Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2143 - Ingeniería de Software 2024-1

Tarea Introduccion a *Rails*

Objetivos

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

Fecha de Entrega

27 de marzo, 23:59

Soccer API

En esta tarea usted tendrá que crear una API para registrar, crear, actualizar, y borrar equipos, partidos y jugadores que pertenecen al fútbol chileno, además de algunas consultas relacionando distintos modelos. Para simplificar el modelo relacional de la tarea se harán algunos supuestos:

- No se requiere diferenciar entre una tarjeta amarilla y una roja, ambas son consideradas como una tarjeta.

Tips:

- Realizar la tarea con tiempo.
- Crear el proyecto con: `rails new SoccerApi --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, ya que utilizamos una automatización para revisar los modelos.

Model Team

- Atributos:
  - `name` de tipo `string`
  - `stadium` de tipo `string`
  - `capacity` de tipo `integer`
  - `city` de tipo `string`
- Asociaciones: Un equipo puede jugar varios partidos y también puede tener varios jugadores.

Model Match

- Atributos:
  - `teamA` de tipo `references`
  - `teamB` de tipo `references`
  - `state` de tipo `boolean`
  - `result` de tipo `string`
- Asociaciones: Un partido pertenece exactamente a dos equipos
- Nota: Un resultado del tipo "2-1" indica que el equipo teamA anotó 2 goles y el equipo teamB anotó 1, por lo que gana el equipo teamA

Model Player

- Atributos:
  - `name` de tipo `string`
  - `goal` de tipo `integer`
  - `assist` de tipo `integer`
  - `card` de tipo `integer`
  - `team` de tipo `references`
- Asociaciones: Cada jugador hace referencia a unicamente un equipo, pero un equipo puede tener varios de estos.

Rutas solicitadas

Las rutas solicitadas se dividen en 3 partes, donde cada una deberá contener los endpoints de la parte anterior. El puntaje de cada parte estará asociado a tests que revisarán que se usen los endpoint de la parte a revisar y los anteriores, y el puntaje será la porporción de tests aprobados en cada una.

Parte 1 (Easy): 2 ptos

1. `GET /teams`

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

   Ejemplo `response body`:

```json
[
    {
        "id": 4,
        "name": "Cobreloa",
        "stadium": "Zorros del desierto",
        "capacity": 12102,
        "city": "Calama",
        "created_at": "2024-03-07T14:03:40.827Z",
        "updated_at": "2024-03-07T14:03:40.827Z"
    },
    {
        "id": 5,
        "name": "Cobresal",
        "stadium": "El cobre",
        "capacity": 11240,
        "city": "El Salvador",
        "created_at": "2024-03-07T14:03:41.827Z",
        "updated_at": "2024-03-07T14:03:41.827Z"
    }
]
```

2. `POST /teams`

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

   Ejemplo `request body`:

```json
{
    "team": {
        "name": "Santiago Wanderers",
        "stadium": "Elias Figueroa Brander",
        "capacity": 18000,
        "city": "Valparaiso"
    }
}
```

   Ejemplo `response body`:

```
{
    "id": 6,
    "name": "Santiago Wanderers",
    "stadium": "Elias Figueroa Brander",
    "capacity": 18000,
    "city": "Valparaiso",
    "created_at": "2024-03-07T14:05:21.827Z",
    "updated_at": "2024-03-07T14:05:21.827Z"
}
```

3. `GET /teams/:id`

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

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

   Ejemplo `response body`:

```
{
    "id": 5,
    "name": "Cobresal",
    "stadium": "El cobre",
    "capacity": 11240,
    "city": "El Salvador",
    "created_at": "2024-03-07T14:03:41.827Z",
    "updated_at": "2024-03-07T14:03:41.827Z"
}
```

4. `DELETE /teams/:id`

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

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

   Ejemplo response body:

```
{}
```

5. `DELETE /teams`

   Deberá eliminar todos los equipos.

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

   Ejemplo url: http://localhost:3000/teams

   Ejemplo response body:

```
[]
```

Parte 2 (Medium): 3 ptos

1. `GET /teams/:team_id/matches`

   Deberá entregar los partidos del equipo solicitado.

   Ejemplo url: http://localhost:3000/teams/6/matches

   Ejemplo response body:

```
[
    {
        "id": 2,
        "teamA": 6,
        "teamB": 4,
        "state": true,
        "result": "2-1",
        "created_at": "2024-03-07T14:12:04.800Z",
        "updated_at": "2024-03-07T14:12:04.800Z"
    },
    {
        "id": 3,
        "teamA": 5,
        "teamB": 6,
        "state": false,
        "result": "-",
        "created_at": "2024-03-07T14:15:44.603Z",
        "updated_at": "2024-03-07T14:15:44.603Z"
    }
]
```

2. `POST /matches`

   Deberá crear un nuevo partido, notar que este debe estar asociado a los equipos indicados.

   Ejemplo request body:

```
{
 match:{
        "teamA": 4,
        "teamB": 5,
        "state": true,
        "result": "1-1",
    }
}
```

   Ejemplo response body:

```
{
    "id": 2,
    "teamA": 4,
    "teamB": 5,
    "state": true,
    "result": "1-1",
```

```
    "created_at": "2024-03-07T14:15:48.243Z",
    "updated_at": "2024-03-07T14:15:48.243Z"
}
```

3. `PATCH /matches/:match_id`

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

   Ejemplo url: localhost:3000/matches/4

   Ejemplo request body:

```
{
    "match": {
        "result": "3-0"
    }
}
```

   Ejemplo response body:

```
{
    "id": 4,
    "teamA": 4,
    "teamB": 5,
    "state": true,
    "result": "3-0",
    "created_at": "2024-03-07T14:15:48.243Z",
    "updated_at": "2024-03-07T14:17:08.526Z"
}
```

4. `GET /matches/:team`

   Deberá retornar todos los partidos cuyo teamA o teamB sea el equipo `team` solicitado

   Ejemplo url: localhost:3000/matches/Cobreloa

   Notar que Cobreloa es el equipo de id 4 en este ejemplo

   Ejemplo response body:

```
[
    {
        "id": 4,
        "teamA": 4,
        "teamB": 5,
        "state": true,
        "result": "3-0",
        "created_at": "2024-03-07T14:15:48.243Z",
        "updated_at": "2024-03-07T14:17:08.526Z"
    }
]
```

Parte 3 (Hard): 1 pto

1. `POST /player/:team_id`

   Deberá crear un nuevo jugador para el equipo indicado por la id.

   Ejemplo url: player/8

   Ejemplo request body:

```
{
    "player": {
        "name": "Arturo Vidal",
        "goal": 0,
        "assist": 3,
        "card": 5
    }
}
```

   Ejemplo response body:

```
{
    "id": 1,
    "name": "Arturo Vidal",
    "goal": 0,
    "assist": 3,
    "card": 5
    "team_id": 8,
    "created_at": "2023-08-17T21:31:17.521Z",
    "updated_at": "2023-08-17T21:31:17.521Z"
}
```

2. `GET /players/:player_id/team`

   Deberá el equipo asociado al jugador solicitado.

   Ejemplo response body: Igual a Parte 1, Ruta 3

3. `GET /players/topGoals/:quantity`

   Deberá entregar los mejores `quantity` jugadores anotando goles considerando a todos los jugadores, los mejores anotadores son simplemente quienes han anotado más goles

4. `GET /players/topAssists/:quantity`

   Deberá entregar los mejores `quantity` jugadores repartiendo asistencias considerando a todos los jugadores, los mejores asistidores son aquellos cuyo mayor porcentaje de estadísticas ofensivas sean asistencias

   Tip: Ordenar usando como llave la siguiente fórmula: (Assist)/(Assist + Goal)

5. `GET /players/topCards/:quantity`

   Deberá entregar los mejores `quantity` jugadores evitando recibir tarjetas, los mejores jugadores limpios son aquellos quienes han recibido menos tarjetas

6. `GET /teams/:team_id/players`

   Para el equipo del id solicitado, entregar todos los jugadores asociados a el.

   Ejemplo response body:

```
[
        {
            "id": 1,
            "name": "Arturo Vidal",
            "goal": 0,
            "assist": 3,
            "card": 5,
            "team_id": 8,
            "created_at": "2023-08-17T21:31:17.521Z",
            "updated_at": "2023-08-17T21:31:17.521Z"
        },

        {
            "id": 2,
            "name": "Carlos Palacios",
            "goal": 5,
            "assist": 3,
            "card": 0,
            "team_id": 8,
            "created_at": "2023-08-17T21:31:17.521Z",
            "updated_at": "2023-08-17T21:31:17.521Z"
        }
]
```

7. `DELETE /teams/matches/low`

   Deberá eliminar y entregar a el peor equipo, para esto deben recordar que un partido ganado entrega 3 puntos, uno empatado 1 punto y perder un partido no entrega puntos. Se define a el peor equipo como aquel que tenga la menor cantidad de puntos, si hay dos equipos con la misma cantidad de puntos será peor el que haya jugado más partidos

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 dentro de los siguientes dias 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/IIC2143/2024-1-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.