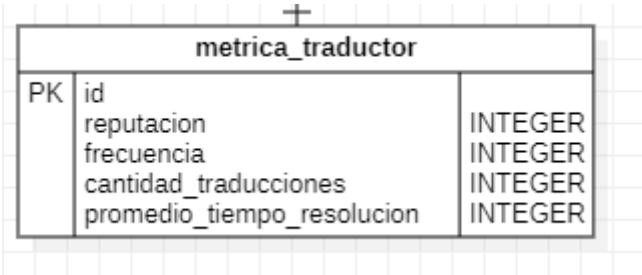
Segundo Parcial 2021

Segundo Parcial Modelo - 2021.pdf

Dudas

☐ Vale la pena poner los atributo el promedio_de_trabajo, cantidad_traducciones, frecuencia? Si no se pusieran, no sería menos performante hacer los calculos (por ejemplo ,para calcular la cantidad de traducciones habria que agarrar todas las traducciones y hacer un size)?



Campos que PODRÍA tener la tabla metrica_traductor

☐ Está bien crear una tabla archivo? Para determinar el archivo fuente y el final de la traducción

☐ Esta bien tener un balanceador de carga en microservicios? y si fuera asi, cómo manejamos el tema de las sesiones? ya que tenemos tokens

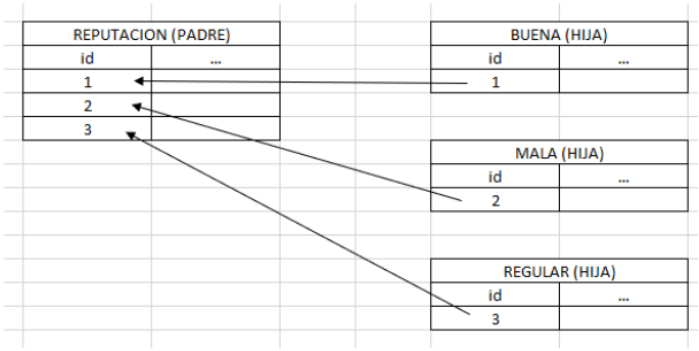
☐ Si o si un microservicio es un cliente pesado?

☐ En la pregunta 4 de arquitectura, no podría ser una cola de mensajes para el pago también? Como el caso que se había discutido en clase, sobre el telepase. Pensamos que el procesamiento de un pago deberia ser algo mas de estilo sincrónico para dejarle al usuario con la tranquilidad de que se pago fue recibido.

Notas

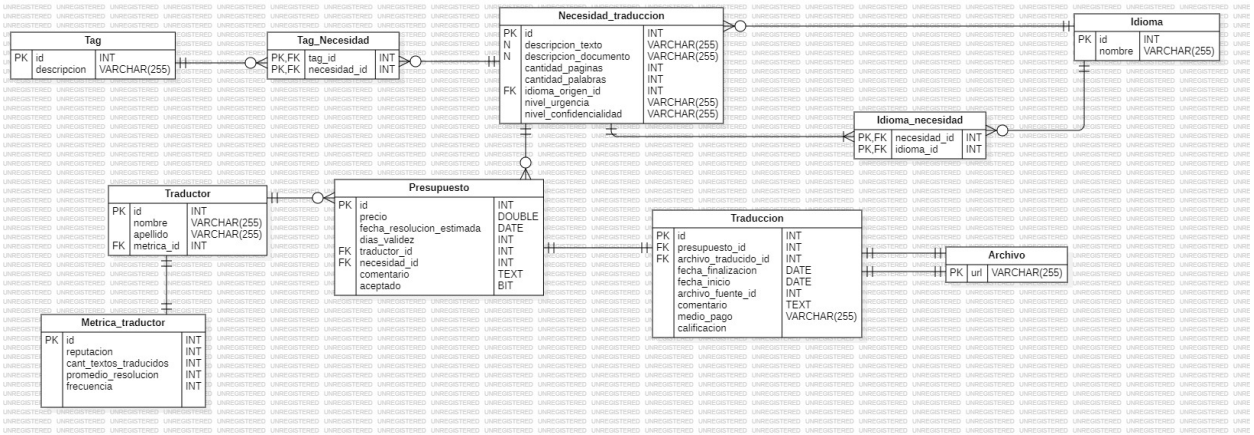
- FK siempre del many

- Text es más largo que VARCHAR
- JOIN ventajas: nos podemos traer de la clase padre, todas las hijas (porque tiene sus ids) y es útil ésta cuando las hijas tienen atributos en común



Ejemplo JOIN

Modelado de datos



- nivel_urgencia y nivel_confidencialidad decidimos modelarlo con un ENUM, ya que consideramos que no es un atributo que tenga gran frecuencia de cambio. Por este motivo, a la hora de modelarlo, se guardará en la tabla de Necesidad_traduccion como un campo de tipo VARCHAR, y se utilizará un converter para pasar de este tipo de dato al tipo de dato del ENUM y viceversa.
    - Otra alternativa hubiera sido hacer tablas para el nivel de urgencia y el de confidencialidad, para tenerlos previamente tipados ahi, pero no lo hicimos de esta manera ya que es menos performante debido a la necesidad de hacer joins, y además no son campos que varíen mucho en el tiempo.

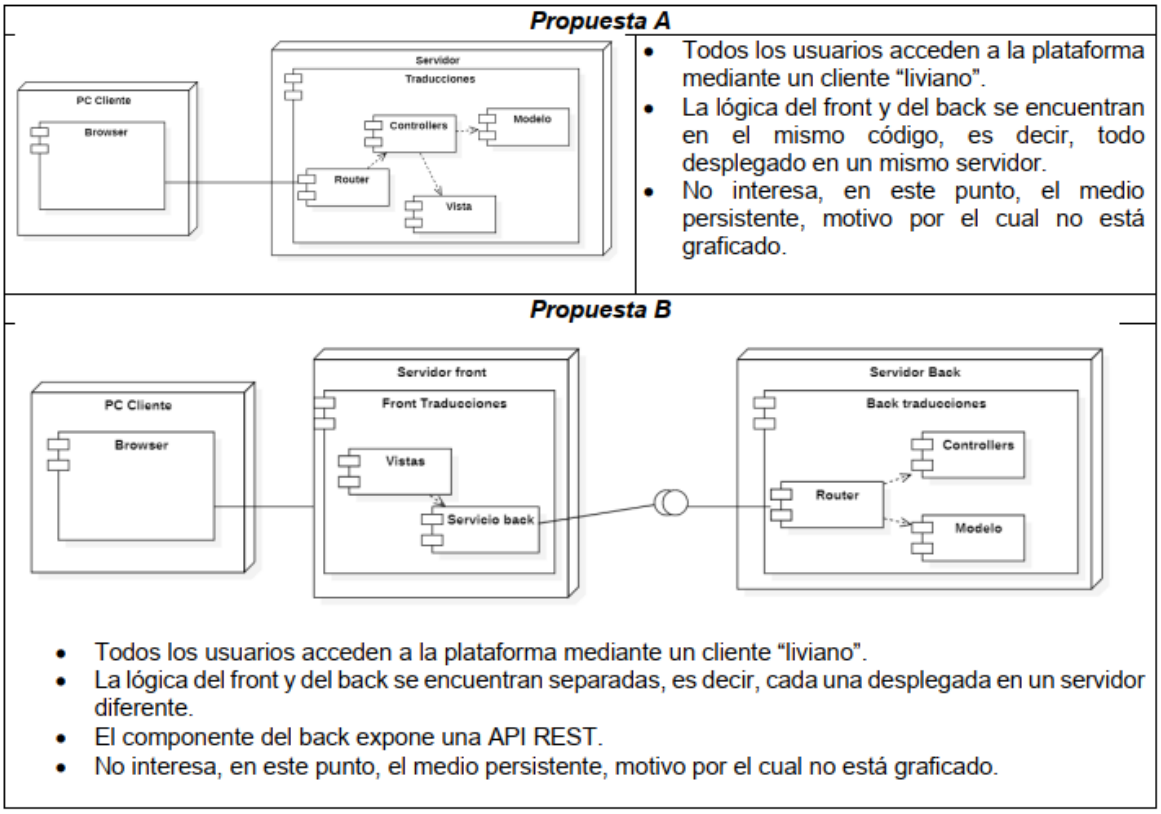
- La entidad necesidad_traducción debe ser persistida debido a que se debe permitir la búsqueda web de la misma
- Creamos una tabla traductor ya que consideramos necesario mantener una trazabilidad de quién genera el presupuesto
- Decidimos hacer una entidad Metrica_traductor para guardar toda la información relacionada con las estadísticas del mismo
- Creamos una entidad traducción que sea la oficialización de la necesidad_traducción con el presupuesto seleccionado

Arquitectura

> 1. Sabiendo que los traductores tendrán una frecuencia de uso alta del aplicativo y que los dueños de los textos y/o documentos no necesariamente tendrán la misma frecuencia de uso; ¿considera que podría ser beneficioso utilizar un SSO (Single Sign On)? ¿Por qué? Justifique adecuadamente su respuesta

RTA: No vale la pena utilizar un SSO dado que nuestro sistema no tiene muchos componentes y funcionalidades para las que se requiera logearse. No se especifica ningún requerimiento que establezca que para poder ser traductor y solicitar traducciones se requiera de inicio de sesión distintos usuarios. Además, este componente que gestiona los accesos puede fallar y perder todos los accesos a los sistemas relacionados.

> 2. Compare las siguientes propuestas de arquitectura en base a los siguientes atributos de calidad, detallando los aspectos (pivotes) que tuvo en cuenta para su comparación:
>
> - Performance
> - Mantenibilidad
> - Escalabilidad



	Propuesta A	**Propuesta B**
Performance	Es performante ya que no requiere del tiempo de respuesta del back y te devuelve la vista de manera más rápida y eficiente.	Esta propuesta no es tan performante ya que existe un tiempo de respuesta entre los servidores, pues los mismos deben integrar mediante API Rest.
Mantenibilidad	En caso de que se quisiera modificar algo del front unicamente o algo del back, se debe desplegar de nuevo el aplicativo.	Se encuentra desacoplado, resultando más efectivo para realizar cambios y desplegar sólo la parte del servidor correspondiente.
Escalabilidad	No es muy escalable dado que puede ocurrir que algunas partes de la aplicación escalen más rápidas que otras, por lo que conviene tener separada la lógica.	Podremos escalar cada parte por separado, tanto en back como el front.

> 3. En base a lo desarrollado en el punto anterior y sabiendo que, debido al escaso tiempo para la ejecución del proyecto, se tomó la

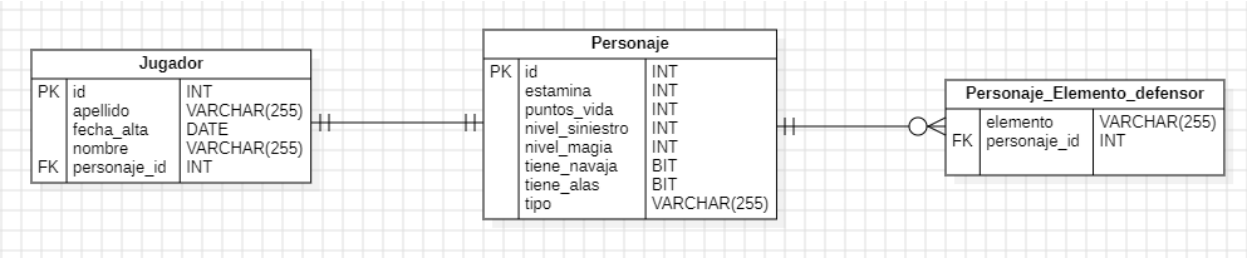
> decisión de “salir a producción” con un front que pronto cambiará, ¿con cuál de las dos propuestas se quedaría? ¿Por qué? Tenga en cuenta que puede seleccionar alguna de ellas y proponer un cambio o mejora

RTA: La B debido a que se integra mediante el API REST procurando una mejor comunicación que con el uso de un monolito, por ejemplo. Además, esta propuesta al tener las lógicas separadas se pueden desplegar por separando, aumentando la mantenibilidad y escalabilidad. De la misma manera, se podría aplicar un microservicio en vez de un monolito

> 4. Sabiendo que todos los medios de pagos candidatos a ser soportados en la plataforma permiten una integración mediante API REST y mediante cola de mensajes; ¿Qué estrategia de integración de las anteriormente mencionadas elegiría? ¿Por qué? Justifique adecuadamente su respuesta

RTA: Se podría utilizar una cola de mensajes asincrónica para la facturas de los pagos pero para la efectuar el pago, se puede realizar la misma de manera sincrónica con una API REST (como la de Mercado Pago). Tomamos estas decisiones teniendo en cuenta la experiencia de usuario, en pos de darle la seguridad a ellos que se efectuaron las transacciones.

Persistencia



- Para elemento defensor decidimos guardar la referencia de la clase ya que los tipos de elementos son completamente stateless, por ende no amerita que se persista en la

base de datos. Se usa un converter para transformar entre el string que se guarda en la base de datos y el tipo de dato correspondiente en el mundo de objetos.

```
public class ElementoDefensorConverter implements AttributteConverter<ElementoDefensor, String> {

    @Override
    public String convertToDatabaseColumn(ElementoDefensor elemento) {
        String elementoEnBase = null;

        if(elemento.getClass().getName().equals("Arco")) {
            elementoEnBase = "arco";
        }
        else if(medioDeNotificacion.getClass().getName().equals("Espada")) {
            elementoEnBase = "espada";
        }
        else if(elemento.getClass().getName().equals("Escudo")) {
            elementoEnBase = "escudo";
        }
        return elementoEnBase;
    }

    @Override
    public ElementoDefensor convertToEntityAttribute(String s) {
        ElementoDefensor elemento = null;

        if(s.equals("arco")) {
            elemento = new Arco();
        }
        else if(s.equals("espada")) {
            elemento = new Espada();
        }
        else if(s.equals("escudo")) {
            elemento = new Escudo();
        }
        return elemento ;
    }

}
```

- Para el caso particular del ladrón y del mago, decidimos mappear la herencia con la estrategia Single Table porque para recuperar la información no haría falta hacer joins y se podría especificar el tipo de cada personaje mediante la columna discriminadora Tipo_Personaje. Sin embargo, en el caso de que se agregaran más personajes con distintos atributos, se recomendaría cambiar la estrategia a un Joined (dado que las clases entre sí tienen elementos en común)