Aplicación FIWARE mejorada

Práctica 2 – Gestión de Datos en Entornos Inteligentes

Objetivo

Construir una aplicación FIWARE con funcionalidades de creación, modificación y borrado de datos, y una apariencia estética cuidada.

Entrega

Un archivo ZIP con la carpeta de la aplicación web. **NO INCLUYAS la carpeta *.venv*** en el archivo comprimido de entrega. En cambio, **incluye un fichero *requirements.txt*** obtenido ejecutando *pip freeze > requirements.txt* en la carpeta principal de la práctica.

[1p] Modelo de datos ampliado

1. Añadir al modelo de datos el tipo de entidad ***Employee***. Cada empleado trabajará en un solo *Store*. Tendrá los atributos *id*, *name*, *email*, *dateOfContract*, *category* ('Manager', 'Regular', 'Intern'), *salary*, *skills* ('MachineryDriving', 'WritingReports', 'CustomerRelationships'), *username*, *password*.
2. Añadir al tipo entidad ***Store*** los atributos *image*, *url*, *telephone*, *countryCode* (2 caracteres), *capacity* (metros cúbicos), *description* (texto amplio), *temperature*, *relativeHumidity*.
3. Añadir al tipo de entidad ***Product*** los atributos *image* (tipo *Text* que almacenará la imagen en Base64) y *color* (tipo *Text* que almacenará el color RGB en hexadecimal).
4. Crear el **diagrama de entidades UML** usando, p. ej. *draw.io* y mostrarlo en el apartado *Home* de la aplicación.

[2p] Formularios de entrada de datos

1. Añadir las vistas de **creación / modificación** de un ***Product***, un ***Store*** y un ***Employee*** mediante formulario.
2. Añadir una vista de **creación / modificación** de una ***Shelf*** dentro de un *Store*.
3. Utilizar el mayor número de elementos distintos de entrada.
4. Incluir las **reglas de validación HTML y JS** apropiadas.
5. La vista de creación y modificación es la misma, simplemente, en la vista de modificación los datos existentes aparecerán precargados en el formulario.
6. El formulario incluirá un **botón de confirmación** y otro de **cancelación**.

Codificación Base64 y Orion

La codificación Base64 asigna un carácter del conjunto de 64 caracteres [A-Z][a-z][0-9]{+,/} a cada grupo de 6 bits en la secuencia original. En concreto, cada grupo de 3 octetos (24 bits) de la secuencia original se codifica con 4 caracteres del conjunto mencionado. Si la secuencia original no es múltiplo de 3, se añaden 1 o 2 bits de *padding*. Este hecho se refleja añadiendo 1 carácter "=" al final de la secuencia de caracteres producida si se ha añadido 1 bit de *padding* y 2 caracteres "=" si se han añadido 2 bits.

Orion no acepta algunos caracteres en el valor de un atributo. En concreto, no acepta el carácter "=" utilizado como *padding* en la codificación Base64. Por tanto, debemos **eliminar dichos caracteres** en caso de estar presentes y **volverlos a añadir cuando recuperemos datos de Orion**.

P. ej., el siguiente código codifica en Base64 la secuencia de bits *data*, eliminando los caracteres de *padding* "=":

```
enc_data_wo_pad = base64.b64encode(data).decode('utf-8').rstrip('=')
```

En cambio, el siguiente código vuelve a añadir los caracteres de *padding* a partir de una secuencia de caracteres en la que se han eliminado los caracteres de *padding*:

```
enc_data_w_pad = enc_data_wo_pad.ljust(math.ceil(len(enc_data_wo_pad) / 4) * 4, '=')
```

Proveedores de contexto externo

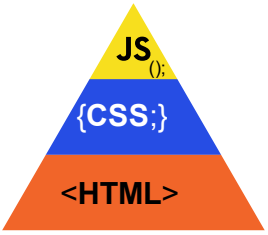
Los atributos ***temperature*** y ***relativeHumidity*** de un *Store* serán proporcionados por un proveedor de contexto externo (la aplicación que corre en el contenedor *tutorial*). El atributo ***tweets*** de un *Store* también será proporcionado por un proveedor externo (de nuevo, la aplicación que corre en el contenedor *tutorial*). El registro de estos dos proveedores externos se hace siguiendo las peticiones a Orion descritas en el tutorial *Context Providers*.

Suscripciones a Orion

En nuestra aplicación recibiremos desde Orion las **notificaciones** contempladas en el tutorial *Subscriptions*, i.e. **cambio de precio de un *Product*** y **bajo *stock* de un *Product* en un *Store***. El alta de estas suscripciones en Orion se efectúa como en dicho en tutorial.

Enviar notificaciones desde el servidor al navegador para que se actualicen los elementos de la interfaz de usuario involucrados. Usar Flask-SocketIO en el servidor y Socket.IO en el navegador.

Interfaz de usuario HTML + CSS + JS pura



Como principios generales:

- Cuando algo se pueda hacer tanto mediante CSS como mediante JS, usar CSS
- Evitar al máximo generar código HTML en el código JS. Siempre que se pueda, el código JS deberá actualizar el valor de los atributos elementos HTML ya presentes en la página en lugar de añadir nuevos elementos.

Visualización de imágenes codificadas en Base64

El *tag* HTML `<img>` acepta en el atributo *src* un valor de tipo Data URL. En concreto, podemos mostrar una imagen PNG mediante `<img src="<base64-encoded-image>">`. Si trabajamos con imágenes JPEG, bastaría con cambiar `image/png` por `image/jpg`.

[3p] Estructura de la interfaz

1. Vistas ***Products*, *Stores* y *Employees***
    a. Mostrar la lista de entidades como una **tabla** donde **cada entidad incluirá su imagen, su nombre y dos enlaces tipo botón de borrado y modificación**. Además, cada entidad en la tabla mostrará los siguientes **atributos específicos**:

- *Product*: *color*, *size*
- *Store*: *countryCode*, *temperature*, *relativeHumidity*
- *Employee*: *category*, *skills*

b. Añadir al principio de la vista un enlace tipo **botón para añadir una nueva entidad**.

2. Vista ***Product***
   a. Mostrar la **tabla de *InventoryItems* agrupada por *Store***: para cada *Store*, mostrar una fila con el nombre del *Store* y el valor de *stockCount* para ese producto y, a continuación, las filas de con los valores de *shelfCount* para las distintas *Shelfs* que contienen ese *Product*.
   b. En cada encabezado de grupo *Store* añadir un botón tipo enlace que permita **añadir un *InventoryItem* con ese *Product* a otra *Shelf*** que no sea ninguna de las que ya contiene ese *Product*. Los posibles valores de la *Shelf* para el nuevo *InventoryItem* deben restringirse mediante un elemento tipo *select* que cargue dinámicamente las *Shelfs* de ese *Store* que todavía no contengan ese *Product*.
3. Vista ***Store***
   a. Mostrar la **tabla de *InventoryItems* agrupada por *Shelf***: para cada *Shelf*, mostrar una fila con su id y su nivel de llenado (utilizar una barra de progreso) y, a continuación, las filas de con los *Products*, incluyendo sus atributos *name*, *price*, *size*, *color*, *stockCount* y *shelfCount*.
   b. Añadir un botón tipo enlace que permita **añadir una *Shelf* a ese *Store***.
   c. En cada encabezado de grupo *Shelf* añadir un botón tipo enlace que permita **modificar los datos de esa *Shelf***.
   d. En cada encabezado de grupo *Shelf* añadir un botón tipo enlace que permita **añadir un *InventoryItem* de otro *Product*** que todavía no esté presente en esa *Shelf*. Los posibles valores de *Product* para el nuevo *InventoryItem* deben restringirse mediante un elemento tipo *select* que cargue dinámicamente los *Products* existentes que todavía no estén presentes en esa *Shelf*.
   e. Incluir en cada *InventoryItem* un botón tipo enlace que permita **comprar una unidad de ese producto**. Para ello, utiliza la siguiente petición a Orion (que se corresponde con la función *update_attrs(id, attrs_vals)* de la librería *ngsiv2.py* que implementaste):

      PATCH */v2/entities/<inventoryitem_id>/attrs*
      Body:

```
{
      "shelfCount": {"type":"Integer", "value": {"$inc": -1}},
      "stockCount": {"type":"Integer", "value": {"$inc": -1}}
}
```

   f. Incluir la información de la **temperatura y humedad** relativa utilizando iconos y colores diferentes en función de los valores que tomen. Mostrar el valor numérico al lado del cada icono.
   g. Incluir los ***tweets* de ese Store** después de la tabla de *InventoryItems*. Usar un icono que recuerde a X (Twitter) a la izquierda de cada *tweet*.
   h. Incluir un **apartado de notificaciones**, donde se mostrará cada notificación que se reciba (p. ej. la notificación de *stock* bajo para alguno de los productos del *Store*).
4. Cuando se reciba una **notificación de cambio de precio** de un producto, **reflejar dicho cambio en todas las vistas** donde aparece.

[2p] Aspectos visuales

1. En la vista ***Employee***, incluir la **foto del empleado** con una transición CSS que **amplie la foto** cuando se pasa el ratón por encima de ella.
2. En la vista ***Store***, incluir una **foto del almacén** con una transición CSS que **amplíe la foto** y, al mismo tiempo, una animación CSS que **rote la imagen 360 grados**.
3. Para la barra de progreso que indica el nivel de llenado de una ***Shelf***, utilizar distintos **colores según el nivel de llenado**.
4. Utilizar **colores, iconos, elementos visuales, etc. para compactar la información mostrada**, especialmente en las vistas con **tablas**. Utilizar, p. ej. los iconos de Font Awesome, escogiendo de modo adecuado. P. ej. los siguientes atributos se pueden mostrar de esta forma:
    - *color*: con un cuadrado del color correspondiente.
    - *country*: con un icono con la bandera del país.
    - *category*: con un icono distinto según la categoría.
    - *skills*: con iconos que muestren los distintos valores.
5. La **barra de navegación** deberá resaltar la sección (*Home / Products / Stores / Employees*) activa en cada momento y permanecer visible cuando se efectúe *scroll*.
6. Añadir una **pestaña *Stores Map*** a la barra de navegación. Esta vista mostrará las imágenes de los *Stores* sobre el mapa de Berlín. Cuando se pase el ratón sobre un Store, se deberá mostrar una tarjeta (imagen + texto) con sus atributos principales. Cuando se pulse sobre un *Store*, se deberá acceder a su página de detalle.

[2p] Interfaz de usuario Bootstrap

Crear una versión de la aplicación usando Bootstrap, cambiando todos los elementos HTML por sus versiones Bootstrap. Utilizar los iconos de Bootstrap. En la medida de lo posible, incorporar elementos Bootstrap para los que no hay un equivalente HTML, p. ej. Accordion, Badges, Cards…