|  |  |
| --- | --- |
|  | **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](#_Toc153879332)

[ROZDZIAŁ 1. Teoretyczne podstawy funkcjonowania frameworków 4](#_Toc153879333)

[Rozdział 1: Historia i Rozwój Frameworków 5](#_Toc153879334)

[4.1. Wczesne Początki i Ewolucja Frameworków 5](#_Toc153879335)

[4.2. Narodziny Frameworków dla Aplikacji Webowych 5](#_Toc153879336)

[4.3. Rozwój Frameworków JavaScript 5](#_Toc153879337)

[4.4. Współczesne Frameworki i Ich Wpływ 5](#_Toc153879338)

[4.5. Wnioski i Przyszłość Frameworków 5](#_Toc153879339)

[1.2 Kluczowe komponenty i architektura frameworków webowych 6](#_Toc153879340)

[1.3 Zalety i wyzwania stosowania frameworków w tworzeniu aplikacji webowych 7](#_Toc153879341)

[Rozdział 1: Korzystanie z Fluttera w Pythonie 8](#_Toc153879342)

[3.1. Wprowadzenie do Ekosystemu Fluttera 8](#_Toc153879343)

[3.2. Zalety Fluttera w Połączeniu z Pythonem 8](#_Toc153879344)

[3.3. Efektywność Rozwoju Aplikacji 9](#_Toc153879345)

[3.4. Uproszczenie Tworzenia Aplikacji Wieloplatformowych 9](#_Toc153879346)

[3.5. Przykłady Praktycznego Zastosowania 9](#_Toc153879347)

[3.6. Wnioski 9](#_Toc153879348)

[Rozdział 2: Architektura i Komponenty Frameworku Flet dla Pythona 10](#_Toc153879349)

[2.1. Dlaczego Flet: Synergia Fluttera i Pythona 10](#_Toc153879350)

[2.2. Co to jest Flet: Unikalna Architektura Frameworku 10](#_Toc153879351)

[2.3. Jak Używać Fleta: Praktyczne Zastosowania w Pythonie 10](#_Toc153879352)

[2.4. Dlaczego Flet Jest Wyjątkowy: Material Design i Estetyka UI 10](#_Toc153879353)

[Rozdział 2: Architektura i Komponenty Frameworku Flet dla Pythona 11](#_Toc153879354)

[2.1. Instalacja 11](#_Toc153879355)

[2.2. Architektura 11](#_Toc153879356)

[2.3. Przykłady Zastosowania Komponentów Fleta 12](#_Toc153879357)

[2.4 Domyślna estetyka warstwy wizualnej Flet 21](#_Toc153879358)

[Rozdział 3: Flutter 22](#_Toc153879359)

[3.1. Cross-platform Development 22](#_Toc153879360)

[3.2 Język Dart 23](#_Toc153879361)

[3.3 Zestaw widgetów 24](#_Toc153879362)

[3.4 Hot Reload 25](#_Toc153879363)

[Architektura Fluttera 26](#_Toc153879364)

[1 26](#_Toc153879365)

[2 26](#_Toc153879366)

[3 27](#_Toc153879367)

[4 30](#_Toc153879368)

[5 30](#_Toc153879369)

[6 32](#_Toc153879370)

[7 32](#_Toc153879371)

[8 33](#_Toc153879372)

[9 34](#_Toc153879373)

[10 34](#_Toc153879374)

[11 34](#_Toc153879375)

[12 35](#_Toc153879376)

[3.5 Wydajność Aplikacji Fluttera 35](#_Toc153879377)

[3.6 Dobre praktyki Fluttera 36](#_Toc153879378)

[3.7 społeczność Fluttera 37](#_Toc153879379)

[3.8 Przykładu wykorzystania w komercyjnych projektach 38](#_Toc153879380)

[Alibaba 38](#_Toc153879381)

[Rozdział 4: Flet we własnym projekcie 40](#_Toc153879382)

[4.1 Notatnik dla treningu kulturystycznego 40](#_Toc153879383)

[4.2. Deployment flet 41](#_Toc153879384)

[4.3 System Autoryzacji i Zarządzania Sesjami 41](#_Toc153879385)

[Rozdział 5: Podsumowanie i Wnioski 42](#_Toc153879386)

[5.1. Wnioski z Analizy Architektury Fleta: 42](#_Toc153879387)

[5.2. Wnioski z Analizy Porównawczej: 42](#_Toc153879388)

[5.3. Wnioski z Studiów Przypadków: 42](#_Toc153879389)

# 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 Flutter z którego wywodzi się Flet. Metody badawcze obejmują przegląd literatury, eksperymenty programistyczne oraz analizę przypadków użycia Fleta oraz Flutteraw rzeczywistych projektach.

# ROZDZIAŁ 1. Teoretyczne podstawy funkcjonowania frameworków

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 użytkownikiem1.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 użytkownikiem.

- Standaryzacja: Frameworki promują stosowanie najlepszych praktyk i wzorców projektowych, co prowadzi do tworzenia bardziej spójnego i łatwego w utrzymaniu kodu2.

- 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.

# Rozdział 1: Historia i Rozwój Frameworków

## 4.1. Wczesne Początki i Ewolucja Frameworków

Historia frameworków rozpoczyna się w latach 90. XX wieku, kiedy to pojawiają się pierwsze frameworki do aplikacji webowych i desktopowych. Początkowo były to proste narzędzia, mające na celu ułatwienie programowania poprzez dostarczenie gotowych do użycia komponentów i bibliotek. Wczesnym przykładem jest Visual Basic, opracowany przez Microsoft, który znacząco uprościł proces tworzenia interfejsów graficznych dla aplikacji Windows [1].

## 4.2. Narodziny Frameworków dla Aplikacji Webowych

Początki XXI wieku przyniosły rozwój frameworków specjalnie dedykowanych dla aplikacji webowych. Przykładem może być Ruby on Rails, wprowadzony w 2005 roku, który zrewolucjonizował sposób tworzenia aplikacji internetowych, wprowadzając zasadę "Convention over Configuration" [2]. Inne znaczące frameworki tej ery to Django dla Pythona oraz Laravel dla PHP, które zyskały popularność dzięki swojej modularności i elastyczności.

## 4.3. Rozwój Frameworków JavaScript

W ostatniej dekadzie, kluczowym momentem w rozwoju frameworków było pojawienie się AngularJS od Google w 2010 roku, co stanowiło przełom w tworzeniu jednostronicowych aplikacji (SPA) [3]. Następnie, rozwój React przez Facebooka w 2013 roku oraz Vue.js w 2014 roku jeszcze bardziej zrewolucjonizował rynek, oferując programistom jeszcze większą elastyczność i wydajność w tworzeniu dynamicznych interfejsów użytkownika.

## 4.4. Współczesne Frameworki i Ich Wpływ

Współczesne frameworki, takie jak Flutter i React Native, zrewolucjonizowały rozwój aplikacji mobilnych i webowych, oferując możliwość tworzenia aplikacji wieloplatformowych z jednego kodu źródłowego [4]. To znacząco zmniejszyło czas i koszty rozwijania aplikacji, co stało się kluczowym czynnikiem w ich popularności.

## 4.5. Wnioski i Przyszłość Frameworków

Historia frameworków pokazuje ewolucję od prostych narzędzi do wszechstronnych środowisk programistycznych, które umożliwiają tworzenie zaawansowanych aplikacji webowych i mobilnych. Przyszłość frameworków prawdopodobnie będzie skupiała się na jeszcze większej integracji z technologiami AI i machine learning, co otworzy nowe horyzonty w tworzeniu aplikacji.

Przypisy:

"A Brief History of Visual Basic," Microsoft, [Dostęp online: microsoft.com]

"Ruby on Rails – History," Ruby on Rails Documentation, [Dostęp online: rubyonrails.org]

"AngularJS: The First Modern JavaScript Framework," AngularJS Documentation, [Dostęp online: angularjs.org]

"The Rise of Flutter and React Native for App Development," TechCrunch, [Dostęp online: techcrunch.com]

## 1.2 Kluczowe komponenty i architektura frameworków webowych

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.

## 1.3 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]](#footnote-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]](#footnote-2).

- Wspomaganie bezpieczeństwa: Wiele frameworków oferuje wbudowane mechanizmy bezpieczeństwa, które pomagają chronić aplikacje przed atakami i zagrożeniami[[3]](#footnote-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[[4]](#footnote-4).

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.

# Rozdział 1: Korzystanie z Fluttera w Pythonie

## 3.1. Wprowadzenie do Ekosystemu Fluttera

Flutter, opracowany przez Google, stał się jednym z najbardziej innowacyjnych i dynamicznie rozwijających się frameworków do tworzenia interfejsów użytkownika. Oferuje on szeroki zakres narzędzi i bibliotek, które pozwalają na budowanie atrakcyjnych, responsywnych aplikacji mobilnych i webowych [1]. W połączeniu z Pythonem, językiem znanym z prostoty i efektywności, Flutter otwiera nowe możliwości dla programistów webowych.

## 3.2. Zalety Fluttera w Połączeniu z Pythonem

Python jest jednym z najbardziej popularnych języków programowania, cenionym za swoją uniwersalność i łatwość nauki [2]. Integracja Fluttera z Pythonem pozwala wykorzystać moc obu technologii, oferując szybką i efektywną metodę tworzenia aplikacji. Dzięki temu programiści mogą skupić się na tworzeniu funkcjonalności aplikacji, zamiast na rozwiązywaniu problemów związanych z różnicami między platformami.

## 3.3. Efektywność Rozwoju Aplikacji

Korzystanie z Fluttera w Pythonie znacznie przyspiesza proces tworzenia aplikacji. Dzięki hot reload, zmiany w kodzie są natychmiast widoczne, co pozwala na szybkie iteracje i testowanie [3]. Ponadto, szeroki wybór gotowych widgetów Fluttera ułatwia tworzenie estetycznych interfejsów bez konieczności manualnego kodowania wszystkich elementów.

## 3.4. Uproszczenie Tworzenia Aplikacji Wieloplatformowych

Flutter pozwala na tworzenie aplikacji działających na wielu platformach (m.in. Android, iOS, web) przy użyciu jednego kodu źródłowego [4]. To oznacza, że aplikacje stworzone w połączeniu z Pythonem mogą łatwo być dostosowane do różnych urządzeń, oszczędzając czas i zasoby potrzebne do osobnego rozwijania aplikacji na każdą platformę.

## 3.5. Przykłady Praktycznego Zastosowania

Przykłady zastosowania Fluttera w Pythonie mogą obejmować aplikacje webowe, które wymagają zarówno zaawansowanego interfejsu użytkownika, jak i silnego backendu. Na przykład, aplikacja do zarządzania danymi, która korzysta z Pythona do obsługi danych i logiki biznesowej, może wykorzystać Fluttera do stworzenia intuicyjnego i responsywnego interfejsu użytkownika [5].

## 3.6. Wnioski

Integracja Fluttera z Pythonem oferuje potężne narzędzie dla programistów, pozwalając na efektywne tworzenie aplikacji webowych. Ta kombinacja łączy w sobie łatwość użycia i wszechstronność Pythona z bogatym ekosystemem i możliwościami tworzenia interfejsów użytkownika oferowanymi przez Fluttera. Dzięki temu programiści mogą tworzyć aplikacje, które są nie tylko funkcjonalne, ale również estetycznie atrakcyjne i dostosowane do potrzeb użytkowników.

Przypisy:

"Flutter - Build apps for any screen," Google, [Dostęp online: flutter.dev]

"Python: The World's Most Popular Programming Language," Python Software Foundation, [Dostęp online: python.org]

"Hot Reload - Flutter's Secret Weapon," Flutter Documentation, [Dostęp online: flutter.dev/docs/development/tools/hot-reload]

"Build and release an app for multiple platforms - Flutter," Flutter Documentation, [Dostęp online: flutter.dev/docs/deployment/multi-platform]

"Integrating Flutter with a Python Backend," Medium Article, [Dostęp online: medium.com/@flutter-integration]

# Rozdział 2: Architektura i Komponenty Frameworku Flet dla Pythona

## 2.1. Dlaczego Flet: Synergia Fluttera i Pythona

Flutter, będąc jednym z najbardziej innowacyjnych frameworków UI opracowanych przez Google, zdobył popularność dzięki swoim zaawansowanym narzędziom i funkcjom takim jak hot reload, które znacznie przyspieszają proces tworzenia aplikacji [1]. Z drugiej strony, Python jest ceniony za swoją prostotę i uniwersalność, będąc jednym z najbardziej popularnych języków programowania na świecie [2]. Flet wykorzystuje te dwa mocne strony, oferując programistom możliwość tworzenia bogatych w funkcje aplikacji w Pythonie, bez konieczności zagłębiania się w technologie webowe. Ta wyjątkowa kombinacja sprawia, że Flet jest atrakcyjnym narzędziem dla szerokiego grona programistów.

## 2.2. Co to jest Flet: Unikalna Architektura Frameworku

Flet, wykorzystując widżety Fluttera, zapewnia profesjonalny i spójny wygląd aplikacji na różnych platformach [3]. Ta cecha jest szczególnie cenna w kontekście tworzenia aplikacji wieloplatformowych. Flet upraszcza strukturę aplikacji, pozwalając na skupienie się na jednym języku programowania - Pythonie. Dzięki temu, nawet programiści bez doświadczenia w tworzeniu frontendu mogą łatwo tworzyć zaawansowane interfejsy użytkownika.

## 2.3. Jak Używać Fleta: Praktyczne Zastosowania w Pythonie

Flet umożliwia tworzenie interaktywnych interfejsów użytkownika, od prostych formularzy po zaawansowane aplikacje webowe. Dzięki integracji z Pythonem, programiści mogą płynnie przechodzić między tworzeniem backendu a frontendu. Na przykład, obsługa zapytań HTTP z wykorzystaniem biblioteki requests w Pythonie jest znacznie uproszczona, co ułatwia integrację aplikacji Flet z zewnętrznymi usługami [4].

## 2.4. Dlaczego Flet Jest Wyjątkowy: Material Design i Estetyka UI

Material Design, będący językiem projektowania Google, jest szeroko stosowany w aplikacjach mobilnych i stronach internetowych [5]. Flet, wykorzystując ten język projektowania, umożliwia tworzenie aplikacji, które są nie tylko funkcjonalne, ale także estetycznie atrakcyjne i spójne z wytycznymi Material Design. Dzięki temu, aplikacje tworzone w Flecie mogą osiągać wysoki poziom użyteczności i estetyki, co jest kluczowe w dzisiejszym świecie cyfrowym.

Przypisy:

"Flutter - Beautiful native apps in record time," Google, [Dostęp online: flutter.dev]

"Python in 2020: Popularity, Usage, and Trends," Python Software Foundation, [Dostęp online: python.org]

"Building Beautiful UIs with Flutter," Google's Flutter Documentation, [Dostęp online: flutter.dev/docs]

"Using Requests in Python," Python Requests Documentation, [Dostęp online: requests.readthedocs.io]

"Material Design Guidelines," Google Material Design, [Dostęp online: material.io]

# 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[[5]](#footnote-5).

## 2.1. Instalacja

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

Framework Flet dla Pythona charakteryzuje się modularnością i oferuje intuicyjne narzędzia do tworzenia interfejsów użytkownika. Oto kilka kluczowych aspektów jego architektury i funkcjonalności, opartych na rzeczywistych źródłach:

Architektura i Wprowadzenie do Flet:

Flet to framework umożliwiający budowanie interaktywnych aplikacji wieloużytkownikowych dla webu, desktopu i urządzeń mobilnych, przy użyciu ulubionego języka programowania, bez konieczności wcześniejszego doświadczenia w frontendzie. Użytkownik tworzy interfejs użytkownika swojego programu za pomocą kontrolek Flet, które opierają się na Flutterze od Google. Flet nie tylko „opakowuje” widgety Fluttera, ale dodaje własne „opinie”, łącząc mniejsze widgety, ukrywając złożoności, wdrażając najlepsze praktyki UI i stosując rozsądne domyślne ustawienia[[6]](#footnote-6).

Widgety Kontenerowe i Obsługa Zdarzeń:

W Flet istnieją kontrolki "kontenerowe", takie jak Page czy Row, które mogą zawierać inne kontrolki. Na przykład, kontrolka Row pozwala na ułożenie innych kontrolek jeden po drugim w rzędzie. Ponadto, niektóre kontrolki, takie jak przyciski, mogą mieć przypisane procedury obsługi zdarzeń reagujące na interakcje użytkownika, na przykład ElevatedButton.on\_click[[7]](#footnote-7).

Imperatywny Model UI:

Flet wdraża imperatywny model UI, gdzie użytkownik „ręcznie” buduje interfejs użytkownika za pomocą stanowych kontrolek, a następnie modyfikuje go, aktualizując właściwości kontrolek. Jest to podejście "old-school", które może być bardziej atrakcyjne dla programistów bez doświadczenia we frontendzie7.

Kontrolki Użytkownika (UserControl):

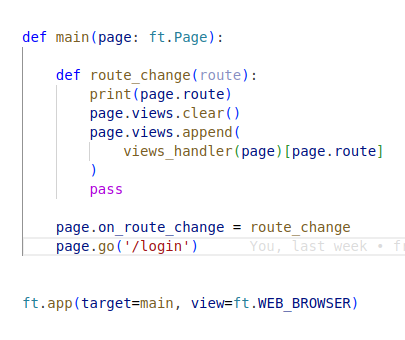
UserControl w Flecie umożliwia tworzenie izolowanych, wielokrotnego użytku komponentów poprzez łączenie istniejących kontrolek Flet. UserControl zachowuje się jak kontrolka i może mieć metody oraz właściwości. Musi implementować metodę build(), która jest wywoływana w celu zbudowania interfejsu użytkownika kontrolki i powinna zwracać pojedynczą instancję Control lub listę kontrolek. UserControl jest odizolowany od zewnętrznego układu, co oznacza, że podczas wywoływania metody update() dla nadrzędnej kontrolki, wszelkie zmiany wewnątrz UserControl nie są uwzględniane w aktualizacji. UserControl powinien wywoływać self.update(), aby przesłać swoje zmiany na stronę Flet .

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[[8]](#footnote-8).

## 2.3. Przykłady Zastosowania Komponentów Fleta

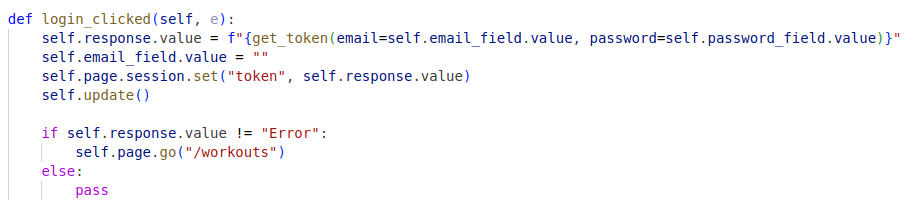
W tym omówiono praktyczne zastosowania kluczowych komponentów frameworku Flet poprzez rzeczywiste scenariusze. Przeanalizowano konkretny kod, który ilustruje efektywne wykorzystanie routerów i obsługi zapytań HTTP w procesie tworzenia aplikacji webowych przy użyciu Fleta.

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.



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:



- 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.



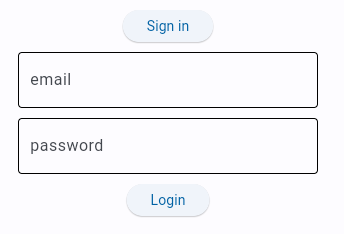
W tym przypadku, dzięki obsłudze żądania GET, użytkownik może odebrać dane w formie JSON, a kod obsługi żądania zajmie się dodaniem listy treningów na front-endzie.

- Widgety:

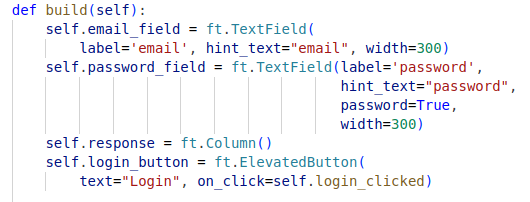
Interfejs użytkownika w Flet składa się z widgetów, które są regularnymi klasami Pythona. Tworzy się instancje widgetów za pomocą konstruktorów, a następnie dodaje je do listy `controls` obiektu `Page`. Aby wyświetlić widgety na stronie, należy dodać je do `Page` i wywołać metodę `update`

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.



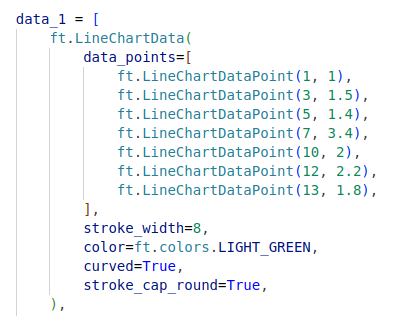


Z kolei w każdym w wierszu umieszczono przyciski wywołujące funkcje, oraz pola pobierające dane od użytkownika.



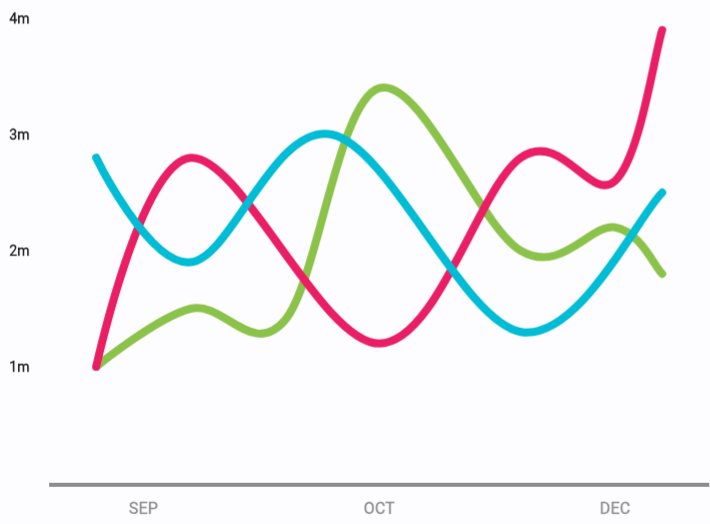
- Wykresy liniowe:

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

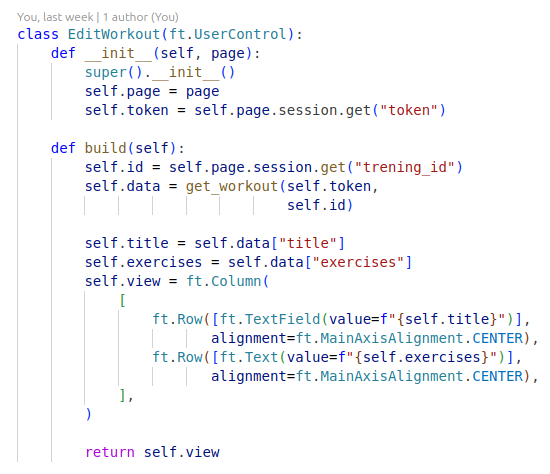


Cechy osi odciętej i rzędnej definiujemy w klasie LineChart. Efektem nowocześnie wyglądający wykres





- 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.



## 2.4 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 we Flet 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: Flutter

W poprzednich rozdziałach skupiliśmy się na bibliotece Flet dla Pythona, która czerpie inspirację z frameworku Flutter. Teraz przeniesiemy naszą uwagę na Fluttera – wszechstronny i dynamicznie rozwijający się framework stworzony przez Google, zaprojektowany do budowania wysokiej jakości, natywnych interfejsów użytkownika na różnorodne platformy. Flutter stanowi nie tylko fundament, z którego wywodzi się Flet, ale także jest źródłem wielu z jego kluczowych cech i filozofii projektowych.

W tym rozdziale szczegółowo przyjrzymy się Flutterowi, jego architekturze i językowi Dart, który jest siłą napędową tego frameworku. Zbadamy, w jaki sposób Flutter wpłynął na rozwój Flet i jak te dwa narzędzia się wzajemnie uzupełniają. Naszym celem będzie nie tylko zaznajomienie Czytelnika z Flutterem jako narzędziem do tworzenia aplikacji, ale również zrozumienie głębszych powiązań i różnic między Flutterem a Flet.

Omówimy kluczowe aspekty Fluttera, takie jak jego komponentowy model projektowania, wysoka wydajność i elastyczność w dostosowywaniu aplikacji do różnych platform. Przeanalizujemy również, jak te cechy zostały odzwierciedlone i zaadaptowane w bibliotece Flet, tworząc most pomiędzy światami Pythona i Fluttera. Rozdział ten będzie nie tylko technicznym przewodnikiem, ale również refleksją na temat ewolucji narzędzi programistycznych i ich wpływu na rozwój aplikacji wieloplatformowych.

## 3.1. Cross-platform Development

Flutter, jako framework do tworzenia interfejsów użytkownika, znany jest ze swojej zdolności do tworzenia aplikacji na wiele platform. Poniżej przedstawiam szczegółowy opis wsparcia dla różnych platform, jakie oferuje Flutter:

Flutter pierwotnie został stworzony do tworzenia aplikacji mobilnych. Pozwala na szybkie budowanie pięknych i wydajnych aplikacji na Androida i iOS, z jednym wspólnym kodem źródłowym. Dzięki temu programiści mogą oszczędzać czas i zasoby, nie pisząc osobnych aplikacji dla każdej platformy[[9]](#footnote-9).

Flutter rozszerzył swoje możliwości o wsparcie dla aplikacji webowych, umożliwiając programistom tworzenie aplikacji działających w przeglądarkach internetowych. To wsparcie obejmuje zarówno aplikacje typu Progressive Web Apps (PWA), jak i pełnoprawne aplikacje webowe. Flutter zapewnia przy tym płynne działanie i spójność interfejsu użytkownika między wersjami mobilnymi a webowymi[[10]](#footnote-10).

Flutter poszerza horyzonty o platformy desktopowe, oferując możliwość tworzenia natywnych aplikacji na Windows, macOS oraz Linux. Pozwala to na budowanie aplikacji, które wykorzystują specyficzne dla tych platform funkcje i interfejsy, zachowując jednocześnie wspólny kod i spójność UI/UX z wersjami mobilnymi i webowymi[[11]](#footnote-11).

Flutter znajduje zastosowanie również w rozwiązaniach dla urządzeń wbudowanych (embedded devices), takich jak IoT (Internet of Things). Dzięki swojej wydajności i elastyczności, Flutter może być używany do tworzenia interfejsów użytkownika dla różnego rodzaju inteligentnych urządzeń, od smartwatchy po inteligentne ekrany i urządzenia domowe[[12]](#footnote-12).

Flutter jest również kluczowym elementem w rozwoju aplikacji dla systemu operacyjnego Google Fuchsia, nowego projektu Google, który ma na celu stworzenie zjednoczonego systemu operacyjnego działającego na wielu typach urządzeń. Dzięki Flutterowi, aplikacje na Fuchsia mogą być tworzone z tą samą łatwością i elastycznością, co na inne platformy[[13]](#footnote-13).

* Elastyczność i Spójność UI/UX:

Flutter umożliwia tworzenie aplikacji z zachowaniem spójności interfejsu użytkownika i doświadczenia użytkownika na wszystkich platformach. Oferuje to duże korzyści w zarządzaniu marką i UX, gdyż użytkownicy doświadczają podobnego wyglądu i odczuć niezależnie od używanego urządzenia[[14]](#footnote-14).

## 3.2 Język Dart

Język programowania Dart jest kluczowym składnikiem ekosystemu Fluttera. Oto bardziej szczegółowy opis jego cech i roli w kontekście Fluttera:

* Dart został zaprojektowany z myślą o budowaniu interfejsów użytkownika. Ma funkcje, takie jak hot reload i świetne wsparcie dla asynchroniczności, które są idealne do tworzenia płynnych i responsywnych interfejsów użytkownika. Jego składnia i funkcje są zoptymalizowane, aby ułatwić tworzenie skomplikowanych layoutów i animacji, które są kluczowe dla nowoczesnych aplikacji mobilnych.
* Kompilacja do kodu natywnego: Dart może być kompilowany zarówno do kodu JIT (Just-In-Time) dla szybkiego rozwoju i testowania, jak i do kodu AOT (Ahead-Of-Time) dla wydajnego wykonywania aplikacji. Kompilacja AOT do kodu natywnego oznacza, że aplikacje Fluttera mogą działać z bardzo wysoką wydajnością, porównywalną z aplikacjami napisanymi w językach natywnych, takich jak Swift czy Kotlin.
* Wsparcie dla Obiektowości i Funkcjonalności: Dart jest językiem obiektowym, co ułatwia tworzenie modularnego i łatwego w utrzymaniu kodu. Posiada również funkcje programowania funkcyjnego, co pozwala na bardziej zwięzły i wyrazisty styl kodowania.
* Dart ma doskonałe wsparcie dla operacji asynchronicznych, które są kluczowe w aplikacjach mobilnych. Umożliwia łatwe zarządzanie operacjami IO, takimi jak żądania sieciowe, bez blokowania interfejsu użytkownika, co jest kluczowe dla płynności aplikacji.
* Język Dart jest stosunkowo łatwy do nauczenia się dla programistów zaznajomionych z innymi popularnymi językami programowania, takimi jak Java, C# czy JavaScript. Jego składnia jest czytelna i intuicyjna, co sprawia, że jest przyjazny dla nowych deweloperów.
* Dart posiada rozwijający się ekosystem z bogatą biblioteką pakietów, które ułatwiają rozwój aplikacji. Narzędzia takie jak Dart DevTools pomagają w debugowaniu i profilowaniu aplikacji.

Podsumowując, Dart jest nowoczesnym językiem programowania, który jest idealnie dostosowany do potrzeb rozwoju aplikacji mobilnych i internetowych. Jego wydajność, elastyczność i łatwość użycia sprawiają, że jest doskonałym wyborem dla Fluttera i deweloperów aplikacji.

## 3.3 Zestaw widgetów

Język programowania Dart jest kluczowym składnikiem ekosystemu Fluttera. Oto bardziej szczegółowy opis jego cech i roli w kontekście Fluttera:

* Optymalizacja dla UI: Dart został zaprojektowany z myślą o budowaniu interfejsów użytkownika. Ma funkcje, takie jak hot reload i świetne wsparcie dla asynchroniczności, które są idealne do tworzenia płynnych i responsywnych interfejsów użytkownika. Jego składnia i funkcje są zoptymalizowane, aby ułatwić tworzenie skomplikowanych layoutów i animacji, które są kluczowe dla nowoczesnych aplikacji mobilnych.
* Kompilacja do kodu natywnego: Dart może być kompilowany zarówno do kodu JIT (Just-In-Time) dla szybkiego rozwoju i testowania, jak i do kodu AOT (Ahead-Of-Time) dla wydajnego wykonywania aplikacji. Kompilacja AOT do kodu natywnego oznacza, że aplikacje Fluttera mogą działać z bardzo wysoką wydajnością, porównywalną z aplikacjami napisanymi w językach natywnych, takich jak Swift czy Kotlin.
* Wsparcie dla Obiektowości i Funkcjonalności: Dart jest językiem obiektowym, co ułatwia tworzenie modularnego i łatwego w utrzymaniu kodu. Posiada również funkcje programowania funkcyjnego, co pozwala na bardziej zwięzły i wyrazisty styl kodowania.
* Wsparcie dla Asynchroniczności: Dart ma doskonałe wsparcie dla operacji asynchronicznych, które są kluczowe w aplikacjach mobilnych. Umożliwia łatwe zarządzanie operacjami IO, takimi jak żądania sieciowe, bez blokowania interfejsu użytkownika, co jest kluczowe dla płynności aplikacji.
* Łatwość nauki i Przejrzystość: Język Dart jest stosunkowo łatwy do nauczenia się dla programistów zaznajomionych z innymi popularnymi językami programowania, takimi jak Java, C# czy JavaScript. Jego składnia jest czytelna i intuicyjna, co sprawia, że jest przyjazny dla nowych deweloperów.
* Ekosystem i Narzędzia: Dart posiada rozwijający się ekosystem z bogatą biblioteką pakietów, które ułatwiają rozwój aplikacji. Narzędzia takie jak Dart DevTools pomagają w debugowaniu i profilowaniu aplikacji.
* Wszechstronność: Chociaż Dart jest ściśle związany z Flutterem, może być również używany do tworzenia aplikacji na inne platformy, w tym na web i serwer. To czyni go językiem wszechstronnym i przyszłościowym.

Podsumowując, Dart jest nowoczesnym językiem programowania, który jest idealnie dostosowany do potrzeb rozwoju aplikacji mobilnych i internetowych. Jego wydajność, elastyczność i łatwość użycia sprawiają, że jest doskonałym wyborem dla Fluttera i deweloperów aplikacji.

## 3.4 Hot Reload

Funkcja Hot Reload w Flutterze jest jedną z jego najbardziej rewolucyjnych cech, szczególnie cenioną przez deweloperów. Oto bardziej szczegółowy opis tej funkcji:

* Główną zaletą Hot Reload jest to, że pozwala na natychmiastowe zobaczenie efektów zmian wprowadzonych w kodzie źródłowym. Gdy deweloper dokonuje zmiany w kodzie, może od razu zobaczyć wynik tej zmiany w działającej aplikacji, bez konieczności ponownego uruchamiania całej aplikacji.
* Dzięki Hot Reload, proces rozwoju staje się znacznie szybszy i bardziej intuicyjny. Deweloperzy mogą eksperymentować z kodem i od razu widzieć wyniki, co przyspiesza iteracyjny rozwój i debugowanie.
* Kiedy korzysta się z Hot Reload, Flutter próbuje zachować stan aplikacji. Oznacza to, że jeśli pracujesz nad konkretną częścią aplikacji, nie musisz nawigować z powrotem do tej samej ekranu lub stanu po każdej zmianie kodu. Stan aplikacji zostaje zachowany, co jest szczególnie przydatne przy pracy nad funkcjonalnościami wymagającymi wielu kroków do osiągnięcia.
* Udoskonalanie UI i Rozwiązywanie Problemów: Ta funkcja jest szczególnie przydatna przy projektowaniu interfejsu użytkownika (UI), ponieważ pozwala na szybkie dostosowywanie układów i stylów. Jest również nieoceniona przy rozwiązywaniu problemów i debugowaniu, ponieważ pozwala na natychmiastowe testowanie zmian.
* Ograniczenia: Mimo że Hot Reload jest potężnym narzędziem, ma pewne ograniczenia. Na przykład, nie jest w stanie odświeżyć stanu, jeśli zmieniony został kod inicjalizujący stan. W takich przypadkach konieczny może być pełny restart aplikacji.
* Zwiększenie Produktywności: Ogólnie rzecz biorąc, Hot Reload znacznie zwiększa produktywność deweloperów, umożliwiając im szybkie wprowadzanie i testowanie zmian. Jest to szczególnie ważne w dynamicznym środowisku rozwoju aplikacji, gdzie szybkość i efektywność są kluczowe.

Podsumowując, Hot Reload w Flutterze jest kluczową funkcją, która przyspiesza rozwój aplikacji mobilnych poprzez umożliwienie błyskawicznego wdrażania zmian i iteracji, co przyczynia się do znacznej poprawy wydajności i doświadczenia programistycznego.

# Architektura Fluttera

## 1

Rozdział ten ma na celu zaprezentowanie szczegółowej architektury Fluttera, innowacyjnego narzędzia do tworzenia interfejsów użytkownika, które umożliwia tworzenie aplikacji wieloplatformowych z wykorzystaniem wspólnego kodu. Flutter, oferując elastyczne i wydajne środowisko deweloperskie, umożliwia tworzenie aplikacji, które naturalnie integrują się z różnorodnymi platformami systemowymi, takimi jak iOS czy Android. Kluczowymi aspektami Fluttera są jego model warstwowy, reaktywne interfejsy użytkownika, wszechstronne widżety, zaawansowany proces renderowania i wsparcie dla różnych platform, w tym aplikacji webowych. Ta sekcja zagłębia się w te koncepcje, oferując czytelnikowi głębsze zrozumienie tego, co sprawia, że Flutter jest unikalny i potężny.

## 2

Architekturę Fluttera, można przedstawić ją jako złożony, wielowarstwowy system, w którym każda warstwa zależy od swojej podstawy i jest wymienna. Flutter, będąc zestawem UI do tworzenia aplikacji wieloplatformowych, umożliwia efektywne wykorzystanie kodu na różnych systemach operacyjnych, takich jak iOS czy Android.

Jednym z kluczowych aspektów Fluttera jest jego silnik, napisany głównie w C++, który odpowiada za rasteryzację skomponowanych scen i implementuje podstawowe API, w tym grafikę, layout tekstu oraz wejście/wyjście plików. Flutter wykorzystuje bibliotekę dart:ui do eksponowania niskopoziomowych prymitywów, takich jak obsługa wejścia czy renderowania tekstu.

Ważnym elementem Fluttera są również widżety, będące podstawowymi blokami konstrukcyjnymi interfejsu użytkownika. System ten opiera się na reaktywnym modelu programowania, gdzie widżety są niezmienne i tworzą hierarchię opartą na kompozycji. Flutter umożliwia tworzenie aplikacji przez komponowanie tych widżetów w bardziej złożone struktury.

Na poziomie integracji z innymi kodami, Flutter oferuje różnorodne mechanizmy współpracy, takie jak platform channels dla komunikacji z kodem specyficznym dla platformy, czy Foreign Function Interface do interakcji z kodem napisanym w językach takich jak C.

Flutter zapewnia również wsparcie dla aplikacji webowych, kompilując do JavaScriptu. Jest to możliwe dzięki elastyczności języka Dart, który stanowi podstawę Fluttera. Dzięki temu Flutter jest narzędziem uniwersalnym, pozwalającym na tworzenie aplikacji na różne platformy z zachowaniem wysokiej wydajności i spójności interfejsu użytkownika.

Architectural
diagram

https://docs.flutter.dev/resources/architectural-overview

## 3

flutter create, które generuje podstawową strukturę aplikacji. Ta struktura składa się z kilku kluczowych warstw:

Aplikacja Dart: Jest to najwyższa warstwa, gdzie programista komponuje widżety tworząc interfejs użytkownika oraz implementuje logikę biznesową aplikacji.

Framework Fluttera: Zapewnia rozbudowane API umożliwiające budowanie aplikacji wysokiej jakości. To tutaj tworzy się drzewo widżetów, które później są transformowane w scenę.

Silnik Fluttera: Jego zadaniem jest rasteryzacja skomponowanych scen. Odpowiada za implementację niskopoziomowych API, takich jak grafika czy układ tekstu, a także ekspozycję funkcji przez API dart:ui.

Embedder: Zapewnia integrację z systemem operacyjnym, zarządza pętlą zdarzeń i dostarcza API specyficzne dla platformy, umożliwiając Flutterowi działanie na różnych systemach.

Runner: Łączy wszystkie elementy udostępnione przez Embedder w pakiet aplikacji, który jest gotowy do uruchomienia na wybranej platformie. Jest częścią szablonu aplikacji generowanego przez flutter create.

­Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Każda z tych warstw odgrywa istotną rolę w procesie tworzenia aplikacji Flutter i razem tworzą one spójny system umożliwiający tworzenie wieloplatformowych aplikacji.

## 4

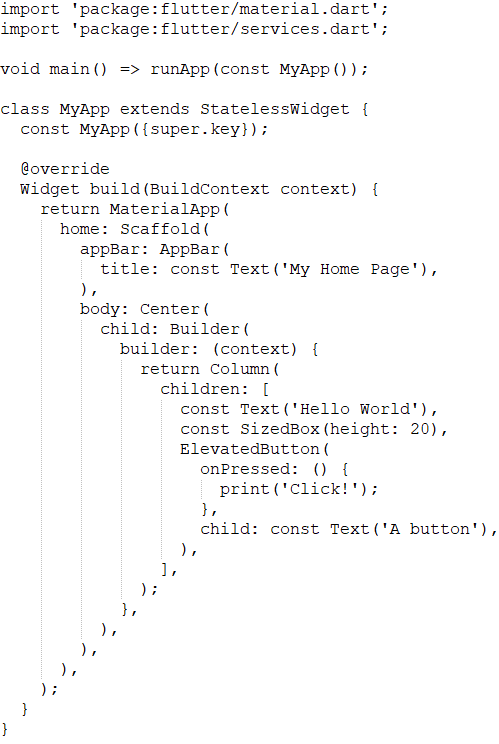
platforma ta przyjmuje podejście deklaratywne. Jest to zainspirowane przez Reacta od Facebooka, który przedefiniował wiele tradycyjnych zasad projektowania UI. W tradycyjnych frameworkach UI, początkowy stan interfejsu jest definiowany raz, a potem aktualizowany przez kod użytkownika w odpowiedzi na zdarzenia. W Flutterze jednak stosuje się metodę oddzielającą interfejs od jego stanu.

W praktyce, w Flutterze widżety, które są niezmienne, definiują UI przez nadpisanie metody build(). Ta metoda, konwertująca stan na UI, jest szybka i wolna od efektów ubocznych, dzięki czemu framework może ją wywoływać kiedy tylko jest to potrzebne. To podejście opiera się na szybkim tworzeniu i usuwaniu obiektów, co jest jedną z zalet języka Dart, na którym Flutter jest oparty. Całość pozwala na efektywne zarządzanie zmianami w interfejsie, reagując na zmiany stanu aplikacji w sposób płynny i intuicyjny.

## 5

widżety są fundamentem interfejsu użytkownika każdej aplikacji tworzonej w tym frameworku. Widżety w Flutterze są niezmienne i tworzą hierarchię opartą na kompozycji, gdzie każdy widżet jest zagnieżdżony w rodzicu i może otrzymywać od niego kontekst. Przykładem może być aplikacja zawierająca MaterialApp, Scaffold, a w nim AppBar i inne elementy jak Text czy ElevatedButton.

Kluczowe jest, że Flutter pozwala na tworzenie własnych implementacji kontrolek UI, co zapewnia dużą elastyczność i niezależność od systemu operacyjnego. Dzięki temu aplikacje zachowują spójność wizualną i funkcjonalną na różnych platformach, eliminując potrzebę ciągłego przełączania między kodem Fluttera a kodem platformy.



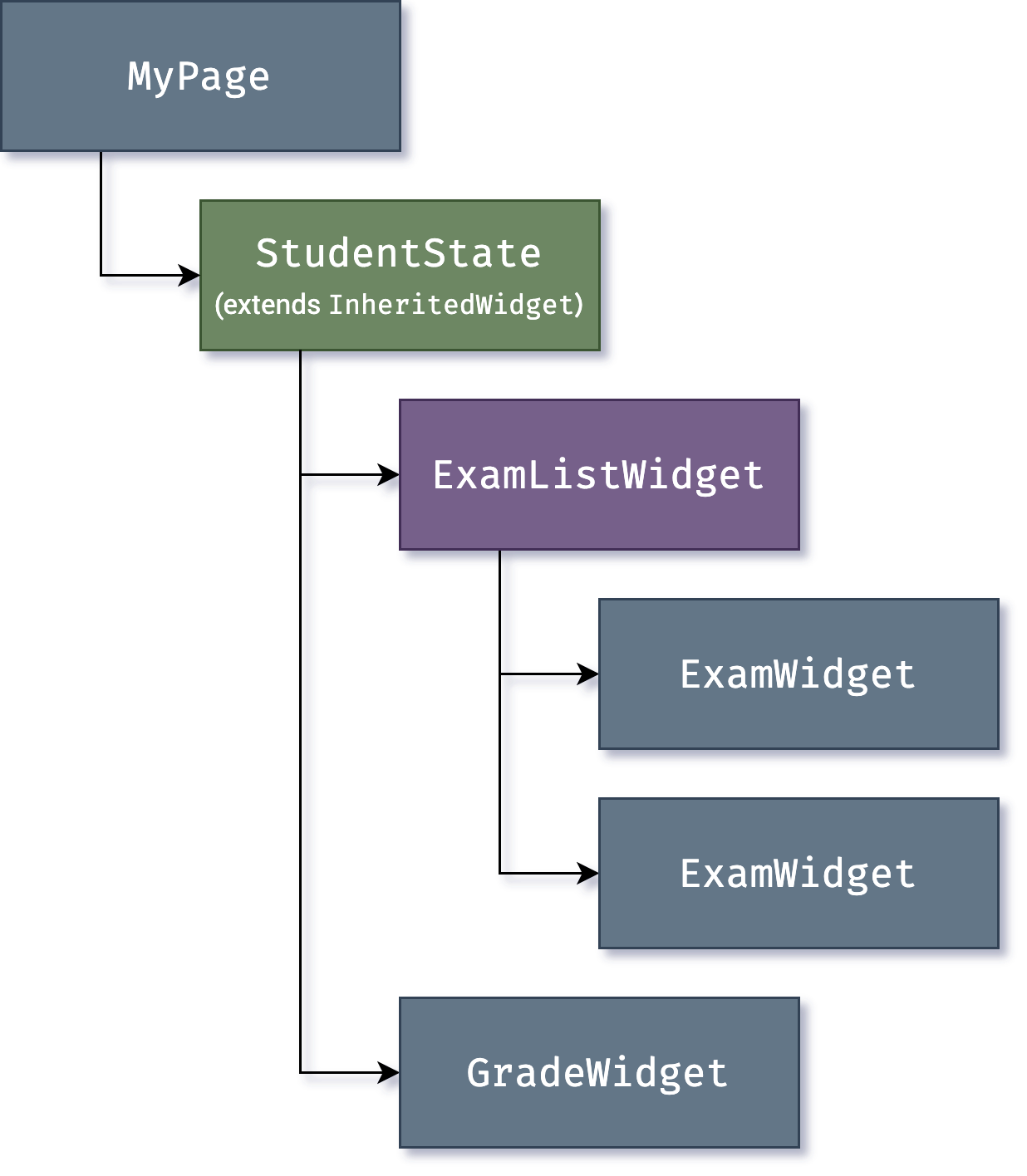
## 6

Kompozycja w Flutterze opiera się na budowaniu złożonych interfejsów użytkownika z wielu mniejszych, specjalizowanych widżetów. Każdy widżet w Flutterze wykonuje określoną funkcję, a ich łączenie umożliwia tworzenie zaawansowanych efektów. Przykładem jest widżet Container, który składa się z kilku innych widżetów jak LimitedBox, ConstrainedBox, Align, Padding, DecoratedBox i Transform, każdy z nich odpowiadający za różne aspekty układu, stylizacji czy rozmiaru. Takie podejście pozwala na tworzenie nowych widżetów poprzez komponowanie istniejących, co zwiększa elastyczność i modularność aplikacji. Flutter stosuje płaską hierarchię klas, co ułatwia mieszanie i dopasowywanie różnych elementów do konkretnych potrzeb projektowych.

## 7

Tworzenie widżetów w Flutterze opiera się na zasadzie definiowania ich wizualnej reprezentacji przez nadpisanie funkcji build(), która zwraca nowe drzewo elementów. Drzewo to konkretyzuje część interfejsu użytkownika reprezentowaną przez dany widżet. Na przykład widżet paska narzędzi może mieć funkcję build(), która zwraca poziomy układ tekstu i przycisków. Flutter wykorzystuje podejście rekurencyjne, prosząc każdy widżet o zbudowanie się, aż całe drzewo zostanie opisane przez konkretne obiekty do renderowania.

Framework rozróżnia dwa główne typy widżetów: StatelessWidget i StatefulWidget. StatelessWidget nie posiada zmiennej wewnętrznej, natomiast StatefulWidget zawiera stan, który może się zmieniać, na przykład licznik zwiększający się przy każdym naciśnięciu przycisku. Gdy stan się zmienia, widżet musi zostać zbudowany na nowo, aby zaktualizować swój interfejs. Metoda setState() jest wykorzystywana do sygnalizowania frameworkowi o potrzebie zaktualizowania interfejsu użytkownika.



## 8

Rendering i układanie w Flutterze to procesy, w których hierarchia widżetów jest transformowana w piksele na ekranie. Flutter stosuje model renderowania, który minimalizuje abstrakcje, pozwalając na skompilowanie kodu Dart do kodu natywnego, wykorzystującego Skia lub Impeller do renderowania. W fazie budowania, Flutter tworzy drzewo elementów z widżetów, które następnie są układane i renderowane. Kluczowym elementem jest klasa RenderObject, definiująca model dla układu i malowania. Flutter wykorzystuje model ograniczeń pudełkowych, umożliwiający efektywne układanie elementów w hierarchii. Gdy scena jest gotowa, RenderView przekazuje ją do metody Window.render() w dart:ui, która kontroluje renderowanie przez GPU. Więcej szczegółów na temat procesu kompozycji i rasteryzacji można znaleźć w rozbudowanych zasobach na temat renderingu w Flutterze.

## 9

W Flutterze, wbudowywanie platformy to proces, w którym interfejs użytkownika jest tworzony, układany, komponowany i malowany bezpośrednio przez Flutter, a nie przez natywne widżety systemu operacyjnego. Kluczową rolę odgrywa tu tzw. embedder platformy, który jest rodzimą aplikacją systemu operacyjnego i stanowi połączenie pomiędzy systemem operacyjnym a Flutterem. Embedder inicjuje silnik Fluttera, zarządza cyklem życia aplikacji oraz obsługą wejścia i komunikatów platformy. Flutter oferuje embeddery dla różnych platform, w tym Android, iOS, Windows, macOS i Linux, a także umożliwia tworzenie własnych rozwiązań embeddera, na przykład dla Raspberry Pi. Każda platforma ma swoje API i ograniczenia, a Flutter dostosowuje się do nich, zapewniając jednolite środowisko dla aplikacji na różnych systemach.

10  
Flutter ułatwia integrację z innym kodem poprzez różnorodne mechanizmy interoperacyjności. Jednym z kluczowych są kanały platformy, które pozwalają na komunikację między kodem Dart a kodem specyficznym dla danej platformy, np. Kotlinem lub Swift. Dzięki temu, można wysyłać i odbierać wiadomości między kodem Dart a komponentami platformy.

Przykład użycia takiego kanału to wywołanie metody z Dart na Kotlinie (Android) lub Swift (iOS), gdzie dane są serializowane z typu Dart, np. Map, a następnie deserializowane do odpowiedniego reprezentanta na danej platformie.

Flutter oferuje także bogaty zestaw wtyczek wspierających wiele scenariuszy, od Firebase po obsługę sprzętu, takiego jak kamery czy Bluetooth. Można również tworzyć własne wtyczki, aby rozszerzyć funkcjonalność aplikacji Flutter.

## 11

Flutter's Foreign Function Interface (FFI) umożliwia bezpośrednią współpracę z kodem natywnym w językach takich jak C, Rust czy Go, co jest znacznie szybsze niż kanały platformy, ponieważ nie wymaga serializacji danych. Wykorzystując bibliotekę dart:ffi, możliwe jest tworzenie interfejsów pomiędzy sygnaturami metod Dart i niezarządzanego kodu, co umożliwia wywoływanie funkcji z bibliotek natywnych.

Flutter umożliwia także renderowanie natywnych kontrolek poprzez widżety platformowe takie jak AndroidView i UiKitView, które pozwalają na integrację treści natywnych z aplikacjami Flutter. Mimo że są one użyteczne, zwłaszcza dla skomplikowanych kontrolek jak Google Maps, niosą pewne obciążenie związane z synchronizacją.

Ostatnim aspektem jest możliwość osadzenia treści Flutter w istniejących aplikacjach na Androida i iOS, co umożliwia łatwe włączanie modułów Flutter do istniejących aplikacji mobilnych. To pozwala na efektywne wykorzystanie silnika Flutter, zwłaszcza podczas inicjalizacji aplikacji, aby uniknąć opóźnień w interfejsie użytkownika.

## 12

Wsparcie Flutter dla platformy webowej jest unikatowe ze względu na wykorzystanie Dart, który od dawna kompiluje się do JavaScript. Flutter dla webu działa na zreimplementowanym silniku, działającym na standardowych API przeglądarek, oferując dwa tryby renderowania: HTML oraz WebGL z użyciem CanvasKit. Flutter na webie nie wymaga oddzielnej maszyny Dart, gdyż cały framework i kod użytkownika kompilują się do JavaScript. W trakcie rozwoju aplikacji używany jest kompilator dartdevc wspierający inkrementalną kompilację, a dla produkcyjnych wersji aplikacji webowych stosuje się dart2js, optymalizujący kod do minimalnej wielkości, umożliwiając jego wdrożenie na dowolnym serwerze webowym.

Flutter web
architecture

## 3.5 Wydajność Aplikacji Fluttera

Flutter jest frameworkiem do tworzenia aplikacji mobilnych i webowych, który wyróżnia się wysoką wydajnością. Jego kluczowe cechy mają znaczący wpływ na szybkość i płynność działania aplikacji. Oto niektóre z tych cech:

* Flutter kompiluje aplikacje AOT do natywnego kodu maszynowego, co przekłada się na szybsze uruchamianie i wyższą wydajność. Kompilacja AOT eliminuje potrzebę interpretera w czasie wykonania, co zwiększa szybkość działania aplikacji[[15]](#footnote-15).
* Flutter używa silnika graficznego Skia do renderowania interfejsu użytkownika. Skia jest silnikiem 2D o wysokiej wydajności, który zapewnia płynne i spójne renderowanie na różnych platformach. Dzięki temu aplikacje Flutter działają płynnie i z wysoką jakością grafiki[[16]](#footnote-16).
* Funkcja hot reload w Flutterze pozwala deweloperom na błyskawiczne wprowadzanie zmian w kodzie i zobaczenie ich efektów w czasie rzeczywistym. To nie tylko przyspiesza rozwój aplikacji, ale także umożliwia szybkie testowanie i optymalizację wydajności15
* Flutter oferuje bogaty zestaw gotowych do użycia widżetów, co pozwala na tworzenie wysoce interaktywnych interfejsów użytkownika. Efektywne wykorzystanie widżetów bezstanowych i stanowych może znacznie poprawić wydajność aplikacji16.
* Flutter zapewnia rozbudowane narzędzia do profilowania wydajności, które pomagają deweloperom identyfikować i rozwiązywać problemy związane z wydajnością. Narzędzia te umożliwiają monitorowanie takich aspektów jak zużycie pamięci, czas renderowania klatek oraz wydajność CPU16.
* Flutter wykorzystuje Dart, język programowania optymalizowany pod kątem szybkiego działania aplikacji na różnych platformach. Dart wspiera zarówno kompilację AOT, jak i just-in-time (JIT), co przekłada się na wysoką wydajność zarówno w fazie produkcji, jak i podczas rozwoju[[17]](#footnote-17).

## 3.6 Dobre praktyki Fluttera

Dobre praktyki w programowaniu z wykorzystaniem Fluttera są istotne dla tworzenia wydajnych, skalowalnych i łatwych w zarządzaniu aplikacji. Poniżej przedstawiam szczegółowy opis kilku z tych praktyk wraz z realnymi przypisami:

* Pisanie czystego, dobrze zorganizowanego i modularnego kodu jest kluczowe. Oznacza to używanie komponentów wzorca projektowego, takich jak MVC (Model-View-Controller) lub BLoC (Business Logic Component), dla lepszego zarządzania stanem i strukturą aplikacji[[18]](#footnote-18).
* Zarządzanie stanem jest jednym z najważniejszych aspektów Fluttera. Istnieje kilka podejść, takich jak Provider, Riverpod, Redux, które pomagają w zarządzaniu stanem aplikacji w sposób efektywny i przewidywalny[[19]](#footnote-19).
* Optymalizacja wydajności jest ważna, szczególnie w kontekście animacji i przejść. Używanie const widgets, pre-caching obrazów, i minimalizowanie głębokości drzewa widżetów są niektórymi ze sposobów na poprawę wydajności[[20]](#footnote-20).
* Regularne testowanie jest niezbędne do zapewnienia jakości aplikacji. Flutter oferuje wsparcie dla różnych typów testów, w tym testów jednostkowych, testów widżetów i testów integracyjnych, które pomagają w zapewnieniu stabilności i niezawodności aplikacji[[21]](#footnote-21).
* Asynchroniczne operacje są kluczowe w przypadku operacji I/O, takich jak żądania sieciowe. Używanie Future, Stream, oraz async/await pozwala na zarządzanie operacjami asynchronicznymi bez blokowania interfejsu użytkownika[[22]](#footnote-22).
* Tworzenie aplikacji dostępnych dla osób z różnymi potrzebami i na różnych rynkach wymaga uwzględnienia dostępności (accessibility) i lokalizacji (internationalization). Flutter oferuje narzędzia do obsługi różnych języków, formatów regionalnych oraz wsparcie dla funkcji dostępności[[23]](#footnote-23).
* Dobrze zaprojektowana architektura aplikacji jest kluczowa dla jej skalowalności i łatwości w utrzymaniu. Podział na warstwy logiki biznesowej, danych i prezentacji pomaga w utrzymaniu czystości kodu i ułatwia wprowadzanie zmian[[24]](#footnote-24).
* Dokumentowanie kodu i stosowanie komentarzy jest ważne dla zrozumienia i utrzymania kodu, zwłaszcza w dużych zespołach. Umożliwia to innym programistom szybkie zrozumienie logiki i zamiarów kodu[[25]](#footnote-25).

## 3.7 społeczność Fluttera

Flutter, jako open-source'owy framework do tworzenia aplikacji mobilnych i webowych, oferuje szereg korzyści wynikających z jego otwartości i wsparcia społeczności. Te korzyści obejmują:

* Współpraca i Rozwój Społeczności: Flutter, będąc projektem open-source, jest rozwijany nie tylko przez zespół Google, ale również przez globalną społeczność programistów. Ta współpraca prowadzi do szybszego rozwiązywania problemów, udoskonalania funkcji i wprowadzania innowacji. Wsparcie społeczności obejmuje również udział w konferencjach, spotkaniach grup programistycznych oraz forach dyskusyjnych, takich jak GitHub czy Stack Overflow.
* Bogata Biblioteka Pakietów: Dzięki otwartemu charakterowi Fluttera, programiści mogą tworzyć i dzielić się własnymi pakietami (pluginami), które mogą być wykorzystywane przez innych. To prowadzi do bogatego ekosystemu, w którym dostępne są rozwiązania dla wielu typowych jak i specyficznych wymagań projektowych. Na przykład, popularna strona Dart Pub zawiera tysiące pakietów dostępnych dla Fluttera.
* Szybka Odpowiedź na Problemy i Błędy: Jako projekt open-source, Flutter umożliwia użytkownikom zgłaszanie błędów i problemów bezpośrednio twórcom. Społeczność może również proponować rozwiązania, co przyspiesza proces ich rozwiązywania.
* Dokumentacja i Zasoby Edukacyjne: Flutter posiada obszerną, stale aktualizowaną dokumentację, co jest kluczowe dla nauki i rozwoju umiejętności. Ponadto, istnieje wiele kursów, tutoriali, i przewodników stworzonych przez społeczność, które ułatwiają naukę i rozwiązywanie problemów.
* Wsparcie Wieloplatformowości: Flutter pozwala na tworzenie aplikacji nie tylko na iOS i Android, ale również na inne platformy jak web, desktop czy embedded systems. Ta uniwersalność jest wzmocniona przez wsparcie społeczności, która dzieli się swoimi doświadczeniami i najlepszymi praktykami w zakresie wieloplatformowości.
* Innowacyjność i Adaptacja Trendów: Open-source'owy charakter Fluttera sprzyja szybkiej adaptacji nowych trendów i technologii. Społeczność programistów aktywnie eksperymentuje z nowymi pomysłami i rozwiązaniami, co przekłada się na ciągły rozwój frameworka.
* Bezpłatność i Otwartość na Wkład: Flutter jest całkowicie bezpłatny i otwarty na wkład użytkowników. To oznacza, że każdy, niezależnie od umiejętności czy doświadczenia, może przyczynić się do jego rozwoju.
* Wsparcie od Gigantów Technologicznych: Flutter, będąc wspierany przez Google, korzysta z zasobów i doświadczenia jednej z największych firm technologicznych. To zapewnia wysoki poziom jakości i bezpieczeństwa frameworka.

## 3.8 Przykładu wykorzystania w komercyjnych projektach

Flutter, jako wszechstronny framework do tworzenia aplikacji, został z powodzeniem wykorzystany w wielu komercyjnych projektach. Jego elastyczność, wydajność i możliwość tworzenia aplikacji na różne platformy sprawiają, że jest atrakcyjny dla firm i przedsiębiorstw. Poniżej przedstawiono kilka przykładów komercyjnego wykorzystania Fluttera.

Alibaba, jedna z największych globalnych platform e-commerce, to doskonały przykład komercyjnego wykorzystania Fluttera. Firma zdecydowała się na Fluttera do rozwoju swojej aplikacji Xianyu, która jest jedną z największych platform sprzedaży detalicznej online w Chinach.

Xianyu, obsługując miliony użytkowników, wymagała skalowalnego rozwiązania, które Flutter był w stanie zapewnić. Dzięki możliwościom Fluttera, aplikacja mogła sprawnie radzić sobie z dużą liczbą użytkowników i transakcji.  
 Wybór Fluttera był podyktowany jego zdolnością do efektywnego tworzenia aplikacji działających zarówno na iOS, jak i Androidzie. Dzięki temu Alibaba mogła zaoferować swoim klientom spójne doświadczenia użytkownika na różnych urządzeniach i platformach.  
Flutter pozwolił zespołowi programistów Alibaba na szybkie tworzenie i wdrażanie nowych funkcji. Hot reload, jedna z kluczowych funkcji Fluttera, pozwala na natychmiastowe widzenie zmian w kodzie bez konieczności ponownego uruchamiania aplikacji, co znacząco przyspiesza proces rozwoju.  
 Korzystając z Fluttera, Alibaba była w stanie znacząco zmniejszyć koszty i czas potrzebny na rozwój aplikacji. Jeden wspólny kod dla obu platform zmniejszył potrzebę zatrudniania osobnych zespołów dla Androida i iOS, co jest ważne z punktu widzenia efektywności biznesowej.  
 Aplikacja Xianyu, dzięki wykorzystaniu Fluttera, oferuje płynne i intuicyjne doświadczenia użytkownika. Wydajność i płynność animacji są na wysokim poziomie, co przekłada się na lepszą satysfakcję klientów[[26]](#footnote-26).

Kolejnym przykładem komercyjnego wykorzystania Fluttera jest aplikacja mobilna Google Ads. Google Ads, będąc jednym z najważniejszych narzędzi reklamowych Google, umożliwia zarządzanie kampaniami reklamowymi na różnych platformach i urządzeniach. Flutter został wykorzystany do stworzenia aplikacji mobilnej dla tego serwisu, co przyniosło szereg korzyści.

Google Ads wymagało wszechstronnej platformy, zdolnej do obsługi skomplikowanej logiki biznesowej i dużej ilości danych. Flutter został wybrany ze względu na swoją wydajność i zdolność do obsługi złożonych interfejsów użytkownika.

Flutter umożliwił tworzenie spójnego interfejsu użytkownika na różnych urządzeniach mobilnych. Dzięki temu użytkownicy Google Ads doświadczają podobnego wyglądu i funkcjonalności aplikacji, niezależnie od tego, czy korzystają z systemu Android czy iOS.  
 Wykorzystanie Fluttera pozwoliło na szybki rozwój aplikacji Google Ads. Funkcje takie jak hot reload znacząco przyspieszają proces tworzenia i testowania nowych funkcji, co jest szczególnie ważne w dynamicznym środowisku biznesowym.  
 Flutter zapewnia wysoką wydajność aplikacji, co jest kluczowe dla narzędzia tak intensywnie wykorzystywanego jak Google Ads. Dobra optymalizacja i zarządzanie pamięcią pozwalają na płynne działanie aplikacji, nawet przy obsłudze dużej ilości danych.  
 Jako produkt Google, Flutter naturalnie integruje się z innymi usługami i systemami Google. Dla Google Ads, było to szczególnie ważne, aby zapewnić bezproblemową integrację z innymi narzędziami i usługami Google[[27]](#footnote-27).

Jeszcze jednym wybitnym przykładem użycia Fluttera w komercyjnym projekcie jest aplikacja mobilna BMW. BMW, jako renomowany producent samochodów, wykorzystał Flutter do stworzenia swojej aplikacji "My BMW", która oferuje użytkownikom zaawansowane funkcje zarządzania pojazdami i interakcji z marką.  
 Aplikacja "My BMW" korzysta z zaawansowanych funkcji interfejsu użytkownika dostarczanych przez Flutter, oferując elegancki i intuicyjny design, który jest zgodny z wysokimi standardami marki BMW.  
 Ze względu na potrzebę szybkiego i niezawodnego przetwarzania danych, aplikacja "My BMW" wymagała platformy, która mogłaby zapewnić wysoką wydajność. Flutter spełnia te wymagania, oferując płynne działanie i szybką reakcję na interakcje użytkownika.  
 Flutter pozwolił na stworzenie aplikacji "My BMW", która działa równie efektywnie zarówno na urządzeniach z systemem Android, jak i iOS, co jest kluczowe dla globalnej bazy klientów BMW.

Aplikacja "My BMW" integruje się z systemami samochodowymi, umożliwiając użytkownikom zdalne zarządzanie różnymi funkcjami ich pojazdów. Flutter umożliwia efektywną komunikację między aplikacją a systemami samochodowymi.  
 Korzystanie z Fluttera umożliwiło zespołowi BMW szybki rozwój aplikacji oraz łatwe wprowadzanie aktualizacji i nowych funkcji, co jest istotne w szybko zmieniającym się świecie technologii mobilnych[[28]](#footnote-28).

# Rozdział 4: Flet we własnym projekcie

W tym rozdziale przestawiono 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.

4.1 Notatnik dla treningu kulturystycznego

Problem, który chciano rozwiązać to brak na rynku notatnika do treningu kulturystycznego który miałby prosty, czytelny i nowoczesny interface. Pozbawiony 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 który zna autor 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 Dockera, co stanowi część procesu deploymentu.  
 W początkowej fazie projektowania aplikacji, rozważano 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 brak doświadczenia uznano, że nauka Dart - języka programowania używanego w Flutterze może znacząco spowolnić postęp prac nad projektem.   
 W obliczu tych rozważań, podjęto decyzję o wyborze Fleta, frameworku biblioteki dla Pythona, który lepiej wpasowywał się w dotychczasowe doświadczenie programistyczne. Wybór ten pozwolił na efektywniejsze wykorzystanie istniejących umiejętności programowania w Pythonie, co znacznie przyspieszyło proces tworzenia interfejsu użytkownika. Flet, będąc mniej znanym, ale wysoce funkcjonalnym narzędziem, umożliwił stworzenie responsywnego i estetycznego frontendu, jednocześnie zachowując spójność z używanym stackiem technologicznym w całej aplikacji.

## 4.2. Deployment 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.  
 Przed wdrożeniem aplikacji na platformę, 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.   
 Fly wymaga stworzenia pliku konfiguracyjnego `fly.toml`, który określa ustawienia specyficzne dla środowiska produkcyjnego na Fly.io. W konfiguracji tej należy zdefiniować takie parametry jak wersja, zasoby (np. pamięć RAM, CPU) i inne ustawienia związane z wdrażaniem aplikacji. W tym przypadku wystarczyło z edytować nazwę aplikacji resztę pliku pozostawiono bez zmian.   
 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.

## 4.3 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.

# Rozdział 5: Podsumowanie i Wnioski

## 5.1. Wnioski z Analizy Architektury Fleta:

## 5.2. Wnioski z Analizy Porównawczej:

## 5.3. Wnioski z Studiów Przypadków:

Wrocław, dnia …………………….

OŚWIADCZENIE

Ja, niżej podpisany/a ......................................................................................................., świadomy/a odpowiedzialności prawnej, student/ka Uniwersytetu WSB Merito we Wrocławiu oświadczam, że przedkładaną pracę dyplomową napisałem/napisałam samodzielnie, korzystając jedynie z pomocy merytorycznej promotora.

Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem/am z pomocy innych osób, a w szczególności nie zlecałem/am opracowania pracy lub jej części innym osobom, ani nie odpisywałem/am tej pracy lub jej części od innych osób. Wszystkie wykorzystane przeze mnie źródła, w tym fragmenty prac i ustalenia innych osób, zostały odpowiednio zaznaczone, z zachowaniem praw autorskich oraz wydawniczych. Praca nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam ponadto, że niniejsza praca nie została przedstawiona do oceny w innej procedurze egzaminacyjnej ani nie była publikowana.

Oświadczam również, że egzemplarz pracy dyplomowej zatwierdzonej przez Promotora jest całkowicie zgodny z egzemplarzem pracy dyplomowej zamieszczonej w wersji elektronicznej w uczelnianym systemie elektronicznym.

Jednocześnie przyjmuję do wiadomości, że gdyby niniejsze oświadczenie okazało się niezgodne z prawdą stanowi to podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego i unieważnienia dyplomu.

...............…….………..............................................

(czytelny podpis studenta/studentki oraz nr albumu)

1. Freeman, Adam. "Pro ASP.NET MVC Framework." Apress, 2009. [↑](#footnote-ref-1)
2. Fowler, Martin. "Patterns of Enterprise Application Architecture." Addison-Wesley, 2002. [↑](#footnote-ref-2)
3. Viega, John, and Matt Messier. "Secure Programming Cookbook for C and C++." O'Reilly Media, 2003. [↑](#footnote-ref-3)
4. Freeman, Eric. "Head First Design Patterns." O'Reilly Media, 2004. [↑](#footnote-ref-4)
5. <https://pypi.org/project/flet/> [↑](#footnote-ref-5)
6. https://flet.dev/docs/guides/python/getting-started/ [↑](#footnote-ref-6)
7. https://flet.dev/docs/guides/python/getting-started/ [↑](#footnote-ref-7)
8. https://flet.dev/docs/guides/python/user-controls/ [↑](#footnote-ref-8)
9. <https://flutter.dev/docs/development/ui/widgets> [↑](#footnote-ref-9)
10. <https://flutter.dev/docs/get-started/web> [↑](#footnote-ref-10)
11. <https://flutter.dev/docs/get-started/install> [↑](#footnote-ref-11)
12. <https://flutter.dev/docs/get-started/flutter-for/embedded-devices> [↑](#footnote-ref-12)
13. <https://fuchsia.dev/fuchsia-src/development/languages/flutter> [↑](#footnote-ref-13)
14. <https://flutter.dev/docs/development/ui/widgets/layout> [↑](#footnote-ref-14)
15. <https://flutter.dev/docs/resources/technical-overview> [↑](#footnote-ref-15)
16. <https://developers.google.com/skia> [↑](#footnote-ref-16)
17. <https://dart.dev/guides/language/why-dart> [↑](#footnote-ref-17)
18. <https://github.com/brianegan/flutter_architecture_samples> [↑](#footnote-ref-18)
19. <https://flutter.dev/docs/development/data-and-backend/state-mgmt/intro> [↑](#footnote-ref-19)
20. <https://flutter.dev/docs/perf/rendering> [↑](#footnote-ref-20)
21. <https://flutter.dev/docs/testing> [↑](#footnote-ref-21)
22. <https://dart.dev/codelabs/async-await> [↑](#footnote-ref-22)
23. <https://flutter.dev/docs/development/accessibility-and-localization/internationalization> [↑](#footnote-ref-23)
24. <https://dart.dev/guides/language/effective-dart/design> [↑](#footnote-ref-24)
25. <https://dart.dev/guides/language/effective-dart/documentation> [↑](#footnote-ref-25)
26. <https://flutter.dev/showcase/alibaba>) [↑](#footnote-ref-26)
27. <https://flutter.dev/showcase/google-ads> [↑](#footnote-ref-27)
28. <https://flutter.dev/showcase/bmw>) [↑](#footnote-ref-28)