1.8 Навигация и маршрутизация

Навигация и маршрутизация

Навигация и маршрутизация - это важная функция одностраничных приложений (SPA), которая позволяет организовать пользовательский интерфейс приложения в виртуальные страницы (представления) и "переходить" между ними, в то время как URL-адрес приложения отражает его текущее состояние.

Для мобильных приложений навигация и маршрутизация служат для глубокой привязки к конкретным частям приложения.

Оказалось, что внедрение навигации и маршрутизации в Flet заняло больше усилий, чем ожидалось, поскольку реализация основана на API Flutter Navigator 2.0 и потребовала заменить абстракцию "Page" Flet на "Page and Views". Новый API навигации и маршрутизации Flutter предлагает значительные улучшения, такие как:

* Программный контроль над историей просмотров.
* Простой способ перехватить вызов кнопки "Назад" в AppBar.
* Надежная синхронизация с историей браузера.



# Маршрут страницы

Маршрут страницы - это часть URL-адреса приложения после символа #: Стандартный маршрут приложения, если он не установлен в URL-адресе

приложения пользо например /store ,

|  |  |  |  |
| --- | --- | --- | --- |
| вателем, - это | / |  | . Все м |
| /authors/1/books/2 | | | |

аршруты начинаются с / ,

.

Маршрут приложения можно получить, прочитав свойство page.route , например:

import flet as ft

def main(page: ft.Page):

page.add(ft.Text(f"Initial route: {page.route}")) # Начальный путь

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

Скопируйте URL-адрес приложения, откройте новую вкладку браузера, вставьте URL-адрес, измените его часть после # на /test и нажмите enter. Вы должны увидеть "Initial route: /test".

Каждый раз, когда маршрут в URL изменяется (путем редактирования URL или навигации по истории браузера с помощью кнопок "Назад" / "Вперед"), Flet вызывает обработчик событий page.on\_route\_change :

import flet as ft

def main(page: ft.Page):

page.add(ft.Text(f"Initial route: {page.route}")) # Начальный путь

def route\_change(route):

page.add(ft.Text(f"New route: {route}")) # Новый путь

page.on\_route\_change = route\_change

page.update()

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

Теперь попробуйте обновить хеш URL несколько раз, а затем используйте кнопки "Назад" / "Вперед" в браузере! Вы должны увидеть новое сообщение, добавленное на страницу, каждый раз, когда маршрут изменяется.



Маршрут может быть изменен программно, обновив свойство page.route :

import flet as ft

def main(page: ft.Page):

page.add(ft.Text(f"Initial route: {page.route}")) # Начальный путь

def route\_change(route):

page.add(ft.Text(f"New route: {route}")) # Новый путь

def go\_store(e):

page.route = "/store"

page.update()

page.on\_route\_change = route\_change

page.add(ft.ElevatedButton("Go to Store", on\_click=go\_store)) #

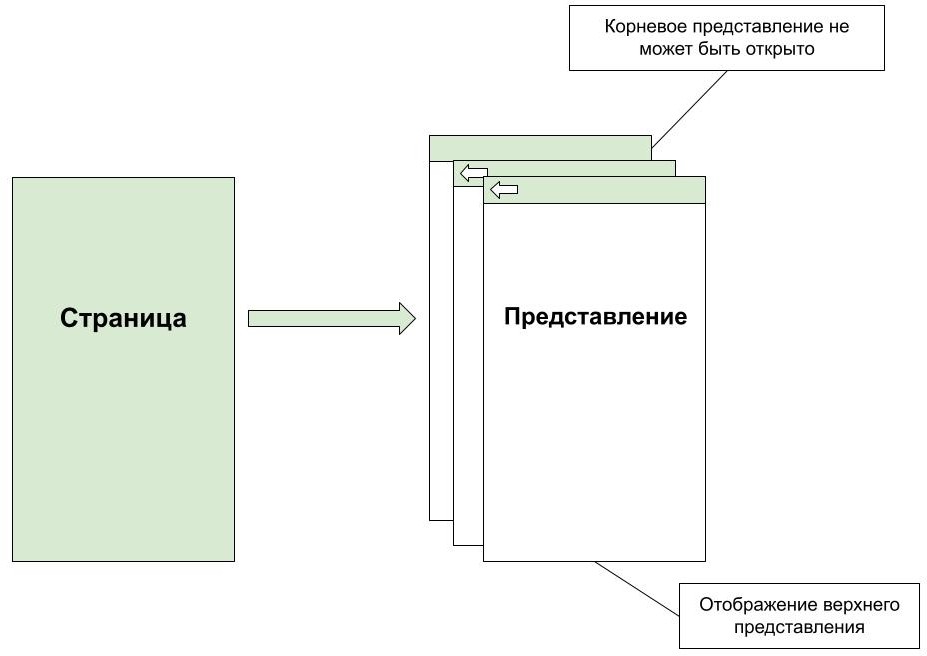
Перейти в магазин

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

Нажмите кнопку "Go to Store (Перейти в магазин)", и вы увидите, что URL-адрес приложения изменяется, и новый элемент добавляется в историю браузера. Вы можете использовать кнопку "Назад" браузера для перехода к предыдущему маршруту.

# Представления страницы

Теперь страница Flet - это не просто одна страница, но контейнер для View , расположенных друг на друга, как бутерброд:



Коллекция представлений представляет собой историю навигации. Страница имеет

свойство для доступа к коллекции представлений.

page.views

Последнее представление в списке - это то, которое в настоящее время отображается на странице. Список представлений должен содержать хотя бы один элемент (корневое представление).

Чтобы смоделировать переход между страницами, измените и добавьте новое представление в конец списка page.view .

page.route

Удалите последнее представление из коллекции и измените маршрут на "предыдущий" в обработчике событий page.on\_view\_pop , чтобы вернуться назад.

# Создание представлений при изменении маршрута

Для построения надежной навигации, в программе должно быть одно место, которое создает список представлений в зависимости от текущего маршрута. Иными словами, стек истории навигации (представленный списком представлений) должен быть функцией маршрута.

Это место - обработчик событий page.on\_route\_change .

Давайте сложим все в законченный пример, позволяющий осуществлять навигацию между двумя страницами:

import flet as ft

def main(page: ft.Page):

page.title = "Пример пути"

def route\_change(route):

page.views.clear()

page.views.append(

ft.View(

"/",

[

ft.AppBar(title=ft.Text("Flet приложение"),

bgcolor=ft.colors.SURFACE\_VARIANT),

ft.ElevatedButton("Посетить магазин",

on\_click=lambda \_: page.go("/store")),

],

)

)

if page.route == "/store":

page.views.append(

ft.View(

"/store",

[

ft.AppBar(title=ft.Text("Магазин"),

bgcolor=ft.colors.SURFACE\_VARIANT),

ft.ElevatedButton("Вернуться на главную",

on\_click=lambda \_: page.go("/")),

],

)

)

page.update()

def view\_pop(view):

page.views.pop()

top\_view = page.views[-1]

page.go(top\_view.route)

page.on\_route\_change = route\_change

page.on\_view\_pop = view\_pop

page.go(page.route)

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

Попробуйте переключаться между страницами с помощью кнопок "Посетить магазин" и "Вернуться домой", кнопок "Назад"/"Вперед" в браузере, вручную изменяя маршрут в URL.

*Примечание*

*Чтобы "перемещаться" между страницами, мы использовали -*

*page.go(route)*

*вспомогательный метод, который обновляет page.route , вызывает обработчик*

*событий вызывает*

|  |  |  |
| --- | --- | --- |
| *page.on\_route\_change* | | |
|  | *page.update()* | *.* |

*для обновления представлений и, наконец,*

Обратите внимание на использование обработчика событий page.on\_view\_pop . Он срабатывает, когда пользователь нажимает автоматическую кнопку "Назад" в управлении AppBar. В обработчике мы удаляем последний элемент из коллекции

представлений и переходим к корневому представлению под ним.

# Шаблоны маршрутов

Flet предлагает TemplateRoute - вспомогательный класс на основе библиотеки repath, который позволяет сопоставлять маршруты в стиле ExpressJS и парсить их параметры, например, /account/:account\_id/orders/:order\_id .

отлично работает с событием изменения маршрута:

TemplateRoute

troute = TemplateRoute(page.route)

if troute.match("/books/:id"):

print("ID книги:", troute.id)

elif troute.match("/account/:account\_id/orders/:order\_id"):

print("Учетная запись:", troute.account\_id, "Заказ:",

troute.order\_id)

else:

print("Неизвестный путь")

Вы можете узнать больше о поддерживаемом синтаксисе шаблона в библиотеке repath [здесь.](https://github.com/nickcoutsos/python-repath#parameters)

# URL Стратегия для Веб-приложения

Flet поддерживают два способа настройки маршрутизации на основе URL:

* Path (по умолчанию) - пути считываются и записываются без хеша. Например, fletapp.dev/path/to/view .
* Hash - пути считываются и записываются в фрагмент хеша. Например, fletapp.dev/#/path/to/view .

Для изменения стратегии URL используйте

параметр метода flet.app() , например:

route\_url\_strategy

ft.app(target=main, route\_url\_strategy="hash")

Стратегия URL для *Flet Server* может быть настроена с помощью переменной

path

среды

|  |  |  |
| --- | --- | --- |
| FLET\_ROUTE\_URL\_STRATEGY | | |
|  | hash | . |

умолчанию), либо на

, которую можно установить либо на

(по