Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2513-Tecnologías y Aplicaciones Web

Tarea 3

Entrega:

- **Fecha y hora**: Viernes 4 de octubre del 2024, a las 23:59
- **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 sistema de chat, basándose en su nombre de usuario. Cada usuario podrá acceder a su chat personal, crear nuevos chats, editar mensajes existentes o eliminarlos si lo desea.

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.

Especificaciones

Entidades y modelos

Se les entrega un diagrama de clases, el cual se encuentra en el *root* en formato `.jpg`. Este tiene como finalidad que puedan entender la relación entre las entidades principales a desarrollar en la base de datos de la aplicación:

- **User**: Esta entidad representa a un usuario de la aplicación. Cada usuario es identificado de manera única por su *username* de tipo *string*, que actúa como la `PK` de la entidad. Un usuario puede participar en múltiples chats como uno de los dos participantes (relacionado por la `FK` *username1* y *username2* en la entidad *Chat*) y también puede enviar múltiples mensajes (relacionado por la `FK` *username* en la entidad *Message*).
- **Chat**: Esta entidad representa una conversación entre dos usuarios. Cada registro en esta tabla tiene un identificador único *Id* de tipo *int*, que sirve como la `PK`. Un chat siempre contiene a dos usuarios, que se vinculan a través de las `FK` *username1* y *username2*, ambas referenciando a la entidad *User*. Un chat puede contener múltiples mensajes, los cuales se almacenan en la entidad *Message* y se relacionan con el *Chat* mediante la `FK` *chatId*.
- **Message**: Esta entidad representa un mensaje dentro de un chat. Cada mensaje tiene un identificador único *Id* de tipo *int*, que actúa como la `PK`. Además, la entidad contiene el texto del mensaje, la referencia al chat al que pertenece (*chatId*), y el usuario que lo envía (*username*). Las `FK` *chatId* y *username* establecen relaciones con las entidades *Chat* y *User*, respectivamente.

Basándose en la descripción proporcionada, deberán implementar los 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 correctamente las relaciones entre los modelos.

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, 5 instancias de la entidad *User*, 3 instancias de *Chats* asociados entre distintos usuarios y 3 mensajes por cada chat.

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 `/chats` para manejar las peticiones relacionadas con la entidad **Chat**.
- Asigne la ruta `/messages` para manejar las peticiones relacionadas con la entidad **Message**.

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.

Routes

Para cada modelo, se deberá implementar las siguientes rutas

- **User**:
  - Creación un usuario:
    - Método `POST`
    - Ruta `'/'`
  - Listado de todos los usuarios:
    - Método `GET`
    - Ruta `'/'`
  - Detalle completo de un usuario:
    - Método `GET`
    - Ruta `'/:username'`
  - Listado de todos los chats de un usuario:
    - Método `GET`
    - Ruta `'/:username/chats'`
- **Chat**:
  - Crear un chat entre dos usuarios:
    - Método `POST`
    - Ruta `'/'`
  - Obtener los mensajes de un chat:
    - Método `GET`
    - Ruta `'/:id/messages'`
  - Eliminar un chat:
    - Método `DELETE`
    - Ruta `'/:id'`
  - Obtener un chat:
    - Método `GET`
    - Ruta `'/:id'`
  - Obtener el chat de dos usuarios:
    - Método `GET`
    - Ruta `'/:username1/:username2'`
- **Message**:
  - Crear un mensaje:
    - Método `POST`
    - Ruta `'/'`
  - Obtener un mensaje especifico:
    - Método `GET`
    - Ruta `'/:id'`
  - Actualizar un mensaje:
    - Método `PUT`
    - Ruta `'/:id'`
  - Eliminar un mensaje:
    - Método `DELETE`
    - Ruta `'/:id'`

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: '[MODEL] 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.

Documentación

Este ítem tiene como propósito:

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

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. Para poder ver los cambios en el chat, se recomienda utilizar el navegador en modo incógnito e ir actualizando la página una vez enviado el mensaje. Su backend debe acoplarse correctamente al funcionamiento del frontend, el no cumplimiento de esta regla ocasionará que la revisión sea mediante postman con un 4.0 como nota máxima.

- **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.

- El método `DELETE` del chat, debe eliminar en cascada los mensajes asociados a este, para ambos usuarios.

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 total de puntos.

- **Entidades [8 puntos]**: Definir correctamente las entidades, junto con sus atributos y relaciones correctamente.
- **Migraciones y seeds [10 puntos]**: Migrar correctamente el modelo y que las seeds cumplan con el mínimo estipulado en el punto de las seeds.
- **Routes.js [1.5 puntos]**: Importar e instanciar correctamente las rutas especificadas en el punto de Routes.js.
- **EndPoints [36 puntos]:** Crear y exportar correctamente todos los métodos especificados en el punto 1.4. Además de incluir las respuestas de estado HTTP correspondientes a los endpoints.
- **Documentación [9.5 puntos]:** Deben indicar cómo correr tu programa y responder las preguntas indicadas en el *template* que te facilitamos.

Dudas

Pueden dejar sus preguntas sobre instalación o enunciado en las issues del repositorio del curso. No se van a responder dudas por correo.

Política de atraso

Pueden revisar la política de atraso en el programa del curso.

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 como el Código de Honor de la Escuela de Ingeniería.

Debido a la naturaleza de la disciplina en la que se enmarca el curso, está permitido el uso de código escrito por una tercera parte, pero solo bajo ciertas condiciones. Primero que todo, el uso de código ajeno siempre debe estar visible y correctamente referenciado, indicando la fuente de donde se obtuvo. Por otro lado, se permite el uso de código encontrado en internet u otra fuente de información similar, siempre y cuando su autor sea externo al curso, o en su defecto, sea parte del equipo docente del curso. Es decir, se puede hacer referencia a código ajeno al curso y código perteneciente al curso pero solo aquel escrito por el equipo docente, como material o ayudantías. Luego, compartir o usar código de una evaluación actual o pasada se considera una falta a la ética.