## Tps The Sims!!



Cosas que Ale tiene que tener en cuenta e implementar:

Agregar Logs en todas las clases con @Slf4j de la librería de lombok

Cuando se finaliza cada parte, se debe hacer un PR para que sea corregido antes de continuar con la siguiente parte  
Usar @RequestMapping en los controllers

En la parte de Jpa debe usar @OneToMany

### Parte 1 (Simple Post)

#### Desarrollo

En base a los videos y teoría vamos a crear una api super simple.

Sabemos en nuestro sistemas existen Personas 👫. Las personas se aburren fácil 😓, así que vamos a modelarles Peliculas 🎥 para que no se aburran 😁, es decir una persona puede tener muchas Peliculas 🎥🎥🎥. Y ademas le vamos a modelar un libro 📕, es decir, las personas tienen un libro 📕

Los datos que sabemos de una Persona son su nombre, apellido, dni, y fecha de nacimiento

Los datos que sabemos de una Movie por ahora es solo su name

Los datos que sabemos de un Libro por ahora solo su ISBN (es un id que identifica a un libro)

#### Post

Se busca que en el post pasemos todos los datos necesarios para modelar a una persona. Y que la respuesta del post sea el nombre de la persona, una lista de sus películas y el libro que tiene asociado  
  
Es simple! pero es para comenzar a adaptarnos a Spring

#### Postman

Para realizar un post necesitamos usar un programa llamado Postman. Dejamos un link a un video explicativo en los recursos

#### Recursos

* Video Jesus Spring: <https://www.youtube.com/watch?v=BYoo8CLO7bE&t=3s&ab_channel=JesusLedesma>
* Postman: [https://www.youtube.com/watch?v=jY1kC4S\_hkk ab channel=CristhianGarciaDigital](https://www.youtube.com/watch?v=jY1kC4S_hkk&ab_channel=CristhianGarciaDigital)
* Curso spring: <https://www.youtube.com/watch?v=9SGDpanrc8U>

### Parte 2 (Rest Template)

Los sistemas nunca están aislados, siempre se comunican con otros. Pero como?? A través de nuestros clients y este sistema no será la excepción. Los microservicios siempre estan en comunicacion con otros microservicios, los hacen a través de json y rest

#### Desarrollo

##### Libro

Hasta ahora de un Libro solo sabemos su ISBN. Pero es base a ese isbn vamos a consultar a una api para conocer sus demás datos title, author, publisher, publishedDate y category

Tenemos que agregar un client que consulte a la api de Google books para traer estos datos

* url: [https://www.googleapis.com/books/v1/volumes?q=isbn:{isbn}](https://www.googleapis.com/books/v1/volumes?q=isbn:%7Bisbn%7D)

Cuando realicemos la pegada vamos a notar que google book retorna un objeto grande con muchos datos, hay que transformar ese objeto grande en el objeto book que nuestro sistema necesita

Un ISBN real con el que pueden hacer la pegada es 0307951189

Pero pueden buscar mas isbn en google de otros libros

Ahora el objeto Book debe poder contener la información que fuimos a buscar: title, author, publisher, publishedDate y category

##### Película

Al igual que sucede con el libro, solo conocemos un dato, su name. Queremos obtener más datos de una película en base a su nombre, por lo tanto vamos a consumir una api que nos provea esta información

* url: [https://imdb8.p.rapidapi.com/title/find?q=[aca](https://imdb8.p.rapidapi.com/title/find?q=%5Baca) va el titulo de pelicula sin corchetes]

Esta api necesita que le envíen dos headers para autorizar la request

* x-rapidapi-host: imdb8.p.rapidapi.com
* x-rapidapi-key': c42b9c3cbfmsh58412fc2b88e92bp12b98bjsna78136cbbfa2

Cuando realicemos la pegada vamos a notar que imd8 retorna un objeto grande con muchos datos, hay que transformar ese objeto grande en varios objetos películas que nuestro sistema necesita  
Por cada título que se ingresó en el post, se debe tener un objeto película

Además tener en cuenta que cuando le pegamos a la api de imdb8, te retorna una lista de opciones que coinciden con el título, quedarse solo con el primer elemento de esa lista  
Vuelvo a reiterar, se debe pegar a la api por cada película que ingresaron en el post, son varias pegadas a la misma api. Por cada pegada por película, la api de imdb8 retorna muchos resultados, quedarse solo con el primero por cada película

Con lo que retorne la api hay que llenar el objeto película con los datos: seriesStartYear, seriesEndYear, numberOfEpisodes y runningTimeInMinutes

##### Output

Ahora el output debe ser el objeto persona completo, donde se visualice toda la información nueva que obtuvimos del libro y de las películas

#### Recursos

* Video Jesus Spring: <https://www.youtube.com/watch?v=BYoo8CLO7bE&t=5s&ab_channel=JesusLedesma>
* Rest template: <https://www.baeldung.com/rest-template>
* Rest template con headers: <https://attacomsian.com/blog/spring-boot-resttemplate-post-request-json-headers>

### Parte 3 (Repositorio en Memoria)

Hasta ahora simplemente retornamos el objeto que creamos, pero qué pasa si lo queremos guardar! Necesitamos una base de datos, generalmente en spring para esto se necesita usar JPA, pero es un modulo mas complejo, asi que por ahora nuestros repositorios serán solo un Map<String, Persona> en memoria

#### Desarrollo

1. Se pide que ahora cuando se crea a una persona, esta quede persistida en nuestro repositorio en memoria (el map :D)
2. El guardar nuestros objetos nos da la posibilidad de seguir creciendo y no solo hacer un post simple. Ahora podemos hacer una api CRUD!!

#### Repository

Un repositorio es una clase donde se encuentra la lógica para obtener elementos de una base de datos. Como por el momento no trabajamos con una base de datos, tendremos un Map<String, Persona> que será usado para guardar en él, las personas que crearemos

A su vez un repositorio tiene que tener métodos para interactuar con este Map. Sus métodos más básicos son

* Save: Usado para crear y Usado para modificar
* FindById: Usado para buscar por id
* DeleteById: Usado para eliminar en base a un id

#### CRUD

Toda api del tipo CRUD cumple con sus siglas:

* Create --> Post --> Crear una persona
* Read --> Get --> Obtener una persona guardada en base a su id
* Update --> Put o Patch --> Modificar una persona guardada
* Delete --> Delete --> Eliminar una persona guardada

Tenemos que hacer una api que cumpla con estos verbos REST. Donde su recurso principal será obviamente una Persona

#### Recursos

* CRUD: <https://www.dariawan.com/tutorials/rest/http-methods-spring-restful-services/>
* Map en Kotlin:

<https://www.youtube.com/watch?v=VKwiXbm-XU8&ab_channel=CristianHenao>

* CRUD + JPA (Ver la interacción con el repositorio, pero no hacer lo de jpa que es @entity @repository):
* <https://www.section.io/engineering-education/spring-boot-crud-api/>

### Parte 4 (Jpa)

Hasta ahora guardamos todo en un repositorio en memoria. Pero si apagamos nuestra app los datos se borran. Por esta razón se usan bases de datos, pero conectarse a ellas tiene un poco más de complejidad. Spring data jpa nos ayuda a solucionarlo

#### Desarrollo

Implementar JPA para guardar una Persona con su respectivas Películas y libro

#### Recursos

* Videos Jesus: <https://www.youtube.com/watch?v=d698GQttHjQ&ab_channel=JesusLedesma>
* CRUD + JPA: <https://www.section.io/engineering-education/spring-boot-crud-api>
* CRUD: <https://www.bezkoder.com/spring-boot-jpa-crud-rest-api/>
* Relaciones: <https://www.adictosaltrabajo.com/2020/04/02/hibernate-onetoone-onetomany-manytoone-y-manytomany/>