3.1 Пользовательскиие элементы управления

Пользовательскиие элементы управления

Пользовательскиие элементы управления (UserControl) позволяют создавать изолированные повторно используемые компоненты, объединяя существующие элементы управления Flet. Пользовательскиие элементы управления ведут себя как Control , могут иметь методы и свойства.

Ниже приведен минимальный пример пользовательского элемента управления:

import flet as ft

class GreeterControl(ft.UserControl):

def build(self):

return ft.Text("Привет!")

def main(page):

page.add(GreeterControl())

ft.app(target=main)

должен реализовывать метод build() , который вызывается для построения интерфейса пользовательского элемента управления и должен

возвращать экземпляр одного или список элементов управления.

UserControl

Control

наследуется от Stack , поэтому несколько дочерних элементов будут расположены друг над другом. Если вам нужно расположить интерфейс элемента

UserControl

Column

управления иначе, используйте Row , компоновками, например:

или другие элементы управления

class GreeterControl(ft.UserControl):

def build(self):

return ft.Column([

ft.TextField(label="Ваше имя"),

ft.ElevatedButton("Войти")

])

изолирован от внешнего макета, т. е. когда

UserControl

метод update() вызывается для родительского элемента управления, любые

изменения не включаются в обновление

self.update()

|  |  |  |
| --- | --- | --- |
| внутри | UserControl | |
| UserControl | |  |

дайджеста.

должен вызвать

для отправки своих

изменений на страницу Flet, например:

import flet as ft

class Counter(ft.UserControl):

def add\_click(self, e):

self.counter += 1

self.text.value = str(self.counter)

self.update()

def build(self):

self.counter = 0

self.text = ft.Text(str(self.counter))

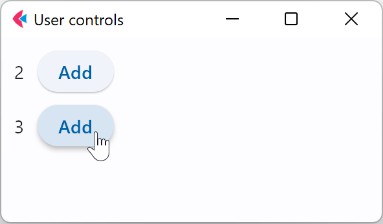
return ft.Row([self.text, ft.ElevatedButton("Добавить",

on\_click=self.add\_click)])

def main(page):

page.add(Counter(), Counter())

ft.app(target=main)



Вы можете объявить обработчики событий (например, ) и ссылки на элементы управления (например, self.text ) как члены класса или

UserControl

def add\_click(self, e)

реализовать всю логику внутри метода build() с использованием

локальных переменных и внутренних функций. Например, приведенный выше код можно переписать так:

class Counter(ft.UserControl):

def build(self):

self.counter = 0

text = ft.Text(str(self.counter))

def add\_click(e):

self.counter += 1

text.value = str(self.counter)

self.update()

return ft.Row([text, ft.ElevatedButton("Добавить",

on\_click=add\_click)])

*ПРИМЕЧАНИЕ*

*не может быть объявлен как локальная переменная, поскольку он не будет*

*counter*

*add\_clic*

*self.counter*

*видим внутри метода класса*

*.*

*k, поэтому он должен быть объявлен как поле*

Пользовательский элемент управления может иметь конструктор для передачи пользовательских данных, например:

import flet as ft

class Counter(ft.UserControl):

def init (self, initial\_count):

super(). init ()

self.counter = initial\_count

def build(self):

text = ft.Text(str(self.counter))

def add\_click(e):

self.counter += 1

text.value = str(self.counter)

self.update()

return ft.Row([text, ft.ElevatedButton("Добавить",

on\_click=add\_click)])

# затем используйте контрол

def main(page):

page.add(

Counter(100),

Counter(200))

ft.app(target=main)

*ПРИМЕЧАНИЕ*

*должен всегда вызываться в вашем собственном*

*super(). init ()*

*конструкторе.*

Пользовательский элемент управления предоставляет методы "хука" жизненного цикла приложения:

* - вызывается после добавления присвоения ему временного идентификатора.

did\_mount()

UserControl

will\_unmount()

* - вызывается перед удалением

на страницу и со страницы.

Используя эти методы, мы можем реализовать простой элемент управления "обратного отсчета":

UserControl

import flet as ft

import time, threading

class Countdown(ft.UserControl):

def init (self, seconds):

super(). init ()

self.seconds = seconds

def did\_mount(self):

self.running = True

self.th = threading.Thread(target=self.update\_timer, args=(),

daemon=True)

self.th.start()

def will\_unmount(self):

self.running = False

def update\_timer(self):

while self.seconds and self.running:

mins, secs = divmod(self.seconds, 60)

self.countdown.value = "{:02d}:{:02d}".format(mins, secs)

self.update()

time.sleep(1)

self.seconds -= 1

def build(self):

self.countdown = ft.Text()

return self.countdown

def main(page: ft.Page):

page.add(Countdown(120), Countdown(60))

ft.app(target=main)

В этом примере мы используем методы жизненного цикла для создания контрола обратного отсчета, который отсчитывает время до нуля.