Tarea 2: Base de Datos

Dylhan Aros
dylhan.aros@sansano.usm.cl

Daniel Dueñas
daniel.duenas@usm.cl

Nicolás Goméz
nicolas.gomezma@usm.cl

Fabian Maldonado
fabian.maldonado@usm.cl

Eva Wang
eva.wang@usm.cl

Fernando Banz
fernando.banz@sansano.usm.cl

2023-02

1 Introducción

Después de haber ayudado a Ken a optimizar su sistema de gestión de contactos en la tarea anterior, tu grupo de desarrollo llamó la atención de una empresa multinacional.

La empresa está buscando un grupo de desarrollo experto de la Universidad Federico Santa María que sea capaz de programar un sistema de gestión de usuarios y comunicaciones interno que tenga características especiales que lo diferencien de los sistemas tradicionales. Este sistema permitirá a los usuarios registrarse y ser administrados de manera efectiva, donde los administradores podrán crear, modificar, y bloquear cuentas de usuario, mientras que los usuarios podrán obtener información de otros usuarios y marcar ciertos correos electrónicos como favoritos.

La empresa ya tiene una idea de cómo quiere hacer el sistema y cómo se llamará (CommuniKen). Han definido los endpoints y las funcionalidades que debe tener el cliente, pero requieren ayuda para desarrollar el software y el modelo lógico.

Debes tener cuidado, ya que la empresa quiere que se desarrolle el cliente de la aplicación en Python y que la API se haga en JavaScript (o TypeScript si desean implementar características avanzadas). Se usará PostgreSQL como sistema de bases de datos. Adicionalmente, utilizarán el paquete PrismaJS

para conectarse al DBMS y, lo más importante, deben usar el paquete Elysia como framework de servidor HTTP.

La empresa tiene mucha fe en tu equipo de desarrollo, ¡no los decepcionen!

2 Requerimientos

- El cliente (donde el usuario podrá acceder) será un programa (.py) de estilo CLI (Command Line Interface) que haga consultas HTTP a la API. *Para mas referencia de como funciona ir a la sección de Cliente.*

  Se tiene que crear un archivo de texto plano llamado "requirements.txt" de tal forma que tengan que escribir todas las dependencias que ocupen en el proyecto, de forma que si uno escribe en la terminal **pip -r requirements.txt** se instale todo lo necesario para poder correr la aplicación.

  Además se tiene que crear un archivo de texto plano llamado README.txt que de instrucciones breves de como poder correr el cliente sin ningún inconveniente.

- La API se tendrá que desarrollar sobre **Bun** usando el framework web HTTP **Elysia**

- El sistema de base de datos que se ocupara es PostgreSQL

- Para conectar la API al sistema de base de datos hay que usar el paquete llamado PrismaJS.

  Se tiene que escribir una breve explicación en un archivo de texto plano llamado "README.txt" que permita dar instrucciones breves de como poder ejecutar la API.

- Tanto como el cliente y la API deben estar correctamente comentadas, no se pide en un formato en especifico, pero se pide ser ordenado con el código y usar buenas practicas.

- **NO** se puede asumir que las consultas que se hagan a la API se harán siempre de forma correcta, es decir, que si alguien hace una consulta que no cumpla con los requerimientos del endpoint de la API debe de dar su código HTTP de error de respuesta correspondiente.

  Esto no significa que las consultas que haga el cliente a la API se hagan de forma incorrecta, solo significa que en caso de que otro servicio que no sea el cliente quiera ocupar la API se asegure que se haga de forma correcta.

- Cada vez que llegue una consulta a la API se debe "printear" un log (un registro) mostrando que llego una consulta describiendo brevemente lo que ocurre.

  Por ejemplo, si llega una consulta de registro y se hizo correctamente se podría mostrar el siguiente log.

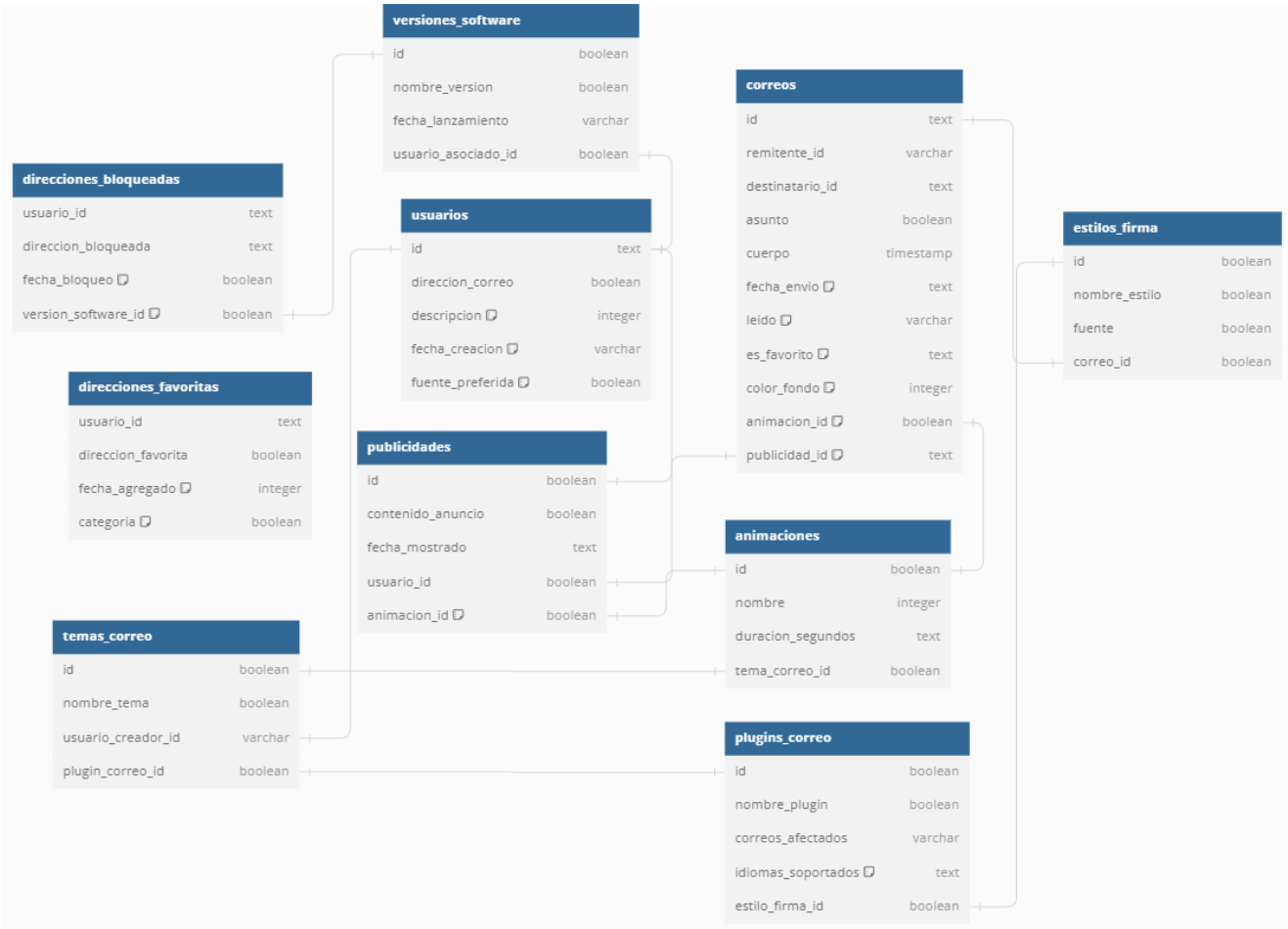
  Ejemplo:

  [15:40] Se ha registrado el usuario: nickname de forma correcta

3 Modelo lógico

Como se había explicado, la empresa no tiene mucha idea de como hacer un modelo lógico, te puedes dar cuenta de que el modelo le faltan relaciones, están mal definida la mayoría de los tipo de datos para cada atributo, como también hay tablas que claramente son innecesarias para el trabajo por el cual te han contratado.

Dado el modelo inicial de la empresa, tienes que arreglarlo, dejando el modelo lo más claro posible, definiendo relaciones de forma correcta, eliminando atributos innecesarios y definiendo de forma correcta los tipos de datos.



4 Endpoints

- Registrar usuario: **/api/registrar**

  Tipo de petición: **POST**

  Se le tiene que enviar al servidor un JSON con este formato

```
{
    "nombre": "Daniel",
    "correo": "daniel.duenas@usm.cl",
    "clave": "clavecita123",
    "descripcion": "Una descripcion que puede escribir el usuario"
}
```

  La respuesta del servidor se indica al final de la sección

- Bloquear un usuario: **/api/bloquear**

  Tipo de petición: **POST**

  Se le tiene que enviar al servidor un JSON con este formato

```
{
    "correo": "daniel.duenas@usm.cl",
    "clave": "clavecita123",
    "correo_bloquear": "fernando.banz@sansano.usm.cl"
}
```

  La respuesta del servidor se indica al final de la sección

- Obtener información publica de cierto usuario: **/api/informacion/:correo:**

  Tipo de petición: **GET**

  Si se accede a la ruta **/api/informacion/sebastian.torrealba@usm.cl**, debería mostrarme la siguiente informacion del usuario:

```
{
    "estado" 200,
    "nombre": "Daniel",
    "correo": "daniel.duenas@usm.cl",
    "descripcion": "Una descripcion que puede escribir el usuario"
}
```

  En caso de respuesta incorrecta, se indica al final de la sección.

- Marcar como favorito un correo electrónico: **/api/marcarcorreo**

  Tipo de petición: **POST**

```
{
    "correo": "daniel.duenas@usm.cl",
    "clave": "clavecita123",
    "id_correo_favorito": 1
}
```

  La respuesta del servidor, se indica al final de la sección.

- Desmarcar como favorito un correo electrónico: **/api/desmarcarcorreo**

  Tipo de petición: **DELETE**

```
{
    "correo": "fernando.banz@sansano.usm.cl",
    "clave": "clavecita123",
    "id_correo_favorito": 1
}
```

  La respuesta del servidor, se indica al final de la sección.

- Guardar como contacto una dirección de correo electrónico:

  Importante

  Al momento de hacer las consultas pueden ser exitosas o ocurrir un error al hacer la petición, por lo tanto, para todas las peticiones hay que dar la siguiente respuesta.

  Al hacer una petición, tenemos una de dos posibilidades.

  - En caso de que se haya realizado correctamente:

```
{
    "estado": 200,
    "mensaje": "Se realizo la peticion correctamente"
}
```

  - En caso de que no se haya cumplido lo necesario para realizar la petición

```
{
    "estado": 400,
    "mensaje": "Ha existido un error al realizar la peticion"
}
```

  En el caso de las peticiones de tipo **GET** no hay que dar la respuesta de tipo "correcta" mostrada superiormente, hay que mostrar la de error en caso de que exista algún problema.

  Además, tienen que adaptar el mensaje de error o que se realizó satisfactoriamente dependiendo el contexto de la petición.

5 Cliente

- Al iniciar el cliente se abrirá un "prompt" donde se pedirá el correo electrónico y la clave del usuario, en caso de que sean correctas, se deberá mostrar el menú de opciones, si no, deberá terminar la ejecución del cliente.
- El menú deberá mostrar las siguientes opciones:
  - 1. Enviar un correo
    (En caso de que no exista el correo, deberá mostrar un mensaje de error)
  - 2. Ver información de una dirección de correo electrónico
    (En caso de que no exista el usuario, hay que dar un mensaje de error en el cliente)
  - 3. Ver correos marcados como favoritos
  - 4. Marcar correo como favorito
  - 5. Terminar con la ejecución del cliente

  Se deberá mostrar el menú hasta que sea elegida la opción 5

6 Consejos y recomendaciones

- Se recomienda empezar la tarea con la mayor antelación posible, ya que, puede ser extensa y hacerla los últimos días de plazo puede que sea una experiencia muy estresante.
- Se recomienda usar el paquete **requests** de Python para la parte del cliente
- Si quieres probar el endpoint de la API puedes ocupar programas como **Postman** o en la terminal con el comando **curl**
- El material de la ayudantía debería ser suficiente para poder realizar toda la tarea, pero de todas maneras, permitimos buscar información en internet para realizar la tarea.

7 Calificación

- Por cada endpoint de la API se darán (+6 pts)

  Se considera que fue creado correctamente creado el endpoint de la API si efectivamente verifica si están todos los datos necesarios para insertar el dato en la tabla y da mensajes de errores respectivos en caso de que no se cumpla

- Crear un modelo lógico claro, definiendo atributos y tipos de datos de forma correcta, como también generando relaciones con llaves primarias y foráneas de forma adecuada. (+22 pts)

  Si se considera que tienes un 50% bueno, entonces tienes +11 pts, es decir, el puntaje es proporcional al porcentaje correcto que se tenga en el modelo

- Uso incorrecto de Git: (-20 pts)

  Un uso adecuado de Git implica realizar commits y push al repositorio. No requerimos la creación de branches ni merges, pero entregar la tarea con un único commit conlleva una penalización significativa.

- No se respeta el formato de entrega: (-10 pts)

  (Condiciones de entrega están al final del documento)

- Buen diseño del cliente: (+5pts)

  Se considera un buen diseño, si se hace un diseño llamativo mas que un simple menú.

- Programar la tarea con Typescript: (+15 pts)

  Se considerara un uso de Typescript correcto si usan tipos de datos e interfaces de forma correcta, si definen la mayoría de las variables como "any", no se agregará el puntaje adicional

  En caso de que el puntaje total exceda los 100 puntos, el exceso se acumulará para futuras tareas. Sin embargo, si el promedio de tareas supera los 100 puntos, se aplicará la fórmula:

  $$Nf = \min(100, N)$$

  donde N es la nota actual y Nf la nota final.

8 Condiciones de entrega

Lea detenidamente cada punto:

- La base de datos a utilizar es **PostgreSQL**. Si utiliza una base de datos diferente y el ayudante no puede ejecutarla, no estamos obligados a revisar su tarea.
- En el repositorio de la tarea existirán dos carpetas, una llamada **client** y otra llamada **api**, donde escribirán el código respectivo de cada uno.
- La entrega se realizará a través de la plataforma **Aula**, donde deberán subir un archivo de texto (.txt) con el nombre y rol de cada miembro del grupo y un enlace al repositorio de GitLab con la tarea.
- También al mismo nivel de las carpetas de client y api se debe crear un archivo (.txt) que tenga los nombres de los integrantes del grupo con su respectivo rol
- Se debe hacer **una** entrega por grupo.
- Los ayudantes **no** están obligados a resolver dudas cuando queden menos de 18 horas para la entrega.