Proyecto de procesamiento de lenguaje natural (1.1 v)

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

Jesús Manuel Nieto Carracedo

20 de abril de 2024

Índice

1. Descripción general del proyecto **2**

2. Arquitectura del sistema **2**

3. Modelo de datos **3**

3.1 Usuarios, roles, pauta 3

4. Componentes **4**

4.1 Ejemplo de una vista tipo índice(index) 5

4.2 Ejemplo de una vista tipo cabecera-líneas 6

5. Vistas **6**

5.1 Usuarios 6

5.1.1 Index principal de usuarios 6

5.1.2 Editor de usuario 7

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**:

- **Persistenca**: 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**

  - **Virtualiazación:** 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 de 30 posiciones.
    * **username_glifing**: será el nombre de usario 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 de 30 posiciones.
    * **last_name**: apellidos, cadena de 30 posiciones.
    * **email**: correo electrónico, se deberá verificar que se ha introducido un email correcto, cadena de 75 posiciones.
    * **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)

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 enlazo 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)

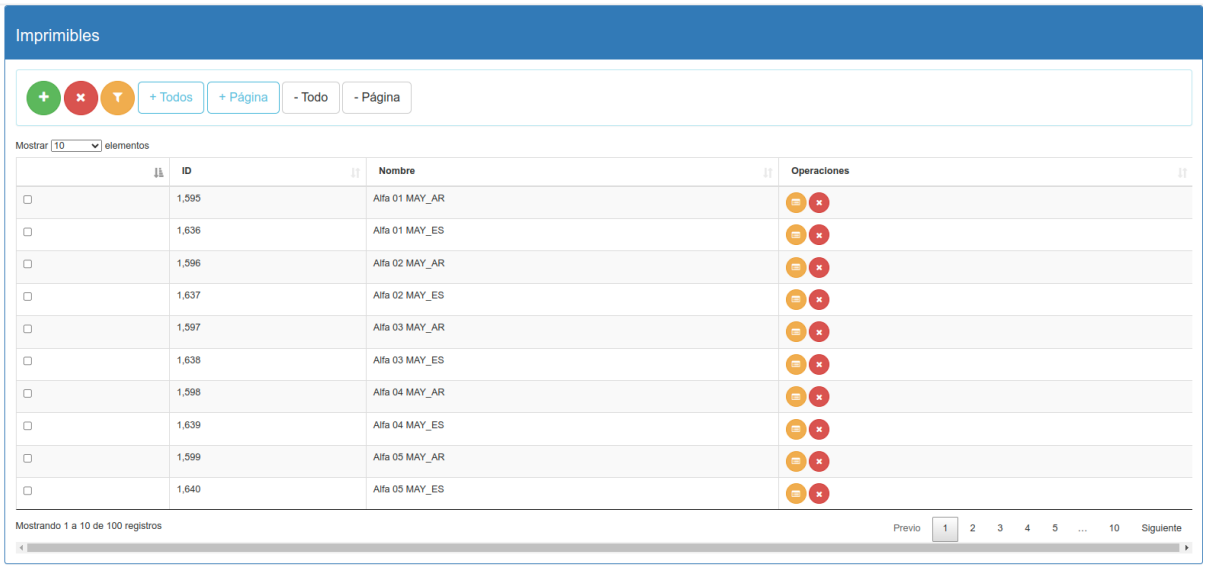

Figura 1: Ejemplo de una vista tipo índice(index)

4.2 Ejemplo de una vista tipo cabecera-líneas

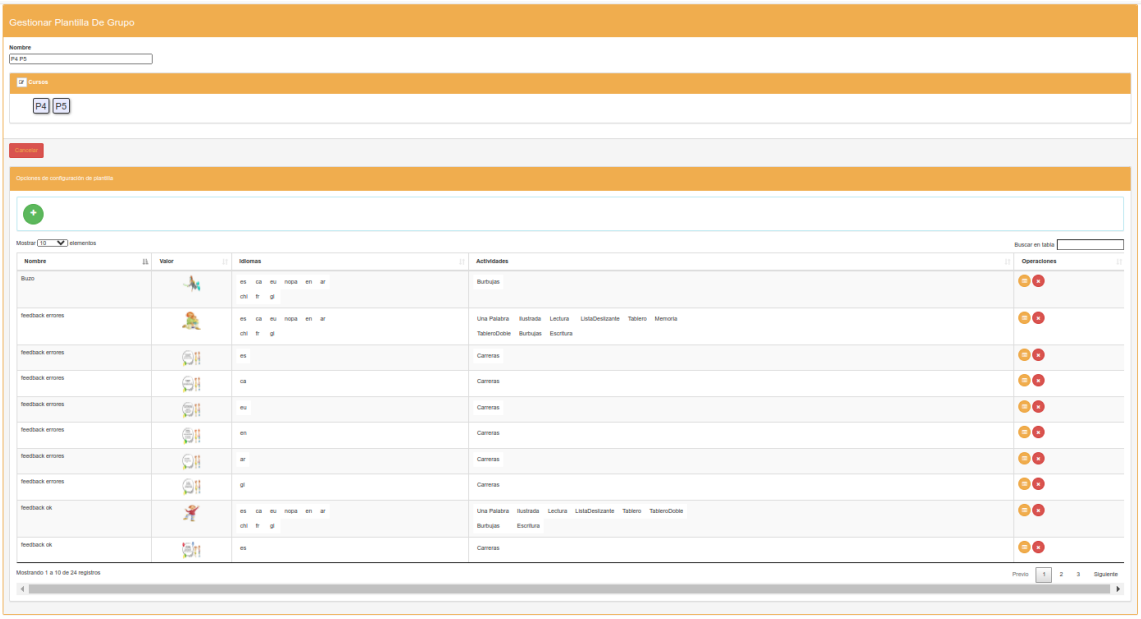

Figura 2: Ejemplo de una vista tipo cabecera-líneas

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

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.