# ACTIVIDADES UD7 - SERVIDOR

Mediante las tareas que se detallan en este documento vamos a conseguir la parte de los resultados de aprendizaje cubiertos por la UD7, atendiendo a sus correspondientes criterios de evaluación:

| RESULTADOS DE APRENDIZAJE | CRITERIOS DE EVALUACIÓN |
|---|---|
| RA5. Desarrolla aplicaciones Web identificando y aplicando mecanismos para separar el código de presentación de la lógica de negocio. | c) Se han utilizado objetos y controles en el servidor para generar el aspecto visual de la aplicación web en el cliente.<br>d) Se han utilizado formularios generados de forma dinámica para responder a los eventos de la aplicación Web.<br>h) Se ha probado y documentado el código. |
| RA6. Desarrolla aplicaciones web de acceso a almacenes de datos, aplicando medidas para mantener la seguridad y la integridad de la información | f) Se han creado aplicaciones web que permitan la actualización y la eliminación de información disponible en una base de datos. |
| RA8. Genera páginas web dinámicas analizando y utilizando tecnologías y frameworks del servidor web que añadan código al lenguaje de marcas. | a) Se han identificado las diferencias entre la ejecución de código en el servidor y en el cliente web.<br>b) Se han reconocido las ventajas de unir ambas tecnologías en el proceso de desarrollo de programas.<br>c) Se han identificado las tecnologías y frameworks relacionadas con la generación por parte del servidor de páginas web con guiones embebidos.<br>d) Se han utilizado estas tecnologías y frameworks para generar páginas web que incluyan interacción con el usuario.<br>e) Se han utilizado estas tecnologías y frameworks, para generar páginas web que incluyan verificación de formularios.<br>f) Se han utilizado estas tecnologías y frameworks para generar páginas web que incluyan modificación dinámica de su contenido y su estructura.<br>g) Se han aplicado estas tecnologías y frameworks en la programación de aplicaciones web. |

Consideraciones adicionales:

- Se ha de continuar comentando el código mediante el código de tarea.
- Cualquier copia (ya sea de fuentes externas, literal de los apuntes...) que no sea una respuesta original será calificada con 0. El código no estructurado, que presente dificultades para se leído no podrá ser evaluado.

| Actividad 1: Comentarios en el código | | | | |
|---|---|---|---|---|
| **Código**: UD7.1 | **CE**: RA5.h | IE: I3, I6 | **Puntos**: 10 | **Estimación**: 30 min |

**DESCRIPCIÓN**

Para cada una de las actividades que siguen a ésta, inserta comentarios en el código para poder delimitar cada una de las soluciones que des. Para eso, inserta un comentario previo a cada fragmento de código que crees, especificando el código de cada tarea.

El código de cada tarea se formará con el código de la actividad, más la letra que identifica a la tarea en particular.

Ten en cuenta que pueden ser comentarios en Python o en HTML.

Las respuestas que no sean identificables mediante comentario con el correspondiente código **podrán no ser evaluadas**.

**ENLACES**

- Comentarios en Python
- Comentarios en HTML

| Actividad 2: Vistas, plantillas y URLs | | | | |
|---|---|---|---|---|
| **Código**: UD7.2 | **CE**: RA5.c, RA6.f | IE: I3, I6 | **Puntos**: 10 por cada CE | **Estimación**: 4 h |

**DESCRIPCIÓN**

Vamos a empezar a completar la aplicación mercaelx, pero autogenerando los formularios de la aplicación. Para ello crearemos las vistas, plantillas y URLs de forma que no tengamos que definir los formularios, sino que se generen automáticamente. Entre paréntesis se detalla a qué criterio de evaluación contribuye cada uno de los siguientes apartados:

a) (RA6.f) Crea todas las vistas detalladas en la Tabla 1 del Anexo I, para las apps core y directorio_comercios. (**4.5 puntos**)

b) (RA5.c) Crea todas las URLs asociadas a las vistas anteriores. Ahora, por cada URL acabada en "_detail" tendrás adicionalmente tres acabadas en "_create", "_update" y "_delete". Respeta la nomenclatura iniciada en la unidad anterior. (**3 puntos**)

c) (RA5.c) Crea una plantilla "base_create_update.html" que recoja las partes comunes de todas las plantillas para crear y actualizar (básate en la plantilla del portfolio). En esta plantilla base utilizaremos variables pasadas por contexto (ver TIP 2), para completar el h2 y el href del botón eliminar. Tendrás que especificar en las vistas esta template a utilizar, ya que no se seguirá la nomenclatura por defecto. Las variables serán (defínelas todas como atributos en las vistas, aunque puede que hayas de darle valor nulo dependiendo del caso): (**3 puntos**)

- ✔ titulo_creacion y titulo_actualizacion: serán el texto del h2, dependiendo de si estamos en modo crear o actualizar. Tendrás que mostrar una u otra en función de que la acción sea crear, o actualizar.
- ✔ url_borrado: tendrá el nombre de la url de borrado, a la que habrá que llamar si se

pulsa el botón de borrar (solo habilitado en modo actualizar).

d) (RA6.f) Crea un mixin específico para pasar las variables de la tarea anterior por contexto, en todas las vistas que vayan a utilizar base_create_udpate.html. Para ello, extiende el método get_context_data (**3 puntos**)

e) (RA5.c) Crea una plantilla "base_confirm_delete.html" (básate en la plantilla del portfolio) que recoja los elementos comunes de las plantillas de borrado. Tendrás que especificar en las vistas la template a utilizar, ya que no se seguirá la nomenclatura por defecto. Se pasará por contexto las siguientes variables (defínelas como atributos en las vistas): (**2 puntos**)

- ✔ titulo: para completar el h2.
- ✔ mensaje_confirmación: para completar el elemento <p>.

f) (RA6.f) Crea un mixin específico para las vistas basadas en DeleteView que compruebe si existen dependencias en otras tablas. Se ha de cumplir lo siguiente:

- ✔ Al intentar borrar un objeto, si existen dependencias, se obtendrá un error. Para evitar esto e informar al usuario que es necesario borrar antes las dependencias, se mostrará un mensaje de error constatando esta circunstancia. Para ello, se ha de sobreescribir el método form_valid, según el esqueleto de ejemplo en TIP 1. (**1.5 puntos**)
- ✔ Construye el mensaje de error de forma dinámica, con la representación textual del objeto a eliminar. (**1 punto**)

g) (RA5.c) Vamos a establecer una ordenación diferente para las clases ListView, respecto a la ordenación establecida en el modelo, de la unidad anterior, mediante un nuevo parámetro de la URL: (**2 puntos**)

- ✔ El parámetro se llamará "ordering", y podrá tener el valor "asc" o "desc"
- ✔ Con "asc" los registros se ordenarán por id ascendentemente, y con "desc" descendentemente (por id).
- ✔ Para ello hemos de sobreescribir el método get_queryset e interceptar el parámetro pasado por la URL (ver TIP 3).

**DESCRIPCIÓN**

**TIP 1**: Un ejemplo de sobreescritura del método form_valid para verificar dependencias al borrar un objeto es el siguiente. Complétalo:

```
def form_valid(self, request, *args, **kwargs):
    try:
        super().delete(*args, **kwargs)
    except:
            messages.error(self.request, "Existen dependencias para el objeto {}. Elimine antes dichas dependencias".format{})
        return HttpResponseRedirect(reverse('home'))
```

**TIP 2**: Para pasar variables a una plantilla a través del contexto de una CBV, puedes utilizar el método get_context_data, como se puede ver en el ejemplo de este enlace.

**TIP 3**: Para sobreescribir el método get_queryset de las ListView, se muestra un ejemplo en este enlace. Solo para cambiar el orden, llama al método get_queryset de la superclase y almacena el resultado en una variable, a partir de la cual aplica order_by. Devuelve el resultado ordenado.

Para recuperar el parámetro ordering de la URL, básate en este enlace, donde explica cómo hacerlo (basándose en los **query_params** del request).

| **Actividad 3: Formularios** | | | | |
|---|---|---|---|---|
| **Código**: UD7.3 | **CE**: RA5.d | IE: I3, I6 | **Puntos**: 10 | **Estimación**: 30 min |

**DESCRIPCIÓN**

En esta actividad vamos a definir los formularios que van a estar asociados a cada una de las vistas, con sus correspondientes particularidades.

a) Definición de un formulario por cada par CreateView-UpdateView, con las siguientes características (**5 puntos**):

- ✔ El nombre [nombre del modelo]Form, basado en la clase ModelForm.
- ✔ Se ha de situar cada uno en su correspondiente forms.py, dentro de cada app.
- ✔ Se han de incluir todos los campos de cada modelo.
- ✔ No es necesario hacer modificaciones en el layout.

b) Mediante un método clean, implementa una validación que compruebe que los campos de teléfono tienen 9 dígitos, y las direcciones de correo electrónico introducidas siguen una estructura válida, en los siguientes formularios (**5 puntos**):

- ✔ AsociacionForm
- ✔ ComercioForm

| **Actividad 4: Interactividad** | | | | |
|---|---|---|---|---|
| **Código**: UD7.4 | **CE**: RA8 | IE: I3, I6 | **Puntos**: 10 | **Estimación**: 3h |

**DESCRIPCIÓN**

En esta actividad vamos a añadir interactividad y estilismo a los formularios creados anteriormente. Para ello, realiza las siguientes tareas:

a) Configura el framework de mensajes de Django (incluido el código de error para que se tome la clase de Bootstrap correcta). Añade un mensaje de éxito a cada una de las vistas de creación y actualización que incluya la representación textual del objeto. (**2 puntos**)

b) Modifica la plantilla base_create_udpate.html para que se utilicen los tags de crispy forms, y se inserte el formulario correctamente como se mostró en el ejemplo guiado. (**1 punto**)

c) Sobreescribe el método __init__ de cada una de las clases de formulario para definir un helper y un layout (como se mostró en el ejemplo guiado). (**1 punto**)

d) Configura el layout de los formularios que se detalla en el Anexo I - Tabla 2. (**6 puntos**, divididos proporcionalmente por el número de formularios)

# ANEXO I

**Tabla 1**: definición de vistas

**core**

| Nombre | Características |
|---|---|
| ProvCreateView<br>ProvUpdateView | - Modelo: Provincia<br>- Plantilla: base_create_udpate.html<br>- success_url: provincia_update |
| ProvDeleteView | - Modelo: Provincia<br>- Plantilla: base_confirm_delete.html<br>- success_url: provincia_list<br>- Verificación de dependencias |
| CiudadCreateView<br>CiudadUpdateView | - Modelo: Ciudad<br>- Plantilla: base_create_udpate.html<br>- success_url: ciudad_update |
| CiudadDeleteView | - Modelo: Ciudad<br>- Plantilla: base_confirm_delete.html<br>- success_url: ciudad_list<br>- Verificación de dependencias |
| DistCreateView<br>DistUpdateView | - Modelo: Distrito<br>- Plantilla: base_create_udpate.html<br>- success_url: distrito_update |
| DistDeleteView | - Modelo: Distrito<br>- Plantilla: base_confirm_delete.html<br>- success_url: distrito_list<br>- Verificación de dependencias |

**directorio_comercios**

| Nombre | Características |
|---|---|
| AsoCreateView<br>AsopdateView | - Modelo: Asociacion<br>- Plantilla: base_create_udpate.html<br>- success_url: asociacion_update |
| AsoDeleteView | - Modelo: Asociacion |

| | |
|---|---|
| | - Plantilla: base_confirm_delete.html<br>- success_url: asociacion_list<br>- Verificación de dependencias |
| CategoriaCreateView<br>CategoriaUpdateView | - Modelo: Categoria<br>- Plantilla: base_create_udpate.html<br>- success_url: cat_update |
| CategoriaDeleteView | - Modelo: Categoria<br>- Plantilla: base_confirm_delete.html<br>- success_url: cat_list<br>- Verificación de dependencias |
| SubcategoriaCreateView<br>SubcategoriaUpdateView | - Modelo: Subcategoria<br>- Plantilla: base_create_udpate.html<br>- success_url: subcat_update |
| SubcategoriaDeleteView | - Modelo: Subcategoria<br>- Plantilla: base_confirm_delete.html<br>- success_url: subcat_list |
| ComercioCreateView<br>ComercioUpdateView | - Modelo: Comercio<br>- Plantilla: base_create_udpate.html<br>- success_url: comercio_update |
| ComercioDeleteView | - Modelo: Comercio<br>- Plantilla: base_confirm_delete.html<br>- success_url: comercio_list |

**Tabla 2**: definición de formularios

| Nombre | Características |
|---|---|
| ProvinciaForm | - Código: utiliza 3 columnas del grid<br>- Nombre: utiliza el resto de columnas<br>Los dos campos se dispondrán en una misma fila |
| CiudadForm | - Código: utiliza 3 columnas del grid<br>- Nombre: utiliza el resto de columnas<br>Los dos campos (código y nombre) se dispondrán en una misma fila<br>- Provincia: dispuesto en una sola fila, abarcará todo el ancho. |
| DistritoForm | - Nombre: utiliza 6 columnas del grid |

| | |
|---|---|
| | - Ciudad: utiliza 6 columnas del grid |
| AsociacionForm | - Nombre: utiliza todas las columnas de la fila<br>- Direccion: utiliza todas las columnas de la fila<br>- Ciudad: utiliza todas las columna de la fila<br>- Correo electrónico: utiliza 6 columnas<br>- Teléfono: utiliza 6 columnas<br>Correo electrónico y teléfono se disponen en la misma fila |
| CategoriaForm | - Código: utiliza 3 columnas del grid<br>- Nombre: utiliza el resto de columnas<br>Los dos campos se dispondrán en una misma fila |
| SubcategoriaForm | - Código: utiliza 3 columnas del grid<br>- Nombre: utiliza el resto de columnas<br>Estos dos campos se dispondrán en una misma fila<br>- Categoría: abarca toda la fila. |
| ComercioForm | - Nombre: ocupará 6 columnas.<br>- Asociación: ocupará 6 columnas.<br>Campos nombre y asociación dispuestos en una sola fila, uno al lado del otro.<br>- Ciudad, ocupará 4 columnas<br>- Código postal, ocupará 4 columnas<br>Ciudad y código postal en una misma fila<br>- Dirección, abarcará todas las columnas.<br>- Correo electrónico: utiliza 6 columnas<br>- Teléfono: utiliza 6 columnas<br>Correo electrónico y teléfono se disponen en la misma fila<br>- Asociacion: 12 columnas. |