Universidad ORT Uruguay

Facultad de Ingeniería

Diseño de aplicaciones II Obligatorio I

Evidencia del diseño y especificación de la API.

Juan Manuel Gallicchio 233335 Federico Carbonell 224359

https://github.com/ORT-DA2/CarbonellGallicchio

Índice

Índice	2
Evidencia del diseño y especificación de la API	3
Estilo de arquitectura	3
Client–server	3
Stateless	3
Cacheable	3
Uniform interface	3
Layered system	4
Información sobre la implementación	4
Verbos HTTP	4
Códigos de respuesta	5
Mecanismo de autenticación	5
Endpoints	6
Administrators	7
Audio Content	9
Categories	11
Consultation	12
Problematics	12
Psychologists	13
Sessions	15

Evidencia del diseño y especificación de la API

El propósito de este documento es informar como esta compuesta la Api, para facilitar el uso de la misma. Se detallara el estilo de arquitectura utilizado y cómo está compuesta, también los códigos de error utilizados y el mecanismo de autentificación.

Estilo de arquitectura

El estilo de arquitectura utilizado es REST este indica seis principios que deben cumplirse para obtener dicha arquitectura. A continuación una breve explicación de cómo se implementaron en nuestra Api.

Client-server

El servidor el cual brinda los servicios al cliente se encuentra separado de la implementación que usará, expone una serie de endpoints los cuales se podrán utilizar para realizar operaciones. Esto nos genera portabilidad y extensibilidad ya que no depende de la implementación del cliente siendo la api independiente.

Stateless

Cada request realizada el servidor crea los servicios nuevamente, por dicha razón es necesario enviarle todos los datos necesarios a cada llamada realizada para que el servidor sea capaz de procesarla.

Cacheable

Se le indica al cliente que respuesta de servicios puede ser cacheada para no tener que realizarla nuevamente. En esta entrega no se implementó pero es posible hacerlo si se desea.

Uniform interface

Rest nuevamente define cuatro principios para cumplir este principio, estos son: identification of resources, manipulation of resources through representations; self-descriptive messages, y hypermedia.

Se puede apreciar cómo se cumplen estos principios ya que se identifican claramente los recursos en cada endpoint, se serializan los recursos al mapearlos en modelos y tambien se crearon distintos proyectos de excepciones para poder customizar los mensajes dependiendo de la situación.

Layered system

El sistema se encuentra dividido en varias capas, principalmente estas son WebApi, Adapter, Business Logic y Data Access. Para tener mayor información recomendamos consultar el documento "Descripción del diseño" donde profundizamos sobre cómo está compuesto en detalle todo el sistema.

Información sobre la implementación

Profundizaremos sobre los verbos utilizados, los códigos de error manejados y los endpoints de la Api.

Verbos HTTP

Los verbos utilizados para la implementación son cuatro y estos son:

- GET sirve para obtener información sobre un elemento.
- POST sirve para agregar un elemento.
- PUT sirve para modificar la información de un elemento.
- DELETE sirve para eliminar un elemento.

La combinación de todos los verbos genera que REST sea CRUD, todos los anteriores son impotentes a excepción del POST ya que modifica el estado del servidor.

Códigos de respuesta

Los códigos de error utilizados se pueden separar en familias, son los siguientes:

Success:

- 200 Indica éxito en el request realizado y se retorna el elemento.
- 201 Indica que la creación del elemento fue correcta, se utiliza para el verbo POST.
- 204 Indica éxito en el request pero el retorno es nulo, se utiliza para los verbos PUT o DELETE.

Error:

- 400 Indica que existe un error sobre el elemento enviado, algún atributo obligatorio vacío por ejemplo.
- 404 Indica que el elemento solicitado no fue encontrado o no existe.

Authorization:

- 401 Indica la falta del token de administrador.
- 403 Indica la falta de permisos para realizar la operación.

Internal error:

• 500 Indica un error en el servidor que no fue esperado.

Mecanismo de autenticación

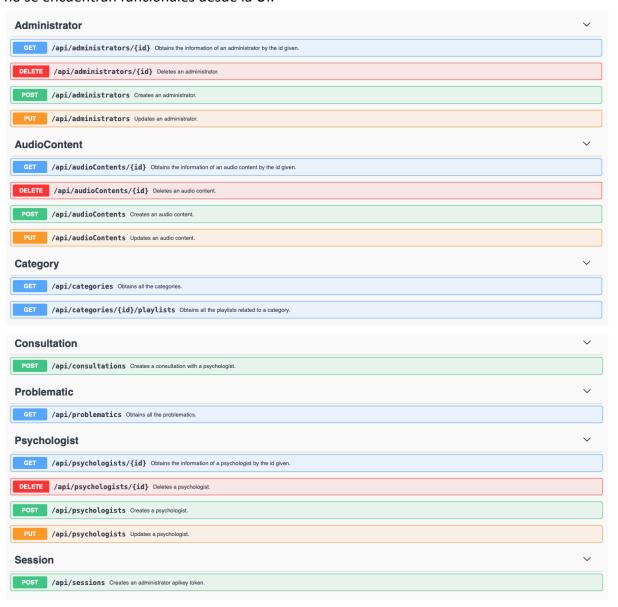
Para la autenticación de ciertos servicios es necesario un token administrador. Para la implementación de esta fue utilizado un AuthorizationFilter, este filter es ejecutado antes de realizar la request al controller por lo que si no se presenta un token o este no presenta los permisos necesarios la request se rechaza y no se ejecuta el controller.

Endpoints

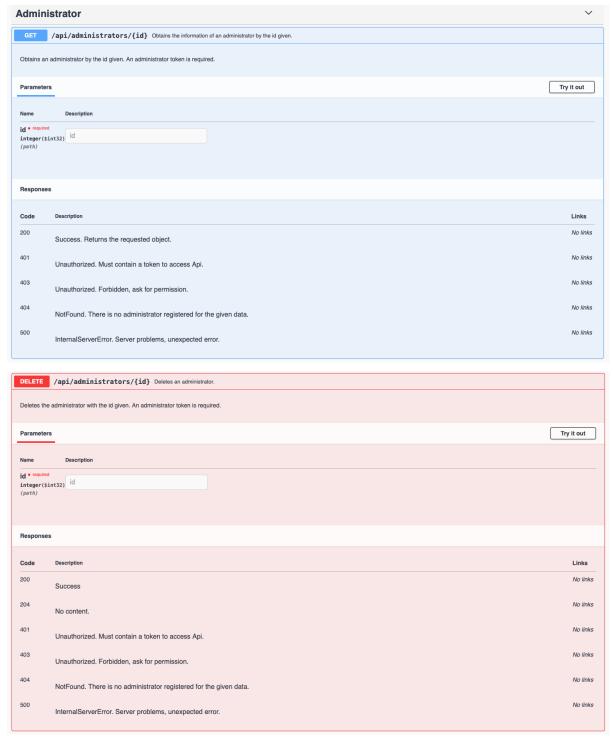
Para cada recurso se implementó un controller para realizar una correcta separación de responsabilidades.

Para documentar estos se utilizó swagger, luego de compilar el proyecto WebApi se puede visualizar en https://localhost:5001/swagger/index.html.

Para esta entrega no logramos agregar el token por lo que los endpoints que necesitan este no se encuentran funcionales desde la UI.

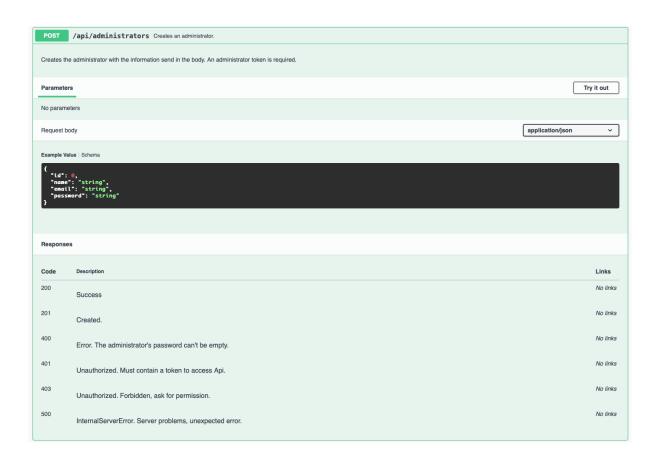


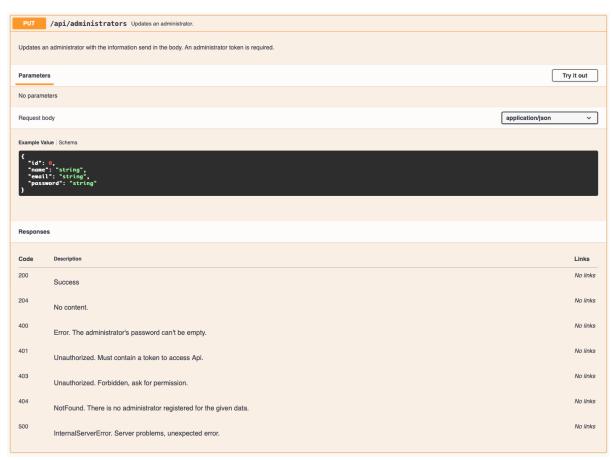
Administrators



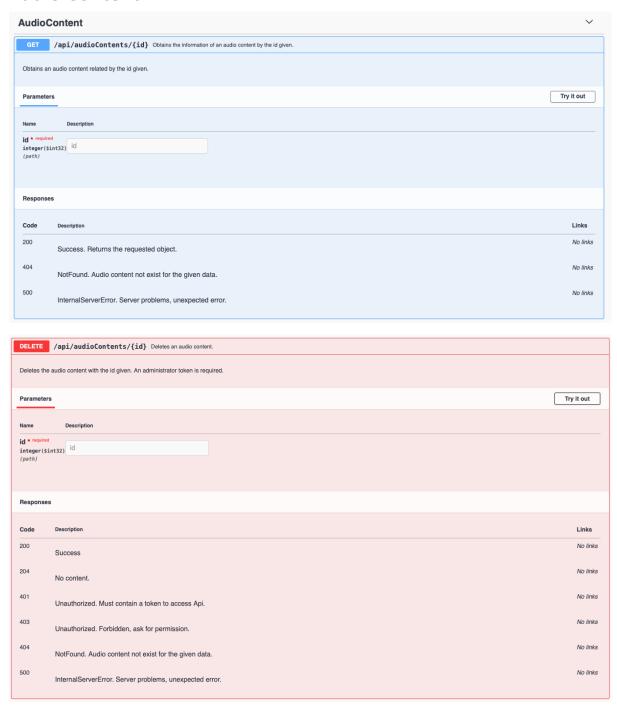
Errores faltantes para los verbos POST y PUT de Administrators:

- 400: Error. The administrator's email can't be empty.
- 400: Error. The administrator's password can't be empty.



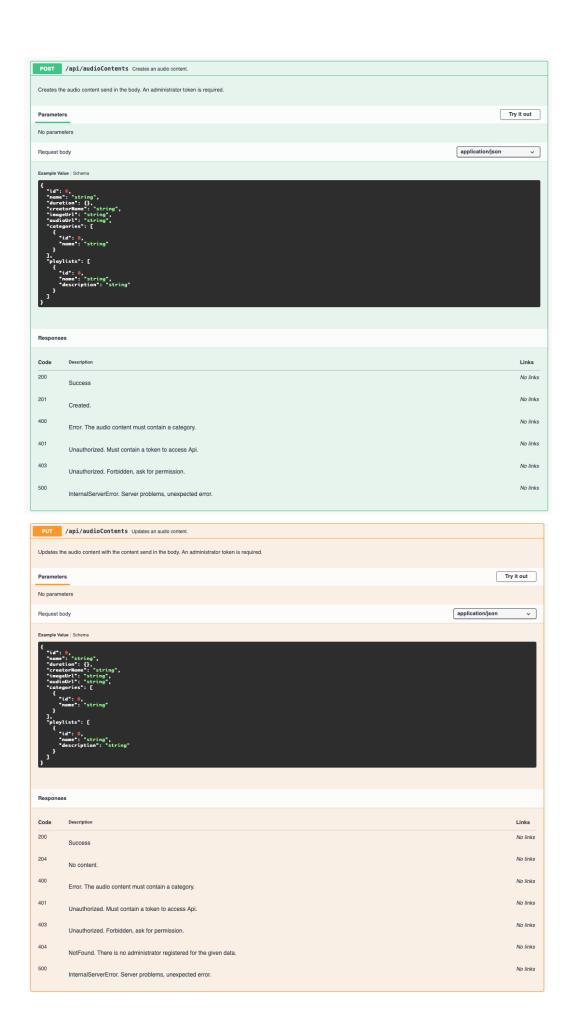


Audio Content

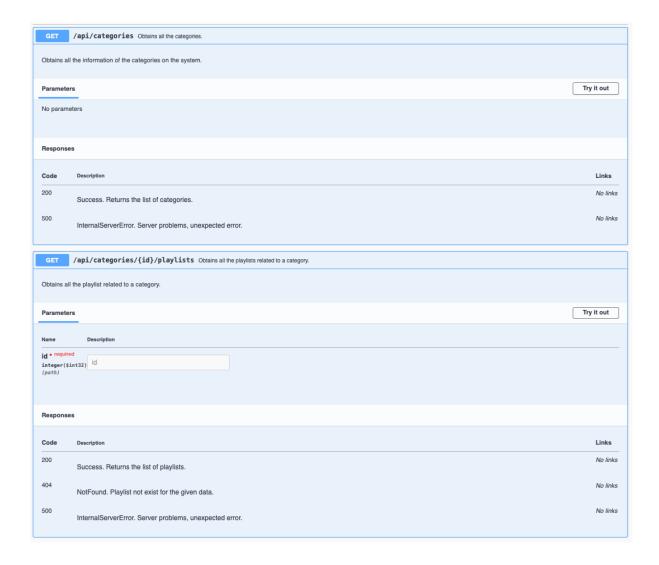


Errores faltantes para los verbos POST y PUT de Audio Content:

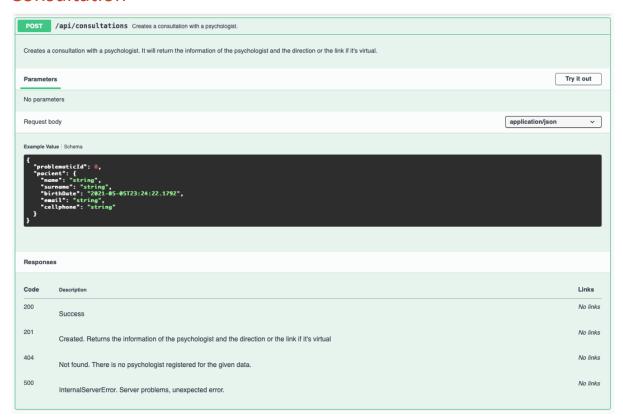
- 400: Error. The audio content must contain a playlist.
- 400: Error. The audio content name can't be empty.



Categories



Consultation

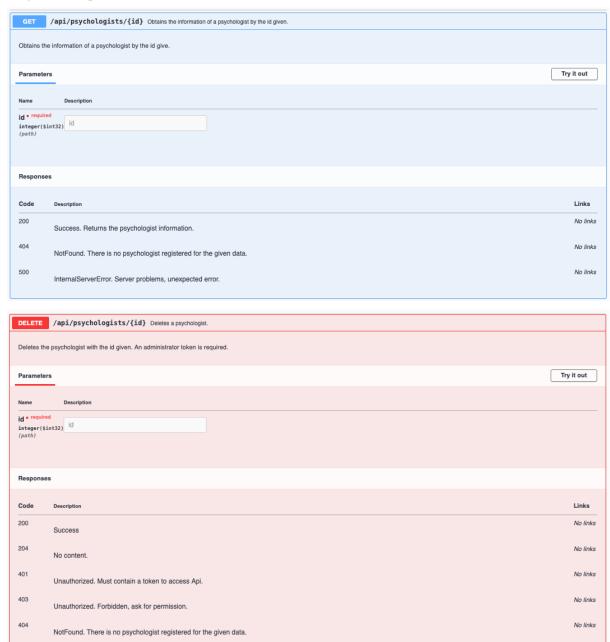


Problematics



Psychologists

500



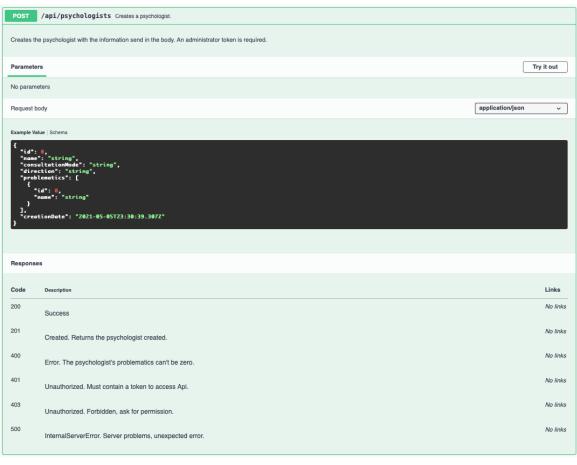
Errores faltantes para los verbos POST y PUT de Psychologist:

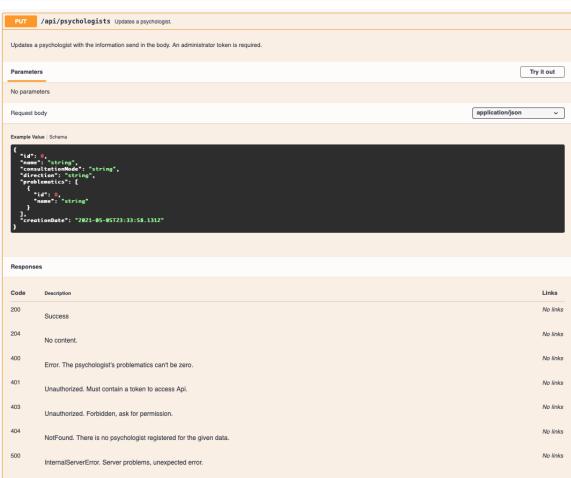
- 400: Error. The psychologist's consultation mode can't be empty.
- 400: Error. The psychologist's direction can't be empty.

InternalServerError. Server problems, unexpected error.

- 400: Error. The psychologist's problematics can't be zero.
- 400: Error. The psychologist's consultation mode must be 'Virtual' or 'Presencial.
- 400: Error. The psychologist's problematics must be exactly three

No links





Sessions

