**UNIVERSIDAD NACIONAL AUTONOMA DE NICARAGUA**



**FACULTAD DE CIENCIAS Y TECNOLOGIA**

**Carrera:** Ingeniería en Telemática

**Componente:** Software como un servicio

**Grupo:**1

**Docente:** Erving Montes

**Elaborado por:**

* Joysi Julissa Garcia Soza.
* Rosa Arellys Orozco Lira.

**Fecha**: 06 de octubre de 2024

**¡A la libertad por la Universidad!**

**INFORMACION.**

**Nombre:** Nimbus2024API.

**Descripción:** Nimbus2024API Fue creada en octubre del 2024 con el fin de lograr una amplia interacción entre usuarios. Nimbus2024API es una herramienta creativa para que las personas convivan de manera saludable. No se requiere autenticación para acceder, todos los recursos están completamente disponibles para su entera disposición.

**Versión:** v1.0.0

**Servidor Base:**

**POLITICAS DE USO.**

**Nimbus2024API** es gratuita y de uso abierto. Solicitamos a todas las personas que por favor respeten nuestras políticas de uso.

Políticas:

* Sea amable y amigable con los usuarios de Nimbus2024API.
* Si detecta vulnerabilidades en el sistema, reportarlas de manera adecuada y responsable.
* No está permitido utilizarla con fines maliciosos, ilegales o no autorizados.
* Desarrolladores que deseen integrar la API en sus aplicaciones de manera ética y conforme a las leyes.
* Los desarrolladores deben de implementar medidas de seguridad razonables en sus aplicaciones para proteger los datos obtenidos a través de Nimbus2024API.
* No está permitido replicar, copiar o modificar partes de la API sin autorización.
* Los usuarios de la API son responsables del uso adecuado de la misma y de cualquier consecuencia derivada de su integración.
* Nimbus2024API se reserva el derecho de cancelar el acceso de cualquier usuario que viole estas políticas de uso, infrinja las leyes aplicables o haga uso inapropiado de la API.

**SOPORTE Y CONTACTO.**

Para preguntas o soporte sobre el uso de Nimbus2024API, pueden ponerse en contacto a través del correo [**soporteNimbus2024API@gmail.com**](mailto:soporteNimbus2024API@gmail.com) **.**

**CODIGOS DE LA API Y PARA QUE FUNCIONA.**

**class BooksController < ApplicationController**

BooksController hereda de ApplicationController, lo que le da acceso a funcionalidades básicas de Rails y a métodos de autenticación y autorización.

**before\_action :set\_book, only: %i[show update destroy**

before\_action es un callback que se ejecuta antes de las acciones especificadas (en este caso, show, update, y destroy). Este método se usa para establecer la variable @book con el libro correspondiente, evitando la repetición de código en cada acción.

**Acciones del controlador**

**Index (GET /books)**

**def index**

**@books = Book.all**

**render json: @books**

**end**

Recupera todos los libros de la base de datos y los devuelve en formato JSON.

**Show (GET /books/**

**def show**

**render json: @book**

**end**

Muestra un libro específico (ya establecido en el callback) y lo devuelve en formato JSON.

**Create (POST /books)**

**def create**

**@book = Book.new(book\_params)**

**if @book.save**

**render json: @book, status: :created, location: @book**

**else**

**render json: @book.errors, status: :unprocessable\_entity**

**end**

**end**

Crea un nuevo libro con los parámetros permitidos. Si se guarda correctamente, devuelve el libro creado con un estado 201 (creado). Si hay errores, devuelve los errores de validación.

**Update (PATCH/PUT /books/**

**def update**

**if @book.update(book\_params)**

**render json: @book**

**else**

**render json: @book.errors, status: :unprocessable\_entity**

**end**

**end**

Actualiza un libro existente con los nuevos parámetros. Si la actualización es exitosa, devuelve el libro actualizado. Si hay errores, devuelve los errores de validación.

**Destroy (DELETE /books/**

**def destroy**

**@book.destroy!**

**end**

Elimina el libro especificado. Si se elimina correctamente, normalmente se devolvería un estado 204 (sin contenido), aunque aquí no se especifica un render después de la eliminación.

**Métodos Privados**

**Set Book**

**private**

**def set\_book**

**@book = Book.find(params[:id])**

**end**

Busca un libro por su ID y lo asigna a @book. Esto es utilizado por las acciones que requieren un libro específico**.**

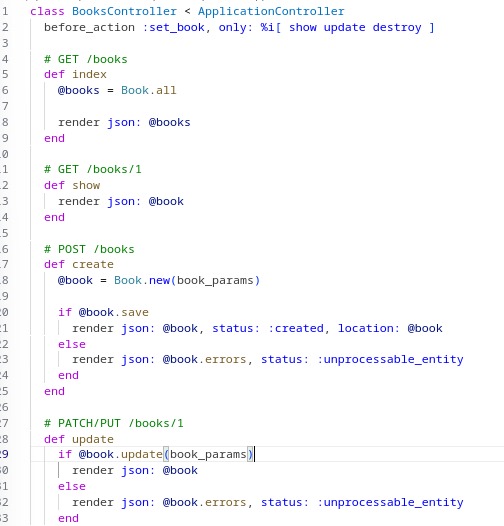
**Permitir Parámetros**

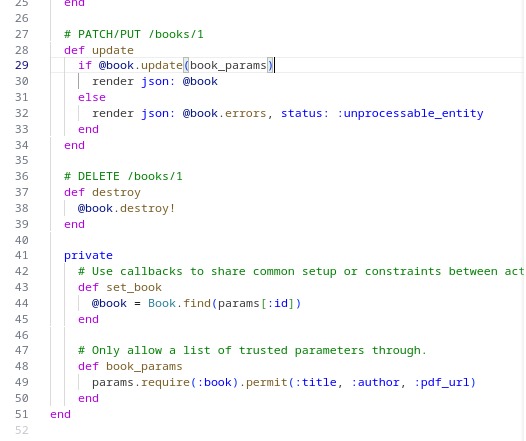
**def book\_params**

**params.require(:book).permit(:title, :author, :pdf\_url)**

**end**

Define qué parámetros son permitidos al crear o actualizar un libro, asegurando que solo se reciban los datos esperados.

.



class HousesController < ApplicationController

Esta línea define la clase HousesController, que hereda de ApplicationController. Esto significa que hereda todas las funcionalidades y configuraciones básicas que definiste en ApplicationController.

**2. Filtros (before\_action)**

before\_action :set\_house, only: %i[ show update destroy ]

before\_action es un callback que se ejecuta antes de que se llame a cualquier acción en el controlador. En este caso, set\_house se ejecutará antes de las acciones show, update y destroy.

Esto evita la repetición de código, ya que la función set\_house se encargará de encontrar la casa correspondiente según el id proporcionado en los parámetros.

**3. Acciones del Controlador**

**3.1. Index**

def index

@houses = House.all

render json: @houses

end

Método: GET /houses

Descripción: Este método recupera todas las casas de la base de datos usando House.all y las devuelve en formato JSON.

Renderiza: La respuesta se renderiza como JSON, que es el formato estándar para APIs.

**3.2. Show**

def show

render json: {

house: @house,

characters: @house.characters,

spells: Spell.all,

books: Book.all

}

end

Método: GET /houses/:id

Descripción: Este método devuelve la información de una casa específica, así como sus personajes y otros recursos (hechizos y libros).

Renderiza: Crea un objeto JSON que incluye la casa (@house), sus personajes (@house.characters), y listas de todos los hechizos y libros. Esto permite que quien consuma la API obtenga información relacionada en una sola respuesta.

**3.3. Create**

def create

@house = House.new(house\_params)

if @house.save

render json: @house, status: :created, location: @house

else

render json: @house.errors, status: :unprocessable\_entity

end

end

Método: POST /houses

Descripción: Este método crea una nueva casa. Primero, se inicializa un nuevo objeto House con los parámetros permitidos.

Guardar: Intenta guardar la nueva casa en la base de datos. Si tiene éxito, devuelve la casa recién creada con un estado 201 (creado). Si falla, devuelve los errores de validación con un estado 422 (entidad no procesable).

**3.4. Update**

def update

if @house.update(house\_params)

render json: @house

else

render json: @house.errors, status: :unprocessable\_entity

end

end

Método: PATCH/PUT /houses/:id

Descripción: Este método actualiza una casa existente. Usa @house.update para aplicar los cambios.

Renderiza: Si la actualización es exitosa, devuelve la casa actualizada. Si falla, devuelve los errores de validación.

**3.5. Destroy**

def destroy

@house.destroy!

end

Método: DELETE /houses/:id

Descripción: Este método elimina una casa existente. Usa @house.destroy! para eliminar la casa de la base de datos. Si la eliminación tiene éxito, devuelve un estado 204 (sin contenido).

**4. Métodos Privados**

**4.1. set\_house**

def set\_house

@house = House.find(params[:id])

end

Descripción: Este método busca la casa en la base de datos utilizando el id proporcionado en los parámetros. Se ejecuta antes de las acciones show, update, y destroy. Almacena el objeto House en @house, que puede ser usado en esas acciones.

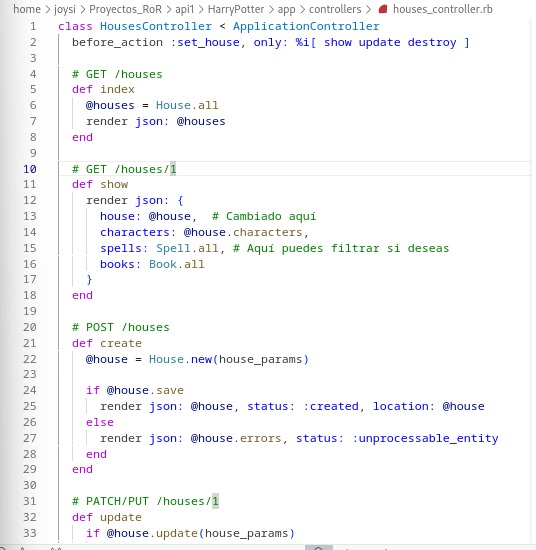
**4.2. house\_params**

def house\_params

params.require(:house).permit(:name, :founder, :colors, :animal)

end

Descripción: Este método permite definir qué parámetros son aceptables cuando se crean o actualizan casas. params.require(:house) asegura que haya un objeto house en los parámetros, y permit especifica cuáles atributos se pueden usar (en este caso, name, founder, colors, animal). Esto es crucial para proteger tu API de ataques maliciosos.





1. **Definición de Rutas para Recursos**

resources :books

resources :spells

resources :books: Crea rutas RESTful para el recurso "books". Esto incluye las rutas para las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de libros.

resources :spells: Hace lo mismo para el recurso "spells".

**2. Rutas Anidadas para Casas y Personajes**

resources :houses do

resources :characters, only: [:index]

end

resources :houses: Crea rutas RESTful para el recurso "houses".

Anidación: Dentro de resources :houses, defines que las rutas para "characters" estarán anidadas. Esto significa que puedes obtener personajes específicos de una casa.

only: [:index]: Esto limita las rutas generadas para characters solo a la acción index, que generalmente se usa para listar los personajes de una casa específica. La ruta resultante sería algo como /houses/:house\_id/characters.

**3. Rutas Globales para Personajes**

resources :characters

Esto mantiene la ruta global para "characters", lo que permite realizar operaciones CRUD sobre personajes de manera independiente de las casas.

**4. Rutas para Autenticación**

devise\_for :users

Este comando configura las rutas necesarias para la autenticación de usuarios utilizando la gema Devise. Esto incluye rutas para registro, inicio de sesión, cierre de sesión, recuperación de contraseñas, etc.

**5. Ruta de Verificación de Salud**

get "up" => "rails/health#show", as: :rails\_health\_check

get "up": Define una ruta que responde a una solicitud GET en la URL /up.

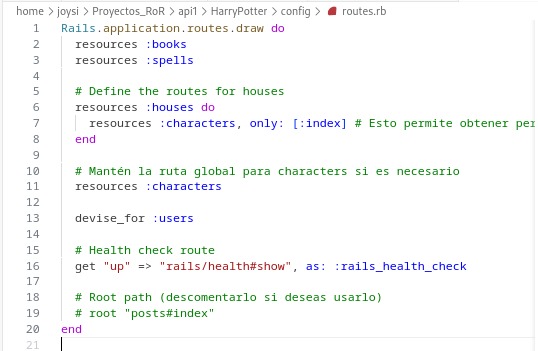
"rails/health#show": Esta ruta se conecta a la acción show del controlador HealthController dentro del espacio de nombres rails. Es útil para monitorear el estado de la aplicación.

as: :rails\_health\_check: Le da un nombre a la ruta, lo que te permite referenciarla en otras partes de tu código.

**6. Ruta Raíz**

# root "posts#index"

Esta línea está comentada, pero si la descomentas, definirá la ruta raíz de tu aplicación (es decir, la página que se carga cuando accedes a la URL base del servidor). En este caso, se configuraría para que se dirigiera al índice del controlador Posts.



**Creación de Registros**

Casas (Hogwarts Houses)

houses = House.create([

{ name: "Gryffindor", founder: "Godric Gryffindor", colors: "Rojo y Dorado", animal: "León" },

{ name: "Slytherin", founder: "Salazar Slytherin", colors: "Verde y Plata", animal: "Serpiente" },

{ name: "Hufflepuff", founder: "Helga Hufflepuff", colors: "Amarillo y Negro", animal: "Tejón" },

{ name: "Ravenclaw", founder: "Rowena Ravenclaw", colors: "Azul y Plata", animal: "Águila" }

])

Esta parte crea registros para las cuatro casas de Hogwarts, cada una con su nombre, fundador, colores y animal representativo. Los registros se almacenan en la variable houses.

Personajes (Characters)

characters = Character.create([

{ name: "Harry Potter", house: houses[0], species: "Humano", patronus: "Ciervo", wand: "Holly" },

{ name: "Draco Malfoy", house: houses[1], species: "Humano", patronus: "Serpiente", wand: "Hawthorn" }

])

Aquí se crean registros para dos personajes, Harry Potter y Draco Malfoy, asociándolos a sus respectivas casas. Cada personaje tiene atributos como nombre, especie, patronus y varita.

Hechizos (Spells)

spells = Spell.create([

{ name: "Expelliarmus", spell\_type: "Defensivo", effect: "Desarma al oponente" },

{ name: "Lumos", spell\_type: "Encantamiento", effect: "Crea luz" }

])

Se crean dos hechizos, cada uno con un nombre, tipo y efecto.

Libros (Books)

books = Book.create([

{ title: "Harry Potter y la piedra filosofal", author: "J.K. Rowling", pdf\_url: "https://drive.google.com/uc?id=1sMU6YGp7wLLnlPCdwbtK3ypWMkFgHo-j" },

{ title: "Harry Potter y la cámara secreta", author: "J.K. Rowling", pdf\_url: "https://drive.google.com/uc?id=1sO76igvphb49O1yypJAqsIqgv3l1lkxM" },

{ title: "Harry Potter y el prisionero de Azkaban", author: "J.K. Rowling", pdf\_url: "https://drive.google.com/uc?id=1sTkOd1hKFvov0J\_5P7DUVlCixCDrQdwX" },

{ title: "Harry Potter y el cáliz de fuego", author: "J.K. Rowling", pdf\_url: "https://drive.google.com/uc?id=1simkBXjSeqXw-sy5Oo7LqGXg6HKYtDxK" },

{ title: "Harry Potter y la orden del fénix", author: "J.K. Rowling", pdf\_url: "https://drive.google.com/uc?id=1siz8cuyteqKozahPEwfb\_dS\_PCgMRDZA" },

{ title: "Harry Potter y el misterio del príncipe", author: "J.K. Rowling", pdf\_url: "https://drive.google.com/uc?id=1spTopTWRqzbFJQy3bHrgyAR9m4hAiG\_z" },

{ title: "Harry Potter y las reliquias de la muerte", author: "J.K. Rowling", pdf\_url: "https://drive.google.com/uc?

id=1ssIgeRGyVlbPKtmh2fNw2MWm\_p86NTcJ" }

])

Finalmente, se crean registros para varios libros de la saga Harry Potter, incluyendo su título, autor y una URL en PDF.





class CreateSpells < ActiveRecord::Migration[7.2]

La clase se llama CreateSpells, que sigue la convención de nombrar migraciones de acuerdo con la acción que realizan (en este caso, "crear hechizos").

Hereda de ActiveRecord::Migration[7.2], lo que indica que se utiliza la versión 7.2 de ActiveRecord.

**2. Método change**

def change

Este método contiene el código que se ejecutará para aplicar la migración. Rails sabe cómo revertirlo automáticamente si es necesario.

**3. Creación de la Tabla**

create\_table :spells do |t|

Aquí se crea la tabla spells. El bloque |t| permite definir las columnas de la tabla.

**4. Definición de las Columnas**

t.string :name

t.string :spell\_type # Cambia 'type' a 'spell\_type'

t.string :effect

t.string :name: Crea una columna llamada name que almacenará el nombre del hechizo.

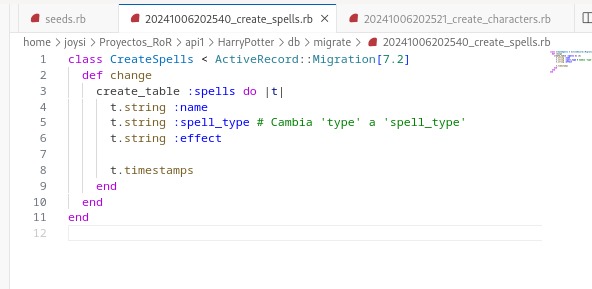
t.string :spell\_type: Crea una columna llamada spell\_type para almacenar el tipo de hechizo. Se ha elegido spell\_type en lugar de type para evitar conflictos con un método reservado en ActiveRecord.

t.string :effect: Crea una columna llamada effect que almacenará la descripción del efecto del hechizo.

**5. Timestamps**

t.timestamps

Esto agrega automáticamente dos columnas: created\_at y updated\_at, que Rails utiliza para rastrear cuándo se creó y se actualizó cada registro.



class CreateCharacters < ActiveRecord::Migration[7.2]

La clase se llama CreateCharacters, que indica que su propósito es crear una tabla de personajes. Hereda de ActiveRecord::Migration[7.2], lo que significa que está utilizando la versión 7.2 de ActiveRecord.

**2. Método change**

def change

Este método contiene el código que se ejecutará para aplicar la migración. Rails puede revertir automáticamente este cambio si es necesario.

**3. Creación de la Tabla**

create\_table :characters do |t|

Aquí se crea la tabla characters. El bloque |t| permite definir las columnas de la tabla.

**4. Definición de las Columnas**

t.string :name

t.references :house, foreign\_key: true

t.string :species

t.string :patronus

t.string :wand

t.string :name: Crea una columna llamada name que almacenará el nombre del personaje.

t.references :house, foreign\_key: true: Crea una columna llamada house\_id que almacenará la referencia a la casa a la que pertenece el personaje. foreign\_key: true asegura que haya una relación de clave foránea con la tabla houses, lo que garantiza la integridad referencial (es decir, que solo se puedan referenciar casas que existan).

t.string :species: Crea una columna llamada species que almacenará la especie del personaje (por ejemplo, humano, elfo, etc.).

t.string :patronus: Crea una columna llamada patronus que almacenará el patronus del personaje.

t.string :wand: Crea una columna llamada wand que almacenará la información sobre la varita del personaje.

**5. Timestamps**

t.timestamps

Esto agrega automáticamente dos columnas: created\_at y updated\_at, que Rails utiliza para rastrear cuándo se creó y se actualizó cada registro.



**validación con curl:**

**probando con método post:**

