1.2 Создание приложений Flet на Python

Создание приложений Flet на Python

Чтобы написать приложение на Flet, вам не нужно быть гуру фронтенда, но рекомендуется иметь базовые знания Python и объектно-ориентированного программирования.

В этом руководстве мы изучим структуру приложения Flet, научимся выводить данные с использованием элементов управления Flet, запрашивать данные от пользователя и создавать базовые макеты страниц. Мы также рассмотрим некоторые варианты упаковки и развертывания, чтобы доставить готовое приложение вашим пользователям.

# Установка модуля

**flet**

Flet требует Python 3.7 или выше. Для начала работы с Flet вам нужно сначала установить модуль flet:

flet

pip install flet

ПРИМЕЧАНИЕ Для обновления модуля выполните:

pip install flet --upgrade

Чтобы установить предварительный выпуск Flet (для продвинутых пользователей) выполните:

pip install flet --pre

ОСТОРОЖНО Я рекомендую устанавливать предварительные сборки в виртуальное окружение.

### Linux

Для работы приложений Flet на Linux и WSL требуется установка библиотек GStreamer. Скорее всего, они уже установлены в вашей системе, но если вы получаете ошибку при загрузке общих библиотек: **libgstapp-1.0.so.0: не удается открыть объектный файл общего доступа: такого файла или каталога нет**, при запуске приложения Flet, то вам нужно установить GStreamer.

Для установки GStreamer на Ubuntu/Debian выполните следующие команды:

sudo apt-get update

sudo apt-get install libgstreamer1.0-dev libgstreamer-plugins-base1.0- dev libgstreamer-plugins-bad1.0-dev gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0- plugins-ugly gstreamer1.0-libav gstreamer1.0-doc gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio

Вы можете обратить к [этому](https://gstreamer.freedesktop.org/documentation/installing/on-linux.html?gi-language=c) руководству по установке на других дистрибутивах Linux.

### WSL

Приложения Flet могут работать на WSL2. Если вы получаете ошибку "**не удается открыть дисплей**", следуйте [этому](https://github.com/microsoft/wslg/wiki/Diagnosing-%22cannot-open-display%22-type-issues-with-WSLg) руководству для устранения неполадок.

# Базовая структура приложения

Минимальное приложение Flet имеет следующую структуру:

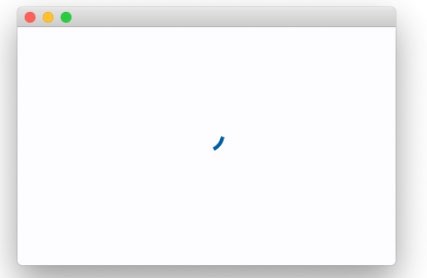
import flet as ft

def main(page: ft.Page):

# добавляем/обновляем элементы управления на Странице

pass

ft.app(target=main)

*ПРИМЕЧАНИЕ*

*Этот раздел называется "базовым", потому что позже мы рассмотрим более приближенные к реальности подходы к структуре приложения с повторно используемыми элементами управления.*

Типичная программа Flet заканчивается вызовом , где приложение

flet.app()

начинает ожидать новых пользовательских сессий. Функция main() является входной точкой в приложение Flet. Она вызывается в новом потоке для каждой

пользовательской сессии с переданным в него экземпляром Page . При запуске приложения Flet в браузере для каждой открытой вкладки или страницы создается новая пользовательская сессия. При запуске в качестве десктопного приложения

создается только одна сессия.

похож на "холст", специфичный для пользователя, визуальное состояние

Page

пользовательской сессии. Чтобы построить пользовательский интерфейс приложения, вы добавляете и удаляете элементы управления на страницу, обновляете их свойства. Приведенный выше пример кода будет отображать просто пустую страницу для каждого пользователя.

По умолчанию приложение Flet запускается в собственном окне ОС, что очень удобно для разработки. Однако, вы можете открыть его в новом окне браузера,

изменив вызов следующим образом:

flet.app

ft.app(target=main, view=ft.WEB\_BROWSER)

*СПРАВКА*

*Внутри каждое приложение Flet является веб-приложением, и даже если оно открыто в собственном окне ОС, на фоне все равно запускается встроенный веб- сервер. Веб-сервер Flet называется "Fletd" и по умолчанию он слушает случайный TCP- порт. Вы можете указать пользовательский TCP-порт, а затем открыть приложение в браузере вместе с настольным представлением:*

flet.app(port=8550, target=main)

Откройте

[http://localhost:](http://localhost/)<port>

вашего приложения Flet.

# Элементы управления

в вашем браузере, чтобы увидеть веб-версию

Пользовательский интерфейс состоит из элементов управления (или виджетов). Чтобы сделать элементы управления видимыми для пользователя, их нужно добавить на страницу или внутрь других элементов управления. Страница - это верхний элемент управления. Вложение элементов управления друг в друга можно представить как дерево, где страница является корнем.

Элементы управления - это обычные классы Python. Создайте экземпляры элементов управления с помощью конструкторов с параметрами, соответствующими их свойствам, например:

t = ft.Text(value="Привет, мир!", color="green")

Чтобы отобразить элемент управления на странице, добавьте его в список элементов управления страницы и вызовите page.update() , чтобы отправить изменения страницы в браузер или настольный клиент:

import flet as ft

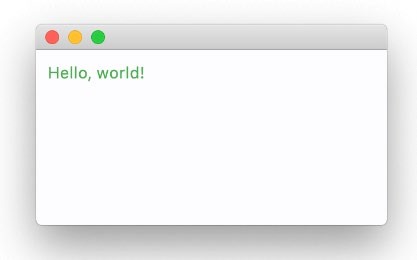
def main(page: ft.Page):

t = ft.Text(value="Hello, world!", color="green")

page.controls.append(t)

page.update()

ft.app(target=main)

*ПРИМЕЧАНИЕ*

*В следующих примерах я буду показывать только содержимое функции main .*

Вы можете изменить свойства элемента управления, и пользовательский интерфейс будет обновлен после следующего вызова page.update() :

t = ft.Text()

page.add(t) # это ярлык для page.controls.append(t), а затем

page.update()

for i in range(10):

t.value = f"Шаг {i}"

page.update()

time.sleep(1)

Некоторые элементы управления - это "контейнеры", которые могут содержать

другие элементы управления. Например, элемент управления располагать другие элементы управления в ряд:

Row

позволяет

page.add(

ft.Row(controls=[

ft.Text("A"),

ft.Text("B"),

ft.Text("C")

])

)

или

и

TextField

рядом:

ElevatedButton

page.add(

ft.Row(controls=[

ft.TextField(label="Ваше имя"),

ft.ElevatedButton(text="Произнеси мое имя!")

])

)

достаточно умный, чтобы отправить только изменения, сделанные

page.update()

с момента его последнего вызова, поэтому вы можете добавить несколько новых элементов управления на страницу, удалить некоторые из них, изменить свойства

page.update()

других элементов управления, а затем вызвать обновления, например:

для пакетного

for i in range(10):

page.controls.append(ft.Text(f"Строка {i}"))

if i > 4:

page.controls.pop(0)

page.update()

time.sleep(0.3)

Некоторые элементы управления, например кнопки, могут иметь обработчики событий, реагирующие на ввод пользователя,

например, ElevatedButton.on\_click :

def button\_clicked(e):

page.add(ft.Text("Кликнули!"))

page.add(ft.ElevatedButton(text="Нажми меня",

on\_click=button\_clicked))

И более продвинутый пример для простого списка задач:

import flet as ft

def main(page):

def add\_clicked(e):

page.add(ft.Checkbox(label=new\_task.value))

new\_task.value = ""

new\_task.focus()

new\_task.update()

new\_task = ft.TextField(hint\_text="Что нужно сделать?", width=300)

page.add(ft.Row([new\_task, ft.ElevatedButton("Добавить",

on\_click=add\_clicked)]))

ft.app(target=main)

*СПРАВКА*

*Flet реализует императивную модель пользовательского интерфейса, где вы "вручную" строите пользовательский интерфейс с состоянием элементов управления, а затем изменяете его, обновляя свойства элементов управления. Flutter реализует декларативную модель, где пользовательский интерфейс автоматически перестраивается при изменении данных приложения. Управление состоянием приложения в современных фронтенд-приложениях - это сложная задача, и "старомодный" подход Flet может быть более привлекателен для программистов без опыта работы с фронтендом.*

### Свойство

**visible**

У каждого элемента управления есть свойство visible , которое по умолчанию истинно - элемент управления отображается на странице.

Установка

в

полностью предотвращает отображение элемента

управления (и всех его дочерних элементов, если таковые имеются) на холсте страницы. Скрытые элементы управления не могут быть сфокусированы или выбраны с помощью клавиатуры или мыши, и они не генерируют никаких событий.

visible

false

### Свойство

**disabled**

У каждого элемента управления есть свойство disabled , которое по умолчанию ложно - элемент управления и все его дочерние элементы включены.

disabled

TextField

Свойство

данных, такими как

в основном используется с элементами управления ввода

, Dropdown , Checkbox , кнопки.

Однако, может быть установлен для родительского элемента управления,

disabled

и его значение будет распространяться на всех детей рекурсивно.

Например, если у вас есть форма с несколькими элементами управления ввода, вы

disabled

можете установить свойство индивидуально:

для каждого элемента управления

first\_name = ft.TextField()

last\_name = ft.TextField()

first\_name.disabled = True

last\_name.disabled = True

page.add(first\_name, last\_name)

Или вы можете поместить элементы управления формы в контейнер, например, Column , и затем установить для столбца свойство disabled :

first\_name = ft.TextField()

last\_name = ft.TextField()

c = ft.Column(controls=[

first\_name,

last\_name

])

c.disabled = True

page.add(c)

# Ссылки на элементы управления

Элементы управления Flet являются объектами, и для доступа к их свойствам нам нужно хранить ссылки (переменные) на эти объекты.

Рассмотрим следующий пример:

import flet as ft

def main(page):

first\_name = ft.TextField(label="First Name", autofocus=True)

last\_name = ft.TextField(label="Last Name")

greetings = ft.Column()

def btn\_click(e):

greetings.controls.append(ft.Text(f"Hello, {first\_name.value}

{last\_name.value}!"))

first\_name.value = ""

last\_name.value = ""

page.update()

first\_name.focus()

page.add(

first\_name,

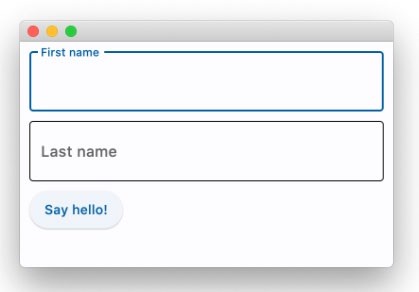
last\_name,

ft.ElevatedButton("Say Hello", on\_click=btn\_click),

greetings,

)

ft.app(target=main)

В самом начале метода мы создаем три элемента управления, которые

main()

on\_click

TextField

собираемся использовать в обработчике

кнопки: два

для

имени и фамилии и - контейнер для приветственных сообщений. Мы

Column

создаем элементы управления со всеми их свойствами и в конце метода main() , в вызове page.add() , мы используем их ссылки (переменные).

Когда добавляется все больше и больше элементов управления и обработчиков событий, становится сложно сохранить все определения элементов управления в одном месте, поэтому они распределяются по телу main() . Глядя на

параметры page.add() , трудно представить (без постоянного перехода к определениям переменных в IDE), как будет выглядеть конечная форма:

page.add(

first\_name,

last\_name,

ft.ElevatedButton("Say hello!", on\_click=btn\_click),

greetings,

)

Является ли ли

|  |  |  |  |
| --- | --- | --- | --- |
|  | first\_name | | TextField |
| greetings | |  | |

, установлен ли у него autofocus ? Является строкой или столбцом?

Flet предоставляет утилиту Ref , которая позволяет определить ссылку на элемент управления, использовать эту ссылку в обработчиках событий и установить ссылку на реальный элемент управления позже, при построении дерева. Идея пришла из

React.

Чтобы определить новую типизированную ссылку на элемент управления:

first\_name = ft.Ref[ft.TextField]()

Для доступа к элементу управления на который мы ссылаемся (дереференцирование управления), используйте свойство Ref.current :

# очистить поле имя

first\_name.current.value = ""

Чтобы присвоить элемент управления ссылке, установите

свойство на ссылку:

Control.ref

page.add(

ft.TextField(ref=first\_name, label="Имя", autofocus=True)

)

*ПРИМЕЧАНИЕ*

*Все элементы управления Flet имеют свойство ref .*

Мы могли бы переписать нашу программу, используя ссылки:

import flet as ft

def main(page):

first\_name = ft.Ref[ft.TextField]()

last\_name = ft.Ref[ft.TextField]()

greetings = ft.Ref[ft.Column]()

def btn\_click(e):

greetings.current.controls.append(

ft.Text(f"Hello, {first\_name.current.value}

{last\_name.current.value}!")

)

first\_name.current.value = ""

last\_name.current.value = ""

page.update()

first\_name.current.focus()

page.add(

ft.TextField(ref=first\_name, label="First name",

autofocus=True),

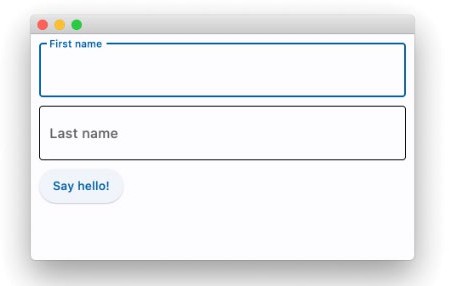
ft.TextField(ref=last\_name, label="Second name"),

ft.ElevatedButton("Say hello!", on\_click=btn\_click),

ft.Column(ref=greetings),

)

ft.app(target=main)

Теперь мы можем четко видеть в управления, из которых она построена.

page.add()

структуру страницы и все элементы

Да, логика становится немного более многословной, так как вам нужно

добавить для доступа к элементу управления ref , но это вопрос

.current.

личных предпочтений :)