Departamento de Informática
Universidad Técnica Federico Santa María

Requisitos de Software

Proyecto: BIA

Integrantes:

Nombres y Apellidos	Email	ROL USM
Rolando Javier Casanueva Quezada	rolando.casanueva.12@sansano.usm.cl	201204505-8
Ricardo Francisco Carrasco Aguirre	ricardo.carrasco.12@sansano.usm.cl	201204510-k

Contenido del Entregable 2

Desarrollo del Prototipo

Para este entregable si bien se espera un prototipo funcional, la utilidad de la página y la manera como se ha planificado la evolución de nuestra aplicación, ha llevado a un desarrollo parcial. Nuestra planificación está funcionando correctamente y se han generado los avances esperados para la fecha.

La aplicación se ha desarrollado en Django mediante la integración de JavaScript para ciertas utilidades, es aquí donde nuestro sistema ya se encuentra moldeado. Existen vistas preliminares de nuestro sistema, utilizando un template, tanto para la administración como para el buscador, sin embargo sus funcionalidades las mantenemos relegadas para la tercera etapa.

Como presentación del prototipo de la aplicación para esta entrega, ya se encuentran dispuestos los modelos de la aplicación y sus vistas. Las cuales pueden ser vistas en Anexos.

En términos de avances también hemos logrado realizar ciertas pruebas en lo que son redes neuronales para lo cual hemos hecho un estudio de lo que es el funcionamiento de redes feed forward.

Esto puede verse en un estudio que realizamos sobre la red FF en la carpeta anexos del repositorio. Documentos\Anexos

```
neuron = dict()
neuron['weights'] = [value1, value2, value3, ..., valueN]

layer = list()
...
layer = [neuron1, neuron2, neuron3, ..., neuronM]

network = list()
...
network = [layer1, layer2, layer3, ..., layerZ]
```

Figura 1: Formato de la red neuronal

Para la clasificación de datos se genera una ecuación sigmoidal que tiene la siguiente forma

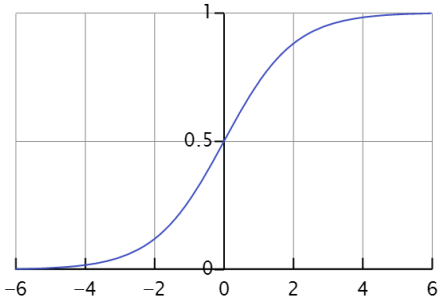
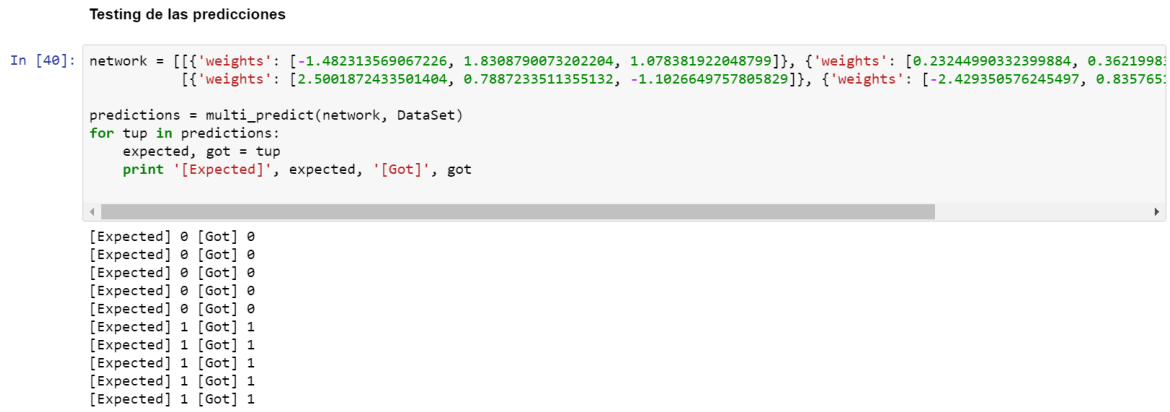


Figura 2: Gráfico de interacción sigmoidal en la red

Para poder precisar un etiquetado en formato binario, 0 referente a si equivale al etiquetado realizado y 1 si no lo es. Para que luego de tener una red entrenada, podamos generar un éxito real del 90% o más.

Aquí adelante presentamos un pequeño ejemplo realizado sobre semillas, la cual fue entrenada con un gran conjunto de datos, los que esperaríamos fuesen los datos subidos y etiquetados en la primera etapa de nuestra aplicación.



Testing de las predicciones

```
In [40]: network = [[{'weights': [-1.482313569067226, 1.8308790073202204, 1.078381922048799]}, {'weights': [0.23244990332399884, 0.3621998
                    [{'weights': [2.5001872433501404, 0.7887233511355132, -1.1026649757805829]}, {'weights': [-2.429350576245497, 0.835765

         predictions = multi_predict(network, DataSet)
         for tup in predictions:
             expected, got = tup
             print '[Expected]', expected, '[Got]', got

         [Expected] 0 [Got] 0
         [Expected] 0 [Got] 0
         [Expected] 0 [Got] 0
         [Expected] 0 [Got] 0
         [Expected] 0 [Got] 0
         [Expected] 1 [Got] 1
         [Expected] 1 [Got] 1
         [Expected] 1 [Got] 1
         [Expected] 1 [Got] 1
         [Expected] 1 [Got] 1
```

Figura 3: Resultados de investigación en redes FF

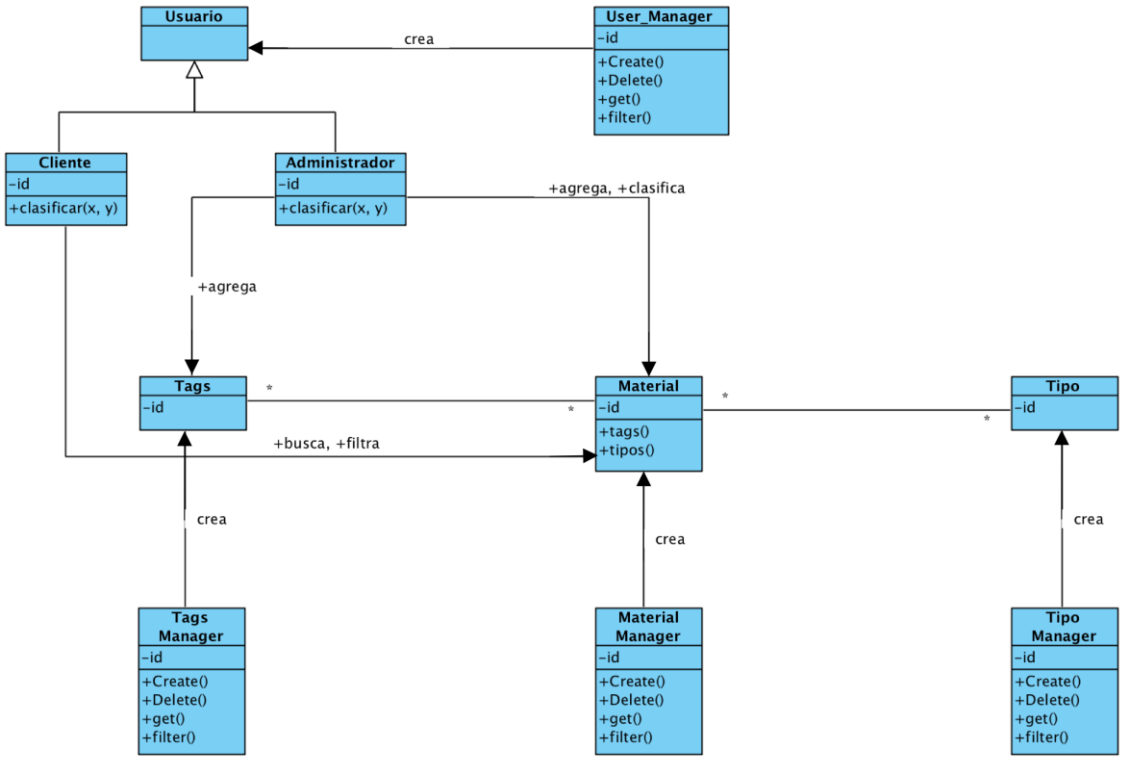
Esto señala que la elaboración de este tipo de red, será satisfactorio una vez se haya logrado entrenar con la suficiente cantidad de datos.

Selección de Patrones de Diseño

Intención	Patrón de Diseño	Razonamiento
Como equipo, y con la debida aprobación del cliente, creemos que la aplicación debe ser escalable a una red neuronal	MVC	Usaremos Django, el cual es MVC. En estricto rigor deseamos usar Python para poder utilizar las librerías de Keras en un futuro (las cuales proveen de las herramientas para trabajar con una red neuronal). Como Django es de Python, es la mejor forma de trabajar en este caso. La clase TagsManager deberá tener implementada la

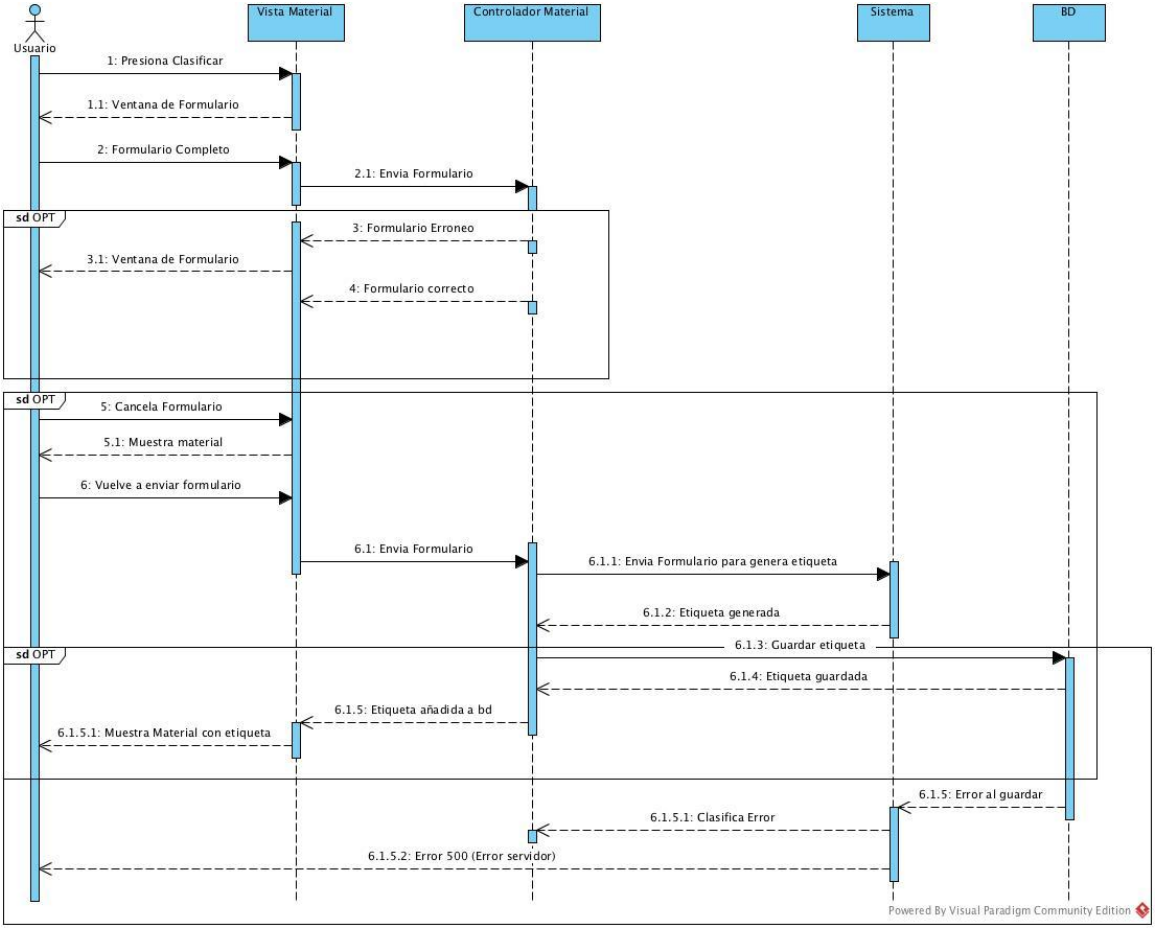
		clasificación mediante Keras.
Se desea crear distintos tipos de datos o varias familias de datos, las cuales deben funcionar de distinta manera para cada caso o perfil de usuario	Abstract-Factory	La clase Tags debe ser capaz de mostrar los distintos tipos de tags para cada material, pero estos tag pertenecen a algunas familias de clasificación las cuales pueden ser agrupadas y mostradas sin que el usuario sepa qué tipo de datos es el que está viendo en su material. La clase TagsManager, que es el controlador de Tags, donde el administrador entregará la información sobre el tipo de tags que desea crear.

Diagrama de Clases

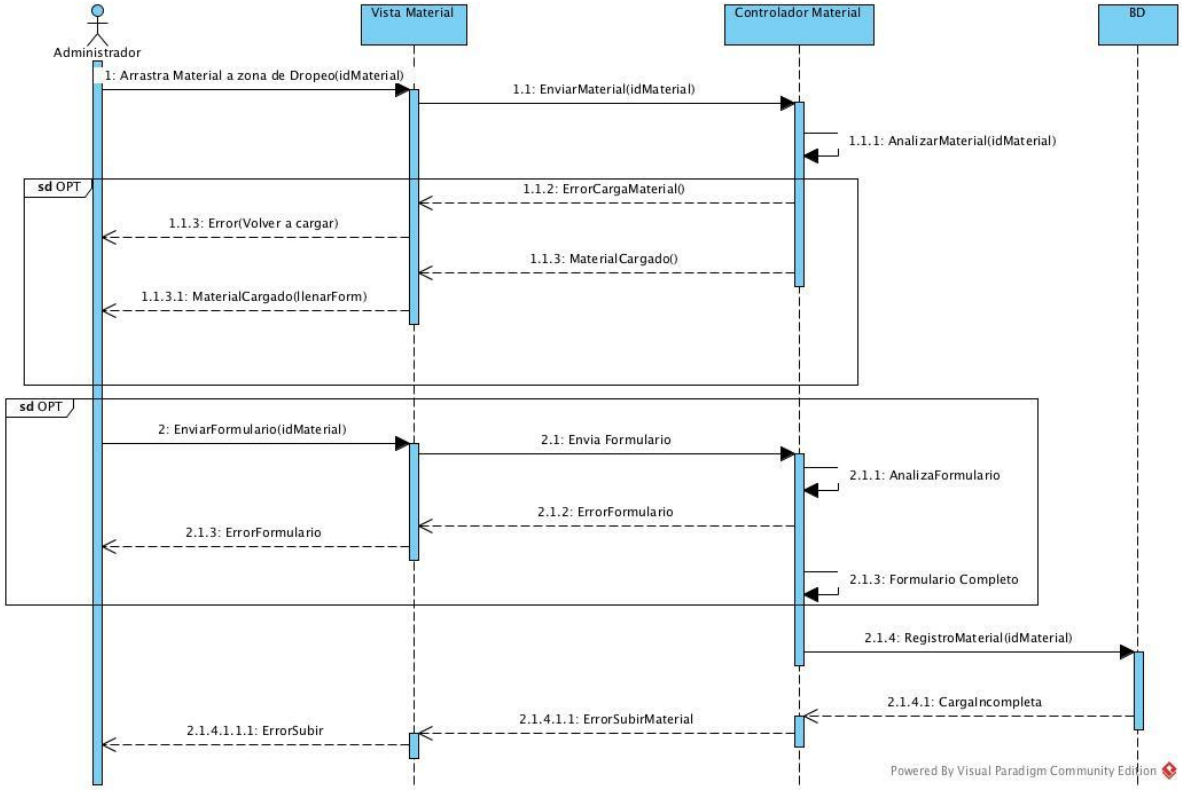


Diagramas de Secuencia

Usuario Clasifica Material



Administrador agrega nuevo material



Análisis de Trade-off

1. [Question] Se obtiene la pregunta:
   - ¿Qué nuevas funciones se deben agregar para poder almacenar el historial de búsqueda de un usuario?

2. [Options] Se plantean las opciones:
   - O1: Crear nueva tabla en la BD con el registro de actividad de un usuario.
   - O2: A cada material agregarle una tabla con los usuarios que lo han visitado.

3. [Criteria] Se define el Criterio:
   - C1: Escalabilidad
   - C2: Usabilidad
   - C3: Rendimiento
   - C4: Mantenimiento
   - C5: Complejidad

4. [Trade-Off] Se realiza la tabla con el análisis. El puntaje está definido de la siguiente manera:
    - ++ (Muy fuerte)
    - + (Fuerte)
    - 0 (No disponible)
    - - (Baja)
    - -- (Muy baja).

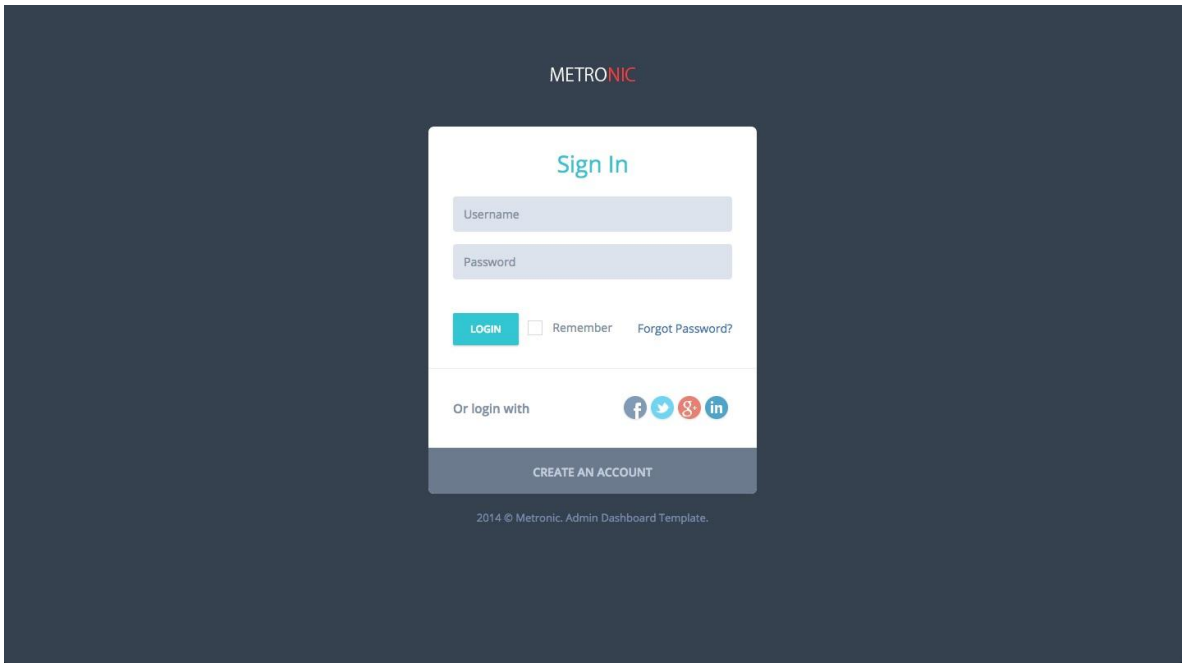
Criterio\Opciones	O1	O2
C1	++	--
C2	+	-
C3	++	--
C4	-	+
C5	0	+

- La opción 1 posee una gran cantidad de beneficios, partiendo por la Escalabilidad que es uno de los factores más grandes en este proyecto. La cantidad de material que habrá será mucho mayor a la cantidad de usuarios, por lo tanto tener sólo una tabla para cada usuario hace que la BD no se sature rápidamente. Por otro lado, el mantenimiento de esta tabla se hace difícil ya que puede haber casos en que el material que esté guardado como historial haya desaparecido o bien haya sido reemplazado por otro, por lo que hay que trabajar en una técnica para suplir eso.
- La opción 2 no tiene impacto alguno en la Escalabilidad, Usabilidad y Rendimiento ya que hay que tener tantas tablas como material se tenga guardado. Esto hace imposible escalar con el tiempo a una gran cantidad de materiales. Sin embargo, el Mantenimiento se hace muy simple ya que si un material se borra se elimina de inmediato la referencia de un usuario a este.

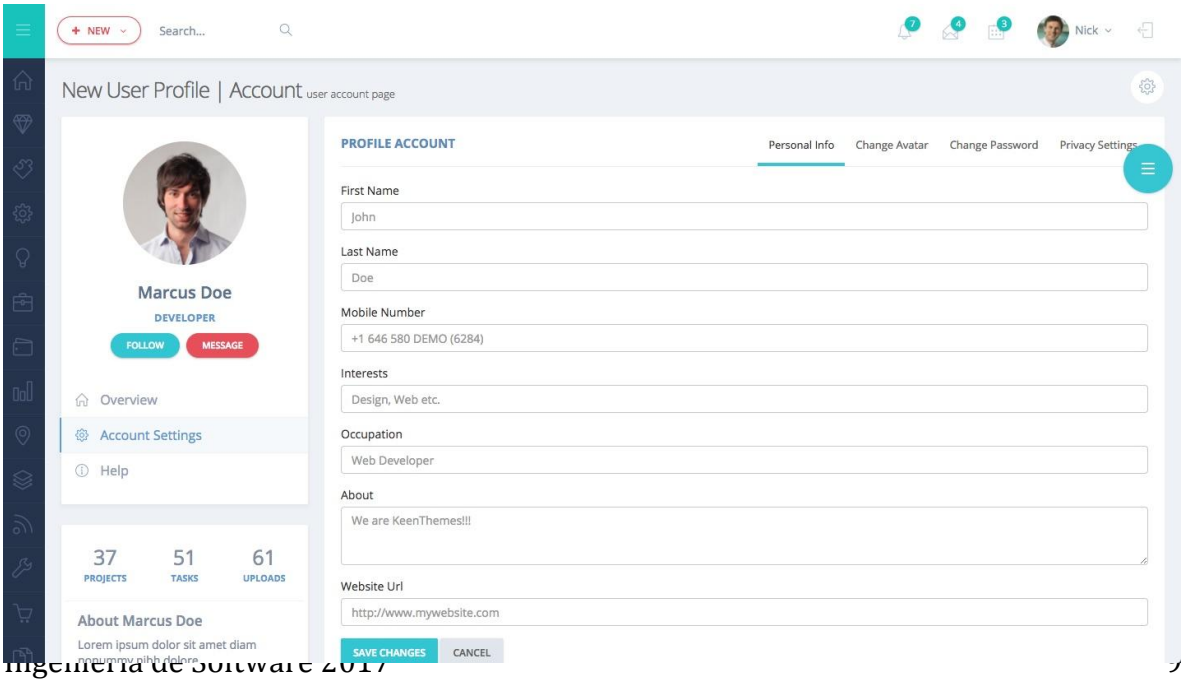
La opción 1 es aquella que hemos seleccionado como equipo de trabajo ya que favorece enormemente la Escalabilidad, Usabilidad y Rendimiento, que son los factores claves de un correcto funcionamiento de la plataforma. También se eligió porque al momento de buscar el historial, si se realiza con la opción 2 se debe buscar en cada uno de los materiales si existe alguna referencia al usuario en cuestión, sin embargo, en la opción 1 sólo se debe acceder a la tabla que posee el usuario.

Anexos

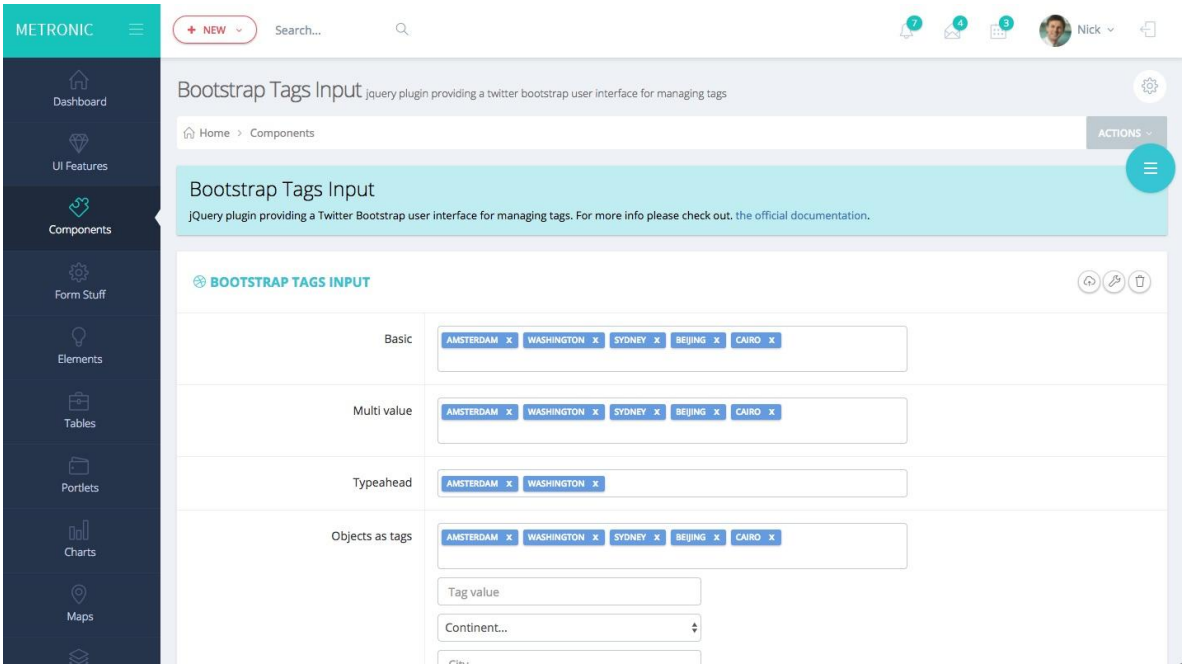
1. Vista Login/Register



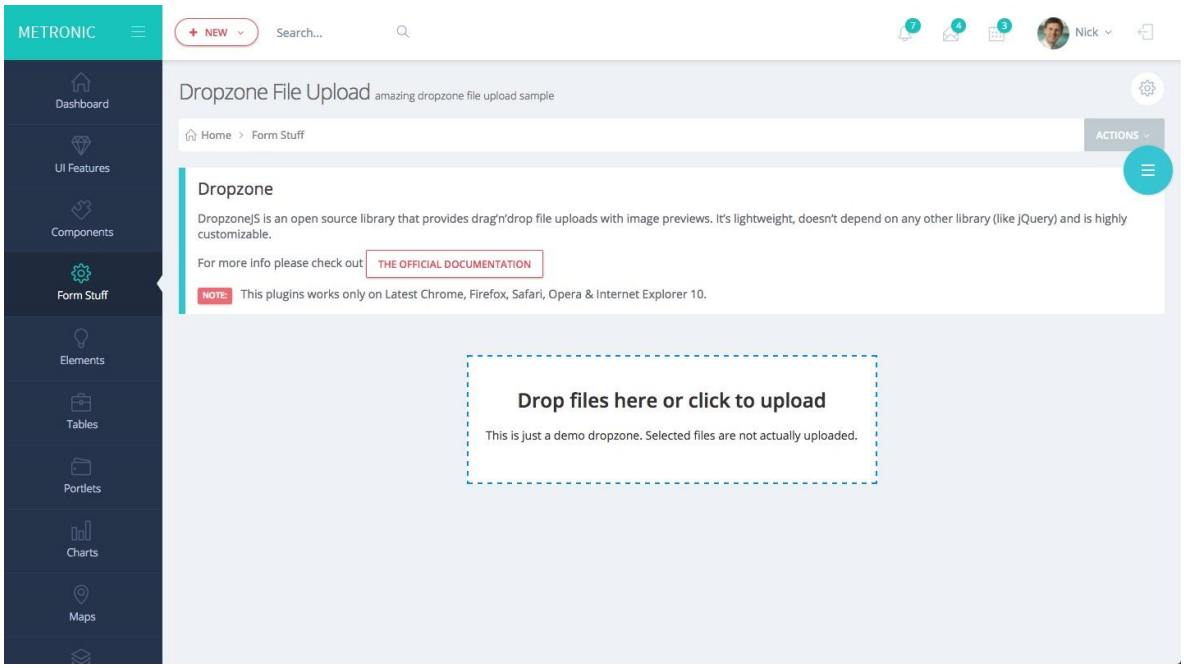
2. Vista Perfil Usuario



3. Vista Sistema de Tags



4. Vista sistema de subida de archivos



5. Vista del buscador