Obligatorio 1 - Diseño de aplicaciones 2
Evidencia del diseño y especificación de la API.

Agustín Hernandorena (233361)
Joaquín Lamela (233375)

https://github.com/ORT-DA2/233375-233361Obl

API Rest

En particular primero debemos decir que es una API. Siendo así qué es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. Siendo así que una API permite que sus servicios se comuniquen con otros sin necesidad de saber cómo están implementados, otorgando flexibilidad, simplificación en el diseño, entre otras características.

Pero por otro lado, encontramos a REST. Siendo que es un estilo de arquitectura para la creación de sistemas distribuidos basados en hipermedia. REST es independiente de cualquier protocolo subyacente y no está necesariamente unido a HTTP. Sin embargo, en las implementaciones más comunes de REST se usa HTTP como protocolo de aplicación, y esta guía se centra en el diseño de API de REST para HTTP, tal como es el caso de nuestro sistema. REST se compone de una lista de principios que se deben cumplir en el diseño de la arquitectura de una API.

Principios REST

Pasaremos a analizar y verificar que nuestra API cumple con cada uno de los principios REST.

Interfaz uniforme

- Nuestra API está diseñada en torno a recursos, que son cualquier tipo de objeto, dato o servicio al que puede acceder el cliente. Por ejemplo, tenemos el recurso User (Id, Nombre, Apellido, Mail, Nombre de usuario, contraseña).
- En nuestra API, un recurso tiene un identificador, que es un URI que identifica de forma única ese recurso.
- Los clientes interactúan con un servicio mediante el intercambio de representaciones de recursos. El servidor envía los datos via JSON pero el mecanismo de almacenamiento interior (una base de datos en nuestro caso) para el cliente es transparente.
- La representación del recurso que le llega al cliente, es la suficiente para que el mismo pueda crearlo, modificarlo y borrarlo, esto suponiendo que para estas acciones el mismo tenga permisos. Esto debido a que incluye el uso de verbos HTTP, cómo GET, POST, PUT y DELETE.
- Se utilizan mensajes descriptivos por medio de las características del protocolo HTTP, las cuales son verbos y códigos de estado.
- Para obtener una API jerárquica y con ciertas reglas, procuramos el uso de nombres en **plural.**

Peticiones sin estado

Las API REST usan un modelo de solicitud sin estado. Las solicitudes HTTP deben ser independientes y pueden producirse en cualquier orden, por lo que no es factible conservar la información de estado transitoria entre solicitudes, es por esto que decimos que todas las consultas tienen la información necesaria para entenderla y no se pueden utilizar datos almacenados. El estado no guardará información de las últimas consultas y tratará a cada una como una nueva.

Por ejemplo tenemos que:

GET api/users/12345
DELETE api/users/12345
En la segunda petición hemos tenido que indicar el identificador del recurso que queremos borrar.
El servidor no guarda los datos de la consulta previa que tenía el cliente en particular. Una petición del tipo DELETE api/users debe dar error, ya que falta el identificador y el servidor almacena los datos.

Cacheable

Cuando se manda una request se puede establecer que se cachee o no, en caso de que se cachee, se le da la posibilidad al cliente de usar la respuesta después.

En nuestra API decidimos no cachear las respuestas del servidor, sabiendo de que no será muy alto el número de solicitudes hacia un determinado recurso, y por lo tanto, no tendremos problemas de rendimiento en ese sentido.

Separación de cliente y servidor

Esto se refiere a que para que REST exista deben haber dos actores, un cliente que consuma y un servidor que almacene o genere información.

En ese sentido, nuestra API sigue los siguientes criterios:

- El cliente y servidor están separados, su unión es mediante la interfaz uniforme.
- Mientras la interfaz no cambie, podremos cambiar el cliente o el servidor sin problemas.

Sistema dividido en capas

Capas distribuidas jerárquicamente en la que se restringe el comportamiento de forma que disminuya la visibilidad directa entre componentes. Además, permite que se dividan las capas en distintos servidores.

En ese sentido, nuestra API sigue los siguientes criterios:

- Un cliente no sabe inicialmente (ni debería interesarle) si está conectada directamente con el servidor o si hay componentes intermedios que incrementan la seguridad o distribuyen la carga entre los servidores de la aplicación.
- El uso de capas intermedias, en un futuro servirá para aumentar la escalabilidad o para implementar nuevas políticas de seguridad.

Mecanismos de autenticación de request

Lo que sucede es que dentro de nuestro sistema encontramos, hay diferentes acciones las cuales únicamente pueden ser realizadas por los administradores. Estas son por ejemplo: creación de una categoría, creación de un punto turístico, creación de un hospedaje, modificación de un hospedaje, eliminación de un hospedaje, actualización de una reserva, entre otras. Pero para poder realizar estas consultas debe haber un mecanismo de autenticación para que no cualquier usuario de la API pueda hacer request a los diferentes endpoints, pudiendo así hacer lo que él desee.

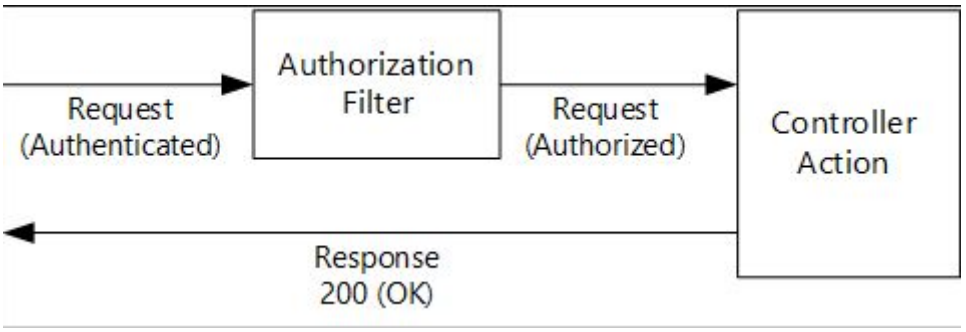
Es por eso que encontramos que uno de los mecanismos de autenticación de los request los cuales son requeridos y realizados únicamente por administradores del sistema, utilizan la implementación de un filtro de autorización. Siendo así que la autenticación y autorización son las bases de la seguridad en aplicaciones. La autenticación establece la identidad de un usuario validando las credenciales, mientras que la autorización determina si un usuario está autorizado para realizar una acción solicitada. Siendo que sí nuestra API cumple con dichas características, se trata de una API protegida, es decir que autentica solicitudes y autoriza el acceso al recurso solicitada en función de la identidad establecida.

Siendo así que los filtros permiten que se ejecute el código antes o después de determinadas fases de la canalización del procesamiento de la solicitud. De tal forma que para el desarrollo de nuestra API se utilizó un filtro de autorización qué tal lo dice la palabra es de autorización. Se ejecutan en primer lugar y sirven para averiguar si el usuario está autorizado para realizar la solicitud. Los filtros de autorización pueden cortocircuitar la canalización si una solicitud no está autorizada, es decir, permiten evitar llegar al controller en caso de no cumplir con las políticas de seguridad, que es lo que se busca cuando un usuario no está autorizado a realizar alguna de las acciones mencionadas anteriormente.

Las operaciones que nos permiten autorizar o denegar una solicitud están comprendidas en la interfaz IAuthorizationFilter. Para implementar el filtro, entonces, creamos una clase que implemente dicha interfaz y el método OnAuthorization (AuthorizationFilterContext context) en donde se implementa la lógica de autorización. Dentro de esta clase hacemos uso de una clase llamada *UserManagement* que se encarga de manejar la lógica relativa a las sesiones de los usuarios.

Dentro del método mencionado, lo que hacemos es verificar que en el header de la solicitud esté contenido un identificador (token), y que el mismo se encuentre en la tabla donde se encuentran los usuarios que se encuentran logueados actualmente.

Lo que sucede es que en los casos que no se cumple con la política de seguridad se le asigna un ContentResult al resultado de la solicitud, y automáticamente se responde la solicitud no llegando de esta manera al método del Controller.



Descripción general de códigos de error

De tal manera que primero debemos introducirnos acerca de que es un código de estado HTTP. De forma tal que un código de estado HTTP es un mensaje que devuelve el servidor cada vez que el navegador realiza una petición al servidor. Si el servidor es capaz de devolver el contenido que solicita el navegador y no existe ningún error, estos códigos HTTP no son visibles para el usuario. En cambio, si algo va mal, el servidor devuelve un código de estado HTTP que indica que algo no salió como esperaba con un mensaje de error incluido en la implementación de nuestra Web Api, de forma tal que es entendible para él usuario de aplicación entender qué fue lo que ha salido mal.

De tal manera que en función del código de estado que devuelve el servidor, el usuario de la Web Api, además del mensaje de error, observando el código de estado HTTP se podrá hacer una idea del tipo de error que se ha producido. De forma que es un buen principio utilizarlos cuando se tiene una API, ya que es lo que da noción de qué situación el usuario se encuentra cuando realiza peticiones a la misma.

Es por eso que encontramos que los códigos de estado los podemos dividir en cinco categorías diferentes, de manera que visualizando únicamente el permitir dígito del código de estado le permite identificar al usuario de la API, al tipo de respuesta que la misma le está dando. Siendo así que las diferentes categorías son:

- Código de estado 1xx: Se trata de respuestas de carácter informativo dónde él servidor le notifica al cliente que la petición actual continua.

- Código de estado 2xx: Se trata de respuestas qué son satisfactorias. Indica que la acción solicitada por el cliente ha sido recibida, entendida, aceptada y procesada correctamente.

- Código de estado 3xx: Se trata de códigos los cuales hacen referencia a que la solicitud ha sido recibida. Sin embargo, para asegurar un procesamiento exitoso es necesario que el cliente tome una acción adicional cómo, por ejemplo, una redirección.

- Código de estado 4xx: Estos códigos de estado hacen referencia a errores a que se ha presentado un error de cliente. Esto quiere decir que se ha recibido la solicitud, pero esta no se puede llevar a cabo.

- Código de estado 5xx: Estos códigos HTTP también muestran errores, pero por el lado del servidor.

De tal forma que existen más de 70 códigos de estado HTTP, esto da la idea de que todos los códigos existentes se deberían utilizar lo cual no es cierto, ya que la mayoría de los desarrolladores no conocen a todos, de tal manera que si se utilizan códigos de estado poco conocidos, se está guiando a qué se consuma nuestra API, pero luego se redirigen a otra pagina a buscar cual es el significado del código de estado qué les está llegando.

De tal manera que siguiendo con las buenas prácticas en el desarrollo de la API, decidimos utilizar no mucha cantidad de códigos de estado, y tampoco códigos pocos conocidos.

Entonces, en resumen, todo lo anterior nos conlleva a preguntar, ¿cuántos códigos de estado debe usar en la API?
Y la respuesta que encontramos a esto fue lo que se implementó de manera satisfactoria en nuestra API. Se podrían implementar con los códigos más utilizados (200, 400 y 500), y a partir de allí extendernos dependiendo de las necesidades de la aplicación. Pero en particular la API no debería tener más de ocho códigos de error. Esto debido a que una buena práctica, es utilizar pocos y conocidos códigos de estado HTTP.

Siendo así que los códigos de estados HTTP qué se utilizan y devuelven en nuestra api son:

Código de estado	**Mensaje de estado**	**Descripción**
200	Ok	La solicitud ha tenido éxito.
201	Creado	La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello. Ésta es típicamente la respuesta enviada después de una petición PUT
204	Sin contenido	La petición se ha completado con éxito pero su respuesta no tiene ningún contenido, aunque los encabezados pueden ser útiles
400	solicitud incorrecta	No se puede procesar la solicitud porque tiene un formato incorrecto o es incorrecta.
401	Sin autorizar	La información de autenticación necesaria falta o no es válida para el recurso.

403	prohibido	Se denegó el acceso al recurso solicitado. Es posible que el usuario no tenga permisos suficientes
404	No encontrado	El recurso solicitado no existe.
500	error interno del servidor	Se produjo un error interno del servidor al procesar la solicitud.

Ahora, pasaremos a explicar en qué casos usamos cada código de error y con qué finalidad.

200 OK: Este código generalmente lo usamos en los casos en que damos respuesta a solicitudes HTTP GET exitosas, para indicarle al cliente que la solicitud que realizó ha tenido éxito.

201 CREATED: Este código lo usamos cuando se realiza una petición HTTP POST o HTTP PUT, y se crea o modifica un nuevo recurso de forma exitosa, en ese sentido, consideramos apropiado devolver este código que indica que un nuevo recurso ha sido creado.

204 NO CONTENT: Cuando se realiza una petición HTTP DELETE, y la petición se ha completado con éxito pero su respuesta no tiene ningún contenido (debido a que el objeto se ha eliminado).

400 BAD REQUEST: Generalmente, este código de error lo utilizamos cuando se realiza una solicitud HTTP POST o HTTP PUT con alguno de los campos requeridos de forma inválida, en este sentido, nos parece apropiado indicar que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.

401 UNAUTHORIZED: Este código de error lo usamos para impedir que un usuario que no se encuentre logueado pueda hacer uso de las funcionalidades que requieren autenticación.

403 FORBIDDEN: Este código de error lo utilizamos cuando verificamos que un usuario esté logueado para que pueda realizar una determinada funcionalidad que requiere autenticación, y se tiene que el token recibido por header no es válido.

404 NOT FOUND: Este código de error lo utilizamos cuando se realiza una petición HTTP GET y no se encuentra el recurso solicitado, así como también en los casos en que se realiza un HTTP POST o HTTP PUT para crear o modificar un recurso que tiene asociado otro recurso, y no es posible encontrar ese recurso.

500 INTERNAL SERVER ERROR: Este código de error es utilizado cuando ocurre un problema interno, como es el caso de un problema en la base de datos.

Descripción de los resources de la API

Para observar de forma resumida las operaciones disponibles sobre cada uno de los recursos, y el resultado que se espera obtener, ir a sección 1 de Anexo.

Región controller

GET	/api/regions
Resource	Regions
Description	Devuelve todas las regiones existentes en el sistema.
Parameters	No recibe parametros.
Responses	A) **200:** Ocurre cuando la peticion es exitosa y se devuelven todas las regiones existentes. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no hay regiones existentes. **C) 500:** Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

GET	/api/regions/{id}
Resource	Regions
Description	Realiza la busqueda de una region en el sistema a partir del id ingresado, y en caso de que exista la retorna.
Parameters	Recibe el id de la region a buscar.
Responses	A) **200:** Ocurre cuando la peticion es exitosa y se devuelve la region que posee el id parametro. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no existe una region con el id buscado. **C) 500:** Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

Category controller

GET	/api/categories/
Resource	Categories
Description	Devuelve todas las categorias existentes en el sistema.
Parameters	No recibe parametros.
Responses	A) **200:** Ocurre cuando la peticion es exitosa y se devuelven todas las categorias existentes. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no hay categorias existentes. **C) 500:** Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

GET	/api/categories/{id}
Resource	Categories
Description	Realiza la busqueda de una categoria en el sistema a partir del id ingresado, y en caso de que exista la retorna.
Parameters	Recibe el id de la categoria a buscar.
Responses	A) **200**: Ocurre cuando la peticion es exitosa y se devuelve la categoria que posee el id parametro. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no existe una categoria con el id buscado. **C) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

POST	/api/categories
Resource	Categories
Description	Crea y retorna una nueva categoria con la informacion contenida en el body (el cual es obligatorio). Esto lo realiza siempre y cuando haya un administrador logueado en el sistema y los campos sean correctos.
Parameters	No recibe parametros.

Body	{
	"name": "string"
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se crea la categoria con la informacion contenida en el body. Una vez creada, la misma se retorna. B) **400:** Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. C) 401: Ocurre cuando se intenta realizar la operacion sin estar logueado. D**) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

Tourist Spot controller

GET	/api/touristSpots
Resource	Tourist Spots
Description	Devuelve todos los puntos turisticos existentes en el sistema.
Parameters	No recibe parametros.
Responses	A) **200**: Ocurre cuando la peticion es exitosa y se devuelven todos los puntos turisticos B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no hay puntos turisticos existentes. **C) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

GET	/api/touristSpots/{id}
Resource	Tourist Spots
Description	Realiza la busqueda de un punto turistico en el sistema a partir del id ingresado, y en caso de que exista la retorna.
Parameters	Recibe el id del punto turistico a buscar.
Responses	A) **200:** Ocurre cuando la peticion es exitosa y se devuelve el punto turistico que posee el id parametro. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no existe un punto turistico con el id buscado. **C) 500:** Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

POST	/api/touristSpots
Resource	Tourist Spots
Description	Crea y retorna un nuevo punto turistico con la informacion contenida en el body (el cual es obligatorio). Esto se realiza siempre y cuando haya un administrador logueado en el sistema y los campos sean correctos.
Parameters	No recibe parametros.

Body	{
	"name": "string",
	"description": "string",
	"regionId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
	"imagePath": "string",
	"listOfCategoriesId": ["3fa85f64-5717-4562-b3fc-2c963f66afa6"]
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se crea el punto turistico con la informacion contenida en el body. Una vez creado, el mismo se retorna. B) **400:** Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. C) **401:** Ocurre cuando se intenta realizar la operacion sin estar loguedo. D) **404:** Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existe la region o las categorias asociadas. c**) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

GET	/api/touristSpots/getByRegion/
Resource	Tourist Spots
Description	Realiza la busqueda de los puntos turisticos en el sistema a partir del id de la region ingresado, y en caso de que existan los retorna.
Parameters	Recibe el id de la region por la cual se quieren obtener los puntos turisticos.
Responses	A) **200:** Ocurre cuando la peticion es exitosa y se devuelven los puntos turisticos que se encuentran en la region buscada. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no existe un punto turistico para la region buscada o que no exista la region pasada por parametro. **C) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

GET	/api/touristSpots/getByCategories/
Resource	Tourist Spots
Description	Realiza la busqueda de los puntos turisticos en el sistema a partir de los ids de las categorias ingresadas, y en caso de que existan los retorna.
Parameters	Recibe una lista de ids de las categorias por las cuales se quieren obtener los puntos turisticos que las tienen.
Responses	A) **200**: Ocurre cuando la peticion es exitosa y se devuelven los puntos turisticos que tienen las categorias buscadas. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no existen puntos turisticos para las categorias buscadas o que no exista alguna de las categorias pasadas por parametro. **C) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

GET	/api/touristSpots/getByCategoriesAndRegion/
Resource	Tourist Spots
Description	Realiza la busqueda de los puntos turisticos en el sistema a partir de los ids de las categorias y region ingresadas y en caso de que existan los retorna.
Parameters	Recibe una lista de ids de las categorias y un id de la region por las cuales se quieren obtener los puntos turisticos que los tienen.
Responses	A) **200**: Ocurre cuando la peticion es exitosa y se devuelven los puntos turisticos que tienen las categorias y region buscadas. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no existen puntos turisticos para las categorias y region buscadas, o que no exista alguna de las categorias o la region pasadas por parametro **C) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

Lodging controller

GET	/api/lodgings
Resource	Lodgings
Description	Devuelve todos los hospedajes existentes en el sistema.
Parameters	No recibe parametros.
Responses	A) **200**: Ocurre cuando la peticion es exitosa y se devuelven todos los hospedajes. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no hay hospedajes existentes. **C) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

GET	/api/lodgings/{id}
Resource	Lodgings
Description	Realiza la busqueda de un hospedaje en el sistema a partir del id ingresado, y en caso de que exista lo retorna.
Parameters	Recibe el id del hospedaje a buscar.
Responses	A) **200:** Ocurre cuando la peticion es exitosa y se devuelve el hospedaje que posee el id parametro. B) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no existe un hospedaje con el id buscado. **C) 500:** Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

POST	/api/lodgings
Resource	Lodgings
Description	Crea y retorna un nuevo hospedaje con la informacion contenida en el body (el cual es obligatorio). Esto se realiza siempre y cuando haya un administrador logueado en el sistema y los campos sean correctos.
Parameters	No recibe parametros.

Body	{
	"name": "string",
	"description": "string",
	"quantityOfStars": 0,
	"address": "string",
	"images": ["string"],
	"pricePerNight": 0,
	"isAvailable": true,
	"touristSpotId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se crea el hospedaje con la informacion contenida en el body. Una vez creado, el mismo se retorna. B) **400:** Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. C) **401:** Ocurre cuando se intenta realizar la operacion sin estar logueado. D) **404:** Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existe el punto turistico asociado c**) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

DELETE	/api/lodgings/{id}
Resource	Lodgings
Description	Elimina un hospedaje cuyo id coincide con el pasado por parametro. Esto lo realiza siempre y cuando haya un administrador logueado en el sistema.
Parameters	Recibe el id del hospedaje a eliminar.
Responses	A) **204**: Ocurre cuando la peticion es exitosa y se ha elminado correctamente el hospedaje. No muestra contenido. B) **401:** Ocurre cuando se intenta realizar la operacion sin estar logueado. C) **404:** Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existe el hospedaje a eliminar D**) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

PUT	/api/lodgings/{id}
Resource	Lodgings
Description	Modifica y retorna el hospedaje cuyo id coincide con el parametro, actualizando sus datos con la informacion contenida en el body (el cual es obligatorio). Esto se realiza siempre y cuando haya un administrador logueado y los campos sean correctos.
Parameters	Recibe el id del hospedaje a actualizar.

Body	{
	"name": "string",
	"description": "string",
	"quantityOfStars": 0,
	"address": "string",
	"images": ["string"],
	"pricePerNight": 0,
	"isAvailable": true,
	"touristSpotId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se actualiza el hospedaje con la informacion contenida en el body. Una vez actualizado, el mismo se retorna. B) **400:** Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. C) **401:** Ocurre cuando se intenta realizar la operacion sin estar logueado. D) **404:** Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existe el hospedaje que se quiere actualizar. E) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

Search of lodgings controller

POST	/api/searchOfLodgings
Resource	Search of lodgings
Description	Crea y retorna el resultado de realizar una nueva busqueda de hoespedajes con la informacion contenida en el body (el cual es obligatorio). Esto se realiza siempre y cuando los campos sean correctos.
Parameters	No recibe parametros.

Body	{
	"checkIn": "2020-10-15T02:50:33.028Z",
	"checkOut": "2020-10-15T02:50:33.028Z",
	"quantityOfAdult": 0,
	"quantityOfChilds": 0,
	"quantityOfBabies": 0,
	"touristSpotIdSearch": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se retornan los hospedajes disponibles de acuerdo a la informacion de la busqueda. B) **400:** Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. C) **404:** Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existen hospedajes para la busqueda o no existe el punto turistico asociado a la busqueda. c) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

Reserve controller

POST	/api/reserves
Resource	Reserves
Description	Crea y retorna una nueva reserva con la informacion contenida en el body (el cual es obligatorio). Esto se realiza siempre y cuando los campos sean correctos.
Parameters	No recibe parametros.

Body	{
	"name": "string",
	"lastName": "string",
	"email": "string",
	"checkIn": "2020-10-15T02:57:18.772Z",
	"checkOut": "2020-10-15T02:57:18.772Z",
	"quantityOfAdult": 0,
	"quantityOfChild": 0,
	"quantityOfBaby": 0,
	"idOfLodgingToReserve": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se crea la reserva con la informacion contenida en el body. Una vez creada, la misma se retorna. B) **400**: Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. C) **404**: Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existe el hospedaje a reservar. D) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

GET	/api/reserves/{id}
Resource	Reserves
Description	Realiza la busqueda de una reserva en el sistema a partir del id ingresado, y en caso de que exista la retorna.
Parameters	Recibe el id de la reserva a buscar.
Responses	A) **200**: Ocurre cuando la peticion es exitosa y se devuelve la reserva que posee el id parametro. B) **404**: Ocurre cuando no existe el recurso solicitado, es decir, no existe una reserva con el id buscado. **C) 500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

PUT	/api/reserves/{idForUpdateReserve}
Resource	Reserves
Description	Modifica y retorna la reserva cuyo id coinicde con el parametro, actualizando sus datos con la informacion contenida en el body (el cual es obligatorio). Esto se realiza siempre y cuando haya un administrador logueado y los campos sean correctos.
Parameters	Recibe el id de la reserva a actualizar.

Body	{
	"description": "string",
	"stateOfReserve": 0
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se actualiza la reserva con la informacion contenida en el body. Una vez actualizada, la misma se retorna. B) **400:** Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. C) 401: Ocurre cuando se intenta realizar la operacion sin estar logueado. D) **404:** Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existe la reserva que se quiere actualizar. E) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

User controller

GET	/api/users
Resource	Users
Description	Devuelve todos los usuarios existentes en el sistema, siempre y cuando se este logueado.
Parameters	No recibe parametros.
Responses	A) **200**: Ocurre cuando la peticion es exitosa y se devuelven todos los usuarios. B) **401:** Ocurre cuando se intenta realizar la operacion sin estar logueado. **C) 404:** Ocurre cuando no existe el recurso solicitado, es decir, no hay usuarios existentes. D) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

GET	/api/users/{id}
Resource	Users
Description	Realiza la busqueda de un usuario en el sistema a partir del id ingresado, y en caso de que exista lo retorna siempre y cuando el solicitante este logueado.
Parameters	Recibe el id del usuario a buscar.
Responses	A) **200**: Ocurre cuando la peticion es exitosa y se devuelve el usuario que posee el id parametro. B) **401:** Ocurre cuando se intenta realizar la operacion sin estar logueado. C) **404:** Ocurre cuando no existe el recurso solicitado, es decir, no existe un usuario con el id buscado. D) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

POST	/api/users
Resource	Users
Description	Crea y retorna una nuevo usuario con la informacion contenida en el body (el cual es obligatorio). Esto se realiza siempre y cuando los campos sean correctos y el solicitante este logueado.
Parameters	No recibe parametros.

Body	{
	"name": "string",
	"lastName": "string",
	"userName": "string",
	"password": "string",
	"mail": "string"
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se crea el usuario con la informacion contenida en el body. Una vez creada, el mismo se retorna. B) **400:** Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. C) **401:** Ocurre cuando se intenta realizar la operacion sin estar logueado. D) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

DELETE	/api/users/{id}
Resource	Users
Description	Elimina un usuario cuyo id coincide con el pasado por parametro. Esto lo realiza siempre y cuando haya un administrador logueado en el sistema.
Parameters	Recibe el id del usuario a eliminar.
Responses	A) **204**: Ocurre cuando la peticion es exitosa y se ha elminado correctamente el usuario. No muestra contenido. B) **401:** Ocurre cuando se intenta realizar la operacion sin estar logueado. C) **404:** Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existe el usuario a eliminar D) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

POST	/api/users/login
Resource	Users
Description	Crea y retorna una nueva sesion de usuario con la informacion del usuario a loguear contenida en el body (el cual es obligatorio). Esto se realiza siempre y cuando los campos sean correctos.
Parameters	No recibe parametros.

Body	{
	"email": "string",
	"password": "string"
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se crea la sesion de usuario con la informacion contenida en el body. Una vez creada, la misma se retorna. B) **400:** Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. c) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	No recibe ningun header adicional a los por defecto.

PUT	/api/users/{id}
Resource	Users
Description	Modifica y retorna el usuario cuyo id coincide con el parametro, actualizando sus datos con la informacion contenida en el body (el cual es obligatorio). Esto se realiza siempre y cuando haya un administrador logueado y los campos sean correctos.
Parameters	Recibe el id del usuario a actualizar.

Body	{
	"id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
	"name": "string",
	"lastName": "string",
	"userName": "string",
	"password": "string",
	"mail": "string"
	}

Responses	A) **201**: Ocurre cuando la peticion es exitosa y se actualiza el usuario con la informacion contenida en el body. Una vez actualizado, el mismo se retorna. B) **400:** Ocurre cuando el servidor no puede procesar la peticion debido a algo que es percibido como un error del cliente, y muestra un mensaje indicando el error ocurrido. C) **401:** Ocurre cuando se intenta realizar la operacion sin estar logueado. D) **404:** Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existe el usuario que se quiere actualizar. E) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

DELETE	/api/users/logout
Resource	Users
Description	Elimina la sesion de usuario asociada al usuario que actualmente se encuentra loguedo y realiza la peticion.
Parameters	No recibe ningun parametro.
Responses	A) **200**: Ocurre cuando la peticion es exitosa y se ha elminado correctamente la sesion de usuario. Se muestra un mensaje indicando que se ha cerrado correctamente la sesion actual. B) **401:** Ocurre cuando se intenta realizar la operacion sin estar logueado. C) **404:** Ocurre cuando no existe el recurso solicitado, en este caso, cuando no existe el usuario a eliminar D) **500**: Ocurre cuando sucede un problema interno en el servidor.
Headers	Recibe un header adicional a los por defecto con el token de la sesion del administrador que se encuentra logueado y desea realizar la operación.

Anexo

Sección 1

Descripción resumida de cada una de las operaciones.
Para realizar las operaciones contenidas en las tablas que poseen un (*) es necesario estar logueado cómo administrador en el sistema. Sino de otra forma, no podrá realizarse esta operación debido a que saltara un código de estado 401, con un mensaje de error acerca de qué no se está logueado y no tiene autorización para realizar la operación si no se está logueado.

Regions controller

Resource	Get	Post	Put	Delete
/api/regions	Devuelve todas las regiones existentes	-	-	-
/api/regions/{id}	Devuelve la región identificada con el id buscado	-	-	-

Category controller

Resource	Get	Post	Put	Delete
/api/categories	Devuelve todas las categorías existentes	Crea una nueva categoría y la devuelve en caso de crearse (*)	-	-
/api/categories/{id}	Devuelve la categoría identificada con el id buscado	-	-	-

Tourist Spot controller

Resource	Get	Post	Put	Delete
/api/touristSpots	Devuelve todos los puntos turísticos existentes	Crea un nuevo punto turístico y lo devuelve en caso de crearse (*)	-	-

/api/touristSpots/{id}	Devuelve el punto turístico identificado con el id buscado	-	-	-
/api/touristSpots/getByRegion	Devuelve los puntos turísticos que se ubican en la región indicada en el parámetro.	-	-	-
/api/touristSpots/getByCategories	Devuelve los puntos turísticos que tienen las categorías indicadas por parámetro.	-	-	-
/api/touristSpots/getByCategoriesAndRegion	Devuelve los puntos turísticos que tienen la región y las categorías indicadas por parámetro.	-	-	-

Lodging controller

Resource	Get	Post	Put	Delete
/api/lodgings	Devuelve todos los hospedajes existentes.	Crea un nuevo hospedaje y lo retorna en caso de crearse (*)	-	-
/api/lodgings/{id}	Devuelve el hospedaje identificado con el id buscado.	-	Modifica el hospedaje identificado con el id seleccionado y lo retorna (*)	Elimina el hospedaje identificado con el id. (*)

SearchOfLodging controller

Resource	Get	Post	Put	Delete
/api/searchOfLodgings	-	Crea una búsqueda con los datos pasados, obteniendo cómo respuesta todos los hospedajes que cumplan.	-	-

Reserve controller

Resource	Get	Post	Put	Delete
/api/reserves	-	Crea una reserva con los datos pasados y la retorna en caso de crearla	-	-
/api/reserves/{id}	Devuelve la reserva identificada con el id buscado.	-	-	-
/api/reserves/{idForUpdateReserve}	-	-	Actualiza la reserva que tiene el id buscado con la información contenida en el body. (*)	-

User controller

Resource	Get	Post	Put	Delete
/api/users	Devuelve todos los usuarios existentes (*)	Crea un nuevo usuario (*)	-	-
/api/users/{id}	Devuelve el usuario identificado con el id buscado. (*)	-	Modifica el usuario identificado con el id seleccionado (*)	Elimina el usuario identificado con el id. (*)
/api/users/login	-	Inicia sesión el usuario identificado con los datos ingresados.	-	-
/api/users/logout	-	-	-	Cierra la sesión el usuario, ingresando el token de sesión. (*)