```
git remote add origin https://github.com/Jair-Academlo/EJERCICIO-2.git
```

Semana 2: Tasks

Basado en el ejercicio de la semana pasada, nuestro senior developer nos hizo el siguiente comentario:

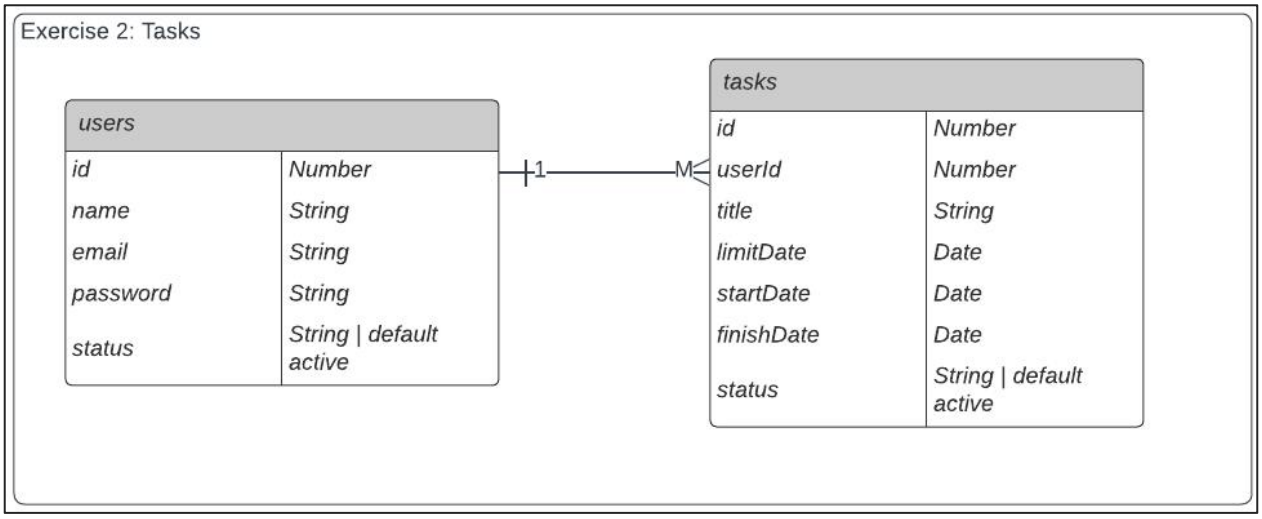
"*Muy buen trabajo con tu solución de la semana pasada. Creo que podemos mejorar en ciertas partes, pero creo que tu trabajo fue muy eficaz y al cliente le gusto.*

Nuestro próximo cliente es un administrador de proyectos (Project manager), este se encarga de delegar tareas a los miembros de su equipo, por lo que necesita que desarrollemos una aplicación capaz de registrar usuarios (los miembros de su equipo) y crear tareas para esos usuarios.

*De acuerdo con el cliente, cada tarea tiene una fecha de inicio, una fecha límite para completar la tarea y una fecha de terminación, esta cuando se completa la tarea. El cliente está interesado en que pueda traer la información de las tareas pendientes (**active**), las que fueron completadas (**completed**), las que no fueron completadas a tiempo (**late**) y las que fueron canceladas (**cancelled**).*

Al crear una tarea, se debe proporcionar el usuario (su id) para poder asignarle la tarea, y se debe proporcionar la fecha límite de dicha tarea."

Crea un nuevo proyecto de Express, conéctate a una nueva base de datos (tasks) y genera los siguientes modelos:



Implementar los siguientes endpoints:

/api/v1/users		
HTTP Verb	**Route**	**Description**
POST	*/*	*Crear usuario (enviar name, email, y password por req.body)*
GET	*/*	*Obtener a todos los usuarios **activos***
PATCH	***/:id***	*Actualizar perfil de usuario (solo name y email)*
DELETE	***/:id***	*Deshabilitar cuenta de usuario*

/api/v1/tasks		
HTTP Verb	**Route**	**Description**
POST	*/*	*Crear tarea (enviar title, userId, y limitDate por req.body) La fecha para ser aceptada por Sequelize debe ser del siguiente formato "YYYY-MM-DD HH:mm:ss"*
GET	*/*	*Obtener a todas las tareas registradas*
GET	***/:status***	*Obtener las tareas de acuerdo con el status que nos envíen.*
PATCH	***/:id***	*Actualizar de una tarea de acuerdo con el id.*
DELETE	***/:id***	*Cancelar la tarea (status **cancelled**)*

Para el endpoint **GET /:status**, *validar que el valor dinámico sea alguno de los siguientes (**active, completed, late, cancelled**), en caso de que no, enviar un error al cliente.*

Para el endpoint **PATCH /:id**

- *Se debe validar que la tarea (task) exista en la bd con status **active**, en caso de que no, enviar error.*
- *Se enviara un valor por req.body, el cual es el **time**, es decir, la hora en la que el usuario termino la tarea.*
- *Se debe comparar la fecha de entrega (**limitDate**) del registro encontrado, con el valor que se envía por req.body.*
    - *Ejemplo, si el valor de limitDate es "6 de junio" y la **time** que envía el usuario (la fecha que termino la tarea) es de "5 de junio", el usuario termino la tarea a tiempo (con status **completed**).*

- *Si el valor de **limitDate** es “6 de junio” y la **time** que envía el usuario es de “8 de junio”, el usuario termino la tarea tarde (con status **late**).*
- *Deberán investigar el como comparar fechas por su cuenta.*

Instala *express-validator* y úsalo para validar los siguientes campos antes de crear un usuario o una tarea:

Modelo	**Campos**
Users	Name, email, password
Tasks	Title, userId

Instalar *dotenv* y aplicarlo para usar variables de entorno para nuestras credenciales de nuestra base de datos.

Utilizar lo visto para aplicar *error handling* en nuestro proyecto (catchAsync, globalErrorHandler y AppError) y optimizar nuestro código.