SuperTODO

Implementa un proyecto web Django para **gestión de tareas**.

La idea es disponer de un software en el que se pueda visualizar tus tareas pendientes (completadas), así como añadir nuevas y/o editar ya existentes.

1. Requerimientos

Para que puedas trabajar con normalidad en este ejercicio debes tener instalado en tu máquina:

1. `uv` → Gestor de paquetería y proyectos Python.
2. `just` → Lanzador de comandos como recetas.

2. Puesta en marcha

Se proporciona una *receta* `just` para la puesta en marcha del proyecto:

```
just setup
```

¿Qué ha ocurrido?

- Se ha creado un entorno virtual en la carpeta `.venv`
- Se han instalado las dependencias del proyecto.
- Se ha creado un proyecto Django en la carpeta `main`
- Se han aplicado las migraciones iniciales del proyecto.
- Se ha creado un *superusuario* con credenciales: `admin` - `admin`
- Se ha establecido el *timezone* a `Atlantic/Canary` en `settings.py`

3. Aplicaciones

Habrá que añadir las siguientes aplicaciones:

`shared`	Artefactos compartidos.
`tasks`	Gestión de tareas.

Se proporciona una *receta* `just` para añadir una aplicación:

```
just startapp <app>
```

Esta receta no sólo crea la carpeta de la aplicación sino que añade la línea correspondiente de configuración en la variable `INSTALLED_APPS` del fichero `settings.py`.

4. Modelos

4.1. `tasks.Task`

Campo	Tipo
`name`$^{(*)}$	*str*
`description`$^{(\varnothing)}$	*str*
`slug`$^{(u)}$	*str*
`completed`$^{(*\Delta=\texttt{False})}$	*bool*
`created_at`$^{(*)}$	*datetime*
`updated_at`$^{(*)}$	*datetime*

4.2. Carga de datos

Una vez que hayas creado los modelos y aplicado las migraciones, puedes cargar datos de prueba. Antes de nada asegúrate de que tu proyecto pasa los *tests core*.

Se proporciona una *receta* `just` para ello:

```
just test tests/test_core.py
```

Si todo ha ido bien, puedes cargar los datos. Se proporciona una *receta* `just` para ello:

```
just load-data
```

5. Requisitos de diseño

El proyecto deberá cumplir con los siguientes **requisitos de diseño** a nivel de organización interna del código:

- Utiliza herencia de plantillas, como mínimo, para la plantilla `base.html`.
- Utiliza inclusión de plantillas, como mínimo, para la cabecera `header.html`.
- Utiliza estáticos, como mínimo, para definir una hoja de estilos CSS.
- Utiliza formularios de modelo para todos los casos propuestos.

6. URLs

`/tasks/` ⇒ `tasks.views.task_list()`

- Mostrar el listado de tareas.
- Se debe mostrar el título SuperTODO y el subtítulo Todas las tareas.
- Cada tarea debe incluir:
  - Nombre de la tarea.
  - Si la tarea está completada debe aparecer junto a ella el *emoji* check. Y si no está completada el *emoji* cross.
  - Enlace a su detalle.
  - Enlace para cambio de estado (completada/no completada).
  - Enlace para editar la tarea.
  - Enlace para borrar la tarea.
- Debe existir un enlace para ver todas las tareas.
- Debe existir un enlace para ver las tareas completadas.
- Debe existir un enlace para ver las tareas pendientes.
- Debe existir un enlace para añadir una nueva tarea.
- Si no hay ninguna tarea en la base de datos, se debe mostrar el mensaje: Nada que hacer. ¡Enhorabuena!

`/tasks/pending/` ⇒ `tasks.views.task_list_pending()`

- Mostrar el listado de tareas pendientes.
- Se debe mostrar el título SuperTODO y el subtítulo Tareas pendientes.
- Cada tarea debe incluir:
  - Nombre de la tarea.
  - Si la tarea está completada debe aparecer junto a ella el *emoji* check. Y si no está completada el *emoji* cross.
  - Enlace a su detalle.
  - Enlace para cambio de estado (completada/no completada).
  - Enlace para editar la tarea.
  - Enlace para borrar la tarea.
- Debe existir un enlace para ver todas las tareas.
- Debe existir un enlace para ver las tareas completadas.
- Debe existir un enlace para ver las tareas pendientes.
- Debe existir un enlace para añadir una nueva tarea.
- Si no hay ninguna tarea en la base de datos, se debe mostrar el mensaje: Nada que hacer. ¡Enhorabuena!

`/tasks/completed/` ⇒ `tasks.views.task_list_completed()`

- Mostrar el listado de tareas completadas.
- Se debe mostrar el título SuperTODO y el subtítulo Tareas completadas.
- Cada tarea debe incluir:
  - Nombre de la tarea.
  - Si la tarea está completada debe aparecer junto a ella el *emoji* check. Y si no está completada el *emoji* cross.
  - Enlace a su detalle.
  - Enlace para cambio de estado (completada/no completada).
  - Enlace para editar la tarea.
  - Enlace para borrar la tarea.
- Debe existir un enlace para ver todas las tareas.
- Debe existir un enlace para ver las tareas completadas.
- Debe existir un enlace para ver las tareas pendientes.
- Debe existir un enlace para añadir una nueva tarea.
- Si no hay ninguna tarea en la base de datos, se debe mostrar el mensaje: Nada que hacer. ¡Enhorabuena!

`/tasks/add/` ⇒ `tasks.views.add_task()`

- Añadir una nueva tarea.
- Se debe mostrar el título SuperTODO y el subtítulo Añadir tarea.
- El formulario debe incluir los siguientes campos:
  - Nombre de la tarea.
  - Descripción de la tarea.
- Una vez almacenada la nueva tarea, se debe redirigir al listado de tareas.
- Debe existir un enlace Cancelar que lleve al listado de todas las tareas.

`/tasks/comprar-el-pan/` ⇒ `tasks.views.task_detail()`

- Detalle de una tarea.
- Se debe mostrar el título SuperTODO.
- Se deben mostrar los campos:
  - Nombre de la tarea.
  - El texto Completada si la tarea está completada o Pendiente si la tarea está pendiente.
  - Descripción de la tarea.
- Debe existir un enlace Volver que lleve al listado de todas las tareas.

`/tasks/comprar-el-pan/delete/` ⇒ `tasks.views.delete_task()`

- Borrar una tarea.
- Redirigir al listado de tareas.

`/tasks/comprar-el-pan/edit/` ⇒ `tasks.views.edit_task()`

- Editar una tarea.
- Se debe mostrar el título SuperTODO y el subtítulo Editar tarea.
- El formulario debe incluir los siguientes campos:
  - Nombre de la tarea.
  - Descripción de la tarea.
- Una vez editada la tarea, se debe redirigir al detalle de la tarea actual.
- Debe existir un enlace Cancelar que lleve al listado de todas las tareas.

`/tasks/comprar-el-pan/toggle/` ⇒ `tasks.views.toggle_task()`

- Cambiar el estado de una tarea.
- Si la tarea estaba "completada" pasa a "pendiente" y viceversa.
- Una vez modificada la tarea, se debe redirigir al listado de tareas.

7. Administración

El modelo `Task` debe estar accesible desde la **interfaz administrativa** de Django.

