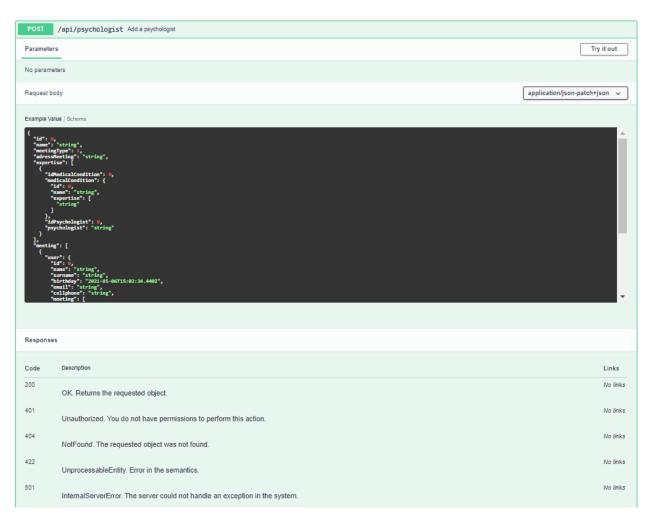


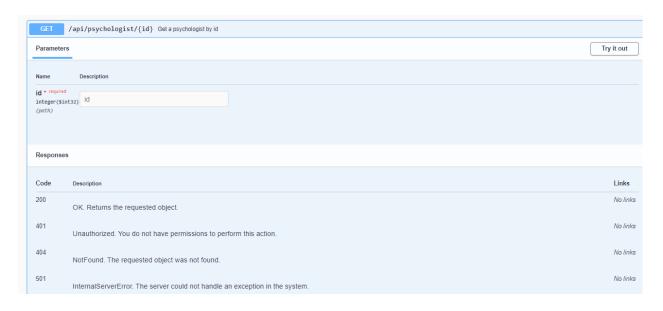


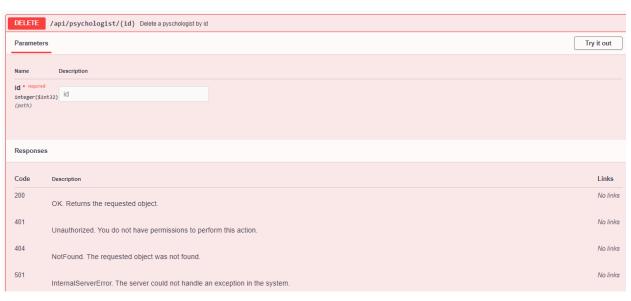


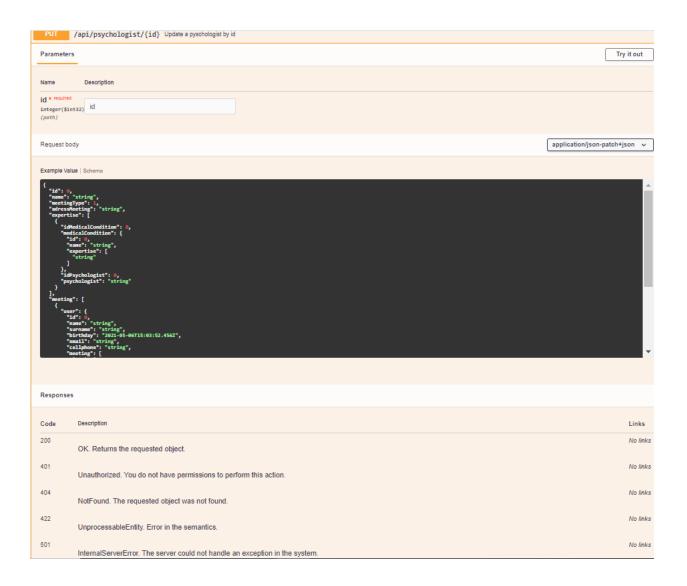
Psychologist



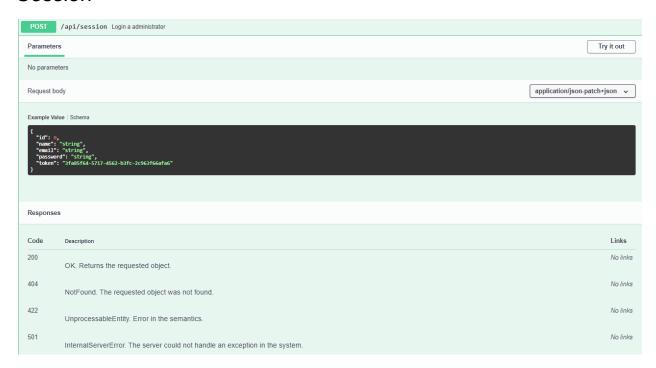




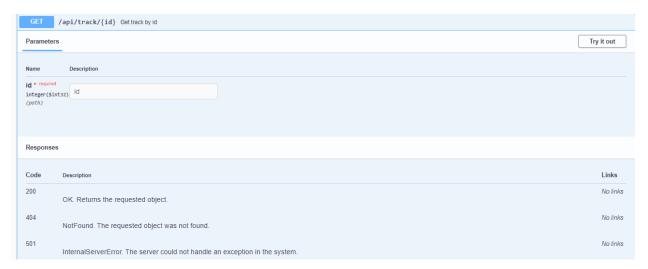


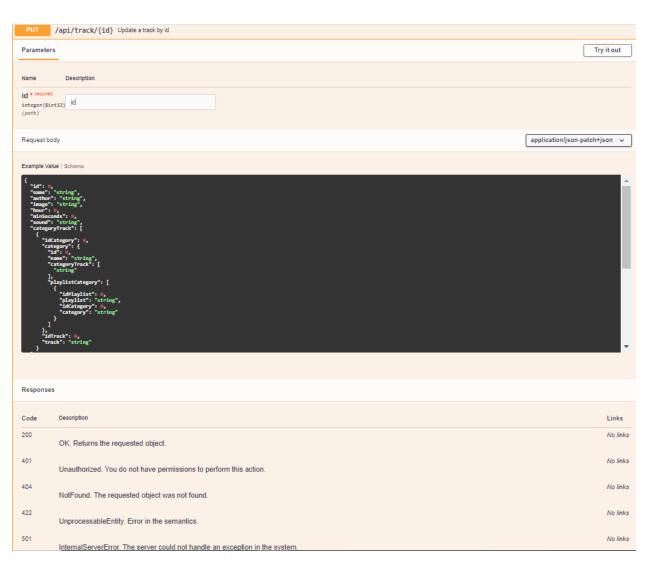


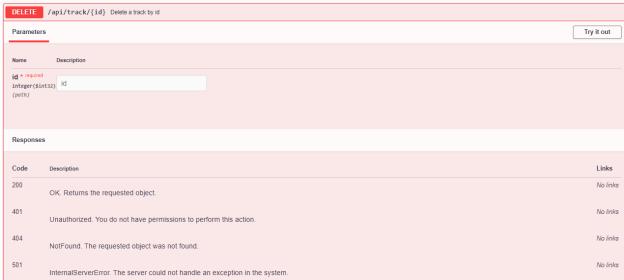
Session

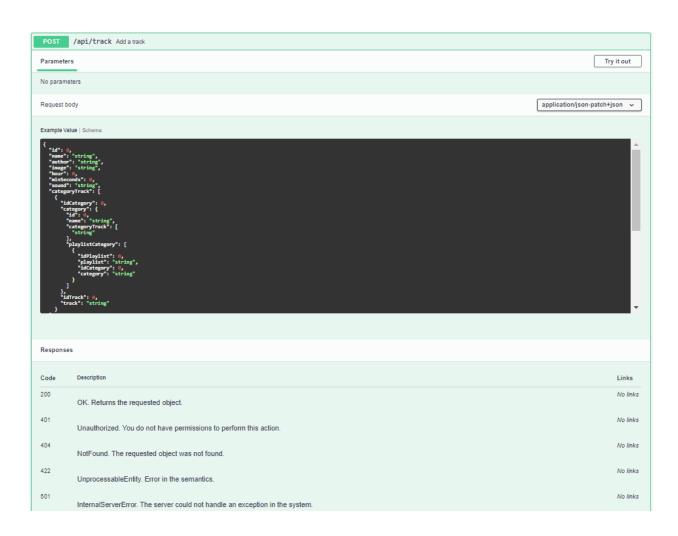


Track



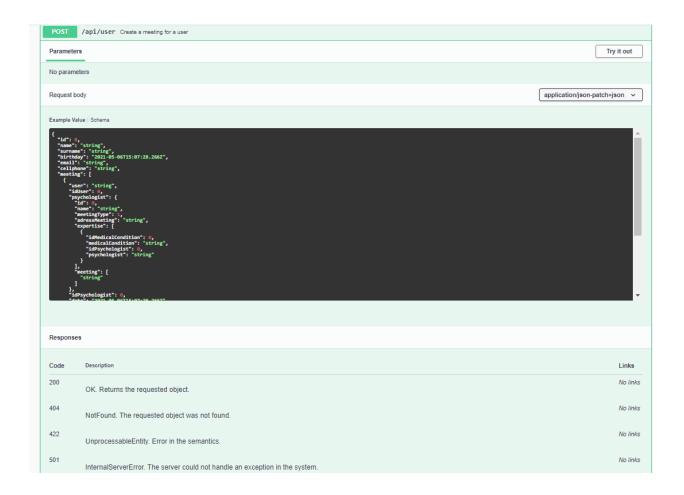








User



Evidencia REST

• <u>Cliente-servidor</u>

Tenemos por un lado un servidor el cual le brinda al cliente servicios, expone una serie de endpoint los cuales el usuario puede usar para poder hacer operaciones. Estos endpoint son los descritos anteriormente.

Sin estado

El servidor cada vez que recibe una nueva request crea los servicios nuevamente, esto se lo especificamos en el startup de nuestra app, lo cual hace que cada request que le llegue no tenga una trazabilidad sino que se trata siempre como una nueva request que llega.

• Se puede almacenar en caché

Al tener una aplicación sin estado, el usuario debe ser capaz de almacenar los datos que considere necesarios para poder llevar un estado, como por ejemplo para la autenticación del mismo debe guardar el token que el servidor le provee y utilizarlo en el header de las request que hace.

Interfaz uniforme

- Mensajes autodescriptivos
- Identificación de recursos
- Manipulación de recursos a través de representaciones

En nuestra aplicación cumplimos con estas cuatro condiciones, esto se puede apreciar viendo cada endpoint que se describió anteriormente, como identificamos los recursos, como serializamos los recursos mapearlos con entidades de nuestro sistema, como manejamos los errores brindándole al usuario un texto informativo para que pueda realmente darse cuenta de lo que pasó.

• Sistema en capas

Como se muestra en el documento de descripción de diseño tenemos una aplicación que está dividida en capas, en el documento mostramos y describimos cada una de estas capas. Por otra parte utilizamos todos los verbos http (GET / PUT / POST / DELETE) dependiendo la operación que queramos realizar, identificándose correctamente.