

# Tribu

Implementa un proyecto web Django que permita gestionar una pequeña red social.

La idea es disponer de un software en el que se pueda compartir contenido mediante usuarios autenticados.

## 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:

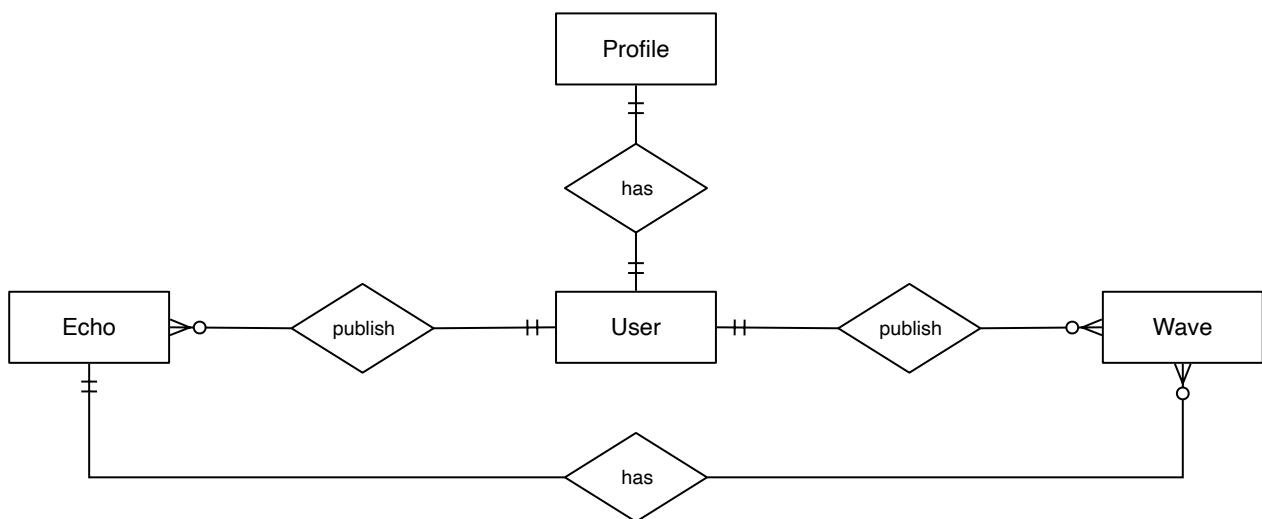
<code>shared</code>	Artefactos compartidos.
<code>accounts</code>	Gestión de autenticación.
<code>echos</code>	Publicaciones en la red social.
<code>waves</code>	Respuestas a publicaciones en la red social.
<code>users</code>	Perfiles de usuario.

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. echos.Echo

Campo	Tipo
<code>content<sup>(*)</sup></code>	<code>str</code>
<code>created_at<sup>(*Δ=auto)</sup></code>	<code>datetime</code>
<code>updated_at<sup>(*Δ=auto)</sup></code>	<code>datetime</code>
<code>user<sup>(*)</sup></code>	<code>foreign key</code>

### 4.2. waves.Wave

Campo	Tipo
<code>content<sup>(*)</sup></code>	<code>str</code>
<code>created_at<sup>(*Δ=auto)</sup></code>	<code>datetime</code>
<code>updated_at<sup>(*Δ=auto)</sup></code>	<code>datetime</code>
<code>user<sup>(*)</sup></code>	<code>foreign key</code>
<code>echo<sup>(*)</sup></code>	<code>foreign key</code>

### 4.3. users.Profile

Campo	Tipo
avatar <sup>(*Δ=noavatar.png)</sup>	<i>image</i>
bio <sup>(∅)</sup>	<i>str</i>
user <sup>(*)</sup>	<i>one2one</i>

### 4.4. Carga de datos

Una vez que hayas **creado los modelos** asegúrate de que tu proyecto pasa los *tests core*. Se proporciona una *receta just* para ello:

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

Ahora **aplica las migraciones** y, si todo ha ido bien, procede a la carga de datos iniciales. 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.
- Utiliza [conversores de ruta personalizados](#), como mínimo, para los distintos modelos.
- Utiliza “middleware” de [mensajes](#), como mínimo, para los mensajes de confirmación.
- Utiliza [URL canónica](#) para los modelos que procedan.

## 6. URLs

### 6.1. main.urls

`/` ⇒ `shared.views.index()`

- Página de inicio.
- Debe aparecer el nombre del proyecto [Tribu](#).
- Debe existir un enlace para ver todos los “echos”.

## 6.2. accounts.urls

`/login/` ⇒ `accounts.views.user_login()`

- Inicio de sesión.
- Se debe solicitar nombre de usuario y contraseña.
- Si el usuario y/o contraseña son incorrectos, se debe mostrar un mensaje de error: **Incorrect username or password**.
- Si el usuario y contraseña son correctos, se debe iniciar la correspondiente sesión y redirigir al índice del proyecto.
- Hay que tener en cuenta la redirección en el caso de que tengamos un parámetro `next` desde GET.
- Debe existir un enlace para registrarse (“signup”).

`/logout/` ⇒ `accounts.views.user_logout()`

- Cierre de sesión.
- Una vez cerrada la sesión, se debe redirigir al índice del proyecto.

`/signup/` ⇒ `accounts.views.user_signup()`

- Registro de usuario.
- Campos a desplegar en el formulario:
  - Nombre de usuario.
  - Contraseña.
  - Nombre.
  - Apellidos.
  - Correo electrónico.
- Aquí se debe crear también el perfil del usuario registrado. Será un perfil “vacío” pero vinculado con el usuario.
- Si el registro fue exitoso, se debe redirigir al índice del proyecto.
- Debe existir un enlace para logearse (“login”).

### 6.3. echos.urls

`/echos/` ⇒ `echos.views.echo_list()`

- Listado de todos los “echos” de la red social.
- Requiere estar autenticado.
- Deben estar ordenados de forma *descendente* por su *fecha de creación*.
- Para cada “echo” se debe mostrar:
  - El contenido (truncado si es mayor de **20 palabras** con *puntos suspensivos*).
  - El *nombre de usuario* que lo escribió con un enlace a su perfil.
  - Un enlace al detalle del “echo”.
  - El momento en el que se escribió el “echo” en formato: **11 months, 4 weeks ago**.
- Si no hay ningún “echo” se debe mostrar el mensaje: **No echos yet**.
- Se debe mostrar el número total de “echos” con el mensaje: **Tribu has posted 23 echos so far!**
  - Tener en cuenta el singular o el plural. Por ejemplo: **Tribu has posted 1 echo so far!**
  - Si no hay ningún “echo” no se debe mostrar este mensaje.
- Se debe mostrar un enlace para añadir un nuevo “echo”: **Add echo**.

`/echos/add/` ⇒ `echos.views.add_echo()`

- Añadir un nuevo “echo”.
- Requiere estar autenticado.
- Formulario únicamente con el contenido.
- Si la operación fue exitosa, se debe redirigir al detalle del “echo” creado.
- Si la operación fue exitosa, se debe mostrar el mensaje: **Echo added successfully**.
- Se debe mostrar un enlace **Cancel** que lleve al listado de “echos”.

[/echos/45/](#) ⇒ echos.views.echo\_detail()

- Detalle del “echo” con clave primaria 45.
- Requiere estar autenticado.
- Implementa y utiliza el método `get_absolute_url()` en el modelo Echo.
- Para el “echo” en cuestión se debe mostrar:
  - El contenido **completo**.
  - El *nombre de usuario* que lo escribió con un enlace a su perfil.
  - El momento en el que se escribió el “echo” en formato: **11 months, 4 weeks ago**.
- Se deben mostrar **sólo los 5 “waves” más recientes** ordenados de forma *descendente* por su *fecha de creación*.
- Para cada “wave” del “echo” en cuestión se debe mostrar:
  - El contenido **completo**.
  - El *nombre de usuario* que lo escribió con un enlace a su perfil.
  - El momento en el que se escribió el “wave” en formato: **2 months, 3 weeks ago**.
- Debe existir un enlace **View all waves** para ver todos los “waves” del “echo” **siempre y cuando** existan en total más de 5 “waves”.
- Si no hay ningún “wave” asociado a este “echo” se debe mostrar el mensaje: **No waves yet**.
- Debe existir un enlace **Add wave** para añadir un “wave” al “echo” en cuestión.
- Si el “echo” en cuestión no existe, se debe devolver una respuesta 404.
- Si el usuario logeado es el autor del “echo” en cuestión debe aparecer un enlace **Delete echo** para borrar el “echo”.
- Si el usuario logeado es el autor del “echo” en cuestión debe aparecer un enlace **Edit echo** para editar el “echo”.
- Por cada “wave” mostrado debe aparecer un enlace **Delete wave** para borrar el “wave” siempre que el usuario logeado sea el autor del “wave” en cuestión.
- Por cada “wave” mostrado debe aparecer un enlace **Edit wave** para editar el “wave” siempre que el usuario logeado sea el autor del “wave” en cuestión.

[/echos/45/waves/](#) ⇒ echos.views.echo\_waves()

- Detalle del “echo” con clave primaria 45.
- Todo es igual que para [/echos/45/](#) salvo que:
  - Se mostrarán **todos los “waves”** del “echo” independientemente de cuántos existan.
  - **No** debe existir un enlace para ver todos los “waves”.

`/echos/45/edit/ ⇒ echos.views.edit_echo()`

- Editar el “echo” con clave primaria 45.
- Requiere estar autenticado.
- Formulario únicamente con el contenido.
- Si la operación fue exitosa, se debe redirigir al detalle del “echo” editado.
- Si la operación fue exitosa, se debe mostrar el mensaje: **Echo updated successfully**.
- Se debe mostrar un enlace **Cancel** que lleve al detalle del “echo” en cuestión.
- Si el “echo” en cuestión no existe, se debe devolver una respuesta 404.
- Si el usuario logeado no es el autor del “echo”, se debe devolver una respuesta 403.

`/echos/45/delete/ ⇒ echos.views.delete_echo()`

- Borrar el “echo” con clave primaria 45.
- Requiere estar autenticado.
- Si la operación fue exitosa, se debe redirigir al listado de “echos”.
- Si la operación fue exitosa, se debe mostrar el mensaje: **Echo deleted successfully**.
- Si el “echo” en cuestión no existe, se debe devolver una respuesta 404.
- Si el usuario logeado no es el autor del “echo”, se debe devolver una respuesta 403.

`/echos/45/waves/add/ ⇒ echos.views.add_wave()`

- Añadir un “wave” al “echo” con clave primaria 45.
- Requiere estar autenticado.
- Formulario únicamente con el contenido.
- Si la operación fue exitosa, se debe redirigir al detalle del “echo” en cuestión.
- Si la operación fue exitosa, se debe mostrar el mensaje: **Wave added successfully**.
- Se debe mostrar un enlace **Cancel** que lleve al detalle del “echo” en cuestión.
- Si el “echo” en cuestión no existe, se debe devolver una respuesta 404.

`/waves/27/edit/ ⇒ waves.views.edit_wave()`

- Editar el “wave” con clave primaria 27.
- Requiere estar autenticado.
- Formulario únicamente con el contenido.
- Si la operación fue exitosa, se debe redirigir al detalle del “echo” en cuestión.
- Si la operación fue exitosa, se debe mostrar el mensaje: **Wave updated successfully**.
- Se debe mostrar un enlace **Cancel** que lleve al detalle del “echo” en cuestión.
- Si el “echo” en cuestión no existe, se debe devolver una respuesta 404.
- Si el usuario logeado no es el autor del “wave”, se debe devolver una respuesta 403.

`/waves/27/delete/ ⇒ waves.views.delete_wave()`

- Borrar el “wave” con clave primaria 27.
- Requiere estar autenticado.
- Si la operación fue exitosa, se debe redirigir al detalle del “echo” en cuestión.
- Si la operación fue exitosa, se debe mostrar el mensaje: **Wave deleted successfully**.
- Si el “wave” en cuestión no existe, se debe devolver una respuesta 404.
- Si el usuario logeado no es el autor del “wave”, se debe devolver una respuesta 403.

#### 6.4. `users.urls`

`/users/ ⇒ users.views.user_list()`

- Mostrar listado de usuarios de la red social.
- Requiere estar autenticado.
- Deben aparecer los nombres de usuario.
- Debe existir un enlace al perfil de cada usuario.

/users/guido/ ⇒ echos.views.user\_detail()

- Perfil del usuario con nombre de usuario **guido**.
- Requiere estar autenticado.
- Deben aparecer los siguientes campos de **usuario**:
  - Nombre.
  - Apellidos.
  - Nombre de usuario.
  - Correo electrónico.
- Deben aparecer los siguientes campos de **perfil**:
  - Biografía.
  - Fotografía de “avatar”.
- Debe aparecer un enlace para **editar el perfil** (*sólo cuando se trata del usuario logeado*).
- Sólo deben aparecer los **5 últimos “echos”** publicados por el usuario ordenados de forma *descendente* por su *fecha de creación*.
- Para cada “echo” se debe mostrar:
  - El contenido (truncado si es mayor de **20 palabras** con *puntos suspensivos*).
  - El *nombre de usuario* que lo escribió con un enlace a su perfil.
  - Un enlace al detalle del “echo”.
  - El momento en el que se escribió el “echo” en formato: **11 months, 4 weeks ago**.
- Se debe mostrar un enlace para ver todos los “echos” del usuario, siempre y cuando el número total de “echos” sea mayor que 5.
- Si no hay ningún “echo” se debe mostrar el mensaje: **No echos yet**.
- Si el usuario en cuestión no existe, se debe devolver una respuesta 404.

/users/guido/echos/ ⇒ echos.views.user\_echos()

- Perfil del usuario con nombre de usuario **guido**.
- Todo es igual que para /users/guido/ salvo que:
  - Se mostrarán **todos los “echos”** del usuario independientemente de cuántos existan.
  - **No** debe existir un enlace para ver todos los “echos”.

/users/@me/ ⇒ users.views.my\_user\_detail()

- Perfil del usuario logeado.
- Es una simple redirección a /users/guido/ (*como ejemplo*)

/users/guido/edit/ ⇒ users.views.edit\_profile()

- Editar el perfil de usuario.
- Requiere estar autenticado.
- Formulario con campos de biografía y “avatar”.
- Si el usuario logeado no corresponde con el perfil a editar, se debe devolver una respuesta 403.
- Si el usuario en cuestión no existe, se debe devolver una respuesta 404.
- Si la operación fue exitosa, se debe redirigir a /users/@me/
- Si la operación fue exitosa, se debe mostrar el mensaje: **Profile updated successfully.**

## 7. Administración

Los siguientes modelos deben estar accesibles desde la **interfaz administrativa** de Django:

- echos.Echo
- waves.Wave
- users.Profile