IIC3103 - Taller de Integración
Departamento Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica

Enunciado Tarea 1

Objetivo

El objetivo de esta tarea es contruir una API a partir de una especificación técnica, y desplegar este servicio públicamente a través de internet.

Trabajo a realizar

Dos grandes hoteleros que querían construir el hotel más grande del mundo se reunieron a dialogar sobre el asunto y comenzaron por el primer y más obvio tema a discutir: cuántas habitaciones tendría.

- ¿Qué te parece si construimos un hotel con 1000 habitaciones?
- No, porque si alguien construyera uno de 2000 habitaciones, nuestro hotel ya no sería tan grande. Mejor hagámoslo de 10 000.
- Pero podría ser que alguien construyera uno de 20 000 y volveríamos a quedarnos con un hotel pequeño. Construyamos un hotel con 1 000 000 de habitaciones, ése sería un hotel grande.
- Y qué tal si alguien construyera uno con...

Como siempre podría llegar a haber un hotel más grande, llegaron a la conclusión de que era necesario hacer un hotel con habitaciones infinitas de manera que ningún otro hotel del mundo pudiera superar su tamaño.

Luego de comenzar la construcción del hotel, originalmente de un solo piso y con infinitas piezas horizontalmente, se dieron cuenta de que necesitaban más espacio, por lo que comenzaron a construir verticalmente una cantidad infinita de pisos, cada uno con infinitas habitaciones.

Sin embargo, cuando estaban a punto de terminar la construcción del hotel, se dieron cuenta de un grave problema: Habían olvidado construir los ascensores del hotel.

Es ahí cuando los dueños del hotel se pusieron en contacto con los alumnos del ramo **IIC3103 Taller de Integración**, para ayudarlos a solucionar este problema, y habilitar un sistema de infinitos ascensores para el edificio.

Además, los dueños del hotel les comentan que los calculistas del proyecto no habían calculado correctamente los costos de construir el hotel infinito, por lo que les queda poco presupuesto y solo podrán pagar por un sistema de ascensores de capacidad finita.

Índice

Descripción general de la API

Comportamiento

- La API debe ser capaz de crear tantos ascensores como sean necesarios para cumplir con el transporte de los pasajeros a sus habitaciones.
- Cada ascensor tendrá una capacidad (en kilogramos) limitada, definida al momento de creación del ascensor.
- Cada pasajero será identificado por un id único, y tendrá asociado un peso en kilogramos conocido.
- El ascensor no puede permitir que se supere el límite de peso. Es decir, no permitirá la entrada de otro pasajero en caso que su peso actual más el peso del pasajero supere el péso máximo del ascensor.
- Los ascensores deben ir al piso más cercano donde se haya solicitado, siempre y cuando este siga con la dirección y sentido actual que esté siguiendo el ascensor en ese momento.
- Los ascensores solo deben parar en los pisos en que hayan solicitado subirse o bajarse del ascensor.
- Por temas de seguridad, un ascensor debe moverse con las puertas cerradas.
- Se asume que los pisos desde el - ∞ al ∞ ya existen.

Estados de los usuarios

Los usuarios pueden encontrarse en uno de los siguientes cuatro estados:

1. **Esperando afuera del ascensor:** El usuario está en un piso, pero aún no ha interactuado con el ascensor. Simplemente espera la oportunidad de solicitarlo.
2. **Solicitado afuera del ascensor:** El usuario ha presionado el botón de llamada desde el piso en el que se encuentra, solicitando que el ascensor se detenga en ese piso para recogerlo.
3. **Esperando adentro del ascensor:** El usuario ya ha ingresado al ascensor, pero aún no ha indicado su destino. En este estado, el usuario está dentro del ascensor, esperando para seleccionar el piso al que desea ir.
4. **Solicitado adentro del ascensor:** El usuario, ya dentro del ascensor, ha presionado el botón correspondiente al piso al que quiere ir. En este estado, el ascensor tiene una solicitud de destino que debe cumplir.

Los usuarios siempre deberían cambiar de estado en este orden: 1 -> 2 -> 3 -> 4 -> 1 ...

Ejemplo con 2 usuarios

Si el ascensor estuviera en el piso 2, primero lo pide una persona del piso 4, y luego una persona en el piso 1, sería incorrecto que el ascensor se mueva al piso 1 (aunque sea más cercano) ya que la "ruta" que debería seguir (cronológicamente) es ir al piso 4, y si fuera al piso 1 estaría yendo en sentido contrario.

Luego de haber recogido al pasajero en el piso 4, si su destino era el piso 8, lamentablemente va a tener que bajar hasta el piso 1 a buscar al primer pasajero (ya que esta solicitud estaba "programada" desde antes que él "se suba al ascensor y apriete el botón para ir al piso 8"), asumiendo que el primer pasajero no se subió a ningún otro ascensor.

En cambio, si su destino era el piso 3, estaría bien que pasara por el 3 antes de ir al 1 ya que es el más cercano y mantiene la dirección "programada" (ir del 4 al 1).

Si el primer pasajero se hubiera subido a otro ascensor, sería correcto que el ascensor fuera del piso 4 al piso 8 de inmediato.

Descripción detallada de la API

Interfaces

Los modelos deben tener, al menos, los siguientes campos:

Elevator

Cada ascensor estará definido según la siguiente interfaz:

```
{
        "id": string,
        "level": int,
        "max_weight": int,
        "doors": string, ("open" | "closed")
        "users": List[User]
}
```

Level

Cada piso estará definido según la siguiente interfaz:

```
{
        "id": int,
        "users": List[User]
}
```

User

Cada usuario estará definido según la siguiente interfaz:

```
{
        "id": string,
        "destination": int,
        "weight": int,
        "status": string, ("waiting" | "requested")
}
```

Error

Todos los errores dentro de la API deben seguir el siguiente formato:

```
{
        "error": string
}
```

Es importante considerar que el texto del error en sí no se corregirá (no se hará la comparación literal de los mensajes), basta con que se levanten los errores y sus estados de error correctos en los casos que corresponda.

Rutas, Solicitudes y Respuestas

Las rutas de su API deben seguir al pie de la letra[1] las rutas, métodos HTTP, cuerpos de solicitud y de respuesta que se detallan a continuación.

Resetear estado

Este servicio permite resetear el estado de la API (eliminar todos los ascensores y usuarios). Será usado por el corrector automático para poder partir los tests desde un estado "limpio".

Método HTTP y Ruta	POST /reset	status
Request Body	{}	
Response Body	{"message": "ok"}	200

[1] Es importante que estas se sigan al pie de la letra ya que la corrección de esta tarea será automatizada.

Crear ascensor

Este servicio permite crear un ascensor, indicando su peso máximo. Al crear el ascensor, el servicio retorna su identificador. Todos los ascensores comienzan vacios y ubicados en el primer piso del hotel, con las puertas cerradas.

Método HTTP y Ruta	POST /elevators	status

Request Body	{
	"max_weight": int
	}	

Response Body	Elevator	201 Created
Errors	Error("missing parameter: max_weight")	400 Bad request
En caso de ingresar algo que no sea un entero.	Error("invalid max_weight <max_weight>")	400 Bad request

Obtener ascensores

Método HTTP y Ruta	GET /elevators	status
Response Body	List[Elevator]	200 Ok
Errors	Este servicio nunca debería fallar.	

Obtener ascensor

Método HTTP y Ruta	GET /elevators/:elevator_id	status
Path Parameters	elevator_id: identificador único del ascensor	
Response Body	Elevator	200 Ok
Errors	Error("elevator with id <id> not found")	404 Not found

Mover ascensor

Método HTTP y Ruta	PATCH /elevators/:elevator_id	status
Request Body	{	

	"level": int }	
Response Body	Elevator	200 Ok
Errors	Error("elevator with id <id> not found")	404 Not found
	Error("missing parameter: level")	400 Bad request
Los valores posibles para **level** son de - ∞ a ∞ [2]	Error("invalid level <level>")	400 Bad request
El ascensor tiene las puertas abiertas	Error("elevator doors are open")	400 Bad request
El ascensor no debería haberse movido a ese piso	Error("elevator should not move to level X")	400 Bad request

Abrir / cerrar puertas del ascensor

Método HTTP y Ruta	PATCH /elevators/:elevator_id	status
Request Body	{"doors": "open" | "closed"}	
Response Body	Elevator	200 Ok
Errors	Error("elevator with id <id> not found")	404 Not found
	Error("missing parameter: doors")	400 Bad request
Los valores posibles para **doors** son "open" y "closed"	Error("invalid door state <doors>")	400 Bad request

Obtener piso

Método HTTP y Ruta	GET /levels/:level_id	status
Path Parameters	level_id: número del piso	
Response Body	Level	200 Ok
Errors Valores incorrectos para level podrían ser: {}, [], “hola”, ...	Error("invalid level <level>")	400 Bad request

Crear usuario

[2] el piso puede ser cualquier número del conjunto de los enteros $\mathbb{Z}$.

Crea y retorna un usuario

Método HTTP y Ruta	POST /users	status
Request Body	{"weight": int, "level_id": int}	
Response Body	User	201 Created
En caso de ingresar un usuario incorrectamente	Error("missing parameter: weight")	400 Bad request
Valores incorrectos para level_id podrían ser: {}, [], “hola”, ...	Error("invalid level <level_id>")	400 Bad request
Valores incorrectos para weight podrían ser {}, [], “hola”, ...	Error("invalid weight <weight>")	400 Bad request

Usuario llama al ascensor desde un piso

Método HTTP y Ruta	POST /levels/:level_id/request	status
Request Body	{"user_id": str}	200
Valores incorrectos para level_id podrían ser: {}, [], “hola”, ...	Error("invalid level <level_id>")	400 Bad request
Usuario no encontrado en el piso	Error("user <user_id> not found")	404 Not found
Valores incorrectos para user_id podrían ser: {}, [], “hola”, ...	Error("invalid user <user_id>")	400 Bad request

Usuario sube a ascensor

Método HTTP y Ruta	PUT /elevators/:elevator_id/users/:user_id	status
Path parameters	elevator_id: identificador del ascensor User_id: identificador del usuario	
Response Body	Elevator	200 Ok
Ascensor no encontrado	Error("elevator <elevator_id> not found")	404 Not found
Usuario no encontrado en el piso	Error("user <user_id> not found")	404 Not found
El usuario no se encuentra en el mismo nivel que el ascensor	Error("user and elevator are not on the same level")	400 Bad request

El usuario ya se encuentra arriba del ascensor	Error("user <user_id> already on elevator <elevator_id>")	400 Bad request
El ascensor tiene las puertas cerradas y el usuario no puede entrar	Error("elevator doors are closed")	400 Bad request
Ascensor superó el límite de peso.	Error("elevator max weight exceeded with X kg")	400 Bad request

Usuario solicita ir a un piso desde adentro del ascensor

Método HTTP y Ruta	PUT /elevators/:elevator_id/requests	status
Request Body	{"user_id": str, "level_id": int}	200
Valores incorrectos para level_id podrían ser: {}, [], “hola”, ...	Error("invalid level <level_id>")	400 Bad request
Valores incorrectos para elevator_id podrían ser: {}, [], “hola”, ...	Error("invalid elevator <elevator_id>")	400 Bad request
Valores incorrectos para user_id podrían ser: {}, [], “hola”, ...	Error("invalid user<user_id>")	400 Bad request
Usuario no encontrado en el ascensor	Error("user <user_id> not found")	404 Not found
Ascensor no encontrado	Error("elevator <elevator_id> not found")	404 Not found

Usuario baja de ascensor

Método HTTP y Ruta	DELETE /elevators/:elevator_id/users/:user_id	status
Path parameters	elevator_id: identificador del ascensor User_id: identificador del usuario	
Response Body	Elevator	200 Ok
	Error("elevator <elevator_id> not found")	404 Not found
	Error("user <user_id> not found")	404 Not found

El ascensor no se encuentra en el piso que el usuario había pedido	Error("user requested level X but elevator is on level Y")	400 Bad request
El usuario ya se encuentra arriba del ascensor	Error("user <user_id> already left elevator <elevator_id>")	400 Bad request
El ascensor tiene las puertas cerradas y el usuario no puede salir	Error("elevator doors are closed")	400 Bad request

Versionamiento de código

El sitio y todo su código fuente deberá estar versionado en un repositorio en GitHub, creado en el classroom del curso:

- Link para creación de repositorio: https://classroom.github.com/a/0CjzKmev

Tester

Para probar sus APIs, pueden descargar la imagen de docker del tester: dedarritchon/tester-tarea-1-2024-2, con el comando:

```
docker pull dedarritchon/tester-tarea-1-2024-2:latest
```

Para luego correr el tester con el comando:

```
docker run --rm --net=host -e HOST="http://localhost:8000" dedarritchon/tester-tarea-1-2024-2:latest
```

Indicando en la variable de ambiente HOST la url de su API.

Entregables

Para esta tarea deberán entregar una imagen de docker de su API, subida a un repositorio de docker hub público. Para esto, deben hacer correr:

```
docker build . -t repository-name:tag
```

Para crear una imagen con el tag repository-name:tag, y

```
docker push repository-name:tag
```

Para subirlo al repositorio repository-name. El entregable en Canvas será el identificador **repository-name:tag**, con el que cualquier persona (público) pueda descargarlo usando docker pull.

Fecha de entrega

La tarea debe ser entregada a más tardar el día 4 de septiembre antes de las 18:00, y deberá quedar desplegada y disponible durante el periodo de corrección.

Requisitos mínimos

Las tareas que no cumplan con las siguientes condiciones no serán corregidas y serán evaluados con la nota mínima:

- El código deberá estar versionado en su totalidad en el repositorio de Github Classroom de esta tarea.
- La API debe reflejar fielmente el código entregado en el repositorio.
- El estado de la API debe tener una persitencia de al menos 10 minutos.

Penalizaciones

Se descontarán 0,2 puntos de la nota de la tarea por cada hora de atraso en la entrega, contados a partir de la fecha estipulada en el punto anterior.

En caso de no poder desplegar la API, se evaluará con nota máxima 5,0.

Cualquier intento de copia, plagio o acto deshonesto en el desarrollo de la tarea, será penalizado con nota 1,1 de acuerdo a la política de integridad académica del DCC.

Versiones del documento

- **Versión 1:** 19/08/2024