Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2143 - Ingeniería de Software 2024-2

Tarea Introducción a *Rails*

Objetivos

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

Fecha de Entrega

24 de Agosto del 2024, 23:59

DCCalendar API

En esta tarea usted tendrá que crear una API para registrar, crear, actualizar, y borrar eventos, calendarios y usuarios. Además, su API debe permitir algunas consultas relacionando distintos modelos.

Tips:

- Realizar la tarea con tiempo.
- Manejar los `DELETE` que afecten a los eventos mediante eliminado en cascada.
- Cuando se indica que cierto modelo debe tener cierto atributo, se espera que su solución verifique que esto ocurra y no permita la presencia de vacíos, nulos o posibles datos que rompan las validaciones.
- El código base de la tarea incluirá los tests que permiten ver cuantos de los objetivos a logrado, si existen problemas en la generación de controladores y modelos por ya contar con los tests, pueden utilizar `-skip-test-framework` para no sobreescribir los tests.
- Para ejecutar la batería de tests con la que podrán testear sus tareas deberán utilizar el comando `rails test` en su consola.

Modelos mínimos

En esta sección 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**, ya que de esta forma los atributos en los JSON de respuesta tendrán los nombres que se pide para cada endpoint. En caso de no utilizar los nombres mencionados, la corrección fallará, ya que utilizamos automatización de pruebas para revisar los modelos.

Model Calendar

- **Atributos:**
  - `name` de tipo `string`
  - `description` de tipo `string`
- **Asociaciones:** Un calendario puede tener múltiples eventos.
- **Consideraciones:** Al eliminar un calendario, todos sus eventos deben ser eliminados.

Model Event

- **Atributos:**
  - `title` de tipo `string`
  - `description` de tipo `string`
  - `date` de tipo `datetime`
- **Asociaciones:** Una evento pertenece exactamente a un solo calendario.

Model User

- **Atributos:**
  - `name` de tipo `string`
  - `email` de tipo `string`
- **Asociaciones:** Cada usuario puede poseer (estar suscrito) a múltiples calendarios.
- **Consideraciones:** Al eliminar un usuario, no deben eliminarse los calendarios a los que está suscrito.

Rutas solicitadas

Las rutas solicitadas se dividen en 3 partes, donde cada una contiene los endpoints respectivos. Cada parte se evaluara usando tests que verificaran el funcionamiento correcto de los distintos endpoints. El puntaje final sera calculado en base a la proporción de pruebas aprobadas en cada parte.

Parte 1 (Easy): 2 ptos

1. `GET /calendars`

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

   Ejemplo `response body`:

```
[
    {
        "id": 1,
        "name": "Personal",
        "description": "Citas medicas, juntas con amigos, etc.",
        "created_at": "2025-08-11T03:44:35.955Z",
        "updated_at": "2025-08-11T03:44:35.955Z"
    },
    {
        "id": 2,
        "name": "Programacion IIC2143",
        "description": "Fechas de pruebas y tareas del curso Ingenieria de
            Software.",
        "created_at": "2025-08-11T03:45:16.054Z",
        "updated_at": "2025-08-11T03:45:16.054Z"
    }
]
```

2. `GET /calendars/:id`

   Debe retornar al calendario cuyo `id` se entrega en la url.

   Ejemplo url: http://localhost:3000/calendars/2

   Ejemplo `response body`:

```
{
    "id": 2,
    "name": "Programacion IIC2143",
    "description": "Fechas de pruebas y tareas del curso Ingenieria de Software.",
    "created_at": "2025-08-11T03:45:16.054Z",
    "updated_at": "2025-08-11T03:45:16.054Z"
}
```

3. `POST /calendars`

   Deberá crear un nuevo calendario en base a los atributos enviados.

   Ejemplo `request body`:

```
{
    "calendar": {
        "name": "Programacion IIC2143",
        "description": "Fechas de pruebas y tareas del curso Ingenieria de
            Software."
    }
}
```

Ejemplo `response body`:

```
{
    "id": 2,
    "name": "Programacion IIC2143",
    "description": "Fechas de pruebas y tareas del curso Ingenieria de Software.",
    "created_at": "2025-08-11T03:45:16.054Z",
    "updated_at": "2025-08-11T03:45:16.054Z"
}
```

4. `GET /users`

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

   Ejemplo `response body`:

```
[
    {
        "id": 1,
        "name": "Pedro Pascal",
        "email": "pedrito.paskalispunk@uc.cl",
        "created_at": "2025-08-11T03:50:39.077Z",
        "updated_at": "2025-08-11T03:50:39.077Z"
    },
    {
        "id": 2,
        "name": "Taylor Swift",
        "email": "taylor13@swift.com",
        "created_at": "2025-08-11T03:51:32.641Z",
        "updated_at": "2025-08-11T03:51:32.641Z"
    },
    {
        "id": 3,
        "name": "Juan Perez",
        "email": "juanperez@gmail.com",
        "created_at": "2025-08-11T03:52:43.562Z",
        "updated_at": "2025-08-11T03:52:43.562Z"
    }
]
```

5. `POST /users`

   Deberá crear un nuevo usuarios en base a los atributos enviados.

Ejemplo `request body`:

```
{
    "user": {
        "name": "Pedro Pascal",
        "email": "pedrito.paskalispunk@uc.cl"
    }
}
```

Ejemplo `response body`:

```
{
    "id": 7,
    "name": "Pedro Pascal",
    "email": "pedrito.paskalispunk@uc.cl",
    "created_at": "2025-08-11T03:50:39.077Z",
    "updated_at": "2025-08-11T03:50:39.077Z"
}
```

6. `DELETE /users/:id`

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

   No debe retornar contenido y el status debe ser `204 (No Content)`

   Ejemplo url: http://localhost:3000/users/3

7. `DELETE /users`

   Deberá eliminar todos los usuarios existentes en la base de datos.

   No debe retornar contenido y el status debe ser `204 (No Content)`

   Ejemplo url: http://localhost:3000/users

Parte 2 (Medium): 3 ptos

1. `POST /events/:calendar_id`

   Deberá crear un nuevo evento que pertenezca al calendario de `id :calendar_id`.

   Ejemplo url: http://localhost:3000/events/2

   Ejemplo request body:

```
{
    "event": {
        "title": "Interrogacion 1",
        "date": "2025-09-26T00:00:00Z",
        "description": "Primera prueba de Ing. Software del 2025"
    }
}
```

   Ejemplo response body:

```
{
    "id": 1,
    "title": "Interrogacion 1",
    "description": "Primera prueba de Ing. Software del 2025",
    "date": "2025-09-26T00:00:00.000Z",
    "calendar_id": 2,
    "created_at": "2025-08-11T04:03:11.475Z",
    "updated_at": "2025-08-11T04:03:11.475Z"
}
```

2. `DELETE /events/:id`

   Deberá eliminar el evento cuyo `id` coincida con el entregado en la url.

   No debe retornar contenido y el status debe ser `204 (No Content)`

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

3. `DELETE /calendars/:id`

   Deberá eliminar el calendario cuya `id` coincida con el entregado en la url.

   No debe retornar contenido y el status debe ser `204 (No Content)`

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

4. `DELETE /calendars`

   Deberá eliminar todos los calendarios.

   No debe retornar contenido y el status debe ser `204 (No Content)`

   Ejemplo url: http://localhost:3000/calendars

5. `GET /events/:calendar_id`

   Deberá retornar todos los eventos que pertenezcan al calendario con `id :calendar_id`.

   Ejemplo url: http://localhost:3000/events/2

   Ejemplo response body:

```
[
    {
        "id": 2,
        "title": "Interrogacion 1",
        "description": "Primera prueba de Ing. Software del 2025",
        "date": "2025-09-26T00:00:00.000Z",
        "calendar_id": 2,
        "created_at": "2025-08-11T04:06:05.551Z",
        "updated_at": "2025-08-11T04:06:05.551Z"
    },
    {
        "id": 3,
        "title": "Interrogacion 2",
        "description": "Segunda prueba de Ing. Software del 2025",
        "date": "2025-11-08T00:00:00.000Z",
        "calendar_id": 2,
        "created_at": "2025-08-11T04:06:48.963Z",
        "updated_at": "2025-08-11T04:06:48.963Z"
    }
]
```

6. `PATCH /events/:id`

   Deberá actualizar el evento de `id :id` dados los datos entregados en la request a la url.

   Ejemplo url: http://localhost:3000/events/3

   Ejemplo request body:

```
[
    "event":{
    "description": "Esta es la segunda interrogacion del curso, estudien porfis."
}
]
```

   Ejemplo response body:

```
{
    "description": "Esta es la segunda interrogacion del curso, estudien porfis.",
    "calendar_id": 2,
    "id": 3,
    "title": "Interrogacion 2",
    "date": "2025-11-08T00:00:00.000Z",
    "created_at": "2025-08-11T04:06:48.963Z",
    "updated_at": "2025-08-11T04:09:06.704Z"
}
```

Parte 3 (Hard): 1 pto

1. `POST /users/:user_id/subscribe/:calendar_id`

   Deberá suscribir al usuario de `id` `:user_id` al calendario de `id` `:calendar_id`.

   Ejemplo url: http://localhost:3000/users/4/subscribe/2

   Ejemplo `response body`:

```
{
    "msg": "Se suscribio Juan Perez a Programacion IIC2143"
}
```

2. `GET /events/next/:calendar_id`

   Debe retornar el siguiente evento más próximo a una fecha (entregada en el body) del un calendario de `id` `:calendar_id`.

   Ejemplo url: http://localhost:3000/events/next/2

   Ejemplo `request body`:

```
{
    "nearDate": "2025-09-14T00:00:00Z"
}
```

   Ejemplo `response body`:

```
{
    "id": 2,
    "title": "Interrogacion 1",
    "description": "Primera prueba de Ing. Software del 2025",
    "date": "2025-09-26T00:00:00.000Z",
    "calendar_id": 2,
    "created_at": "2025-08-11T04:06:05.551Z",
    "updated_at": "2025-08-11T04:06:05.551Z"
}
```

3. `GET /events/next3/:user_id`

   Debe retornar los 3 eventos más próximos a una fecha (entregada en el body) considerando todos los eventos de los calendarios a los que está suscrito un usuario de `id` `:user_id`.

   Ejemplo url: http://localhost:3000/events/next3/4

   Ejemplo `request body`:

```
{
    "nearDate": "2025-09-14T13:00:00Z"
}
```

   Ejemplo `response body`:

```
[
    {
        "id": 8,
```

```
        "title": "Comida familiar",
        "description": "",
        "date": "2025-09-14T13:30:00.000Z",
        "calendar_id": 1,
        "created_at": "2025-08-11T04:21:55.090Z",
        "updated_at": "2025-08-11T04:21:55.090Z"
    },
    {
        "id": 10,
        "title": "Viaje a ver a mis abuelos",
        "description": "",
        "date": "2025-09-16T18:00:00.000Z",
        "calendar_id": 1,
        "created_at": "2025-08-11T04:23:15.051Z",
        "updated_at": "2025-08-11T04:23:15.051Z"
    },
    {
        "id": 2,
        "title": "Interrogacion 1",
        "description": "Primera prueba de Ing. Software del 2025",
        "date": "2025-09-26T00:00:00.000Z",
        "calendar_id": 2,
        "created_at": "2025-08-11T04:06:05.551Z",
        "updated_at": "2025-08-11T04:06:05.551Z"
    }
]
```

Evaluación

Para evaluar que su API funciona correctamente, se realizarán consultas sobre su API donde se verificará que este entregue la respuesta y estado correctos. Cada test tendrá puntaje y dificultad asociado y este indicará si Pasa (1) o No Pasa (0), por su parte. El puntaje total corresponderá a la suma de todos los tests. Se les facilitará una batería de tests con la que podrán testear sus tareas utilizando el comando `rails test` en su consola.

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 importante que el código de su entrega corresponda a las carpetas que conforman su proyecto de RoR sin estar estar contenidas dentro de otra carpeta del tipo DCCalendarAPI. El no cumplir con este aspecto supondrá un descuento de 5 décimas a la nota obtenida por el alumno. De manera similar, la entrega de código ejecutable que presente errores o aspectos harcodeados en su archivo .env, usen una versión de ruby diferente a la del curso o cualquier otro motivo que dificulte la corrección de su tarea, serán penalizados con 5 décimas a su nota final por cada motivo de descuento
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. Cualquier información oficial adicional sobre la tarea será publicada como Issue en el repositorio del curso.

Ruta de ejemplo

`https://github.com/IIC2143/2025-2-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 u herramienta. 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.