|  |  |
| --- | --- |
|  | **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

Spis treści

[Wstęp 4](#_Toc154141702)

[Rozdział 1: Teoretyczne podstawy funkcjonowania frameworków 5](#_Toc154141703)

[1.1. Historia i Rozwój Frameworków 5](#_Toc154141704)

[Rozdział 2: Flet 7](#_Toc154141705)

[2.1. Dlaczego Flet 7](#_Toc154141706)

[2.2. Czym jest Flet 7](#_Toc154141707)

[2.2.1 Architektura 8](#_Toc154141708)

[2.2.2 Imperatywny Model UI 8](#_Toc154141709)

[2.3. Praktyczne zastosowanie w projekcie 8](#_Toc154141710)

[2.3.1. Minimalna aplikacja 9](#_Toc154141711)

[2.3.2 Sesje 10](#_Toc154141712)

[2.3.3 Zastosowania Routerów 11](#_Toc154141713)

[2.2.4 Widgety 13](#_Toc154141714)

[2.4 Domyślna estetyka warstwy wizualnej Flet 16](#_Toc154141715)

[2.5 Deployement 16](#_Toc154141716)

[Rozdział 3: Flutter 17](#_Toc154141717)

[3.1. Cross-platform Development 17](#_Toc154141718)

[3.2 Język Dart 18](#_Toc154141719)

[3.3 Zestaw widgetów 19](#_Toc154141720)

[3.4 Hot Reload 20](#_Toc154141721)

[3.5 Architektura Fluttera 21](#_Toc154141722)

[3.5.1 Warstwy Fluttera 22](#_Toc154141723)

[3.5.2 Daklaratywność 24](#_Toc154141724)

[3.5.3 Widżety 25](#_Toc154141725)

[3.5.4 Renderowanie 27](#_Toc154141726)

[3.5.4 Integracja z innymi językami 28](#_Toc154141727)

[3.10 Dobre praktyki Fluttera 30](#_Toc154141728)

[3.11 społeczność Fluttera 31](#_Toc154141729)

[3.12 Przykładu wykorzystania w komercyjnych projektach 31](#_Toc154141730)

[3.8.1 Alibaba 32](#_Toc154141731)

[3.8.2 Google Ads 32](#_Toc154141732)

[3.8.3. Aplikacja mobilna BMW 33](#_Toc154141733)

[Rozdział 4: Podsumowanie i Wnioski 34](#_Toc154141734)

# 

# 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.

Dzięki gotowym komponentom i szablonom, deweloperzy mogą skupić się na bardziej złożonych aspektach aplikacji, zamiast na podstawowych elementach struktury użytkownikiem. Frameworki promują stosowanie najlepszych praktyk i wzorców projektowych, co prowadzi do tworzenia bardziej spójnego i łatwego w utrzymaniu kodu2. Wiele frameworków oferuje wbudowane mechanizmy bezpieczeństwa, które chronią aplikacje przed powszechnymi zagrożeniami, jak ataki SQL injection czy cross-site scripting3.

## 1.1. Historia i Rozwój Frameworków

Rozwój frameworków webowych rozpoczął się na początku XXI wieku, wprowadzając nowe kierunki w tworzeniu aplikacji internetowych. Jeden z pierwszych kamieni milowych to Ruby on Rails, zaprezentowany w 2004 roku przez Davida Heinemeiera Hanssona. Rails, oparty na Ruby, wprowadził filozofię "konwencji ponad konfiguracją" oraz RESTful design aplikacji, co znacząco przyczyniło się do uproszczenia i przyspieszenia procesu tworzenia aplikacji webowych[[1]](#footnote-1).

W 2005 roku na scenę wkroczył Django, framework Pythona stworzony przez Adriana Holovaty'ego i Jacoba Kaplan-Mossa. Django zyskał popularność dzięki filozofii "baterie w zestawie", oferując bogaty zestaw funkcjonalności out-of-the-box, co czyniło go atrakcyjnym wyborem dla szybkiego rozwoju aplikacji[[2]](#footnote-2).

Kolejnym znaczącym krokiem był wprowadzony w 2010 roku AngularJS przez Google, który zrewolucjonizował tworzenie jednostronicowych aplikacji. Jego deklaratywne szablony i system wstrzykiwania zależności znacząco uproszczały rozwój frontendu[[3]](#footnote-3). W 2016 roku Angular przeszedł gruntowną transformację wprowadzając TypeScript i zwiększając wydajność i skalowalność[[4]](#footnote-4).

Rok 2011 to narodziny Ember.js, stworzonego przez Yehudę Katza. Ember zyskał uznanie za swoje silne przekonanie o konieczności stosowania konwencji ponad konfiguracją, skupiając się na produktywności i długoterminowej utrzymaniu kodu[[5]](#footnote-5).

W 2013 roku Facebook wprowadził React, który szybko stał się jednym z najpopularniejszych narzędzi do tworzenia interfejsów użytkownika. Jego innowacyjne podejście do wirtualnego DOM i architektury opartej na komponentach miało ogromny wpływ na rozwój frontendu[[6]](#footnote-6).

Rok 2014 przyniósł Vue.js, stworzony przez Evana You, który szybko zdobył popularność dzięki swojej prostocie i elastyczności, łącząc w sobie najlepsze cechy Reacta i Angulara[[7]](#footnote-7).

W 2016 roku, na scenę wkroczył Svelte, stworzony przez Richa Harrisa. Ten framework zaskoczył rynek swoim unikalnym podejściem do kompilacji, eliminując potrzebę wirtualnego DOM i generując wysoce wydajny, imperatywny kod[[8]](#footnote-8).

W kontekście backendu, w 2010 roku pojawił się Express.js, minimalistyczny i elastyczny framework dla Node.js, który zrewolucjonizował tworzenie serwerów webowych[[9]](#footnote-9).

Frameworki te, każdy na swój sposób, przyczyniły się do kształtowania współczesnego internetu, udowadniając, że innowacje technologiczne są nie tylko możliwe, ale także niezbędne dla ciągłego rozwoju cyfrowego świata. Od Rails przez Django, Angulara, Ember.js, Reacta, aż po Vue.js i Svelte, każdy z tych frameworków wnosił coś unikalnego do ekosystemu web developmentu, tworząc podstawy dla przyszłych innowacji i rozwoju.

# Rozdział 2: flet

## 2.1. Dlaczego Flet

Flutter, będąc jednym z najbardziej innowacyjnych frameworków UI opracowanych przez Google, zdobył popularność dzięki swojej multiplatformowości i zaawansowanym narzędziom, które znacznie przyspieszają proces tworzenia aplikacji[[10]](#footnote-10). Z drugiej strony, Python jest ceniony za swoją prostotę i uniwersalność, będąc jednym z najbardziej popularnych języków programowania na świecie[[11]](#footnote-11). 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.

Material Design, będący językiem projektowania Google, jest szeroko stosowany w aplikacjach mobilnych i stronach internetowych[[12]](#footnote-12). Flet, wykorzystując ten język projektowania, umożliwia tworzenie aplikacji, które są nie tylko funkcjonalne, ale także estetyczne. Dzięki temu, aplikacje tworzone w Flecie mogą osiągać wysoki poziom użyteczności i estetyki, co jest kluczowe w dzisiejszym świecie cyfrowym.

Flet wyróżnia się swoją bezproblemową instalacją i brakiem potrzeby stosowania dodatkowego SDK. Jako biblioteka Pythona, jest łatwo instalowany za pomocą menedżera pakietów pip, co zapewnia szybki i efektywny proces konfiguracji. Istotnym atutem jest jego natychmiastowa gotowość do użycia bez konieczności konfiguracji złożonego środowiska deweloperskiego, co czyni go atrakcyjnym wyborem dla programistów Pythona.

## 2.2. Czym jest Flet

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[[13]](#footnote-13).

Framework Flet dla Pythona charakteryzuje się modularnością i oferuje intuicyjne narzędzia do tworzenia interfejsów użytkownika.

### 2.2.1 Architektura

Flet to framework umożliwiający budowanie interaktywnych aplikacji używanych przez wielu użytkowników, dla webu, desktopu i urządzeń mobilnych, przy użyciu Pythona, 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[[14]](#footnote-14).

2.2.2 Imperatywny Model UI

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

## 2.3. Praktyczne zastosowanie w 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, przeanalizowane konkretne wyzwania, z jakimi się spotkano, oraz korzyści płynące z wyboru Fleta do implementacji poszczególnych funkcji czy modułów.

Problem, który chciano rozwiązać to brak na rynku notatnika do treningu kulturystycznego który miałby prosty, czytelny, 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.

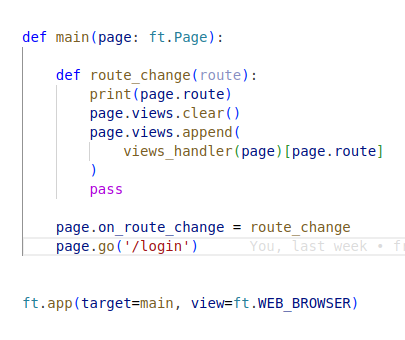
W początkowej fazie projektowania aplikacji, rozważano użycie Fluttera do stworzenia interfejsu użytkownika. Flutter, będąc wszechstronnym i popularnym frameworkiem do tworzenia aplikacji mobilnych i webowych, wydawał się atrakcyjnym wyborem. Jednakże, z uwagi na brak doświadczenia uznano, że nauka Dart - 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.

W rozdziale 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.

2.3.1. 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.

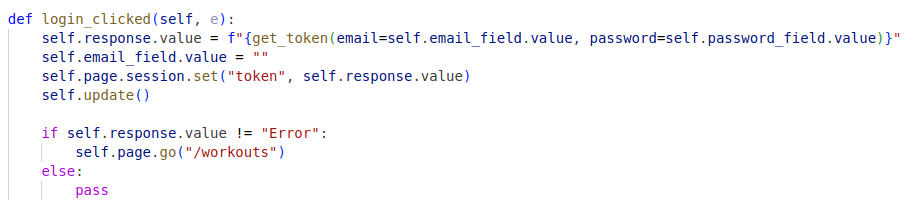


### 2.3.2 Sesje

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.

`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:



### 2.3.3 Zastosowania Routerów

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

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

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



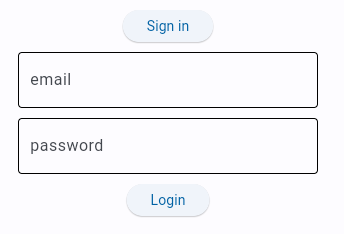
W tym przypadku, dzięki obsłudze żądania GET, użytkownik może odebrać dane w formie JSON, a kod obsługi żądania zajmie się dodaniem listy treningów na front-endzie.

### 2.2.4 Widgety

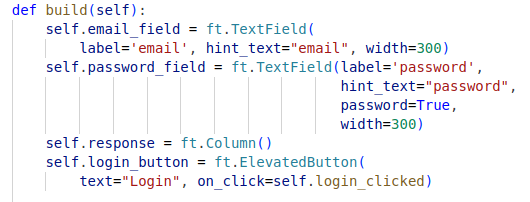
Interfejs użytkownika w Flet składa się z widgetów, które są regularnymi klasami Pythona. Tworzy się instancje widgetów za pomocą konstruktorów, a następnie dodaje je do listy `controls` obiektu `Page`. Aby wyświetlić widgety na stronie, należy dodać je do `Page` i wywołać metodę `update`.

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.

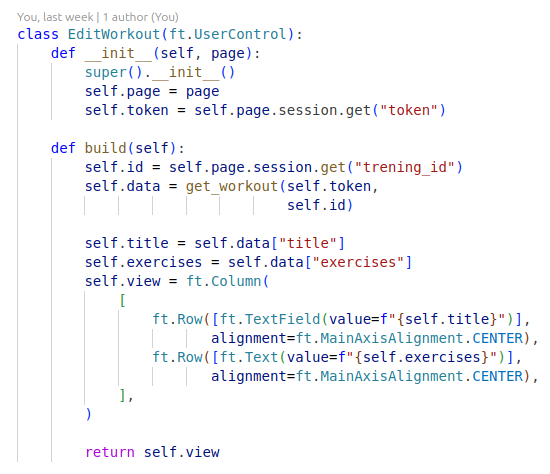




Z kolei w każdym w wierszu umieszczono przyciski wywołujące funkcje, oraz pola pobierające dane od użytkownika.



UserControl w Flecie umożliwia tworzenie izolowanych, wielokrotnego użytku komponentów poprzez łączenie istniejących widgetów Flet. UserControl zachowuje się jak klasa i może mieć metody oraz właściwości. Musi implementować metodę build(), która jest wywoływana w celu zbudowania interfejsu użytkownika. 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 .



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

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.

## 2.5 Deployement

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.

# 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 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[[15]](#footnote-15).

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[[16]](#footnote-16).

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[[17]](#footnote-17).

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[[18]](#footnote-18).

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[[19]](#footnote-19).

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[[20]](#footnote-20).

## 3.2 Język Dart

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. 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. 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. 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.  
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

Flutter, znany ze swojej innowacyjnej platformy do tworzenia aplikacji mobilnych i internetowych, wyróżnia się bogatym zestawem widżetów, które stanowią podstawowe elementy budowy interfejsu użytkownika. W tym rozdziale przyjrzymy się bliżej różnym rodzajom widżetów dostępnym w Flutterze, ich zastosowaniu i kluczowym aspektom, odwołując się do oficjalnej dokumentacji Fluttera i innych źródeł.

Widżety w Flutterze są porównywalne do komponentów w innych frameworkach, jak React. Charakteryzują się tym, że wszystko w Flutterze jest widżetem - od elementów layoutu, takich jak `Row` i `Column`, po elementy interfejsu, jak przyciski i suwaki (Flutter Widgets - [Dokumentacja Flutter](https://flutter.dev/docs/development/ui/widgets)). Istnieją dwa główne rodzaje widżetów: Widżety Bezzstanowe (StatelessWidget) i Widżety Stanowe (StatefulWidget). Pierwsze z nich, jak `Text`, nie zmieniają swojego stanu po utworzeniu, podczas gdy drugie, na przykład `Checkbox`, mogą go zmieniać w odpowiedzi na zdarzenia lub dane wejściowe[[21]](#footnote-21)

Widżety layoutu, takie jak `Row`, `Column`, `Container` i `Stack`, pozwalają na tworzenie elastycznych układów poziomych, pionowych, zapewniają box model dla stylizacji i pozycjonowania, a także umożliwiają nakładanie widżetów na siebie. Są one fundamentem dla tworzenia struktury każdej aplikacji[[22]](#footnote-22). Widżety interaktywne, jak przyciski, pola tekstowe, suwaki i przełączniki, odgrywają kluczową rolę w interakcji użytkownika z aplikacją[[23]](#footnote-23).

Kluczowym aspektem Fluttera jest jego struktura oparta na drzewie widżetów. Każdy widżet w tym drzewie ma rodzica i może mieć dzieci, tworząc hierarchiczną strukturę. Kontekst każdego widżetu jest odniesieniem do miejsca, w którym widżet znajduje się w drzewie, co jest istotne dla zarządzania stanem i nawigacji w aplikacji[[24]](#footnote-24). Widżety stanowe mają także określony cykl życia, z etapami takimi jak `initState`, `build`, `dispose`, co pozwala na precyzyjne zarządzanie zmianami stanu i zasobami[[25]](#footnote-25).

Praktyczne zastosowanie widżetów w Flutterze obejmuje tworzenie całego interfejsu użytkownika, od prostych ekranów po skomplikowane struktury nawigacyjne, jak również stylizację i layout. Przykłady takich zastosowań to `ListView` dla przewijanych list, `GridView` dla układania elementów w siatce, czy `CustomScrollView` dla zaawansowanych efektów przewijania.

Podsumowując, widżety w Flutterze są kluczowe nie tylko do tworzenia interfejsu, ale także do zapewnienia płynności i responsywności aplikacji. Ich modularna natura ułatwia ponowne wykorzystanie kodu i utrzymanie aplikacji, a szeroki zakres dostępnych widżetów pozwala na tworzenie zaawansowanych, estetycznie atrakcyjnych i funkcjonalnych aplikacji. Flutter, dzięki swojej architekturze opartej na widżetach, oferuje wyjątkowe podejście do budowy interfejsów użytkownika, łącząc prostotę z elastycznością w projektowaniu.

## 3.4 Hot Reload

Flutter, platforma stworzona z myślą o tworzeniu aplikacji mobilnych i internetowych, wyróżnia się innowacyjną funkcją znaną jako Hot Reload. Ta funkcja znacząco usprawnia proces deweloperski, umożliwiając programistom błyskawiczne wprowadzanie zmian w kodzie i natychmiastowe ich wyświetlanie bez potrzeby restartowania całej aplikacji. W tym rozdziale szczegółowo omówimy funkcjonalność Hot Reload w Flutterze, jej zalety, mechanizm działania i najlepsze praktyki, opierając się na źródłach takich jak Flutter Dev Guide i Flutter Architecture.

Hot Reload w Flutterze pozwala deweloperom na szybkie i wygodne testowanie zmian w kodzie. Kiedy programista wprowadza zmiany w kodzie źródłowym aplikacji, może natychmiast zobaczyć efekty tych zmian w działającej aplikacji, eliminując potrzebę jej całkowitego restartowania. Jest to szczególnie użyteczne podczas pracy nad interfejsem użytkownika i układem, gdyż pozwala na natychmiastowe zobaczenie wprowadzonych zmian [[26]](#footnote-26).

Mechanizm działania Hot Reload w Flutterze polega na wstrzykiwaniu zaktualizowanego kodu źródłowego do działającej instancji Dart VM (maszyny wirtualnej Dart). Proces ten nie wymaga ponownego ładowania całej aplikacji, co pozwala na zachowanie stanu aplikacji. Dzięki temu Hot Reload jest niezwykle przydatny podczas pracy nad różnymi aspektami interfejsu użytkownika, ponieważ deweloperzy mogą błyskawicznie obserwować wpływ swoich zmian[[27]](#footnote-27).

Jedną z głównych zalet Hot Reload jest zwiększenie produktywności programistów. Funkcja ta znacząco przyspiesza proces tworzenia i iteracji aplikacji. Dodatkowo, Hot Reload ułatwia szybkie prototypowanie, pozwalając deweloperom na eksperymentowanie z różnymi rozwiązaniami UI i UX bez długich przestojów. Kluczowym elementem jest też zdolność do zachowania stanu aplikacji, co jest nieocenione przy testowaniu interakcji i sekwencji użytkownika.

Aby maksymalnie wykorzystać potencjał Hot Reload, deweloperzy powinni stosować się do kilku najlepszych praktyk. Zaleca się używanie lokalnych stanów w widżetach, aby zachować stan przy przeładowaniu. Ponadto, warto unikać złożonych operacji inicjalizacyjnych w `main()` i `initState()`, ponieważ Hot Reload nie przetwarza ponownie tych sekcji. Optymalne zarządzanie stanem aplikacji jest również kluczowe, aby uniknąć niechcianych efektów ubocznych po Hot Reload.

Podsumowując, Hot Reload w Flutterze to potężne narzędzie, które znacząco ulepsza proces deweloperski. Jego zdolność do szybkiego wprowadzania zmian, przy jednoczesnym zachowaniu stanu aplikacji, sprawia, że jest nieoceniony w szybkim prototypowaniu i iteracji UI/UX. To jedna z kluczowych funkcji, która wyróżnia Fluttera na tle innych frameworków do tworzenia aplikacji.

## 3.5 Architektura Fluttera

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.

### 3.5.1 Warstwy Fluttera

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 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 obsługi 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>

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.

### 3.5.2 Daklaratywność

Platforma 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.

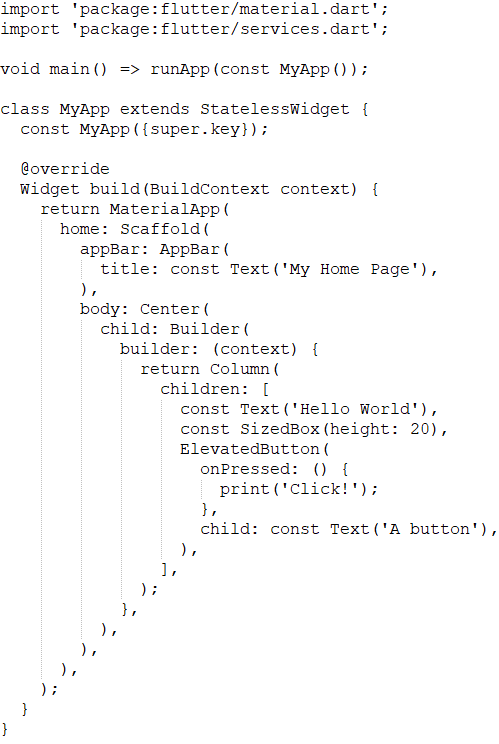
W praktyce, w Flutterze widżety, które są niezmienne, definiują UI przez nadpisanie metody build().

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 zarządzanie zmianami w interfejsie, reagując na zmiany stanu aplikacji.

### 3.5.3 Widżety

Widżety są fundamentem interfejsu użytkownika każdej aplikacji tworzonej w tym frameworku. We 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.

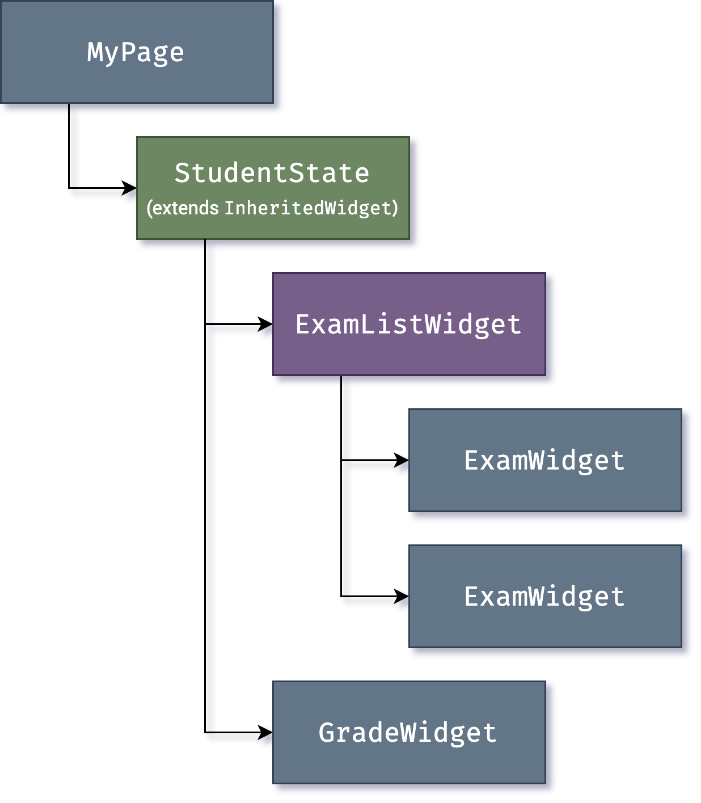
Kluczowe jest, że Flutter pozwala na tworzenie własnych implementacji kontrolek UI, co zapewnia niezależność od systemu operacyjnego. Dzięki temu aplikacje zachowują spójność wizualną i funkcjonalną na różnych platformach, eliminując potrzebę ciągłego przełączania między kodem Fluttera a kodem platformy.



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.

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.



## 3.5.4 Renderowanie

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.

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.

## **3.5.4 Integracja z innymi językami**

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.

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.

Wsparcie Flutter dla platformy webowej jest unikatowe ze względu na wykorzystanie Dart, który kompiluje się do JavaScript. 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

## 3.10 Dobre praktyki Fluttera

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[[28]](#footnote-28).

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[[29]](#footnote-29).

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[[30]](#footnote-30).

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[[31]](#footnote-31).

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[[32]](#footnote-32).

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[[33]](#footnote-33).

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[[34]](#footnote-34).

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[[35]](#footnote-35).

## 3.11 społeczność Fluttera

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

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. 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. Jako projekt open-source, Flutter umożliwia użytkownikom zgłaszanie błędów i problemów bezpośrednio twórcom. Społeczność może również proponować rozwiązania, co przyspiesza proces ich rozwiązywania. 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. Flutter, będąc wspierany przez Google, korzysta z zasobów i doświadczenia jednej z największych firm technologicznych. To zapewnia wysoki poziom jakości i bezpieczeństwa frameworka.

## 3.12 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.

### 3.8.1 Alibaba

Alibana to 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[[36]](#footnote-36).

### 3.8.2 Google Ads

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[[37]](#footnote-37).

### 3.8.3. Aplikacja mobilna BMW

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[[38]](#footnote-38).

# Rozdział 4: Podsumowanie i Wnioski

Podsumowując pracę nad biblioteką "Flet" dla Pythona, staje się jasne, że projekt ten otwiera nowe możliwości w dziedzinie tworzenia aplikacji multiplatformowych. Z ogromnym entuzjazmem obserwuję starania twórców "Flet", aby osiągnąć stabilizację i uruchomienie wersji 1.0. Ta wersja ma za zadanie zaoferować wszechstronność poprzez ujednolicone pakiety na takie platformy jak Windows, Linux, macOS, a także iOS i Android. Taka uniwersalność jest kluczowa dla nowoczesnego programowania aplikacji.

W planach rozwoju "Flet" dostrzegam silny nacisk na optymalizację szybkości i adaptacyjne oraz responsywne interfejsy użytkownika. Te aspekty mają kluczowe znaczenie dla tworzenia intuicyjnych i atrakcyjnych aplikacji. Dodatkowo, wsparcie dla lokalizacji i integracja z zewnętrznymi pakietami Pythona oraz Fluttera otwiera nowe horyzonty w elastycznym i rozbudowanym tworzeniu aplikacji.

Praca z dokumentacją "Flet" była wyzwaniem ze względu na jej dynamikę i nieaktualne treści, ale i tak stanowiła łatwiejszy punkt wejścia niż pierwsze zderzenie z Dartem. Ta początkowa ekspozycja na "Flet" przekonała mnie, że Flutter, z jego już ugruntowanym wsparciem multiplatformowym, jest lepszym wyborem dla moich przyszłych projektów. W ten sposób, "Flet" stał się nie tylko narzędziem do tworzenia aplikacji, ale także doskonałym wprowadzeniem do świata Fluttera, zachęcając do dalszego rozwijania umiejętności i eksplorowania nowych możliwości w tworzeniu aplikacji.

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. (Ruby, Sam; Thomas, Dave; Hansson, David Heinemeier. "Agile Web Development with Rails 6." Pragmatic Bookshelf, 2019) [↑](#footnote-ref-1)
2. (Holovaty, Adrian; Kaplan-Moss, Jacob. "The Definitive Guide to Django." Apress, 2007) [↑](#footnote-ref-2)
3. Green, Brad; Seshadri, Shyam. "AngularJS Up and Running." O'Reilly Media, 2014 [↑](#footnote-ref-3)
4. Freeman, Adam. "Pro Angular 6." Apress, 2018 [↑](#footnote-ref-4)
5. Kratzer, Jamie; White, Erik Hanchett. "Ember.js Cookbook." Packt Publishing, 2016 [↑](#footnote-ref-5)
6. Fedorov, Alex. "React.js Essentials." Packt Publishing, 2015; Banks, Alex; Porcello, Eve. "Learning React." O'Reilly Media, 2017 [↑](#footnote-ref-6)
7. Braun, Matthias. "Vue.js 2 Design Patterns and Best Practices." Packt Publishing, 2018 [↑](#footnote-ref-7)
8. Dell’Orto, Mark Volkmann. "Svelte and Sapper in Action." Manning Publications, 2021 [↑](#footnote-ref-8)
9. Keig, Thomas. "Express in Action." Manning Publications, 2016 [↑](#footnote-ref-9)
10. "Flutter - Beautiful native apps in record time," Google, [Dostęp online: flutter.dev] [↑](#footnote-ref-10)
11. "Python in 2020: Popularity, Usage, and Trends," Python Software Foundation, [Dostęp online: python.org] [↑](#footnote-ref-11)
12. "Material Design Guidelines," Google Material Design, [Dostęp online: material.io] [↑](#footnote-ref-12)
13. <https://pypi.org/project/flet/> [↑](#footnote-ref-13)
14. https://flet.dev/docs/guides/python/getting-started/ [↑](#footnote-ref-14)
15. <https://flutter.dev/docs/development/ui/widgets> [↑](#footnote-ref-15)
16. <https://flutter.dev/docs/get-started/web> [↑](#footnote-ref-16)
17. <https://flutter.dev/docs/get-started/install> [↑](#footnote-ref-17)
18. <https://flutter.dev/docs/get-started/flutter-for/embedded-devices> [↑](#footnote-ref-18)
19. <https://fuchsia.dev/fuchsia-src/development/languages/flutter> [↑](#footnote-ref-19)
20. <https://flutter.dev/docs/development/ui/widgets/layout> [↑](#footnote-ref-20)
21. (StatefulWidget vs StatelessWidget - [Flutter API](https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html)). [↑](#footnote-ref-21)
22. (Layout Widgets - [Flutter Layout Guide](https://flutter.dev/docs/development/ui/layout)) [↑](#footnote-ref-22)
23. (Interactive Widgets - [Flutter Interactivity Guide](<https://flutter.dev/docs/development/ui/interactive)>) [↑](#footnote-ref-23)
24. (Understanding Flutter’s Widget Tree - [Medium Article](https://medium.com/flutter-community/understanding-flutters-widget-tree-4f5b792b4d03)) [↑](#footnote-ref-24)
25. (Flutter Widget Lifecycle - [Flutter Lifecycle Article](https://www.fluttercampus.com/guide/106/stateful-widget-lifecycle-in-flutter/)) [↑](#footnote-ref-25)
26. ([Understanding Flutter Hot Reload - Flutter Dev Guide](https://flutter.dev/docs/development/tools/hot-reload)). [↑](#footnote-ref-26)
27. ([Flutter Hot Reload Mechanism - Flutter Architecture](https://flutter.dev/docs/resources/architectural-overview#hot-reload)) [↑](#footnote-ref-27)
28. <https://github.com/brianegan/flutter_architecture_samples> [↑](#footnote-ref-28)
29. <https://flutter.dev/docs/development/data-and-backend/state-mgmt/intro> [↑](#footnote-ref-29)
30. <https://flutter.dev/docs/perf/rendering> [↑](#footnote-ref-30)
31. <https://flutter.dev/docs/testing> [↑](#footnote-ref-31)
32. <https://dart.dev/codelabs/async-await> [↑](#footnote-ref-32)
33. <https://flutter.dev/docs/development/accessibility-and-localization/internationalization> [↑](#footnote-ref-33)
34. <https://dart.dev/guides/language/effective-dart/design> [↑](#footnote-ref-34)
35. <https://dart.dev/guides/language/effective-dart/documentation> [↑](#footnote-ref-35)
36. <https://flutter.dev/showcase/alibaba>) [↑](#footnote-ref-36)
37. <https://flutter.dev/showcase/google-ads> [↑](#footnote-ref-37)
38. <https://flutter.dev/showcase/bmw>) [↑](#footnote-ref-38)