PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2513 - Sección 1 — Tecnologías y Aplicaciones Web

# Tarea 3

## Entrega

- **Fecha y hora**: Viernes 26 de abril del 2024, a las 22:00
- **Lugar:** Repositorio individual en la organización del curso en GitHub, main branch.

## Objetivos

- **Diseñar y desarrollar** una API que siga los principios RESTful.
- **Definir** los recursos de la API, tales como las entidades y las rutas necesarias para acceder a estas.
- **Producir** documentación efectiva y clara del proceso realizado.

## Descripción

En esta tarea, tendrán la oportunidad de crear una API RESTful desde cero utilizando Koa y Node.js, que será consumida por un frontend preexistente desarrollado en React por el equipo docente. Este proyecto les permitirá comprender cómo se construyen y se manejan las interfaces de programación de aplicaciones en el contexto del desarrollo web moderno, enfocándose en la integración entre el backend y el frontend, y en la importancia de la documentación adecuada.

Para esta tarea, se les proveerá un entorno de desarrollo inicial que incluirá algunas configuraciones básicas de Koa y Node.js. Dentro de este marco de trabajo, deberán desarrollar una API que gestione recursos para permitir que los usuarios escriban y gestionen registros en un diario personal en línea, basándose en su nombre de usuario. Cada usuario podrá acceder a su diario personalizado, crear nuevos registros, editar registros existentes o eliminarlas si lo desea. Utilizarán rutas como `GET` para recuperar información, `DELETE` para eliminar recursos preexistentes, `POST` para crear nuevos recursos y `PUT/PATCH` para actualizar recursos existentes.

Deberán implementar controladores que procesen las solicitudes recibidas y devuelvan respuestas en formato JSON, utilizando middleware para funciones comunes como el manejo de errores y la validación de datos. Adicionalmente, dado que el frontend ya está desarrollado y utiliza axios para las conexiones, deberán asegurarse de que las respuestas de su API se alineen con las expectativas del código frontend, facilitando una integración fluida y funcional.

# 1. Especificaciones

## 1.1. Entidades y modelos

Se les entrega un diagrama de clases, el cual se encuentra en la carpeta `docs`. Este tiene como finalidad que puedan entender la relación entre las dos entidades principales a desarrollar en la base de datos de la aplicación de diario de vida:

- **User**: Esta entidad representa al usuario de la aplicación. Cada usuario es identificado de manera única por su *username*, que actúa como la *primary key* de la entidad. El *username* es de tipo *string* y es necesario para la autenticación y para vincular los registros del diario al usuario correcto.
- **Entry**: Esta entidad representa un registro individual en el diario de un usuario. Cada registro tiene un identificador único *id* de tipo *int*, que sirve como la *primary key*. Además, la entidad Entry incluye los siguientes atributos:
  - *title*: Un campo de tipo *string* que almacena el título del registro del diario.
  - *body*: Un campo de tipo *string* que contiene el texto completo del registro del diario.
  - *date*: Un campo de tipo *date* que registra la fecha en que se escribió o se modificó el registro del diario.
  - *belongs to*: Un campo de tipo *string* que funciona como una clave foránea (*FK*) y establece una relación con la entidad User, indicando a qué usuario pertenece el registro.

Basándose en la descripción proporcionada, deberán implementar ambos modelos en su base de datos. Es fundamental prestar atención a los tipos de datos específicos de cada atributo y asegurarse de que las relaciones entre los modelos reflejen fielmente la estructura y las conexiones indicadas en el diagrama E-R. Esto significa que deberán establecer una relación uno a muchos entre usuarios y registros del diario, donde un único usuario pueda estar vinculado a múltiples registros, pero cada registro pertenezca exclusivamente a un solo usuario.

## 1.2. Migraciones y Seeds

Después de crear correctamente las entidades, el siguiente paso es aplicar migraciones para estructurar la base de datos según estos modelos.

Además, es necesario generar *seeds* para poder probar la funcionalidad y la integridad de su aplicación. Para esto, deben crear *seeds* que incluyan, como mínimo, tres instancias de la entidad *User* y dos registros de diario (*Entry*) asociadas a cada usuario.

## 1.3. Configuración de Rutas en `routes.js`

En el archivo `routes.js`, su tarea es completar las rutas que aún no se han definido. Deberán importar los módulos de rutas necesarios desde el directorio `routes`, utilizando la función `require`, y luego registrarlos con el enrutador de su aplicación. Las rutas deben configurarse de la siguiente manera:

- Asigne la ruta `/users` para manejar las peticiones relacionadas con la entidad **User**.
- Asigne la ruta `/entries` para manejar las peticiones relacionadas con la entidad **Entry**.

Es importante que las rutas se definan exactamente como se especifica arriba. Cualquier diferencia de las rutas puede resultar en que el frontend, ya desarrollado, no sea capaz de comunicarse correctamente con los endpoints del backend.

### 1.4. Routes

Para cada modelo, se deberá implementar las siguientes rutas

- **Rutas para la entidad** `User`:
  - **Creación de Usuario** (`user.create`):
    - Método `POST` en la ruta `'/'`. Este endpoint se encargará de registrar un nuevo usuario. Deberá aceptar y procesar los parámetros necesarios para crear un usuario satisfactoriamente.
  - **Listado de Usuarios** (`user.list`):
    - Método `GET` en la ruta `'/'`. Este endpoint listará todos los usuarios existentes en la base de datos.
  - **Detalle de Usuario** (`user.show`):
    - Método `GET` en la ruta `'/:username'`. Utilizará el parámetro `username` para buscar y retornar un usuario específico.
- **Rutas para la entidad** `Entry`:
  - **Creación de Registro** (`entry.create`):
    - Método `POST` en la ruta `'/'`. Este endpoint creará un nuevo registro en el diario, procesando los parámetros necesarios para su exitosa inserción.
  - **Listado de Registros** (`entry.list`):
    - Método `GET` en la ruta `'/'`. Este endpoint entregará un listado completo de todos los registros del diario almacenados en la base de datos.
  - **Listado de Registros por Usuario** (`entry.listByUser`):
    - Método `GET` en la ruta `'/user/:username'`. Este endpoint devolverá todos los registros asociadas a un usuario específico, identificadas por el `username` proporcionado.
  - **Detalle de una Registro** (`entry.show`):
    - Método `GET` en la ruta `'/:id'`. Este endpoint entregará los detalles de un registro específico, basándose en su `id`.
  - **Actualización de una Registro** (`entry.update`):
    - Método `PATCH` en la ruta `'/:id'`. Este endpoint permitirá la actualización de los datos de un registro existente, utilizando su `id` como identificador.
  - **Eliminación de una Registro** (`entry.delete`):
    - Método `DELETE` en la ruta `'/:id'`. Este endpoint se encargará de eliminar un registro específico, utilizando su `id` como identificador.

Cada endpoint debe ser diseñado para reflejar correctamente la respuesta mediante códigos de estado HTTP consistentes y mensajes informativos. Para lograr esto, cada endpoint debe incluir:

- `ctx.body`: Aquí se establecerá la respuesta correspondiente a la solicitud. Por ejemplo, la respuesta de un usuario se asignaría de la siguiente manera: `ctx.body = user`. Para situaciones de error o cuando un recurso no se encuentra, se debe seguir un formato estandarizado:
  - En caso de que un recurso no se encuentre: `ctx.body = { error: 'Entry not found' }`.
  - En caso de un error durante la operación: `ctx.body = { error: error.message }`.
- `ctx.status`: Este debe reflejar el código de estado HTTP apropiado para cada respuesta dada por el endpoint, garantizando así que el cliente reciba una señal clara del resultado de su solicitud.

### 1.5. Documentación

Este ítem tiene como propósito:

- Indicar apropiadamente cómo correr su programa.
- Completar todos los puntos especificados en el *README.md*

## 2. Consideraciones

- **Código Frontend**:

  Se les entregará un frontend ya implementado. Este sistema está configurado para interactuar con el backend de manera automática. No deben realizar modificaciones en el código del frontend, con la única excepción de actualizar el archivo `config.js` para definir la URL del servidor local (localhost) que usará la aplicación.

- **Código Backend**:

  El código base proporcionado para el backend no debe ser alterado. No se permite la importación de código adicional. Asimismo, está prohibido agregar librerías que no estén ya incluidas en el archivo `Package.json`. El incumplimiento de esta regla resultará en una calificación automática de 1.0, sin posibilidad de re-corrección.

## 3. Rúbrica

A continuación, se describe el criterio de cada uno de los ítems de la rúbrica con la que se evaluará esta entrega. La nota se calcula al 50 % considerando el puntaje descrito.

- **Entidades [3 puntos]**: Definir correctamente las entidades, junto con sus atributos y relaciones correctamente.
- **Migraciones y seeds [2.5 puntos]**: Migrar correctamente el modelo y que las seeds cumplan con el mínimo estipulado en el punto 1.2.
- **Routes.js [1.5 puntos]**: Importar e instanciar correctamente las rutas especificadas en el puntos 1.3.
- **EndPoints [11.5 puntos]**: Crear y exportar correctamente todos los metodos especificados en el punto 1.4. Además de incluir las respuestas de estado HTTP correspondientes a los endpoints.
- **Documentación [7 puntos]**: Deben indicar cómo correr tu programa y responder las preguntas indicadas en el *template* que te facilitamos.

## 4. Integridad académica

Cualquier situación de **falta a la integridad académica** detectada en el contexto del curso (por ejemplo, durante alguna evaluación) tendrá como **sanción un 1**,**1 final en el curso**. Esto sin perjuicio de sanciones posteriores que estén de acuerdo a la Política de Integridad Académica de la Escuela de Ingeniería y de la Universidad, que sean aplicables al caso. Rige para este curso tanto la política de integridad académica del Departamento de Ciencia de la Computación (ver anexo) como el **Código de honor de la Escuela de Ingeniería**.

## Dudas

Cualquier duda que se presente acerca del enunciado debes consultarla en las ***issues*** del repositorio del curso. Recuerden que como equipo docente estaremos atentos para poder ayudarlos :)