

# *Define*

*Sistema de software para anotación semántica*

## Manual técnico

Desarrollo: Antonio Nuñez

Diseño y dirección: Ignacio Arroyo-Fernández

Revisión: Gerardo E. Sierra Martínez

*Grupo de ingeniería lingüística (GIL-UNAM)*



# MANUAL TÉCNICO

Sistema DEFINE

## Contenido

Introducción .....	3
Objetivos .....	3
Tecnologías utilizadas .....	4
Términos utilizados .....	5
Casos de uso .....	6
Diseño del sistema .....	7
Base de datos (modelo) .....	7
Controlador .....	8
Particular.py .....	8
Modulo: Inicio .....	8
Modulo: Importar .....	10
Modulo: Crear grupos .....	11
Modulo: Crear interfaz de relación .....	18
Modulo: Crear historial de palabras clave .....	22
Modulo: Crear historial de definiciones usadas en un grupo .....	24
Modulo: Obtener porcentaje de uso para cada grupo .....	26
Modulo: Crear la relación .....	27
General.py .....	29
Modulo: Inicio .....	29
Modulo: Importar .....	31
Modulo: Crear grupos .....	32
Modulo: Crear interfaz de relación .....	39
Modulo: Crear historial de palabras clave .....	45
Modulo: Crear historial de definiciones usadas en un grupo .....	46
Modulo: Obtener porcentaje de uso para cada grupo .....	49
Modulo: Crear la relación .....	50
Interfaz.py .....	51
Modulo: Inicio .....	52
Modulo: Genera archivo .....	52
Default.py .....	57
Vistas .....	58
Vistas – Modo particular .....	58

Vista: Inicio.....	58
Vista: Importar .....	59
Vista: crearGrupo .....	59
<b>Funciones JavaScript:</b> .....	60
Vista: vistaWordNet .....	61
<b>Funciones JavaScript:</b> .....	62
Vista: relación.....	62
<b>Funciones JavaScript:</b> .....	64
Vistas – Modo General .....	74
Vista: Inicio.....	74
Vista: Importar .....	75
Vista: crearGrupo .....	76
<b>Funciones JavaScript:</b> .....	76
Vista: vistaWordNet .....	77
<b>Funciones JavaScript:</b> .....	78
Vista: relación.....	79
<b>Funciones JavaScript:</b> .....	80
Vistas- Interfaz administrativa .....	91
Vista: Inicio.....	91
Instalación .....	92
Futuras mejoras .....	94

## Introducción

En el Grupo de Ingeniería Lingüística se está llevando a cabo un trabajo de etiquetado, el cual consiste en tener una lista de definiciones ligadas a un término e ir relacionándolas con grupos semánticos que se han ido creando.

Actualmente el proceso se realiza utilizando hojas de cálculo en Excel, donde se tienen las definiciones y los grupos, sin un orden específico. La manera en la cual se organizan los archivos no está estandarizada además de que el trabajo es tardado y pueden existir fallas humanas ya que no hay una manera de validar los datos que escriben.

Se solicita el apoyo de un sistema que brinde la certeza de que los datos que se introducen son los correctos y que apoye al proceso para que sea más rápido, adicionalmente se brindará ciertos apoyos a los etiquetadores.

La propuesta es crear un sistema web en el cual se cargarán los archivos con las definiciones y cada etiquetador podrá ingresar (con un previo registro) para poder realizar su trabajo.

El sistema contará con un control de acceso y le brindará al usuario los apoyos necesarios para facilitar su labor, dicho sistema se implementará primero en el GIL y posteriormente se ampliará su uso para que los datos que registre sean de mayor utilidad y más diversos.

El presente documento tiene como objetivo documentar el desarrollo del sistema para su futura modificación.

## Objetivos

El sistema permitirá el registro de nuevos etiquetadores, para ello utiliza el correo electrónico como identificador único y el nombre como identificador adicional. Los etiquetadores pueden modificar sus contraseñas y recuperarlas en caso de que sea necesario.

El sistema permitirá la carga de datos desde un archivo separado por comas, este archivo contiene todas las definiciones de un término, en el momento en que sean registradas estarán disponibles para todos los etiquetadores.

El sistema cuenta con dos modos de trabajo, el primero es el “Particular”, en este modo el etiquetador selecciona solo un término con el cual trabajará. El segundo modo, llamado “General”, permite la elección de más de un término con el cual se trabajará.

El sistema generará listas aleatorias para que sean etiquetadas, para esto el etiquetador deberá seleccionar un modo de trabajo y el /los términos con los que trabajará. La lista tiene un tamaño de 10 elementos.

El sistema permitirá la opción de crear grupos semánticos utilizando WordNet (con opción de ampliarse a otras herramientas) o grupos personalizados en el caso de que las herramientas no den una opción viable según el criterio del etiquetador.

El sistema mostrará las definiciones de la lista aleatoria y los grupos, el etiquetador debe de seleccionar el grupo(s) que considere adecuado e introducir las palabras clave que lo llevaron a esa conclusión. Este proceso está validado.

El sistema brindará al etiquetador tres apoyos principales: un historial de las definiciones que a utilizado en cierto grupo (las últimas 10), un historial de las palabras clave que otros usuarios han utilizado para etiquetar cierta definición (ordenadas por frecuencia de uso) y el porcentaje de usuarios que han utilizado un grupo.

Finalmente el sistema contará con una interfaz administrativa desde la cual se descargarán los archivos generados con el etiquetado, las opciones de la interfaz se ampliarán en el futuro.

### Tecnologías utilizadas

**Python 2.7:** El framework que utilizamos está basado en esta versión de Python, solo se utiliza la parte estructurada de Python.

<https://www.python.org/download/releases/2.7/>

**Web2Py:** Es un framework que nos permite trabajar con Python en un entorno web, permite el desarrollo de aplicaciones web de manera simple y segura. Maneja un esquema del modelo-vista-controlador donde el modelo es la base de datos, las vistas son los archivos html que son llamados por el controlador, funciones que hacen la parte lógica de las vistas. Se utiliza la versión de desarrollador ya que nos permite utilizar bibliotecas instaladas por nosotros.

<http://www.web2py.com/>

**WordNet:** Es un corpus que utilizamos para sugerir grupos a los etiquetadores y que ellos los utilicen, sirve para estandarizar los nombres de los grupos.

<https://wordnet.princeton.edu/>

**Natural Language Toolkit:** Nos brinda la interfaz para interactuar con el corpus de WordNet.

<http://www.nltk.org/>

Con excepción de WordNet el resto de las herramientas deben de ser instaladas con el fin de poder utilizar / modificar el programa.

WordNet no se necesita instalar ya que para eso se utiliza nltk.

Vea o descargue el software desde: <https://github.com/iarroyof/define-semantic-annotation>

## Términos utilizados

En el presente documento se utilizarán diversos términos que son necesarios especificar para el entendimiento del mismo:

**Término:** Un término, como se puede deducir, es solo la referencia a un conjunto de definiciones, en la mayoría de los casos es igual al nombre de archivo que se cargará al sistema.

**Definición:** Un término contiene varias definiciones, éstas son presentadas al etiquetador para que las relacione con un grupo semántico.

**Grupo semántico:** Grupo con el cual se puede relacionar una definición ya que pertenece a él-

**Etiquetar:** Proceso por el cual un etiquetador relacionará una definición con un grupo semántico.

**Etiquetador:** Es el usuario principal del sistema, es la persona que etiquetara las definiciones con los grupos.

**Palabras / Frases clave:** Cuando se está etiquetando una definición se tiene en cuenta el contenido de la misma, ciertas palabras o frases hacen que el etiquetador seleccione un grupo, las debe de ingresar para que el sistema sepa cuales fueron esas palabras.

**No informativo:** Cuando una definición no da la información suficiente para relacionarse con un grupo se dice que es “No informativo”

## Casos de uso

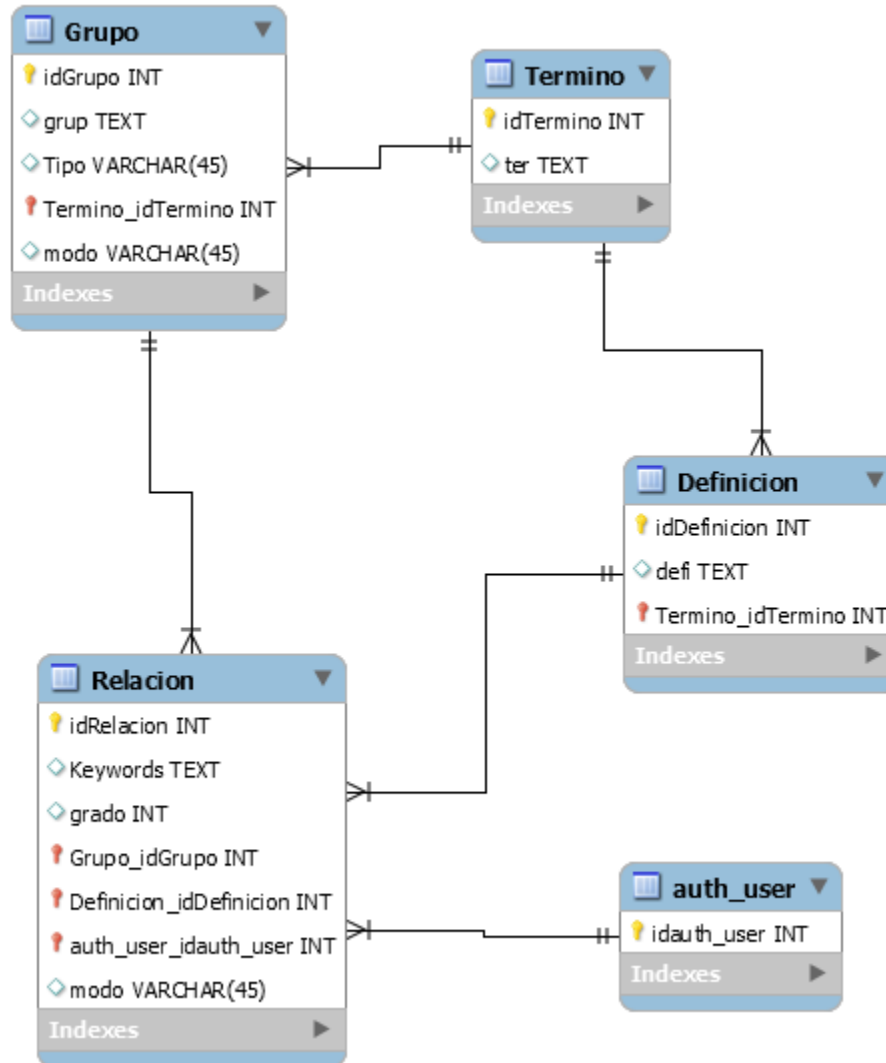
- 1- Registro de usuarios: El caso de uso comienza cuando el usuario entra al sistema y elige la opción "Sign Up", el sistema preguntará por los datos del usuario tales como el nombre, los apellidos, la dirección de correo electrónico y la contraseña. El sistema validará los datos y si son correctos mandará un correo de verificación, el usuario verificará su correo y podrá ingresar al sistema.
- 2- Recuperación de contraseña: El caso de uso comienza cuando el usuario da clic en "Recuperar contraseña", el sistema preguntará por el correo electrónico, el usuario lo introducirá y si es válido el sistema enviará un correo con las instrucciones para reiniciar su contraseña.
- 3- Inicio de sesión: El caso de uso comienza cuando el usuario entra al sistema y selecciona la opción "Log in" o un modo de trabajo, el sistema solicita los datos de acceso y si son válidos permite el acceso del usuario.
- 4- Registro de un nuevo término: El caso de uso comienza cuando el usuario está en la página de inicio de algún modo de trabajo y selecciona "Registrar nuevo término", el sistema muestra los campos necesarios para subir un archivo, el usuario los llena y si todo es correcto el sistema registra el término y las definiciones que se incluyen, además crea un grupo llamado "No informativo".
- 5- Creación de un nueva lista aleatoria: El caso de uso comienza cuando el usuario está en la página de inicio de algún modo de trabajo y selecciona un término o términos para trabajar, el sistema lee esos término y genera una lista aleatoria.
- 6- Creación de grupos: El caso de uso comienza cuando el usuario está en la vista de "relación" y selecciona "Crear grupos", el sistema abre una nueva ventana y muestra las opciones sugeridas por WordNet, el usuario selecciona tantos como quiera y da clic en "Crear grupos", el sistema los registra en la base de datos y refresca la página de "relación".
- 7- Creación de grupos personalizados: El caso de uso comienza cuando el usuario está en la vista para crear grupos sugeridos por WordNet y ninguno se ajusta a sus necesidades, entonces da clic en "Crea grupo personalizado", el sistema muestra la interfaz para el registro de grupos, el usuario llena los campos, el sistema los valida y los ingresa a la base de datos.
- 8- Visualización de los historiales: El caso de uso comienza cuando el usuario selecciona algún historial, el sistema lo crea y lo muestra.
- 9- Etiquetar: El caso de uso comienza cuando el usuario tiene una lista generada y entra a "relación", el sistema muestra el término con el que se está trabajando, la definición, los grupos, los historiales (a petición) y un campo para las palabras clave, el usuario llena estos campos, el sistema los valida y si son correctos los envía a la base de datos.
- 10- Obtener el contenido de la base de datos: El caso de uso comienza cuando el usuario administrador entra a la interfaz administrativa, el usuario selecciona el reporte que desea y el sistema lo crea.



## Diseño del sistema

Como ya se dijo, Web2Py trabaja con el modelo-vista-controlador, empezaremos con el modelo

### Base de datos (modelo)



### Localización del código: db.py

Descripción del código: Básicamente aquí solo se crea la base de datos, se puede consultar la documentación de Web2Py para más información.

### Consideraciones que se tienen que tener sobre la base de datos:

- Sin excepción, siempre debe de existir el registro ID: 1 ter: Nulo en la tabla termino.
- El campo modo en varias tablas solo puede tener dos valores en este momento, 1 o 2, el 1 es el modo particular y el 2 es el modo general.
- El campo tipo en la tabla grupo solo tiene tres valores posibles: Sistema que son los grupos creados por el sistema, en este caso para los “No informativos”; Custom para los grupos

creados por el usuario y “WordNet” para los grupos creados usando la sugerencia de esa herramienta, en el futuro este campo tendrá más valores posibles.

### Ejemplos de modificación:

Si se quiere agregar un campo a la tabla:

```
db.define_table('termino', Field('ter', 'text'), Field('nuevo_campo'))
```

Si se quiere agregar una nueva tabla:

```
db.define_table('nuevo', Field('nuevo'))
```

Código:

---

```
db.define_table('termino', Field('ter', 'text'))
db.define_table('definicion', Field('defi', 'text'), Field('termino_id', 'reference termino', default=1))
db.define_table('grupo', Field('grup'), Field('termino_id', 'reference termino'), Field('tipo'), Field('modo'))
db.define_table('relacion', Field('keywords'), Field('grado'), Field('grupo_id', 'reference grupo'),
Field('definicion_id', 'reference definicion'), Field('usuario', 'reference auth_user'), Field('modo'))
db.define_table('temporal', Field('usuario_id', 'reference auth_user'), Field('definicion_id', 'reference
definicion'), Field('modo'))
```

---

## Controlador

En este programa se tienen tres controladores: particular.py, general.py e interfaz.py.

### Particular.py

Se encarga del modo “Particular”, tiene 16 funciones de las cuales 8 interactúan directamente con el usuario ya sea como vistas o como el atributo ‘action’ de los formularios. Las otras 8 proveen apoyo para que el sistema sea más modular. Se procederá a explicar cada módulo del sistema y el código fuente relacionado.

#### Modulo: Inicio

**Descripción:** El modulo inicio solo se compone de una función: “inicio”, su función principal es comprobar si el etiquetador tiene una lista pendiente, si no es así crea las variables necesarias para la vista.

#### Código fuente relacionado con el método:

---

**Función:** inicio

**Variables:**

- **usuario\_nombre:** Toma su valor de la base de datos y de la sesión actual, se utiliza para mostrar el nombre del usuario en la vista.
- **termino:** La vista necesita que el usuario seleccione un término, para ello se deben de listar los términos existentes, esta variable almacena esa información, la obtiene de la base de datos.

**Vista relacionada:** [particular/inicio](#)

**Tarea:** Crea las variables que se utilizarán en la interfaz principal como lo son el nombre del etiquetador y la lista de términos existentes entre los cuales el etiquetador seleccionará uno para crear una lista aleatoria para etiquetar.

#### **Explicación del flujo:**

Primero buscamos si el etiquetador ya está trabajando en algunas lista de definiciones, si no es así entonces crea las variables usuario\_nombre y termino después revisa si el etiquetador termino de etiquetar su lista antes de cargar esta vista (argumentos en la url), si es así entonces muestra un mensaje indicando ese caso.

Si el usuario ya está trabajando en una lista, entonces lo manda a la vista de relación.

#### **Adornos adicionales:**

- **@auth.requires\_login():** Indica que es necesario un inicio de sesión para que el usuario tenga acceso a esta función y vista.

#### **Ejemplos de modificación:**

- **Mostrar un mensaje diferente al usuario:** Si requiere de una condicional previa entonces agregamos el argumento a la url y comprobamos su valor con request.vars  
**If request.vars.nuevo == "valor deseado":**  
**response.flash = 'mensaje'**
- **Agregar variables con datos del usuario:** auth.user nos devuelve varios datos del usuario, para conocerlos se puede consultar la documentación de Web2Py  
**Usuario\_nueva = auth.user.id**

#### **Pseudocódigo:**

Inicio función Inicio:

Si no hay lista aleatoria de definiciones por etiquetar entonces:

Usuario\_nombre = Primer nombre del usuario con sesión iniciada

Termino = lista de términos en la base de datos

Si Acaba de terminar si lista de definiciones entonces:

Mostrar 'Lista de definiciones terminada'

Fin Si

Si no entonces:

Re direccionar a 'relacion'

Fin Si

Regresar locals

Fin

#### **Código de la función:**

```
@auth.requires_login()
```

```
def inicio():
```

```
    if (len(db((db.temporal.usuario_id == auth.user.id) & (db.temporal.modos == '1')).select()) == 0):
```

```
usuario_nombre = auth.user.first_name
termino = db().select(db.termino.id, db.termino.ter)
if(request.vars.en == "1"):
    response.flash = 'Lista de definiciones terminada'
else:
    redirect(URL('relacion?rd=1'))
return locals()
```

---

## Modulo: Importar

**Descripción:** El modulo importar solo se compone de una función: “importar”, su función es leer un archivo separado por comas y agregarlo a la base de datos, el usuario indica el nombre del término, el contenido del archivo son las definiciones y se crea un grupo “No informativo”

### Código fuente relacionado con el método:

---

**Función:** importar

**Variables:**

- **idTermino:** Justo después de que se inserte el término en la base de datos obtenemos el id que se le asigne y se convierte en el valor de esta variable
- **table:** Es la tabla de la base de datos sobre la cual vaciaremos el contenido del archivo separado por comas.
- **file:** El archivo que importaremos a la base de datos
- **users:** Lista de todos los usuarios registrados

**Vista relacionada:** [particular/importar](#)

**Tarea:** Leer un archivo csv seleccionado por el etiquetador y agregar el contenido de ese archivo a la base de datos, adicionalmente creará el término y los grupos “No informativo”

### Explicación del flujo:

Primero comprobamos si se ha cargado un archivo, si es así primero se registra el término basado en los datos introducidos por el usuario, después obtenemos el id del término que se acaba de insertar. Las siguientes tres líneas importan los datos del archivo, en este proceso cada definición es asignada a un término ‘Nulo’ de manera temporal. Después de importar las definiciones se actualizan todas las que estén vinculadas al término ‘Nulo’ y se vinculan al término que inserto el etiquetador, después se crean los grupos ‘No informativo’ para ese término, esto se hace para los dos modos de trabajo. Finalmente nos re direcciona a inicio.

En caso de que no haya un archivo se muestra un mensaje al usuario.

**Adornos adicionales:**

- **@auth.requires\_login():** Indica que es necesario un inicio de sesión para que el usuario tenga acceso a esta función y vista.

**Ejemplos de modificación:**

- **Agregar otro archivo a otra tabla:** Las variables file y table contienen los valores del archivo que se insertará y la tabla donde se insertará respectivamente, basta con cambiar esos valores para insertar algo diferente, o ampliarlos para insertar más de un archivo al mismo tiempo.

**File** = request.vars.csvfile.[nuevoArchivo](#)

**Table** = db[request.vars.nuevaTabla]

**Table.import\_from\_csv\_file(file)**

### Pseudocódigo:

Inicio función importar

Si hay un archivo que importar entonces

Insertamos el término que el usuario escribió

idTermino = Consultamos el id del último término

table = Elegimos una tabla

file = el archivo que el usuario subió

Importamos los registros

Actualizamos el término de los nuevos registros

Insertamos el grupo "No informativo"

Re direccionamos a inicio

Si no:

Mostramos el mensaje 'Seleccione un archivo'

Fin Si

Regresamos el diccionario

Fin

### Código de la función:

@auth.requires\_login()

**def** importar():

**if** request.vars.csvfile **!=** **None**:

db.termino.insert(ter=request.vars.nombre)

idTermino = db(db.termino.ter == request.vars.nombre).select(db.termino.id)

table = db[request.vars.table]

file = request.vars.csvfile.file

table.import\_from\_csv\_file(file)

db(db.definicion.termino\_id==1).update(termino\_id=idTermino[0]['id'])

users = db().select(db.auth\_user.id)

db.grupo.insert(grup='No informativo', termino\_id=idTermino[0]['id'], tipo="Sistema", modo="1")

db.grupo.insert(grup='No informativo', termino\_id=idTermino[0]['id'], tipo="Sistema", modo="2")

redirect(URL('inicio'))

**else**:

response.flash = 'Selecciona un archivo'

**return** dict()

---

### Modulo: Crear grupos

**Descripción:** El modulo Crear grupos se compone de 6 funciones, 2 tienen una vista relacionada e interactúan con el usuario usando una interfaz, otra interactúa indirectamente con el usuario al ser un 'action' de un formulario, las otras tres son funciones que sirven de apoyo.

El propósito de este módulo es la creación de grupos, tanto con ayuda de una herramienta (WordNet en este documento) o personalizado.

#### Código fuente relacionado con el método:

---

**Función:** crearGrupo

**Variables:**

- **user:** Es el id del usuario que está creando el grupo
- **termino:** Es el id del término con el cuál el grupo estará relacionado.
- **grupos:** Lista de todos los grupos que han sido creados para ese término.

**Vista relacionada:** [particular/crearGrupo](#)

**Tarea:** Crear variables de apoyo a la interfaz para validaciones y leer si el usuario ha ingresado un grupo, si es así entonces agregamos ese grupo a la base de datos.

**Explicación del flujo:**

Primero se crean variables que nos apoyarán tanto en el método como en la vista, la variable grupos es para validar que un grupo no exista, además de la consulta creamos una cadena que contendrá el nombre de cada grupo que se ha creado.

Ya que se crearon las variables de apoyo revisamos si se ha escrito un nombre para el grupo, si no es así la variable será 0 y se mostrará un mensaje diciendo que se ingrese un grupo, si no es así y se escribió un nombre del grupo (es decir, ya se mandó un grupo para registrar), insertamos el grupo con el tipo 'Custom'.

**Adornos adicionales:**

- **@auth.requires\_login():** Indica que es necesario un inicio de sesión para que el usuario tenga acceso a esta función y vista.

**Ejemplos de modificación:**

Este es el único método en el cual el usuario podrá crear sus propios grupos así que no hay mucho que modificar, excepto si se requiere crear distintos tipos de grupos personalizados, en ese caso hay que agregar un if o un switch para que el tipo del grupo sea diferente:

```
if request.vars.tipo == 'nuevo_tipo':  
    db.grupo.insert(grup = request.vars.nombre, termino_id = termino, tipo='nuevo_tipo', modo='1')
```

**Pseudocódigo:**

Inicio función crearGrupo

    User = id del usuario

    Termino = El término al que está vinculado el grupo (está en la url)

    Grupos = lista de grupos en la base de datos para el término.

```

Cadena_grupos = ""
Para cada grupo en grupos hacer:
    Cadena_grupos = grupo['grup'] + " "
Fin Para cada
Si nombre del grupo es 0
    Mostrar mensaje 'Ingresa un grupo'
Si no si el nombre del grupo no está vacío:
    Insertar el grupo en la base de datos
Fin Si
Regresar locals
Fin

```

#### Código de la función:

```

@auth.requires_login()
def crearGrupo():
    user = auth.user.id
    termino = request.vars.termino
    grupos = db((db.grupo.termino_id == termino) & (db.grupo.modos == "1")).select()
    cadena_grupos = ""
    for grupo in grupos:
        cadena_grupos += grupo['grup'] + " "
    if request.vars.nombre == '0':
        response.flash = 'Ingresa un grupo'
    elif request.vars.nombre != "":
        db.grupo.insert(grupo=request.vars.nombre, termino_id=termino, tipo='Custom', modos='1')
    return locals()

```

---

**Función:** vistaWordNet

#### Variables:

- **Wordnet:** Cadena de texto que contiene el código HTML para la vista

**Vista relacionada:** [particular/vistaWordNet](#)

**Tarea:** Crear la variable que contiene el código HTML que generará la vista de la lista de grupos sugeridos.

#### Explicación del flujo:

Se asigna el valor de la función crear\_cadena a la variable wordnet.

#### Ejemplos de modificación:

Esta función y vista deberán de cambiar de nombre cuando haya más de una herramienta para crear grupos, también la url tendrá una variable para indicar el tipo de herramienta que se va a utilizar, en base a esa variable se decidirá que cadena se creará.

```

if request.vars.tipo == 'wordnet':
    cadena = crear_cadena_wordnet()
elif request.vars.tipo == 'wikipedia':
    cadena = crear_cadena_wikipedia()

```

También si se requiere que solo se haga una iteración en la búsqueda en WordNet se tiene que hacer esto:

```
wordnet = crear_cadena(request.vars.ter, False)
```

#### **Pseudocódigo:**

Inicio función vistaWordNet

    Wordnet = código html con los grupos sugeridos por wordnet

    Regresa locals

Fin

#### **Código de la función:**

```
def vistaWordNet():
```

```
    wordnet = crear_cadena(request.vars.ter, True)
```

```
    return locals()
```

---

**Función:** DBGroup

**Vista relacionada:** [particular/vistaWordNet](#)

**Tarea:** Esta función lee los grupos sugeridos seleccionados por el usuario y los inserta en la base de datos.

**Explicación del flujo:** Primero lee si se mandó una lista de grupos o solo uno, en la primer opción procede a hacer una iteración para cada grupo mandado y lo inserta en la base de datos, en la segundo opción simplemente inserta el grupo en la base de datos

#### **Ejemplos de modificación:**

En el futuro esta función insertará grupos de más de una herramienta, para que eso quede registrado en la base de datos debemos de agregar una variable a la vista y mandarla con un método get para leerla en esta función, esa variable será el tipo, después se modificará el insert para que luzca así:

```
db.grupo.insert(grup = request.vars.nombre, termino_id = termino, tipo = request.vars.tipo, modo='1')
```

#### **Pseudocódigo:**

Inicio función DBGroup

    Si los grupos mandados por el usuario son una lista entonces

        Para cada grupo en grupos hacer

            Insertar grupo en la base de datos

        Fin para cada

    Si no

        Insertar en la base de datos el grupo

    Fin Si

    Regresar locals

Fin



### Código de la función:

```
def DBGroup():
    if isinstance(request.vars.grupo, list):
        for grupo in request.vars.grupo:
            db.grupo.insert(grup = grupo, termino_id = request.vars.termino, tipo='wordnet', modo="1")
    else:
        db.grupo.insert(grup = request.vars.grupo, termino_id = request.vars.termino, tipo='wordnet', modo="1")
    return locals()
```

---

**Función:** crear\_cadena

### Parámetros:

- **termino:** El término que buscaremos en WordNet
- **flag:** Variable booleana que nos sirve para saber hasta qué nivel se detendrá la función.
- **cadenas:** Lista de cadenas que contiene cada línea de código HTML, si está vacía se inicializa en la firma de la función

### Variables:

- **lista:** Lista de synsets de WordNet

**Tarea:** Crear una lista de cadenas que es el código HTML que se mostrará en la vista, para ello utiliza un término que es en el que se está trabajando, con ese dato se consulta WordNet y se agregan los datos a la lista, puede avanzar un nivel más de búsqueda al consultar los synsets del término original.

### Explicación del flujo:

Se crea la lista de synsets basados en el término, después consultamos si ya existe un grupo con el nombre del término que buscaremos, si no es así entonces agregamos el código para crear un checkbox a la lista de cadenas, el valor del checkbox será el término con el que estamos trabajando. Después se hace una iteración para cada synset del término, si aún no se inserta la definición en la lista entonces agregamos a la lista el resto de los datos que WordNet nos proporciona como los lemas y la definición, finalmente agregamos un separador.

Si el parámetro flag es verdadero entonces hacemos una lista de lemas y hacemos una iteración con esa lista, si el lema ya está dentro de la lista se imprime “Ya existe”, si no se llama de nuevo esta función con el lema como nuevo término, un falso para que ya no avance más y la lista de cadenas que ya se tiene.

### Ejemplos de modificación:

Lo más importante de esta función es que muestra la lógica que deberán seguir todos las herramientas para sugerir grupos, primero se revisará los synsets o los datos que se proporcionen, después se verifica su no existe, se agrega el checkbox y los datos adicionales que se ofrezcan, algunas herramientas es posible que no necesiten avanzar un nivel más. Esta función se renombrara a crear\_cadena\_wordnet cuando haya más de una herramienta.

Siempre debe de existir un checkbox con el valor del término que se esté manejando en ese momento y el texto debe de decir eso mismo, cuando se agregue ese checkbox, al texto que está fuera de la etiqueta HTML se le debe de poner un '@' para identificar que es el término. Esta es la línea que se debe de mantener cada vez que se cree una cadena, sin importar la herramienta:

```
cadenas.append('<input type=\'checkbox\' name=\'grupo\' value=\''+ str(termino) + '\'/>@' + str(termino))
```

Después solo se agregarán los datos restantes con el formato HTML que se desee, después de hacer esto es importante poner el separador tal cual está en esta función.

```
cadenas.append('-----')
```

### Pseudocódigo:

Inicio función crear\_cadena

    Lista = lista de synsets del término

    Si no existe un grupo con el nombre del término entonces

        Se agrega a cadenas '<input type=\'checkbox\' name=\'grupo\' value=\''+ termino + '\'/>@'

+ termino

    Para cada synset en lista hacer

        Si aún no está en la lista de cadenas

            Lemmas = lista de lemmas del synset

            Agregar cadena de texto con el resto de los datos

        Fin Si

    Fin Para cada

    Agregar separador

Fin Si

Si flag es verdadero

    Para cada synset en lista hacer

        Lemmas = lista de lemmas del synset

        Para cada aux en lemmas hacer

            Si ya está en la lista de cadenas entonces

                Imprimir Ya existe

            Si no

                Cadenas = crear\_cadena(aux, false, cadenas)

            Fin Si

        Fin Para cada

    Fin Para cada

Fin si

Fin

### Código de la función:

```
def crear_cadena(termino, flag, cadenas=list()):
```

```
    lista = wordnet_termino(termino)
```

```
    if(len(db((db.grupo.grup == str(termino)) & (db.grupo.moda == "1")).select())) == 0):
```

```
        cadenas.append('<input type=\'checkbox\' name=\'grupo\' value=\''+ str(termino) + '\'/>@' + str(termino)) #Cada término
```

```
        for synset in lista:
```

```
            if not busca_existencia(str(synset.definition()), cadenas): lemmas = [str(lemma.name()) for
```

```
lemma in synset.lemmas]
```

```
            cadenas.append('<br><strong>Lemmas</strong>: ' + str(lemmas) + ',
```

```
<br><strong>Definition:</strong> (' + synset.definition() + '),<br> <strong>Examples: </strong>' +
```

```
str(synset.examples()).replace("u", "").replace("[]", "No available"))
```

```

cadenas.append('-----')
if flag:
    for synset in lista:
        lemmas = [str(lemma.name()) for lemma in synset.lemmas()]
        for aux in lemmas:
            if busca_existencia(str(aux), cadenas, '@'):
                print 'Ya existe'
            else:
                cadenas = crear_cadena(aux, False, cadenas)
return cadenas

```

---

**Función:** `wornet_termino`

**Parámetros:**

- **termino:** El término que buscaremos en WordNet

**Tarea:** Regresar una lista con los synsets de un término en particular

**Código de la función:**

```

def wordnet_termino(termino):
    return list(wn.synsets(str(termino)))

```

---

**Función:** `busca_existencia`

**Parámetros:**

- **cadena:** La cadena que buscaremos en la lista de cadenas
- **cadenas:** Lista de cadenas, estás son del código HTML
- **identificador:** Nos ayuda a identificar el término que estamos trabajando.

**Variables:**

- **flag:** Variable booleana que nos dice si la cadena ya existe o no.

**Tarea:** Revisar toda la lista de cadenas en busca de una cadena en específico, en el caso del programa es el término con el que estamos trabajando, esto es para evitar grupos iguales en los grupos sugeridos, si se sigue el formato planteado en `crear_cadena` entonces esta función puede ser reutilizable.

**Explicación del flujo:**

Se establece una variable booleana con un valor false, se revisa si se pasó un identificador como parámetro, si es así se agrega ese identificador a la cadena, se hace una iteración en las cadenas y buscamos si nuestra cadena se encuentra en algún elemento de la lista, si es así la variable booleana cambia su valor a verdadero, se regresa esa variable.

**Ejemplos de modificación:**

Más que un cambio en el código, cuando se manda a llamar se pueden usar diferentes identificadores para buscar cadenas en particular, por ejemplo agregar un % antes de cada synset y pasarlo como identificador

#### Pseudocódigo:

Inicio función crear\_cadena

```
    Flag = false
    Si identificador != "" entonces
        Cadena = identificador + cadena
    Fin Si
    Para cada aux en cadenas hacer
        Si lower(cadena) esta en lower(aux) entonces
            Flag = true
    Fin si
    Fin Para cada
    Regresa flag
```

Fin

#### Código de la función:

```
def busca_existencia(cadena, cadenas, identificador=""):
    flag = False
    if identificador != "":
        cadena = str(identificador) + str(cadena)
    for aux in cadenas:
        if cadena.lower() in aux.lower():
            flag = True
    return flag
```

---

#### Modulo: Crear interfaz de relación

**Descripción:** Este módulo es el más importante en cuestión de interacción con el usuario ya que presenta diversos elementos que se le mostrarán al usuario, tales como las definiciones que se etiquetarán, los grupos, los campos que el usuario debe de llenar, etc. También, en caso de ser necesario, crea la muestra aleatoria. Se compone de 4 funciones de las cuales solo 1 tiene interacción directa con el etiquetador.

#### Código fuente relacionado con el método:

---

**Función:** relacion

#### Variables:

- **muestra:** Lista de definiciones que se van a etiquetar, es solo una lista de ids
- **definición:** La primer definición de la muestra, contiene todos los datos de la definición como son el id, la definición y el id de término al que está relaciona
- **termino:** Es el término al cual está relacionada la definición.
- **Historial:** Historial de palabras clave usadas por otros usuarios para esa definición.
- **Grupos:** Todos los grupos que se han creado para ese término
- **Total\_usuarios:** El número de usuarios registrados

- **Porcentaje:** Lista de porcentajes que indican que tanto ha sido utilizado un grupo.

**Vista relacionada:** [particular/relacion](#)

**Tarea:** Crea las variables que se utilizarán en la interfaz para mostrar las definiciones, los grupos y los apoyos que se ofrecen a los etiquetadores, además crea la muestra aleatoria.

#### **Explicación del flujo:**

Se crea una lista vacía que será la muestra aleatoria, después revisamos si el etiquetador ya estaba trabajando en una muestra aleatoria, si no es así entonces procedemos a crear la muestra y mostramos un mensaje diciendo que se acaba de crear la muestra, si ya hay una muestra anterior entonces hacemos que esa muestra sea igual a la que ya existe y, si es el caso, le decimos que continúe desde la sesión anterior.

Después simplemente creamos variables que se utilizarán en la interfaz y revisamos si hay algún error en el previo llenado de datos.

#### **Adornos adicionales:**

- **@auth.requires\_login():** Indica que es necesario un inicio de sesión para que el usuario tenga acceso a esta función y vista.

#### **Ejemplos de modificación:**

- **Si se quiere agregar algún dato extra:** La muestra contiene datos básicos pero con ellos se pueden obtener más, solo hay que revisar la documentación de Web2Py y de bases de datos relacionales.

#### **Pseudocódigo:**

Inicio función relación:

Muestra = lista

Si no hay una muestra previa entonces

Muestra = getMuestra

Mostrar mensaje 'Se ha creado lista aleatoria'

Si no

Muestra = la muestra de la base de datos

Si rd = 1 entonces

Muestra mensaje 'Continúa desde la última sesión'

Fin Si

Fin Si

Definición = consulta la base de datos con muestra[0]

Termino = consulta la base de datos con definición

Historial = getHistorialKeywords

Grupos = consulta todos los grupos del termino

Si error = 0

Mostrar mensaje 'Favor de llenar campos'

Si no si error = 1

Mostrar mensaje 'Las palabras clave no se encuentran en la definicion'

```

        Fin si
        Total_usuarios: consulta todos los usuarios
        Porcentaje = getPorcentajes
        Regresar locals
    Fin
Código de la función:

@auth.requires_login()
def relacion():
    muestra = list()
    if(len(db((db.temporal.usuario_id == auth.user.id) & (db.temporal.modo == '1')).select()) == 0):
        muestra = getMuestra(db(db.definicion.termino_id==request.vars.termino).select())
        response.flash = 'Se ha creado una lista aleatoria'
    else:
        muestra = db((db.temporal.usuario_id == auth.user.id) & (db.temporal.modo == '1')).select()
        if(request.vars.rd == "1"):
            response.flash = 'Continúa desde la última sesión'
        definicion = db(db.definicion.id==muestra[0]['definicion_id']).select()[0]
        termino = db(db.termino.id==definicion['termino_id']).select()[0]
        historial = getHistorialKeywords(db((db.relacion.definicion_id==definicion['id']) & (db.relacion.modo == "1")).select())
        grupos = db((db.grupo.termino_id == termino['id']) & (db.grupo.modo == '1')).select()
        if(request.vars.error == '0'):
            response.flash = 'Favor de llenar los campos'
        elif request.vars.error == '1':
            response.flash = 'Las palabras clave no se encuentran en la definición'
        total_usuarios = len(db(db.auth_user.id > 0).select())
        porcentaje = getPorcentajes(db((db.relacion.modo == "1")).select())
    return locals()

```

---

**Función:** getMuestra

**Parámetros:**

- **Lista\_definiciones:** Lista de todas las definiciones de un término.

**Variables:**

- **i:** Contador.
- **Muestra:** Lista de definiciones en la muestra.
- **Max:** Numero de definiciones que habrá en la muestra
- **Definición:** Numero aleatorio que representa una definición

**Tarea:** Crea una lista de números aleatorios que será la muestra.

**Explicación del flujo:**

Después de crear un par de variables auxiliares revisamos si hay suficientes definiciones para hacer una muestra de 10, si no entonces se hará con las que hay disponibles.

Se hace una iteración mientras no haya 10 elementos en la muestra se hace un numero aleatorio, se revisa que aún no esté en la lista y si es así se agrega la definición relacionada a la muestra, finalmente se registra esa lista en la base de datos y se regresa.

#### Ejemplos de modificación:

- **Si se quiere hacer mayor la muestra:** Solo hay que cambiar el valor de max:  
Max = nuevo\_valor

#### Pseudocódigo:

Inicio función getMuestra

```
I = 0
Muestra = lista
Max = 10
Si el tamaño de la lista de definiciones < 10 entonces
    Max = tamaño de la lista de definiciones
Fin si
Mientras i < max hacer
    Definición = numero aleatorio
    Si no esta en la lista entonces
        Se agrega la definición a la lista
        I++
    Fin si
Fin mientras
registrarLista
Regresa la lista
```

Fin

#### Código de la función:

```
def getMuestra(lista_definiciones):
    i = 0
    muestra = list()
    max = 10
    if(len(lista_definiciones) < 10):
        max = len(lista_definiciones)
    while (i < max):
        definicion = int(random.uniform(0, len(lista_definiciones) - 1))
        if noEnLista(lista_definiciones[definicion]['id'], muestra):
            muestra.append(lista_definiciones[definicion])
            i = i + 1
    registraLista(muestra)
    return db((db.temporal.usuario_id == auth.user.id) & (db.temporal.modos == '1')).select()
```

---

**Función:** noEnLista

#### Parámetros:

- **idDefinicion:** Numero aleatorio que se desea agregar a la lista
- **muestra:** Lista de definiciones aleatorias

**Variables:**

- **flag:** Booleano que nos indica si la definición ya está en la lista.

**Tarea:** Revisa la muestra existente para ver si cierta definición ya está en la lista

**Explicación del flujo:**

Se crea una variable booleana, se hace una iteración en la muestra y se revisa si algún dato es igual, si es así la booleana se vuelve falsa, se regresa esa variable.

**Pseudocódigo:**

```
Inicio función noEnLista
    Flag = true
    Para cada definición en muestra hacer
        Si definición['id'] = idDefincion
            Flag = false
        Fin si
    Fin Para cada
    Regresa flag
Fin
```

**Código de la función:**

```
def noEnLista(idDefinicion, muestra):
    flag = True
    for definicion in muestra:
        if definicion['id'] == idDefinicion:
            flag = False
    return flag
```

---

Modulo: [Crear historial de palabras clave](#)

**Descripción:** Este módulo solo se compone por una función, tiene como entrada una lista de definiciones ya etiquetadas. Básicamente lee todas esas definiciones y almacena las palabras clave utilizadas para cierta definición en un diccionario.

**Código fuente relacionado con el método:**

---

**Función:** getHistorialKeywords

**Parámetros:**

- **lista:** Una lista con los etiquetados de una definición y las palabras clave que se utilizaron.

**Variables:**

- **frase\_clave:** Lista con las palabras clave utilizadas.
- **Aux:** Lista con las palabras clave utilizadas en un registro
- **Conteo\_frase:** Diccionario con el conteo de las palabras clave
- **Ordenado:** Diccionario ordenado

**Vista relacionada:** [particular/relacion](#)



**Tarea:** Revisa los registros para cierta definición y cuenta las palabras clave, las almacena en un diccionario.

### Explicación del flujo:

Primero hace una iteración para cada registro en la lista, por cada registro obtiene una lista de palabras clave y hace una iteración sobre eso, si tiene un espacio antes lo elimina, al final de esa iteración agrega la frase a una lista.

Después hace una nueva iteración para cada frase en frase\_clave, revisa si está en el diccionario y dependiendo de eso lo suma o crea una nueva llave con valor 1, finalmente ordena el diccionario.

### Ejemplos de modificación:

- El diccionario puede ser reordenado o modificar cuantos elementos hay.

### Pseudocódigo:

```
Inicio función getHistorialKeywords
    Frase_clave = lista
    Para cada registro en lista hacer
        Aux = separar las keywords
        Para cada frase en aux hacer
            Si frase[0] = ' ' entonces
                Frase = frase ignorando el primer carácter
            Fin si
            Agregar frase a frase_clave
        Fin Para cada
    Fin Para cada
    Conteo_frase = diccionario
    Para cada frase en frase_clave hacer
        Si frase esta en conteo_frase
            Conteo_frase[frase] += 1
        Si no
            Conteo_frase[frase] = 1
        Fin si
    Fin para cada
    Ordenado = ordenar diccionario
Fin
```

### Código de la función:

```
def getHistorialKeywords(lista):
    frase_clave = []
    for registro in lista:
        aux = registro['keywords'].split(',')
        for frase in aux:
            if frase[0] == ' ':
                frase = frase[1:]
            frase_clave.append(frase)
```

```
conteo_frase = {}
for frase in frase_clave:
    if frase in conteo_frase:
        conteo_frase[frase] += 1
    else:
        conteo_frase[frase] = 1
ordenado = sorted(conteo_frase.items(), key=operator.itemgetter(1), reverse=True)
return ordenado
```

---

Modulo: Crear historial de definiciones usadas en un grupo

**Descripción:** Este módulo solo se compone por una función, esta función es llamada usando Ajax, revisa una lista de grupos y genera una lista de definiciones usadas en ese grupo.

**Código fuente relacionado con el método:**

---

**Función:** getHistorialGrupos

**Variables:**

- **respuestaHtml:** Respuesta que devolverá a la función Ajax
- **user:** usuario que hace la consulta
- **historial:** Historial de los grupos usados por el usuario
- **grupoNombre:** Nombre del grupo
- **contador:** Un contador
- **aux:** Variable de apoyo

**Vista relacionada:** [particular/relacion](#)

**Tarea:** Recibe uno o más grupos, con ese dato busca en la base de datos las relaciones hechas por el etiquetador usando ese grupo y la definición que uso, agrega eso a la respuesta en código HTML y lo regresa.

**Explicación del flujo:**

Crea variables de apoyo, después revisa si se manda más de un grupo si esa así hace una iteración para cada grupo en la lista, obtiene todas las relaciones hechas por un usuario usando ese grupo, con eso obtiene el nombre del grupo y añade ese nombre a la respuesta, declara un contador y una auxiliar, revisa si hay suficientes elementos en el historial para devolver 10 resultados, si no entonces devuelve los que están disponibles, mientras aún no se hayan agregado 10 resultados (o los disponibles) se agrega a la respuesta la definición usada.

Si no es una lista de grupos obtiene todas las relaciones hechas por un usuario usando ese grupo, con eso obtiene el nombre del grupo y añade ese nombre a la respuesta, declara un contador y una auxiliar, revisa si hay suficientes elementos en el historial para devolver 10 resultados, si no entonces devuelve los que están disponibles, mientras aún no se hayan agregado 10 resultados (o los disponibles) se agrega a la respuesta la definición usada.

**Pseudocódigo:**

Inicio función getHistorialGrupos

```

respuestaHTML = ""
user = c
Si se mandó al menos un grupo entonces
    Si hay más de un grupo entonces
        Para cada grupo en grupos hacer
            Historial = lista de grupos usados por ese usuario
            grupoNombre = nombre del grupo
            respuestaHTML += grupoNombre
            contador = 10
            aux = tamaño del historial
            Si el tamaño del historial < 10 entonces
                Contador = tamaño del historial
            Fin si
            Mientras contador > 0 hacer
                respuestaHTML += definición
                contador—
                aux—
            Fin mientras
            respuestaHTML += "</p>"
        Fin para
    Si no
        Historial = lista de grupos usados por ese usuario
        grupoNombre = nombre del grupo
        respuestaHTML += grupoNombre
        contador = 10
        aux = tamaño del historial
        Si el tamaño del historial < 10 entonces
            Contador = tamaño del historial
        Fin si
        Mientras contador > 0 hacer
            respuestaHTML += definición
            contador—
            aux—
        Fin mientras
        respuestaHTML += "</p>"
    Fin si
Si no
    respuestaHTML = "Seleccione un grupo"
Fin si
Regresa respuestaHTML

```

Fin

#### Código de la función:

```

def getHistorialGrupos():
    respuestaHtml = ""
    user = request.vars.c
    if request.vars.grupos != None:
        if isinstance(request.vars.grupos, list):

```

```

for grupo in request.vars.grupos:
    historial = db((db.relacion.grupo_id == grupo) & (db.relacion.usuario == user) &
(db.relacion.modos == "1")).select()
    grupoNombre = db((db.grupo.id == grupo) & (db.grupo.modos == "1")).select()[0]['grup']
    respuestaHtml += "<p><strong>Grupo:</strong> " + grupoNombre + "<br>"
    contador = 10
    aux = len(historial) - 1
    if (len(historial) < 10):
        contador = len(historial)
    while(contador > 0):
        respuestaHtml += "<strong>Definición: </strong>" + db(db.definicion.id ==
historial[aux]['definicion_id']).select()[0]['defi'] + "<br>"
        contador -= 1
        aux -= 1
        respuestaHtml += "<p>"
    else:
        historial = db((db.relacion.grupo_id == request.vars.grupos) & (db.relacion.usuario == user) &
(db.relacion.modos == "1")).select()
        grupoNombre = db((db.grupo.id == request.vars.grupos) & (db.grupo.modos ==
"1")).select()[0]['grup']
        respuestaHtml += "<p><strong>Grupo:</strong> " + grupoNombre + "<br>"
        contador = 10
        aux = len(historial) - 1
        if (len(historial) < 10):
            contador = len(historial)
        while(contador > 0):
            respuestaHtml += "<strong>Definición: </strong>" + db(db.definicion.id ==
historial[aux]['definicion_id']).select()[0]['defi'] + "<br>"
            contador -= 1
            aux -= 1
            respuestaHtml += "<p>"
        else:
            respuestaHtml = "Selecciona un grupo"
return respuestaHtml

```

---

Modulo: Obtener porcentaje de uso para cada grupo

**Descripción:** Este módulo solo se compone por una función, su objetivo es calcular el porcentaje de uso de cada grupo basado en el número de usuarios

**Código fuente relacionado con el método:**

---

**Función:** getPorcentajes

**Parámetros:**

- **grupos:** Lista de relaciones en la base de datos

**Variables:**

- **porcentaje:** Diccionario que almacena la cantidad de veces que ha sido utilizado un grupo

**Vista relacionada:** [particular/relacion](#)

**Tarea:** Guarda el número de usos de un grupo por usuarios

### Explicación del flujo:

Después de crear el diccionario hace una iteración para cada registro en la base de datos, comprobamos que el grupo este en el diccionario y que el usuario no este, si es así le suma un elemento al porcentaje de ese grupo, si no es así entonces agrega el grupo al diccionario y el usuario.

### Pseudocódigo:

```
Inicio función getHistorialKeywords
    Porcentaje = diccionario
    Para cada grupo en grupos hacer
        Si el grupo está en porcentajes y el usuario no está en porcentajes entonces
            Porcentaje[grupo] ++
        Si no
            Porcentaje[grupo] = 1
            Porcentaje[usuario] = 1
        Fin si
    Fin para cada
    Regresa porcentaje
Fin
```

### Código de la función:

```
def getPorcentajes(grupos):
    porcentaje = dict()
    for grupo in grupos:
        if grupo['grupo_id'] in porcentaje and grupo['usuario'] not in porcentaje:
            porcentaje[grupo['grupo_id']] += 1
        else:
            porcentaje[grupo['grupo_id']] = 1
            porcentaje["u" + str(grupo['usuario'])] = 1
    return porcentaje
```

---

### Modulo: Crear la relación

**Descripción:** Este es el modulo más importante del modo particular ya que se encarga de tomar los datos que el etiquetador capturo en el sistema y los escribe en la base de datos, es el action del formulario de relación.

### Código fuente relacionado con el método:

---

**Función:** unir

**Vista relacionada:** [particular/relacion](#)

**Tarea:** Escribir los datos proporcionados por el usuario en la base de datos.

### Explicación del flujo:

Primero hace una validación para ver si los datos fueron introducidos de manera correcta, si es no es así re direcciona a la página de relación, si todo está bien entonces revisa si el etiquetador selecciono más de un grupo, en ese caso primero crea una lista de grados y un contador, mientras

ese contador sea mejor que el tamaño de la lista de grados, es decir, mientras haya al menos un grado que no ha sido registrado, el sistema registrara los datos proporcionados por el etiquetador.

Si solo se seleccionó un grupo entonces solo se inserta el registro sin necesidad de hacer una lista de grados de certeza.

Cuando se termina de hacer eso simplemente se elimina esa definición de la muestra aleatoria y si ya no quedan elementos entonces se manda a la página de inicio, en caso contrario se manda de nuevo a la página de relación.

### Ejemplos de modificación:

- En esta parte se insertan los datos requeridos, en caso de que se desee agregar más contenido a la base de datos entonces, después de modificar el archivo db.py, aquí se pueden agregar esos datos extra, la documentación de Web2Py explica como modificar el insert para que acepte más campos.

### Pseudocódigo:

Inicio función unir

```
Si los datos no son válidos entonces
    Re direccionar a relación
Si no
    Si grupo es una lista entonces
        Grado = lista de grados
        I = 0
        Mientras i < tamaño de grado hacer
            Insertar datos
            I++
        Fin mientras
    Si no
        Insertar datos
    Fin si
    Eliminar registro temporal
    Si ya no hay más definiciones en la muestra entonces
        Re direccionar a inicio
    Si no
        Re direccionar a relación
    Fin si
Fin si
Fin so
Regresa locals()
```

Fin

### Código de la función:

```
@auth.requires_login()
```

```
def unir():
```

```
    if(request.vars.keyword == " or request.vars.grupo == None): #Validación de campos vacios
        redirect(URL('relacion?error=0'))
```

```
    else:
```

```

if isinstance(request.vars.grupo, list):
    request.vars.grado = [x for x in request.vars.grado if x != ""]
    i = 0
    while i < len(request.vars.grupo):
        db.relacion.insert(keywords=request.vars.keyword, grado=request.vars.grado[i],
grupo_id=request.vars.grupo[i], definicion_id=request.vars.definicion, usuario=auth.user.id, modo="1")
        i = i + 1
    else:
        db.relacion.insert(keywords=request.vars.keyword, grado='100', grupo_id=request.vars.grupo,
definicion_id=request.vars.definicion, usuario=auth.user.id, modo="1")
        db((db.temporal.usuario_id==auth.user.id) & (db.temporal.definicion_id == request.vars.definicion) &
(db.temporal.modo == "1")).delete()
        if(len(db((db.temporal.usuario_id==auth.user.id) & (db.temporal.modo == "1")).select()) == 0):
            redirect(URL('inicio?en=1'))
        else:
            redirect(URL('relacion'))
return locals()

```

---

## General.py

Se encarga del modo “General”, tiene 16 funciones de las cuales 8 interactúan directamente con el usuario ya sea como vistas o como el atributo ‘action’ de los formularios, recordemos que en este modo se puede seleccionar más de un término para hacer la lista aleatoria. Las otras 8 proveen apoyo para que el sistema sea más modular. Se procederá a explicar cada módulo del sistema y el código fuente relacionado.

### Modulo: Inicio

**Descripción:** El modulo inicio solo se compone de una función: “inicio”, su función principal es comprobar si el etiquetador tiene una lista pendiente, si no es así crea las variables necesarias para la vista.

### Código fuente relacionado con el método:

---

**Función:** inicio

#### Variables:

- **usuario\_nombre:** Toma su valor de la base de datos y de la sesión actual, se utiliza para mostrar el nombre del usuario en la vista.
- **termino:** La vista necesita que el usuario seleccione un término, para ello se deben de listar los términos existentes, esta variable almacena esa información, la obtiene de la base de datos.

**Vista relacionada:** [general/inicio](#)

**Tarea:** Crea las variables que se utilizarán en la interfaz principal como lo son el nombre del etiquetador y la lista de términos existentes entre los cuales el etiquetador seleccionará uno para crear una lista aleatoria para etiquetar.

#### **Explicación del flujo:**

Primero buscamos si el etiquetador ya está trabajando en algunas lista de definiciones, si no es así entonces crea las variables usuario\_nombre y termino después revisa si el etiquetador termino de etiquetar su lista antes de cargar esta vista (argumentos en la url), si es así entonces muestra un mensaje indicando ese caso.

Si el usuario ya está trabajando en una lista, entonces lo manda a la vista de relación.

#### **Adornos adicionales:**

- **@auth.requires\_login():** Indica que es necesario un inicio de sesión para que el usuario tenga acceso a esta función y vista.

#### **Ejemplos de modificación:**

- **Mostrar un mensaje diferente al usuario:** Si requiere de una condicional previa entonces agregamos el argumento a la url y comprobamos su valor con request.vars  
**If request.vars.nuevo == "valor deseado":**  
**response.flash = 'mensaje'**
- **Agregar variables con datos del usuario:** auth.user nos devuelve varios datos del usuario, para conocerlos se puede consultar la documentación de Web2Py  
**Usuario\_nueva = auth.user.id**

#### **Pseudocódigo:**

Inicio función Inicio:

Si no hay lista aleatoria de definiciones por etiquetar entonces:

Usuario\_nombre = Primer nombre del usuario con sesión iniciada

Termino = lista de términos en la base de datos

Si Acaba de terminar si lista de definiciones entonces:

Mostrar 'Lista de definiciones terminada'

Fin Si

Si no entonces:

Re direccionar a 'relacion'

Fin Si

Regresar locals

Fin

#### **Código de la función:**

```
@auth.requires_login()
```

```
def inicio():
```

```
    if (len(db((db.temporal.usuario_id == auth.user.id) & (db.temporal.modos == '1')).select()) == 0):
```



```
usuario_nombre = auth.user.first_name
termino = db().select(db.termino.id, db.termino.ter)
if(request.vars.en == "1"):
    response.flash = 'Lista de definiciones terminada'
else:
    redirect(URL('relacion?rd=1'))
return locals()
```

---

## Modulo: Importar

**Descripción:** El modulo importar solo se compone de una función: “importar”, su función es leer un archivo separado por comas y agregarlo a la base de datos, el usuario indica el nombre del término, el contenido del archivo son las definiciones y se crea un grupo “No informativo”

### Código fuente relacionado con el método:

---

**Función:** importar

**Variables:**

- **idTermino:** Justo después de que se inserte el término en la base de datos obtenemos el id que se le asigne y se convierte en el valor de esta variable
- **table:** Es la tabla de la base de datos sobre la cual vaciaremos el contenido del archivo separado por comas.
- **file:** El archivo que importaremos a la base de datos
- **users:** Lista de todos los usuarios registrados

**Vista relacionada:** [general/importar](#)

**Tarea:** Leer un archivo csv seleccionado por el etiquetador y agregar el contenido de ese archivo a la base de datos, adicionalmente creará el término y los grupos “No informativo”

### Explicación del flujo:

Primero comprobamos si se ha cargado un archivo, si es así primero se registra el término basado en los datos introducidos por el usuario, después obtenemos el id del término que se acaba de insertar. Las siguientes tres líneas importan los datos del archivo, en este proceso cada definición es asignada a un término ‘Nulo’ de manera temporal. Después de importar las definiciones se actualizan todas las que estén vinculadas al término ‘Nulo’ y se vinculan al término que inserto el etiquetador, después se crean los grupos ‘No informativo’ para ese término, esto se hace para los dos modos de trabajo. Finalmente nos re direcciona a inicio.

En caso de que no haya un archivo se muestra un mensaje al usuario.

**Adornos adicionales:**

- **@auth.requires\_login():** Indica que es necesario un inicio de sesión para que el usuario tenga acceso a esta función y vista.

**Ejemplos de modificación:**

- **Agregar otro archivo a otra tabla:** Las variables file y table contienen los valores del archivo que se insertará y la tabla donde se insertará respectivamente, basta con cambiar esos valores para insertar algo diferente, o ampliarlos para insertar más de un archivo al mismo tiempo.

**File = request.vars.csvfile.nuevoArchivo**

**Table = db[request.vars.nuevaTabla]**

**Table.import\_from\_csv\_file(file)**

### Pseudocódigo:

Inicio función importar

Si hay un archivo que importar entonces

Insertamos el término que el usuario escribió

idTermino = Consultamos el id del último término

table = Elegimos una tabla

file = el archivo que el usuario subió

Importamos los registros

Actualizamos el término de los nuevos registros

Insertamos el grupo "No informativo"

Re direccionamos a inicio

Si no:

Mostramos el mensaje 'Seleccione un archivo'

Fin Si

Regresamos el diccionario

Fin

### Código de la función:

@auth.requires\_login()

**def importar():**

**if request.vars.csvfile != None:**

db.termino.insert(ter=request.vars.nombre)

idTermino = db(db.termino.ter == request.vars.nombre).select(db.termino.id)

table = db[request.vars.table]

file = request.vars.csvfile.file

table.import\_from\_csv\_file(file)

db(db.definicion.termino\_id==1).update(termino\_id=idTermino[0]['id'])

users = db().select(db.auth\_user.id)

db.grupo.insert(grup='No informativo', termino\_id=idTermino[0]['id'], tipo="Sistema", modo="1")

db.grupo.insert(grup='No informativo', termino\_id=idTermino[0]['id'], tipo="Sistema", modo="2")

redirect(URL('inicio'))

**else:**

response.flash = 'Selecciona un archivo'

**return dict()**

### Modulo: Crear grupos

**Descripción:** El modulo Crear grupos se compone de 6 funciones, 2 tienen una vista relacionada e interactúan con el usuario usando una interfaz, otra interactúa indirectamente con el usuario al ser un 'action' de un formulario, las otras tres son funciones que sirven de apoyo.

El propósito de este módulo es la creación de grupos, tanto con ayuda de una herramienta (WordNet en este documento) o personalizado.

#### Código fuente relacionado con el método:

---

**Función:** crearGrupo

**Variables:**

- **user:** Es el id del usuario que está creando el grupo
- **termino:** Es el id del término con el cuál el grupo estará relacionado.
- **grupos:** Lista de todos los grupos que han sido creados para ese término.

**Vista relacionada:** [general/crearGrupo](#)

**Tarea:** Crear variables de apoyo a la interfaz para validaciones y leer si el usuario ha ingresado un grupo, si es así entonces agregamos ese grupo a la base de datos.

#### Explicación del flujo:

Primero se crean variables que nos apoyarán tanto en el método como en la vista, la variable grupos es para validar que un grupo no exista, además de la consulta creamos una cadena que contendrá el nombre de cada grupo que se ha creado.

Ya que se crearon las variables de apoyo revisamos si se ha escrito un nombre para el grupo, si no es así la variable será 0 y se mostrará un mensaje diciendo que se ingrese un grupo, si no es así y se escribió un nombre del grupo (es decir, ya se mandó un grupo para registrar), insertamos el grupo con el tipo 'Custom'.

#### Adornos adicionales:

- **@auth.requires\_login():** Indica que es necesario un inicio de sesión para que el usuario tenga acceso a esta función y vista.

#### Ejemplos de modificación:

Este es el único método en el cual el usuario podrá crear sus propios grupos así que no hay mucho que modificar, excepto si se requiere crear distintos tipos de grupos personalizados, en ese caso hay que agregar un if o un switch para que el tipo del grupo sea diferente:

```
if request.vars.tipo == 'nuevo_tipo':  
    db.grupo.insert(grup = request.vars.nombre, termino_id = termino, tipo='nuevo_tipo', modo='1')
```

#### Pseudocódigo:

Inicio función crearGrupo

    User = id del usuario

    Termino = El término al que está vinculado el grupo (está en la url)

    Grupos = lista de grupos en la base de datos para el término.

    Cadena\_grupos = ""

    Para cada grupo en grupos hacer:

```

        Cadena_grupos = grupo['grup'] + " "
    Fin Para cada
    Si nombre del grupo es 0
        Mostrar mensaje 'Ingresa un grupo'
    Si no si el nombre del grupo no está vacío:
        Insertar el grupo en la base de datos
    Fin Si
    Regresar locals
Fin

```

#### Código de la función:

```

@auth.requires_login()
def crearGrupo():
    user = auth.user.id
    termino = request.vars.termino
    grupos = db((db.grupo.termino_id == termino) & (db.grupo.modos == "1")).select()
    cadena_grupos = ""
    for grupo in grupos:
        cadena_grupos += grupo['grup'] + " "
    if request.vars.nombre == '0':
        response.flash = 'Ingresa un grupo'
    elif request.vars.nombre != "":
        db.grupo.insert(grupo = request.vars.nombre, termino_id = termino, tipo='Custom', modos='1')
    return locals()

```

---

**Función:** vistaWordNet

#### Variables:

- **idTermino:** El id de cada término que el usuario está utilizando
- **Wordnet:** Cadena de texto que contiene el código HTML para la vista

**Vista relacionada:** [general/vistaWordNet](#)

**Tarea:** Crear la variable que contiene el código HTML que generará la vista de la lista de grupos sugeridos.

#### Explicación del flujo:

Se crea una lista, como en este modo se trabaja con más de un término entonces primero revisa todos y cada uno de los términos que se han pasado y añade a la lista los grupos de cada término.

#### Ejemplos de modificación:

Esta función y vista deberán de cambiar de nombre cuando haya más de una herramienta para crear grupos, también la url tendrá una variable para indicar el tipo de herramienta que se va a utilizar, en base a esa variable se decidirá que cadena se creará. Todo esto va dentro de la iteración for.

```

if request.vars.tipo == 'wordnet':
    cadena = crear_cadena_wordnet()
elif request.vars.tipo == 'wikipedia':
    cadena = crear_cadena_wikipedia()

```

También si se requiere que solo se haga una iteración en la búsqueda en WordNet se tiene que hacer esto:

```
wordnet = crear_cadena(request.vars.ter, False, idTermino)
```

### Pseudocódigo:

Inicio función vistaWordNet

    Wordnet = lista

    Para cada elemento en lista de términos hacer

        idTermino = id del término que se está trabajando

        wordnet = crear\_cadena

    Fin para cada

    Regresa locals

Fin

### Código de la función:

```
def vistaWordNet():
```

```
    wordnet = list()
```

```
    for elemento in request.vars.texto:
```

```
        idTermino = db(db.termino.ter == elemento).select()[0]['id']
```

```
        wordnet = crear_cadena(elemento, True, idTermino)
```

```
    return locals()
```

---

**Función:** DBGroup

### Variables:

- **Números:** Está variable cuenta cuantos números hay en el grupo que se está manejando

**Vista relacionada:** [general/vistaWordNet](#)

**Tarea:** Esta función lee los grupos sugeridos seleccionados por el usuario y los inserta en la base de datos.

**Explicación del flujo:** Primero lee si se mandó una lista de grupos o solo uno, en la primer opción procede a hacer una iteración para cada grupo mandado, hace una suma de la cantidad de números que se están manejando en ese grupo y lo inserta en la base de datos utilizando la parte numérica como id del término y el texto después de los números como nombre del grupo, en la segundo opción simplemente inserta el grupo en la base de datos

### Ejemplos de modificación:

En el futuro esta función insertará grupos de más de una herramienta, para que eso quede registrado en la base de datos debemos de agregar una variable a la vista y mandarla con un método get para leerla en esta función, esa variable será el tipo, después se modificará el insert para que luzca así:

```
db.grupo.insert(grup = request.vars.grupo[numeros:], termino_id = request.vars.grupo[numeros-1], tipo =  
request.vars.tipo, modo='2')
```

### Pseudocódigo:

Inicio función DBGroup

    Si los grupos mandados por el usuario son una lista entonces

        Para cada grupo en grupos hacer

            Números = cantidad de dígitos en grupo

            Insertar grupo en la base de datos

        Fin para cada

    Si no

        Números = cantidad de dígitos en grupo

        Insertar en la base de datos el grupo

    Fin Si

    Regresar locals

Fin

### Código de la función:

```
def DBGroup():
```

```
    if isinstance(request.vars.grupo, list):
```

```
        for grupo in request.vars.grupo:
```

```
            numeros = sum(c.isdigit() for c in grupo)
```

```
            db.grupo.insert(grup = grupo[numeros:], termino_id = grupo[numeros-1], tipo='wordnet', modo="2")
```

```
    else:
```

```
        numeros = sum(c.isdigit() for c in request.vars.grupo)
```

```
        db.grupo.insert(grup = request.vars.grupo[numeros:], termino_id = request.vars.grupo[numeros-1],
```

```
        tipo='wordnet', modo="2")
```

```
    return locals()
```

---

**Función:** crear\_cadena

### Parámetros:

- **termino:** El término que buscaremos en WordNet
- **flag:** Variable booleana que nos sirve para saber hasta qué nivel se detendrá la función.
- **cadenas:** Lista de cadenas que contiene cada línea de código HTML, si está vacía se inicializa en la firma de la función

### Variables:

- **lista:** Lista de synsets de WordNet

**Tarea:** Crear una lista de cadenas que es el código HTML que se mostrará en la vista, para ello utiliza un término que es en el que se está trabajando, con ese dato se consulta WordNet y se agregan los datos a la lista, puede avanzar un nivel más de búsqueda al consultar los synsets del término original.

### Explicación del flujo:

Se crea la lista de synsets basados en el término, después consultamos si ya existe un grupo con el nombre del término que buscaremos, si no es así entonces agregamos el código para crear un checkbox a la lista de cadenas, el valor del checkbox será el término con el que estamos trabajando. Después se hace una iteración para cada synset del término, si aún no se inserta la definición en la

lista entonces agregamos a la lista el resto de los datos que WordNet nos proporciona como los lemas y la definición, finalmente agregamos un separador.

Si el parámetro flag es verdadero entonces hacemos una lista de lemas y hacemos una iteración con esa lista, si el lema ya está dentro de la lista se imprime “Ya existe”, si no se llama de nuevo esta función con el lema como nuevo término, un falso para que ya no avance más y la lista de cadenas que ya se tiene.

### Ejemplos de modificación:

Lo más importante de esta función es que muestra la lógica que deberán seguir todos las herramientas para sugerir grupos, primero se revisará los synsets o los datos que se proporcionen, después se verifica su no existe, se agrega el checkbox y los datos adicionales que se ofrezcan, algunas herramientas es posible que no necesiten avanzar un nivel más. Esta función se renombrará a crear\_cadena\_wordnet cuando haya más de una herramienta.

Siempre debe de existir un checkbox con el valor del término que se esté manejando en ese momento y el texto debe de decir eso mismo, cuando se agregue ese checkbox, al texto que está fuera de la etiqueta HTML se le debe de poner un ‘@’ para identificar que es el término.

Esta es la línea que se debe de mantener cada vez que se cree una cadena, sin importar la herramienta:

```
cadenas.append('<input type=\'checkbox\' name=\'grupo\' value=\''+ str(termino) + '\'/>@' + str(termino))
```

Después solo se agregarán los datos restantes con el formato HTML que se desee, después de hacer esto es importante poner el separador tal cual está en esta función.

```
cadenas.append('-----')
```

### Pseudocódigo:

Inicio función crear\_cadena

    Lista = lista de synsets del término

    Si no existe un grupo con el nombre del término entonces

        Se agrega a cadenas '

+ termino

        Para cada synset en lista hacer

            Si aún no está en la lista de cadenas

                Lemmas = lista de lemmas del synset

                Agregar cadena de texto con el resto de los datos

            Fin Si

        Fin Para cada

        Agregar separador

    Fin Si

    Si flag es verdadero

        Para cada synset en lista hacer

            Lemmas = lista de lemmas del synset

            Para cada aux en lemmas hacer

                Si ya está en la lista de cadenas entonces

                    Imprimir Ya existe

                Si no

```

                                Cadenas = crear_cadena(aux, false, cadenas)
                                Fin Si
                                Fin Para cada
                                Fin Para cada
                                Fin si
Fin
Código de la función:

def crear_cadena(termino, flag, cadenas=list()):
    lista = wordnet_termino(termino)
    if(len(db((db.grupo.grup == str(termino)) & (db.grupo.modos == "1"))).select()) == 0):
        cadenas.append('<input type=\'checkbox\' name=\'grupo\' value=\''+ str(termino) + '\'/>@' +
str(termino)) #Cada término
        for synset in lista:
            if not busca_existencia(str(synset.definition()), cadenas):          lemmas = [str(lemma.name()) for
lemma in synset.lemmas()]
                cadenas.append('<br><strong>Lemmas</strong>: ' + str(lemmas) + ',
<br><strong>Definition:</strong> (' + synset.definition() + '),<br> <strong>Examples: </strong>' +
str(synset.examples()).replace("u", "").replace("[]", "No available"))
                cadenas.append('-----')
        if flag:
            for synset in lista:
                lemmas = [str(lemma.name()) for lemma in synset.lemmas()]
                for aux in lemmas:
                    if busca_existencia(str(aux), cadenas, '@'):
                        print 'Ya existe'
                    else:
                        cadenas = crear_cadena(aux, False, cadenas)
    return cadenas

```

---

**Función:** wordnet\_termino

**Parámetros:**

- **termino:** El término que buscaremos en WordNet

**Tarea:** Regresar una lista con los synsets de un término en particular

**Código de la función:**

```

def wordnet_termino(termino):
    return list(wn.synsets(str(termino)))

```

---

**Función:** busca\_existencia

**Parámetros:**

- **cadena:** La cadena que buscaremos en la lista de cadenas
- **cadenas:** Lista de cadenas, estas son del código HTML
- **identificador:** Nos ayuda a identificar el término que estamos trabajando.

**Variables:**



- **flag:** Variable booleana que nos dice si la cadena ya existe o no.

**Tarea:** Revisar toda la lista de cadenas en busca de una cadena en específico, en el caso del programa es el término con el que estamos trabajando, esto es para evitar grupos iguales en los grupos sugeridos, si se sigue el formato planteado en crear\_cadena entonces esta función puede ser reutilizable.

#### Explicación del flujo:

Se establece una variable booleana con un valor false, se revisa si se pasó un identificador como parámetro, si es así se agrega ese identificador a la cadena, se hace una iteración en las cadenas y buscamos si nuestra cadena se encuentra en algún elemento de la lista, si es así la variable booleana cambia su valor a verdadero, se regresa esa variable.

#### Ejemplos de modificación:

Más que un cambio en el código, cuando se manda a llamar se pueden usar diferentes identificadores para buscar cadenas en particular, por ejemplo agregar un % antes de cada synset y pasarlo como identificador

#### Pseudocódigo:

```
Inicio función crear_cadena
    Flag = false
    Si identificador != "" entonces
        Cadena = identificador + cadena
    Fin Si
    Para cada aux en cadenas hacer
        Si lower(cadena) esta en lower(aux) entonces
            Flag = true
        Fin si
    Fin Para cada
    Regresa flag
Fin
```

#### Código de la función:

```
def busca_existencia(cadena, cadenas, identificador=""):
    flag = False
    if identificador != "":
        cadena = str(identificador) + str(cadena)
    for aux in cadenas:
        if cadena.lower() in aux.lower():
            flag = True
    return flag
```

---

#### Modulo: Crear interfaz de relación

**Descripción:** Este módulo es el más importante en cuestión de interacción con el usuario ya que presenta diversos elementos que se le mostrarán al usuario, tales como las definiciones que se etiquetarán, los grupos, los campos que el usuario debe de llenar, etc. También, en caso de ser

necesario, crea la muestra aleatoria. Se compone de 4 funciones de las cuales solo 1 tiene interacción directa con el etiquetador.

#### **Código fuente relacionado con el método:**

---

**Función:** relacion

#### **Variables:**

- **muestra:** Lista de definiciones que se van a etiquetar, es solo una lista de ids
- **definiciones\_merge:** Una unión de todas las definiciones de los términos seleccionados por el usuario.
- **Grupos\_merge:** Lista de todos los grupos de todos los términos.
- **definición:** La primer definición de la muestra, contiene todos los datos de la definición como son el id, la definición y el id de término al que está relaciona
- **término:** Es el término al cual está relacionada la definición.
- **Historial:** Historial de palabras clave usadas por otros usuarios para esa definición.
- **Grupos:** Todos los grupos que se han creado para ese término
- **Total\_usuarios:** El número de usuarios registrados
- **Porcentaje:** Lista de porcentajes que indican que tanto ha sido utilizado un grupo.
- **Términos:** Cadena de texto que contiene los id de los términos seleccionados por el usuario
- **Términos\_texto:** Cadena de texto que contiene los términos seleccionados en el formato "texto=" para las url.

**Vista relacionada:** [general/relacion](#)

**Tarea:** Crea las variables que se utilizarán en la interfaz para mostrar las definiciones, los grupos y los apoyos que se ofrecen a los etiquetadores, además crea la muestra aleatoria.

#### **Explicación del flujo:**

Se crea una lista vacía que será la muestra aleatoria y otras dos listas que serán las uniones de las definiciones y los grupos de los términos seleccionados por el etiquetador, se hace una comprobación de que se hayan seleccionado más de un término, después revisamos si el etiquetador ya estaba trabajando en una muestra aleatoria, si no es así entonces procedemos a crear la muestra y mostramos un mensaje diciendo que se acaba de crear la muestra, si ya hay una muestra anterior entonces hacemos que esa muestra sea igual a la que ya existe y, si es el caso, le decimos que continúe desde la sesión anterior.

Después simplemente creamos variables que se utilizarán en la interfaz y revisamos si hay algún error en el previo llenado de datos.

Como se utilizan más de un término es necesario indicarlo en la interfaz para llamar a la creación de grupos, para ello se crean dos cadenas con los id y el nombre de los términos en un formato para url.

#### **Adornos adicionales:**

- **@auth.requires\_login()**: Indica que es necesario un inicio de sesión para que el usuario tenga acceso a esta función y vista.

#### Ejemplos de modificación:

- **Si se quiere agregar algún dato extra:** La muestra contiene datos básicos pero con ellos se pueden obtener más, solo hay que revisar la documentación de Web2Py y de bases de datos relacionales.

#### Pseudocódigo:

Inicio función relación:

```

Muestra = lista
Definiciones_merge = lista
Grupos_merge = lista
Si hay menos de dos términos seleccionados
    Re direcciona a inicio
Fin si
Si no hay una muestra previa entonces
    Para cada elemento en termino hacer
        Agregar las definiciones del término a definiciones_merge
    Fin para cada
    Muestra = getMuestra
    Mostrar mensaje 'Se ha creado lista aleatoria'
Si no
    Muestra = la muestra de la base de datos
    Si rd = 1 entonces
        Muestra mensaje 'Continua desde la última sesión'
    Fin Si
Fin Si
Definición = consulta la base de datos con muestra[0]
Termino = consulta la base de datos con definición
Historial = getHistorialKeywords
Si error = 0
    Mostrar mensaje 'Favor de llenar campos'
Si no si error = 1
    Mostrar mensaje 'Las palabras clave no se encuentran en la definicion'
Fin si
Total_usuarios: consulta todos los usuarios
Términos = ""
Terminos_texto = ""
Para cada elemento en términos hacer
    Agregar los grupos del término a grupos_merge
    Términos += "termino=" elemento + "&"
    Términos_texto += "texto=" + el nombre del término + "&"
Fin Para cada
Porcentaje = getPorcentajes
Términos += "c=0"
Términos_texto += "c=0"

```

Regresar locals

Fin

### Código de la función:

```
@auth.requires_login()
def relacion():
    muestra = list()
    definiciones_merge = list()
    grupos_merge = list()
    if len(request.vars.termino) < 2:
        redirect(URL('inicio?error=1'))
    if (len(db((db.temporal.usuario_id == auth.user.id) & (db.temporal.modos == '2')).select()) == 0):
        for elemento in request.vars.termino:
            definiciones_merge.append(db((db.definicion.termino_id==elemento).select()))
        muestra = getMuestra(definiciones_merge)
        response.flash = 'Se ha creado una lista aleatoria'
    else:
        muestra = db((db.temporal.usuario_id == auth.user.id) & (db.temporal.modos == '2')).select()
        if (request.vars.rd == "1"):
            response.flash = 'Continúa desde la última sesión'
        definicion = db((db.definicion.id==muestra[0]['definicion_id']).select())[0]
        termino = db((db.termino.id==definicion['termino_id']).select())[0]
        historial = getHistorialKeywords(db((db.relacion.definicion_id==definicion['id']) & (db.relacion.modos ==
"2")).select())
        if (request.vars.error == '0'):
            response.flash = 'Favor de llenar los campos'
        elif request.vars.error == '1':
            response.flash = 'Las palabras clave no se encuentran en la definición'
        total_usuarios = len(db((db.auth_user.id > 0)).select())
        terminos = ""
        terminos_texto = ""
        grupos_merge = list()
        for elemento in request.vars.termino:
            grupos_merge.append(db((db.grupo.termino_id == elemento) & (db.grupo.modos == '2')).select())
            terminos += "termino=" + str(elemento) + "&"
            terminos_texto += "texto=" + str(db((db.termino.id==elemento).select())[0]['ter']) + "&"
        porcentaje = getPorcentajes(request.vars.termino)
        terminos += "c=0"
        terminos_texto += "c=0"
    return locals()
```

---

### Función: getMuestra

#### Parámetros:

- **Lista\_definiciones:** Lista de todas las definiciones de los términos seleccionados.

#### Variables:

- **i:** Contador.
- **Muestra:** Lista de definiciones en la muestra.
- **Max:** Numero de definiciones que habrá en la muestra
- **Tam:** Suma de definiciones en todos los términos

- **Termino:** Numero aleatorio que representa el término que se usará
- **Definición:** Numero aleatorio que representa una definición

**Tarea:** Crea una lista de números aleatorios que será la muestra.

### Explicación del flujo:

Después de crear un par de variables auxiliares revisamos si hay suficientes definiciones para hacer una muestra de 10, si no entonces se hará con las que hay disponibles.

Se hace una iteración mientras no haya 10 elementos en la muestra se hacen dos números aleatorios, como se maneja una lista de listas se necesitan dos números aleatorios, uno para seleccionar la lista y otro para seleccionar la posición de la lista, se revisa que aún no esté en la lista y si es así se agrega la definición relacionada a la muestra, finalmente se registra esa lista en la base de datos y se regresa.

### Ejemplos de modificación:

- **Si se quiere hacer mayor la muestra:** Solo hay que cambiar el valor de max:  
Max = nuevo\_valor

### Pseudocódigo:

```

Inicio función getMuestra
    I = 0
    Muestra = lista
    Max = 10
    Tam = 0
    Para cada lista en lista_definiciones hacer
        Tam += tamaño de la lista
    Fin para cada
    Si el tamaño de la lista de definiciones < 10 entonces
        Max = tamaño de la lista de definiciones
    Fin si
    Mientras i < max hacer
        Termino = numero aleatorio
        Definición = numero aleatorio
        Si no esta en la lista entonces
            Se agrega la definición a la lista
            I++
        Fin si
    Fin mientras
    registrarLista
    Regresa la lista
Fin
  
```

### Código de la función:

```

def getMuestra(lista_definiciones):
    i = 0
    muestra = list()
    max = 10
  
```

```

tam = 0
for lista in lista_definiciones:
    tam += len(lista)
if(tam < 10):
    max = tam
while (i < max):
    termino = int(random.uniform(0, len(lista_definiciones) - 1))
    definicion = int(random.uniform(0, len(lista_definiciones[termino]) - 1))
    if noEnLista(lista_definiciones[termino][definicion]['id'], muestra):
        muestra.append(lista_definiciones[termino][definicion])
    i = i + 1
registraLista(muestra)
return db((db.temporal.usuario_id == auth.user.id) & (db.temporal.modos == '2')).select()

```

---

**Función:** noEnLista

**Parámetros:**

- **idDefinicion:** Numero aleatorio que se desea agregar a la lista
- **muestra:** Lista de definiciones aleatorias

**Variables:**

- **flag:** Booleano que nos indica si la definición ya está en la lista.

**Tarea:** Revisa la muestra existente para ver si cierta definición ya está en la lista

**Explicación del flujo:**

Se crea una variable booleana, se hace una iteración en la muestra y se revisa si algún dato es igual, si es así la booleana se vuelve falsa, se regresa esa variable.

**Pseudocódigo:**

```

Inicio función noEnLista
    Flag = true
    Para cada definición en muestra hacer
        Si definición['id'] = idDefinicion
            Flag = false
        Fin si
    Fin Para cada
    Regresa flag
Fin

```

**Código de la función:**

```

def noEnLista(idDefinicion, muestra):
    flag = True
    for definicion in muestra:
        if definicion['id'] == idDefinicion:
            flag = False
    return flag

```

---

### Modulo: Crear historial de palabras clave

**Descripción:** Este módulo solo se compone por una función, tiene como entrada una lista de definiciones ya etiquetadas. Básicamente lee todas esas definiciones y almacena las palabras clave utilizadas para cierta definición en un diccionario.

#### Código fuente relacionado con el método:

---

**Función:** getHistorialKeywords

#### Parámetros:

- **lista:** Una lista con los etiquetados de una definición y las palabras clave que se utilizaron.

#### Variables:

- **frase\_clave:** Lista con las palabras clave utilizadas.
- **Aux:** Lista con las palabras clave utilizadas en un registro
- **Conteo\_frase:** Diccionario con el conteo de las palabras clave
- **Ordenado:** Diccionario ordenado

#### Vista relacionada: [general/relacion](#)

**Tarea:** Revisa los registros para cierta definición y cuenta las palabras clave, las almacena en un diccionario.

#### Explicación del flujo:

Primero hace una iteración para cada registro en la lista, por cada registro obtiene una lista de palabras clave y hace una iteración sobre eso, si tiene un espacio antes lo elimina, al final de esa iteración agrega la frase a una lista.

Después hace una nueva iteración para cada frase en frase\_clave, revisa si está en el diccionario y dependiendo de eso lo suma o crea una nueva llave con valor 1, finalmente ordena el diccionario.

#### Ejemplos de modificación:

- **El diccionario puede ser reordenado o modificar cuantos elementos hay.**

#### Pseudocódigo:

```
Inicio función getHistorialKeywords
  Frase_clave = lista
  Para cada registro en lista hacer
    Aux = separar las keywords
    Para cada frase en aux hacer
      Si frase[0] = ' ' entonces
        Frase = frase ignorando el primer carácter
      Fin si
      Agregar frase a frase_clave
    Fin Para cada
```

```

Fin Para cada
Conteo_frase = diccionario
Para cada frase en frase_clave hacer
    Si frase esta en conteo_frase
        Conteo_frase[frase] += 1
    Si no
        Conteo_frase[frase] = 1
    Fin si
Fin para cada
Ordenado = ordenar diccionario
Fin

```

### Código de la función:

```

def getHistorialKeywords(lista):
    frase_clave = []
    for registro in lista:
        aux = registro['keywords'].split(',')
        for frase in aux:
            if frase[0] == ' ':
                frase = frase[1:]
            frase_clave.append(frase)
    conteo_frase = {}
    for frase in frase_clave:
        if frase in conteo_frase:
            conteo_frase[frase] += 1
        else:
            conteo_frase[frase] = 1
    ordenado = sorted(conteo_frase.items(), key=operator.itemgetter(1), reverse=True)
    return ordenado

```

---

Modulo: Crear historial de definiciones usadas en un grupo

**Descripción:** Este módulo solo se compone por una función, esta función es llamada usando Ajax, revisa una lista de grupos y genera una lista de definiciones usadas en ese grupo.

### Código fuente relacionado con el método:

---

**Función:** getHistorialGrupos

#### Variables:

- **respuestaHtml:** Respuesta que devolverá a la función Ajax
- **user:** usuario que hace la consulta
- **historial:** Historial de los grupos usados por el usuario
- **grupoNombre:** Nombre del grupo
- **contador:** Un contador
- **aux:** Variable de apoyo

**Vista relacionada:** [general/relacion](#)



**Tarea:** Recibe uno o más grupos, con ese dato busca en la base de datos las relaciones hechas por el etiquetador usando ese grupo y la definición que uso, agrega eso a la respuesta en código HTML y lo regresa.

#### **Explicación del flujo:**

Crea variables de apoyo, después revisa si se manda más de un grupo si esa así hace una iteración para cada grupo en la lista, obtiene todas las relaciones hechas por un usuario usando ese grupo, con eso obtiene el nombre del grupo y añade ese nombre a la respuesta, declara un contador y una auxiliar, revisa si hay suficientes elementos en el historial para devolver 10 resultados, si no entonces devuelve los que están disponibles, mientras aún no se hayan agregado 10 resultados (o los disponibles) se agrega a la respuesta la definición usada.

Si no es una lista de grupos obtiene todas las relaciones hechas por un usuario usando ese grupo, con eso obtiene el nombre del grupo y añade ese nombre a la respuesta, declara un contador y una auxiliar, revisa si hay suficientes elementos en el historial para devolver 10 resultados, si no entonces devuelve los que están disponibles, mientras aún no se hayan agregado 10 resultados (o los disponibles) se agrega a la respuesta la definición usada.

#### **Pseudocódigo:**

Inicio función getHistorialGrupos

    respuestaHTML = ""

    user = c

    Si se mandó al menos un grupo entonces

        Si hay más de un grupo entonces

            Para cada grupo en grupos hacer

                Historial = lista de grupos usados por ese usuario

                grupoNombre = nombre del grupo

                respuestaHTML += grupoNombre

                contador = 10

                aux = tamaño del historial

                Si el tamaño del historial < 10 entonces

                    Contador = tamaño del historial

                Fin si

                Mientras contador > 0 hacer

                    respuestaHTML += definición

                    contador—

                    aux—

                Fin mientras

                respuestaHTML += "</p>"

            Fin para

        Si no

            Historial = lista de grupos usados por ese usuario

            grupoNombre = nombre del grupo

            respuestaHTML += grupoNombre

            contador = 10

            aux = tamaño del historial

```

        Si el tamaño del historial < 10 entonces
            Contador = tamaño del historial
        Fin si
        Mientras contador > 0 hacer
            respuestaHTML += definición
            contador—
            aux—
        Fin mientras
        respuestaHTML += "</p>"
    Fin si
    Si no
        respuestaHTML = "Seleccione un grupo"
    Fin si
    Regresa respuestaHTML

Fin

Código de la función:

def getHistorialGrupos():
    respuestaHtml = ""
    user = request.vars.c
    if request.vars.grupos != None:
        if isinstance(request.vars.grupos, list):
            for grupo in request.vars.grupos:
                historial = db((db.relacion.grupo_id == grupo) & (db.relacion.usuario == user) &
(db.relacion.modos == "1")).select()
                grupoNombre = db((db.grupo.id == grupo) & (db.grupo.modos == "1")).select()[0]['grup']
                respuestaHtml += "<p><strong>Grupo:</strong> " + grupoNombre + "<br>"
                contador = 10
                aux = len(historial) - 1
                if (len(historial) < 10):
                    contador = len(historial)
                    while(contador > 0):
                        respuestaHtml += "<strong>Definición: </strong>" + db(db.definicion.id ==
historial[aux]['definicion_id']).select()[0]['defi'] + "<br>"
                        contador -= 1
                        aux -= 1
                    respuestaHtml += "</p>"
                else:
                    historial = db((db.relacion.grupo_id == request.vars.grupos) & (db.relacion.usuario == user) &
(db.relacion.modos == "1")).select()
                    grupoNombre = db((db.grupo.id == request.vars.grupos) & (db.grupo.modos ==
"1")).select()[0]['grup']
                    respuestaHtml += "<p><strong>Grupo:</strong> " + grupoNombre + "<br>"
                    contador = 10
                    aux = len(historial) - 1
                    if (len(historial) < 10):
                        contador = len(historial)
                        while(contador > 0):
                            respuestaHtml += "<strong>Definición: </strong>" + db(db.definicion.id ==
historial[aux]['definicion_id']).select()[0]['defi'] + "<br>"
                            contador -= 1
                            aux -= 1

```

```
    respuestaHtml += "</p>"
else:
    respuestaHtml = "Selecciona un grupo"
return respuestaHtml
```

---

Modulo: Obtener porcentaje de uso para cada grupo

**Descripción:** Este módulo solo se compone por una función, su objetivo es calcular el porcentaje de uso de cada grupo basado en el número de usuarios

**Código fuente relacionado con el método:**

---

**Función:** getPorcentajes

**Parámetros:**

- **grupos:** Lista de relaciones en la base de datos

**Variables:**

- **porcentaje:** Diccionario que almacena la cantidad de veces que ha sido utilizado un grupo

**Vista relacionada:** [general/relacion](#)

**Tarea:** Guarda el número de usos de un grupo por usuarios

**Explicación del flujo:**

Después de crear el diccionario hace una iteración para cada registro en la base de datos, comprobamos que el grupo este en el diccionario y que el usuario no este, si es así le suma un elemento al porcentaje de ese grupo, si no es así entonces agrega el grupo al diccionario y el usuario.

**Pseudocódigo:**

```
Inicio función getHistorialPorcentaje
    Porcentaje = diccionario
    Grupos = consulta de todos los grupos que se han usado
    Para cada grupo en grupos hacer
        Si el grupo está en porcentajes y el usuario no está en porcentajes entonces
            Porcentaje[grupo] ++
        Si no
            Porcentaje[grupo] = 1
            Porcentaje[usuario] = 1
        Fin si
    Fin para cada
    Regresa porcentaje
Fin
```

**Código de la función:**

```
def getPorcentajes(terminos):
    porcentaje = dict()
    grupos = db((db.relacion.modos == "2")).select()
    for grupo in grupos:
```

```
if grupo['grupo_id'] in porcentaje and grupo['usuario'] not in porcentaje:
    porcentaje[grupo['grupo_id']] += 1
else:
    porcentaje[grupo['grupo_id']] = 1
    porcentaje["u" + str(grupo['usuario'])] = 1
return porcentaje
```

---

#### Modulo: Crear la relación

**Descripción:** Este es el modulo más importante del modo general ya que se encarga de tomar los datos que el etiquetador capturo en el sistema y los escribe en la base de datos, es el action del formulario de relación.

#### Código fuente relacionado con el método:

---

**Función:** unir

**Vista relacionada:** [general/relacion](#)

**Tarea:** Escribir los datos proporcionados por el usuario en la base de datos.

#### Explicación del flujo:

Primero hace una validación para ver si los datos fueron introducidos de manera correcta, si es no es así re direcciona a la página de relación, si todo está bien entonces revisa si el etiquetador selecciono más de un grupo, en ese caso primero crea una lista de grados y un contador, mientras ese contador sea mejor que el tamaño de la lista de grados, es decir, mientras haya al menos un grado que no ha sido registrado, el sistema registrara los datos proporcionados por el etiquetador.

Si solo se seleccionó un grupo entonces solo se inserta el registro sin necesidad de hacer una lista de grados de certeza.

Cuando se termina de hacer eso simplemente se elimina esa definición de la muestra aleatoria y si ya no quedan elementos entonces se manda a la página de inicio, en caso contrario se manda de nuevo a la página de relación.

#### Ejemplos de modificación:

- En esta parte se insertan los datos requeridos, en caso de que se desee agregar más contenido a la base de datos entonces, después de modificar el archivo db.py, aquí se pueden agregar esos datos extra, la documentación de Web2Py explica como modificar el insert para que acepte más campos.

#### Pseudocódigo:

Inicio función unir

    Si los datos no son válidos entonces

        Re direccionar a relación

    Si no

        Si grupo es una lista entonces

            Grado = lista de grados

```

        l = 0
        Mientras i < tamaño de grado hacer
            Insertar datos
            l++
        Fin mientras
    Si no
        Insertar datos
    Fin si
    Eliminar registro temporal
    Si ya no hay más definiciones en la muestra entonces
        Re direccionar a inicio
    Si no
        Re direccionar a relación
    Fin si
Fin so
Regresa locals()
Fin

```

#### Código de la función:

```

@auth.requires_login()
def unir():
    """
    Crea la relación entre una definición y un grupo, además verifica algunas validaciones previas
    """
    if(request.vars.keyword == " or request.vars.grupo == None): #Validación de campos vacíos
        redirect(URL('relacion?error=0'))
    else:
        if isinstance(request.vars.grupo, list):
            request.vars.grado = [x for x in request.vars.grado if x != " ]
            i = 0
            while i < len(request.vars.grupo):
                db.relacion.insert(keywords=request.vars.keyword, grado=request.vars.grado[i],
                grupo_id=request.vars.grupo[i], definicion_id=request.vars.definicion, usuario=auth.user.id, modo="2")
                i = i + 1
            else:
                db.relacion.insert(keywords=request.vars.keyword, grado='100', grupo_id=request.vars.grupo,
                definicion_id=request.vars.definicion, usuario=auth.user.id, modo="2")
                db((db.temporal.usuario_id==auth.user.id) & (db.temporal.definicion_id == request.vars.definicion) &
                (db.temporal.modo == "2")).delete()
                if(len(db((db.temporal.usuario_id==auth.user.id) & (db.temporal.modo == "2")).select()) == 0):
                    redirect(URL('inicio?en=1'))
                else:
                    redirect(URL('relacion?' + request.vars.auxiliar))
    return locals()

```

---

#### Interfaz.py

Se encarga de manejar la interfaz administrativa, por el momento solo cuenta con tres funciones que sirven para un solo modulo y una función extra cuya única funcionalidad es mostrar la interfaz,

está funcionalidad se puede expandir cuando se desee generar archivos con diferentes datos, para ello se puede utilizar la información que se encuentra en la base de datos.

Modulo: Inicio

**Descripción:** Modulo que genera las variables que se mostrarán y usarán en la página de inicio de la interfaz administrativa.

**Código fuente relacionado con el método:**

---

**Función:** inicio

**Variables:**

- **Usuario\_nombre:** Nombre del usuario
- **Usuario\_mail:** Correo electrónico del usuario

**Vista relacionada:** [interfaz/relacion](#)

**Tarea:** Crear variables que se utilizarán en la interfaz gráfica.

**Explicación del flujo:**

Crea las variables.

**Pseudocódigo:**

```
Inicio función inicio
    Usuario_nombre = nombre del usuario
    Usuario_mail = correo electrónico del usuario
    Regresa locals()
Fin
```

**Código de la función:**

```
@auth.requires_login()
def inicio():
    usuario_nombre = auth.user.first_name
    usuario_mail = auth.user.email
    return locals();
```

---

Modulo: Genera archivo

**Descripción:** Modulo que genera los archivos solicitados en un formato de json.

**Código fuente relacionado con el método:**

---

**Función:** generaParticular

**Variables:**

- **Salida:** Diccionario que será la salida del programa.
- **Relaciones:** Todos los etiquetados que se han realizado.

- **Aux:** variable de apoyo
- **Definición\_all:** Todos los datos de una definición, basado en el id del etiquetado.
- **Definición:** Texto de la definición
- **Termino\_all:** Todos los datos de un término, basado en el id del etiquetado.
- **Termino:** Nombre del término.
- **gruposPromedio:** Diccionario que tendrá el promedio de grado de certeza
- **grupos:** La relación que se está manejando.
- **gruposKey:** Diccionario con las palabras clave utilizadas.

**Tarea:** Escribir los datos proporcionados por el usuario en la base de datos.

#### Explicación del flujo:

Crea la variable donde se almacenará la salida y obtiene todos los etiquetados que se han hecho en ese modo, después hace una iteración para cada relación donde define una variable auxiliar y cambia los id la información que el usuario entiende (el texto de la definición, el nombre del término, etc), después crea un diccionario con el promedio del grado de certeza y otro diccionario con las palabras clave utilizadas, agrega esos datos a la variable auxiliar y agrega esa variable a la salida.

#### Ejemplos de modificación:

- Cualquier dato o estadística que se quiera agregar a la salida solo se debe de agregar a la variable auxiliar:  
Desviación = getDesviacion(datos)  
Aux['desviacion'] = desviacion

#### Pseudocódigo:

Inicio función generaParticular

Salida = diccionario

Relaciones = todas las relaciones hechas para este modo

I = 0

Para cada relación en relaciones hacer

Aux = diccionario

Definición\_all = datos de la definición

Definición = texto de la definición

Termino\_all = datos del término

Termino = nombre del término

gruposPromedio = diccionario

grupos = todos los datos de la relación que estamos trabajando

Para cada grupo en grupos hacer

gruposPromedio['Grupo'] = nombre del grupo

gruposPromedio['promedio'] = getPromedio

Fin para cada

gruposKey = diccionario

grupos = todos los datos de la relación que estamos trabajando

Para cada grupo en grupos hacer

gruposKey['Grupo'] = nombre del grupo

gruposKey['keywords'] = keywords del grupo

```

        Fin para cada
        aux['id'] = relacion['id']
        aux['termino'] = termino
        aux['definicion'] = definicion
        aux['Grupo-Promedio'] = gruposPromedio
        aux['Grupo-Keywords'] = gruposKey
        salida[i] = aux
        i += 1
    Fin para cada
    Regresa salida
Fin

```

### Código de la función:

```

def generaParticular():
    salida = dict()
    relaciones = db(db.relacion.modo == "1").select()
    i = 0
    for relacion in relaciones:
        aux = dict()
        definicion_all = db(db.definicion.id == relacion['definicion_id']).select()[0]
        definicion = definicion_all['defi']
        termino_all = db(db.definicion.id == definicion_all['id']).select()[0]['termino_id']
        termino = db(db.termino.id == termino_all).select()[0]['ter']
        gruposPromedio = dict()
        grupos = db(db.relacion.id == relacion['id']).select()
        for grupo in grupos:
            gruposPromedio['grupo'] = db(db.grupo.id == grupo['grupo_id']).select()[0]['grup']
            gruposPromedio['promedio'] = getPromedio(grupo['grupo_id'])
        gruposKey = dict()
        for grupo in grupos:
            gruposKey['grupo'] = db(db.grupo.id == grupo['grupo_id']).select()[0]['grup']
            gruposKey['keywords'] = grupo['keywords']
        aux['id'] = relacion['id']
        aux['termino'] = termino
        aux['definicion'] = definicion
        aux['Grupo-Promedio'] = gruposPromedio
        aux['Grupo-Keywords'] = gruposKey
        salida[i] = aux
        i += 1
    return salida

```

---

**Función:** generaGeneral

### Variables:

- **Salida:** Diccionario que será la salida del programa.
- **Relaciones:** Todos los etiquetados que se han realizado.
- **Aux:** variable de apoyo
- **Definición\_all:** Todos los datos de una definición, basado en el id del etiquetado.
- **Definición:** Texto de la definición



- **Termino\_all:** Todos los datos de un término, basado en el id del etiquetado.
- **Termino:** Nombre del término.
- **gruposPromedio:** Diccionario que tendrá el promedio de grado de certeza
- **grupos:** La relación que se está manejando.
- **gruposKey:** Diccionario con las palabras clave utilizadas.

**Tarea:** Escribir los datos proporcionados por el usuario en la base de datos.

### Explicación del flujo:

Crea la variable donde se almacenará la salida y obtiene todos los etiquetados que se han hecho en ese modo, después hace una iteración para cada relación donde define una variable auxiliar y cambia los id la información que el usuario entiende (el texto de la definición, el nombre del término, etc), después crea un diccionario con el promedio del grado de certeza y otro diccionario con las palabras clave utilizadas, agrega esos datos a la variable auxiliar y agrega esa variable a la salida.

### Ejemplos de modificación:

- Cualquier dato o estadística que se quiera agregar a la salida solo se debe de agregar a la variable auxiliar:  
Desviación = getDesviacion(datos)  
Aux['desviacion'] = desviacion

### Pseudocódigo:

```

Inicio función generaGeneral
  Salida = diccionario
  Relaciones = todas las relaciones hechas para este modo
  I = 0
  Para cada relación en relaciones hacer
    Aux = diccionario
    Definición_all = datos de la definición
    Definición = texto de la definición
    Termino_all = datos del término
    Termino = nombre del término
    gruposPromedio = diccionario
    grupos = todos los datos de la relación que estamos trabajando
    Para cada grupo en grupos hacer
      gruposPromedio['Grupo'] = nombre del grupo
      gruposPromedio['promedio'] = getPromedio
    Fin para cada
    gruposKey = diccionario
    grupos = todos los datos de la relación que estamos trabajando
    Para cada grupo en grupos hacer
      gruposKey['Grupo'] = nombre del grupo
      gruposKey['keywords'] = keywords del grupo
    Fin para cada
    aux['id'] = relacion['id']
    aux['termino'] = termino

```

```

        aux['definicion'] = definicion
        aux['Grupo-Promedio'] = gruposPromedio
        aux['Grupo-Keywords'] = gruposKey
        salida[i] = aux
        i += 1
    Fin para cada
    Regresa salida
Fin

```

### Código de la función:

```

def generaGeneral():
    salida = dict()
    relaciones = db(db.relacion.modulo == "2").select()
    i = 0
    for relacion in relaciones:
        aux = dict()
        definicion_all = db(db.definicion.id == relacion['definicion_id']).select()[0]
        definicion = definicion_all['defi']
        termino_all = db(db.definicion.id == definicion_all['id']).select()[0]['termino_id']
        termino = db(db.termino.id == termino_all).select()[0]['ter']
        gruposPromedio = dict()
        grupos = db(db.relacion.id == relacion['id']).select()
        for grupo in grupos:
            gruposPromedio['grupo'] = db(db.grupo.id == grupo['grupo_id']).select()[0]['grup']
            gruposPromedio['promedio'] = getPromedio(grupo['grupo_id'])
        gruposKey = dict()
        for grupo in grupos:
            gruposKey['grupo'] = db(db.grupo.id == grupo['grupo_id']).select()[0]['grup']
            gruposKey['keywords'] = grupo['keywords']
        aux['id'] = relacion['id']
        aux['termino'] = termino
        aux['definicion'] = definicion
        aux['Grupo-Promedio'] = gruposPromedio
        aux['Grupo-Keywords'] = gruposKey
        salida[i] = aux
        i += 1
    return salida

```

---

**Función:** getPromedio

#### Parámetros:

- **idGrupo:** id del grupo del cual vamos a obtener el promedio de grados de certeza

#### Variables:

- **grupos:** Lista de etiquetados en la cual se utilizó un grupo
- **suma:** Suma de los grados de certeza

**Tarea:** Obtiene el promedio de los grados de certeza usados en un grupo

#### Explicación del flujo:

Obtiene las relaciones en las cuales se ha usado un grupo en específico, se hace una iteración en esa lista y se suman los valores del grado de certeza. Se regresa esa suma entre la cantidad de grupos.

#### Pseudocódigo:

Inicio función getPromedio

    Grupos = lista de etiquetados en los que se usó un grupo

    Suma = 0

    Para cada grupo en grupos hacer

        Suma = grado del grupo

    Fin para cada

    Regresa suma / tamaño de grupos

Fin

#### Código de la función:

```
def getPromedio(idGrupo):
    grupos = db(db.relacion.grupo_id == idGrupo).select()
    suma = 0
    for grupo in grupos:
        suma = int(grupo['grado'])
    return float(suma / len(grupos))
```

---

#### Default.py

Este controlador solo se utiliza para generar la interfaz para el manejo de usuarios (registro, inicio de sesión, reinicio de contraseña), para ello es necesario utilizar un correo electrónico que nos permita enviar mensajes desde una aplicación externa (Con Gmail hay que habilitar esta opción), de manera temporal se utiliza Gmail pero si se dispone de un correo personalizado del servidor se puede remplazar en la función user de default.

```
mail = auth.settings.mailer #Ajustes del mail
mail.settings.server = 'smtp.gmail.com:587' #Servidor que se usará
mail.settings.sender = 'define.gil@gmail.com' #Correo desde el cual se enviarán las notificaciones
mail.settings.login = 'define.gil:defineadmin' #Datos de acceso del correo
auth.settings.registration_requires_verification = True
auth.settings.login_after_registration = False
auth.settings.registration_requires_approval = False
auth.settings.reset_password_requires_verification = True
auth.messages.verify_email = 'Click on the link http://' + request.env.http_host +
URL(r=request,c='default',f='user',args=['verify_email']) + ' /(key)s to verify your email'
auth.messages.reset_password = 'Click on the link http://' + request.env.http_host +
URL(r=request,c='default',f='user',args=['reset_password']) + ' /(key)s to reset your password'
```

---

## Vistas

El tercer elemento en el modo de trabajo de Web2Py son las vistas, estas están directamente relacionadas con ciertas funciones Python en las cuales se procesará la información y las vistas solo la mostrarán. Pueden utilizar HTML, css, JavaScript, Ajax, jQuery, entre otras tecnologías web.

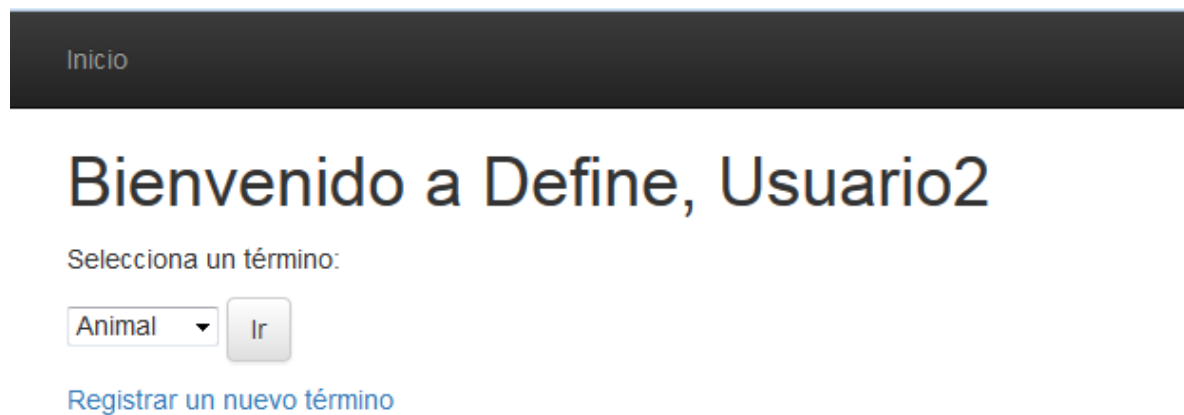
El punto en común de la mayoría de las vistas es que son una extensión de layout.html, es el diseño por default que nos ofrece Web2Py, en la documentación de Web2Py se puede encontrar más información al respecto.

### Vistas – Modo particular

En el modo particular se utilizan 5 vistas diferentes, 4 de ellas tienen solo una función específica y poco o nada de código JavaScript, la otra vista es la principal y en ella se mostrará distintos tipos de información al etiquetador, también es la que tiene una gran cantidad de JavaScript que permite, entre otras cosas, la validación de campos, asignación de valores y mostrar distintas opciones.

Vista: Inicio

Modulo relacionado: [Inicio](#)



**Descripción:** Muestra un mensaje de bienvenida, si hay términos registrados muestra una lista con todos los términos y un enlace para crear más términos, si no se ha registrado ninguno solo muestra el enlace.

**Funcionalidad dinámica:**

- **Mostrar el nombre de usuario:**  
`<h1>Bienvenido a Define, {{=usuario_nombre}}</h1>`
- **Mostrar los términos o un mensaje pidiendo que se registre un término.**  
`{{if len(termino) == 1:}}  
<p> Primero necesitas <a href="importar">registrar un término</a></p>  
{{else:}}`

<p>Selecciona un término:</p>

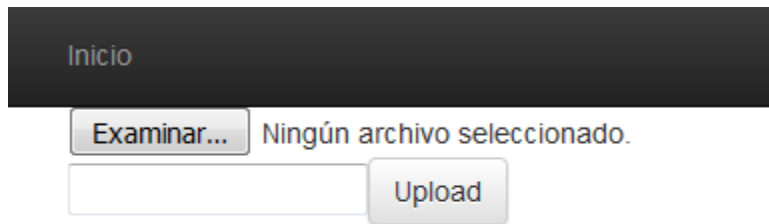
<p>

- **Mostrar los términos registrados:**

```
<select name="termino">
  {{for term in termino:
    if not term['ter'] in 'Nulo':}}
  <option value="{{=term['id']}}">{{=term['ter']}}</option>
  {{pass}}
  {{pass}}
</select>
```

Vista: Importar

Modulo relacionado: [Importar](#)



**Descripción:** Es un formulario en el cual se selecciona un archivo, se nombra el término y se agrega a la base de datos.

Vista: crearGrupo

Modulo relacionado: [Crear grupos](#)



**Descripción:** La interfaz solo nos pide el nombre del grupo personalizado que queremos crear, adicionalmente hace la verificación de si existe el grupo o si no se llenó el campo de nombre.

### Funcionalidad dinámica:

- Obtener la cadena de los grupos existentes y del término al que está relacionado para asignarlos a un campo hidden:

```
<input type="hidden" id="cadena_grupos" value="{{=cadena_grupos}}" />
<input type="hidden" value="{{=termino}}" name="termino" />
```

### Funciones JavaScript:

---

#### Función: Addgroup

**Tarea:** Hace una validación de datos y si todo está correcto manda el formulario al servidor.

#### Explicación del flujo:

Comprueba que se haya introducido el nombre del grupo, si es así entonces comprueba que el valor introducido no este registrado, en caso afirmativo entonces se manda el formulario al servidor, se refresca la ventana padre y se cierra esta ventana.

#### Pseudocódigo:

Inicio función AddGroup

```
Si nombre_id != "" entonces
    Si nombre_id no está en cadena_grupos entonces
        Submit
        Refrescar padre
        Cerrar ventana
    Si no
        Mensaje "El grupo ya existe"
    Fin si
Si no
    Mensaje "Ingrese el nombre del grupo"
Fin si
```

Fin

#### Código de la función:

```
function AddGroup(){
    if(document.getElementById("nombre_id").value != "")

if(document.getElementById("cadena_grupos").value.indexOf(document.getElementById("nombre_id").value) == -1){
    document.getElementById("GroupDB").submit();
    window.opener.location.reload();
    window.close();
} else
    alert("El grupo ya existe");
else
    alert("Ingresa el nombre del grupo");
}
```

---

## Grupos sugeridos para Animal

☐ animal

**Lemmas:** ['animal', 'animate\_being', 'beast', 'brute', 'creature', 'fauna'],

**Definition:** (a living organism characterized by voluntary movement),

**Examples:** No available

**Lemmas:** ['animal', 'carnal', 'fleshly', 'sensual'],

**Definition:** (marked by the appetites and passions of the body),

**Examples:** ['animal instincts', 'carnal knowledge', 'fleshly desire', 'a sensual delight in eating', 'sensual pleasure without vice']

[Crear los grupos seleccionados](#) o [Crea un grupo personalizado](#)

**Descripción:** Muestra los grupos sugeridos por WordNet para un término, cada sugerencia tiene un checkbox para que se pueda seleccionar, además de datos sobre esa sugerencia, se puede crear los grupos seleccionado o crear uno personalizado.

### Funcionalidad dinámica:

- **Mostrar el término para el cual se sugieren los grupos:**  
<h2>Grupos sugeridos para {{=request.vars.texto}}</h2>
- **Mostrar los grupos propuestos por WordNet, eliminando algunas cadenas de apoyo:**

```
{{
  i = 0
  while (i < len(wordnet)):
    if ('@' in wordnet[i]):
      if(wordnet[i+1] != '-----'):}}
      <p>{{=XML(wordnet[i].replace('@', ''))}}</p>
      {{pass}}
    {{else:
      if(wordnet[i] != '-----')}}
      <p>{{=XML(wordnet[i])}}</p>
      {{pass}}
    {{pass}}
    {{i += 1}}
  {{pass}}
  {{if(len(wordnet) == 0):}}
  <p>
    No hay más sugerencias
```

## Funciones JavaScript:

---

**Función:** Addgroup

**Tarea:** Manda el formulario al servidor.

**Explicación del flujo:**

Se manda el formulario al servidor, se refresca la ventana padre y se cierra esta ventana.

**Pseudocódigo:**

Inicio función AddGroup  
    Submit  
    Refrescar padre  
    Cerrar ventana  
Fin

**Código de la función:**

```
function AddGroup(){  
    document.getElementById("GroupDB").submit();  
    window.opener.location.reload();  
    window.close();  
}
```

---

Vista: [relación](#)

Modulo relacionado: [relacion](#)



## Término: Animal

**Definición:** An animal is defined as 'at large' unless it is confined to a property by leash and tether (dogs only), fence, enclosure or under the

Seleccione el grupo semántico al que pertenece:

- ☐ No informativo
- ☐ Animal
- ☐ beast
- ☐ brute
- ☐ creature
- ☐ fauna
- ☐ carnal
- ☐ sensual
- ☐ Animal
- ☐ Animal
- ☐ Prueba

Opciones:

- [Crear grupos](#)
- [Mostrar el porcentaje de uso de cada grupo \(recuerda siempre usar tu propio criterio\)](#)
- [Ver las palabras clave usadas por otros usuarios](#)
- [Ver las definiciones que has usado en este grupo\(s\) \(últimas 10\)](#)

Palabras clave:

Enviar

**Descripción:** Esta vista contiene la definición que se etiquetará, los grupos existentes, la opción de desplegar distintos apoyos y el campo para las palabras clave, además de que valida que todo esté en orden y cuenta con diversas funciones JavaScript para poder desplegar los apoyos.

**Funcionalidad dinámica:**

- **Mostrar el término al que pertenece la definición:**  
`<h2>Término: {{=termino['ter']}}</h2>`
- **Mostrar la definición con la que se está trabajando:**  
`<strong>Definición: </strong>{{=definicion['defi']}}`
- **Agregar valores a campos hidden para que se manden junto con el formulario al servidor:**  
`<input value="{{=definicion['id']}}" type="hidden" name="definicion" />  
<input value="{{=termino['id']}}" type="hidden" name="termi" />`
- **Listar los grupos existentes, agregando un checkbox para seleccionar uno o más grupos, asignando los eventos y añadiendo otros campos que se utilizarán para los grupos:**  
`{{for grupo in grupos:}}  
    <p><input value="{{=grupo['id']}}" id="c{{=grupo['id']}}" type="checkbox"  
    name="grupo" onclick="setHistorial('{{=grupo['id']}}',  
    '{{=grupo['grup']}}');gradoAppear('{{=grupo['id']}}');setTexto('{{=grupo['grup']}}');"  
    />{{=grupo['grup']}} <p id="t{{=grupo['id']}}" style="display:none">Grado de certeza<input  
    type="text" id="g{{=grupo['id']}}" name="grado"  
    onKeyUp="validateAlpha('{{=grupo['id']}}')"/></p><p id="pgrupos" name="porgrupos"  
    style="display:none">{{if grupo['id'] in porcentaje.keys():}}Porcentaje de uso por otros  
    usuarios: {{=(float)((porcentaje[grupo['id']]*100) / (total_usuarios))}}{{else:}} No se ha  
    utilizado{{pass}}</p></p>  
{{pass}}`

- **Escribir el historial de palabras clave utilizadas:**

```

{{if len(historial) > 0:}}
    Palabras clave usadas por otros usuarios:<br>
    {{for historia in historial:}}
        {{=historia}}<br>
    {{pass}}
{{else:}}
    Aún nadie clasifica esta definición.
{{pass}}

```

## Funciones JavaScript:

---

**Función:** openWordnet

**Variables:**

- **Wn:** La ventana emergente

**Tarea:** Abre una ventana emergente para que se puedan crear los grupos.

**Explicación del flujo:**

Abre una ventana emergente con los datos recabados por el modulo.

**Pseudocódigo:**

```

Inicio función AddGroup
    Wn = abrir ventana
Fin

```

**Código de la función:**

```

function openWordnet() {
    var wn =
window.open("/DEFINE/particular/vistaWordNet?id={{=termino['id']}}&ter={{=termino['ter']}}", "_black",
"width=800, height=400, scrollbars=1");
}

```

---

**Función:** setHistorial

**Parámetros:**

- **idGrupo:** El id del grupo que se acaba de seleccionar
- **nombreGrupo:** El nombre del grupo que se acaba de seleccionar

**Variables:**

- **grupos\_historial:** Cadena de texto que contiene todos los grupos seleccionados en el formato "grupos=" para poder pasarlo como parámetro a la función de Ajax.

**Tarea:** Revisa cuando se selecciona o deselecciona un checkbox, crea una cadena de texto basado en eso y ejecuta una función Ajax que nos indicará el historial de definiciones usadas en un grupo o grupos.

#### Explicación del flujo:

Primero revisa si se seleccionó un grupo “No informativo”, si es así se limpia el historial de grupos, después se revisa si el campo está seleccionado, si es así se añade al historial, si no se elimina. Se asigna el valor del historial de grupos seleccionados a un campo hidden y se ejecuta una función Ajax que devolverá el historial y lo mostrará en el lugar indicado por la función Ajax.

#### Pseudocódigo:

Inicio función setHistorial

    Si nombreGrupo == “No informativo” entonces

        Grupos\_historial = “”

    Fin si

    Si Se seleccionó el grupo entonces

        Grupos\_historial += “grupos=” + idGrupo + “&”

    Si no

        Grupos\_historial = borrar el idGrupo de la cadena

    Fin si

    Grupos\_hidden = grupos\_historial

    Función de Ajax

Fin

#### Código de la función:

```
function setHistorial(idGrupo, nombreGrupo) {  
    if(nombreGrupo == "No informativo")  
        grupos_historial = "";  
    if(document.getElementById("c" + idGrupo).checked)  
        grupos_historial += "grupos=" + idGrupo + "&";  
    else  
        grupos_historial = grupos_historial.replace("grupos=" + idGrupo + "&", "");  
    document.getElementById("grupos_hidden").value = grupos_historial + "c={{=auth.user.id}}";  
    ajax('{{=URL(r=request,f='getHistorialGrupos')}}?' +  
document.getElementById("grupos_hidden").value,['grupos_hidden'],'historialGrupo');  
}
```

---

**Función:** setTexto

#### Parámetros:

- **tipo:** Contiene el valor del grupo, si es de tipo normal o no informativo

#### Variables:

- **anterior:** Valor actual del campo de texto en el que se pondrán las keywords.

**Tarea:** Lee el grupo que se seleccionó y si es del tipo “No informativo”, bloquea los demás grupos para que no puedan ser seleccionados y cambia el valor de las palabras clave a “No informativo”, si se deselecciona el grupo no informativo se vuelven a habilitar los grupos.

#### Explicación del flujo:

Si el valor en el campo de texto de las palabras clave es diferente a “Ni informativo” se guarda su valor, si se seleccionó el grupo “No informativo” se cambia el valor del campo de texto y se deshabilitan los checkbox de los grupos, se cuenta cuantos hay seleccionados y si no hay ninguno se vuelven a habilitar los grupos.

#### Pseudocódigo:

```
Inicio función setTexto
    Anterior = ""
    Si key != "No informativo"
        Anterior = key
    Fin si
    Si tipo == "No informativo"
        Key = "No informativo"
        Deshabilitar los grupos
        checkCount = todos los seleccionados
        Si checkCount == 0
            Habilitar los grupos
        Fin Si
    Si no
        Key = anterior
    Fin si
Fin
```

#### Código de la función:

```
function setTexto(tipo) {
    anterior = "";
    if(document.getElementById("key").value != "No informativo")
        anterior = document.getElementById("key").value;
    if (tipo == "No informativo"){
        document.getElementById("key").value = "No informativo";
        $(':checkbox[name=grupo]').not(':checked').attr('disabled', true);
        checkCount = $(':checked').length;
        if (checkCount == 0)
            $(':checkbox[name=grupo]:disabled').attr('disabled', false);
        //document.getElementById("key").setAttribute("disabled", true);
    } else {
        document.getElementById("key").value = anterior;
        //document.getElementById("key").setAttribute("disabled", false);
    }
}
```

---

**Función:** openHistorial

**Tarea:** Revisa la etiqueta de estilo del elemento con id “history”, si no se muestra entonces lo mostramos y cambiamos el texto de la opción, si no ocultamos el contenido y cambiamos de nuevo el texto.

### Explicación del flujo:

Si el estilo del párrafo es no mostrar el texto entonces mostramos el párrafo y cambiamos el texto del párrafo “open”, si no entonces se oculta el párrafo y se cambia de nuevo el texto.

### Pseudocódigo:

```
Inicio función openHistorial
    Si history == “none” entonces
        History = inline
        Open = "<a href='#\" onclick='\"openHistorial()\">Ocultar las palabras clave usadas
por otros usuarios</a>"
    Si no
        History = none
        Open = "<a href='#\" onclick='\"openHistorial()\">Ver las palabras clave usadas por
otros usuarios</a>"
    Fin si
Fin
```

### Código de la función:

```
function openHistorial() {
    if (document.getElementById("history").style.display == "none"){
        document.getElementById("history").style.display = "inline";
        document.getElementById("open").innerHTML = "<a href='#\" onclick='\"openHistorial()\">Ocultar
las palabras clave usadas por otros usuarios</a>";
    }
    else{
        document.getElementById("history").style.display = "none";
        document.getElementById("open").innerHTML = "<a href='#\" onclick='\"openHistorial()\">Ver las
palabras clave usadas por otros usuarios</a>";
    }
}
```

**Función:** gradoAppear

### Parámetros:

- **idGrupo:** Grupo que se seleccionó

### Variables:

- **arrayhidden:** Arreglo que guarda los grupos que se han seleccionado.
- **Cbarray:** Arreglo que almacena los checkbox de los grupos.
- **Index =** Posición del grupo en arrayhidden

**Tarea:** Lee el grupo que se seleccionó, se revisa si está seleccionado y si el primer grupo no lo está (el grupo “No informativo”), si es así agrega el grupo seleccionado al arreglo y si hay más de un

elemento entonces se muestra el campo extra para el grado de certeza, si no es así decide si quitar todos los campos de certeza o solo el que se deselecciono.

### Explicación del flujo:

Crea una lista de grupos, revisa si se seleccionó un grupo diferente a “No informativo”, si es así agrega el grupo al arreglo y, si hay más de un grupo seleccionado, muestra los campos de texto para el grado de certeza.

Si se deselecciono el grupo o si es el “No informativo”, revisa si el grupo “No informativo” esta seleccionado, si es así se quita la selección de otros grupos y se ocultan los campos de texto del grado de certeza, si no es “No informativo” significa que se deselecciono el grupo, se busca su posición en el arreglo y en base a eso se oculta el campo de texto y se elimina del arreglo, si en el arreglo solo hay un elemento, se ocultan todos los campos de texto.

### Pseudocódigo:

Inicio función setTexti

```
Cbarray = Todos los checkbox
Si "c" + idGrupo está seleccionado y cbarray[0] no está seleccionado entonces
    Arrayhidden += "t" + idGrupo
    Si tamaño del arrayhidden > 1 entonces
        I = 0
        Para i = 0 mientras i < tamaño de arrayhidden hacer
            Arrayhidden[i] = "inline"
        Fin para
    Fin si
Si no
    Index = buscar idGrupo en arrayhidden
    Si index > -1 entonces
        Arrayhidden[index] = "none"
        Eliminar index de arrayhidden
    Fin si
    Si tamaño de arrayhidden == 1 entonces
        Arrayhidden[0] = "none"
    Fin si
Fin si
Fin
```

### Código de la función:

```
function gradoAppear(idGrupo) {
    cbarray = document.getElementsByName("grupo");
    if(document.getElementById("c" + idGrupo).checked && !cbarray[0].checked){
        arrayhidden.push("t" + idGrupo);
        if(arrayhidden.length > 1) {
            var i = 0;
            for (i = 0; i < arrayhidden.length; i++) {
                document.getElementById(arrayhidden[i]).style.display = "inline";
            }
        }
    }
}
```

```

    }
  } else {
    if(cbararray[0].checked){
      arrayhidden = [];
      for (var i = 1; i < cbararray.length; i++){
        cbararray[i].checked = false;
        document.getElementById("t" + cbararray[i].value).style.display = "none";
      }
    } else {
      var index = arrayhidden.indexOf("t" + idGrupo);
      if (index > -1) {
        document.getElementById(arrayhidden[index]).style.display = "none";
        arrayhidden.splice(index, 1);
      }
      if(arrayhidden.length == 1) {
        document.getElementById(arrayhidden[0]).style.display = "none";
      }
    }
  }
}

```

---

**Función:** openHistorialGrupo

**Tarea:** Revisa la etiqueta de estilo del elemento con id “historialGrupo”, si no se muestra entonces lo mostramos y cambiamos el texto de la opción, si no ocultamos el contenido y cambiamos de nuevo el texto.

**Explicación del flujo:**

Si el estilo del párrafo es no mostrar el texto entonces mostramos el párrafo y cambiamos el texto del párrafo “open1”, si no entonces se oculta el párrafo y se cambia de nuevo el texto.

**Pseudocódigo:**

Inicio función openHistorialGrupo

    Si historialGrupo == “none” entonces

        HistorialGrupo = inline

        Open1 = "<a href=\"#\" onclick=\"openHistorialGrupo()\">Ocultar las definiciones que has usado en este grupo</a>"

    Si no

        HistorialGrupo = none

        Open1 = "<a href=\"#\" onclick=\"openHistorialGrupo()\">Ver las definiciones que has usado en este grupo(s) (últimas 10)</a>"

    Fin si

Fin

**Código de la función:**

```

function openHistorialGrupo() {
  if (document.getElementById("historialGrupo").style.display == "none"){
    document.getElementById("historialGrupo").style.display = "inline";
    document.getElementById("open1").innerHTML = "<a href=\"#\" 
onclick=\"openHistorialGrupo()\">Ocultar las definiciones que has usado en este grupo</a>";
  }
}

```

```

    }
    else{
        document.getElementById("historialGrupo").style.display = "none";
        document.getElementById("open1").innerHTML = "<a href=#\"
onclick=\\\"openHistorialGrupo()\\\">Ver las definiciones que has usado en este grupo(s) (últimas 10)</a>\";
    }
}

```

---

**Función:** validarKeywords

**Variables:**

- **Definición:** La definición que se está etiquetando.
- **Aux:** Arreglo de palabras clave que el usuario ha escrito.
- **Flag:** Variable booleana que nos indica si la validación fue correcta.

**Tarea:** Detecta la definición que se está manejando y las palabras clave que el usuario escribió, valida si las palabras clave están en la definición si es así regresa verdadero si no regresa falso.

**Explicación del flujo:**

Lee la definición que se está manejando, crea una lista de palabras clave basado en lo que escribió el etiquetador, mientras las palabras clave estén en la definición y aún queden palabras clave por validar, elimina un posible espacio delante de la palabra clave, si la palabra clave está en la definición entonces flag es verdadero, si no está entonces flag se vuelve falso.

Si se seleccionó un grupo “No informativo” está validación no se realiza, en caso de que las palabras no coincidan se muestra ese mensaje.

**Pseudocódigo:**

Inicio función validarKeywords

Definición = definición\_texto

Aux = lista de palabras clave divididas por ‘,’

Flag = true

I = 0

Mientras flag = true y i < tamaño de aux hacer

    Si aux[i][0] = “ ” hacer

        Aux[i] = aux menos el primer carácter

    Fin si

    Si aux[i] esta en definición hacer

        Flag = true

    Si no

        Flag = false

    Fin si

    I++

Fin mientras

Si key = “No informativo” hacer

    Flag = true

Fin si



```

        Si flag = false hacer
            Mostrar mensaje de error
        Fin si
    Regresa flag
Fin

```

### Código de la función:

```

function validarKeywords() {
    definicion = document.getElementById("definicion_texto").innerHTML.replace(/&nbsp;/g, "
").toLowerCase();
    aux = document.getElementById("key").value.toLowerCase().split(',');
    flag = true
    var i = 0
    while (flag && i < aux.length){
        if (aux[i][0] == ' ')
            aux[i] = aux[i].substr(1);
        if (definicion.indexOf(aux[i]) != -1)
            flag = true;
        else
            flag = false;
        i += 1;
    }
    if (document.getElementById("key").value == "No informativo")
        flag = true;
    if(!flag)
        alert("La palabras clave no se encuentran en la definición");
    return flag;
}

```

---

### Función: openPorcentaje

#### Variables:

- **Cbarray:** Arreglo que contiene todos los párrafos que muestran el porcentaje de uso.

**Tarea:** Lee todos los párrafos que tienen información sobre el porcentaje de uso y si están ocultos los muestran, en caso contrario los oculta.

#### Explicación del flujo:

Lee la lista de párrafos que tienen información sobre el porcentaje de grupos, se hace una iteración para ese arreglo si está oculto lo muestra y cambia el texto de la opción, si no entonces lo oculta y cambia el texto.

#### Pseudocódigo:

```

Inicio función openPorcentaje
    Cbarray = porgrupos
    Para i = 0 mientras i < tamaño de cbarray hacer
        Si cbarray[i] = "none" entonces
            Cbarray[i] = "inline"

```

```

                                Open3 = "<a href=\"#\" onclick=\"openPorcentaje()\">Ocultar los
porcentajes</a>"
                                Si no
                                    Cbarray[i] = "none"
                                    Open3 = "<a href=\"#\" onclick=\"openPorcentaje()\">Mostrar el
porcentaje de uso de cada grupo (recuerda siempre usar tu propio criterio)</a>"
                                Fin si
                            Fin para
Fin

```

#### Código de la función:

```

function openPorcentaje() {
    cbarray = document.getElementsByName("porgrupos");
    for(var i = 0; i < cbarray.length; i++){
        if (cbarray[i].style.display == "none"){
            cbarray[i].style.display = "inline";
            document.getElementById("open3").innerHTML = "<a href=\"#\"
onclick=\"openPorcentaje()\">Ocultar los porcentajes</a>";
        }
        else{
            cbarray[i].style.display = "none";
            document.getElementById("open3").innerHTML = "<a href=\"#\"
onclick=\"openPorcentaje()\">Mostrar el porcentaje de uso de cada grupo (recuerda siempre usar tu propio
criterio)</a>";
        }
    }
}

```

---

**Función:** validateAlpha

#### Variables:

- **textInput:** Texto que el usuario escribió en el campo de texto de grado de certeza.

**Tarea:** Valida que solo se escriban números en el grado de certeza.

#### Explicación del flujo:

Lee el texto que hay en el grado de certeza, quita los que no sean números y los escribe en el campo de grado de certeza.

#### Pseudocódigo:

```

Inicio función openPorcentaje
    textInput = "g" + idGrupo
    textInput = quitar lo que no sea numero
    "g" + idGrupo = textInput
Fin

```

#### Código de la función:

```
function validateAlpha(idGrupo){
    var textInput = document.getElementById("g" + idGrupo).value;
    textInput = textInput.replace(/[^0-9]/g, "");
    document.getElementById("g" + idGrupo).value = textInput;
}
```

---

**Función:** validarGrado

**Variables:**

- **cbararray:** Arreglo de los grupos.
- **Grado:** Suma de los grados de certeza.
- **Flag:** Nos indica si la validación fue correcta.

**Tarea:** Valida que la suma de los grados de certeza sea 100.

**Explicación del flujo:**

Hace una lista de los grados de certeza, si hay más de un grupo seleccionado se hace una iteración y si el grupo está seleccionado se añade el valor de grado de certeza a la suma, si solo hay un grupo seleccionado o si la suma es 100 flag se vuelve verdadero, si no se muestra un mensaje de error.

**Pseudocódigo:**

Inicio función validarGrado

    Cbararray: Lista de grupos

    Grado = 0

    Flag = false

    Si hay más de un grupo seleccionado hacer

        Para i = 0 mientras i < tamaño de cbararray hacer

            Si cbararray[i] está seleccionado entonces

                Grado += "g" + cbararray[i]

            Fin si

        Fin para

    Fin si

    Si hay un grupo seleccionado o grado = 100 entonces

        Flag = true

    Si no

        Mostrar mensaje de error

    Fin si

    Regresa flag

Fin

**Código de la función:**

```
function validarGrado(){
    cbararray = document.getElementsByName("grupo");
    grado = 0;
    flag = false;
    if($(".checkbox[name=grupo]:checked").length > 1) {
        for (var i = 0; i < cbararray.length; i++){
```

```

        if(cbararray[i].checked)
            grado += parseInt(document.getElementById('g' + cbararray[i].value).value);
    }
}
if($(".checkbox[name=grupo]:checked").length == 1 || grado == 100)
    flag = true;
else
    alert("Error: Asegurate de seleccionar al menos un grupo y que el grado de certeza sume 100");
return flag;
}

```

---

**Función:** subir

**Tarea:** Valida el grado de certeza y las palabras clave y sube el formulario al servidor.

**Explicación del flujo:**

Revisa los grados de certeza y las palabras clave, si son correctas envía el documento al servidor.

**Pseudocódigo:**

Inicio función subir

    Si validarGrado y validarKeywords entonces

        Subir el formulario

    Fin si

Fin

**Código de la función:**

```

function subir(){
    if(validarGrado() && validarKeywords())
        document.getElementById("relacion_form").submit();
}

```

---

## Vistas – Modo General

En el modo general se utilizan 5 vistas diferentes, 4 de ellas tienen solo una función específica y poco o nada de código JavaScript, la otra vista es la principal y en ella se mostrará distintos tipos de información al etiquetador, también es la que tiene una gran cantidad de JavaScript que permite, entre otras cosas, la validación de campos, asignación de valores y mostrar distintas opciones.

Vista: Inicio

Modulo relacionado: [Inicio](#)

# Bienvenido a Define, ssss

Selecciona un término:

- ☐ Animal
- ☐ Machine

Ir

[Registrar un nuevo término](#)

**Descripción:** Muestra un mensaje de bienvenida, si hay términos registrados muestra una lista con todos los términos y un enlace para crear más términos, si no se ha registrado ninguno solo muestra el enlace, los términos tienen un checkbox.

**Funcionalidad dinámica:**

- **Mostrar el nombre de usuario:**  
`<h1>Bienvenido a Define, {{=usuario_nombre}}</h1>`
- **Mostrar los términos o un mensaje pidiendo que se registre un término.**  

```

{{if len(termino) == 2:}}
<p> Primero necesitas <a href="importar">registrar más de un término</a></p>
{{else:}}
<p>Selecciona un término:</p>
<p>

```
- **Mostrar los términos registrados:**  

```

{{for term in termino:
  if not term['ter'] in 'Nulo':}}
  <input value="{{=term['id']}}" name = "termino" type="checkbox"
/>{{=term['ter']}}<br>
  {{pass}}
{{pass}}

```

Vista: Importar

Modulo relacionado: [Importar](#)

**Descripción:** Es un formulario en el cual se selecciona un archivo, se nombra el término y se agrega a la base de datos.

Vista: [crearGrupo](#)

Modulo relacionado: [Crear grupos](#)

**Descripción:** La interfaz solo nos pide el nombre del grupo personalizado que queremos crear, adicionalmente hace la verificación de si existe el grupo o si no se llenó el campo de nombre.

**Funcionalidad dinámica:**

- Obtener la cadena de los grupos existentes y del término al que está relacionado para asignarlos a un campo hidden:

```
<input type="hidden" id="cadena_grupos" value="{{=cadena_grupos}}" />
<input type="hidden" value="{{=termino}}" name="termino" />
```

**Funciones JavaScript:**

---

**Función:** Addgroup

**Tarea:** Hace una validación de datos y si todo está correcto manda el formulario al servidor.

**Explicación del flujo:**

Comprueba que se haya introducido el nombre del grupo, si es así entonces comprueba que el valor introducido no este registrado, en caso afirmativo entonces se manda el formulario al servidor, se refresca la ventana padre y se cierra está ventana.

#### **Pseudocódigo:**

Inicio función AddGroup

```
Si nombre_id != "" entonces
    Si nombre_id no está en cadena_grupos entonces
        Submit
        Refrescar padre
        Cerrar ventana
    Si no
        Mensaje "El grupo ya existe"
    Fin si
Si no
    Mensaje "Ingrese el nombre del grupo"
Fin si
```

Fin

#### **Código de la función:**

```
function AddGroup(){
    if(document.getElementById("nombre_id").value != "")

if(document.getElementById("cadena_grupos").value.indexOf(document.getElementById("nombre_id").value) == -1){
        document.getElementById("GroupDB").submit();
        window.opener.location.reload();
        window.close();
    } else
        alert("El grupo ya existe");
    else
        alert("Ingresa el nombre del grupo");
}
```

---

Vista: [vistaWordNet](#)

Modulo relacionado: [Crear grupos](#)

## Grupos sugeridos para ['Animal', 'Machine']

☐ animal

**Lemmas:** ['animal', 'animate\_being', 'beast', 'brute', 'creature', 'fauna'],

**Definition:** (a living organism characterized by voluntary movement),

**Examples:** No available

**Lemmas:** ['animal', 'carnal', 'fleshly', 'sensual'],

**Definition:** (marked by the appetites and passions of the body),

**Examples:** ['animal instincts', 'carnal knowledge', 'fleshly desire', 'a sensual delight in eating', 'music is the only sensual pleasure without vice']

☐ beast

**Descripción:** Muestra los grupos sugeridos por WordNet para un término, cada sugerencia tiene un checkbox para que se pueda seleccionar, además de datos sobre esa sugerencia, se puede crear los grupos seleccionado o crear uno personalizado.

### Funcionalidad dinámica:

- **Mostrar el término para el cual se sugieren los grupos:**  
<h2>Grupos sugeridos para {{=request.vars.texto}}</h2>
- **Mostrar los grupos propuestos por WordNet, eliminando algunas cadenas de apoyo:**

```
{{
  i = 0
  while (i < len(wordnet)):
    if ('@' in wordnet[i]):
      if(wordnet[i+1] != '-----'):}}
      <p>{{=XML(wordnet[i].replace('@', ''))}}</p>
      {{pass}}
    {{else:
      if(wordnet[i] != '-----'):}}
      <p>{{=XML(wordnet[i])}}</p>
      {{pass}}
    {{pass}}
    {{i += 1}}
  {{pass}}
  {{if(len(wordnet) == 0):}}
  <p>
    No hay más sugerencias
```

### Funciones JavaScript:

---

**Función:** Addgroup

**Tarea:** Manda el formulario al servidor.

**Explicación del flujo:**



Se manda el formulario al servidor, se refresca la ventana padre y se cierra esta ventana.

#### Pseudocódigo:

```
Inicio función AddGroup
    Submit
    Refrescar padre
    Cerrar ventana
Fin
```

#### Código de la función:

```
function AddGroup(){
    document.getElementById("GroupDB").submit();
    window.opener.location.reload();
    window.close();
}
```

Vista: relación

Modulo relacionado: [relacion](#)

Inicio

## Término: Animal

**Definición:** An animal is defined as 'at large' unless it is confined to a property by leash and tether (dogs only), fence, enclosure or under the control of a person. We also...

**Seleccione el grupo semántico al que pertenece:**

☐ Animal

☐ automobile

**Opciones:**

[Crear grupos](#)

[Mostrar el porcentaje de uso de cada grupo \(recuerda siempre usar tu propio criterio\)](#)

[Ver las palabras clave usadas por otros usuarios](#)

[Ver las definiciones que has usado en este grupo\(s\) \(últimas 10\)](#)

Palabras clave:

**Descripción:** Esta vista contiene la definición que se etiquetará, los grupos existentes, la opción de desplegar distintos apoyos y el campo para las palabras clave, además de que valida que todo esté en orden y cuenta con diversas funciones JavaScript para poder desplegar los apoyos.

#### Funcionalidad dinámica:

- **Mostrar el término al que pertenece la definición:**  
`<h2>Término: {{=termino['ter']}}</h2>`
- **Mostrar la definición con la que se está trabajando:**  
`<strong>Definición: </strong>{{=definicion['defi']}}`
- **Agregar valores a campos hidden para que se manden junto con el formulario al servidor:**  
`<input value="{{=definicion['id']}}" type="hidden" name="definicion" />`

- ```

<input value="{{=termino['id']}}" type="hidden" name="termi" />
<input value="{{=terminos}}" type="hidden" name="auxiliar" />
<input value="" type="hidden" id="grupos_hidden" />

```
- Listar los grupos existentes, agregando un checkbox para seleccionar uno o más grupos, asignando los eventos y añadiendo otros campos que se utilizarán para los grupos:

```

{{for grupos_lista in grupos_merge:
    for grupo in grupos_lista:
        if grupo['grup'] == 'No informativo' and grupo['termino_id'] == termino['id']:}}
        <p><input value="{{=grupo['id']}}" id="c{{=grupo['id']}}" type="checkbox"
name="grupo" onclick="setHistorial('{{=grupo['id']}}',
'{{=grupo['grup']}}');gradoAppear('{{=grupo['id']}}');setTexto('{{=grupo['grup']}}');"
/>{{=grupo['grup']}} <p id="t{{=grupo['id']}}" style="display:none">Grado de certeza<input
type="text" id="g{{=grupo['id']}}" name="grado"
onKeyUp="validateAlpha('{{=grupo['id']}}')"/></p><p id="pgrupos" name="porgrupos"
style="display:none">{{if grupo['id'] in porcentaje.keys():}}Porcentaje de uso por otros
usuarios: {{=(float)((porcentaje[grupo['id']]*100) / (total_usuarios))}}{{else:}} No se ha
utilizado{{pass}}</p></p>
        {{pass}}
        {{if grupo['grup'] != 'No informativo':}}
        <p><input value="{{=grupo['id']}}" id="c{{=grupo['id']}}" type="checkbox"
name="grupo" onclick="setHistorial('{{=grupo['id']}}',
'{{=grupo['grup']}}');gradoAppear('{{=grupo['id']}}');setTexto('{{=grupo['grup']}}');"
/>{{=grupo['grup']}} <p id="t{{=grupo['id']}}" style="display:none">Grado de certeza<input
type="text" id="g{{=grupo['id']}}" name="grado"
onKeyUp="validateAlpha('{{=grupo['id']}}')"/></p><p id="pgrupos" name="porgrupos"
style="display:none">{{if grupo['id'] in porcentaje.keys():}}Porcentaje de uso por otros
usuarios: {{=(float)((porcentaje[grupo['id']]*100) / (total_usuarios))}}{{else:}} No se ha
utilizado{{pass}}</p></p>
        {{pass}}
        {{pass}}
        {{pass}}

```
  - Escribir el historial de palabras clave utilizadas:

```

{{if len(historial) > 0:}}
    Palabras clave usadas por otros usuarios:<br>
    {{for historia in historial:}}
        {{=historia}}<br>
    {{pass}}
{{else:}}
    Aún nadie clasifica esta definición.
{{pass}}

```

## Funciones JavaScript:

---

**Función:** openWordnet

**Variables:**

- **Wn:** La ventana emergente

**Tarea:** Abre una ventana emergente para que se puedan crear los grupos.

**Explicación del flujo:**

Abre una ventana emergente con los datos recabados por el modulo.

### Pseudocódigo:

Inicio función AddGroup  
    Wn = abrir ventana  
Fin

### Código de la función:

```
function openWordnet() {  
    var wn = window.open("/DEFINE/general/vistaWordNet?id={{=termino['id']}}&{{=terminos_texto}}",  
    "_black", "width=800, height=400, scrollbars=1");  
}
```

---

### Función: setHistorial

#### Parámetros:

- **idGrupo:** El id del grupo que se acaba de seleccionar
- **nombreGrupo:** El nombre del grupo que se acaba de seleccionar

#### Variables:

- **grupos\_historial:** Cadena de texto que contiene todos los grupos seleccionados en el formato "grupos=" para poder pasarlo como parámetro a la función de Ajax.

**Tarea:** Revisa cuando se selecciona o deselecciona un checkbox, crea una cadena de texto basado en eso y ejecuta una función Ajax que nos indicará el historial de definiciones usadas en un grupo o grupos.

### Explicación del flujo:

Primero revisa si se seleccionó un grupo "No informativo", si es así se limpia el historial de grupos, después se revisa si el campo está seleccionado, si es así se añade al historial, si no se elimina. Se asigna el valor del historial de grupos seleccionados a un campo hidden y se ejecuta una función Ajax que devolverá el historial y lo mostrará en el lugar indicado por la función Ajax.

### Pseudocódigo:

Inicio función setHistorial  
    Si nombreGrupo == "No informativo" entonces  
        Grupos\_historial = ""  
    Fin si  
    Si Se seleccionó el grupo entonces  
        Grupos\_historial += "grupos=" + idGrupo + "&"  
    Si no  
        Grupos\_historial = borrar el idGrupo de la cadena  
    Fin si  
    Grupos\_hidden = grupos\_historial  
    Función de Ajax  
Fin

### Código de la función:

```
function setHistorial(idGrupo, nombreGrupo) {  
    if(nombreGrupo == "No informativo")  
        grupos_historial = "";  
    if(document.getElementById("c" + idGrupo).checked)  
        grupos_historial += "grupos=" + idGrupo + "&";  
    else  
        grupos_historial = grupos_historial.replace("grupos=" + idGrupo + "&", "");  
    document.getElementById("grupos_hidden").value = grupos_historial + "c={ {=auth.user.id} }";  
    ajax({ {=URL(r=request,f='getHistorialGrupos')}}?' +  
document.getElementById("grupos_hidden").value,['grupos_hidden'],'historialGrupo');  
}
```

---

**Función:** setTexto

### Parámetros:

- **tipo:** Contiene el valor del grupo, si es de tipo normal o no informativo

### Variables:

- **anterior:** Valor actual del campo de texto en el que se pondrán las keywords.

**Tarea:** Lee el grupo que se seleccionó y si es del tipo “No informativo”, bloquea los demás grupos para que no puedan ser seleccionados y cambia el valor de las palabras clave a “No informativo”, si se deselecciona el grupo no informativo se vuelven a habilitar los grupos.

### Explicación del flujo:

Si el valor en el campo de texto de las palabras clave es diferente a “Ni informativo” se guarda su valor, si se seleccionó el grupo “No informativo” se cambia el valor del campo de texto y se deshabilitan los checkbox de los grupos, se cuenta cuantos hay seleccionados y si no hay ninguno se vuelven a habilitar los grupos.

### Pseudocódigo:

Inicio función setTexto

    Anterior = ""

    Si key != “No informativo”

        Anterior = key

    Fin si

    Si tipo == “No informativo”

        Key = “No informativo”

        Deshabilitar los grupos

        checkCount = todos los seleccionados

        Si checkCount == 0

            Habilitar los grupos

        Fin Si

    Si no

        Key = anterior

    Fin si

Fin

#### Código de la función:

```
function setTexto(tipo) {  
    anterior = "";  
    if(document.getElementById("key").value != "No informativo")  
        anterior = document.getElementById("key").value;  
    if (tipo == "No informativo"){  
        document.getElementById("key").value = "No informativo";  
        $(':checkbox[name=grupo]').not(':checked').attr('disabled', true);  
        checkCount = $(':checked').length;  
        if (checkCount == 0)  
            $(':checkbox[name=grupo]:disabled').attr('disabled', false);  
        //document.getElementById("key").setAttribute("disabled", true);  
    } else {  
        document.getElementById("key").value = anterior;  
        //document.getElementById("key").setAttribute("disabled", false);  
    }  
}
```

---

#### Función: openHistorial

**Tarea:** Revisa la etiqueta de estilo del elemento con id “history”, si no se muestra entonces lo mostramos y cambiamos el texto de la opción, si no ocultamos el contenido y cambiamos de nuevo el texto.

#### Explicación del flujo:

Si el estilo del párrafo es no mostrar el texto entonces mostramos el párrafo y cambiamos el texto del párrafo “open”, si no entonces se oculta el párrafo y se cambia de nuevo el texto.

#### Pseudocódigo:

Inicio función openHistorial

    Si history == “none” entonces

        History = inline

        Open = "<a href=\"#\" onclick=\"openHistorial()\">Ocultar las palabras clave usadas por otros usuarios</a>"

    Si no

        History = none

        Open = "<a href=\"#\" onclick=\"openHistorial()\">Ver las palabras clave usadas por otros usuarios</a>"

    Fin si

Fin

#### Código de la función:

```
function openHistorial() {  
    if (document.getElementById("history").style.display == "none"){  
        document.getElementById("history").style.display = "inline";  
    }
```

```

        document.getElementById("open").innerHTML = "<a href='#\" onclick='\"openHistorial()\">Ocultar
las palabras clave usadas por otros usuarios</a>";
    }
    else{
        document.getElementById("history").style.display = "none";
        document.getElementById("open").innerHTML = "<a href='#\" onclick='\"openHistorial()\">Ver las
palabras clave usadas por otros usuarios</a>";
    }
}

```

---

**Función:** gradoAppear

**Parámetros:**

- **idGrupo:** Grupo que se seleccionó

**Variables:**

- **arrayhidden:** Arreglo que guarda los grupos que se han seleccionado.
- **Cbarray:** Arreglo que almacena los checkbox de los grupos.
- **Index =** Posición del grupo en arrayhidden

**Tarea:** Lee el grupo que se seleccionó, se revisa si está seleccionado y si el primer grupo no lo está (el grupo “No informativo”), si es así agrega el grupo seleccionado al arreglo y si hay más de un elemento entonces se muestra el campo extra para el grado de certeza, si no es así decide si quitar todos los campos de certeza o solo el que se deselecciono.

**Explicación del flujo:**

Crea una lista de grupos, revisa si se seleccionó un grupo diferente a “No informativo”, si es así agrega el grupo al arreglo y, si hay más de un grupo seleccionado, muestra los campos de texto para el grado de certeza.

Si se deselecciono el grupo o si es el “No informativo”, revisa si el grupo “No informativo” esta seleccionado, si es así se quita la selección de otros grupos y se ocultan los campos de texto del grado de certeza, si no es “No informativo” significa que se deselecciono el grupo, se busca su posición en el arreglo y en base a eso se oculta el campo de texto y se elimina del arreglo, si en el arreglo solo hay un elemento, se ocultan todos los campos de texto.

**Pseudocódigo:**

Inicio función setTexti

    Cbarray = Todos los checkbox

    Si “c” + idGrupo está seleccionado y cbarray[0] no está seleccionado entonces

        Arrayhidden += “t” + idGrupo

        Si tamaño del arrayhidden > 1 entonces

            I = 0

            Para i = 0 mientras i < tamaño de arrayhidden hacer

                Arrayhidden[i] = “inline”

            Fin para

        Fin si

```

Si no
    Index = buscar idGrupo en arrayhidden
    Si index > -1 entonces
        Arrayhidden[index] = "none"
        Eliminar index de arrayhidden
    Fin si
    Si tamaño de arrayhidden == 1 entonces
        Arrayhidden[0] = "none"
    Fin si
Fin si
Fin

```

#### Código de la función:

```

function gradoAppear(idGrupo) {
    cbararray = document.getElementsByName("grupo");
    if(document.getElementById("c" + idGrupo).checked && !cbararray[0].checked){
        arrayhidden.push("t" + idGrupo);
        if(arrayhidden.length > 1) {
            var i = 0;
            for (i = 0; i < arrayhidden.length; i++) {
                document.getElementById(arrayhidden[i]).style.display = "inline";
            }
        }
    } else {
        if(cbararray[0].checked){
            arrayhidden = [];
            for (var i = 1; i < cbararray.length; i++){
                cbararray[i].checked = false;
                document.getElementById("t" + cbararray[i].value).style.display = "none";
            }
        } else {
            var index = arrayhidden.indexOf("t" + idGrupo);
            if (index > -1) {
                document.getElementById(arrayhidden[index]).style.display = "none";
                arrayhidden.splice(index, 1);
            }
            if(arrayhidden.length == 1) {
                document.getElementById(arrayhidden[0]).style.display = "none";
            }
        }
    }
}

```

---

**Función:** openHistorialGrupo

**Tarea:** Revisa la etiqueta de estilo del elemento con id “historialGrupo”, si no se muestra entonces lo mostramos y cambiamos el texto de la opción, si no ocultamos el contenido y cambiamos de nuevo el texto.

**Explicación del flujo:**

Si el estilo del párrafo es no mostrar el texto entonces mostramos el párrafo y cambiamos el texto del párrafo "open1", si no entonces se oculta el párrafo y se cambia de nuevo el texto.

#### Pseudocódigo:

```
Inicio función openHistorialGrupo
    Si historialGrupo == "none" entonces
        HistorialGrupo = inline
        Open1 = "<a href=\"#\" onclick=\"openHistorialGrupo()\">Ocultar las definiciones
que has usado en este grupo</a>"
    Si no
        HistorialGrupo = none
        Open1 = "<a href=\"#\" onclick=\"openHistorialGrupo()\">Ver las definiciones que
has usado en este grupo(s) (últimas 10)</a>"
    Fin si
Fin
```

#### Código de la función:

```
function openHistorialGrupo() {
    if (document.getElementById("historialGrupo").style.display == "none"){
        document.getElementById("historialGrupo").style.display = "inline";
        document.getElementById("open1").innerHTML = "<a href=\"#\"
onclick=\"openHistorialGrupo()\">Ocultar las definiciones que has usado en este grupo</a>";
    }
    else{
        document.getElementById("historialGrupo").style.display = "none";
        document.getElementById("open1").innerHTML = "<a href=\"#\"
onclick=\"openHistorialGrupo()\">Ver las definiciones que has usado en este grupo(s) (últimas 10)</a>";
    }
}
```

---

#### Función: validarKeywords

##### Variables:

- **Definición:** La definición que se está etiquetando.
- **Aux:** Arreglo de palabras clave que el usuario ha escrito.
- **Flag:** Variable booleana que nos indica si la validación fue correcta.

**Tarea:** Detecta la definición que se está manejando y las palabras clave que el usuario escribió, valida si las palabras clave están en la definición si es así regresa verdadero si no regresa falso.

##### Explicación del flujo:

Lee la definición que se está manejando, crea una lista de palabras clave basado en lo que escribió el etiquetador, mientras las palabras clave estén en la definición y aún queden palabras clave por validar, elimina un posible espacio delante de la palabra clave, si la palabra clave está en la definición entonces flag es verdadero, si no está entonces flag se vuelve falso.



Si se seleccionó un grupo “No informativo” está validación no se realiza, en caso de que las palabras no coincidan se muestra ese mensaje.

### Pseudocódigo:

Inicio función validarKeywords

```
Definición = definición_texto
Aux = lista de palabras clave divididas por ','
Flag = true
I = 0
Mientras flag = true y i < tamaño de aux hacer
    Si aux[i][0] = " " hacer
        Aux[i] = aux menos el primer carácter
    Fin si
    Si aux[i] esta en definición hacer
        Flag = true
    Si no
        Flag = false
    Fin si
    I++
Fin mientras
Si key = “No informativo” hacer
    Flag = true
Fin si
Si flag = false hacer
    Mostrar mensaje de error
Fin si
Regresa flag
```

Fin

### Código de la función:

```
function validarKeywords() {
    definicion = document.getElementById("definicion_texto").innerHTML.replace(/&nbsp;/g, " ");
    aux = document.getElementById("key").value.toLowerCase().split(',');
    flag = true;
    var i = 0;
    while (flag && i < aux.length){
        if (aux[i][0] == ' ')
            aux[i] = aux[i].substr(1);
        if (definicion.indexOf(aux[i]) != -1)
            flag = true;
        else
            flag = false;
        i += 1;
    }
    if (document.getElementById("key").value == "No informativo")
        flag = true;
    if(!flag)
        alert("La palabras clave no se encuentran en la definición");
}
```

```
    return flag;
}
```

---

**Función:** openPorcentaje

**Variables:**

- **Cbarray:** Arreglo que contiene todos los párrafos que muestran el porcentaje de uso.

**Tarea:** Lee todos los párrafos que tienen información sobre el porcentaje de uso y si están ocultos los muestran, en caso contrario los oculta.

**Explicación del flujo:**

Lee la lista de párrafos que tienen información sobre el porcentaje de grupos, se hace una iteración para ese arreglo si está oculto lo muestra y cambia el texto de la opción, si no entonces lo oculta y cambia el texto.

**Pseudocódigo:**

Inicio función openPorcentaje

    Cbarray = porgrupos

    Para i = 0 mientras i < tamaño de cbarray hacer

        Si cbarray[i] = "none" entonces

            Cbarray[i] = "inline"

            Open3 = "<a href='#\" onclick='\"openPorcentaje()\">Ocultar los porcentajes</a>"

        Si no

            Cbarray[i] = "none"

            Open3 = "<a href='#\" onclick='\"openPorcentaje()\">Mostrar el porcentaje de uso de cada grupo (recuerda siempre usar tu propio criterio)</a>"

        Fin si

    Fin para

Fin

**Código de la función:**

```
function openPorcentaje() {
    cbarray = document.getElementsByName("porgrupos");
    for(var i = 0; i < cbarray.length; i++){
        if (cbarray[i].style.display == "none"){
            cbarray[i].style.display = "inline";
            document.getElementById("open3").innerHTML = "<a href='#\"
onclick='\"openPorcentaje()\">Ocultar los porcentajes</a>";
        }
        else{
            cbarray[i].style.display = "none";
            document.getElementById("open3").innerHTML = "<a href='#\"
onclick='\"openPorcentaje()\">Mostrar el porcentaje de uso de cada grupo (recuerda siempre usar tu propio
criterio)</a>";
        }
    }
}
```

}

---

**Función:** validateAlpha

**Variables:**

- **textInput:** Texto que el usuario escribió en el campo de texto de grado de certeza.

**Tarea:** Valida que solo se escriban números en el grado de certeza.

**Explicación del flujo:**

Lee el texto que hay en el grado de certeza, quita los que no sean números y los escribe en el campo de grado de certeza.

**Pseudocódigo:**

```
Inicio función openPorcentaje
    textInput = "g" + idGrupo
    textInput = quitar lo que no sea numero
    "g" + idGrupo = textInput
Fin
```

**Código de la función:**

```
function validateAlpha(idGrupo){
    var textInput = document.getElementById("g" + idGrupo).value;
    textInput = textInput.replace(/[^0-9]/g, "");
    document.getElementById("g" + idGrupo).value = textInput;
}
```

---

**Función:** validarGrado

**Variables:**

- **cbarray:** Arreglo de los grupos.
- **Grado:** Suma de los grados de certeza.
- **Flag:** Nos indica si la validación fue correcta.

**Tarea:** Valida que la suma de los grados de certeza sea 100.

**Explicación del flujo:**

Hace una lista de los grados de certeza, si hay más de un grupo seleccionado se hace una iteración y si el grupo está seleccionado se añade el valor de grado de certeza a la suma, si solo hay un grupo seleccionado o si la suma es 100 flag se vuelve verdadero, si no se muestra un mensaje de error.

**Pseudocódigo:**

```
Inicio función validarGrado
    Cbarray: Lista de grupos
```

```

Grado = 0
Flag = false
Si hay más de un grupo seleccionado hacer
    Para i = 0 mientras i < tamaño de cbararray hacer
        Si cbararray[i] está seleccionado entonces
            Grado += "g" + cbararray[i]
        Fin si
    Fin para
Fin si
Si hay un grupo seleccionado o grado = 100 entonces
    Flag = true
Si no
    Mostrar mensaje de error
Fin si
Regresa flag
Fin

```

#### Código de la función:

```

function validarGrado(){
    cbararray = document.getElementsByName("grupo");
    grado = 0;
    flag = false;
    if($(".checkbox[name=grupo]:checked").length > 1) {
        for (var i = 0; i < cbararray.length; i++){
            if(cbararray[i].checked)
                grado += parseInt(document.getElementById('g' + cbararray[i].value).value);
        }
    }
    if($(".checkbox[name=grupo]:checked").length == 1 || grado == 100)
        flag = true;
    else
        alert("Error: Asegurate de seleccionar al menos un grupo y que el grado de certeza sume 100");
    return flag;
}

```

---

**Función:** subir

**Tarea:** Valida el grado de certeza y las palabras clave y sube el formulario al servidor.

#### Explicación del flujo:

Revisa los grados de certeza y las palabras clave, si son correctas envía el documento al servidor.

#### Pseudocódigo:

```

Inicio función subir
    Si validarGrado y validarKeywords entonces
        Subir el formulario
    Fin si
Fin

```

#### Código de la función:

```
function subir(){  
    if(validarGrado() && validarKeywords())  
        document.getElementById("relacion_form").submit();  
}
```

---

#### Vistas- Interfaz administrativa

Vista: Inicio

Modulo relacionado: [Inicio](#)

Inicio

[Genera el reporte del modo particular](#)

[Genera el reporte del modo general](#)

**Descripción:** Muestra los enlaces para poder generar los reportes

#### Funcionalidad dinámica:

- **Validar si el usuario es administrador:**  
{{if usuario\_mail != 'define.gil@gmail.com':}}  
<h2>  
 No tienes permiso para ver esta parte del sistema  
</h2>  
{{else:}}
-

## Instalación

El etiquetador no tiene que instalar el sistema ya que este funcionará desde la web, solo necesitará un navegador de internet.

Sin embargo, para que el etiquetador pueda trabajar con el sistema es necesario que se instale en algún servidor y que se agregue el sistema en Web2Py.

Para ello es necesario tener las siguientes herramientas instaladas en el servidor (en este orden)-

- 1- **Python 2.7:** Es la versión de Python compatible con el framework de Web2Py, para la instalación se puede consultar la documentación oficial:

<https://www.python.org/doc/>

Hay que destacar que las versiones más actuales de algunos sistemas operativos basados en Linux tienen instalada la versión de Python que necesitamos, para ver eso hay que ejecutar `python --version`, si es la 2.7 entonces podemos pasar a la siguiente herramienta.

- 2- **Natural Language Toolkit:** Gracias a esta herramienta podemos tener acceso al manejo del corpus de WordNet, su instalación puede ser encontrada aquí:

<http://www.nltk.org/install.html>

- 3- **Web2Py para desarrolladores:** Este es el framework, se debe de utilizar la versión de desarrolladores ya que es la que nos permite utilizar bibliotecas instaladas en nuestra versión de Python, se puede obtener:

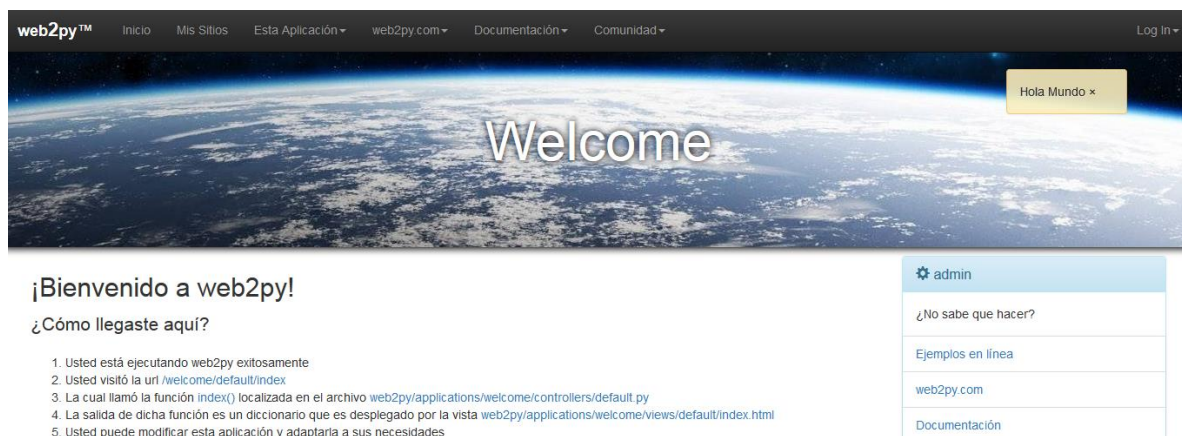
<http://www.web2py.com/init/default/download>

Esta herramienta no se instala, solo se descomprime y se ejecuta.

Si ya se tiene las herramientas en el sistema operativo entonces es momento de ejecutar el framework, para ello nos situamos en el directorio donde descomprimos el framework y ejecutamos:

```
python web2py.py -a [Contraseña que se asignará a Web2Py] -i [URL que se abrirá] -p 8000
```

Con eso ya debe de estar funcionando el framework, se abrirá el navegador con la URL especificada:



Damos clic en “admin”:

web2py™ interfaz administrativa

# web2py™ Web Framework

## Inicio de sesión para la Interfaz Administrativa

Contraseña del Administrador:

Escribimos la contraseña que se asignó al momento de ejecutar el framework y podremos ver la interfaz administrativa

web2py™ interfaz administrativa

sitio ayuda fin de sesión Depurar

[cambie contraseña admin](#) [Recargar rutas](#)

### Aplicaciones instaladas

|                                  |                                                        |
|----------------------------------|--------------------------------------------------------|
| admin (actualmente en ejecución) | <a href="#">Gestionar</a>                              |
| DEFINE                           | <a href="#">Gestionar</a> <a href="#">Deshabilitar</a> |
| Define3                          | <a href="#">Gestionar</a> <a href="#">Deshabilitar</a> |
| examples                         | <a href="#">Gestionar</a> <a href="#">Deshabilitar</a> |
| welcome                          | <a href="#">Gestionar</a> <a href="#">Deshabilitar</a> |

#### Versión

2.11.2-stable+timestamp.2015.05.30.11.29.46  
(Ejecutando en Rocket 1.2.6, Python 2.7.7)

[upgrade now to Version 2.12.3](#)

[Pruebe la interfaz móvil](#)

#### Nueva aplicación

Nombre de la aplicación:

#### Suba e instale una aplicación empaquetada

Nombre de la aplicación:

Subir un paquete:

En la parte de subir un paquete seleccionamos nuestra aplicación .w2p, y con eso debe de estar funcionando nuestra aplicación en la siguiente url: `http://[URL del servidor]:8000/DEFINE`

## Futuras mejoras

- Implementar un sistema de validación para verificar que el etiquetador está haciendo su trabajo de manera correcta.
- Agregar estadísticas y más módulos a la interfaz administrativa.
- Reducir el código al unir los módulos con el mismo funcionamiento entre los diferentes modos.