Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2143 - Ingeniería de Software 2023-2

Tarea Introduccion a *Rails*

Objetivos

- Aprender a utilizar el *framework Ruby on Rails*.
- Entender e implementar el concepto de *API*.

Fecha de Entrega

31 de agosto, 23:59

Cinema API

En esta tarea usted tendrá que crear una API para registrar, crear, actualizar, y borrar películas, directores y rankings, además de algunas consultas relacionando distintos modelos. Para simplificar el modelo relacional de la tarea se harán algunos supuestos.

- Una película solo puede estar dirigida por una directora o director. Por ejemplo, la película Barbie fue dirigida por Greta Gerwig, mientras que Oppenheimer fue dirigida por Christopher Nolan.
- Existirán solamente 3 páginas de rankings, las cuales son IMDb, Rotten Tomatoes y Metacritic.

Tips:

- Realizar la tarea con tiempo.
- Crear el proyecto con: `rails new CinemaApi --database=postgresql --api`

Modelos mínimos

A continuación, se detallarán los modelos mínimos que debe manejar la *API*, junto a sus atributos. Todo lo señalado a continuación, deberá ser implementado utilizando los mismos nombres, en caso contrario, la corrección fallará.

Model Director

- Atributos:
  - `name` de tipo `string`
  - `age` de tipo `integer`
  - `country` de tipo `string`
  - `has_oscars` de tipo `boolean`
- Asociaciones: Un director puede tener varias películas y varios rankings. Si un director es eliminado, también se deben eliminar sus películas y rankings asociados
- Validaciones: Los directores no pueden tener atributos vacíos ni nulos.

Model Movie

- Atributos:
  - `title` de tipo `string`
  - `sinopsis` de tipo `text`
  - `duration` de tipo `integer`
  - `premiere` de tipo `date`
  - `director` de tipo `references`
- Asociaciones: Una película pertenece a únicamente un director o directora.
- *Validaciones*: Las películas no pueden tener atributos vacíos ni nulos.

Model Ranking

- Atributos:
  - `page` de tipo `string`
  - `min_score` de tipo `float`
  - `score` de tipo `float`
  - `max_score` de tipo `float`
  - `director` de tipo `references`
- Asociaciones: Los rankings hace referencia a unicamente un director, pero un director puede tener varios de estos.
- *Validaciones*: Los rankings no pueden tener atributos vacíos ni nulos.

Rutas solicitadas

Existirán 3 partes, cada una poseerá tests que utilizan solamente los endpoints de dicha parte y las anteriores. A su vez, el puntaje de cada parte estará asociado a cuál es la proporción de tests aprobados en cada una.

Parte 1 (Easy): 2 ptos

1. `GET /directors`

   Debe retornar todos los directores que se encuentren en la base de datos.

   Ejemplo `response body`:

```json
[
    {
        "id": 4,
        "name": "Greta Gerwig",
        "age": 40,
        "country": "United States",
        "has_oscars": false,
        "created_at": "2023-08-17T14:03:40.827Z",
        "updated_at": "2023-08-17T14:03:40.827Z"
    },
    {
        "id": 5,
        "name": "Christopher Nolan",
        "age": 53,
        "country": "United Kingdom",
        "has_oscars": false,
        "created_at": "2023-08-17T14:04:17.730Z",
        "updated_at": "2023-08-17T14:04:17.730Z"
    }
]
```

2. `POST /directors`

   Deberá crear un nuevo director a base de los atributos enviados.

   Ejemplo `request body`:

```json
{
    "director": {
        "name": "Quentin Tarantino",
        "age": 60,
        "country": "United States",
        "has_oscars": true
    }
}
```

   Ejemplo `response body`:

```
{
    "id": 6,
    "name": "Quentin Tarantino",
    "age": 60,
    "country": "United States",
    "has_oscars": true,
    "created_at": "2023-08-17T19:55:48.718Z",
    "updated_at": "2023-08-17T19:55:48.718Z"
}
```

3. `GET /director/:id`

   Debe retornar al director cuya `id` se entrega en la url.

   Ejemplo url: `http://localhost:3000/director/5`

   Ejemplo `response body`:

```
{
    "id": 5,
    "name": "Christopher Nolan",
    "age": 53,
    "country": "United Kingdom",
    "has_oscars": false,
    "created_at": "2023-08-17T14:04:17.730Z",
    "updated_at": "2023-08-17T14:04:17.730Z"
}
```

4. `DELETE /director/:id`

   Deberá eliminar el director cuya `id` coincida con la entregada en la url.

   Ejemplo url: http://localhost:3000/director/5

   Ejemplo response body:

```
{}
```

5. `DELETE /directors`

   Deberá eliminar todos los directores.

   NOTA: Este será el primer endpoint que se ejecutara en todos los tests.

   Ejemplo url: http://localhost:3000/directors

   Ejemplo response body:

```
[]
```

6. `GET /directors/oscars`

   Deberá entregar los directores que posean premios Oscar.

   Ejemplo response body: Similar a Parte 1, Ruta 1

Parte 2 (Medium): 3 ptos

1. `GET /director/:director_id/movies`

   Deberá entregar las películas del director solicitado.

   Ejemplo url: http://localhost:3000/director/7/movies

   Ejemplo response body:

```
[
    {
        "id": 4,
        "title": "Once Upon a Time in Hollywood",
        "sinopsis": "A finales de los 60, Hollywood empieza a cambiar y el actor
            Rick Dalton tratara de seguir el cambio. Junto a su doble, ambos veran
             como sus raices les complican el cambio, y al mismo tiempo su
            relacion con la actriz Sharon Tate",
        "duration": 160,
        "premiere": "2019-05-21",
        "director_id": 7,
        "created_at": "2023-08-17T20:35:39.889Z",
        "updated_at": "2023-08-17T20:35:39.889Z"
    },
    {
        "id": 5,
        "title": "Pulp Fiction",
        "sinopsis": "Historias de dos matones, un boxeador y una pareja de
            atracadores de poca monta envueltos en una violencia espectacular e
            ironica.",
        "duration": 154,
        "premiere": "1994-05-21",
        "director_id": 7,
        "created_at": "2023-08-17T20:43:52.124Z",
        "updated_at": "2023-08-17T20:43:52.124Z"
    }
]
```

2. `POST /director/:director_id/movies`

   Deberá crear una nueva película asociada al director indicado.

   Ejemplo request body:

```
{
    "movie": {
        "title": "Oppenheimer",
        "sinopsis": "El fisico J Robert Oppenheimer trabaja con un equipo de
            cientificos durante el Proyecto Manhattan, que condujo al desarrollo
            de la bomba atomica.",
        "duration": 180,
        "premiere": "11/07/2023"
    }
}
```

Ejemplo response body:

```
{
    "id": 4,
    "title": "Oppenheimer",
    "sinopsis": "El fisico J Robert Oppenheimer trabaja con un equipo de
       cientificos durante el Proyecto Manhattan, que condujo al desarrollo de la
        bomba atomica.",
    "duration": 180,
    "premiere": "2023-07-11",
    "director_id": 7,
    "created_at": "2023-08-17T20:35:39.889Z",
    "updated_at": "2023-08-17T20:35:39.889Z"
}
```

3. `PATCH /director/:director_id/movies/:movie_id`

   Deberá editar la película en base de los nuevos atributos entregados en el body.

   Ejemplo url: localhost:3000/director/7/movies/5

   Ejemplo request body:

```
{
    "movie": {
        "title": "Tiempos Violentos"
    }
}
```

   Ejemplo response body:

```
{
    "director_id": 7,
    "title": "Tiempos Violentos",
    "id": 5,
    "sinopsis": "Historias de dos matones, un boxeador y una pareja de atracadores
        de poca monta envueltos en una violencia espectacular e ironica.",
    "duration": 154,
    "premiere": "1994-05-21",
    "created_at": "2023-08-17T20:43:52.124Z",
    "updated_at": "2023-08-17T20:51:05.938Z"
}
```

4. `GET /movies/sinopsis/:keyword`

   Deberá retornar todas las películas cuya sinopsis incluya la palabra `keyword`

   Ejemplo url: localhost:3000/movies/sinopsis/Robert

   Ejemplo response body:

```
[
    {
        "id": 4,
        "title": "Oppenheimer",
```

```
        "sinopsis": "El fisico J Robert Oppenheimer trabaja con un equipo de
            cientificos durante el Proyecto Manhattan, que condujo al desarrollo
            de la bomba atomica.",
        "duration": 180,
        "premiere": "2023-07-11",
        "director_id": 7,
        "created_at": "2023-08-17T20:35:39.889Z",
        "updated_at": "2023-08-17T20:35:39.889Z"
    }
]
```

Parte 3 (Hard): 1 pto

1. `POST /ranking/:director_id`

   Deberá crear un nuevo ranking para el director indicado por la id.

   Ejemplo request body:

```
{
    "ranking": {
        "page": "IMDb",
        "min_score": 0,
        "score": 3211,
        "max_score": 13011
    }
}
```

   Ejemplo response body:

```
{
    "id": 1,
    "page": "IMDb",
    "min_score": 0.0,
    "score": 3211.0,
    "max_score": 13011.0,
    "director_id": 8,
    "created_at": "2023-08-17T21:31:17.521Z",
    "updated_at": "2023-08-17T21:31:17.521Z"
}
```

2. `GET /ranking/:director_id`

   Deberá entregar todos los rankings asociados al director solicitado.

   Ejemplo response body: Igual a Parte 2, Ruta 1

3. `GET /ranking/top/:quantity`

   Deberá entregar los mejores `quantity` rankings considerando a todos los directores, el puntaje debe ser relativo a la escala que posee cada ranking, indicado por los atributos de puntajes mínimos y máximos

   Tip: Ordenar usando como llave la siguiente fórmula: `(score - min_score) / (max_score - min_score)`

4. `GET /movies/:movie_id/director/rankings`

   Dada una película, entregar todos los rankings del director de esa pelicula.

5. `DELETE /director/ranking/low`

   Deberá eliminar al director que posea el puntaje de ranking más bajo, nuevamente se deberá considerar que el puntaje es relativo al mínimo y máximo de cada ranking.

6. `GET /ranking/pages/all`

   Para cada página, entregar todos los rankings asociados a dicha página ordenados por puntaje.

   Ejemplo response body:


```
{
    "imdb": [
        {
            "id": 6,
            "page": "imdb",
            "min_score": 0.0,
            "score": 3211.0,
            "max_score": 13011.0,
            "director_id": 8,
            "created_at": "2023-08-17T21:47:43.051Z",
            "updated_at": "2023-08-17T21:47:43.051Z"
        }
    ],
    "metacritic": [
        {
            "id": 4,
            "page": "metacritic",
            "min_score": 0.0,
            "score": 80.3,
            "max_score": 100.0,
            "director_id": 8,
            "created_at": "2023-08-17T21:46:40.463Z",
            "updated_at": "2023-08-17T21:46:40.463Z"
        }
    ],
    "rotten_tomatoes": [
        {
            "id": 5,
            "page": "rotten_tomatoes",
            "min_score": 0.0,
            "score": 64.0,
            "max_score": 100.0,
            "director_id": 8,
            "created_at": "2023-08-17T21:47:17.596Z",
            "updated_at": "2023-08-17T21:47:17.596Z"
        }
    ]
}
```

NOTA: Solo existirán esas 3 páginas y se escribirán de la misma forma que aparece en el ejemplo, por ende la respuesta siempre retornará 3 listas, aunque estén vacías

Evaluación

Para evaluar que su *API* funciona correctamente, se utilizará un script de Python para realizar consultas sobre su API. Cada test tendrá puntaje y dificultad asociado y este indicará si Pasa (1) o No Pasa (0), por su parte, cada test consistirá en varios llamados a distintos endpoints. El puntaje total corresponderá a la suma de todos los tests. Se les facilitará un script con test públicos para poder testear sus tareas, para la corrección se utilizarán tests secretos que serán publicados una vez se entreguen las notas.

Entrega

La entrega de esta tarea deberá realizarse en un repositorio de GitHub que se les facilitara, es importante que su código se encuentre en la rama *main*, de lo contrario la tarea no se revisara. Es responsabilidad del alumno preocuparse de aceptar la invitación a su repositorio respectivo. No se aceptaran entregas fuera del repositorio entregado ni fuera de plazo.

Ruta de ejemplo

`https://github.com/IIC2143/2023-2-Tarea-UsuarioDeGithub`

Política de integridad académica

Los alumnos de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Docencia de la Escuela de Ingeniería en el SIDING. Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno, sin apoyo en material de terceros. Por “trabajo” se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno copia un trabajo, obtendrá nota final 1.1 en el curso y se solicitará a la Dirección de Pregrado de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral. Por “copia” se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la referencia correspondiente. Lo anterior se entiende como complemento al Reglamento del Alumno de la Pontificia Universidad Católica de Chile. Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.