

---

# **Proyecto de procesamiento de lenguaje natural (3.0v)**

Definición de TFG para procesamiento de voz en Glifing S.L.

Jesús Manuel Nieto Carracedo

20 de abril de 2024



## Índice

<b>1. Descripción general del proyecto</b>	<b>2</b>
<b>2. Arquitectura del sistema</b>	<b>2</b>
<b>3. Modelo de datos</b>	<b>3</b>
3.1 Usuarios, roles, pauta . . . . .	3
3.2 Sesiones . . . . .	4
3.3 Activities . . . . .	5
3.4 Items . . . . .	5
3.5 Status . . . . .	6
<b>4. Componentes</b>	<b>6</b>
4.1 Ejemplo de una vista tipo índice(index) . . . . .	7
4.2 Ejemplo de una vista tipo cabecera-líneas . . . . .	7
<b>5. Vistas</b>	<b>8</b>
5.1 Usuarios . . . . .	8
5.1.1 Index principal de usuarios . . . . .	8
5.1.2 Editor de usuario . . . . .	9
5.2 Sesiones . . . . .	9
5.2.1 Index principal de sesiones . . . . .	9
5.2.2 Editor de sesiones . . . . .	9
5.3 Actividades . . . . .	9
5.3.1 Index principal de actividades . . . . .	9
5.3.2 Editor de actividades . . . . .	9
5.4 Items . . . . .	10
5.4.1 Index principal de items . . . . .	10
5.4.2 Editor de items . . . . .	10
5.5 Status . . . . .	10
5.5.1 Index principal de status . . . . .	10
5.5.2 Editor de estados . . . . .	10
<b>6 Inter-conexión de capas: Api</b>	<b>10</b>
6.1 Operaciones básicas de la Api . . . . .	11
6.2 Operaciones especiales de la Api . . . . .	11

## 1. Descripción general del proyecto

El objetivo general del proyecto es crear una plataforma que nos permita crear y editar un banco de pruebas de sesiones, con actividades de lectura, para obtener una serie de estadísticos y métricas que puedan ser analizados por plataformas externas.

Dado el tiempo asignado para la resolución de este proyecto, y sus fines formativos, nos centraremos en las partes de valor, es decir, en las capas de la aplicación que gestionan los datos relativos a la lectura, por ese motivo, no se pondrá excesivo foco en cuestiones como la seguridad, concurrencia en la edición, etc.. por ello se simplificarán las partes comunes y superfluas, que comúnmente suelen resolverse a través de rutinas, métodos, o funciones estandarizadas en los distintos **frameworks** utilizados.

Haremos uso de un **modelo de desarrollo incremental**, y para ello usaremos las diferentes tecnologías que se indican a continuación.

## 2. Arquitectura del sistema

Se generará un sistema basado en 3 grandes capas, basadas en el modelo **MVC**:

- **Persistencia:** Estará basada en una base de datos documental.
  - Se implementará con MongoDB
- **Controlador:** Para intercambiar información entre persistencia y las vistas del sistema, usaremos una **api-rest**.
  - Se implementará a través del framework backend django-rest-franmework el cual usará como lenguaje de servidor **python 3.x**.
- **Vistas:** Las vistas, tanto para **backend** como para **frontend** implementarán una programación orientada a componentes, implementando el modelo **MVVM**:
  - Se implementará a través del framework frontend angular, donde usaremos **html5** y **typescript**.
  - Para la maquetación usaremos un framework CSS llamado bootstrap
- **Sistemas:** La pila estará compuesta por:
  - **Servidor:** Usaremos linux en una versión basada en **debian**, preferiblemente **Ubuntu**.
  - **Servicio:** Usaremos como servidor **Apache2**, el cual usará el módulo **wsgi** para poder ejecutar código **python**

- **Virtualización:** Usaremos virtualización basada en docker que tendrá el siguiente despliegue:
- **Código:** Se desplegará en el todo el código fuente, pero nada de persistencia, este modelo de arquitectura persigue ser altamente escalable tanto horizontal, como verticalmente.
- **Documentación:** La misma se hará a través de estos sistemas:
  - **Código fuente:** A través de **git**, como plataforma de intercambio de código fuente usaremos github.com
  - **Api:** A través de swagger
  - **Proyecto-cliente:** A través de trello

### 3. Modelo de datos

#### 3.1 Usuarios, roles, pauta

Los usuarios podrán disponer de unos roles, y pueden tener los 3 tipos, o solo alguno de ellos, además en caso de tener el rol de entrenado, tendremos que recoger algún campo adicional. Solo podrá tener el usuario un rol activo al tiempo, esto se hace para facilitar la carga de vistas. Es decir, si un usuario tiene el rol de editor y entrenado, y accede al sistema, tendrá que poder determinar el sistema que rol está usando en ese momento para poder mostrar su vista adecuada.

- Tipos y acciones, será una jerarquía donde el primero podrá hacer todo lo que hacen los siguientes más sus acciones:
  - **1. Administrador:**
    - \* Administrar totalmente al resto de usuarios.
    - \* Acceder y revisar configuraciones en la plataforma.
    - \* Acceso a resultados de usuarios.
  - **2. Editor:**
    - \* Administrar totalmente los paquetes, sesiones y actividades de la plataforma.
    - \* Asignar carga de trabajo a los entrenados
  - **3. Entrenado:**
    - \* Ejecutar las cargas de trabajo que tenga asignadas, en el orden indicado en una lista.
- Información que necesitamos recoger de cada uno de nuestros usuarios:
  - **Comunes:**
    - \* **identificador:** id único. (Usar formato MongoDB)

- \* **username**: nombre de usuario, llevará una máscara de entrada que solo permita el formato nombre.apellido, en el caso de repetirse (existir algún otro en el sistema), se tendrá que añadir un número secuencial. El campo será una cadena.
  - \* **username\_glifing**: será el nombre de usuario que correlaciona con su usuario Glifing, este campo será del mismo tipo que el **username**, pero puede encontrarse vacío, si el usuario no tiene correlación alguna.
  - \* **first\_name**: nombre, cadena.
  - \* **last\_name**: apellidos, cadena.
  - \* **email**: correo electrónico, se deberá verificar que se ha introducido un email correcto, cadena.
  - \* **password**: clave de acceso, usaremos un algoritmo de cifrado que provea el framework Django.
  - \* **avatar**: será una imagen asociada al entrenado. Se almacenará en la base de datos.
- **Entrenado**:
    - \*  **cursos**: será una lista de tipo “curso” para los cuales queremos dar acceso a entrenamiento.
  - **curso**: - **id**: identificador del curso. - **entrenamientos**: lista de sesiones de entrenamiento de tipo “entrenamiento”.
  - **entrenamientos**:
    - \* **date**: fecha en la cual se tiene que realizar la sesión.
    - \* **session\_id**: identificador mongodb de la colección de sesiones.
    - \* **results**: Lista de resultados, de tipo “resultado”, de la realización de la sesión de entrenamiento. (Será definido en próximos incrementos)

### 3.2 Sesiones

Los editores y administradores, tienen que poder editar **sesiones** de trabajo, para ello es necesario disponer de una colección de documentos llamados sesiones. Las sesiones serán el primer nivel de agrupamiento, una sesión deberá tener una lista de actividades.

Las listas de **actividades**, son los distintos ejercicios que tienen que realizar los entrenados. Un ejemplo puede ser una práctica de universidad, la cual contendrá sub-apartados, o un examen que contendrá diferentes preguntas, pues bien a esto lo denominaremos actividades. Las actividades a su vez contendrán elementos que usaremos para configurar la actividad, por tanto dentro de una actividad, habrá una lista de elementos que llamaremos **items**.

Aunque los items, podrían ser diferentes elementos multimedia [“texto”, “audio”, “video”, “imagen”], el proyecto solo soportará texto, para simplificarlo, por tanto no será un requerimiento mínimo del

proyecto, implementar soporte nada más que para texto, ya que esto implicaría tener que generar un módulo para la gestión de inventario de ficheros multimedia asociados.

- Información que necesitamos recoger para las sesiones:
  - **identificador**: id único. (Usar formato MongoDB)
  - **glifing\_session\_id**: id que será un entero, el cual usaremos para la correlación con las sesiones Glifing. En caso de que la sesión no tenga correlación con Glifing, dejar este campo nulo.
  - **name**: nombre de la sesión, tipo cadena.
  - **description**: descripción de la sesión, tipo cadena.
  - **status\_id**: debe reflejar un estado de la colección de estados, es decisión de diseño crear una colección llamada estatus, o bien indicarlo en este campo. Buscar la definición en el apartado correspondiente. A tener en cuenta, una sesión en estado **En pruebas** no se podrá utilizar desde la vista de **player** para garantizar que un entrenado juegue una versión no estable. Y del mismo modo una sesión **activa** tiene que marcarse como “en pruebas\*\* para que pueda ser editable.
  - **activities**: Lista de tipo actividades.
  - **createdUser\_id**: Almacenar el id del usuario que creó la sesión.
  - **modifiedUser\_id**: Almacenar el id del usuario que modificó la sesión.
  - **created**: Almacenar la fecha y hora en la cual se creó la sesión.
  - **modified**: Almacenar la fecha y hora en la cual se modificó la sesión.

### 3.3 Activities

Los documentos de tipo **activities** se encuentran en una lista de actividades en cada sesión. Guardarán la información de configuración que deberá cargar el player, para presentar y coordinar el entrenamiento de cada entrenado.

Las distintas actividades que vamos a realizar se irán detallando en las próximas versiones de este documento.

### 3.4 Items

Los **items** son los elementos de configuración que irán en una lista dentro de las distintas actividades.

Las distintas actividades que vamos a realizar se irán detallando en las próximas versiones de este documento.

### 3.5 Status

Colección de estados posibles para un documento, pudiendo ser:

- **Activo:** El documento que contenga este estado no se podrá utilizar para entrenar, o de forma definitiva. Un documento “activo” solo se podrá usar en modo lectura, nunca en modo escritura, habrá que cambiar primero el estado **En pruebas**. Usaremos este sistema para garantizar que no hay un usuario entrenando por ejemplo en una sesión, cuando alguien está editando la misma.
- **En pruebas:** El documento solo se podrá editar, estará en modo lectura/escritura.

## 4. Componentes

Dado que se está usando un **patrón de diseño basado en componentes**, se tendrá que diseñar cada vista, con especial cuidado en no repetir el código de visualización. Por ejemplo, dado que tendremos que hacer una vista para administrar sesiones, y otra para administrar usuarios, otra seguramente para poder administrar la pauta de entrenados y sesiones, parece lógico generar un componente que permita administrar una lista de colecciones, donde cada elemento tenga unas operaciones a realizar por elemento y una cabecera para las agregaciones de los distintos elementos de cada colección.

Del mismo modo, crear componentes de filtro que se adapte a la naturaleza de la colección administrada, parece mejor idea que crear un diseño de filtro diferente para cada colección.

Dado que vamos a tener que crear muchos elementos de tipo “cabecera-líneas”, (usuarios, sesiones, actividades, pauta, etc...), en esta primera versión de la especificación será importante definir bien, que componentes deberemos crear, que podamos personalizar en sus entradas y salidas, y nos sirvan para manejar distintos tipos de entidades.

Al crear los componentes, tener en cuenta que se pueden, y se deben utilizar librerías ya desarrolladas por terceros, como por ejemplo: - **jHtmlArea: WYSIWYG Html Editor jQuery Plugin**, para crear un editor de texto, para la entrada de datos. Se puede descargar la librería, y enlazar el código del proyecto, para ver como se puede personalizar. - **bootstrap: Plantilla Bootstrap**, plantilla bootstrap donde se puede ver controles de tabla, tipografías, paneles, gráficos, formularios de ejemplo, etc... Es muy buena idea reutilizar o bien esta, o bien cualquier otra, pero la misma para todo el proyecto, y de esta “paleta” de muestra, construyas los distintos componentes de la aplicación.

#### 4.1 Ejemplo de una vista tipo índice(index)

+

×

↑

+ Todos

+ Página

- Todo

- Página

Mostrar

10

▼

elementos

	ID	Nombre	Operaciones
<input type="checkbox"/>	1,595	Alfa 01 MAY_AR	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	1,636	Alfa 01 MAY_ES	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	1,596	Alfa 02 MAY_AR	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	1,637	Alfa 02 MAY_ES	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	1,597	Alfa 03 MAY_AR	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	1,638	Alfa 03 MAY_ES	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	1,598	Alfa 04 MAY_AR	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	1,639	Alfa 04 MAY_ES	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	1,599	Alfa 05 MAY_AR	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	1,640	Alfa 05 MAY_ES	<div><div></div><div></div><div></div></div>

Mostrando 1 a 10 de 100 registros

Previo

1

2

3

4

5

...

10

Siguiente

**Figura 1:** Ejemplo de una vista tipo índice(index)

## 4.2 Ejemplo de una vista tipo cabecera-líneas

[illegible]

**Figura 2:** Ejemplo de una vista tipo cabecera-líneas



## 5. Vistas

Todas las vistas deberán funcionar, de tal forma que cada vez que se modifique un elemento, se grabará el cambio en la base de datos, es decir, no vamos a crear un botón que tenga que presionar el usuario que esté editando la vista, cuando en el ciclo de vida de la página, se detecte un cambio, se guardará todo el documento de la colección.

### 5.1 Usuarios

Tal cual se ha desarrollado para el editor de sesiones actual de Glifing.

#### 5.1.1 Index principal de usuarios

Deberá presentar una lista de usuarios dados de alta en el sistema, donde aparezcan sus campos más representativos. Esta lista podrá tener operaciones **C.R.U.D.** tanto a nivel de documento, como poder seleccionar varios documentos a la vez.

- Operaciones por documentos:
  - **Editar:** nos deberá permitir editar la información del documento.
  - **Eliminar:** nos deberá preguntar antes si queremos eliminar el documento, para eliminarlo posteriormente en caso de ser confirmado.
  - **Ver:** nos deberá permitir visualizar la información del documento. (Es buena idea crear un solo componente para editar documentos, que actúe en modo solo lectura, o modo edición)
  - **Selección:** marcar o desmarcar el documento seleccionado.
- Operaciones de agregación de documentos:
  - **Añadir:** nos permite crear un documento nuevo.
  - **Eliminar:** nos deberá permitir eliminar varios elementos a la vez, previa confirmación a través de un mensaje que nos indique el número de documentos que vamos a eliminar.
  - **Filtrar:** nos permitirá realizar filtros sobre los documentos.
  - **Paginado:** Tenemos que mostrar los documentos paginados, mostrando un máximo de 10 elementos por página por defecto, pero podremos elegir el paginado de 10,25,50,100 elementos. Además tendremos que disponer de botones que permitan seleccionar los elementos de todas las páginas, o quitar los elementos seleccionados de todas las páginas, así como seleccionar o eliminar la selección de página a página.

Además tendrá que disponer en una de sus columnas, de iconos que identifiquen los roles de los que dispone, se puede por ejemplo escribir las dos primeras letras de cada tipo, dentro de un círculo con un color diferente cuando está activa, y en gris cuando no tiene ese rol. Por ejemplo AD,ED,EN.

### **5.1.2 Editor de usuario**

Para todas las vistas de edición, que tengan solo un documento, sin sub-documentos asociados, se mostrará una capa modal la cual mostrará los controles de los documento editables. Esta capa tendrá un botón para poder cerrar la misma, y cada vez que se cierre la capa de edición, deberá verse reflejado el cambio en la vista de índice.

## **5.2 Sesiones**

Tal cual se ha desarrollado para el editor de sesiones actual de Glifing.

### **5.2.1 Index principal de sesiones**

Vista de tipo lista, como se ha descrito para la vista de usuarios.

### **5.2.2 Editor de sesiones**

Vista de tipo cabecera y líneas. En la cabecera tendrá el componente de edición de un documento (ficha de sesiones) y en las líneas se editarán una vista index de las actividades para esa sesión.

## **5.3 Actividades**

Tal cual se ha desarrollado para el editor de sesiones actual de Glifing.

### **5.3.1 Index principal de actividades**

Iría integrada en las líneas del detalle del editor de sesiones.

### **5.3.2 Editor de actividades**

Será de tipo cabecera-líneas, donde la cabecera será un componente de tipo ficha de la actividad, y las líneas será una vista index de los items que contiene la actividad.

## 5.4 Items

Tal cual se ha desarrollado para el editor de sesiones actual de Glifing.

### 5.4.1 Index principal de items

Ir  integrada en las l neas del detalle del editor de actividades.

### 5.4.2 Editor de items

Ser  un formulario modal del componente tipo ficha donde aparece el item.

## 5.5 Status

Seguir  el mismo patr n descrito anteriormente.

### 5.5.1 Index principal de status

Vista de tipo lista, como se ha descrito para la vista de usuarios.

### 5.5.2 Editor de estados

Ser  un formulario modal del componente tipo ficha donde aparece el estado.

## 6 Inter-conexi n de capas: Api

Para la comunicaci n entre la persistencia del sistema y las vistas, se realizar  a trav s de una **api** del tipo **Api Rest**, la cual ser  implementada a trav s de:

- **Lenguaje Servidor:** Se realizar  a trav s de **Python**, sobre el framework **Django-rest**.
- **Documentaci n:** Se escribir  a trav s de **Swagger**.

Entendemos por persistencia, toda informaci n que necesite perdurar en el tiempo:

- **Base de datos:** Se implementar  a trav s de un sistema **MongoDB**, cuyas librer as se conecten a la api.

- **Almacén de ficheros:** Todo tipo de ficheros del sistema, si los hubiera, deberán ser almacenados en una ubicación que pueda ser gestionada directamente por la api, de forma que se pueda definir a través del fichero de configuración de la api. De tal forma que, si se decide cambiar la ruta, o emplazamiento, escalar, etc... sea transparente para las vistas. Por ejemplo, una vista que quiere mostrar, o almacenar un fichero, hará la solicitud a un endpoint de la **Api**, el cual devolverá lo solicitado. Si el administrador de la Api, decide cambiar el volumen físico, usar varias ubicaciones para disponer de redundancia, etc... esta parte quedará encapsulada en la implementación de la Api.
- Etc..

El objetivo de implementar la infraestructura basadas en el sistema de capas descrito hasta el momento, persigue independizar al sistema en piezas modulares estándar, que cumplan las siguientes características:

- **“Caja negra”:** Cada pieza de la infraestructura tendrá entras y salidas claramente diferenciadas.
- **Escalabilidad:** Podremos implementar cada una de nuestras piezas en una máquina física separada, o en una sola máquina para todo el conjunto, esto nos permitirá disponer de:
  - **Escalado horizontal:** Podremos distribuir en varios servidores físicos, la ejecución de la lógica de nuestro sistema, balanceando la carga de trabajo, si en un futuro, la demanda de usuarios así lo requiriese.
  - **Escalado vertical:** Podríamos aumentar la potencia de la máquina física, si la demanda de usuarios así lo requiriese.

## 6.1 Operaciones básicas de la Api

Todos los modelos definidos en el punto 3, deberán operarse a través de la Api.

## 6.2 Operaciones especiales de la Api

En sucesivas versiones se irán definiendo.