Geekbrains

Факультет разработки

Создание веб приложения todo list

**Дипломный проект**

студента 1 года обучения

специальности “Разработчик - Программист”

Выржиковский Ф.Н.

Руководитель

Волчек А.

Москва

2024

**Дипломный проект “Создание веб приложения todo list”**

**Содержание**

Оглавление

[Введение 4](#_Toc167666663)

[Глава 1: Подготовка к разработке веб-приложения 6](#_Toc167666664)

[1.1 Анализ стека web-технологий 6](#_Toc167666665)

[1.2 Попытки в разные фреймворки и языки 13](#_Toc167666666)

[Глава 2: Как я пришел к разработке на фреймворке flet 14](#_Toc167666667)

[2.1 Что такое Flet? 14](#_Toc167666668)

[2.2 Создание нового приложения на Flet 17](#_Toc167666669)

[2.3 Навигация и маршрутизация 18](#_Toc167666670)

[2.4 Асинхронные приложения 26](#_Toc167666671)

[Глава 3: Создание веб приложения Todo app на Flet 32](#_Toc167666672)

[3.1 Приступая к работе 32](#_Toc167666673)

[3.2 Добавление элементов управления страницей и обработка событий 33](#_Toc167666674)

[3.3 Макет страницы 34](#_Toc167666675)

[3.4 Многоразовые компоненты пользовательского интерфейса 37](#_Toc167666676)

[3.5 Просмотр, редактирование и удаление элементов списка 38](#_Toc167666677)

[3.6 Фильтрация элементов списка 44](#_Toc167666678)

[3.7 Финальные доработки и подведение итогов 47](#_Toc167666679)

[Заключение 51](#_Toc167666680)

[Список используемой литературы 51](#_Toc167666681)

# Введение

Описание проекта:

Данный проект представляет собой разработку приложения todo list на Python, которое позволит пользователям эффективно управлять своими задачами, создавать списки дел, устанавливать приоритеты и сроки выполнения задач.

Обоснование темы:

Тема разработки приложения todo list обоснована повседневной необходимостью эффективного планирования времени и управления задачами. Современный ритм жизни требует от людей организованности и систематизации дел, поэтому разработка удобного приложения для управления списками дел будет актуальной и полезной.

Цель проекта:

Целью данного проекта является создание приложение для эффективного управление целями и задачами позволяет организовать работу, добиться результатов, управлять временем и ресурсами. Планирование целей и задач помогает улучшить продуктивность, сфокусировать усилия и достичь успеха в различных областях жизни, будь то работа, учеба, личное развитие и т.д.

Процесс управления целями и задачами предусматривает установление целей, определение приоритетов, планирование и распределение задач, контроль выполнения и оценку результатов. Эффективное планирование и управление целями и задачами помогает рационально использовать ресурсы, повышать производительность и достигать желаемых результатов.

План работы:

Изучение требований и анализ существующих приложений todo list.

Проектирование пользовательского интерфейса и функционала приложения.

Написание кода приложения с учетом заданных требований.

Тестирование приложения на соответствие функциональности и удобству использования.

Доработка приложения и реализация предложенных улучшений.

Специализация:

Дипломный проект выполняется в рамках специализации по программированию на языке Python.

Проблема, которую решает проект:

Проект решает проблему не структурированности и неорганизованности в планировании и выполнении задач. Пользователи смогут эффективно составлять списки дел, устанавливать сроки и приоритеты, что позволит им более осознанно распределять своё время и управлять задачами.

Опыт решения задачи:

Разработка подобных проектов на Python и использование фреймворков могут быть захватывающим и полезным опытом. Этот проект позволил мне не только попробовать свои силы в создании приложений todo list, но и получить ценный опыт работы с данными, анализа потребностей пользователей и оптимизации процесса управления задачами. Процесс создания приложения был увлекательным, позволяя мне на практике применить знания и навыки Python. Работа с фреймворками и разработка приложений на Python открывают широкие возможности для изучения новых концепций программирования, архитектурных решений и методов улучшения пользовательского опыта. Постепенно вы будете осваивать новые инструменты и приемы разработки, что позволит мне совершенствовать свои профессиональные навыки и расширять свой кругозор в области программирования.

Инструменты и технологии:

Использовал в проекте новый для себя фреймворк flet. Который позволяет писать приложения для чего угодно, в моем случае веб. Данный фрейморк включает в себя язык программирования Python.

# Глава 1: Подготовка к разработке веб-приложения

## 1.1 Анализ стека web-технологий

Front-end - это термин, который был введён в обиход ещё в XIX веке, он обозначает часть программы, которая отвечает за отображение. Не смотря на это, в области разработки веб-приложений термин претерпел некоторые изменения. В данном контексте Front-end - это часть приложения, которая выполняет свою работу на клиенте, и ее функционал может быть очень широк. При рассмотрении последних тенденций в развитии технологий, связанных с веб-технологиями, становится ясно, что из-за увеличения возможностей клиентской аппаратуры и развития технологий передачи данных (постепенный рост скорости передачи данных) клиентская часть приложения становится все более “толстой”. В данной статье мы рассмотрим различные технологии, которые используются для создания фронт-энд.

Также будет рассмотрена область их применения. CSS, HTML и JavaScript – три кита в этой системе. У любого front-end программиста есть этот стандартный набор. При использовании HTML не требуется дополнительного уточнения, даже если он и не используется в полной мере. В результате страницы будут сформированы в виде HTML кода.

На данный момент, в области CSS можно рассмотреть такое явление как препроцессоры. Препроцессорами называют вспомогательные средства, которые используются для обработки и хранения информации в CSS. В 2018 году они были добавлены с помощью новых способов использования синтаксических конструкций. Код, который был написан с использованием препроцессора, в дальнейшем будет преобразован к стандартным CSS-кодам, так как именно этот код поддерживает каждый браузер. Основным предназначением препроцессора является расширение или же преобразование стандартных конструкций в языке программирования CSS, добавление новых элементов, которые являются стандартными для большинства языков программирования, а также обеспечение более понятной структуры и иерархии, которые не всегда хорошо выполнены в стандартных препроцессорных конструкциях.

В следующем списке находятся самые популярные препроцессоры:

1.Sass (SCSS) является самым мощным из CSS-препроцессоров. Это сообщество разработчиков насчитывает более тысячи человек. Она имеет значительно более широкий спектр возможностей, чем Less. Данные возможности препроцессора получают дальнейшее развитие благодаря использованию в его составе многофункциональной библиотеки Compass, которая позволяет выйти за рамки CSS и использовать спрайты в автоматическом режиме.

2. Less. Главной его характеристикой является простота, а также стандартизованный для CSS синтаксис. Отмечается, что в качестве недостатков можно отметить отсутствие стандартных синтаксических конструкций, которые используются в стандартном понимании, например условий и циклов. Помимо этого, существует возможность расширения функционала с помощью системы плагинов. К тому же, это является отличным выбором в случае средних и низких по сложности стилей обучения, а также при том, что у вас нет большого количества времени на обучение.

3. Stylus является самым молодым CSS-препроцессором, но при этом он считается наиболее перспективным и удобным в использовании. В основе лежит JavaScript. Выполняет функции стандартного и упрощенного синтаксиса. Однако в нем отсутствуют некоторые знаки, которые несут ответственность за разметку кода. JavaScript является языком программирования, используемым в Front-end части приложения. Начиная с самого начала он разрабатывался под веб и поддерживается всеми современными браузерами. JavaScript обладает низким порогом вхождения, его синтаксис имеет очень лояльное отношение к ошибкам, поэтому для того чтобы начать использовать его, необходимо иметь минимальные знания. Для разработки приложений, которые будут иметь большой охват, следует использовать фреймворки или языки, основанные на JavaScript.

TypeScript обладает возможностью явного статического назначения типов, поддерживает полноценные классы и объекты, подобные классическим, а также имеет модульную систему, что 18 Межвузовский сборник научных трудов призвано повысить скорость разработки, облегчить читаемость, переделку и повторное использования кода, помочь осуществлять поиск ошибок на этапе разработки и компиляции, и, возможно, ускорить выполнение программ. Язык предназначен для разработки и поддержки средних и больших веб-приложений. Таким образом, он имеет больший порог вхождения по сравнению с JavaScript. Еще стоит отметить компании CoffeeSript и Dart. У него еще более низкий порог вхождения по сравнению с JS, что объясняется использованием так называемого «синтаксического сахара», который является добавлением различных синтаксических конструкций. По своим характеристикам Dart схож с TypeSript. Он также поддерживает классическую и строгую типизацию в ООП. Однако, оба этих языка, на данный момент, прекратили свое развитие. В связи с этим, мы не станем углубляться в их изучение в данной статье. Приложения, которые создаются на языке Javascript, называются фреймворками. Они предназначены для создания динамических веб-сайтов, мобильных и настольных приложений. Это обусловлено тем, что они имеют в себе большое количество функциональности и готовы к использованию, что значительно облегчает разработку веб приложения.

1. Angular является наиболее популярным и часто используемым фреймворком на данный момент. Фреймворк, который расширяет HTML, включает в себя набор атрибутов, которые могут динамично изменять разметку. Исходя из этого, можно сделать вывод, что это позволяет использовать декларативное привязывание (“declarative binding”) к событиям для отображения динамических данных и состояний, которые основаны на взаимодействии пользователя и его системы. Создается компанией Google, что означает ее мощную поддержку и постоянное развитие. В сравнении с другими фреймворками имеет более высокий порог вхождения. Первое место занимает по количеству модулей и функциональности. Считается наиболее подходящим для больших и сложных проектов. На сегодняшний день первая версия является наиболее популярной, и при выборе технологий, предпочтение отдается именно ему. Нынешняя версия, хоть и является более современной, имеет более высокий порог вхождения. Это существенно отличает ее от первой версии.

2. React. Наибольшее внимание уделяет созданию интерфейса, в связи с этим он имеет свои плюсы и минусы. Существует благодаря компании Facebook, в его состав входит довольно большое количество сообществ. В основе лежит использование JSX компонентов. Существует много мнений относительно того, какие именно плюсы и минусы имеет использование кода в HTML. В некоторых случаях это является удобным и эффективным способом, но при этом может ухудшить читаемость текста. Его легко использовать, он имеет низкий порог входа, но его функциональность не всегда охватывает все области, которые требуют решения. Это может привести к необходимости поиска дополнительных библиотек.

3. Vue.js – это своего рода компромисс между Angular и React, который часто называют так. Он имеет гораздо больше возможностей, чем React, но при этом его функционал менее объемен и быстрее, чем Angular. Главным недостатком является то, что данный фреймворк является сравнительно новым и не прошел полной проверки при разработке. Это означает, что количество документации по нему в данный момент очень мало.

4. Ember. На сервере он может выполнять связывание данных, автоматическое обновление компонентов и рендеринг. Это стандартная характеристика, которая характерна для современных фреймворков. Он выделяется среди других своей устойчивостью и стабильностью. Его авторы сделали все возможное, чтобы обеспечить максимальную совместимость между версиями. Именно по этой причине он чаще всего используется для осуществления долгосрочных проектов. Стоит отметить, что он имеет четкую соответствие с паттерном MVC, что делает его простым в освоении, но достаточно сложным в применении при иной модели. Кроме того, следует упомянуть о фреймворке Bootstrap, который является так называемым CSS-фреймворком. Для того, чтобы упростить верстку, в нем есть множество шаблонов и классов, которые используются для оформления страницы. Для обозначения серверной части web-приложения, используемой в клиентской архитектуре, используют термин "back-end". Предлагаем рассмотреть некоторые популярные технологии, используемые в работе сервера. Ниже приведен список основных серверных языков для программирования.

1. PHP (Hypertext Preprocessor), который является скриптовой системой, которая может интерпретировать и создавать динамические страницы, можно создать страницу с динамичным содержимым. Распространен и признан как лидер в своей области.

2. Perl (практический анализ и отчетный язык) - переводится как "практический язык, который используется для извлечения информации и составления отчетов". Главным преимуществом языка является его безграничные возможности в области работы с текстом и использования регулярных выражений, которые встроены в его синтаксис. В качестве основного предназначения, можно отметить манипуляцию с текстами, создание сетевого программного обеспечения, игр и разработку пользовательских интерфейсов.

3. Python - это скриптовый язык, который имеет несколько уровней сложности, он направлен на повышение производительности разработки. В нем есть функция, которая поддерживает многопоточность.

4. Ruby (Рубин) является динамическим, интерпретируемым и высокоуровневым языком. Основной плюс в том, что реализация многопоточности осуществляется независимо от операционной системы.

5. Java используется очень сильный уровень абстракции, что позволяет использовать его в качестве высокоуровневого объектно-ориентированного языка программирования. Труды 20 Межвузовского междисциплинарного сборника научных исследований объединяются в единый байт-код, который имеет возможность запускаться в любой операционной системе. Другая важная особенность – это то, что система безопасности имеет высокий уровень гибкости. Она идеально подходит для больших приложений, которые требуют высокую степень защиты.

Ниже перечисленные языки программирования пользуются популярностью в веб-разработке, и их развитие идет достаточно быстро. У каждого из них есть свои преимущества и они подходят для решения определенных задач. Но в данной статье мы будем рассматривать технологии, которые имеют отношение к PHP, так как он популярен, легко поддается изучению и имеет большое количество библиотек, которые можно использовать. Кроме того, этот язык обладает большим количеством сообществ, что позволяет быстро развиваться и расширяться. Далее будет рассмотрен ряд популярных фреймворков.

1. Symfony - имеет в своем арсенале паттерн MVC, который используется в PHP. С помощью этого программного обеспечения можно значительно ускорить процесс разработки и упростить рутинную работу программиста. Оно работает с поддержкой PHP более, чем на 5-й версии. Установлен на многих платформах, таких как MySQL, PostgreSQL и SQLite. В реляционных моделях объекты в виде объектной модели представляют собой сущности, которые представлены в таблицах. Это является технологией, которая называется ORM (Object-Relation Model – объектно-реляционная модель данных). На него распространяется лицензия MIT. В нем присутствует открытый исходный код.

2. Laravel также написан на PHP с использованием паттерна MVC, но в отличие от предыдущего имеет другой язык программирования. На него распространяется лицензия MIT. В нем присутствует открытый исходный код. Применяется в более крупных проектах, чем Symfony. Например, в корпоративном масштабе.

3. Yii (yes it's) - является примером кода, написанного на PHP с использованием паттерна MVC.

В настоящий момент наиболее распространенным является Symfony, который имеет множество библиотек и разработчиков, которые работают с ним. Он может быть использован как для небольших, так и для больших веб-приложений. В нем есть удобство, благодаря которому можно расширять приложение, не уничтожая при этом работающие компоненты. Он легко и просто интегрируется с различными технологиями, например с реляционными базами данных, документами-ориентированными базами данных или технологиями кэширования. Также он может быть использован для управления очередями, создания систем управления потоками данных, а также другими внешними ресурсами. Список основных СУБД, которые используются в веб-приложениях, выглядит следующим образом:

1. MySQL – это свободная реляционная система управления, которая разрабатывается корпорацией Oracle. Для использования в малых и средних web-приложениях. В большинстве случаев, MySQL используют на сервере для локальных или удаленных клиентов. В нем описываются методы и средства обработки и хранения информации, а также способы их реализации. В нем используется 21 язык программирования, включая С и С++, а также кроссплатформенная поддержка операционной системы.

2. PostgreSQL является свободной реляционной системой с открытым исходным кодом, поддерживаемой сообществом программистов. Она имеет независимую систему управления, которая не зависит от операционной системы. По мнению многих, наиболее продуктивными являются высокопроизводительные и надежные механизмы транзакций и репликции, поддерживающие множество языков программирования, а также расширяя систему поддержки встроенных языков, с помощью наследования и легкой расширения. 3.Документно-ориентированная система с открытым исходным кодом, имеющая открытый исходный код, называется "MongoDB". Не требуется создание и описание схем баз данных, также не существует никакой взаимосвязи между ними. В нем использованы языки C, C++ и JavaScript. В формате JSON хранятся данные. Используется в больших объемах данных, таких как веб-страницы и базы данных. Он имеет ряд преимуществ: высокая степень отказоустойчивости, возможность быстрой обработки информации, отсутствие взаимосвязи между таблицами, а также объединение данных из множества различных сущностей в реальном времени. При использовании каждой из перечисленных СУБД можно отметить ее достоинства и недостатки, которые будут зависеть от конкретной задачи. Являясь небольшим по размеру веб-приложением, лучше всего использовать MySQL, который имеет базовый функционал для быстрого проектирования и хорошо знаком многим программистам. Существует несколько вариантов использования PostgreSQL: если это более крупное приложение, то рекомендуется использовать ее для решения более сложных задач. А MongoDB лучше использовать в комплексе с реляционной СУБД, которая обладает большей надежностью и может обеспечить более точное и полное хранение важных данных. В приложении используется для ускорения высоконагруженных компонентов, так называемая, MongoDB.

## 1.2 Попытки в разные фреймворки и языки

Изначально мой выбор пал на фреймворк Vue.js, который действительно популярен в разработке одностраничных приложений и имеет множество крутых примеров, таких как Dashy Dashboard, который стал моим вдохновением. Однако, при изучении документации и работы с Vue.js, я столкнулся с сложностями из-за недостаточного знания JavaScript и TypeScript. Понимание этих языков играет важную роль в работе с фронтенд-фреймворками.

После чего задумался о десктопном решении для macOS, но обнаружил, что PowerShell не отображает некоторые элементы интерфейса, что также стало препятствием в развитии проекта. Эти нюансы и трудности в использовании Vue.js и работе с десктопным решением привели вас к поиску идеального фреймворка, который был бы более подходящим с учетом текущих знаний, удобства работы и потенциальной масштабируемости приложения.

Поиск подходящего инструмента для воплощения и реализации моих идей занял некоторое время, но это важный этап развития и профессионального роста. В итоге изначально почитал про flutter от Google, а далее нашел Flet мой идеальный фремворк для реализации моей идеи.

# Глава 2: Как я пришел к разработке на фреймворке flet

## 2.1 Что такое Flet?

Flet — это фреймворк, который позволяет создавать веб-, настольные и мобильные приложения на Python без предварительного опыта разработки интерфейса.

Вы можете создать пользовательский интерфейс для своей программы с помощью элементов управления Flet, основанных на Flutter от Google. Flet выходит за рамки простой упаковки виджетов Flutter. Он добавляет свою индивидуальность, комбинируя виджеты меньшего размера, упрощая сложность, внедряя лучшие практики пользовательского интерфейса и применяя разумные настройки по умолчанию. Это гарантирует, что ваши приложения будут выглядеть стильно и безупречно, не требуя дополнительных усилий по дизайну с вашей стороны.

Пример кода с использованием flet на языке Python, приложение счетчик:

import flet as ft

def main(page: ft.Page):

page.title = "Flet counter example"

page.vertical\_alignment = ft.MainAxisAlignment.CENTER

txt\_number = ft.TextField(value="0", text\_align=ft.TextAlign.RIGHT, width=100)

def minus\_click(e):

txt\_number.value = str(int(txt\_number.value) - 1)

page.update()

def plus\_click(e):

txt\_number.value = str(int(txt\_number.value) + 1)

page.update()

page.add(

ft.Row(

[

ft.IconButton(ft.icons.REMOVE, on\_click=minus\_click),

txt\_number,

ft.IconButton(ft.icons.ADD, on\_click=plus\_click),

],

alignment=ft.MainAxisAlignment.CENTER,

)

)

ft.app(main)

Для того, чтобы первый раз запустить приложение необходимо ввести в терминал среды разработки visual studio code, следующую команду:

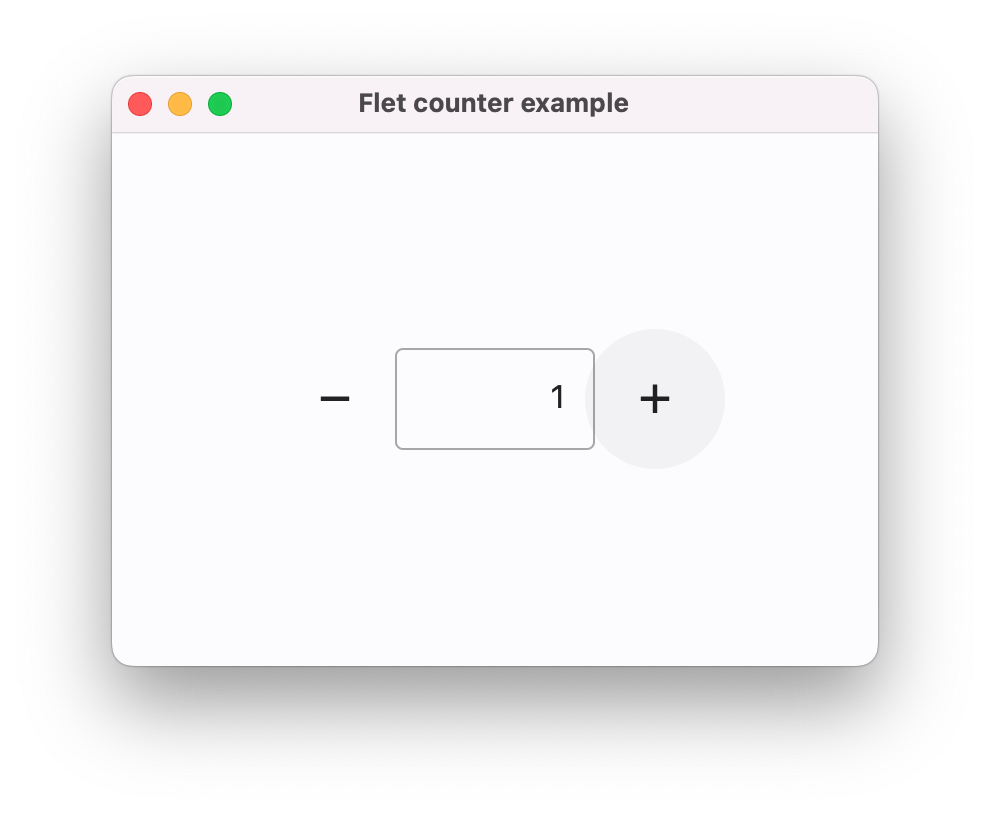
pip install flet

И для того, чтобы запустить данное приложение:

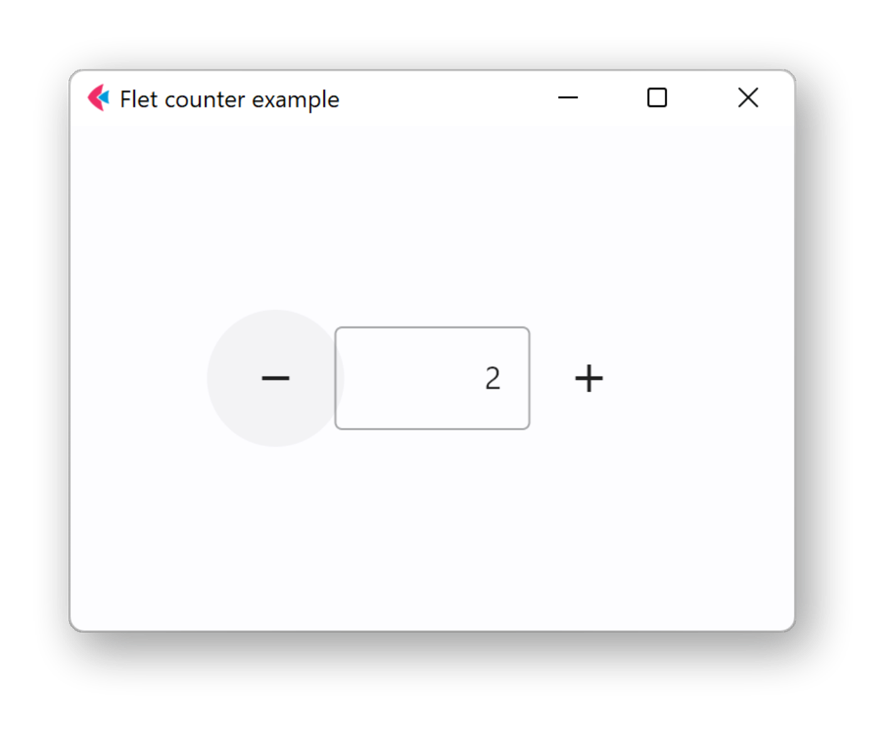
flet run counter.py

После того, как приложение будет запущено оно будет выглядеть абсолютно одинаково на разных платформах.

macOS



Windows



Это и стало основой для моего выбора.

## 2.2 Создание нового приложения на Flet

Для создания минимального функционала на Flet нужно использовать следующую команду:

flet create <project-name>

Где project-name название текущего проекта.

Флет создаст каталог <имя-проекта> со следующим файлом main.py:

import flet as ft

def main(page: ft.Page):

page.add(ft.SafeArea(ft.Text("Hello, Flet!")))

ft.app(main)

В программе Flet есть функция main(), с помощью которой вы можете добавлять элементы пользовательского интерфейса (элементы управления) на страницу или в окно. Приложение завершается блокирующей функцией ft.app(), которая инициализирует приложение Flet и запускает main().

Чтобы создать новое приложение Flet из шаблона «счетчик», выполните следующую команду:

flet create --template counter <имя-проекта>

Или, чтобы создать приложение Flet из шаблона счетчика в вашем текущем каталоге, выполните следующую команду:

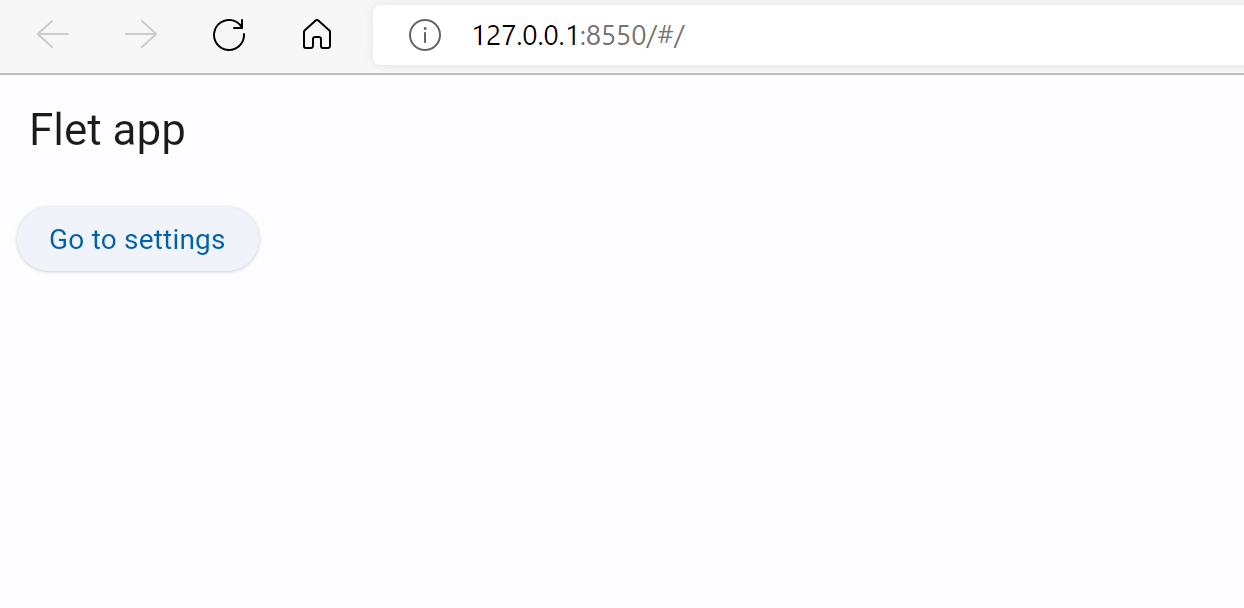
flet create --template counter .

## 2.3 Навигация и маршрутизация

Это важная функция одностраничных приложений (SPA), которая позволяет организовать пользовательский интерфейс приложения в виртуальные страницы (представления) и «перемещаться» между ними, в то время как URL-адрес приложения отражает текущее состояние приложения.

Для мобильных приложений навигация и маршрутизация служат глубокой связью с конкретными частями приложения.

Что ж, добавление навигации и маршрутизации во Flet потребовало больше усилий, чем ожидалось, поскольку реализация основана на API Navigator 2.0 Flutter и требует замены абстракции Flet «Страница» на «Страницы и представления». Новый API навигации и маршрутизации Flutter имеет существенные улучшения, такие как:



Маршрут страницы - это часть адреса URL приложения, которая находится после символа решетки (#).



Для получения текущего маршрута приложения используется свойство page.route. По умолчанию, если маршрут не указан явно в URL-адресе приложения, то используется базовый маршрут "/". Любой определенный маршрут всегда начинается с символа "/", например, "/store", "/authors/1/books/2".

import flet as ft

def main(page: ft.Page):

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

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

При изменении маршрута в URL-адресе (путем редактирования или навигации) веб-приложение вызывает обработчик событий page.on\_route\_change. Этот обработчик реагирует на изменения маршрута, позволяя приложению адаптироваться к новым условиям и информировать об изменениях в маршруте страницы. В результате обработчика событий page.on\_route\_change приложение может выполнять дополнительные действия согласно новому маршруту, обновлять содержимое страницы и предоставлять соответствующий контент пользователю.

import flet as ft

def main(page: ft.Page):

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

def route\_change(e: ft.RouteChangeEvent):

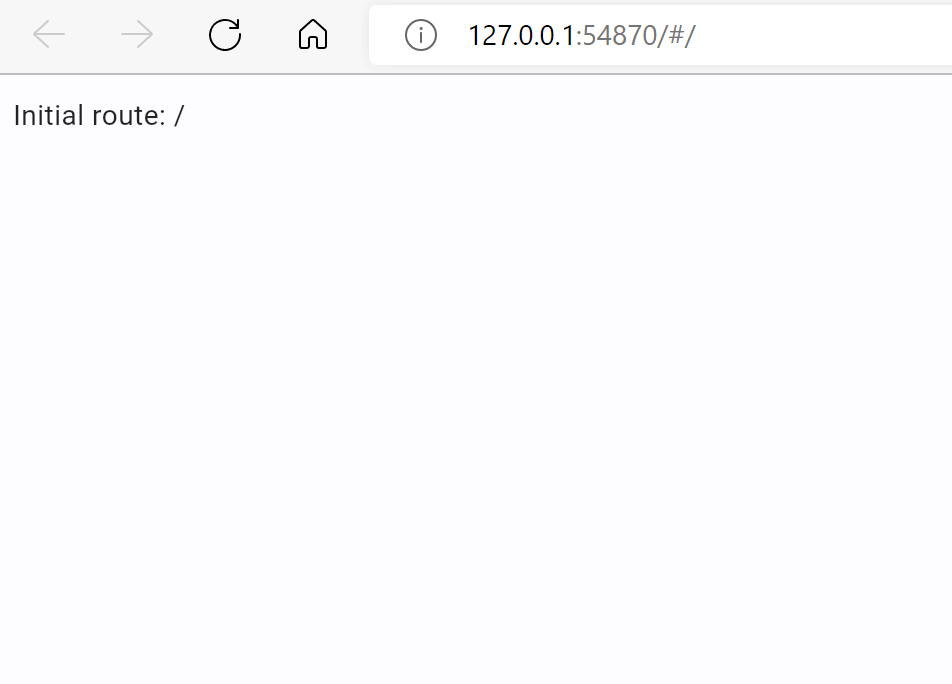
page.add(ft.Text(f"New route: {e.route}"))

page.on\_route\_change = route\_change

page.update()

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

Попробуем вручную обновить фрагмент URL-адреса несколько раз, затем используем кнопки «Назад» и «Вперед» браузера. При каждом изменении маршрута на странице будет добавляться новое сообщение. Это происходит благодаря обработчику событий, который реагирует на изменения маршрута и позволяет вносить соответствующие изменения на страницу. Таким образом, вы будете видеть новое сообщение на странице каждый раз, когда изменяется маршрут в URL-адресе приложения.



Маршрут может быть изменен программно по обновлению свойства page.route:

import flet as ft

def main(page: ft.Page):

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

def route\_change(e: ft.RouteChangeEvent):

page.add(ft.Text(f"New route: {e.route}"))

def go\_store(e):

page.route = "/store"

page.update()

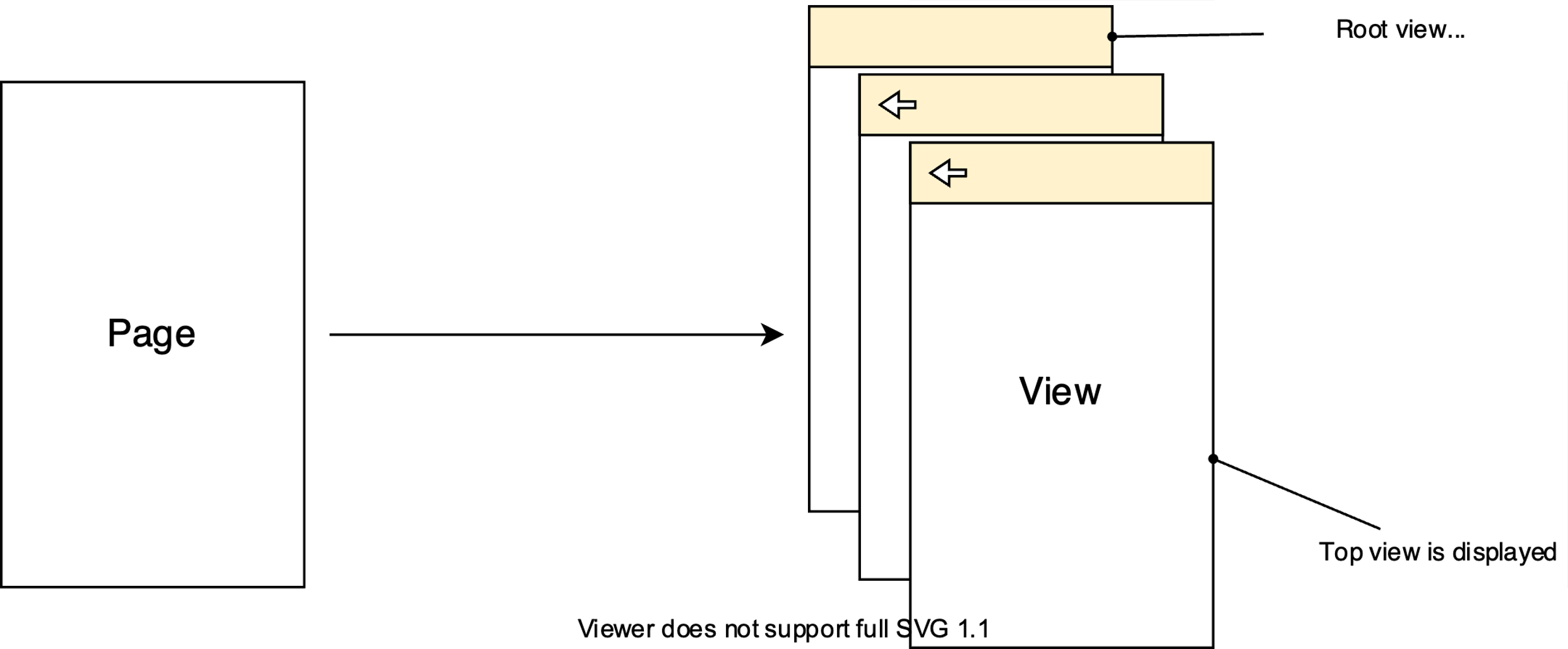
page.on\_route\_change = route\_change

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

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

Нужно нажать на кнопку «Go to store», и вы увидите, что URL-адрес приложения изменен, а в историю браузера добавлен новый элемент. Вы можете использовать кнопку браузера «Back», чтобы перейти к предыдущему маршруту.

Страница Flet представляет собой не просто одну страницу, а контейнер для просмотра, наложенный друг на друга, как сэндвич:



Коллекция представлений используется для отслеживания истории навигации на странице. Для доступа к этой коллекции представлений используется свойство page.views.

Последнее представление в этом списке является текущим представлением, которое отображается на странице в данный момент. Обязательным условием является наличие хотя бы одного элемента в списке представлений, который обычно является корневым представлением.

Для имитации перехода между страницами необходимо изменить маршрут (route) страницы и добавить новое представление в конец списка представлений страницы.

Для возврата к предыдущему представлению из списка представлений можно извлечь последнее представление и изменить маршрут на "предыдущий" в обработчике событий page.on\_view\_pop. Таким образом, будет осуществлен возврат назад по истории навигации на странице.

Чтобы обеспечить надежную навигацию в приложении, необходимо создать централизованное место, которое будет формировать список представлений в зависимости от текущего маршрута. Этот механизм может быть реализован с помощью обработчика событий page.on\_route\_change, который отслеживает изменения маршрута на странице.

Давайте рассмотрим пример, в котором у нас есть две страницы - страница "Главная" и страница "Настройки". При переходе между этими страницами, обработчик событий page.on\_route\_change будет активироваться, и в зависимости от текущего маршрута будет формировать соответствующий список представлений для отображения на странице.

Таким образом, централизованный обработчик событий page.on\_route\_change позволяет удобно управлять навигацией в приложении и адаптировать содержимое страницы в соответствии с выбранным маршрутом, обеспечивая удобство пользователя и эффективное взаимодействие с интерфейсом.

import flet as ft

def main(page: ft.Page):

page.title = "Routes Example"

def route\_change(route):

page.views.clear()

page.views.append(

ft.View(

"/",

[

ft.AppBar(title=ft.Text("Flet app"), bgcolor=ft.colors.SURFACE\_VARIANT),

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

],

)

)

if page.route == "/store":

page.views.append(

ft.View(

"/store",

[

ft.AppBar(title=ft.Text("Store"), bgcolor=ft.colors.SURFACE\_VARIANT),

ft.ElevatedButton("Go Home", 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.AppView.WEB\_BROWSER)

Flet предлагает использовать TemplateRoute, который является служебным классом на основе библиотеки repath. Этот класс позволяет создавать и сопоставлять маршруты, аналогичные тем, что используются в ExpressJS, и анализировать их параметры, например, такие как /account/:account\_id/orders/:order\_id.

TemplateRoute интегрируется отлично с событием изменения маршрута. С помощью TemplateRoute вы можете легко определять необходимые параметры маршрута и анализировать их. Это делает работу с событиями изменения маршрута более гибкой и эффективной, позволяя удобно управлять переходами между различными состояниями вашего приложения и использовать параметры маршрутов для более точной настройки поведения приложения.

troute = TemplateRoute(page.route)

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

print("Book view ID:", troute.id)

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

print("Account:", troute.account\_id, "Order:", troute.order\_id)

else:

print("Unknown route")

Для настройки стратегии URL в веб-приложениях Flet существует два способа:

Path (по умолчанию): в этом режиме пути читаются и записываются без использования хеша в URL-адресе. Например, fletapp.dev/path/to/view.

Hash: в этом режиме пути читаются и записываются с использованием хеш-фрагмента в URL-адресе. Например, fletapp.dev/#/path/to/view.

Для изменения стратегии URL необходимо использовать параметр Route\_url\_strategy при вызове метода flet.app(). Например:

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

Для управления стратегией URL-адресов в Flet Server можно использовать переменную окружения FLET\_ROUTE\_URL\_STRATEGY. Эта переменная позволяет переключаться между двумя опциями: Path (как значение по умолчанию) или Hash. Путем установки соответствующего значения этой переменной вы можете настроить предпочтительную стратегию URL-адресов для вашего сервера Flet.

## 2.4 Асинхронные приложения

Можно реализовать приложение Flet как асинхронное приложение, используя asyncio и другие асинхронные библиотеки Python. В Flet вызов сопрограмм поддерживается естественным образом, поэтому для синхронного запуска их не нужно оборачивать.

По умолчанию Flet выполняет обработчики событий управления в отдельных потоках, но иногда это может привести к неэффективному использованию процессора или к тому, что приложение не выполняет никаких действий во время ожидания ответа HTTP или в режиме sleep ().

С другой стороны, asyncio позволяет реализовать параллелизм в одном потоке путем переключения контекста выполнения между "сопрограммами". Это особенно важно для приложений, которые будут развернуты как статические веб-сайты с использованием Pyodide. Pyodide - это среда выполнения Python, основанная на WebAssembly (WASM), которая работает в браузере. На момент написания этого ответа Pyodide еще не поддерживает многопоточность.

Можно обозначить метод main() в вашем приложении Flet как асинхронный, а затем использовать любые API asyncio внутри этого метода. Это позволит вам использовать асинхронные операции, такие как создание сопрограмм, ожидание других асинхронных вызовов или выполнение параллельных операций в рамках вашего приложения. Асинхронное программирование с помощью asyncio может улучшить производительность и эффективность приложения, особенно при работе с асинхронными операциями ввода/вывода или сетевыми запросами.

import flet as ft

async def main(page: ft.Page):

await asyncio.sleep(1)

page.add(ft.Text("Hello, async world!"))

ft.app(main)

Вы можете использовать await ft.app\_async(main), если приложение Flet является частью более крупного приложения и вызывается из асинхронного кода.

Обработчики событий управления могут быть как синхронными, так и асинхронными.

Если обработчик не вызывает никаких асинхронных методов, это может быть обычный метод синхронизации:

def page\_resize(e):

print("New page size:", page.window\_width, page.window\_height)

page.on\_resize = page\_resize

Однако если обработчик вызывает асинхронную логику, он тоже должен быть асинхронным:

async def main(page: ft.Page):

async def button\_click(e):

await some\_async\_method()

page.add(ft.Text("Hello!"))

page.add(ft.ElevatedButton("Say hello!", on\_click=button\_click))

ft.app(main)

В Python отсутствуют асинхронные лямбда-выражения. Однако вполне допустимо иметь обработчик событий в виде лямбда-функции в асинхронном приложении для решения простых задач. Такой подход может быть удобен для обработки быстрых и простых операций, когда не требуется создание отдельной функции или сопрограммы. Однако, если требуется выполнение более сложных или длительных операций, рекомендуется создать отдельную асинхронную функцию или сопрограмму для более эффективного управления асинхронными задачами в вашем приложении.

page.on\_error = lambda e: print("Page error:", e.data)

Python не поддерживает асинхронные лямбда-выражения, поэтому лучше всего в асинхронных приложениях использовать асинхронные обработчики событий. Такой подход позволит вам более гибко и эффективно управлять асинхронными операциями и задачами без ограничений, которые могут возникнуть при попытке использовать асинхронные лямбда-функции.

Чтобы отложить выполнение кода в асинхронном приложении Flet, вам следует использовать asyncio.sleep() вместо time.sleep(), например:

import asyncio

import flet as ft

def main(page: ft.Page):

async def button\_click(e):

await asyncio.sleep(1)

page.add(ft.Text("Hello!"))

page.add(

ft.ElevatedButton("Say hello with delay!", on\_click=button\_click)

)

ft.app(main)

Чтобы запустить что-то в фоновом режиме, можно использовать page.run\_task(). Например, пользовательский элемент управления «Обратный отсчет», который автоматически обновляется в фоновом режиме, можно реализовать следующим образом:

import asyncio

import flet as ft

class Countdown(ft.Text):

def \_\_init\_\_(self, seconds):

super().\_\_init\_\_()

self.seconds = seconds

def did\_mount(self):

self.running = True

self.page.run\_task(self.update\_timer)

def will\_unmount(self):

self.running = False

async def update\_timer(self):

while self.seconds and self.running:

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

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

self.update()

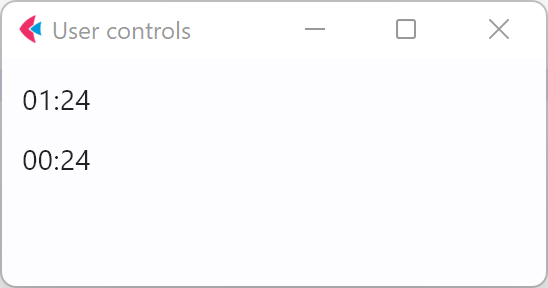
await asyncio.sleep(1)

self.seconds -= 1

def main(page: ft.Page):

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

ft.app(main)



# Глава 3: Создание веб приложения Todo app на Flet

## 3.1 Приступая к работе

Для создания веб-приложения Flet вам не требуются знания HTML, CSS или JavaScript, но важно иметь базовые навыки программирования на Python и понимание объектно-ориентированного подхода к разработке. Для работы с Flet вам необходим Python версии 3.8 или выше. Прежде чем создавать веб-приложение на Python с использованием Flet, сначала необходимо установить модуль flet.

pip install flet

Для начала давайте создадим простое приложение hello-world.

Создайте его можно hello.py со следующим содержимым:

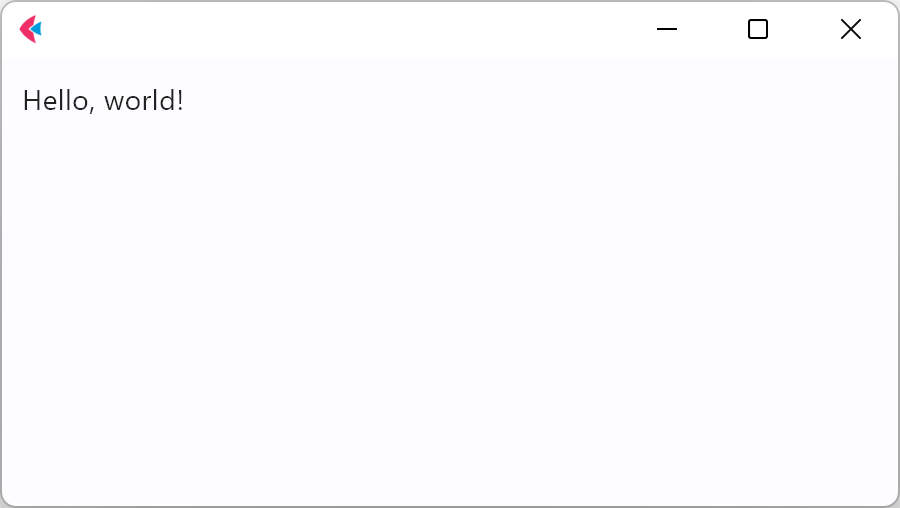
import flet as ft

def main(page: ft.Page):

page.add(ft.Text(value="Hello, world!"))

ft.app(target=main)

Вы увидите новое окно с приветствием, когда запустите данное приложение.



## 3.2 Добавление элементов управления страницей и обработка событий

Для начала создадим файл todo.py с необходимым функционалом: TextField для ввода названия задачи и FloatingActionButton с символом «+», который при нажатии будет добавлять новую задачу в список.

import flet as ft

def main(page: ft.Page):

def add\_clicked(e):

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

new\_task.value = ""

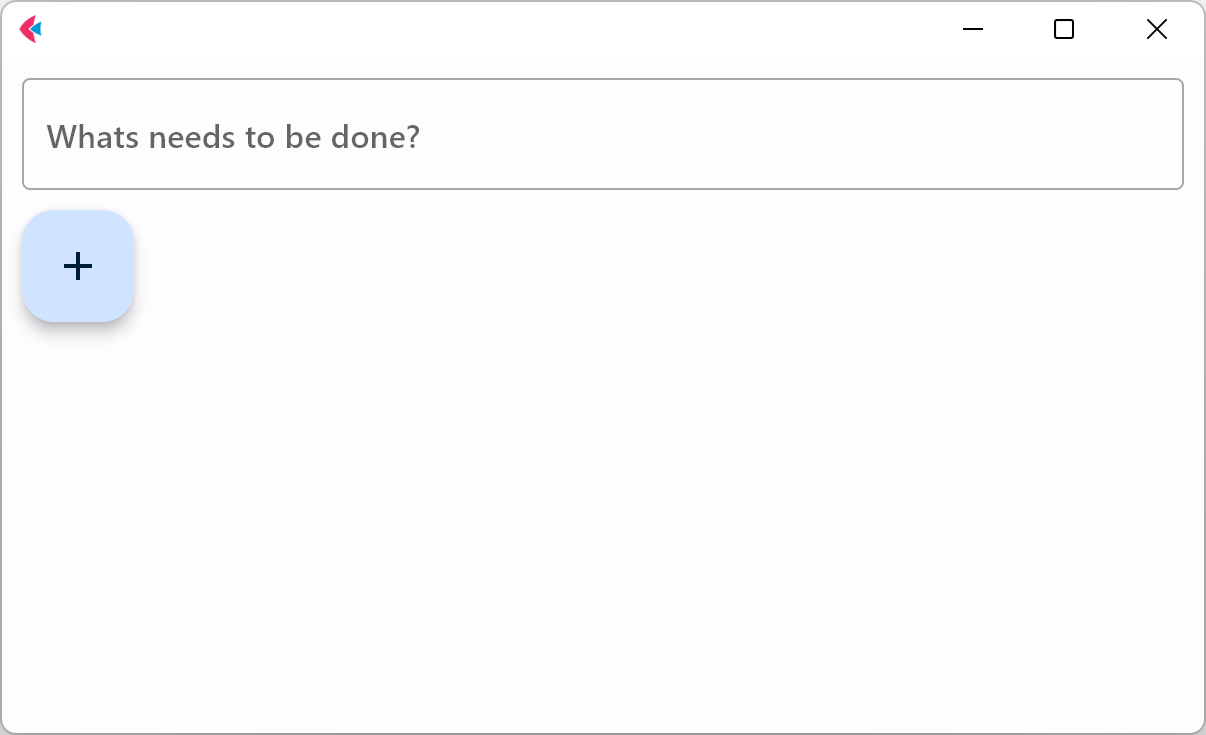
page.update()

new\_task = ft.TextField(hint\_text="What's needs to be done?")

page.add(new\_task, ft.FloatingActionButton(icon=ft.icons.ADD, on\_click=add\_clicked))

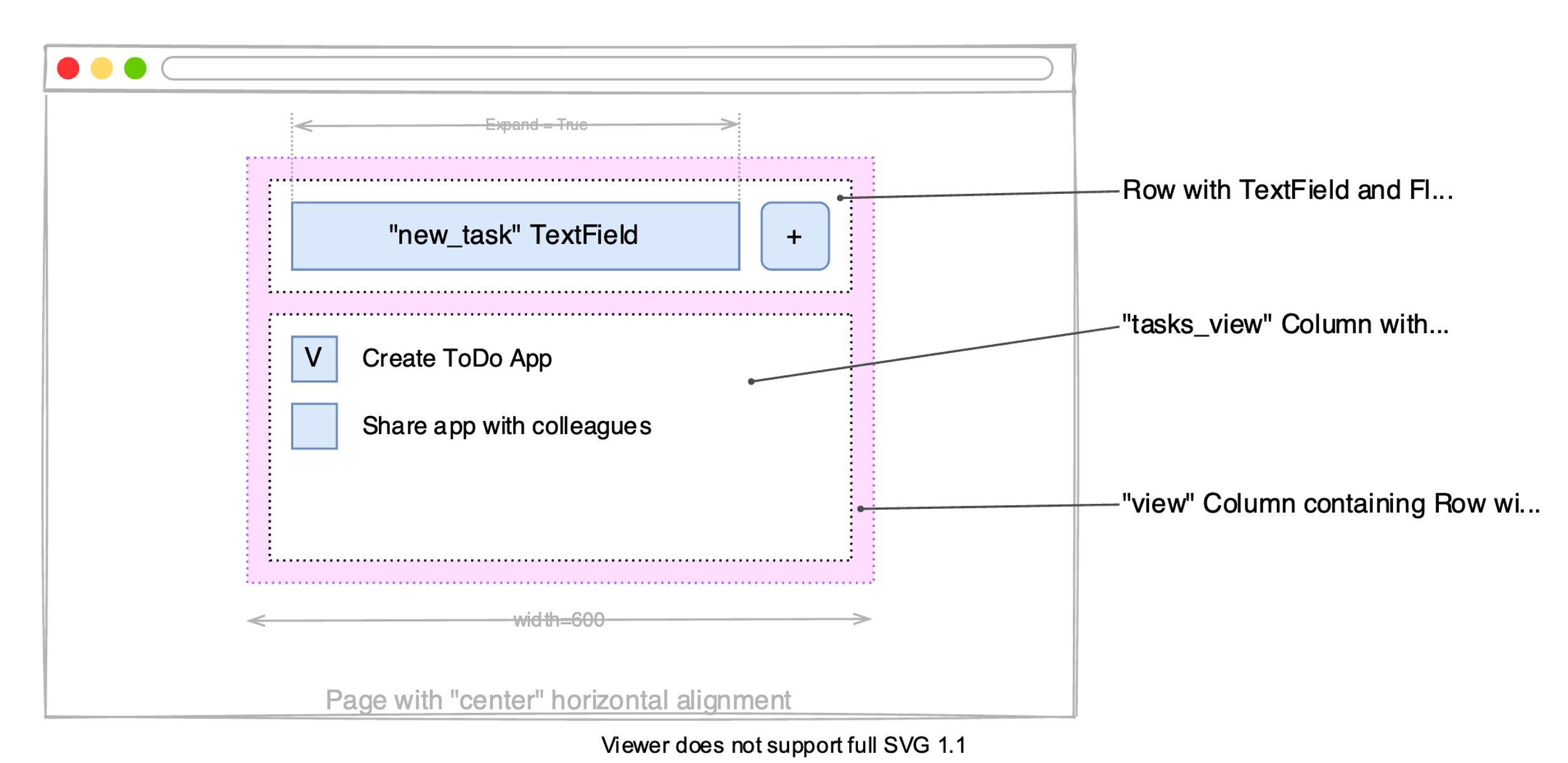
ft.app(target=main)

После запуска приложения, мы увидим следующее:



## 3.3 Макет страницы

Для улучшения внешнего вида приложения мы хотим сделать его центрированным в верхней части страницы и шириной 600 пикселей. TextField и кнопка «+» должны быть выровнены горизонтально и занимать всю доступную ширину приложения.



Row — это элемент управления, который используется для горизонтального размещения дочерних элементов на странице. Column — это элемент управления, который используется для вертикального расположения дочерних элементов управления на странице.

Замените содержимое todo.py следующим:

import flet as ft

def main(page: ft.Page):

def add\_clicked(e):

tasks\_view.controls.append(ft.Checkbox(label=new\_task.value))

new\_task.value = ""

view.update()

new\_task = ft.TextField(hint\_text="What's needs to be done?", expand=True)

tasks\_view = ft.Column()

view=ft.Column(

width=600,

controls=[

ft.Row(

controls=[

new\_task,

ft.FloatingActionButton(icon=ft.icons.ADD, on\_click=add\_clicked),

],

),

tasks\_view,

],

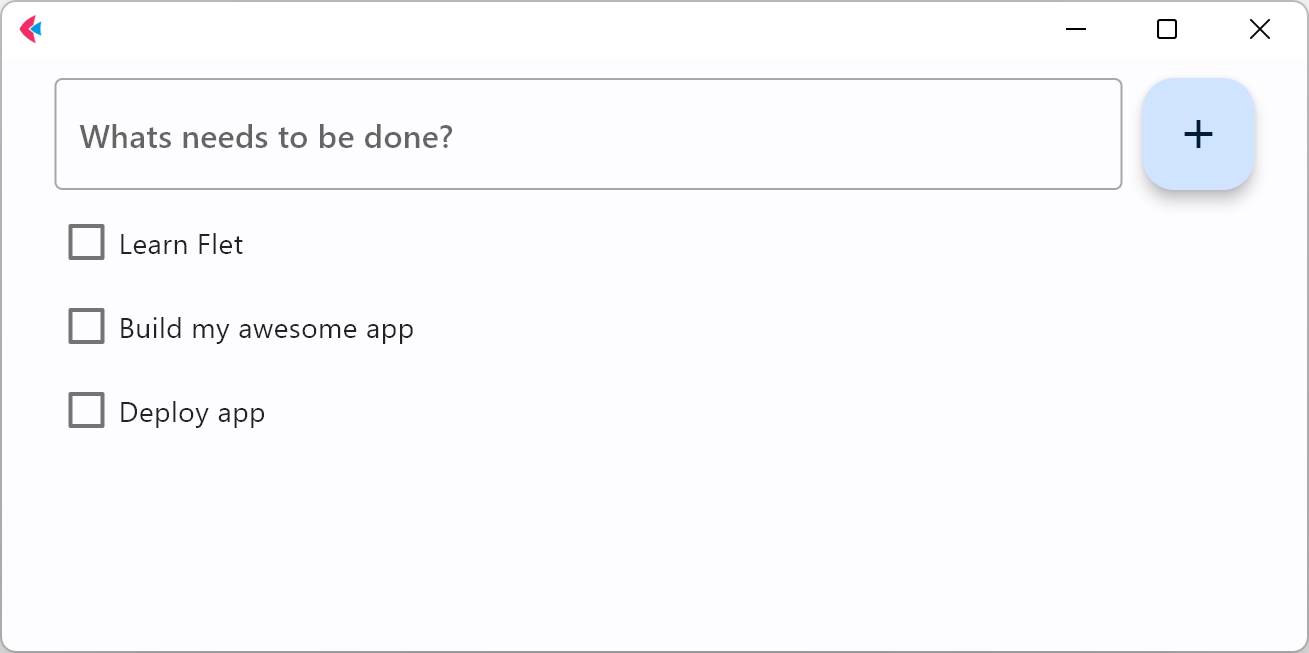
)

page.horizontal\_alignment = ft.CrossAxisAlignment.CENTER

page.add(view)

ft.app(target=main)

Запустим приложение, и мы увидим такую страницу:



## 3.4 Многоразовые компоненты пользовательского интерфейса

Хотя мы могли бы продолжать писать наше приложение в основной функции, лучше всего было бы создать повторно используемый компонент пользовательского интерфейса. Представьте, что вы работаете над заголовком приложения, боковым меню или пользовательским интерфейсом, который станет частью более крупного проекта. Мы настоятельно рекомендуем разрабатывать веб-приложения с учетом возможности создания компонуемых и пере используемых компонентов, даже если в данный момент не представляется конкретное применение для этого.

Чтобы создать повторно используемый компонент приложения ToDo, мы собираемся инкапсулировать логику его состояния и представления в отдельный класс:

import flet as ft

class TodoApp(ft.UserControl):

def build(self):

self.new\_task = ft.TextField(hint\_text="What's needs to be done?", expand=True)

self.tasks = ft.Column()

# application's root control (i.e. "view") containing all other controls

return ft.Column(

width=600,

controls=[

ft.Row(

controls=[

self.new\_task,

ft.FloatingActionButton(icon=ft.icons.ADD, on\_click=self.add\_clicked),

],

),

self.tasks,

],

)

def add\_clicked(self, e):

self.tasks.controls.append(ft.Checkbox(label=self.new\_task.value))

self.new\_task.value = ""

self.update()

def main(page: ft.Page):

page.title = "ToDo App"

page.horizontal\_alignment = ft.CrossAxisAlignment.CENTER

page.update()

# create application instance

todo = TodoApp()

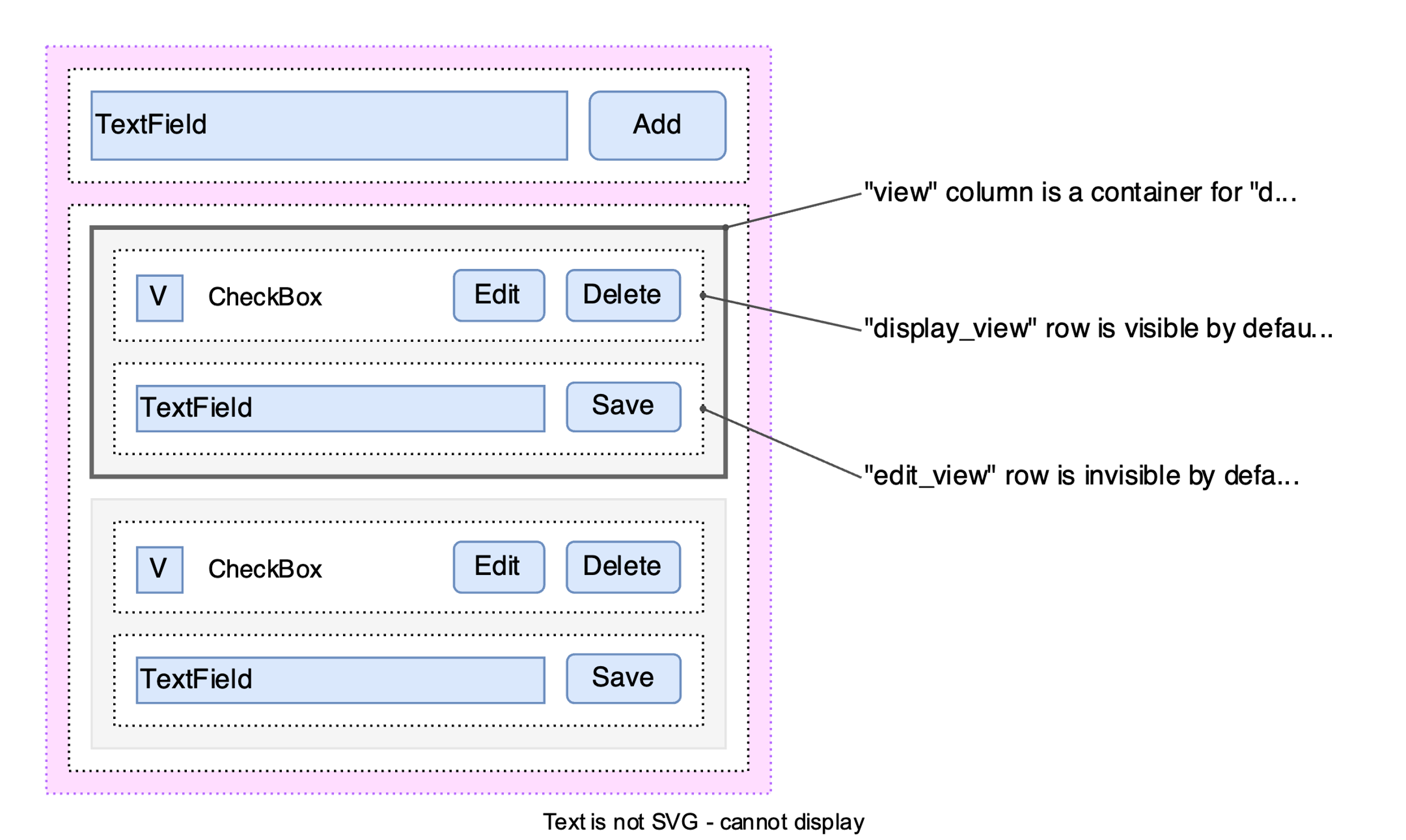
# add application's root control to the page

page.add(todo)

ft.app(target=main)

## 3.5 Просмотр, редактирование и удаление элементов списка

Давайте улучшим наше базовое приложение ToDo, добавив функционал кнопок «Изменить» и «Удалить» рядом с названием каждой задачи. При нажатии на кнопку «Изменить» будет возможность переключить элемент задачи в режим редактирования.



Каждый элемент задачи представлен двумя строками: строка display\_view с флажком, кнопками «Редактировать» и «Удалить» и строка edit\_view с текстовым полем и кнопкой «Сохранить». Столбец view служит контейнером для строк display\_view и edit\_view.

До этого момента текст программного кода был настолько компактным, что его можно было полностью включить в руководство. Однако на следующих этапах мы будем подчеркивать лишь внесенные изменения, не включая полные куски кода во избежание избыточности.

Чтобы инкапсулировать представления и действия элементов задач, мы ввели новый класс Task:

class Task(ft.UserControl):

def \_\_init\_\_(self, task\_name):

super().\_\_init\_\_()

self.task\_name = task\_name

def build(self):

self.display\_task = ft.Checkbox(value=False, label=self.task\_name)

self.edit\_name = ft.TextField(expand=1)

self.display\_view = ft.Row(

alignment=ft.MainAxisAlignment.SPACE\_BETWEEN,

vertical\_alignment=ft.CrossAxisAlignment.CENTER,

controls=[

self.display\_task,

ft.Row(

spacing=0,

controls=[

ft.IconButton(

icon=ft.icons.CREATE\_OUTLINED,

tooltip="Edit To-Do",

on\_click=self.edit\_clicked,

),

ft.IconButton(

ft.icons.DELETE\_OUTLINE,

tooltip="Delete To-Do",

on\_click=self.delete\_clicked,

),

],

),

],

)

self.edit\_view = ft.Row(

visible=False,

alignment=ft.MainAxisAlignment.SPACE\_BETWEEN,

vertical\_alignment=ft.CrossAxisAlignment.CENTER,

controls=[

self.edit\_name,

ft.IconButton(

icon=ft.icons.DONE\_OUTLINE\_OUTLINED,

icon\_color=ft.colors.GREEN,

tooltip="Update To-Do",

on\_click=self.save\_clicked,

),

],

)

return ft.Column(controls=[self.display\_view, self.edit\_view])

def edit\_clicked(self, e):

self.edit\_name.value = self.display\_task.label

self.display\_view.visible = False

self.edit\_view.visible = True

self.update()

def save\_clicked(self, e):

self.display\_task.label = self.edit\_name.value

self.display\_view.visible = True

self.edit\_view.visible = False

self.update()

Мы также внесли изменения в класс TodoApp, чтобы он мог создавать и сохранять новые задачи при нажатии кнопки "Добавить".

class TodoApp(ft.UserControl):

def build(self):

self.new\_task = ft.TextField(hint\_text="What's needs to be done?", expand=True)

self.tasks = ft.Column()

# ...

def add\_clicked(self, e):

task = Task(self.new\_task.value, self.task\_delete)

self.tasks.controls.append(task)

self.new\_task.value = ""

self.update()

Для выполнения операции удаления задачи мы добавили метод Task\_delete() в класс TodoApp. Этот метод принимает экземпляр задачи в качестве параметра.

class TodoApp(ft.UserControl):

# ...

def task\_delete(self, task):

self.tasks.controls.remove(task)

self.update()

После этого мы передали ссылку на метод Task\_delete в конструктор Task и вызвали его в обработчике событий для кнопки "Удалить".

class Task(ft.UserControl):

def \_\_init\_\_(self, task\_name, task\_delete):

super().\_\_init\_\_()

self.task\_name = task\_name

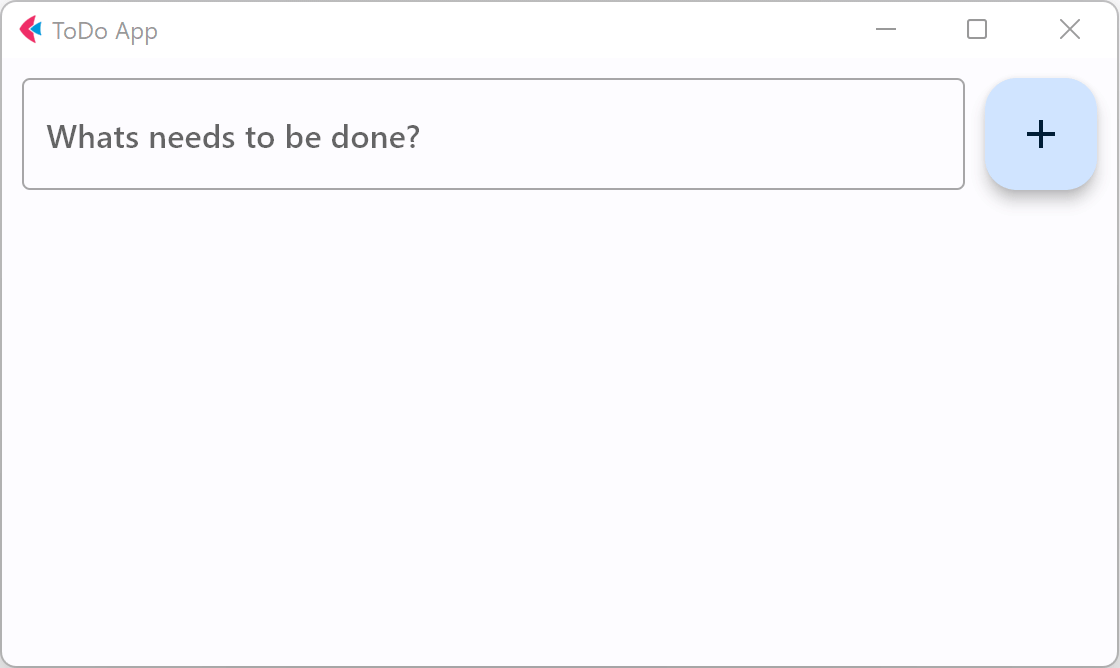
self.task\_delete = task\_delete

# ...

def delete\_clicked(self, e):

self.task\_delete(self)

После запуска приложения попробовали изменить или удалить задачи для проверки функционала.



## 3.6 Фильтрация элементов списка

Для улучшения производительности нашего функционального приложения ToDo, где мы можем создавать, редактировать и удалять задачи, мы хотим добавить возможность фильтрации задач по их статусу. Элемент управления вкладками будет использоваться для отображения выбранного фильтра задач.

class TodoApp(ft.UserControl):

def \_\_init\_\_(self):

self.tasks = []

self.new\_task = ft.TextField(hint\_text="What's needs to be done?", expand=True)

self.tasks = ft.Column()

self.filter = ft.Tabs(

selected\_index=0,

on\_change=self.tabs\_changed,

tabs=[ft.Tab(text="all"), ft.Tab(text="active"), ft.Tab(text="completed")],

)

self.view = ft.Column(

width=600,

controls=[

ft.Row(

controls=[

self.new\_task,

ft.FloatingActionButton(icon=ft.icons.ADD, on\_click=self.add\_clicked),

],

),

ft.Column(

spacing=25,

controls=[

self.filter,

self.tasks,

],

),

],

)

Для того чтобы отображать разные списки задач в зависимости от их статусов, мы приняли решение использовать простой подход: мы сохраняем один общий список задач и меняем только видимость каждой задачи в зависимости от ее статуса. В классе TodoApp мы переопределили метод update(), который перебирает все задачи и обновляет их видимость в соответствии с их статусом. Благодаря этому подходу мы можем эффективно управлять отображением задач без создания отдельных списков.

class TodoApp(ft.UserControl):

# ...

def update(self):

status = self.filter.tabs[self.filter.selected\_index].text

for task in self.tasks.controls:

task.visible = (

status == "all"

or (status == "active" and task.completed == False)

or (status == "completed" and task.completed)

)

super().update()

Фильтрация задач должна выполняться при клике на вкладку или при изменении статуса задачи. Метод update() в классе TodoApp вызывается каждый раз, когда выбрана новая вкладка или происходит изменение статуса задачи, чтобы обновить отображение задач в соответствии с выбранным фильтром.

class TodoApp(ft.UserControl):

# ...

def tabs\_changed(self, e):

self.update()

class Task(ft.UserControl):

def \_\_init\_\_(self, task\_name, task\_status\_change, task\_delete):

super().\_\_init\_\_()

self.completed = False

self.task\_name = task\_name

self.task\_status\_change = task\_status\_change

self.task\_delete = task\_delete

def build(self):

self.display\_task = ft.Checkbox(

value=False, label=self.task\_name, on\_change=self.status\_changed

)

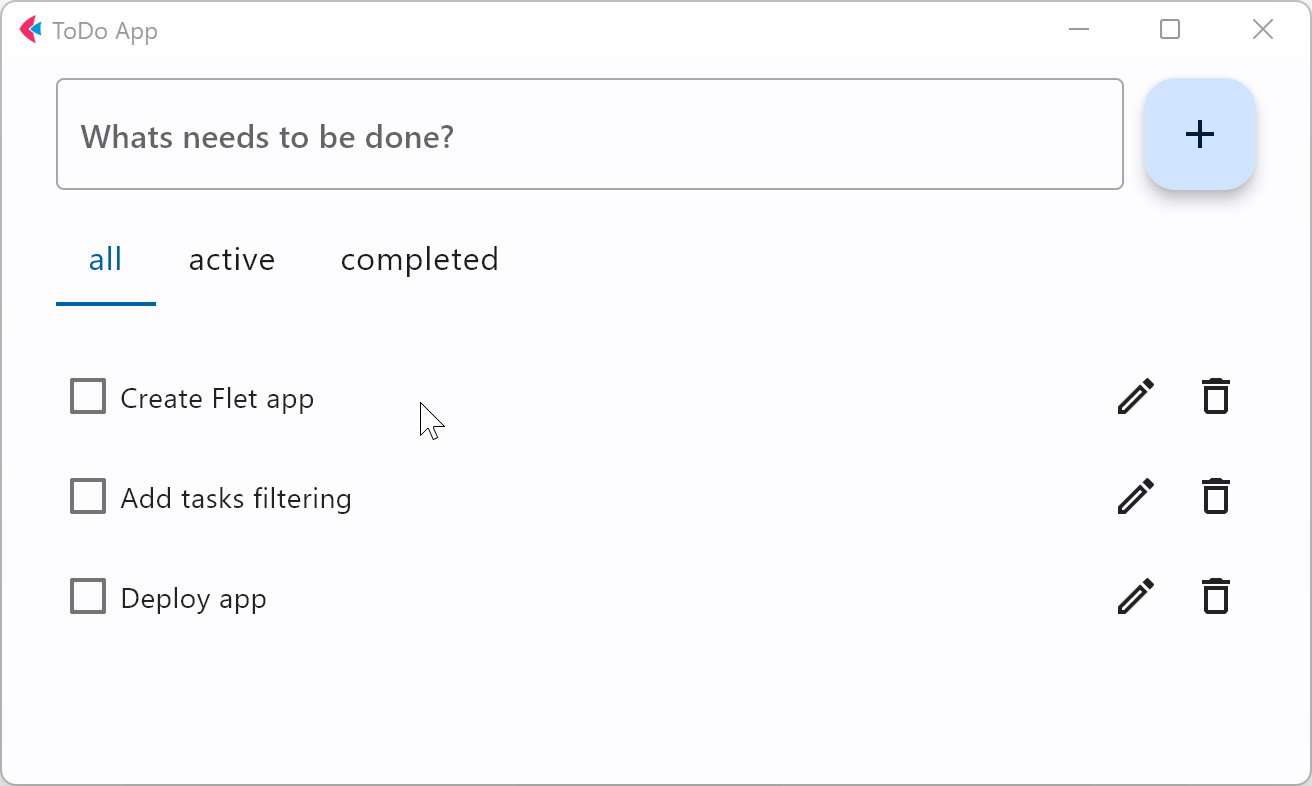
# ...

def status\_changed(self, e):

self.completed = self.display\_task.value

self.task\_status\_change(self)

Запустим приложение и попробуем фильтровать задачи, нажимая на вкладки:



## 3.7 Финальные доработки и подведение итогов

Для завершения разработки нашего приложения Todo мы хотим добавить нижний колонтитул. В нем будут отображаться количество невыполненных задач (с помощью элемента управления «Текст») и кнопка «Очистить завершенные».

class TodoApp():

def \_\_init\_\_(self):

# ...

self.items\_left = ft.Text("0 items left")

self.view = ft.Column(

width=600,

controls=[

ft.Row([ ft.Text(value="Todos", style="headlineMedium")], alignment=ft.MainAxisAlignment.CENTER),

ft.Row(

controls=[

self.new\_task,

ft.FloatingActionButton(icon=ft.icons.ADD, on\_click=self.add\_clicked),

],

),

ft.Column(

spacing=25,

controls=[

self.filter,

self.tasks,

ft.Row(

alignment=ft.MainAxisAlignment.SPACE\_BETWEEN,

vertical\_alignment=ft.CrossAxisAlignment.CENTER,

controls=[

self.items\_left,

ft.OutlinedButton(

text="Clear completed", on\_click=self.clear\_clicked

),

],

),

],

),

],

)

# ...

def clear\_clicked(self, e):

for task in self.tasks.controls[:]:

if task.completed:

self.task\_delete(task)

def update(self):

status = self.filter.tabs[self.filter.selected\_index].text

count = 0

for task in self.tasks.controls:

task.visible = (

status == "all"

or (status == "active" and task.completed == False)

or (status == "completed" and task.completed)

)

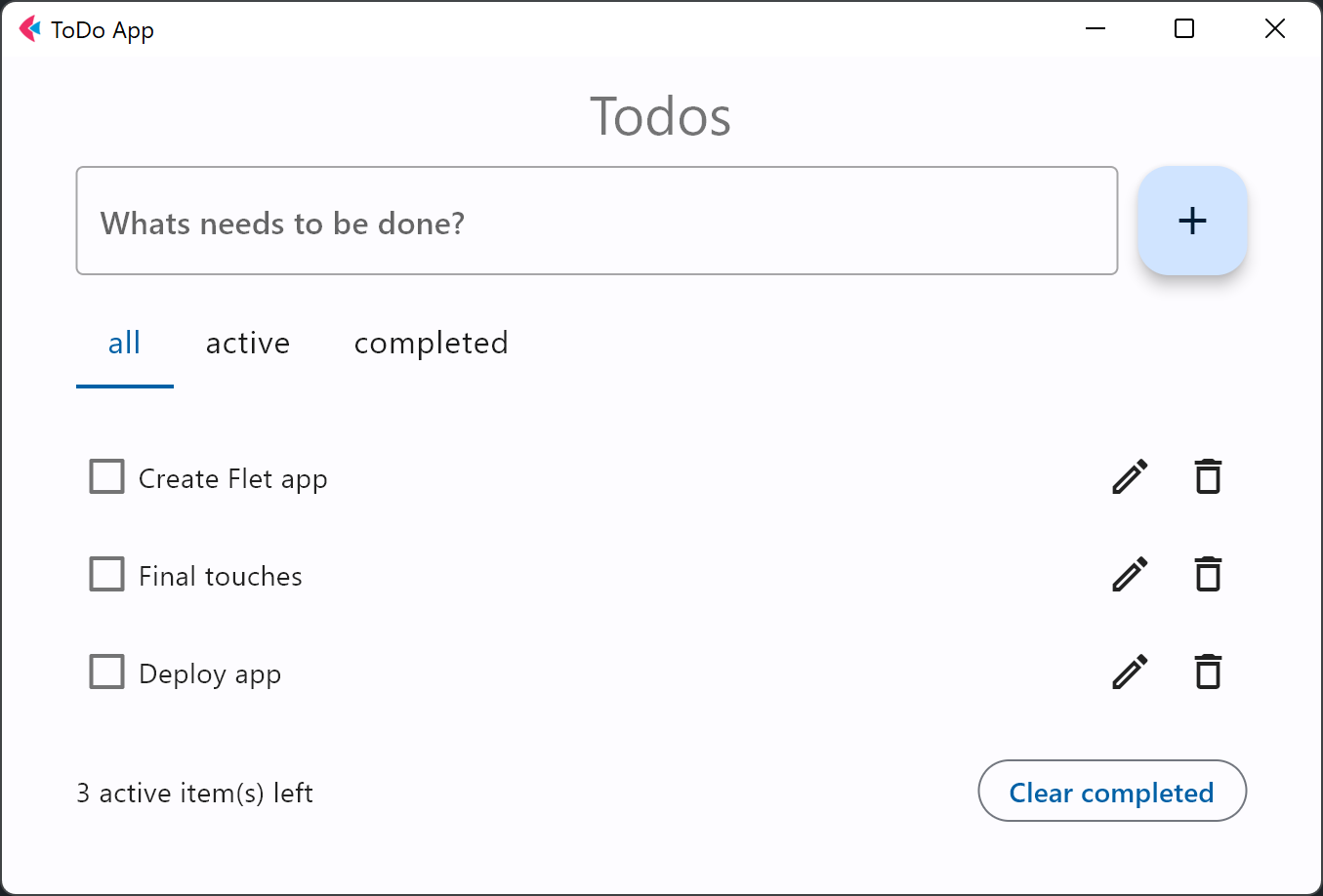
if not task.completed:

count += 1

self.items\_left.value = f"{count} active item(s) left"

super().update()

В итоге мы получаем это:



# Заключение

Знакомство с новым фреймворком позволило мне проверить свои навыки на практике и повысить уверенность в программировании. Однако при создании приложения я не смог реализовать все то, что задумывал и видел в своих планах. Некоторые важные функциональности, такие как сохранение состояния текущих задач, аутентификация, события drag and drop и многое другое, остались за пределами моих текущих возможностей. Но еще не вечер.

# Список используемой литературы

1. "HTML и CSS: Дизайн и построение веб-сайтов" Джон Даккетт - отличное руководство для начинающих веб-разработчиков, которое покрывает основы HTML и CSS.
2. "JavaScript. Подробное руководство" Дэвид Флэнаган - этот книга раскрывает все нюансы JavaScript с практическими примерами и подробным объяснением.
3. "Эффективное использование веб-стандартов. HTML5, CSS3, JavaScript" Джеффри Зельдман - в этой книге вы найдете советы по использованию современных веб-стандартов для создания профессиональных веб-сайтов.
4. "Node.js. Подробное руководство" Педро Тейвао - книга о Node.js, популярной среде выполнения JavaScript на стороне сервера.
5. "React. Быстрый старт" Азат Мардан - данное руководство поможет вам разобраться с библиотекой React и создать интерактивные веб-приложения.
6. "Решение задач с использованием Node.js, Express и MongoDB" Брэд Дэй - книга о том, как разрабатывать веб-приложения с использованием Node.js, Express и MongoDB.
7. "Как работают серверы" Дэн Холмс - в этой книге рассматриваются основы работы веб-серверов, протоколы передачи данных и другие фундаментальные концепции.
8. "CSS-анимация: Практическое руководство" Дэнни Блафмар - это руководство поможет вам освоить различные техники создания анимации с помощью CSS.
9. "Оптимизация веб-сайта: эффективные способы предотвращения роста обеспечивающих данных" Элизабетос Атьанасиадис - в книге обсуждаются методы оптимизации веб-сайтов для лучшей производительности и эффективности.
10. "Vue.js в действии" Эрик Рибланд - данное издание рассматривает популярный фреймворк Vue.js и его возможности для разработки современных веб-приложений.
11. "Учебник Bootstrap 4" Драган Кукрич - это руководство поможет вам освоить фреймворк Bootstrap для быстрой и удобной разработки адаптивных веб-сайтов.
12. "Глубокое погружение в JavaScript" Аксель Раушмайер - книга предлагает подробное изучение JavaScript, покрывая различные аспекты языка.
13. "Fullstack React: Использование React и веб-технологий для создания современных веб-приложений" Антонио Мелиндес - эта книга предлагает подробное руководство по использованию React вместе с другими веб-технологиями для создания полноценных веб-приложений.
14. "Совершенный код: Мастер-класс по разработке компонентов" Стивен Максимилиан - в этой книге вы найдете советы и практические примеры разработки компонентов веб-приложений для достижения более эффективного и качественного кода.
15. "JavaScript: Подробное руководство" Дэвид Флэнаган - эта книга охватывает широкий спектр тем по JavaScript, начиная с основ и заканчивая продвинутыми концепциями, что делает ее отличным ресурсом для изучения языка.
16. "Angular в действии" Джереми Вилкс - данное издание познакомит вас с популярным фреймворком Angular и его возможностями для разработки сложных веб-приложений.
17. <https://flet.dev/docs/>
18. <https://ru.vuejs.org>
19. <https://www.php.net/manual/ru/>
20. <https://css-tricks.com/>
21. <https://www.htmldog.com/>
22. https://javascript.info/