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

IIC2513 - Sección 1 — Tecnologías y Aplicaciones Web

Tarea 2

Entrega

- **Fecha y hora**: Viernes 18 de abril del 2025, 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 suban memes a una red social. Cada Cada usuario podrá crear un meme, comentar en memes de otras personas, darles likes y también, ver los memes más virales!. 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 4 entidades principales a desarrollar en la base de datos de la aplicación de memes:

- **User**: Esta entidad representa al usuario de la aplicación. Cada usuario es identificado de manera única por su *userId*, que actúa como la *primary key* de la entidad. Este atributo es necesario para la autenticación y para vincular los memes al usuario correcto. Además, los usuarios cuentan con el atributo *username* que es de tipo *string*.
- **Meme**: Esta entidad representa un registro de un meme del usuario. Cada registro tiene un identificador único *memeId* de tipo *int*, que sirve como la *primary key*. La entidad incluye Meme incluye los siguientes atrubitos:
  - *tittle*: un campo de tipo *string* que almacena el título del meme
  - *url*: un campo de tipo *string* que almacena la url de donde se encuentra el meme.
  - *likeCount*: un campo de tipo *int* que indica la cantidad de likes que tiene un meme.
  - *userId*: un campo de tipo *int* que indica el id del usuario que ha publicado el meme.
- **Like**: esta entidad representa el registro de un like a un meme, teniendo la información de quién le dio like un meme. Esta entidad presenta los siguientes campos:
  - *likeId*: un campo de tipo *int* que es la *primary key* del modelo. Es el identificador único de un like.
  - *userId*: es un campo de tipo *int* que funciona como la *foreign key* del modelo, estableciendo la relación con la entidad User, indicando qué usuario dio el like.
  - *memeId*: es un campo de tipo *int* que funciona como la *foreign key* del modelo estableciendo la relación con la entidad Meme. Es decir, indica a qué meme le corresponde el like.
- **Comment**: esta entidad repreesnta el registro de un comentario de un meme. Sus campos son:
  - *commentId:* un campo de tipo *int* que es la *primary key* del modelo. Es el identificador único de un comentario.
  - *body*: un campo de tipo *string* que es el texto del comentario creado por un usuario.
  - *userId*: es un campo de tipo *int* que funciona como la *foreign key* del modelo, estableciendo la relación con la entidad User, indicando quién creó el comentario.
  - *memeId*: es un campo de tipo *int* que funciona como la *foreign key* del modelo estableciendo la relación con la entidad Meme. Es decir, indica a qué meme le corresponde el comentario.

Basándose en la descripción proporcionada, deberán implementar los 4 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, por ejemplo, que deberán establecer una relación uno a muchos entre usuarios y memes, donde un único usuario pueda estar vinculado a múltiples memes, pero cada meme pertenezca exclusivamente a un solo usuario. Deberán hacer todas las relaciones.

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*, además de crear 3 instancias de memes (una para cada usuario), tres *Likes* y finalmente tres comentarios a libre elección.

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 `/memes` para manejar las peticiones relacionadas con la entidad **Entry**.
- Asigne la ruta `/likes` para manejar las peticiones relacionadas con la entidad **Like**.
- Asigne la ruta `/comments` para manejar las peticiones relacionadas con la entidad **Comment**.

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**:
    - 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.
  - **Detalle de Usuario**:
    - Método `GET` en la ruta `'/:id'`. Utilizará el parámetro `id` para buscar y retornar un usuario específico.
  - **Actualización de Usuario**:
    - Método `PUT` en la ruta `'/:id'`. Este endpoint utilizará el parámetro `id` para actualizar los datos de un usuario específico.
  - **Eliminación de Usuario**
    - Método `DELETE` en la ruta `'/:id'`. Este endpoint utilizará el parámetro `id` para eliminar a un usuario específico.
- **Rutas para la entidad** `Meme`:
  - **Creación de Meme**:

- Método `POST` en la ruta `'/'`. Este endpoint creará un nuevo Meme, procesando los parámetros necesarios para su exitosa inserción.

- **Listado de Memes**:
  - Método `GET` en la ruta `'/'`. Este endpoint entregará un listado completo de todos los memes almacenados en la base de datos.
- **Listado de Memes Ordenado**:
  - Método `GET` en la ruta `'/'`. Este endpoint entregará un listado completo de todos los memes almacenados en la base de datos, ordenados de mayor a menor por cantidad de likes.
- **Detalle de un Meme**:
  - Método `GET` en la ruta `'/:id'`. Este endpoint entregará los detalles de un meme específico, basándose en su `id`.
- **Comentarios de un Meme**:
  - Método `GET` en la ruta `'/:id/comments'`. Este endpoint entregará todos los comentarios de un meme en específico, basándose en su `id`.
- **Actualización de un Meme**:
  - Método `PATCH` en la ruta `'/:id'`. Este endpoint permitirá la actualización de los datos de un meme existente, utilizando su `id` como identificador.
- **Eliminación de un Meme:**
  - Método `DELETE` en la ruta `'/:id'`. Este endpoint se encargará de eliminar un meme específico, utilizando su `id` como identificador.

- **Rutas para la entidad** `Comment`:
  - **Creación de Comentario**:
    - Método `POST` en la ruta `'/'`. Este endpoint creará un comentario, procesando los parámetros necesarios para su exitosa inserción.
  - **Eliminación de un Comentario:**
    - Método `DELETE` en la ruta `'/:id'`. Este endpoint se encargará de eliminar un comentario específico, utilizando su `id` como identificador. El comentario puede ser eliminado únicamente por quién creó el meme, o por quién comentó dicho comentario.

- **Rutas para la entidad** `Like`:
  - **Creación de Like**:
    - Método `POST` en la ruta `'/'`. Este endpoint creará un nuevo Like, procesando los parámetros necesarios para su exitosa inserción. Además, debe aumentar en 1 el valor de *likeCount* del modelo *Meme*.
  - **Eliminación de un Like:**
    - Método `DELETE` en la ruta `'/:id'`. Este endpoint se encargará de eliminar un like específico, utilizando su `id` como identificador. Además, debe disminuir en 1 el valor de *likeCount* del modelo *Meme*.
  - **Comprobación de un Like:**
    - Método `GET` en la ruta `'/'`. Este endpoint se encargará de retornar en el body de la respuesta un True si un usuario dio like a un meme, y False en caso contrario. Debe utilizar *userId* y *memeId* para comprobarlo.

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. Es decir, deberás retornar los códigos 2XX, 4XX y 5XX relacionados a cada endpoint.

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. Cabe destacar que no todos los endpoints son utilizados por el frontend entregado. Aún así, deberán crear todos los endpoints indicados para el backend.
- **Código Backend**: El código base proporcionado para el backend no debe ser alterado. Es decir, 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 [11 puntos]**: Definir correctamente las entidades, junto con sus atributos y relaciones correctamente.
- **Migraciones y seeds [13.5 puntos]**: Migrar correctamente el modelo y que las seeds cumplan con el mínimo estipulado en el punto 1.2.
- **Routes.js [2 puntos]**: Importar e instanciar correctamente las rutas especificadas en el puntos 1.3.
- **EndPoints [45.5 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 [10 puntos]**: Deben indicar cómo correr tu programa y responder las preguntas indicadas en el *template* que te facilitamos.

4. Bonus

Se entregará un bonus de 5 décimas a aquellos que entreguen una documentación en un archivo .collections de Postman para las entidades *User* y *Memes*.

Para cada endpoint, deberán entregar:

- Método del endpoint
- Url completa del endpoint
- Body con los parámetros necesarios para el endpoint
- Ejemplo de respuesta exitosa (200/201)
- Ejemplo de respuesta errónea (4XX)

El bonus se otorgará de la siguiente forma:

- 5 décimas: documentando la totalidad de endpoints de las entidades *User* y *Meme*, además de contar con todo lo solicitado para cada endpoint.
- 2.5 décimas: documentando más del 50 % de los endpoints de las entidades *User* y *Meme*, además de contar con todo lo solicitado para cada endpoint.
- 0 décimas: documentando menos del 50 % de las entidades *User* y *Meme*

5. 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 :)