|  | **Wydział Finansów i Zarządzania Kierunek: Informatyka** |
| --- | --- |

**Bogusz Mędykowski**

(numer albumu: 76581)

**Tworzenie Aplikacji Webowych Z Użyciem Frameworka Flet**

Inżynierska praca projektowa

**Opiekun merytoryczny:**

**mgr inż. Wojciech Barczyński**

Wrocław 2023

[**Wstęp 3**](#_xikxr51bewo0)

[**Rozdział 1: Teoretyczne podstawy funkcjonowania frameworków do tworzenia aplikacji webowych 4**](#_n62v7afyzddl)

[1.1 Wprowadzenie do frameworków aplikacji webowych 4](#_4wi6b455h197)

[1.2. Historia rozwoju frameworków webowych 5](#_37orctymh73w)

[1.3 Kluczowe komponenty i architektura frameworków webowych 5](#_anpnm9vfbnru)

[1.5 Zalety i wyzwania stosowania frameworków w tworzeniu aplikacji webowych 7](#_6cy3mey44scs)

[**Rozdział 2: Architektura i Komponenty Frameworku Flet dla Pythona 8**](#_mnzw796fhtk1)

[2.1. Instalacja Fleta 8](#_1bkd0lor6lhv)

[2.2. Architektura Fleta 9](#_w5ya3hwo2mxe)

[2.3. Przykłady Zastosowania Komponentów Fleta 10](#_gum6hpcb6vbk)

[2.x. Deployement na przykładzie własnej aplikacji. 18](#_ljno9oklxyp3)

[2.x Domyślna estetyka warstwy wizualnej Flet 19](#_6wzckk6i5okp)

[**Rozdział 3: Analiza Porównawcza Frameworku Flet z Innymi Popularnymi Rozwiązaniami 20**](#_opqgr3g26b6d)

[**Rozdział 4: Studia Przypadków Użycia Frameworku Flet w Rzeczywistych Projektach 21**](#_7rfij37zli46)

[**\*\*Rozdział 5: Podsumowanie i Wnioski\*\* 23**](#_se4of8jaeied)

[5.1. \*\*Wnioski z Analizy Architektury Fleta:\*\* 23](#_ss6dss9wr21l)

[5.2. \*\*Wnioski z Analizy Porównawczej:\*\* 23](#_1fsyh5pw8x1o)

[5.3. \*\*Wnioski z Studiów Przypadków:\*\* 24](#_2aihcdz0f22i)

[5.4. \*\*Refleksja nad Rozwojem Fleta i Tendencjami Rynkowymi:\*\* 24](#_3me7vmeod895)

# 

# Wstęp

W obliczu dynamicznego rozwoju technologii informacyjnych, istotne staje się skuteczne tworzenie aplikacji webowych. Niniejsza praca skupia się na frameworku Flet dla Pythona, innowacyjnym narzędziu powstałym w 2021 roku, które w związku z tym stanowi stosunkowo nową bibliotekę, co sprawia, że dostępność źródeł informacji na jej temat jest ograniczona.

Motywacją do podjęcia tego tematu było doświadczenie autora, który, znając język Python, stanął przed wyzwaniem stworzenia nowoczesnej strony internetowej, niechętny korzystaniu z innych języków programowania. W tym kontekście Flet stanowił rozwiązanie dla jego problemu, łącząc elastyczność Pythona z zaawansowanymi możliwościami Fluttera.

Celem pracy jest przede wszystkim dogłębne zrozumienie architektury i funkcjonalności Fleta, z uwzględnieniem jego niedostępności źródeł informacji w literaturze ze względu na jego młody wiek. Ponadto, praca ma na celu ocenę użyteczności i efektywności Fleta w kontekście tworzenia aplikacji webowych, szczególnie w porównaniu do innych popularnych frameworków.

W zakres pracy wchodzi analiza techniczna frameworku. Metody badawcze obejmują przegląd literatury, eksperymenty programistyczne oraz analizę przypadków użycia Fleta w rzeczywistych projektach.

Struktura pracy obejmuje kilka rozdziałów. W pierwszym rozdziale przedstawione są teoretyczne podstawy funkcjonowania frameworków do tworzenia aplikacji. Drugi rozdział skupia się na architekturze i komponentach Fleta. Trzeci rozdział zawiera analizę porównawczą Fleta z innymi frameworkami. Czwarty rozdział obejmuje studia przypadków użycia Fleta. Ostatni rozdział to podsumowanie i wnioski wynikające z przeprowadzonych badań.

# Rozdział 1: Teoretyczne podstawy funkcjonowania frameworków do tworzenia aplikacji webowych

## 1.1 Wprowadzenie do frameworków aplikacji webowych

Frameworki aplikacji webowych to zestaw narzędzi i bibliotek służących do tworzenia i zarządzania stronami internetowymi oraz aplikacjami webowymi. Cechują się one standardowymi szablonami i komponentami, które usprawniają proces projektowania i programowania, redukując potrzebę pisania powtarzalnego kodu. Frameworki oferują również szeroki zakres funkcji wspierających takie aspekty, jak bezpieczeństwo, zarządzanie danymi i interakcja z użytkownikiem[1].

Podstawowymi zaletami korzystania z frameworków są:

- Zwiększona wydajność: Dzięki gotowym komponentom i szablonom, deweloperzy mogą skupić się na bardziej złożonych aspektach aplikacji, zamiast na podstawowych elementach struktury[1].

- Standaryzacja: Frameworki promują stosowanie najlepszych praktyk i wzorców projektowych, co prowadzi do tworzenia bardziej spójnego i łatwego w utrzymaniu kodu[2].

- Bezpieczeństwo: Wiele frameworków oferuje wbudowane mechanizmy bezpieczeństwa, które chronią aplikacje przed powszechnymi zagrożeniami, jak ataki SQL injection czy cross-site scripting (XSS)[3].

Frameworki aplikacji webowych są niezbędnym narzędziem w arsenale nowoczesnego programisty, umożliwiając tworzenie skomplikowanych aplikacji w sposób efektywny i bezpieczny.

1^: Freeman, Adam. "Pro ASP.NET MVC Framework." Apress, 2009.

2^: Fowler, Martin. "Patterns of Enterprise Application Architecture." Addison-Wesley, 2002.

3^: Viega, John, and Matt Messier. "Secure Programming Cookbook for C and C++." O'Reilly Media, 2003

## 

## 1.2. Historia rozwoju frameworków webowych

## 

\*\*Wczesne lata (lata 90.):\*\* Rozwój frameworków webowych rozpoczął się w latach 90., kiedy to Internet zaczął się dynamicznie rozwijać. W tym czasie powstały pierwsze narzędzia do tworzenia stron internetowych, w tym CGI (Common Gateway Interface), który umożliwiał serwerom uruchamianie skryptów po stronie serwera[1].

\*\*Rozwój wczesnych technologii (koniec lat 90.):\*\* Pojawienie się technologii takich jak ASP (Active Server Pages) od Microsoftu oraz JSP (JavaServer Pages) od Sun Microsystems znacznie rozszerzyło możliwości tworzenia dynamicznych stron internetowych[2]. Te technologie umożliwiły rozwijanie bardziej interaktywnych aplikacji webowych.

\*\*Wprowadzenie MVC (2005):\*\* W 2005 roku nastąpił przełom z wprowadzeniem Ruby on Rails, który popularizował wzorzec architektoniczny Model-View-Controller (MVC). MVC umożliwiał deweloperom oddzielenie logiki aplikacji od interfejsu użytkownika, co ułatwiło tworzenie i utrzymanie dużych aplikacji internetowych[3].

\*\*Era Full-Stack i Microframeworków (2005 - 2010):\*\* W tym okresie pojawiły się frameworki takie jak Django dla Pythona, które oferowały rozwiązania "full-stack" obejmujące zarówno frontend, jak i backend[4]. Równocześnie, na rynku pojawiły się microframeworki, takie jak Flask (Python) i Sinatra (Ruby), które oferowały lżejsze i bardziej elastyczne podejście do tworzenia aplikacji webowych[5].

\*\*Rozwój JavaScript i Single-Page Applications (2010 - obecnie):\*\* W ostatniej dekadzie, z uwagi na wzrost popularności JavaScript i rozwój Node.js, frameworki takie jak Angular, React i Vue.js zyskały na popularności. Te nowoczesne narzędzia umożliwiają tworzenie zaawansowanych Single-Page Applications (SPA), które oferują bogatsze doświadczenia użytkownika[6].

## 1.3 Kluczowe komponenty i architektura frameworków webowych

Frameworki webowe, będące kluczowymi narzędziami w projektowaniu aplikacji internetowych, składają się z kilku istotnych komponentów i są zbudowane w oparciu o różne wzorce architektoniczne.

Początki frameworków webowych sięgają wczesnych lat 90., kiedy to Tim Berners-Lee wprowadził HTML (Hypertext Markup Language), stanowiący fundament webu i pozwalający na strukturyzację dokumentów z hiperłączami. Specyfikacja HTML z 1991 roku wspierała tylko tekst i składała się z zaledwie 18 tagów. Dwa lata później został opublikowany HTML 4.0, będący pierwszą specyfikacją wspierającą CSS (Cascading Style Sheets), co stanowiło kolejny krok w rozwoju technologii webowych https://www.freecodecamp.org/news/brief-history-of-frontend-frameworks/#:~:text=Learn%20the%20history%20and%20evolution,frontend%20frameworks%20up%20until%20today】https://jellyfish.tech/blog/web-development-evolution-from-2000s-to-2020/】.

W latach 2000, web stał się główną platformą rozwoju technologicznego, z standardowym zestawem technologii takich jak HTML, CSS i JavaScript. Te technologie stanowiły podstawę dla wczesnych frameworków frontendowych https://medium.com/@Abhishek-Tiwari/the-history-and-evolution-of-web-development-from-html-to-the-modern-web-982e3f90e891#:~:text=,structuring%20of%20documents%20with%20hyperlinks】.

Jednym z najważniejszych wzorców architektonicznych w frameworkach webowych jest Model-View-Controller (MVC), który rozdziela dane aplikacji (Model), interfejs użytkownika (View) i logikę biznesową (Controller). Model zarządza danymi i logiką biznesową, View prezentuje te dane użytkownikowi, a Controller odpowiada za obsługę interakcji użytkownika z modelem. Inne popularne wzorce obejmują Model-View-ViewModel (MVVM) i Model-View-Presenter (MVP), które również zapewniają modułową i elastyczną strukturę dla aplikacji webowych.

Ewolucja frameworków webowych odzwierciedla rosnące potrzeby tworzenia bardziej złożonych i interaktywnych aplikacji internetowych. Od prostych stron HTML po zaawansowane aplikacje wykorzystujące nowoczesne frameworki, rozwój ten podkreśla ciągłe dążenie do ulepszania i innowacji w dziedzinie technologii webowych.

## 1.5 Zalety i wyzwania stosowania frameworków w tworzeniu aplikacji webowych

Korzyści wynikające z użycia frameworków aplikacji webowych, w tym Fleta, są liczne i zróżnicowane. Oto niektóre z nich:

- Szybszy rozwój aplikacji: Frameworki zapewniają gotowe komponenty i szablony, co pozwala na szybszy rozwój aplikacji, ponieważ deweloperzy nie muszą tworzyć wszystkiego od zera[1].

- Standaryzacja i struktura projektu: Frameworki promują stosowanie wzorców projektowych i najlepszych praktyk, co prowadzi do bardziej spójnej struktury projektu i ułatwia utrzymanie kodu[2].

- Wspomaganie bezpieczeństwa: Wiele frameworków oferuje wbudowane mechanizmy bezpieczeństwa, które pomagają chronić aplikacje przed atakami i zagrożeniami[3].

- Wsparcie społeczności: Popularne frameworki, takie jak Flet, cieszą się dużą społecznością deweloperów, co oznacza dostęp do bogatej bazy wiedzy i rozwiązań problemów[4].

Jednak stosowanie frameworków może również wiązać się z pewnymi wyzwaniami, takimi jak:

- Nauka i przystosowanie się: Deweloperzy muszą nauczyć się obsługi konkretnego frameworka, co może wymagać czasu i wysiłku[5].

- Mniejsza elastyczność: Niektóre frameworki narzucają pewne ograniczenia i struktury, co może być ograniczające w przypadku nietypowych projektów[^6^].

- Dodatkowy narzut czasowy: Wprowadzenie frameworka do projektu może wymagać dodatkowego czasu na konfigurację i dostosowanie[7].

Warto jednak pamiętać, że wybór frameworka zależy od konkretnego projektu i jego wymagań, a właściwa ocena korzyści i wyzwań jest niezbędna przed podjęciem decyzji o jego użyciu.

1^: Freeman, Adam. "Pro ASP.NET MVC Framework." Apress, 2009.

2^: Fowler, Martin. "Patterns of Enterprise Application Architecture." Addison-Wesley, 2002.

3^: Viega, John, and Matt Messier. "Secure Programming Cookbook for C and C++." O'Reilly Media, 2003.

7^: Freeman, Eric. "Head First Design Patterns." O'Reilly Media, 2004.

# Rozdział 2: Architektura i Komponenty Frameworku Flet dla Pythona

Flet to framework interfejsu użytkownika (UI), który umożliwia programistom tworzenie interaktywnych aplikacji internetowych, desktopowych i mobilnych wyłącznie w języku Python, bez konieczności znajomości technologii webowych, takich jak HTML, CSS czy JavaScript​​. Jego główną cechą jest wykorzystanie widżetów Fluttera, co zapewnia profesjonalny i spójny wygląd aplikacji na różnych platformach. Flet upraszcza architekturę aplikacji, eliminując potrzebę oddzielnych front-endów i back-endów, obsługuje funkcje w czasie rzeczywistym oraz dla wielu użytkowników​​ https://pypi.org/project/flet/.

## 2.1. Instalacja Fleta

W kontekście rozwoju aplikacji webowych, biblioteka Flet wyróżnia się swoją bezproblemową instalacją i brakiem potrzeby stosowania dodatkowego SDK. Flet, będąc biblioteką Pythona, jest łatwo instalowany za pomocą menedżera pakietów pip, co zapewnia szybki i efektywny proces konfiguracji. Istotnym atutem Flet jest jego natychmiastowa gotowość do użycia bez konieczności konfiguracji złożonego środowiska deweloperskiego, co czyni go atrakcyjnym wyborem dla programistów Pythona pragnących tworzyć interfejsy użytkownika w swoich aplikacjach webowych.

## 2.2. Architektura Fleta

Flet opiera swoją architekturę na zasadzie modułowości, co oznacza, że główne funkcjonalności są podzielone na mniejsze, samodzielne komponenty. Ta modularność umożliwia deweloperom bardziej efektywne zarządzanie kodem, ponieważ poszczególne części aplikacji są odseparowane i mogą być rozwijane niezależnie.

Budowa klasowa frameworku Flet dla Pythona jest zaprojektowana tak, aby umożliwić tworzenie interfejsu użytkownika w sposób imperatywny i intuicyjny. Oto kluczowe aspekty tej struktury:

- widgety Kontenerowe: Niektóre widgety w Flet, takie jak `Page` lub `Row`, działają jako kontenery, które mogą zawierać inne widgety. Na przykład, widget `Row` pozwala na ułożenie innych widgetów jeden po drugim w rzędzie【24†source】.

- Obsługa Zdarzeń: widgety w Flet mogą mieć przypisane procedury obsługi zdarzeń, które reagują na interakcje użytkownika. Na przykład, przycisk `ElevatedButton` może mieć przypisaną funkcję `on\_click`, która jest wywoływana po kliknięciu przycisku【25†source】.

- Imperatywny Model UI: Flet implementuje imperatywny model UI, co oznacza, że użytkownik "ręcznie" buduje interfejs użytkownika za pomocą stanowych widgetów, a następnie modyfikuje go, aktualizując właściwości widgetów. Jest to podejście "old-school", które może być bardziej atrakcyjne dla programistów bez doświadczenia we frontendzie【26†source】.

Ta struktura klasowa umożliwia tworzenie złożonych interfejsów użytkownika w sposób prosty i elastyczny, wykorzystując znajomość Pythona i obiektowe podejście do programowania.

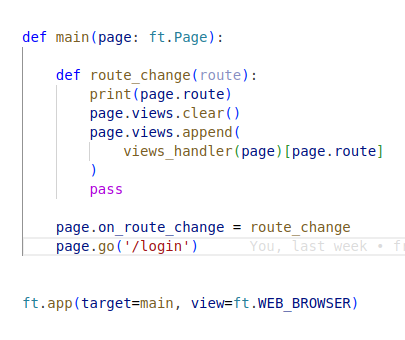
- Routery: Stanowią one fundamentalny element do obsługi trasowania żądań HTTP. Poprzez zdefiniowanie, jakie funkcje mają obsługiwać konkretne ścieżki URL, routery umożliwiają deweloperom skoncentrowanie się na logice aplikacji.

Przy analizie integracji z językiem Python warto zauważyć, że Flet wykorzystuje pełnię elastyczności i czytelności kodu tego języka. Integracja ta umożliwia programistom płynne przejście między Pythonem a Fletem, co przekłada się na spójność w tworzeniu zarówno backendu, jak i frontendu aplikacji webowych.

## 2.3. Przykłady Zastosowania Komponentów Fleta

W tym podrozdziale przybliżymy praktyczne zastosowania kluczowych komponentów frameworku Flet poprzez rzeczywiste scenariusze. Przeanalizujemy konkretny kod, który ilustruje efektywne wykorzystanie routerów i obsługi zapytań HTTP w procesie tworzenia aplikacji webowych przy użyciu Fleta.

1. \*\*Podstawowa Struktura Aplikacji\*\*: Minimalna aplikacja Flet zawiera funkcję `main`, która przyjmuje instancję `Page`. Ta funkcja stanowi punkt wejścia do aplikacji Flet i jest wywoływana w nowym wątku dla każdej sesji użytkownika. `Page` działa jak "płótno" specyficzne dla sesji użytkownika, na którym dodaje się i aktualizuje widgety【22†source】.

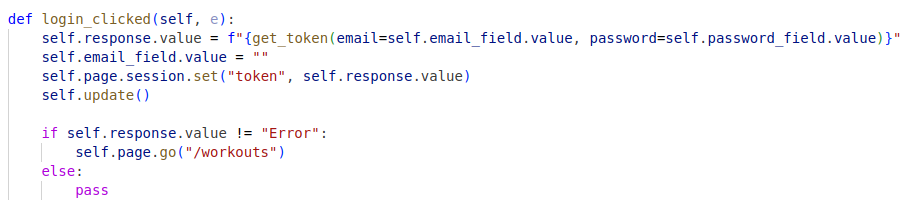


Na załączonym przykładzie po zdefiniowaniu routera. Użytkownik wysyłany jest na stronę “/login”.

`Page.session` jest właściwością obiektu `Page` w Flet, która reprezentuje sesję użytkownika. Każda sesja jest unikalna dla użytkownika i jest używana do przechowywania danych między różnymi żadądaniami w aplikacji.

Dzięki `Page.session`, możemy przechowywać dane specyficzne dla użytkownika, takie jak informacje o zalogowaniu, preferencje użytkownika, stan formularzy, itp. Sesja ta jest osobna dla każdego użytkownika, co pozwala na indywidualną obsługę.

Przykładowo, możemy używać `Page.session` do zapamiętania tokena uwierzytelniającego, który otrzymaliśmy z backendu podczas logowania:



`Page.session` w Flet stanowi potężne narzędzie do tworzenia interaktywnych, stanowych aplikacji webowych w Pythonie, umożliwiając tworzenie bogatych w funkcje aplikacji, które są dostosowane do potrzeb i preferencji poszczególnych użytkowników.

- Przykład Zastosowania Routerów:

W tym przykładzie, dzięki użyciu routera, definiujemy dwihttps://inzynierka.fly.dev e różne ścieżki URL. Pierwsza ścieżka ("/") obsługuje stronę główną aplikacji, a druga ("/workouts") pozwala na wyświetlanie listy treningów.



- Przykład Zastosowania Obsługi Zapytań HTTP:

W tym przykładzie skoncentrujmy się na obsłudze konkretnego żądania HTTP. Wykorzystamy do tego bibliotekę Pythona requests. Użytkownik przeniósł się na stronę “/workouts”. wywołało to funkcję odpytującą backend stojący na osobnym serwerze.   

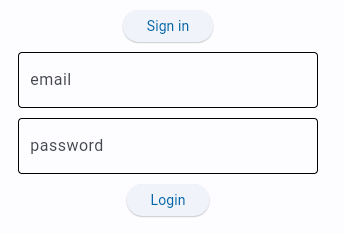

````

W tym przypadku, dzięki obsłudze żądania GET, użytkownik może odebrać dane w formie JSON, a kod obsługi żądania zajmie się dodaniem listy treningów na front-endzie.

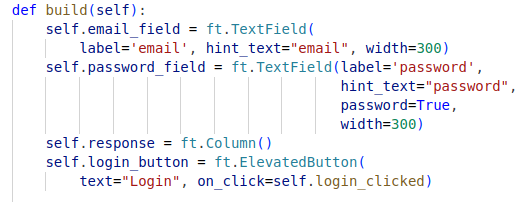
- Widgety:

Interfejs użytkownika w Flet składa się z widgetów, które są regularnymi klasami Pythona. Tworzy się instancje widgetów za pomocą konstruktorów, a następnie dodaje je do listy `controls` obiektu `Page`. Aby wyświetlić widgety na stronie, należy dodać je do `Page` i wywołać metodę `update`

Widgety Kontenerowe: Niektóre widgety w Flet, takie jak `Column` czy `Row`, działają jako kontenery, które mogą zawierać inne widgety. Na przykład, widget `Column` pozwala na ułożenie innych widgetów jeden pod drugim w kolumnie. W przykładzie poniżej umieszczono wiersze (Row), które wyświetlą się jeden pod drugim.  

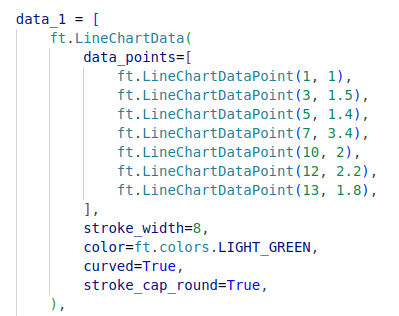



Z kolei w każdym w wierszu umieszczono przyciski wywołujące funkcje, oraz pola pobierające dane od użytkownika.



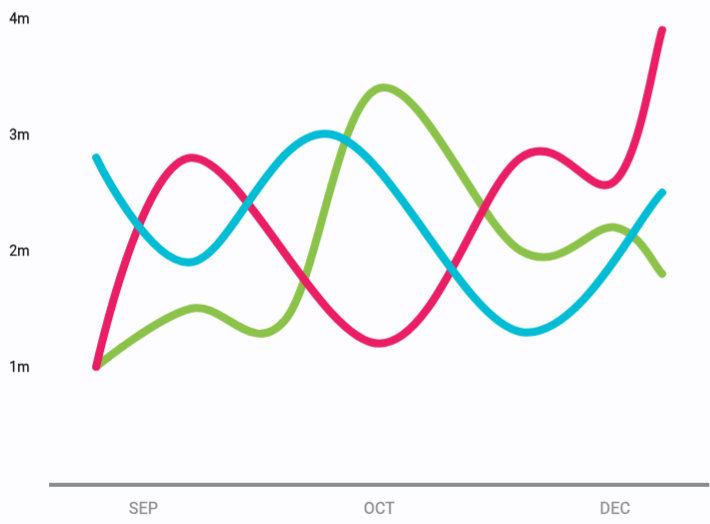
- Wykresy liniowe:

Klasa LineChartData zawiera informacje o pozycji punktów na wykresie oraz wyglądzie pojedynczej lini.

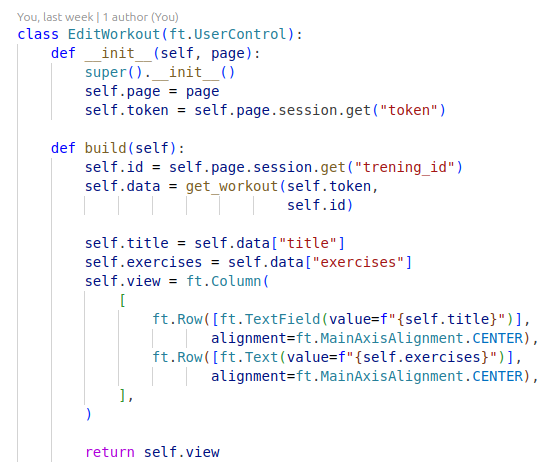


Cechy osi odciętej i rzędnej definiujemy w klasie LineChart. Efektem nowocześnie wyglądający wykres





- Tworzenie własnych widgetów do wielokrotnego wykorzystania: Używamy dziedziczenia po Page oraz UserControl. Aby zawartość mogła zaistnieć w warstwie wizualnej, potrzebna jest metoda build która zwróci zdefiniowane elementy.



## 2.x Domyślna estetyka warstwy wizualnej Flet

Material Design jest to język projektowania opracowany przez Google, który koncentruje się na zapewnieniu spójności, intuicyjności i estetyki w aplikacjach cyfrowych. Jest szeroko stosowany w aplikacjach mobilnych i stronach internetowych. W przypadku biblioteki Flet w Pythonie, Material Design odgrywa kluczową rolę, umożliwiając tworzenie aplikacji zgodnych z tymi wytycznymi.

Flet pozwala na łatwą personalizację elementów Material Design, dzięki czemu deweloperzy mogą dostosować aplikacje do specyficznych wymagań i marki.

Material Design w Flut zawiera wsparcie dla płynnych animacji i przejść, co znacząco poprawia estetykę i doświadczenie użytkownika.

Flet zapewnia bogaty zestaw gotowych do użycia widgetów Material Design, które ułatwiają i przyspieszają proces tworzenia aplikacji.

# Rozdział 3: Analiza Porównawcza Frameworku Flet z Innymi Popularnymi Rozwiązaniami

# Rozdział 4: Studia Przypadków Użycia Frameworku Flet w Rzeczywistych Projektach

W tym rozdziale skoncentrujemy się na analizie konkretnych przypadków użycia frameworku Flet w praktyce, opierając się na własnym projekcie. Poprzez studia przypadków, zbadamy różnorodność zastosowań Fleta, jego efektywność w różnych scenariuszach, a także ewentualne wyzwania, jakie deweloperzy mogą napotkać podczas implementacji.

\*Rozważania dotyczące własnego projektu:\*

Na początku tego rozdziału przedstawimy ogólne informacje o projekcie, jego cele, zakres i kontekst, aby czytelnik był w stanie lepiej zrozumieć, w jakim środowisku zastosowano framework Flet. Następnie, dla każdego przypadku użycia, przeanalizujemy konkretne wyzwania, z jakimi się spotkaliśmy, oraz korzyści płynące z wyboru Fleta do implementacji poszczególnych funkcji czy modułów.

Problem, który chciałem rozwiązać to stworzenie notatnika do treningu kulturystycznego który miałby prosty, czytelny i nowoczesny interface. Pozbawiony funkcji zbędnych funkcji. W dalszych iteracjach aplikacja mogłaby stać się portalem społecznościowym gdzie rozpisane treningi można by dzielić z innymi użytkownikami. Publikacja zdjęć i filmów oraz zostawianie komentarzy. Wymaga to od systemu skalowalności. Językiem programowania którego używałem wcześniej jest Python. Wszystko to złożyło się na wybór:  
- Django REST Framework do budowy backendu. System używa bazy danych PostgreSQL, zapewniając funkcjonalność zarządzania danymi. Backend jest hostowany na platformie Amazon Web Services (AWS), co wiąże się z wykorzystaniem usług chmurowych w infrastrukturze aplikacji. Całość systemu jest konfigurowana i uruchamiana za pomocą Docker Compose, co stanowi część procesu deploymentu.

W początkowej fazie projektowania aplikacji, rozważałem użycie Fluttera do stworzenia interfejsu użytkownika. Flutter, będąc wszechstronnym i popularnym frameworkiem do tworzenia aplikacji mobilnych i webowych, wydawał się atrakcyjnym wyborem. Jednakże, z uwagi na konieczność nauki Dart - języka programowania używanego w Flutterze - przewidywałem, że może to znacząco spowolnić postęp prac nad projektem.

W obliczu tych rozważań, podjąłem decyzję o wyborze Fleta, frameworku biblioteki dla Pythona, który lepiej wpasowywał się w moje dotychczasowe doświadczenie programistyczne. Wybór ten pozwolił mi na efektywniejsze wykorzystanie istniejących umiejętności programowania w Pythonie, co znacznie przyspieszyło proces tworzenia interfejsu użytkownika. Flet, będąc mniej znanym, ale wysoce funkcjonalnym narzędziem, umożliwił mi stworzenie responsywnego i estetycznego frontendu, jednocześnie zachowując spójność z używanym stackiem technologicznym w całej aplikacji.

## 4.x. Deployement flet

Wdrażanie aplikacji stworzonej przy użyciu biblioteki Flet na platformie Fly.io jest procesem, który wyróżnia się prostotą i efektywnością. Fly.io jest usługą hostingową zaprojektowaną do szybkiego deployowania aplikacji, a jej integracja z Flet sprawia, że proces ten jest jeszcze bardziej uproszczony. Oto krótki opis procesu wdrażania aplikacji Flet na Fly.io:

Konfiguracja dla Fly.io:

- Przed wdrożeniem aplikacji na platformę Fly.dev, istotnym krokiem jest zainstalowanie narzędzia flyctl za pomocą menedżera pakietów Pythona, pip. flyctl jest interfejsem wiersza poleceń (CLI), który umożliwia zarządzanie aplikacjami na Fly.dev.

- Aplikacja Flet wymaga stworzenia pliku konfiguracyjnego `fly.toml`, który określa ustawienia specyficzne dla środowiska produkcyjnego na Fly.io. W moim przypadku wystarczyło z edytować nazwę aplikacji resztę pliku pozostawiłem bez zmian

- W konfiguracji tej należy zdefiniować takie parametry jak wersja, zasoby (np. pamięć RAM, CPU) i inne ustawienia związane z wdrażaniem aplikacji.

Konteneryzacja:

- Fly.io wspiera konteneryzację, więc zaleca się umieszczenie aplikacji Flet w kontenerze, na przykład za pomocą Docker. Na serwerze Docker jest już zainstalowany . Oznacza to stworzenie `Dockerfile`, który definiuje jak aplikacja ma być budowana i uruchamiana w środowisku kontenera.

- Konteneryzacja zapewnia, że aplikacja będzie działać spójnie niezależnie od środowiska, co jest kluczowe dla efektywnego deploymentu.

- Wdrożenie na Fly.io: Po skonfigurowaniu aplikacji i środowiska kontenera, można użyć CLI (Command Line Interface) Fly.io do wdrożenia aplikacji. Proces ten zwykle obejmuje inicjalizację aplikacji na Fly.io, przesłanie obrazu kontenera i uruchomienie aplikacji. Po wdrożeniu, Fly.io oferuje narzędzia do monitorowania działania aplikacji i zarządzania nią, co pozwala na łatwe wprowadzanie zmian i aktualizacji.

\*Przykłady konkretnych zastosowań Fleta:\*

- System Autoryzacji i Zarządzania Sesjami:

Opiszemy, w jaki sposób Flet został użyty do stworzenia systemu autoryzacji oraz zarządzania sesjami w kontekście naszego projektu. Przeanalizujemy, jak framework ułatwił implementację mechanizmów bezpieczeństwa oraz zapewnił elastyczność w dostosowywaniu polityk autoryzacyjnych.

W omawianej aplikacji internetowej, Django pełni kluczową rolę w zarządzaniu danymi oraz użytkownikami. Jest odpowiedzialny za całościowe przechowywanie danych, w tym szczegółowych informacji o użytkownikach, ich uprawnieniach, sesjach i wszelkich innych danych związanych z funkcjonowaniem aplikacji. System autentykacji i zarządzania użytkownikami w Django jest wysoce rozwinięty, zapewniając bezpieczne i efektywne przechowywanie danych użytkowników, jak również ich weryfikację i autoryzację.

W tej architekturze, aplikacja stworzona przy użyciu biblioteki Flet działa jako klient, który komunikuje się z serwerem Django poprzez REST API. Flet nie przechowuje bezpośrednio żadnych danych użytkowników ani innych informacji, z wyjątkiem klucza API, który jest używany do uwierzytelniania i komunikacji z serwerem Django. Klucz API jest przechowywany w sesji instancji aplikacji Flet, co pozwala na bezpieczne połączenie z serwerem i dostęp do danych bez konieczności ich przechowywania na stronie klienta.

Takie podejście ma kilka kluczowych zalet. Po pierwsze, centralizuje zarządzanie danymi i bezpieczeństwem, co znacząco upraszcza procesy deweloperskie i zarządzanie infrastrukturą. Po drugie, zwiększa bezpieczeństwo danych, ponieważ wrażliwe informacje są przechowywane i przetwarzane na serwerze, co minimalizuje ryzyko ich wykradzenia z urządzeń klienckich. Ponadto, dzięki oddzieleniu logiki aplikacji klienckiej (Flet) od logiki serwerowej (Django), osiąga się większą modularność i elastyczność, co ułatwia rozwój i skalowanie poszczególnych komponentów systemu.

- Obsługa Formularzy i Walidacja Danych:

Skupimy się na zastosowaniu Fleta do obsługi formularzy i walidacji danych. Przeanalizujemy, jak framework ułatwił proces tworzenia interaktywnych formularzy, a także jak skutecznie zarządzał walidacją wprowadzanych danych.

- Komunikacja Zewnętrzna API:

Przedstawimy przypadki, w których Flet został wykorzystany do komunikacji z zewnętrznymi API. Skoncentrujemy się na elastyczności frameworku w obszarze integracji z różnymi interfejsami programistycznymi oraz jego zdolności do obsługi żądań HTTP.

\*Podsumowanie przypadków użycia:\*

Na końcu tego rozdziału dokonamy podsumowania, wyciągając wnioski dotyczące efektywności Fleta w różnych obszarach projektu. Przyjrzymy się również ewentualnym trudnościom, z jakimi się spotkaliśmy, oraz zastanowimy się nad możliwościami optymalizacji i doskonalenia naszego podejścia do wykorzystania Fleta w przyszłych projektach.

# Rozdział 5: Podsumowanie i Wnioski

W zakończeniu naszej analizy frameworka Flet dla Pythona, dokonamy kompleksowego podsumowania naszych badań, uwzględniając kluczowe aspekty architektury, analizy porównawczej z innymi frameworkami, oraz studiów przypadków z praktycznym zastosowaniem Fleta. Poniżej przedstawione są główne punkty, które zostaną poruszone w tym rozdziale:

## 5.1. Wnioski z Analizy Architektury Fleta:

Przeanalizujemy kluczowe elementy architektury Fleta, zwracając uwagę na jej zalety, takie jak modularność i elastyczność. Ocenimy, w jaki sposób Flet integruje się z językiem Python i jakie korzyści płyną z tego związku.

## 5.2. Wnioski z Analizy Porównawczej:

Dokonamy podsumowania porównania Fleta z innymi popularnymi frameworkami pod kątem wydajności, elastyczności i łatwości użycia. Zidentyfikujemy mocne strony Fleta w kontekście konkurencyjnych rozwiązań oraz potencjalne obszary do rozwoju.

## 5.3. Wnioski z Studiów Przypadków:

Podsumujemy praktyczne przypadki zastosowania Fleta, analizując różnorodność projektów, w których framework odniósł sukces. Skoncentrujemy się na efektywności Fleta w różnych kontekstach, uwzględniając zarówno zalety, jak i wyzwania.

## 5.4. Refleksja nad Rozwojem Fleta i Tendencjami Rynkowymi:

Zakończymy rozdział refleksją nad potencjalnymi kierunkami rozwoju Fleta. Przyjrzymy się również ogólnym tendencjom i innowacjom w dziedzinie frameworków do tworzenia aplikacji webowych, aby lepiej zrozumieć miejsce Fleta w dynamicznym środowisku programistycznym.