Guia completa de Flet 0.85.1

Desarrollo de aplicaciones graficas, web y de escritorio con Python

Incluye instalacion, estructura de proyecto, propiedades, controles, codigo de ejemplo, pruebas, patrones de navegacion, temas, estado, formularios, tablas, empaquetado y un laboratorio interactivo para modificar propiedades en vivo.

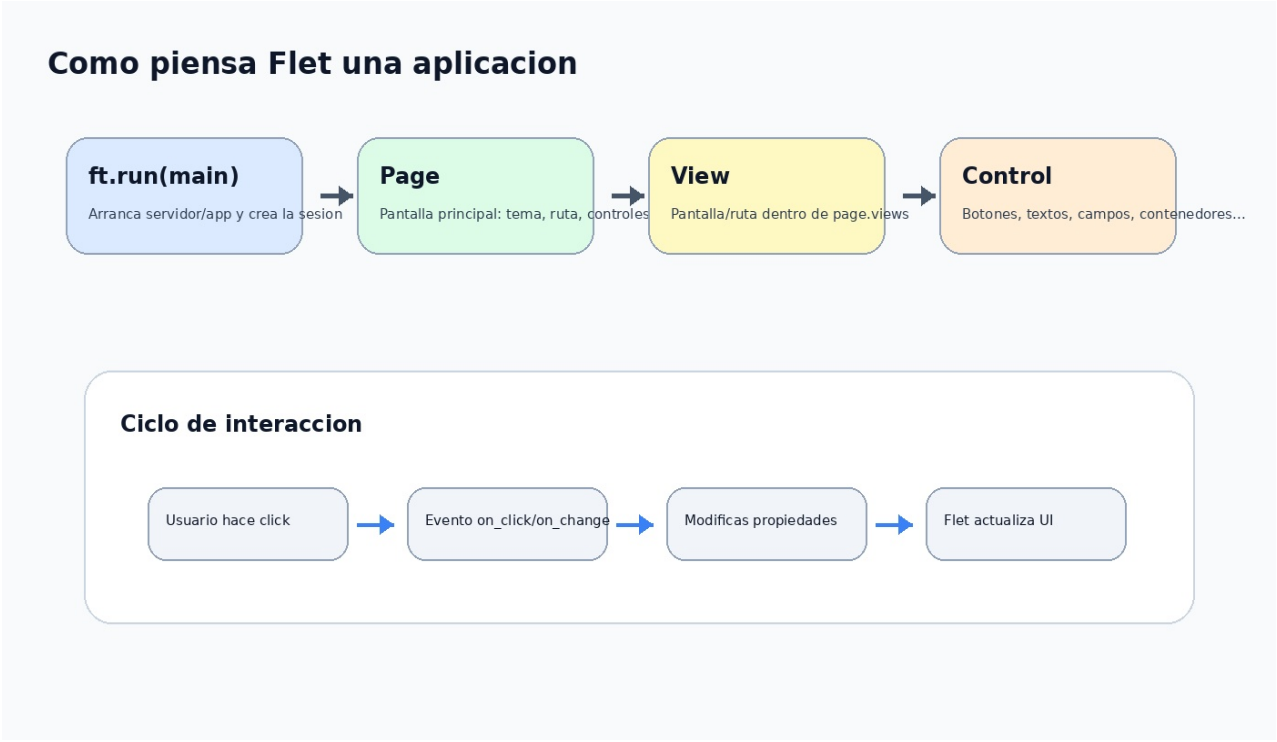

Figura 1. Vista conceptual: Page, View, Control y ciclo de eventos.

Version de referencia: Flet 0.85.1. La version fue verificada en PyPI, donde aparece como publicada el 13 de mayo de 2026. La API de Flet cambia con frecuencia; por eso conviene revisar la documentacion oficial cuando se trabaje en produccion.

Tabla de contenido

1. Que es Flet y como se piensa una app

Flet es un framework para construir interfaces graficas con Python. La idea central es escribir controles en Python y dejar que Flet renderice la interfaz usando tecnologia basada en Flutter. Con Flet puedes crear aplicaciones de escritorio, aplicaciones web y proyectos empaquetables para otras plataformas, sin escribir HTML, CSS, JavaScript o Dart para la interfaz basica.

La unidad mental mas importante es esta: una aplicacion Flet se compone de una funcion main, una Page, una o varias View y muchos controles. Los controles son objetos como Text, Button, TextField, Container, Row, Column, DataTable o NavigationBar. Tu programa modifica propiedades de esos controles y Flet sincroniza esos cambios con la interfaz.

```
import flet as ft


def main(page: ft.Page):
    page.title = "Mi primera app"
    page.add(ft.Text("Hola, Flet", size=24))


ft.run(main)
```

Idea clave: no dibujas pixeles manualmente. Construyes un arbol de controles. Page contiene controles o views; los layouts organizan controles hijos; cada control tiene propiedades visuales, funcionales y eventos.

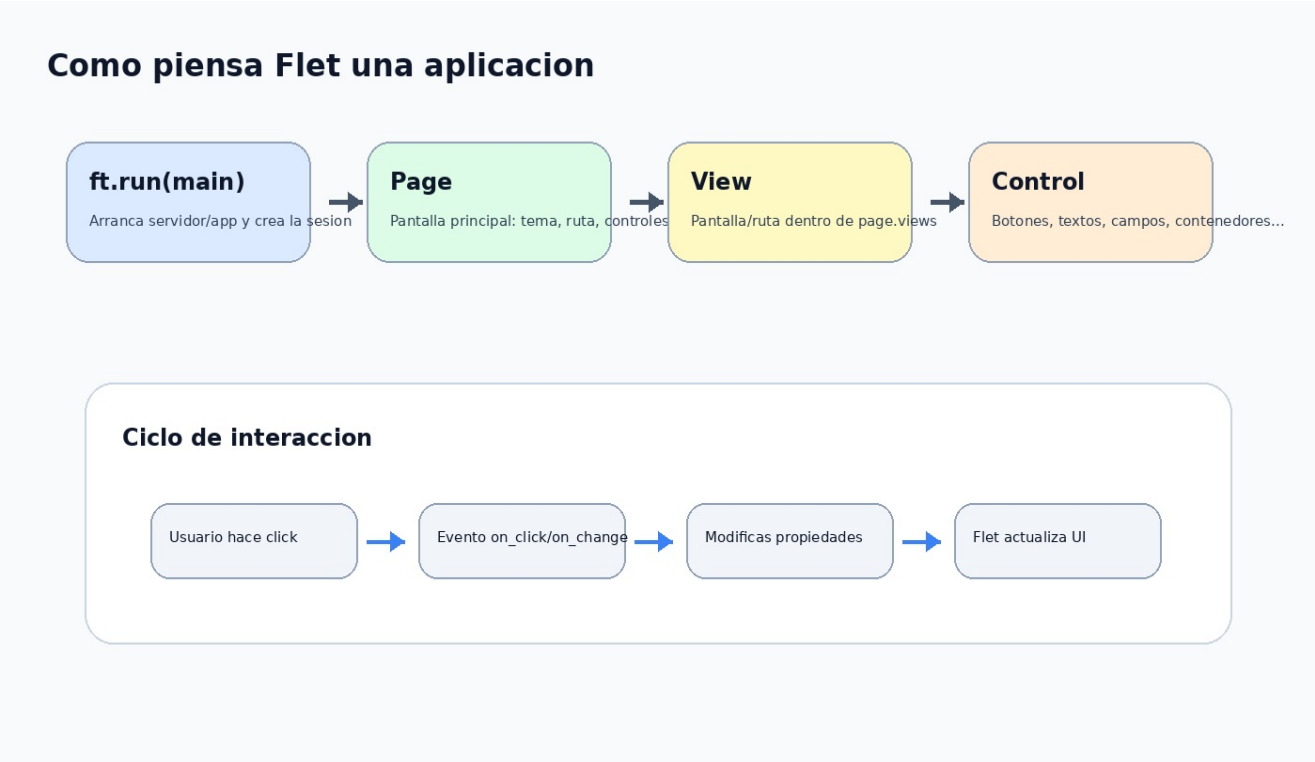

Figura 2. Arquitectura didactica de una app Flet.

Cuando usar Flet

Caso	Por que Flet sirve
Aplicaciones academicas	Permite construir herramientas visuales con Python sin frontend complejo.
Dashboards pequenos o medianos	Combina tablas, graficos, formularios, filtros y navegacion.
Aplicaciones CRUD	Puedes conectar bases de datos, crear formularios y listar registros.
Prototipos rapidos	La sintaxis es corta y el hot reload acelera pruebas.

Caso	Por que Flet sirve
Apps multiplataforma	Un mismo codigo puede ejecutarse como escritorio o web con comandos diferentes.

2. Instalacion actualizada y primeros comandos

Flet 0.85.1 requiere Python moderno. La forma mas ordenada de trabajar es usar un entorno virtual por proyecto. La documentacion actual recomienda flujos con uv, aunque pip tambien funciona.

Opcion A: instalacion con uv

```
mkdir mi_app_flet
cd mi_app_flet
uv init --python=">=3.10"
uv venv

# Windows
.venv\Scripts□ctivate

# Linux / macOS
source .venv/bin/activate

uv add "flet[all]"
uv run flet --version
uv run flet doctor
```

Opcion B: instalacion con pip

```
mkdir mi_app_flet
cd mi_app_flet
python -m venv .venv

# Windows
.venv\Scripts□ctivate

# Linux / macOS
source .venv/bin/activate

pip install "flet[all]"
flet --version
flet doctor
```

Crear una app nueva

El comando flet create crea una estructura basica con README.md, pyproject.toml, src/assets/icon.png, src/main.py y carpetas de storage. Si ya existen README.md o pyproject.toml, el comando puede reemplazarlos, asi que conviene crear el proyecto en una carpeta limpia.

```
# Con uv
uv run flet create

# Con pip o comando global
flet create
```

Ejecutar como escritorio o web

```
# Escritorio
flet run
flet run src/main.py

# Web
flet run --web
flet run --web src/main.py

# Vigilar cambios en carpetas y subcarpetas
flet run --recursive src/main.py
```

Flet ejecuta aplicaciones como escritorio o web con el mismo codigo. El modo web se activa con --web. Durante desarrollo, flet run incluye hot reload para ver cambios rapidamente.

3. Anatomia recomendada de un proyecto Flet

Para proyectos pequenos puedes trabajar con un solo main.py. Para proyectos reales conviene separar vistas, componentes, servicios, modelos y utilidades. Esta separacion evita que main.py se vuelva inmanejable.

```
mi_app/
  pyproject.toml
  README.md
  src/
    main.py
    assets/
      icon.png
      logo.png
    views/
      home_view.py
      login_view.py
      dashboard_view.py
    components/
      app_bar.py
      cards.py
      form_fields.py
      navigation.py
    services/
      api_service.py
      database_service.py
      auth_service.py
    models/
      user.py
      product.py
    utils/
      validators.py
      constants.py
  storage/
    data/
    temp/
```

Regla practica de organizacion

Carpeta	Que guardar ahi
views	Pantallas completas: login, home, dashboard, settings.
components	Piezas reutilizables: tarjetas, barras, formularios, botones personalizados.
services	Conexion con APIs, base de datos, archivos, autenticacion.
models	Clases o dataclasses que representan datos: User, Product, Task.
utils	Funciones generales: validadores, formateadores, constantes.
assets	Imagenes, iconos, fuentes y archivos estaticos.

4. Page, View y Control: la base de todo

Page

Page es el contenedor principal de la sesion. Desde page configuras titulo, tema, alineacion, scroll, rutas, vistas, overlay, barras de navegacion y acciones globales. En la documentacion oficial actual, Page se describe como un contenedor de controles View; al iniciar una sesion se crea automaticamente una Page y una vista raiz.

```
def main(page: ft.Page):
    page.title = "Sistema academico"
    page.theme_mode = ft.ThemeMode.LIGHT
    page.padding = 20
    page.spacing = 12
    page.scroll = ft.ScrollMode.AUTO
    page.horizontal_alignment = ft.CrossAxisAlignment.CENTER
    page.vertical_alignment = ft.MainAxisAlignment.START

    page.add(ft.Text("Bienvenido"))
```

Propiedad / metodo	Uso principal
title	Titulo de la ventana o pestana.
theme_mode	LIGHT, DARK o SYSTEM.
theme / dark_theme	Temas personalizados para modo claro y oscuro.
padding / spacing	Espaciado general de la pagina.
horizontal_alignment / vertical_alignment	Alineacion de controles dentro de Page.
scroll	Activa desplazamiento si el contenido no cabe.
route	Ruta actual de la aplicacion.
views	Pila de pantallas View usadas en navegacion.
appbar, navigation_bar, drawer	Elementos globales de navegacion.
overlay	Lista de controles flotantes como dialogos y pickers.
add(), clean(), update()	Agregar, limpiar y actualizar controles.
open(), close()	Mostrar o cerrar dialogos, snackbars o sheets.
navigate(), push_route()	Cambiar de ruta de forma sincronizada.

View

View representa una pantalla o ruta. Una app de varias pantallas suele reconstruir page.views cuando cambia page.route. La ultima View de page.views es la que se muestra.

```
ft.View(
    route="/login",
    controls=[
        ft.AppBar(title=ft.Text("Login")),
        ft.TextField(label="Usuario"),
        ft.TextField(label="Contrasena", password=True),
        ft.Button("Entrar"),
    ],
)
```

Control

Control es la base conceptual de todo lo visible o interactivo. Text, Button, TextField, Container, Row, Column y DataTable son controles. Los controles tienen propiedades comunes como visible, disabled, expand, opacity y tooltip, y propiedades especificas segun su funcion.

5. Propiedades comunes de los controles

Muchas propiedades se repiten en varios controles. Aprenderlas reduce muchisimo la curva de aprendizaje, porque una vez entiendes width, height, expand, visible, disabled, tooltip, opacity y eventos, puedes aplicar la misma logica en muchos controles.

Propiedad	Tipo aproximado	Que hace	Ejemplo
visible	bool	Muestra u oculta el control.	visible=False

Propiedad	Tipo aproximado	Que hace	Ejemplo
disabled	bool	Desactiva el control y, en contenedores, puede propagarse a hijos.	disabled=True
expand	bool/int	Hace que el control ocupe espacio disponible en Row, Column, View o Page.	expand=True
expand_loose	bool	Permite expandir sin obligar a llenar todo el espacio.	expand_loose=True
width / height	number	Define ancho y alto.	width=300
opacity	number	Transparencia entre 0 y 1.	opacity=0.5
tooltip	str	Texto de ayuda al pasar el mouse.	tooltip="Guardar"
col	dict/int	Columnas en ResponsiveRow.	col={"sm":12,"md":6}
badge	Badge	Insignia encima del control.	badge="3"
rtl	bool	Direccion de texto derecha a izquierda.	rtl=True

Propiedades visuales frecuentes

Propiedad	Controles comunes	Uso
bgcolor	Container, Button, DataTable, AppBar	Color de fondo.
color	Text, Icon, Button	Color de texto, icono o contenido.
padding	Container, Page, View	Espacio interno.
margin	Container	Espacio externo.
border	Container, DataTable	Borde alrededor.
border_radius	Container, Card, Image, DataTable	Bordes redondeados.
shadow	Container, Card	Sombra visual.
gradient	Container, DataTable	Fondo degradado.
alignment	Container, Row, Column	Alinea contenido o hijos.
animate / animate_*	Container y controles animables	Transiciones suaves al cambiar propiedades.

Eventos frecuentes

Evento	Cuando ocurre	Controles tipicos
on_click	Click o toque.	Button, IconButton, Container, ListTile
on_change	Cambio de valor.	TextField, Checkbox, Switch, Dropdown, Slider
on_submit	Enter o envio desde campo.	TextField
on_focus / on_blur	Entrar o salir del foco.	TextField y controles focalizables
on_hover	Mouse entra o sale.	Container, Button
on_long_press	Pulsacion larga.	Botones, ListTile, Container
on_select_all	Seleccionar todos.	DataTable
on_route_change	Cambio de ruta.	Page
on_view_pop	Boton atras o cierre de vista.	Page/View

```
contador = ft.Text("0")


def sumar(e):
    contador.value = str(int(contador.value) + 1)
    # En Flet actual muchas veces se actualiza al terminar el evento.
    # page.update() sigue siendo util en callbacks externos o procesos async.

ft.Button("Sumar", on_click=sumar)
```

6. Layout: organizar la pantalla

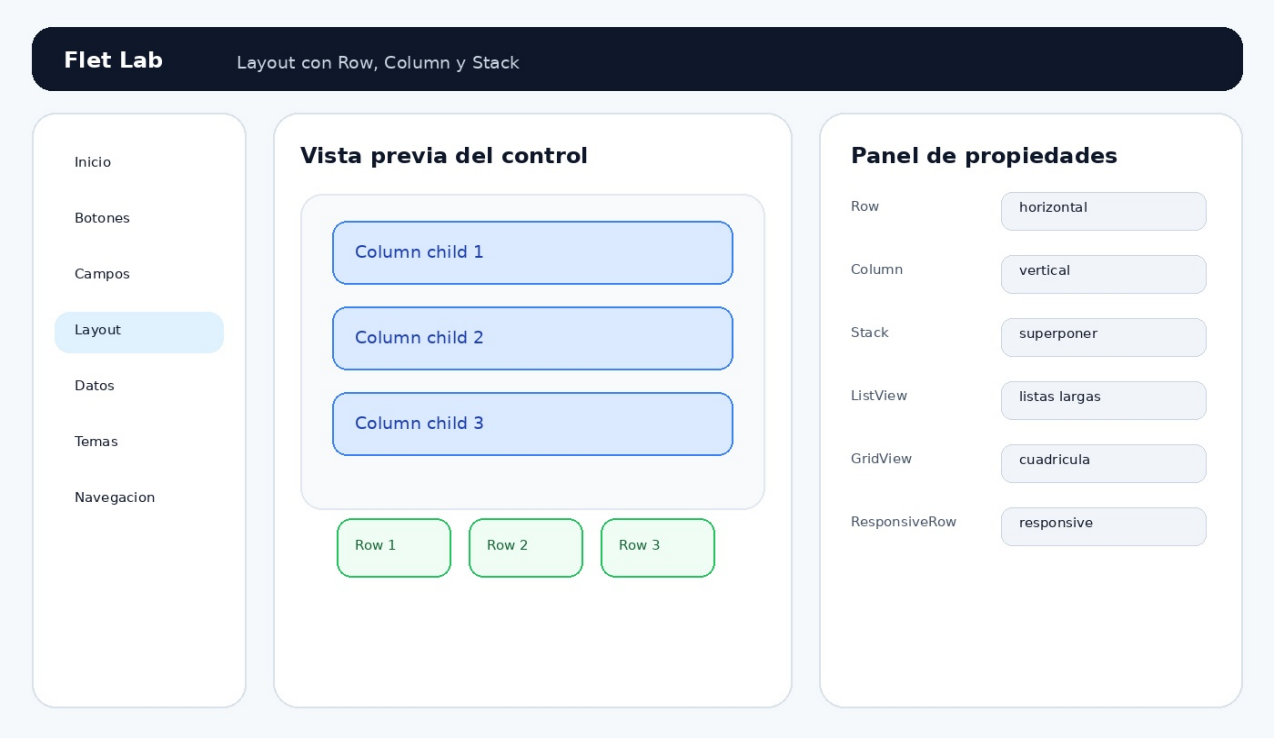

Figura 3. Resultado visual esperado de un laboratorio de layout.

Column

Column organiza controles verticalmente. Es el layout mas comun para formularios, secciones y pantallas completas.

```
ft.Column(
    spacing=12,
    horizontal_alignment=ft.CrossAxisAlignment.CENTER,
    controls=[
        ft.Text("Registro", size=24, weight=ft.FontWeight.BOLD),
        ft.TextField(label="Nombre"),
        ft.Button("Guardar"),
    ],
)
```

Propiedad	Uso
controls	Lista de hijos en orden vertical.
spacing	Espacio entre hijos.
alignment	Alineacion vertical de los hijos.
horizontal_alignment	Alineacion horizontal dentro de la columna.
scroll	Permite desplazamiento en contenido largo.
wrap, run_spacing, run_alignment	Permiten organizar hijos en multiples corridas si aplica.
intrinsic_width	Hace que la columna mida segun el hijo mas ancho.

Row

Row organiza controles horizontalmente. Es ideal para botones, barras de acciones, filtros y elementos que van uno al lado del otro.

```
ft.Row(
    alignment=ft.MainAxisAlignment.CENTER,
    vertical_alignment=ft.CrossAxisAlignment.CENTER,
    spacing=10,
    controls=[
        ft.Button("Cancelar"),
        ft.FilledButton("Guardar"),
    ],
)
```

Propiedad	Uso
controls	Lista de hijos en orden horizontal.
alignment	Distribucion horizontal: START, CENTER, END, SPACE_BETWEEN, etc.
vertical_alignment	Alineacion vertical de los hijos.
spacing	Espacio horizontal entre controles.
wrap	Permite que los controles salten a otra linea si no caben.
run_spacing / run_alignment	Espaciado y alineacion de filas cuando wrap=True.
intrinsic_height	Ajusta altura segun el hijo mas alto.

Container

Container permite decorar y posicionar un unico control hijo. Es fundamental para crear tarjetas, cajas, fondos, separaciones, bordes, areas clicables y animaciones.

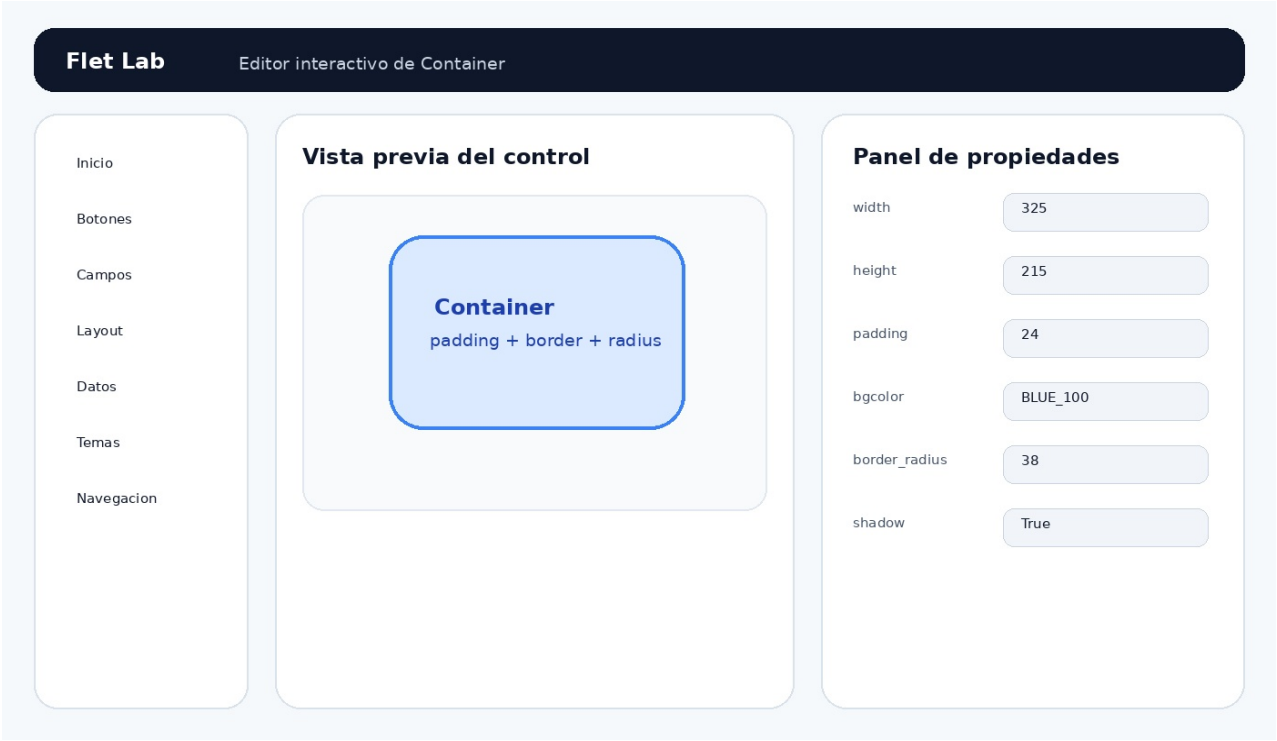

Figura 4. Propiedades visuales de Container.

```
ft.Container(
    content=ft.Text("Tarjeta", size=20),
    width=320,
    height=160,
    padding=24,
    margin=10,
    alignment=ft.Alignment.CENTER,
    bgcolor=ft.Colors.BLUE_100,
    border=ft.border.all(2, ft.Colors.BLUE),
    border_radius=24,
    shadow=ft.BoxShadow(blur_radius=12, spread_radius=1),
    on_click=lambda e: print("click en container"),
)
```

Propiedad	Uso
content	Control hijo unico. Si necesitas varios, usa Column/Row dentro.
width, height	Tamano fijo del contenedor.
padding, margin	Espacio interno y externo.
alignment	Alineacion del content dentro del contenedor.
bgcolor, gradient, image	Fondo solido, degradado o imagen.
border, border_radius	Bordes y esquinas redondeadas.
shadow	Sombra para efecto de tarjeta.
ink	Efecto visual tipo material al hacer click.
animate	Animacion implicita al cambiar valores.
blur, blend_mode, color_filter	Efectos visuales avanzados.

Stack

Stack superpone controles. Es util para portadas, badges, etiquetas encima de imagenes, capas y disenos personalizados.

```
ft.Stack(
    controls=[
        ft.Image(src="fondo.jpg", width=400, height=240, fit=ft.ImageFit.COVER),
        ft.Container(left=20, top=20, content=ft.Text("Nuevo"), bgcolor=ft.Colors.RED),
    ]
)
```

ListView, GridView y ResponsiveRow

ListView sirve para listas largas y con scroll eficiente. GridView organiza elementos en cuadricula. ResponsiveRow permite adaptar la pantalla por columnas segun el ancho, parecido a un sistema responsive de 12 columnas.

```
ft.ResponsiveRow(
    controls=[
        ft.Container(content=ft.Text("Card 1"), col={"sm": 12, "md": 6, "lg": 4}),
        ft.Container(content=ft.Text("Card 2"), col={"sm": 12, "md": 6, "lg": 8}),
    ]
)
```

7. Texto, iconos, imagenes y contenido visual

Text

```
ft.Text(
    "Resultado final",
    size=28,
    weight=ft.FontWeight.BOLD,
    color=ft.Colors.BLUE,
    italic=False,
    text_align=ft.TextAlign.CENTER,
    max_lines=2,
    overflow=ft.TextOverflow.ELLIPSIS,
    selectable=True,
)
```

Propiedad	Uso
value	Texto que se muestra.
size	Tamano de fuente.
weight	Grosor: NORMAL, BOLD, W_600, etc.
color	Color del texto.
italic	Cursiva.
font_family	Fuente especifica.
text_align	Alineacion horizontal.
max_lines	Limite de lineas.
overflow	Comportamiento si el texto no cabe.
selectable	Permite seleccionar el texto.

Markdown

```
ft.Markdown(
    """
# Ayuda
Texto en **negrita**, enlaces, listas y codigo.

- Paso 1
- Paso 2
    """
)
```

Icon e Image

```
ft.Icon(name=ft.Icons.SAVE, color=ft.Colors.BLUE, size=32)

ft.Image(
    src="logo.png",
    width=180,
    height=120,
    fit=ft.ImageFit.CONTAIN,
    border_radius=12,
)
```

Control	Propiedades importantes
Icon	name, color, size, tooltip.
Image	src, src_base64, width, height, fit, repeat, border_radius, opacity.
Video	src, autoplay, controls, volume, playlist, fit.
WebView	url, html, javascript, navigation events.
Lottie / Rive	Animaciones vectoriales para interfaces mas vivas.
Canvas	Dibujo personalizado con formas y coordenadas.

8. Botones y acciones

Los botones ejecutan acciones. En Flet moderno puedes usar Button como boton material general, ademas de variantes como TextButton, FilledButton, FilledTonalButton, OutlinedButton, IconButton, FloatingActionButton y PopupMenuButton segun el estilo.

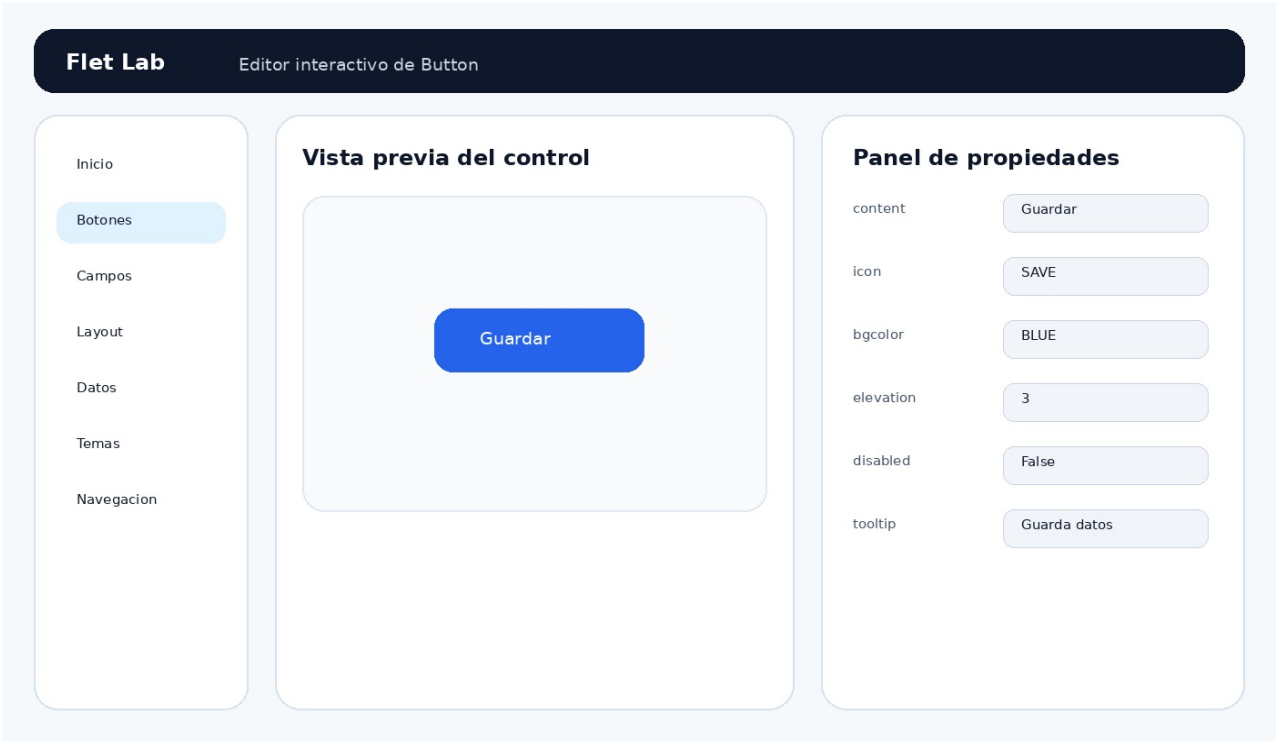

Figura 5. Laboratorio de propiedades de un Button.

```
ft.Button(
    content="Guardar",
    icon=ft.Icons.SAVE,
    bgcolor=ft.Colors.BLUE,
    color=ft.Colors.WHITE,
    elevation=3,
    tooltip="Guardar cambios",
    on_click=guardar,
)
```

Propiedad	Descripcion
content	Texto o control dentro del boton.
icon	Icono del boton.
icon_color	Color del icono.
bgcolor	Color de fondo.
color	Color del contenido.
elevation	Elevacion/sombra material.
style	ButtonStyle para estados, formas, colores y padding.
url	Abre una URL al hacer click.
autofocus	El boton recibe foco al cargar.
disabled	Desactiva interaccion.
on_click	Funcion ejecutada al presionar.

Variantes de botones

```
ft.TextButton("Cancelar")
ft.FilledButton("Continuar")
ft.FilledTonalButton("Mas suave")
ft.OutlinedButton("Ver detalles")
ft.IconButton(icon=ft.Icons.DELETE, tooltip="Eliminar")
ft.FloatingActionButton(icon=ft.Icons.ADD, on_click=agregar)
```

Regla visual: usa FilledButton para la accion principal, OutlinedButton para una accion secundaria, TextButton para acciones ligeras, IconButton para acciones compactas y FloatingActionButton para crear/agregar en pantallas moviles o dashboards.

9. Entradas de datos y formularios

TextField

TextField permite introducir texto, numeros, contrasenas, mensajes largos y datos de formulario. Es uno de los controles que mas propiedades tiene.

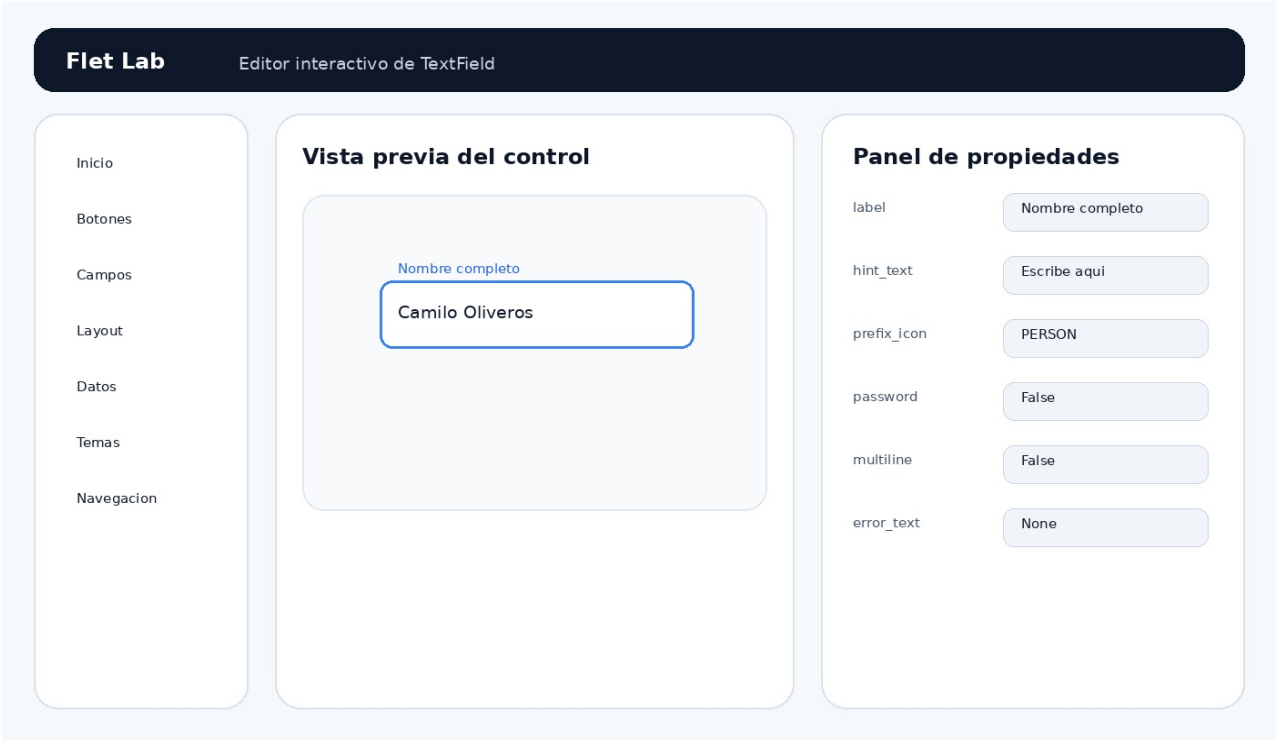

Figura 6. Laboratorio de propiedades de TextField.

```
nombre = ft.TextField(
    label="Nombre completo",
    hint_text="Escribe tu nombre",
    prefix_icon=ft.Icons.PERSON,
    autofocus=True,
    max_length=60,
    keyboard_type=ft.KeyboardType.TEXT,
)

password = ft.TextField(
    label="Contrasena",
    password=True,
    can_reveal_password=True,
)
```

Propiedad	Uso
label	Etiqueta visible del campo.
hint_text	Texto de ayuda dentro del campo.
value	Valor actual del campo.
password	Oculta el texto escrito.

Propiedad	Uso
can_reveal_password	Muestra icono para revelar contrasena.
multiline	Permite varias lineas.
min_lines / max_lines	Controla altura de campos multilinea.
max_length	Limita caracteres.
keyboard_type	Tipo de teclado: text, number, email, phone, url.
read_only	Permite ver pero no editar.
disabled	Desactiva el campo.
prefix / suffix	Texto o control antes/despues del valor.
prefix_icon / suffix_icon	Iconos decorativos o funcionales.
error_text	Mensaje de error debajo del campo.
helper_text	Ayuda permanente.
autofocus	Foco automatico al cargar.
autocorrect / capitalization	Correccion y capitalizacion.
cursor_color	Color del cursor.
on_change	Evento al cambiar texto.
on_submit	Evento al presionar Enter.

Validacion basica de formulario

```
import flet as ft


def main(page: ft.Page):
    nombre = ft.TextField(label="Nombre")
    correo = ft.TextField(label="Correo")
    salida = ft.Text()

    def guardar(e):
        ok = True
        if not nombre.value.strip():
            nombre.error_text = "El nombre es obligatorio"
            ok = False
        else:
            nombre.error_text = None

        if "@" not in correo.value:
            correo.error_text = "Correo invalido"
            ok = False
        else:
            correo.error_text = None

        salida.value = "Guardado" if ok else "Corrige los errores"
        salida.color = ft.Colors.GREEN if ok else ft.Colors.RED

    page.add(nombre, correo, ft.FilledButton("Guardar", on_click=guardar), salida)

ft.run(main)
```

Controles de seleccion y entrada

Control	Uso	Propiedades clave
Checkbox	Aceptar condiciones o seleccionar opciones multiples.	label, value, tristate, on_change
Switch	Activar/desactivar configuraciones.	label, value, active_color, on_change
RadioGroup + Radio	Elegir una sola opcion.	value, content, label
Dropdown	Elegir en una lista desplegable.	label, value, options, editable, on_change

Control	Uso	Propiedades clave
Slider	Seleccionar un valor numerico en rango.	min, max, divisions, label, value
RangeSlider	Seleccionar intervalo de valores.	start_value, end_value, min, max
DatePicker	Seleccionar fecha.	first_date, last_date, value, on_change
TimePicker	Seleccionar hora.	value, on_change
AutoComplete	Sugerencias al escribir.	suggestions, on_select
SearchBar	Busqueda con sugerencias.	controls, view_hint_text, on_change
SegmentedButton	Opciones tipo pestañas/botones segmentados.	segments, selected, allow_multiple_selection

```
ft.Dropdown(
    label="Ciudad",
    options=[
        ft.dropdown.Option("Popayan"),
        ft.dropdown.Option("Cali"),
        ft.dropdown.Option("Bogota"),
    ],
    on_change=lambda e: print(e.control.value),
)

ft.Slider(min=0, max=100, divisions=10, label="{value}")
ft.Checkbox(label="Acepto terminos", value=False)
ft.Switch(label="Modo oscuro", value=True)
```

10. Navegacion: rutas, AppBar, NavigationBar, Drawer y View stack

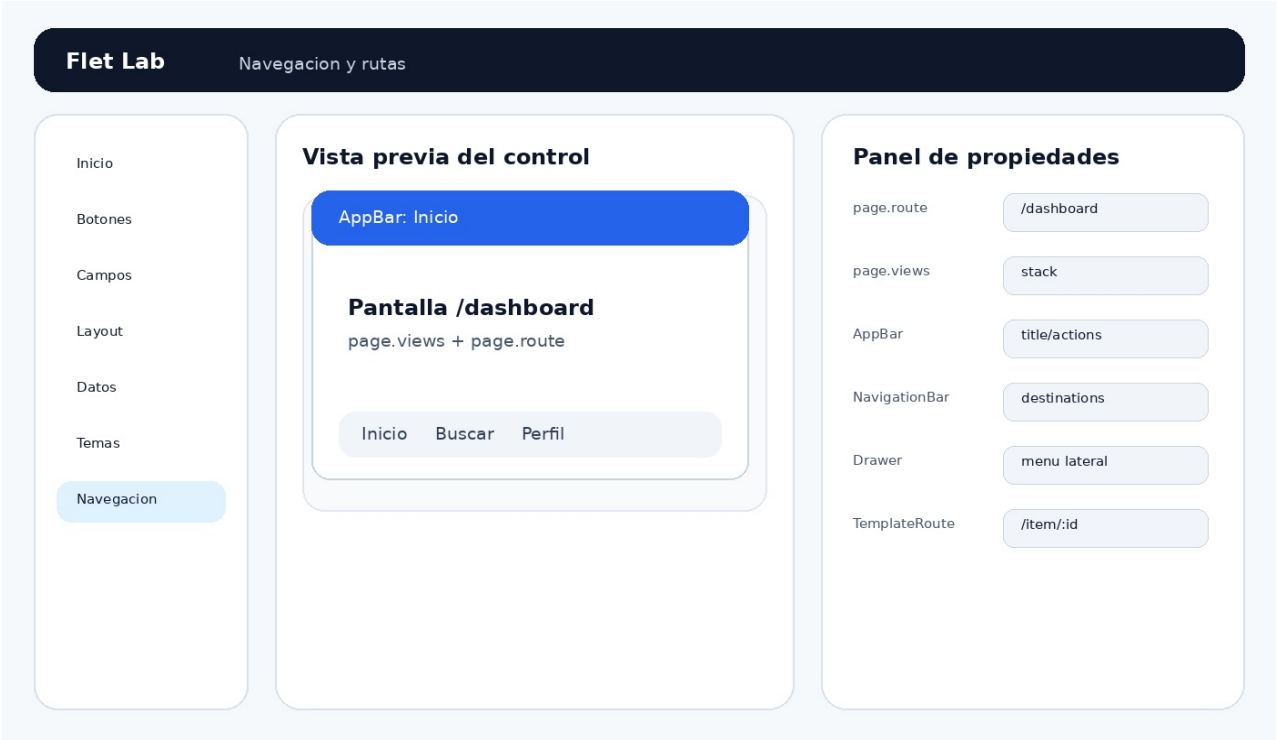

Figura 7. Navegacion con rutas, AppBar y barra inferior.

La navegacion actual de Flet se basa en rutas y en una pila de vistas. page.route indica la ruta actual; page.views contiene el historial visual; page.on_route_change reconstruye la pila; page.on_view_pop maneja atras. Para que sea confiable, usa una sola fuente de verdad: construir page.views desde page.route.

```
import flet as ft


def main(page: ft.Page):
    page.title = "App con rutas"

    def route_change(e=None):
        page.views.clear()
        page.views.append(
            ft.View(
                route="/",
                controls=[
                    ft.AppBar(title=ft.Text("Inicio")),
                    ft.Text("Pantalla principal"),
                    ft.Button("Ir a ajustes", on_click=lambda e: page.navigate("/settings")),
                ],
            )
        )

        if page.route == "/settings":
            page.views.append(
                ft.View(
                    route="/settings",
                    controls=[
                        ft.AppBar(title=ft.Text("Ajustes")),
                        ft.Text("Configuracion de la app"),
                    ],
                )
            )
        page.update()

    async def view_pop(e):
        if e.view is not None:
            page.views.remove(e.view)
            top_view = page.views[-1]
            await page.push_route(top_view.route)

    page.on_route_change = route_change
    page.on_view_pop = view_pop
    route_change()

ft.run(main)
```

Control/concepto	Uso
AppBar	Barra superior con titulo, acciones y boton atras/menu.
BottomAppBar	Barra inferior material para acciones.
NavigationBar	Navegacion inferior, ideal para 3-5 destinos.
NavigationRail	Navegacion lateral compacta para escritorio/tablet.
NavigationDrawer	Menu lateral desplegable.
View	Pantalla asociada a una ruta.
TemplateRoute	Permite rutas con parametros como /books/:id.
page.navigate()	Navegacion desde callbacks normales.
await page.push_route()	Navegacion dentro de funciones async.

11. Feedback: Dialogos, SnackBar, BottomSheet, Banner y progreso

Los controles de feedback informan al usuario de errores, confirmaciones, cargas o decisiones. Se muestran normalmente con page.open(control) y se cierran con page.close(control).

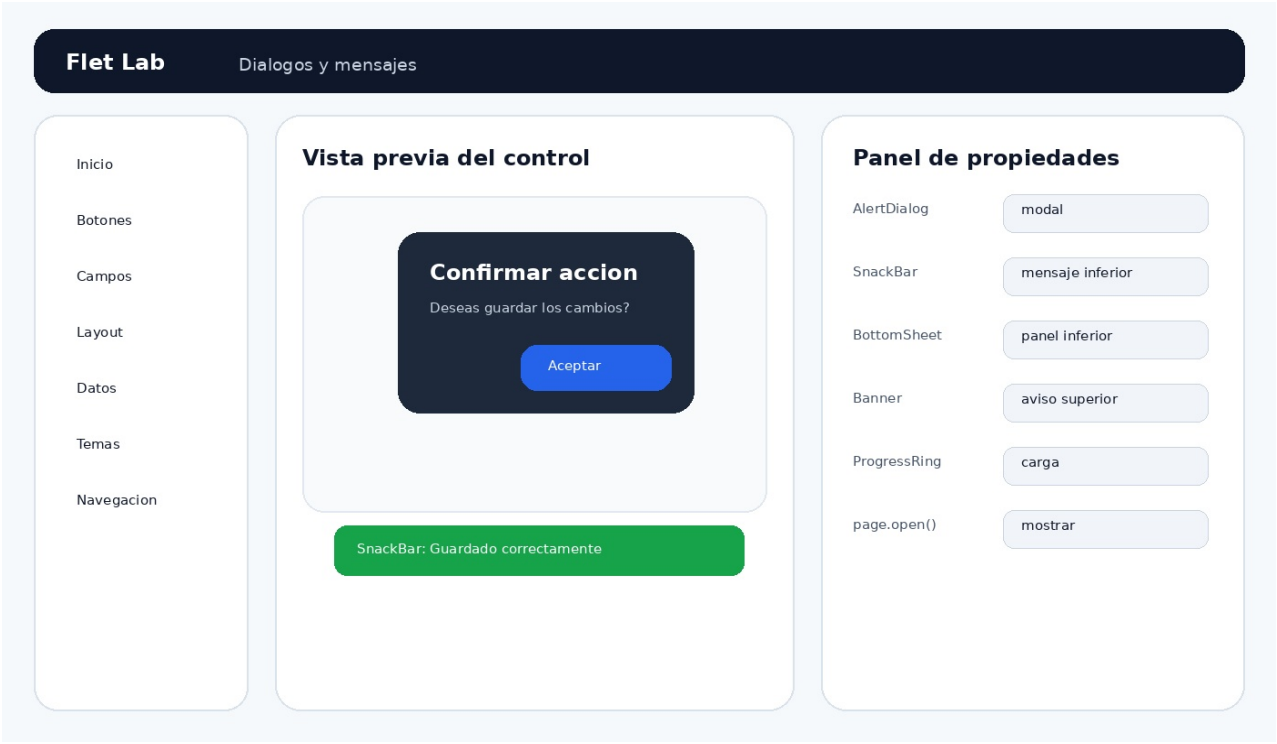

Figura 8. Dialogo modal y SnackBar.

```
def main(page: ft.Page):
    dialog = ft.AlertDialog(
        title=ft.Text("Confirmar"),
        content=ft.Text("Deseas guardar los cambios?"),
        actions=[
            ft.TextButton("Cancelar", on_click=lambda e: page.close(dialog)),
            ft.FilledButton("Guardar", on_click=lambda e: page.close(dialog)),
        ],
    )

    page.add(ft.Button("Abrir dialogo", on_click=lambda e: page.open(dialog)))
```

Control	Uso	Propiedades clave
AlertDialog	Confirmaciones, errores importantes o decisiones.	title, content, actions, modal
SnackBar	Mensaje temporal inferior.	content, action, duration, bgcolor
BottomSheet	Panel inferior con informacion o formulario.	content, open, dismissible
Banner	Aviso persistente superior.	content, actions, leading
ProgressBar	Progreso lineal.	value, color, bgcolor
ProgressRing	Carga circular.	value, stroke_width, color

12. Datos: tablas, listas, tarjetas y graficos

Card y ListTile

```
ft.Card(
    content=ft.Container(
        padding=16,
        content=ft.Column([
            ft.Text("Producto", size=20, weight=ft.FontWeight.BOLD),
            ft.Text("Precio: 120.000"),
        ]),
    )
)

ft.ListTile(
    leading=ft.Icon(ft.Icons.PERSON),
    title=ft.Text("Ana Perez"),
    subtitle=ft.Text("Estudiante activa"),
    trailing=ft.Icon(ft.Icons.CHEVRON_RIGHT),
    on_click=lambda e: print("abrir detalle"),
)
```

DataTable

DataTable muestra datos tabulares con columnas, filas y celdas. Sirve para inventarios, listados, resultados y reportes. Para tablas muy grandes puede convenir una extension especializada o paginacion propia.

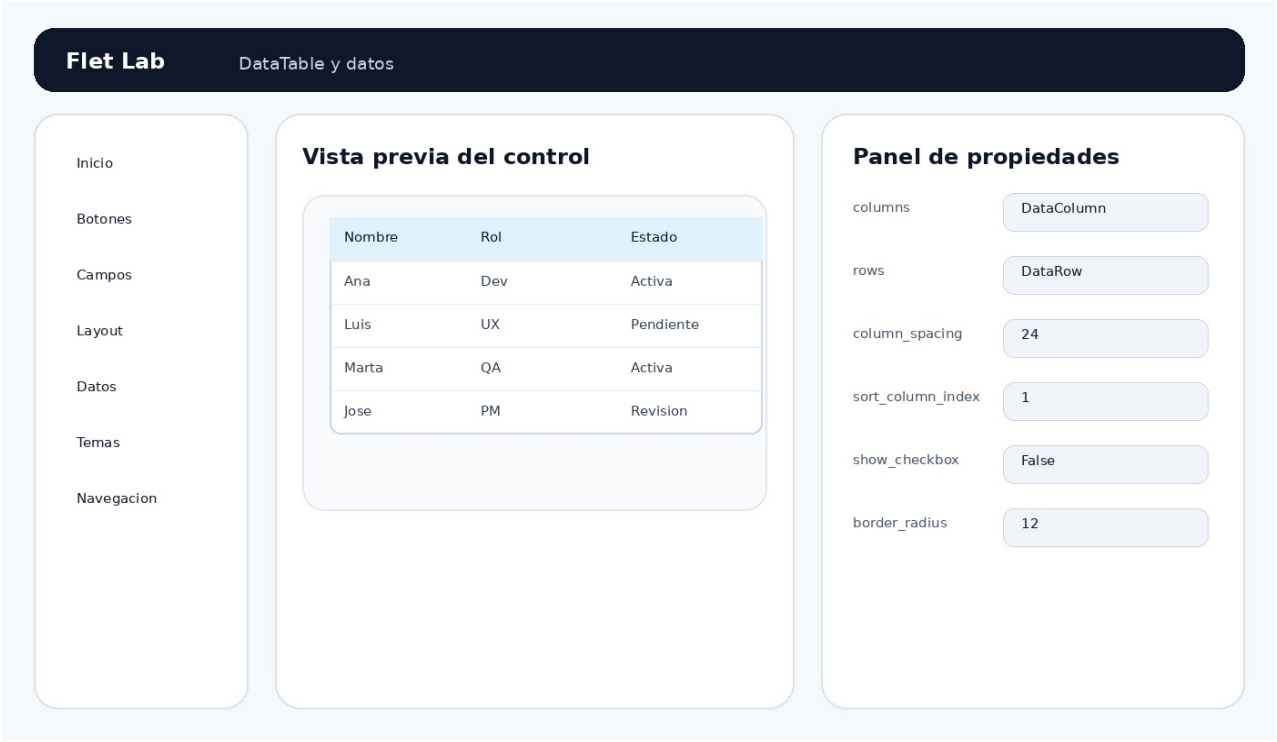

Figura 9. Tabla de datos con propiedades editables.

```
ft.DataTable(
    border=ft.border.all(1, ft.Colors.GREY_300),
    border_radius=10,
    columns=[
        ft.DataColumn(label=ft.Text("Nombre")),
        ft.DataColumn(label=ft.Text("Rol")),
        ft.DataColumn(label=ft.Text("Estado")),
    ],
    rows=[
        ft.DataRow(cells=[
            ft.DataCell(ft.Text("Ana")),
            ft.DataCell(ft.Text("Dev")),
            ft.DataCell(ft.Text("Activa")),
        ]),
        ft.DataRow(cells=[
            ft.DataCell(ft.Text("Luis")),
            ft.DataCell(ft.Text("UX")),
            ft.DataCell(ft.Text("Pendiente")),
        ]),
    ],
)
```

Elemento	Propiedades importantes
DataTable	columns, rows, bgcolor, border, border_radius, column_spacing, heading_row_color, data_row_color, sort_column_index, sort_ascending, show_checkbox_column, horizontal_lines, vertical_lines.
DataColumn	label, numeric, tooltip, on_sort, heading_row_alignment.
DataRow	cells, selected, color, on_select_changed.
DataCell	content, placeholder, show_edit_icon, on_tap.

Graficos y dashboards

Flet incluye controles de graficos como BarChart, LineChart y PieChart. Son utiles para dashboards de ventas, notas, costos, indicadores, conteos o series temporales. Para analisis mas pesado puedes calcular datos con pandas/numpy y mostrar resultados en Flet.

```
ft.BarChart(
    bar_groups=[
        ft.BarChartGroup(x=0, bar_rods=[ft.BarChartRod(from_y=0, to_y=40)]),
        ft.BarChartGroup(x=1, bar_rods=[ft.BarChartRod(from_y=0, to_y=65)]),
    ],
    left_axis=ft.ChartAxis(labels_size=40),
    bottom_axis=ft.ChartAxis(labels=[]),
)
```

13. Temas, colores, fuentes, estilos y modo oscuro

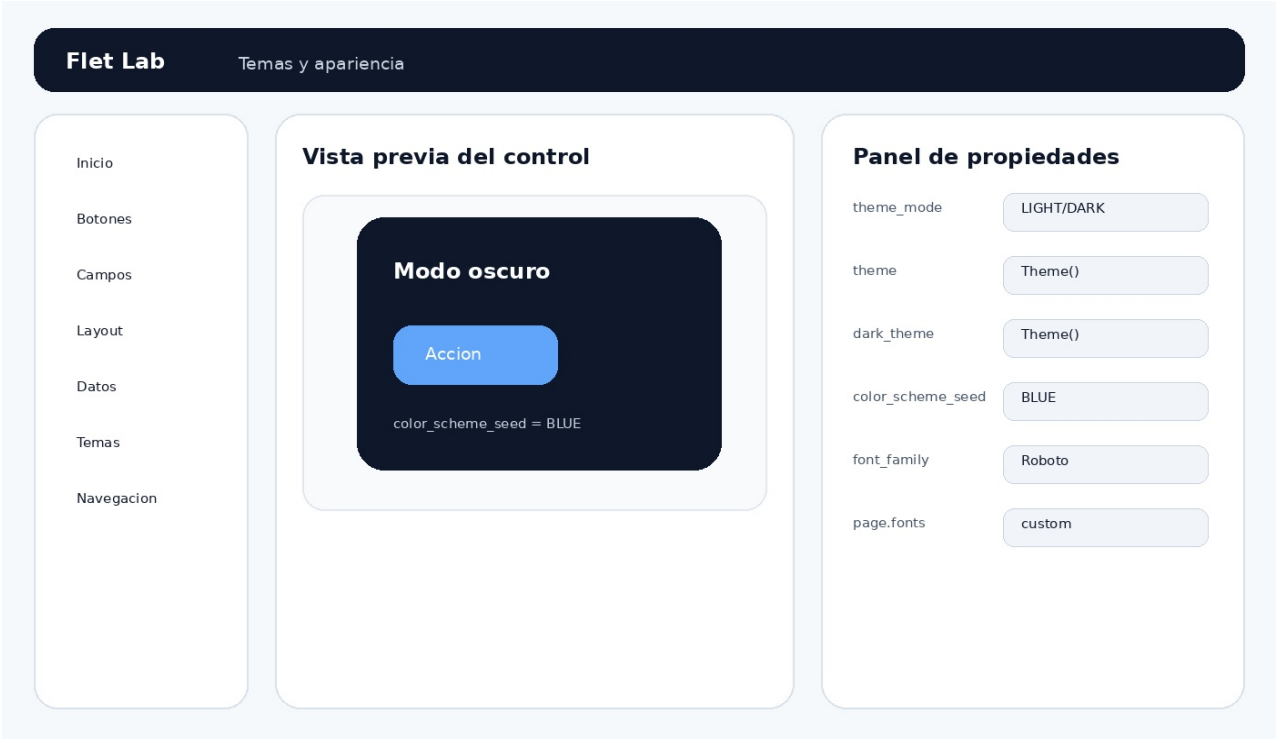

Figura 10. Cambio de tema, colores y modo oscuro.

El tema define la identidad visual de la app. Puedes configurar modo claro, oscuro o automatico, color base, fuentes y estilos. Es recomendable definir colores en un archivo constants.py o theme.py para no repetirlos.

```
def main(page: ft.Page):
    page.theme_mode = ft.ThemeMode.LIGHT
    page.theme = ft.Theme(color_scheme_seed=ft.Colors.BLUE)
    page.dark_theme = ft.Theme(color_scheme_seed=ft.Colors.BLUE_GREY)

    def cambiar_tema(e):
        page.theme_mode = (
            ft.ThemeMode.DARK
            if page.theme_mode == ft.ThemeMode.LIGHT
            else ft.ThemeMode.LIGHT
        )

    page.add(ft.Button("Cambiar tema", on_click=cambiar_tema))
```

Propiedad	Uso
page.theme_mode	LIGHT, DARK o SYSTEM.
page.theme	Tema principal.
page.dark_theme	Tema para modo oscuro.
color_scheme_seed	Color base del esquema.
page.fonts	Carga fuentes personalizadas desde assets.
font_family	Fuente usada por controles de texto.
ButtonStyle	Personaliza botones por estado.
TextStyle	Personaliza textos y columnas de tablas.

Evita meter colores quemados en todos los controles. Para apps grandes, crea variables como PRIMARY, SURFACE, TEXT_MUTED o usa el esquema del tema. Asi puedes cambiar la apariencia completa sin reescribir cada pantalla.

14. Eventos, estado, validacion y programacion asincrona

Estado simple

El estado puede estar en variables, clases o estructuras separadas. En apps pequenas basta con variables y controles. En apps medianas conviene crear clases de servicio o componentes personalizados.

```
class EstadoApp:
    def __init__(self):
        self.contador = 0


def main(page: ft.Page):
    estado = EstadoApp()
    texto = ft.Text("0", size=32)

    def sumar(e):
        estado.contador += 1
        texto.value = str(estado.contador)

    page.add(texto, ft.Button("Sumar", on_click=sumar))
```

Async

Usa async cuando una operacion pueda tardar: consultar una API, leer archivos grandes, esperar un temporizador o ejecutar tareas externas. Esto ayuda a que la interfaz no se sienta congelada.

```
import asyncio
import flet as ft

async def main(page: ft.Page):
    estado = ft.Text("Listo")

    async def cargar(e):
        estado.value = "Cargando..."
        await page.update_async()
        await asyncio.sleep(2)
        estado.value = "Datos cargados"
        await page.update_async()

    page.add(estado, ft.Button("Cargar", on_click=cargar))

ft.run(main)
```

Patron de validacion recomendado

Paso	Descripcion
1. Leer valores	Obtener .value de TextField, Dropdown, Checkbox, etc.
2. Validar	Comprobar obligatorios, formato, rangos y reglas de negocio.
3. Mostrar errores	Usar error_text, SnackBar o AlertDialog segun gravedad.
4. Guardar	Enviar a API, base de datos o archivo si no hay errores.
5. Limpiar/actualizar UI	Vaciar formulario, refrescar tabla o navegar.

15. Laboratorio interactivo: codigo para cambiar propiedades en vivo

El siguiente ejemplo es una base de laboratorio. La idea es tener una zona de vista previa y un panel de propiedades. Cuando cambias un campo del panel, modificas la propiedad del control y ves el resultado. Puedes ejecutarlo como escritorio con flet run main.py o como web con flet run --web main.py.

Este codigo es el punto de partida para construir la app interactiva que pediste: una pagina donde cada modulo muestra un control y permite cambiar dinamicamente sus propiedades.

```
import flet as ft


def main(page: ft.Page):
    page.title = "Flet Lab - Propiedades en vivo"
    page.theme_mode = ft.ThemeMode.LIGHT
    page.padding = 20

    preview_button = ft.Button(
        content="Guardar",
        icon=ft.Icons.SAVE,
        bgcolor=ft.Colors.BLUE,
        color=ft.Colors.WHITE,
        elevation=2,
    )

    txt_content = ft.TextField(label="content", value="Guardar")
    txt_width = ft.TextField(label="width", value="180")
    sw_disabled = ft.Switch(label="disabled", value=False)
    dd_color = ft.Dropdown(
        label="bgcolor",
        value="BLUE",
        options=[ft.dropdown.Option(x) for x in ["BLUE", "GREEN", "RED", "PURPLE"]],
    )

    def aplicar(e=None):
        preview_button.content = txt_content.value
        preview_button.width = int(txt_width.value or 180)
        preview_button.disabled = sw_disabled.value
        preview_button.bgcolor = getattr(ft.Colors, dd_color.value)

    for c in [txt_content, txt_width, sw_disabled, dd_color]:
        c.on_change = aplicar

    page.add(
        ft.Row(
            expand=True,
            controls=[
                ft.Container(
                    expand=2,
                    padding=30,
                    border_radius=20,
                    bgcolor=ft.Colors.GREY_100,
                    content=ft.Column([
                        ft.Text("Vista previa", size=24, weight=ft.FontWeight.BOLD),
                        preview_button,
                    ]),
                ),
                ft.Container(
                    expand=1,
                    padding=20,
                    border_radius=20,
                    bgcolor=ft.Colors.WHITE,
                    content=ft.Column([
                        ft.Text("Propiedades", size=22, weight=ft.FontWeight.BOLD),
                        txt_content,
                        txt_width,
                        dd_color,
                        sw_disabled,
                    ]),
                ),
            ],
        )
    )

ft.run(main)
```

Extension del laboratorio con Tabs

Para cubrir varios controles, crea una pestaña por familia: botones, campos, contenedores, tablas, layout y temas. Cada pestaña puede tener su propia funcion.

```
def button_lab():
    return ft.Column([ft.Text("Button Lab"), ...])


def textfield_lab():
    return ft.Column([ft.Text("TextField Lab"), ...])


def container_lab():
    return ft.Column([ft.Text("Container Lab"), ...])

page.add(
    ft.Tabs(
        selected_index=0,
        tabs=[
            ft.Tab(text="Button", content=button_lab()),
            ft.Tab(text="TextField", content=textfield_lab()),
            ft.Tab(text="Container", content=container_lab()),
        ],
    )
)
```

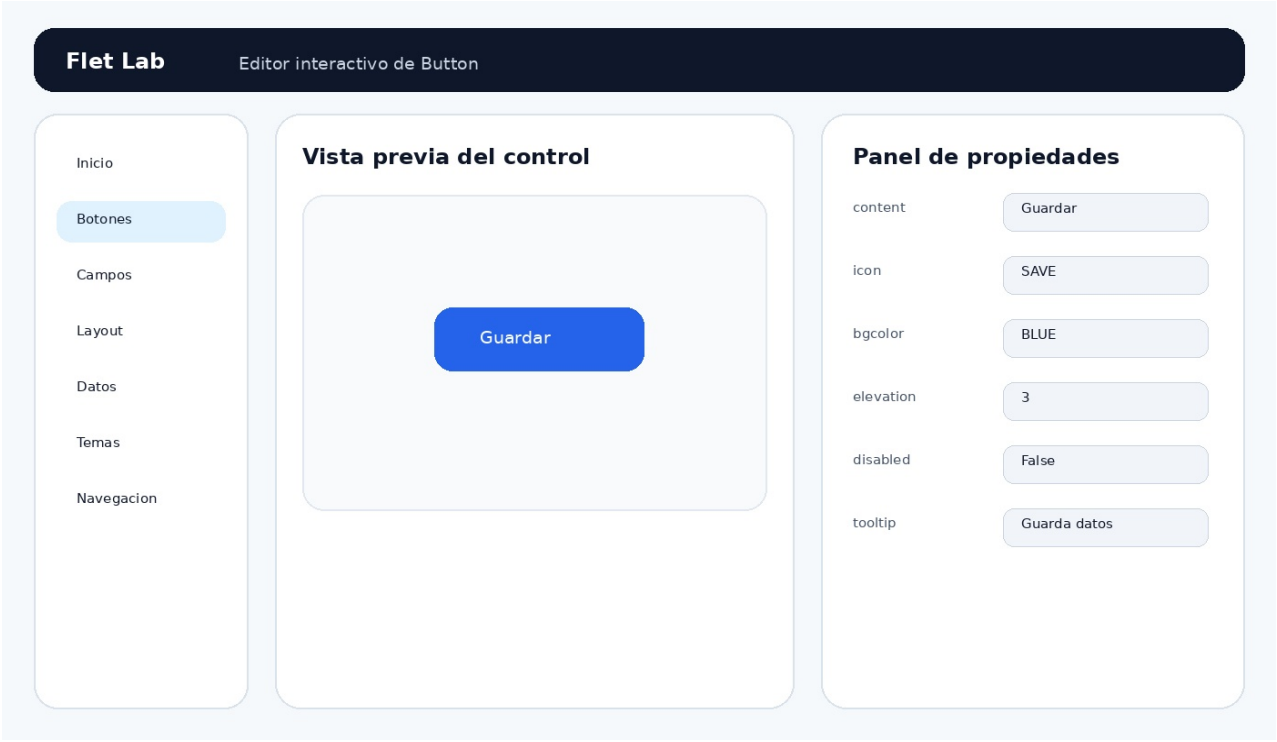

Figura 11. Ejemplo de pagina interactiva para propiedades de Button.

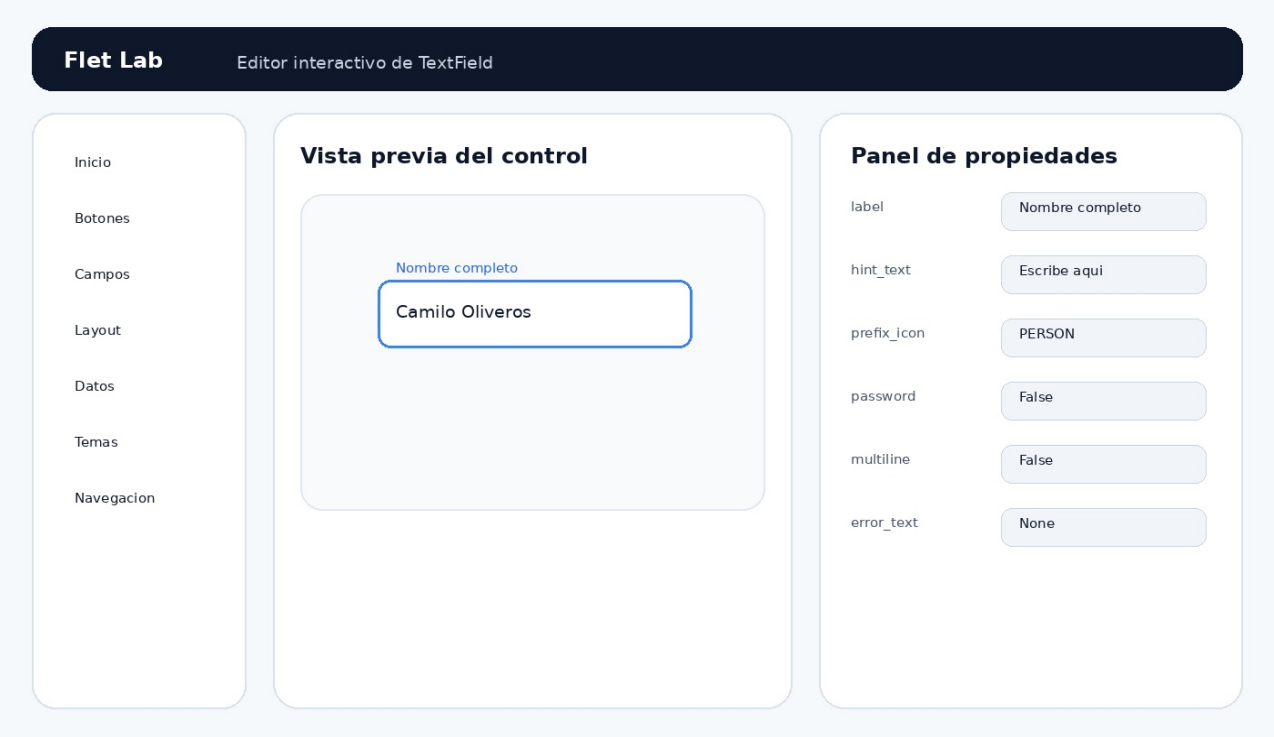

Figura 12. Ejemplo de pagina interactiva para propiedades de TextField.

16. Mini proyectos practicos

Mini proyecto 1: lista de tareas

```
import flet as ft


def main(page: ft.Page):
    page.title = "Lista de tareas"
    page.padding = 20

    entrada = ft.TextField(label="Nueva tarea", expand=True)
    tareas = ft.Column(spacing=8)

    def agregar(e):
        if not entrada.value.strip():
            entrada.error_text = "Escribe una tarea"
            return
        entrada.error_text = None
        tareas.controls.append(ft.Checkbox(label=entrada.value))
        entrada.value = ""

    page.add(
        ft.Text("Mis tareas", size=28, weight=ft.FontWeight.BOLD),
        ft.Row([entrada, ft.FilledButton("Agregar", on_click=agregar)]),
        tareas,
    )

ft.run(main)
```

Mini proyecto 2: login visual

```
def login_view(page: ft.Page):
    usuario = ft.TextField(label="Usuario", prefix_icon=ft.Icons.PERSON)
    clave = ft.TextField(label="Contrasena", password=True, can_reveal_password=True)
    msg = ft.Text()

    def entrar(e):
        if usuario.value == "admin" and clave.value == "1234":
            page.navigate("/dashboard")
        else:
            msg.value = "Credenciales incorrectas"
            msg.color = ft.Colors.RED

    return ft.View(
        route="/login",
        controls=[
            ft.AppBar(title=ft.Text("Login")),
            ft.Container(
                width=420,
                padding=30,
                border_radius=20,
                bgcolor=ft.Colors.SURFACE_CONTAINER_HIGHEST,
                content=ft.Column([usuario, clave, ft.FilledButton("Entrar", on_click=entrar),
 msg]),
            ),
        ],
    )
```

Mini proyecto 3: dashboard con tarjetas

```
def metric_card(title, value, icon, color):
    return ft.Card(
        content=ft.Container(
            padding=18,
            width=220,
            content=ft.Row([
                ft.Icon(icon, color=color, size=32),
                ft.Column([
                    ft.Text(title, size=13, color=ft.Colors.GREY),
                    ft.Text(value, size=26, weight=ft.FontWeight.BOLD),
                ]),
            ]),
        )
    )

page.add(
    ft.ResponsiveRow([
        metric_card("Ventas", "128", ft.Icons.SHOPPING_CART, ft.Colors.BLUE),
        metric_card("Usuarios", "84", ft.Icons.PERSON, ft.Colors.GREEN),
        metric_card("Alertas", "5", ft.Icons.WARNING, ft.Colors.ORANGE),
    ])
)
```

17. Build, exportacion, web, escritorio y Android

Durante desarrollo se usa flet run. Para generar paquetes o builds se usa flet build. Los destinos dependen de tu sistema operativo, herramientas instaladas y configuracion del proyecto.

```
# Escritorio / plataforma
flet build windows
flet build macos
flet build linux

# Web
flet build web

# Android
flet build apk
flet build aab

# Ejecutar la app web en desarrollo
flet run --web src/main.py
```

Destino	Comando	Notas
Desarrollo escritorio	flet run	Abre ventana nativa con hot reload.
Desarrollo web	flet run --web	Abre navegador con puerto local.
Windows	flet build windows	Genera app para Windows si el entorno lo soporta.

Destino	Comando	Notas
macOS	flet build macos	Normalmente se construye en macOS.
Linux	flet build linux	Requiere dependencias del sistema.
Web	flet build web	Genera carpeta web publicable.
Android APK	flet build apk	Paquete instalable de prueba.
Android AAB	flet build aab	Formato usado para Play Store.

Compilar para movil puede requerir SDKs, Java, Android tooling y configuracion adicional. Si falla un build, primero corre flet doctor y revisa el log completo.

18. Catalogo rapido de controles

El catalogo oficial de Flet incluye muchos controles. Esta tabla resume los mas importantes por familia para que sepas donde buscar y que aprender primero.

Familia	Controles principales	Para que sirven
Texto	Text, Markdown, RichText, SelectionArea	Mostrar texto simple, enriquecido o seleccionable.
Layout	Row, Column, Container, Stack, ResponsiveRow, GridView, ListView, SafeArea, Divider, VerticalDivider	Organizar visualmente la pantalla.
Botones	Button, TextButton, FilledButton, FilledTonalButton, OutlinedButton, IconButton, FloatingActionButton, PopupMenuButton	Ejecutar acciones.
Entrada	TextField, Checkbox, Switch, Radio, RadioGroup, Dropdown, Slider, RangeSlider, DatePicker, TimePicker	Capturar datos del usuario.
Navegacion	AppBar, BottomAppBar, NavigationBar, NavigationRail, NavigationDrawer, Tabs, View, Router	Moverse entre pantallas.
Datos	DataTable, ListTile, Card, ExpansionTile, ReorderableListView, AutoComplete, SearchBar	Mostrar y seleccionar informacion.
Feedback	AlertDialog, SnackBar, Banner, BottomSheet, ProgressBar, ProgressRing	Informar, confirmar o mostrar carga.
Media	Image, Video, Camera, WebView, Lottie, Rive, Map, Canvas, Screenshot	Imagen, video, camara, web, mapas y dibujo.
Adaptativos/Cupertino	CupertinoButton, CupertinoSwitch, CupertinoSlider, CupertinoTextField, CupertinoDatePicker, CupertinoNavigationBar	Apariencia estilo iOS/adaptativa.
Avanzados	AnimatedSwitcher, Draggable, DragTarget, ShaderMask, Semantics, TransparentPointer, WindowDragArea	Animacion, arrastre, accesibilidad y comportamiento avanzado.

Orden de aprendizaje recomendado

Nivel	Aprende
1	Text, Button, TextField, Row, Column, Container.
2	Checkbox, Dropdown, Slider, DatePicker, forms y validacion.
3	Card, ListView, DataTable, SnackBar, AlertDialog.
4	Page, View, rutas, AppBar, NavigationBar, Drawer.
5	ResponsiveRow, temas, animaciones, dashboards.
6	Async, APIs, almacenamiento, build y distribucion.

19. Errores comunes y soluciones

Problema	Causa probable	Solucion
No abre la app	No estas en la carpeta correcta o main.py no existe.	Ejecuta flet run src/main.py o revisa estructura.
ModuleNotFoundError: flet	Entorno virtual no activado o Flet no instalado.	Activa .venv e instala pip install "flet[all]".
Los cambios no se ven	No actualizaste control o evento externo no hizo update.	Usa page.update() o await page.update_async().
La pantalla se desborda	Falta scroll o layout responsive.	Usa page.scroll, ListView o ResponsiveRow.
Boton no responde	on_click no asignado o funcion mal definida.	Verifica def funcion(e): y on_click=funcion.
TextField no valida	No estas revisando .value.	Usa campo.value y asigna error_text.
Rutas raras en web	Stack de views no se reconstruye bien.	Centraliza logica en page.on_route_change.
Build falla	Faltan SDKs o dependencias.	Corre flet doctor y revisa el log.
Imagen no carga	Ruta incorrecta o asset no incluido.	Coloca archivo en assets y usa src correcto.
UI poco profesional	Sin tema, margenes o jerarquia.	Define theme, padding, spacing, cards y tipografia.

Checklist antes de entregar una app

Revision	Si/No
La app corre con flet run.	
La app corre con flet run --web.	
No hay errores en consola.	
Los formularios validan datos.	
Los botones principales tienen feedback.	
La navegacion funciona con atras/adelante.	
La pantalla no se rompe en anchos pequenos.	
Los colores y fuentes son consistentes.	
Las rutas y componentes estan organizados.	
flet doctor no muestra problemas importantes.	

20. Fuentes consultadas

Esta guia fue organizada con base en la documentacion oficial de Flet y la pagina de PyPI del paquete flet. Las capturas incluidas son representaciones visuales didacticas creadas para esta guia; al ejecutar los ejemplos, el aspecto puede variar segun sistema operativo, tema, version de Flutter/Flet y configuracion.

Fuente	Contenido consultado
PyPI - flet	Version 0.85.1 y fecha de publicacion.
Flet docs - Installation / create / running app	Instalacion, flet create, flet run, modo web y hot reload.
Flet docs - Controls catalog	Catalogo general de controles disponibles.
Flet docs - Page / Control	Concepto de Page, View y propiedades comunes de Control.
Flet docs - Button, TextField, Container, Row, Column, DataTable	Propiedades y patrones de uso de controles principales.

Fuente	Contenido consultado
Flet docs - Navigation and Routing	Modelo page.route, page.views, on_route_change y on_view_pop.

Referencias web: https://pypi.org/project/flet/ ; https://flet.dev/docs/ ; https://flet.dev/docs/controls/ ; https://flet.dev/docs/getting-started/ ; https://flet.dev/docs/cookbook/navigation-and-routing/ .