



Politechnika Łódzka

Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

Aplikacja mobilna wspomagająca monitorowanie dziennego spożycia kalorii z wykorzystaniem technologii **speech to text**

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

Promotor: dr inż. Krzysztof Lichy

Dyplomant: Adam Jóźwiak

Nr albumu: 216786

Kierunek: Informatyka Stosowana

Specjalność: Technologie gier i symulacji komputerowych

Łódź, 2020/2021



Instytut Informatyki

90-924 Łódź, ul. Wólczańska 215, budynek B9

tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

Spis treści

1 Wstęp	4
1.1 Problematyka	4
1.2 Cel i zakres pracy	5
2 Teoria	5
2.1 Rynek aplikacji mobilnych	5
2.2 Programowanie aplikacji na urządzenia mobilne	8
2.3 Speech to text	9
2.4 Bazy danych i ich rodzaje	11
2.5 Chmurowe bazy danych	13
2.6 API	14
2.7 Analiza konkurencyjnych rozwiązań	16
2.7.1 Fitatu	16
2.7.2 Yazio	19
2.7.3 Fatsecret	21
2.8 Wymagania funkcjonalne	24
2.9 Wymagania niefunkcjonalne	24
3 Technologie i narzędzia wykorzystane w projekcie	25
3.1 Dart	25
3.2 Flutter	26
3.3 Cloud Firestore	26
3.4 Android Studio	27
3.5 Android Emulator	27
3.6 VIM	28
4 Proces powstawania aplikacji	29
4.1 Aplikacja od strony logiki	29
4.2 Aplikacja od strony użytkownika	35
5 Podsumowanie	44
6 Indeks rysункów	45

1 Wstęp

1.1 Problematyka

W epoce internetu XXI wieku dostęp do wiedzy jest wyjątkowo łatwy. Każdy ma możliwość zdobycia informacji na interesujący go w mniejszym, lub większym stopniu temat, a gdy media oraz tak zwani "influencerzy" promują zdrowy styl życia, popularnym trendem ostatnich lat stało się dbanie o własne ciało. Trend popędzany również łatwym dostępem do najróżniejszych portali społecznościowych i chęcią pokazania swoich zdjęć na forum publicznym sprawił, że co roku znacznie zwiększa się ogólna liczba ludzi chętnych do uczęszczania na siłownię i odbywania regularnych treningów.

RównieowącznączęściądbaniaoSwojego ciała, oprócz treningów siłowych, jest zastosowanie odpowiedniej diety. Nie jest to zadanie trywialne, ponieważ jest ono w pełni zależne od zapotrzebowania osoby zainteresowanej. W związku z tym wiele osób chętnych do rozpoczęcia procesu odchudzania, sięga po jedną z najprostszych i skutecznych metod, a mianowicie liczenie zjedzonych kalorii. Rekomendowana ilość dziennego ich spożycia jest zależna od szerokiego wachlarza czynników, takich jak przykładowo: płeć, wiek, wzrost, tryb życia, jednak w pracy naukowej zatytułowanej "How many calories should I eat a day?" [7], autor podsumował, iż rekommendowaną dzienną ilością spożytych kalorii dla przeciętnego mieszkańców Stanów Zjednoczonych, jest:

- mężczyzna - około 2500 kcal
- kobieta - około 2000 kcal

Bez odpowiedniego notowania procedura liczenia kalorii nie jest jednak efektywna, stąd ludzie przyzwyczajeni do wygody oferowanej przez technologię, sięgają po różnego rodzaju aplikacje, mające jeszcze bardziej ułatwić to, i tak już trywialne zadanie.

Choć na rynku istnieje wiele aplikacji udostępniających taką funkcjonalność, używanie ich jest często zadaniem nieprzyjemnym i czasochłonnym, polegającym na wykonaniu całych serii kliknięć przeprowadzających użytkownika z jednego menu do drugiego, w celu wyszukania i dodania składników do listy spożytych artykułów, co długofalowo może przełożyć się na porzucenie chęci prowadzenia szczegółowych notatek. Brakuje na rynku rozwiązania, które udostępniłoby łatwy i szybki w użyciu interfejs, oraz w bardzo krótkim okresie czasowym pozwoliłoby wyszukać i dodać interesujące użytkownika produkty. Dodatkowym atutem byłoby wzbogacenie

takiej aplikacji o moduł wykorzystujący mowę ludzką, do jeszcze większego usprawnienia całego procesu.

1.2 Cel i zakres pracy

Celem niniejszej pracy dyplomowej jest stworzenie aplikacji mobilnej, mającej za zadanie wspomóc użytkownika w prowadzeniu notatek świadczących o dziennym spożyciu kalorii. Aplikacja udostępniać będzie moduł speech to text, który w jeszcze większym stopniu ułatwi proces wyszukiwania produktów, a całość będzie uporządkowana w postaci przejrzystego i prostego w obsłudze interfejsu graficznego.

Praca obejmuje implementację logiki przy użyciu języka programowania *Dart*, natomiast wygląd aplikacji, oraz interakcje pomiędzy poszczególnymi modułami zarządzane są przy pomocy frameworku *Flutter*.

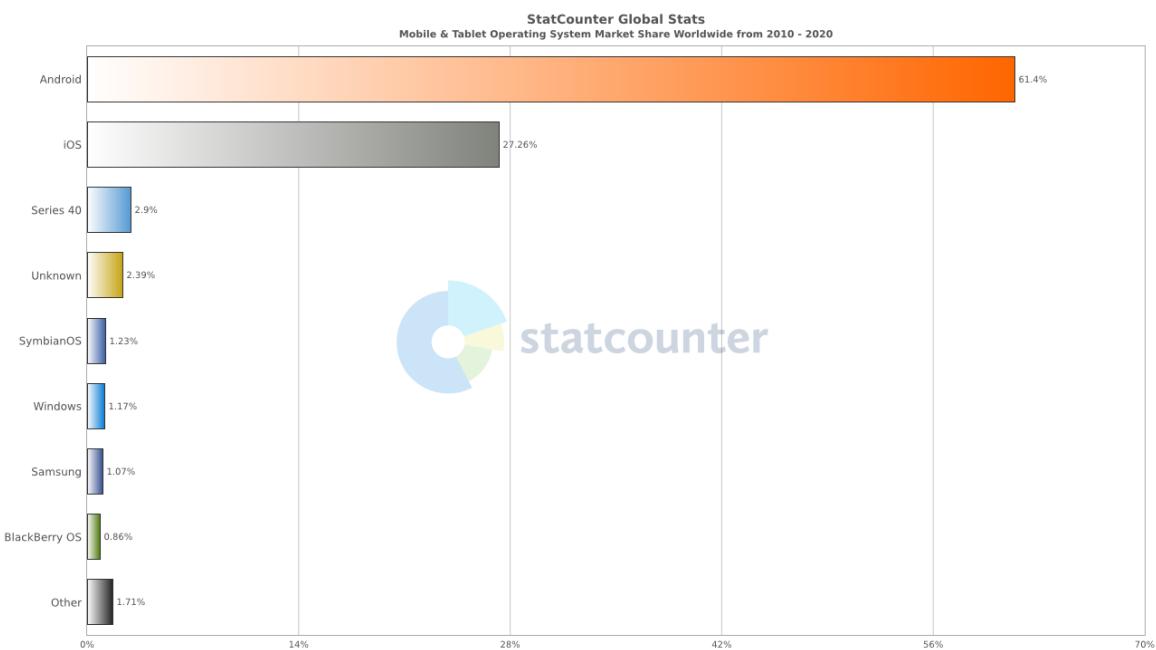
2 Teoria

2.1 Rynek aplikacji mobilnych

Urządzenia mobilne są obok komputerów uważane za jedne z najbardziej rewolucyjnych i przełomowych technologii opracowanych w XX wieku. I choć pierwsze prototypy powstawały już wcześniej niż lata czterdzieste, nie przypominały w obsłudze urządzeń dostępnych dzisiaj i wykorzystywane były głównie w celach militarnych. Oficjalnie za pierwszy telefon w historii uważany jest egzemplarz Motorola DynaTAC 8000x, wynaleziony przez Martina Coopera [10]. I choć zamienienie prototypu na urządzenie dostępne komercyjnie zajęło następną dekadę, tak relatywnie ciężkie początki telefonów zmieniły się w prawdziwą lawinę nowych rozwiązań. Choć nie jest to wiedza powszechna, tak pierwszy smartphone powstał już w 1994 roku, a był to IBM's Simon [8]. Tak więc przez następne lata firmy prześcigały się technologicznie, każda przyczyniając się w mniejszym lub większym stopniu do rozwoju rynku urządzeń mobilnych, aż do 2007 roku, niosącego wydarzenie znane powszechnie jako jedno z najważniejszych w historii telefonów komórkowych. Rok w którym Steve Jobs zaprezentował na konferencji MacWorld 2007 pierwszy model smartphone'a iPhone 2G, który w pełni pozbył się przycisków na rzecz dotykowego ekranu. Moment ten wyznaczył tor po jakim aż po dziś dzień poruszają się wszystkie firmy zajmujące się tworzeniem urządzeń mobilnych i w ten sposób oficjalnie rozpoczęła się era smartphone'ów.

Początkowo wszelkie aplikacje mobilne były dostarczane głównie przez producentów telefonów i były one "wbudowane" - nie istniała możliwość zainstalowania ich po zakupie produktu. W roku 1983 Steve Jobs planował już jednak stworzenie miejsca, w którym oprogramowanie mogłoby być kupowane przy użyciu linii telefonicznych [16]. Tak oto w 2008 roku została uruchomiona usługa "App Store", wraz z flotą 500 dostępnych aplikacji. Jednak Apple jest firmą restrykcyjną i tworzenie nowych aplikacji na system operacyjny iOS nie było zadaniem trywialnym. Na szczęście codziennego użytkownika i ogólnego rozwoju, iOS nie posiadał monopolu na aplikacje mobilne, ponieważ w 2005 roku Google wykupiło firmę, której flagowy produkt jest bardzo dobrze pod nazwą Android OS [2].

Android jest to system operacyjny oparty na rodzinie systemów Linux, co bezpośrednio przełożyło się na jego darmowość i pozwoliło szturmem podbić świat, jak można zaobserwować na raporcie przygotowanym przez firmę StatCounter [12].



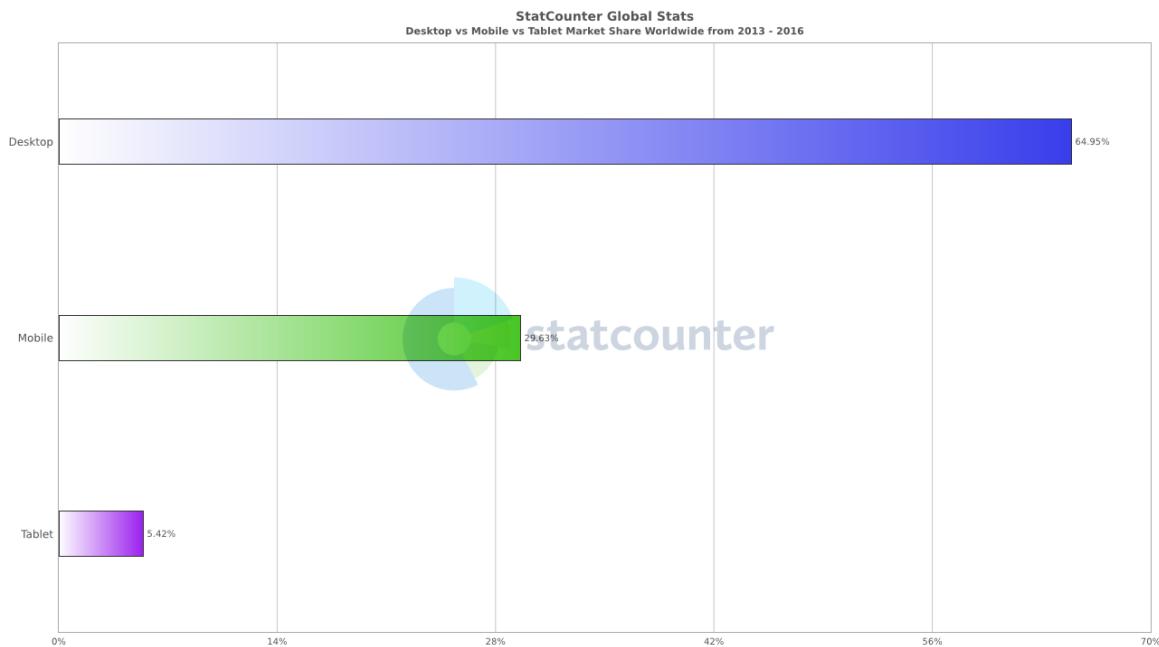
Rysunek 1: Wykres udziału w rynku poszczególnych mobilnych systemów operacyjnych na przestrzeni 10 lat

Źródło: <https://gs.statcounter.com/os-market-share/mobile/worldwide>

Latwa dostępność, ilość urządzeń na których jest on zainstalowany, szeroki wachlarz możliwości - te cechy napędzały ambicje twórców aplikacji mobilnych. Zaoferowany przez firmę Google "Play Store", w przeciwieństwie do App Store pozwala na darmowe udostępnienie produktów o ile spełniają one odpowiednie warunki.

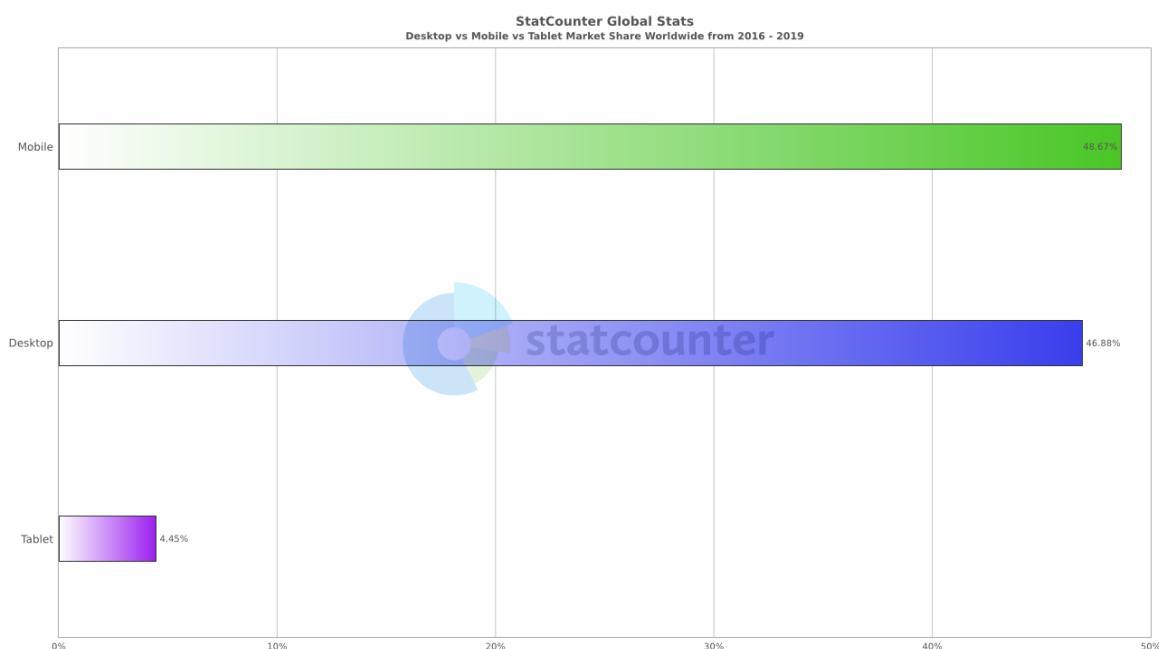
Wraz z tak prężnym rozwojem rynku smartphone'ów, rozpoczęła się zmiana trendów. I o ile od początku XXI wieku światem rządziły komputery, oferujące cały zasób możliwości od nauki,

po rozrywce, tak smartphony z roku na rok coraz bardziej zaczęły przypominać urządzenia desktopowe.



Rysunek 2: Komputery, a smartfony na przestrzeni lat 2013-2016

Źródło: <https://gs.statcounter.com/os-market-share/desktop-mobile-tablet/worldwide/yearly-2013-2016-bar>



Rysunek 3: Komputery, a smartfony na przestrzeni lat 2016-2019

Źródło: <https://gs.statcounter.com/os-market-share/desktop-mobile-tablet/worldwide/yearly-2016-2019-bar>

Urządzenia mobilne były i są w stanie zaoferować dużą większą wygodę użytkowania, prze-

ność, oraz system znacznie łatwiejszy do przyswojenia dla przeciętnego człowieka, nie wymagający szczególnej wiedzy do poprawnego zarządzania. Stąd u młodzieży umiejętność obsługi komputerów stacjonarnych zaczęła zanikać, jako że przestaje ona być tak kluczowa jak była jeszcze 10 lat temu. Jak podaje strona Broadbandsearch [13], już w 2016 roku większość "ruchu" internetowego była spowodowana przez urządzenia mobilne, bo aż 51.3%, i po rok 2021 oscyluje stabilnie w granicach 52%. Jak podaje ten sam portal, różnice pomiędzy rokiem 2013, a 2019 w ilości czasu spędzanego dziennie na użytkowaniu smartphonów i desktopów wygląda następująco:

- Średni czas użytkowania komputerów/laptopów spadł ze 144 minut dziennie, do 128 minut
- Średni czas użytkowania urządzeń mobilnych wzrósł z 88 minut, do średnio 203 minut dziennie

Takie wyniki sprawiły, że coraz więcej firm musiało zacząć liczyć się z rynkiem urządzeń mobilnych, a obecnie częstą praktyką przy tworzeniu aplikacji przeznaczonych zarówno na komputery stacjonarne, jak i smartphony, lub przy pisaniu stron internetowych, jest stylizowanie ich najpierw na mniejsze ekrany, a następnie dostosowywanie pod urządzenia desktopowe. W przypadku marketingu, zamiast "optymalizować" go pod kątem urządzeń mobilnych, zaczęto tworzyć zupełnie oddzielne kampanie marketingowe zarówno dla desktopów jak i smartphone'ów [15].

2.2 Programowanie aplikacji na urządzenia mobilne

Proces tworzenia aplikacji na systemy mobilne jest w dużym stopniu podobny do ich desktopowych odpowiedników. Projektując taką aplikację, należy początkowo wziąć pod uwagę rodzaj systemu operacyjnego, na którym ma ona działać, gdyż różne systemy mają swoje własne wymagania. To przekłada się często na fakt, że jedna aplikacja musi zostać w większości przepisana, aby spełniać warunki narzucone przez różne OS. Moc obliczeniowa jest kolejnym ważnym czynnikiem branym pod uwagę przy tworzeniu takiej aplikacji, gdyż o ile sprzęt dostępny w komputerach pozwala na wydajne działanie skomplikowanych systemów, tak smartphone'owy hardware, pomimo ogromnego postępu, ze względu na swoje rozmiary nadal jest ograniczony, stąd prostota i wydajność aplikacji są kluczowe dla bezproblemowego jej użytkowania.

Obecnie rynek aplikacji mobilnych jest relatywnie młody, dlatego w przyszłości można oczekiwać, że powstanie język programowania / środowisko programistyczne unifikujące tworzenie aplikacji zarówno desktopowych, jak i mobilnych, jednak do czasu powstania takiego systemu, programiści muszą samodzielnie dostosowywać wygląd swoich projektów pod smartphony. Nie

tylko rozmiar ekranu ma znaczenie, ale także współczynnik proporcji (z ang. Aspect Ratio), lub fakt, że urządzenia mobilne mają wbudowany akcelerometr pozwalający na obrócenie ekranu o 90°. Wiele aplikacji pisanych jest tak, aby wyświetlane były w odpowiednim dla wizji programisty współczynniku proporcji, jednak większość tworzona jest zgodnie ze standardem przemysłowym, który dla urządzeń desktopowych wynosi 16:9, a dla mobilnych 9:16. Fakt ten, oraz rozmiary ekranu nie pozwalają na stworzenie jednolitego interfejsu dla obu typu urządzeń, stąd jedna aplikacja może wyglądać zupełnie inaczej w zależności od używanej platformy.

Ostatnim ważnym czynnikiem wyróżniającym proces tworzenia aplikacji mobilnej, jest sposób jej testowania. Dla komputerów stacjonarnych, programiści zazwyczaj mają dostęp do platformy na którą przeznaczyć chęć swój produkt, przykładowo przy pisaniu aplikacji web'owej, działanie jej testuje się w przeglądarce. W przypadku aplikacji mobilnych, używa się specjalnych emulatorów, na których instaluje się swoje oprogramowanie, a następnie na bieżąco sprawdza jego działanie. Z jednej strony, pozwala to na szybkie, bezpieczne i wygodne testowanie, jednak emulatory te nie są w pełni poprawną symulacją prawdziwego systemu, więc często nie jest możliwe sprawdzenie wszystkich możliwych, zaawansowanych interakcji użytkownika.

2.3 Speech to text

Popularną opinią jest, iż lenistwo idzie w parze z ludzkim rozwojem technologicznym. Wiele przypadków potwierdza tą regułę, sam Bill Gates, założyciel firmy Microsoft powiedział: "I choose a lazy person to do a hard job. Because a lazy person will find an easy way to do it." W duchu tych słów, ludzie jako rasa mają w zwyczaju być leniwymi, dlatego naturalną rzeczą stał się fakt, iż ręczne wpisywanie poleceń nie jest tak optymalną techniką "komunikacji" z maszynami, jaką mogłyby być mowa.

W roku 1952, Bell Laboratories stworzyło system "Audrey", który był w stanie rozpoznać wypowiedziane pojedyńcze cyfry, a 10 lat później, "Shoebox" od IBM mógł zrozumieć i odpowiedzieć na 16 słówek w języku angielskim [1]. W latach siedemdziesiątych XX wieku, Amerykański Departament Obrony rozpoczął projekt o nazwie "SUR" - The Speech Understanding Research, którego owocem był "Harpy", system będący w stanie rozpoznać 1000 słów. Współzawodnictwo wielu firm w zakresie rozpoznawania mowy doprowadzało do powstawania dużej ilości nowych modeli, oraz metod, jeszcze bardziej optymalizujących cały proces.

Do roku 2001, estymowana dokładność systemów rozpoznawania mowy wynosiła około 80%, i przez dłuższy nie zostały osiągnięte godne uwagi postępy. Sytuacja jednak zmieniła się dzięki firmie Google, która uaktywniła w roku 2012 usługę Google Voice Search. Okazała się ona być

ogromnym sukcesem, jako że była to darmowa aplikacja dostępna na urządzeniach mobilnych, stąd dzięki jej szerokiej dostępności, firma miała okazję zebrać dane pochodzące od milionów użytkowników, co przełożyło się bezpośrednio na poprawę dokładności algorytmu. Z czasem kolejne firmy zaczęły inwestować w technologię, a okres od roku 2010 do 2020 jest uważany za szczególnie ważny dla historii technologii rozpoznawania mowy. Tak oto w roku 2021 użytkownicy mają dostęp do:

- Google Voice Search / Google Assistant - Google
- Siri - Apple
- Cortana - Microsoft
- Alexa - Amazon
- I wiele innych ...

Firmy te prześcigają się o tytuł dokładności rozpoznawania mowy. W roku 2016, IBM osiągnęło wartość błędu 6,9%, z czego im mniejsza jest ta wartość, tym lepiej działa system. W 2017, Microsoft udało się osiągnąć błąd o wartości 5,9%, jednak IBM szybko odpowiedziało swoim 5,5%. Firmy te nadal pozostają w tyle przy Google, którego błąd w 2017 roku osiągnął 4,9% i do dziś pozostaje najdokładniejszym systemem rozpoznawania mowy dostępnym na rynku. Wraz z upływem czasu, technologia ta będzie coraz szybsza i efektywniejsza i bardzo możliwe jest, iż staną się ona głównym interfejsem służącym komunikacji pomiędzy człowiekiem, a maszyną.

Przez większą część historii technologii "Speech Recognition", najpopularniejszą metodą był HMM - "Hidden Markov Model" [14], opracowany już w latach osiemdziesiątych. W dużym skrócie, zamiast nasłuchiwania słów i szukania w nich dźwiękowego wzorca, ma on za zadanie przewidzieć prawdopodobieństwo tego, że nieznane dźwięki mogą być słowami.

Proces analizy rozpoczyna się od zebrania informacji dźwiękowej, zwykle przy użyciu mikrofonu, i już na tym stadium pojawiają się pierwsze problemy, takie jak:

- wyłapanie odpowiednich słów spośród dźwięków tła - szumu samochodów na ulicy, lub innych osób rozmawiających w bliskiej odległości.
- w przypadku szybko mówiącego rozmówcy, rozróżnienie w którym momencie wypowiedzi słowa kończą się, oraz kiedy zaczynają się kolejne.

- rozróżnianie różnych akcentów i sposobów mówienia pojedynczych jednostek, na przykład to samo zdanie wypowiedziane przez 10 letnią dziewczynkę, brzmieć będzie zupełnie inaczej, wypowiedziane przez 70 letniego mężczyznę.
- odróżnianie homofonów - słów wymawianych w ten sam sposób, ale mających inne znaczenie
- poprawne zinterpretowanie zdań brzmiących w bardzo podobny sposób, ale oznaczających zupełnie różne rzeczy

Dodatkowo, różne języki mają własną gramatyczną strukturę, co dodatkowo utrudnia stworzenie zunifikowanego systemu, zdolnego do przetwarzania wielu języków w podobny sposób.

Kolejnym krokiem jest przetworzenie zebranych danych na spektrogram przy użyciu Szybkiej Transformaty Fouriera [5], oraz rozszczepienie rezultatu na mniejsze części, które poddawane są analizie mającej na celu określenie w jakich momentach znajdują się słowa. Następnie każde z nich są porównywane pod względem fonetycznego brzmienia i w ten sposób program stara się przewidzieć dokładne słowo, wypowiedziane przez użytkownika.

W dzisiejszych czasach, technologia rozpoznawania mowy w głównej mierze opiera się na algorytmach sztucznej inteligencji połączonych z HMM.

2.4 Bazy danych i ich rodzaje

Sam koncept baz danych jest starszy niż same komputery stacjonarne, przykładowo w postaci szafek kartotekowych. Pierwsza baza danych została zbudowana w latach sześćdziesiątych XX wieku, jednak ich historia w postaci znanej obecnie rozpoczyna się dopiero w 1970 roku, gdy naukowiec Edgar F. Codd opublikował pracę zatytułowaną "A Relational Model of Data for Large Shared Banks" [3]. Przedstawiała ona nowy sposób modelowania danych, przy pomocy budowania połączonych ze sobą tabel, mogących przechować jednorazowo jakąkolwiek formę danych. W tych czasach ilość pamięci była droga, a relacyjna baza danych pozwalała na odpowiednie, efektywne nią zarządzanie. Przez następne dwie dekady, technologia ta preżnie się rozwijała, dominując rynek światowy, a język SQL (Structured Query Language) stał się standardem pozwalającym aplikacjom na dostęp do wszelkich danych, w ilości i postaci w jakiej potrzebowały.

Relacyjne bazy danych zostały zaprojektowane w czasach poprzedzających popularność internetu i były przeznaczone do operowania na jednej maszynie. W przypadku, gdy przytłaczająca ilość danych wykraczała poza możliwości pojedynczego urządzenia, jedynym sposobem

było połączenie wielu serwerów w klastry, a to sprawiało że przeniesienie tradycyjnej bazy danych opartej na SQL, było zadaniem wyjątkowo skomplikowanym i często dającym złe rezultaty. Aby temu zaradzić, zaczęto przykładać uwagę do technologii istniejącej już w latach sześćdziesiątych, a która w 1998 roku została przemianowana na NoSQL (Not only SQL), czyli "Nierelacyjne bazy danych". Nie było to rozwiązanie perfekcyjne, gdyż nie posiadało wielu zalet relacyjnych baz danych, jednak było szybsze i znacznie lepiej skalowalne, cechy które w okresie po internetowej rewolucji WEB 2.0, gdy w bardzo szybkim tempie pojawiało się zapotrzebowanie na kolejne jednostki pamięci, były na wagę złota. Obecnie, coraz to większą popularność zaczynają pozyskiwać chmurowe (cloud database), oraz autonomiczne bazy danych (self-driving database).

Najpopularniejsze rodzaje baz danych [17]

- Relacyjne bazy danych (Relational databases) - Rekordy w bazie zorganizowane są w postaci tabel posiadających wiersze i kolumny. Różne tabele mogą posiadać pomiędzy sobą związki pozwalające im na wspólną kooperację.
- Obiektowe bazy danych (Object-oriented databases) - Dane są przechowywane w postaci obiektów, posiadających w sobie określone pola.
- Rozproszone bazy danych (Distributed databases) - Dane przechowywane mogą być w wielu plikach, na wielu urządzeniach, rozproszone w różnych sieciach.
- Nierelacyjne bazy danych (NoSQL databases) - Pozwalały na przechowywanie i manipulowanie danymi, które nie są ustrukturyzowane, w przeciwieństwie do relacyjnych baz danych, definiujących jak skomponowane mają być dane dostarczone do bazy. Ich popularność rosła wprost proporcjonalnie do ilości aplikacji web-owych i ich złożoności.
- Grafowe bazy danych (Graph databases) - Bazy, gdzie związki pomiędzy danymi są równie ważne, co same dane.
- Bazy danych OLTP (Online Transaction Processing) - Szybkie, analityczne bazy danych, projektowane z myślą o dużej ilości transakcji wykonywanych pomiędzy wieloma użytkownikami.
- Hurtownia danych (Data warehouses) - Baza zaprojektowana pod kątem szybkiej wymiany danych i ich analizy.

- Wielomodelowa baza danych (Multi-model database) - Ich celem jest połączenie wielu typów modeli baz danych w jeden, zintegrowany back-end. Mogą dzięki temu pomieścić wiele typów danych.
- Autonomiczne bazy danych (Self-driving databases) - Najnowsze i najbardziej rewolucyjne, oparte na chmurowych bazach danych, wykorzystujące uczenie maszynowe do zautomatyzowania ich zarządzania.

2.5 Chmurowe bazy danych

Ideą chmurowych baz danych jest zmniejszenie zapotrzebowania firm na zakup własnych fizycznych serwerów, a zamiast tego zaoferowanie im płatnego dostępu do maszyn wirtualnych, na których dana firma mogłaby postawić swoją bazę danych. Czym jednak są maszyny wirtualne?

W skrócie, jest to wirtualne środowisko, funkcjonujące jako komputer, którego komponenty takie jak procesor, pamięć i interfejs sieciowy znajdują się na fizycznym systemie. Możliwe jest jednoczesne istnienie wielu maszyn wirtualnych na jednym systemie, zwanym inaczej "hostem". Dzięki oprogramowaniu "hypervisor", możliwe jest odseparowanie fizycznej maszyny, od wirtualnej, co stwarza bezpieczne środowisko do pracy i w przypadku, gdyby przykładowo na maszynie wirtualnej znalazł się wirus, nie wpłynie on na pracę hosta, jego działanie będzie ograniczone jedynie w zakresie maszyny wirtualnej.

Aby korzystać z możliwości oferowanych przez "chmurę", firma może zdecydować się na stworzenie własnej, prywatnej infrastruktury, lub skorzystanie z udostępnianych publicznie serwisów chmurowych. Do najpopularniejszych obecnie należą:

- AWS - Amazon Web Services
- Google Cloud Platform
- Microsoft Azure
- IBM Cloud
- Oracle Cloud

Dużą zaletą serwisów chmurowych jest możliwość łatwego dostosowania ich pod własne potrzeby, przykładowo w przypadku zapotrzebowania na większą ilość pamięci, lub większą moc

obliczeniową wystarczy zazwyczaj zapłacić więcej za abonament, a nie trzeba się przejmować rozbudowywaniem własnej infrastruktury, co zajęłoby znacznie więcej czasu.

2.6 API

W wielu przypadkach przy tworzeniu aplikacji, programiści muszą zawrzeć informacje, których samodzielne wypisanie i skatalogowanie mogłoby zająć więcej czasu, niż sam proces pisania oprogramowania. Pojawia się również pytanie po co to wszystko robić, skoro wcześniej zrobił to już ktoś inny i dał on dostęp do pożądanych funkcjonalności. Dodatkowo co jeśli dana aplikacja jest zależna od innej usługi, przykładowo w przypadku gdy programista chciałby mieć dostęp do aktualnej pogody w dowolnym rejonie na świecie? Z pomocą przychodzi API, czyli Application Programming Interface, mające za zadanie nie tylko ułatwić programistom tworzenie aplikacji, ale także umożliwić komunikację pomiędzy różnymi usługami. Czym w takim razie dokładnie jest API?

W momencie gdy przeciętny użytkownik chciałby wejść na witrynę internetową "www.facebook.com", aby sprawdzić swój profil, lub przeczytać wiadomości, przeglądarka próbuje wysłać zapytanie do serwera firmy Facebook, z prośbą o przekazanie informacji o tym jak wygląda strona, oraz o dostosowanie jej pod konkretnego użytkownika. Serwer odsyła wtedy potrzebne informacje, które przeglądarka, znana inaczej jako "klient", jest w stanie zinterpretować, a następnie wyświetlić pożądany przez użytkownika wynik. Dla klienta, serwer Facebook'a jest API, co oznacza, że za każdym razem gdy dana osoba chce wejść na jakąkolwiek stronę internetową, wchodzi ona w interakcję z API danego serwera, nie znaczy to jednak, że jest ono tożsame z samym serwerem. Jest to jedynie jego część, odpowiedzialna za przetwarzanie zapytań i wysyłanie odpowiedzi - pakietów informacji mogących przybrać rozmaite formy, w zależności od potrzeb.

Istnieje wiele rodzajów API, mogą się dzielić na różne typy, ale także posiadać odmienne architektury. Głównymi typami API są:

- Open APIs - znane również jako API publiczne, są szeroko dostępne do użytku z minimalnymi ograniczeniami. Mogą być albo w pełni "otwarте", lub wymagać jedynie klucza API - ciągu znaków dzięki którym API będzie mogła zidentyfikować aplikację do której przesyłane będą dane.
- Internal APIs - w przeciwieństwie do Open APIs, są one "ukryte" i przeznaczone do użytku wewnętrz firm, w celu przekazywania zasobów pomiędzy zespołami danej spółki.

- Partner APIs - są one podobne do Open APIs, jednak dostęp do nich jest ograniczony, często wymagający opłaty.
- Composite APIs - typ, mający za zadanie dać dostęp do różnych serwisów i źródeł danych jednocześnie. Są one szczególnie użyteczne w architekturze mikroserwisowej [18], gdzie aby wykonać jedno zadanie, użytkownik może potrzebować informacji od paru serwisów jednocześnie. Używanie Composite APIs może zredukować obciążenie serwera i poprawić płynność działania aplikacji.

Architektury API mają za zadanie sprecyzować ograniczenia nakładane na różne protokoły, definiujące zasady którymi należy się kierować przy wysyłaniu zapytań - jakie komendy należy użyć, oraz jakie typy danych są akceptowalne. Najpopularniejszymi architekturami i protokołami są:

- REST (Representational State Transfer) - w przeciwieństwie do innych wymienionych nie jest protokołem. Kilkoma przykładowymi i najważniejszymi warunkami jakie powinien spełniać interfejs API RESTful korzystający z protokołu HTTP są [11]:
 - Interfejsy API REST są oparte na zasobach — dowolnym obiekcie, danych lub usłudze, które są dostępne dla klienta.
 - Zasób ma identyfikator URI służący do unikatowej jego identyfikacji.
 - Interakcja klientów z usługą odbywa się przez wymianę reprezentacji zasobów. Najczęściej używanym formatem wymiany danych jest notacja JSON.
 - Interfejsy API REST korzystają z ujednoliconego interfejsu, co ułatwia rozdzielenie implementacji klienta od implementacji usługi.
 - Interfejsy API REST korzystają z bezstanowego modelu żądań. Żądania HTTP powinny być niezależne i mogą występować w dowolnej kolejności, dlatego zachowywanie informacji o stanie przejściowym między żądaniami nie jest możliwe. Informacje są przechowywane jedynie w zasobach, a każde żądanie powinno być niepodzielną operacją.
 - Interfejsy API REST są sterowane za pomocą hipermedialnych linków, zawartych w reprezentacji.
- JSON-RPC i XML-RPC (Remote Procedural Call Protocol) - żądanie może posiadać wiele parametrów, ale oczekiwany jest tylko jednego wyniku. W przeciwieństwie do architektury

REST, nie działa na zasobach, ale na wywoływaniu odpowiednich metod, a identyfikator URI zamiast zasobu identyfikuje serwer, stąd nie przechowuje żadnych informacji w swoich parametrach. Oba protokoły różnią się od siebie jedynie notacją, za pomocą której są zakodowane wywołania.

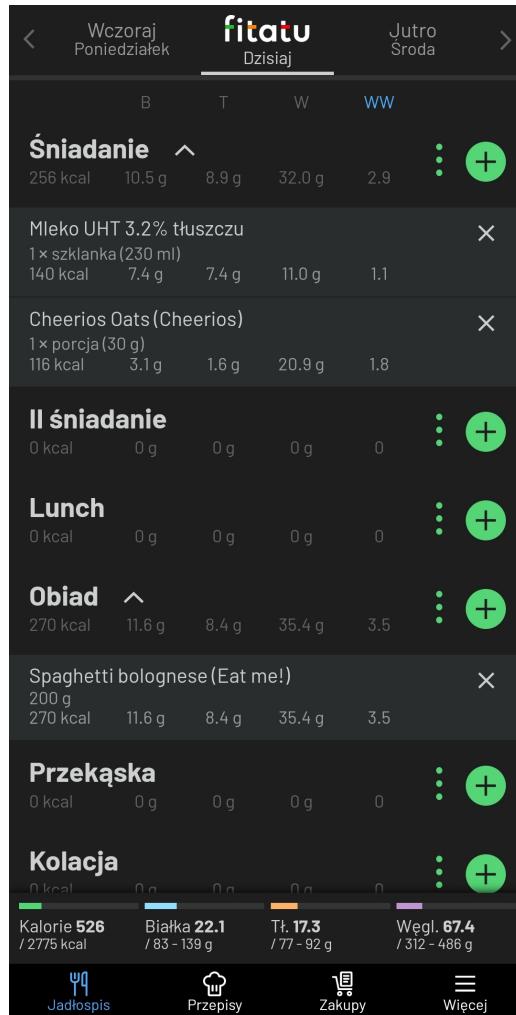
- SOAP (Simple Object Access Protocol) - protokół oparty na notacji XML, używający do komunikacji protokołu HTTP. Został on zaprojektowany jako pośrednik, umożliwiający łatwą komunikację pomiędzy aplikacjami napisanymi w różnych językach programowania.

2.7 Analiza konkurencyjnych rozwiązań

Istnieje bardzo dużo aplikacji oferujących możliwość liczenia kalorii, jednak w większości różnią się one jedynie szatą graficzną, oraz dostępem do funkcjonalności, których część jest nierzadko niedostępna do darmowego użytkowania.

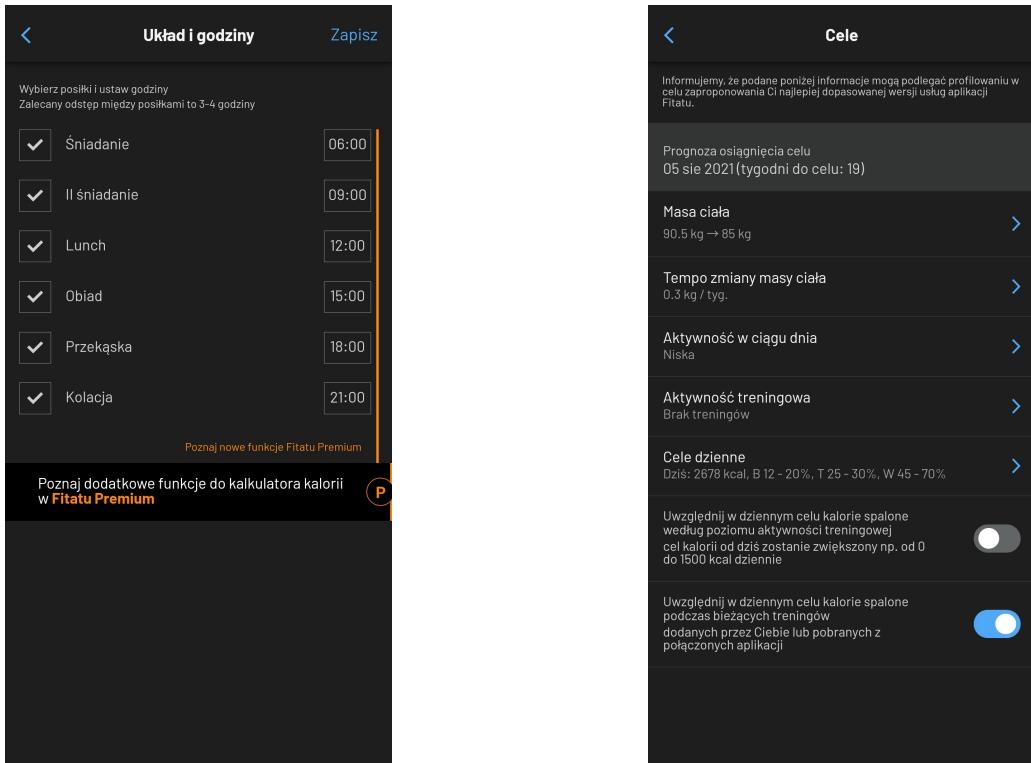
2.7.1 Fitatu

Jest to aplikacja najprostsza w obsłudze z pozostałych rozwiązań, jako że stara się zawrzeć jak najwięcej informacji i funkcjonalności na głównej stronie co jednocześnie ułatwia jej użytkowanie, ale przekłada się na jej czytelność, ukazując duże ilości tekstu, co może przytłoczyć nowego użytkownika.

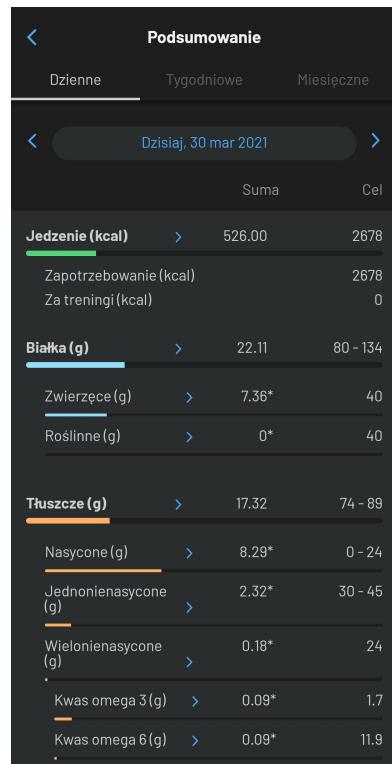


Rysunek 4: Fitatu - Ekran główny

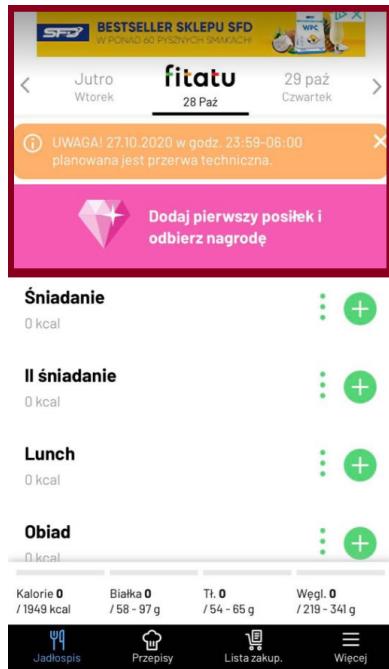
Zalety	Wady
Rozpisanie najpopularniejszych posiłków jednego pod drugim, co ułatwia rozplanowanie kolejnych dań	Bardzo duży rozmiar banerów informacyjnych, zasłaniających większą część interfejsu
Menu aplikacji jest modyfikowalne, pozwala na określenie w których godzinach jest spożywany dany posiłek, oraz na łatwe zmodyfikowanie celów użytkownika.	Mały rozmiar paska podsumowującego spożyte danego dnia wartości odżywcze
Aplikacja posiada bardzo dokładną listę najróżniejszych składników odżywcznych, rozpisanych w bardzo przejrzysty sposób, co pozwala na niezwykle szczegółowe rozplanowanie diety	Bliskie rozłożenie przycisków do dodawania posiłków, co przekłada się na mniejszy komfort użytkowania



Rysunek 5: Fitatu - Ekrany modyfikacji



Rysunek 6: Fitatu - Ekran szczegółowego podsumowania



Rysunek 7: Fitatu - Ekran zawierający banery informacyjne

2.7.2 Yazio

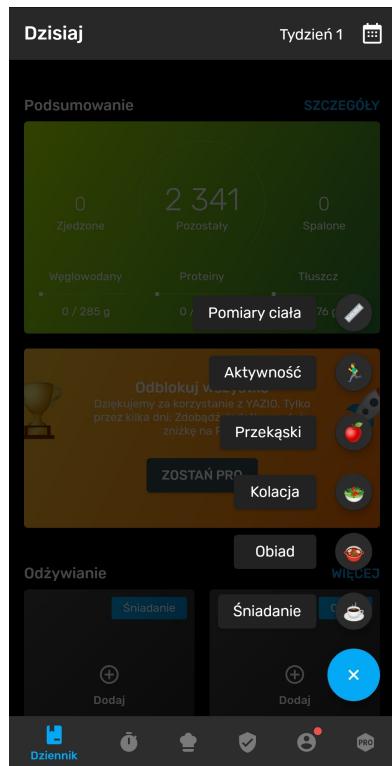
Yazio jest aplikacją najbardziej ograniczoną pod względem funkcjonalności dostępnych w wersji darmowej, aby mieć dostęp do większości z nich należy posiadać płatną wersję PRO. Do zalet aplikacji należą:

- Duże pole podsumowujące dzień usytuowane na środku ekranu, w przejrzysty sposób prezentujące informacje o postępie danego dnia.



Rysunek 8: Yazio - Ekran główny

- Dodawanie jedzenia i interesujących użytkownika aktywności znajduje się pod jednym przyciskiem "+"



Rysunek 9: Yazio - Ekran z klikniętym przyciskiem "+"

Największą wadą aplikacji jest fakt jak bardzo ograniczona jest ona w wersji darmowej. W przypadku chęci sprawdzenia informacji o zjedzonych wartościach odżywcznych, użytkownik ma dostęp jedynie do ilości zjedzonych kalorii, węglowodanów, protein, oraz tłuszcza, natomiast cała reszta, przykładowo witaminy nie są dostępne za darmo.

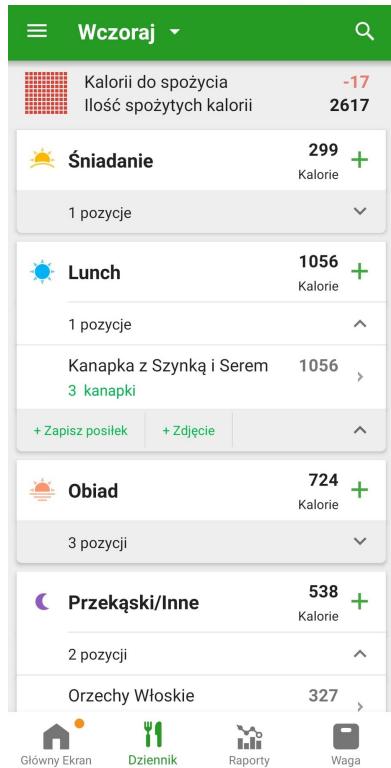


Rysunek 10: Yazio - Ekran podsumowujący

Jeśli użytkownik nie chce inwestować pieniędzy, spośród konkurencji Yazio wypada najgorzej.

2.7.3 Fatsecret

Pod względem wizualnym, Fatsecret łączy najlepsze cechy konkurencji, prezentując główny ekran podobny do Fitatu, jednak przedstawiony w sposób dużo bardziej minimalistyczny, stąd czytelniejszy, a także posiadający ciekawie przedstawione podsumowanie w postaci kwadratowej siatki, zapełnianej wraz z postępem zjedzonych posiłków. Nie jest to jednak rozwiązanie tak czytelne, jak przypadku aplikacji Yazio.

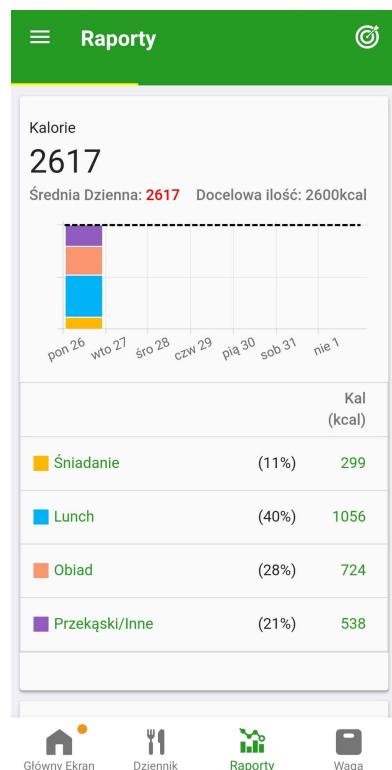


Rysunek 11: Fatsecret - Ekran główny

Zalety	Wady
Podobnie jak w przypadku Fitatu, rozpisanie najpopularniejszych posiłków jednego pod drugim, jednak w dużo czytelniejszy sposób	Ilość wartości odżywczych, która jest mniej dokładna, niż w przypadku aplikacji Fitatu, jednak zdedykowanie większa niż w bezpłatnej wersji Yazio
Menu podsumowujące znajdujące się tuż pod posiłkami i przedstawiające dane w postaci aż dwóch różnych diagramów.	Do odblokowania wszystkich funkcji aplikacja zmusza użytkowania do założenia konta.
Sekcja "Raporty" pozwala zobaczyć podsumowanie na przestrzeni dłuższego okresu czasu.	



Rysunek 12: Fatsecret - Ekran podsumowujący



Rysunek 13: Fatsecret - Ekran raportu

2.8 Wymagania funkcjonalne

Wymagania funkcjonalne jest to grupa założeń, opisujących co ma realizować system, jak ma się zachowywać w określonych sytuacjach, jakie usługi ma dostarczać i jaką ma pełnić funkcję.

- Formularz rejestracyjny powinien pozwolić w prosty i szybki sposób założyć konto w aplikacji, co pozwala użytkownikowi przechowywać informacje w chmurowej bazie danych.
- Panel logowania ma udostępniać formularz identyczny do panelu rejestracji, jednak pozwolić ma na uzyskanie dostępu do konta na które rejestrowane są zjedzone produkty.
- Zjedzone produkty mogą zostać dodane do listy na 2 sposoby:
 - Przy pomocy pola tekstowego, poprzez wpisanie odpowiedniej frazy.
 - Przy pomocy przytrzymania przycisku rozpoczęjącego nasłuchiwanie telefonu, w celu rozpoznawania mowy. Aplikacja nie obsługuje wielu fraz wypowiadanych jednocześnie, więc wyśle ona zapytanie do API żywonościowego załączając tylko pierwsze wypowiadane słowo.

Po dodaniu jedzenia, zostaje ono zapisane w bazie danych, a następnie można je wyświetlić po przejściu na ekran podsumowujący. Przy pomocy kalendarza, użytkownik ma także możliwość zmienić obecnie wyświetlany dzień, w celu sprawdzenia produktów spożytych w innej dacie.

- Całkowita ilość spożytych kalorii jest na bieżąco sumowana i wyświetlana na ekranie głównym. Wartość ta jest zmieniana w zależności od obecnie wyświetlonej daty.

2.9 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne opisują ograniczenia, przy zachowaniu których system powinien realizować swoje zadania. Dotyczą przykładowo niezawodności, bezpieczeństwa, przenośności [9].

- Dane użytkownika mają być dostępne do odczytu tylko i wyłącznie dla niego, po zalogowaniu na konto.
- W przypadku błędnie wpisanej frazy podczas wyszukiwania, aplikacja ma zwrócić okno z tekstoną wiadomością, informującą użytkownika o problemie przy znalezieniu produktu.

- Aplikacja powinna dostosować szatę graficzną do różnych rozdzielczości ekranu.
- Aplikacja nie powinna móc działać w tle i wykorzystywać zasoby.
- Aplikacja ma być zaprojektowana pod system operacyjny "Android".
- Aplikacja oczekuje dostępu do mikrofonu.
- Aplikacja ma mieć prosty interfejs i pozwolić użytkownikowi na szybką naukę jej obsługi.

3 Technologie i narzędzia wykorzystane w projekcie

3.1 Dart

Dart jest językiem programowania opracowanym przez firmę Google i pierwszy raz ukazanym na konferencji Aarhus w 2010 roku. Jego głównym celem było wprowadzenie konkurencji dla JavaScriptu, dominującego języka używanego w przeglądarkach internetowych. Cechuje go fakt, że jest językiem programowania obiektowego, opartym na klasach ze składnią podobną do języków z rodziny "C", a jego kod może zostać skompilowany na zarówno kod maszynowy, przy użyciu maszyny wirtualnej Dart VM, jak i przy pomocy kompilatora "dart2js", na kod JavaScript, co przekłada się na jego kompatybilność z wieloma obecnymi przeglądarkami internetowymi. Dart VM umożliwia łatwe uruchamianie kodu na różnych platformach, bez potrzeby pisania aplikacji pod konkretny system, a także oferuje dwa różne tryby kompilacji [4]:

- Just-in-time compiler (JIT) - pozwala na inkrementalną kompilację, umożliwiającą szybkie sprawdzenie wyniku wprowadzonych w kodzie zmian, z dostępem do wielu narzędzi służących debuggowaniu, bez potrzeby oczekiwania na całkowitą kompilację całego programu.
- Ahead-of-time compilation (AOT) - w momencie gdy aplikacja jest gotowa do wydania, Dart AOT umożliwia kompilację na kod maszynowy dla architektury ARM (używanej głównie w procesorach, lub urządzeniach mobilnych), bądź x86_64 (charakterystycznej dla komputerów stacjonarnych).

Dart wspiera pojedyncze dziedziczenie, gdzie klasa podzielona może dziedziczyć tylko po jednej klasie nadzędnej, interfejsy, domieszki, klasy abstrakcyjne, a także typy opcjonalne. O ile język ten wciąż nie zastąpił JavaScript na rynku aplikacji webowych, tak zyskuje coraz większą popularność dzięki platformie programistycznej opracowanej przez Google, o nazwie "Flutter".

3.2 Flutter

Flutter jest względnie młodym frameworkiem, gdyż został pierwszy raz zaprezentowany w roku 2015 i miał początkowo służyć jako nowoczesna platforma do tworzenia aplikacji na system Android, a jego wersja alfa (0.0.6) została wydana w 2017 roku. Opracowany na bazie języka programistycznego Dart, Flutter skupia się na udostępnieniu narzędzi do szybkiego i łatwego projektowania interfejsu użytkownika, będąc przy tym bardzo wydajnym. Obecnie, umożliwia on używanie tego samego kodu na wielu platformach, takich jak Android, czy iOS, jako że w przeciwieństwie do konkurencyjnych rozwiązań, nie wykorzystuje on narzędzi oferowanych przez dany system [6]. Zamiast tego, Flutter opiera się w pełni na własnym, zoptymalizowanym systemie "widget'ów", co pozwala wyróżnić go pod względem szybkości i wydajności, i tak jak w przypadku systemu Windows zwykło się mówić, że wszystko w Windows jest "oknem", tak we Flutterze wszystko jest "widgetem". Tworzą one wspólnie drzewo hierarchiczne, gdzie każdy widget dziedziczy po innym, nadzędnym komponencie, aż do widgetu będącego kontenerem przechowującym całą aplikację, zwanego zazwyczaj MaterialApp, lub CupertinoApp. Sam framework Flutter'a jest relatywnie mały i ograniczony, a większość bardziej zaawansowanych komponentów jest udostępniana w formie pakietów i pluginów, które programiści w zależności od potrzeb mogą dowolnie załączać do swojego projektu.

3.3 Cloud Firestore

Cloud Firestore jest jedną z usług dostępnych w Google Firebase - platformie której głównym celem jest wspomaganie twórców aplikacji mobilnych poprzez oferowanie zestawu gotowych narzędzi. Firestore jest nierelacyjną, chmurową bazą danych, która jest w stanie w czasie rzeczywistym informować aplikację o zmianach w swojej bazie, co pozwala na ciągłe wyświetlanie najnowszych danych, bez potrzeby specjalnego, interwałowego ich sprawdzania. Posiada ona wsparcie dla trybu offline i w przypadku gdy urządzenie utraci dostęp do internetu, Cloud Firestore buforuje dane, z których aktywnie korzysta aplikacja, a następnie po odzyskaniu połączenia synchronizuje wszelkie lokalne zmiany z danymi znajdującymi się w chmurze. Dane w Firestore przechowywane są zgodnie z modelem danych NoSQL, w dokumentach zawierających pola mapowane na wartości. Przetrzymywane są one w kolekcjach - kontenerach na dokumenty, których można używać do organizowania danych i tworzenia zapytań. Dokumenty są w stanie obsłużyć wiele różnych typów danych, od prostych łańcuchów i liczb, po złożone, zagnieżdżone obiekty. Ponadto, dane mogą być odczytywane na poziomie

dokumentu, bez konieczności pobierania całej kolekcji lub jakichkolwiek zagnieżdżonych pod-kolekcji.

3.4 Android Studio

Android Studio jest oficjalnym narzędziem do tworzenia aplikacji na system operacyjny "Android", opartym na środowiskach programistycznych firmy IntelliJ i oprócz standardowego zestawu narzędzi i edytora kodu, oferuje szereg ułatwień z myślą o aplikacjach mobilnych. Jako fundacji do budowania rozwiązania, Android Studio używa "Gradle", uznanego oprogramowania open-source służącego do automatyzacji całego procesu. Do łatwego testowania aplikacji, Android Studio posiada wbudowany emulator, pozwalający na ściągnięcie dowolnej, pożąданiej wersji systemu operacyjnego, na telefony o różnych rozmiarach ekranu. Środowisko oferuje również wsparcie dla systemu kontroli wersji "Git", stąd proces zapisania wszelkich zmian i wysłania ich na repozytorium może zostać w pełni wykonany przy pomocy Android Studio.

3.5 Android Emulator

Do aktywnego sprawdzania aplikacji mobilnych na desktopach, używa się specjalnych emulatorów, których zadaniem jest stworzenie wirtualnego środowiska przypominającego w działaniu realne urządzenie przenośne, poprzez naśladowanie sprzętu znajdującego się w urządzeniach mobilnych. Odzwierciedlenie realnego procesora jest zadaniem wyjątkowo wymagającym pod względem zasobów, przekładającą się na szybkość z jaką działa dany emulator, jednak dzięki wspomnianemu wcześniej w przypadku maszyn wirtualnych oprogramowaniu "Hypervisor", gdy architektury procesorów urządzenia emulowanego jak i komputera stacjonarnego są ze sobą zgodne, możliwe jest użycie procesora hosta, jako procesora w emulowanym urządzeniu.

Największą zaletą używania emulatorów jest oczywiście prędkość, z jaką można testować swoją aplikację, bez zbędnej potrzeby budowania jej, a następnie eksportowania na posiadane urządzenie mobilne, instalowania, a w końcu sprawdzania. Inną mniej oczywistą zaletą, jest testowanie aplikacji pod kątem interakcji z fizycznymi sensorami, jak przykładowo akcelerometr, gdyż łatwo jest dostosować ustawienia w wirtualnym środowisku, aby następnie mogły one poprawnie działać już na realnym urządzeniu.

Wraz z zaletami pojawiają się także ograniczenia. Najpopularniejszą architekturą dla urządzeń mobilnych jest ARM v7a, jednak większość desktopów używa x86_64, stąd bez procesora z architekturą ARM w większości przypadków nie ma nawet okazji skorzystać z przyspieszonej

emulacji, wspomnianej wcześniej. Stworzenie jednego wirtualnego urządzenia może zająć nawet do 10GB miejsca na dysku, stąd posiadanie wielu różnych urządzeń do testowania w znacznym stopniu ogranicza dostępną pamięć systemową. Wydajność z jaką operuje emulator jest ściśle związana z urządzeniem na którym jest on uruchomiony, stąd może on nie tylko działać wolniej niż zazwyczaj, ale także "wyrzucać" błędy, niekoniecznie związane z działaniem samej aplikacji, lub emulatora.

Emulatory nigdy nie będą w stanie w pełni oddać działania prawdziwego urządzenia i nie będą w stanie zaoferować wszystkich dostępnych w urządzeniach mobilnych możliwości, także testowanie na fizycznym urządzeniu zawsze będzie procederem wymaganym, do prawidłowego projektowania aplikacji mobilnych.

3.6 VIM

VIM jest zaawansowanym edytorem tekstu opracowanym w 1991 roku, opartym na edytorze "vi", domyślnie dołączanym do większości systemów UNIX. Jest dystrybuowany jako "charityware" - dostęp do niego jest darmowy, ale twórca zachęca do wpłaty pieniędzy na cele charytatywne. VIM charakteryzuje szeroki wachlarz możliwości jakie oferuje, a w odpowiednich rękach pozwala bardzo zwiększyć wydajność pracy. Używanie VIMa nie wymaga myszki, jego głównym celem jest ograniczenie konieczności poruszania ręką pomiędzy klawiaturą, a myszką, na rzecz wykonywania każdej czynności przy takim samym ustawieniu rąk. Wraz z rozwojem aplikacji, stale dodawane są nowe funkcjonalności, mające w jeszcze większym stopniu usprawnić proces pisania. VIM zezwala na modyfikowanie i dodawanie własnych skrótów klawiszowych, oraz posiada wiele opcji konfiguracji, stąd zależnie od preferencji i potrzeb można go dostosować do wielu postaci.

Do podstawowych opcji udostępnianych przez VIM można zaliczyć:

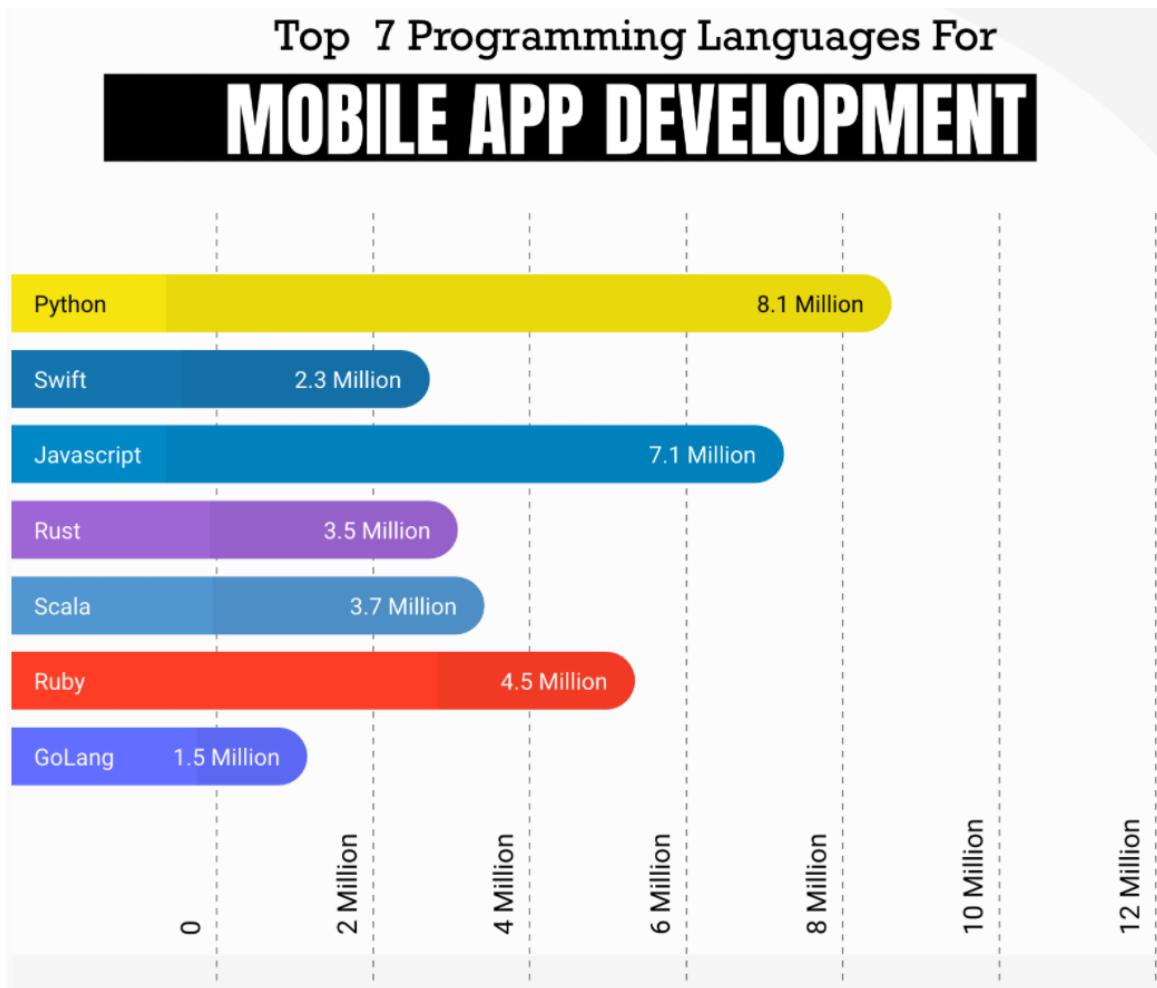
- poruszanie się kursorem w trybie normalnym przy użyciu takich klawiszy jak:
"h, j, k, l, ..."
- do przełączania się pomiędzy różnymi trybami służą klawisze: "i, a, o, v, ESC, :, ..."

Przedstawione wyżej opcje to tylko wierzchołek góry lodowej, a poprawne nauczenie się możliwości oferowanych przez VIM może zająć wiele miesięcy.

4 Proces powstawania aplikacji

4.1 Aplikacja od strony logiki

Proces tworzenia aplikacji rozpoczął się od podjęcia decyzji w jakiej technologii powinna zostać ona napisana. Różne języki programowania charakteryzuje szereg zalet i wad, przykładowo najpopularniejszym obecnie wykorzystywanym przy projektowaniu aplikacji mobilnych językiem jest Python,

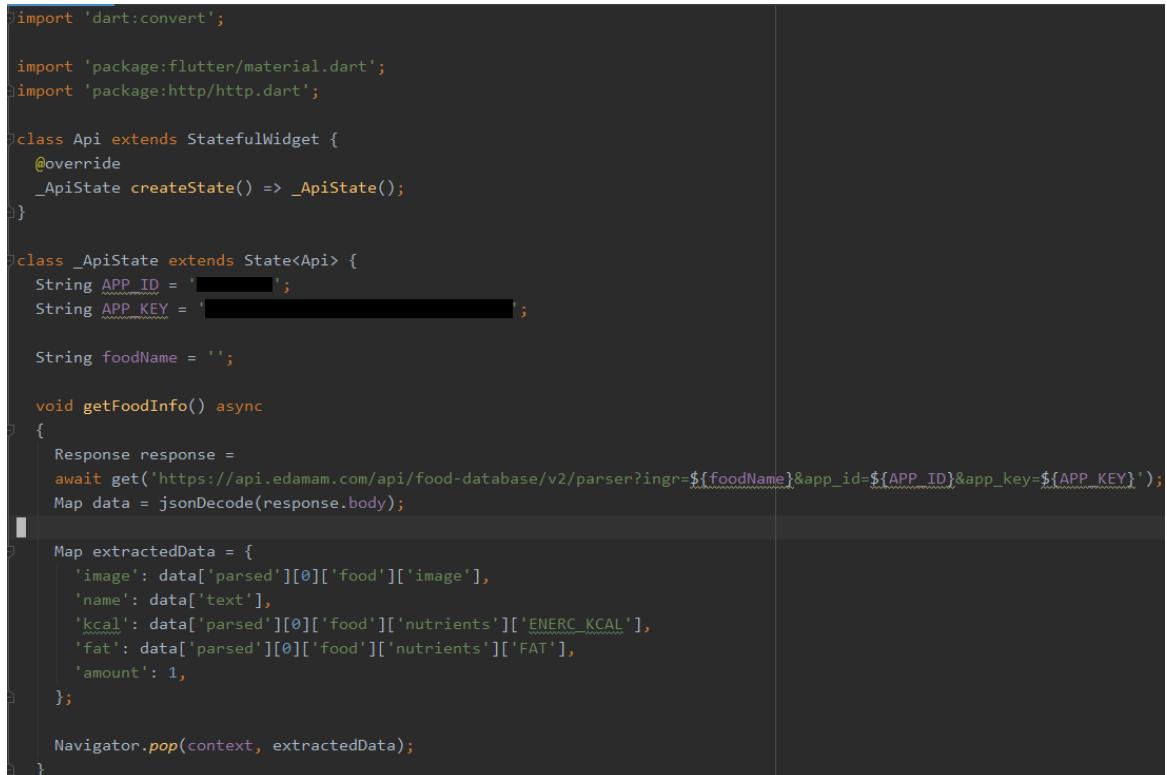


Rysunek 14: Najczęściej wykorzystywane języki do tworzenia aplikacji mobilnych

ze względu na swoją prostotę i czytelność kodu, jednak jego największą wadą jest bardzo słaba optymalizacja, przekładająca się bezpośrednio na szybkość działania aplikacji. Rozwiązaniem dla tego typu problemu mogła być przykładowo Java, bądź podobny do niej, choć w większym stopniu nastawiony na pisanie aplikacji mobilnych Kotlin. Mniej znanym, jednak zyskującym popularność frameworkiem, stworzonym w pełni z myślą o aplikacjach mobilnych jest Flutter, wprowadzający dużo ciekawych rozwiązań rewolucjonizujących proces pisania aplikacji mobilnych. Dzięki własnemu, wbudowanemu systemowi widgetów, pozwala na bezproblemowe

tworzenie wieloplatformowych, dobrze zoptymalizowanych aplikacji, a także w bardzo dużym stopniu ułatwia projektowanie interfejsu użytkownika.

Po ostatecznym wyborze Flutter'a jako domyślnej technologii, nadszedł czas na zaprojektowanie podstawowej logiki aplikacji. Najważniejszym jej elementem, jest pozyskiwanie danych o zjedzonych produktach, dlatego należało wybrać odpowiednie API żywieniowe, a następnie zaimplementować komunikację, między nim, a aplikacją.



```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart';

class Api extends StatefulWidget {
  @override
  _ApiState createState() => _ApiState();
}

class _ApiState extends State<Api> {
  String APP_ID = 'XXXXXXXXXX';
  String APP_KEY = 'XXXXXXXXXXXXXX';

  String foodName = '';

  void getFoodInfo() async
  {
    Response response =
    await get('https://api.edamam.com/api/food-database/v2/parser?ingr=${foodName}&app_id=${APP_ID}&app_key=${APP_KEY}');
    Map data = jsonDecode(response.body);

    Map extractedData = {
      'image': data['parsed'][0]['food']['image'],
      'name': data['text'],
      'kcal': data['parsed'][0]['food']['nutrients'][ENERC_KCAL],
      'fat': data['parsed'][0]['food']['nutrients'][FAT],
      'amount': 1,
    };

    Navigator.pop(context, extractedData);
  }
}
```

Rysunek 15: Moduł komunikujący się z API żywnościovym

Do pozyskiwania informacji o składnikach odżywcznych zostało wykorzystane API od firmy "Edamam", jako że udostępniało korzystniejszą darmową ofertę, w porównaniu do innych konkurencyjnych API. Jego wadą jest relatywnie niewielka ilość danych żywieniowych, jednak nie nakłada ono ograniczeń do ilości wysyłanych zapytań. Początkowo, komunikacja oparta była jedynie na wpisaniu frazy do pola tekstowego, a następnie potwierdzeniu za pomocą przycisku. Informacja o jedzeniu była następnie przekazywana do obiektu API, jako oddzielnego ekranu, co ułatwiło wymianę danych. Ekrany we Flutterze są układane na siebie w postaci stosu, gdzie podstawowym jest ekran opisany jako główny, a każdy kolejny nakładany jest na poprzedni. Po znalezieniu pożdanego wyniku i zwróceniu go przez API, ekran ten był usuwany ze stosu, a pozyskane dane przekazywane były do ekranu głównego.

```
    ),
    decoration: InputDecoration(
        labelText: 'Input food name',
    ), // InputDecoration
    onChanged: (String foodName) {
        this.foodName = foodName;
    },
), // TextField
SizedBox(height: 20.0),
FlatButton.icon(
    onPressed: () async {
        final data = await Navigator.pushNamed(context, '/loading', arguments: this.foodName);
        setState(() {
            foodData = data;
        });

        if(foodData != null)
        {
            bool isFound = false;
            for(final i in foodConsumed)
            {
                if(i['name'] == foodData['name'])
                {
                    i['amount'] += 1;
                    isFound = true;
                    break;
                }
            }
            if(!isFound){
                foodConsumed.add(foodData);
            }
        }
    },
    icon: Icon(Icons.fastfood),
    label: Text('Search food info'),
), // FlatButton.icon
```

Rysunek 16: Pole tekstowe, oraz przycisk zatwierdzający wpisaną frazę

Informacje pozyskane z API przechowywane były w odpowiedniej liście. Dodany został przycisk, pozwalający przejść na ekran ukazujący wszystkie wyszukane produkty i wyświetlający je w formie wierszy.

```
Widget fillList(List<Map> data){  
    List<Widget> list = new List<Widget>();  
    data.forEach((element) {  
        list.add(Padding(  
            padding: const EdgeInsets.all(8.0),  
            child: Row(  
                children: [  
                    Image.network(element['image'], scale: 3.0),  
                    SizedBox(width: 8,),  
                    Text(element['amount'].toString() + 'x'),  
                    SizedBox(width: 8,),  
                    Text(element['name'].toString().capitalize()),  
                    SizedBox(width: 8,),  
                    Text((element['kcal'] * element['amount']).toString()),  
                    SizedBox(width: 8,),  
                    Text((element['fat'] * element['amount']).toString()),  
                ],  
            ), // Row  
        )); // Padding  
    });  
    return new Column(children: list);  
}
```

Rysunek 17: Metoda wyświetlająca listę zjedzonych produktów w postaci wierszy

Następnym ważnym krokiem było zaimplementowanie speech to text i zintegrowanie go z projektem. Istnienie biblioteki dla Flutter'a dającej dostęp do wbudowanego w telefon i opartego na technologii Google rozpoznawania głosu w dużym stopniu ułatwiło cały proces. Moduł ten wymaga od użytkownika zezwolenia na używanie mikrofonu, dostępu do internetu, a także najnowszej wersji domyślnej w większości systemów aplikacji "Google", do poprawnego jego działania. Po przytrzymaniu przycisku ze znakiem mikrofonu, telefon wydaje dźwięk sygnalizujący rozpoczęcie nasłuchiwanego, a następnie po odsłuchaniu wypowiedzianej frazy, wysyła zapytanie do API żywonościowego zwracającego pożądany produkt.

```

void listen() async {
    if(!_isListening) {
        bool available = await _speech.initialize(
            onStatus: (val) => print('onStatus: $val'),
            onError: (val) => print('onError: $val'),
        );
        if(available) {
            setState(() {
                _isListening = true;
            });
            _speech.listen(
                onResult: (val) => setState(() {
                    _text = val.recognizedWords;
                    if(val.hasConfidenceRating && val.confidence > 0) {
                        _confidence = val.confidence;
                    }
                }),
            );
        }
    } else {
        setState(() {
            _isListening = false;
        });
        _speech.stop();
    }
}

```

Rysunek 18: Metoda odpowiedzialna za nasłuchiwanie

Po otrzymaniu działającego systemu, należało go zoptymalizować i doprowadzić do czytelniejszej, a także efektywniejszej postaci. Zostały dodane alert boxy, informujące użytkownika o nieprawidłowych stanach aplikacji i zabezpieczające jej działanie.

Do tego momentu, wszelkie informacje o zjedzonych produktach były przechowywane w pamięci jedynie w czasie działania aplikacji, a po wyłączeniu jej zostały utracone. W związku z tym, ostatnim dużym krokiem było przechowywanie danych w zewnętrznej bazie danych, aby użytkownik miał do nich dostęp niezależnie od urządzenia z którego korzysta. Odpowiednią do tego celu usługą okazał się być Google Cloud Firestore, nierelacyjna baza danych przechowująca informacje w dokumentach i kolekcjach. Aby zapewnić poprawne i bezpieczne odczytywanie danych, został dodany ekran rejestracji, oraz logowania, nadające każdemu użytkownikowi pry-

watne identyfikatory, na podstawie których baza decyduje skąd pobierane i dokąd wysyłane będą informacje o zjedzonych produktach.

```
Future updateUserData(Food food) async {
    if (food != null) {
        return await foodCollection
            .document(uid)
            .collection(selectedDate)
            .document(food.name)
            .setData({
                'name': food.name ?? '',
                'kcal': food.kcal ?? '0',
                'fat': food.fat ?? '0',
                'image': food.imageUrl ?? null,
                'amount': food.amount ?? 0
            });
    } else {
        throw Exception('No food data transferred to database');
    }
}

Future<List<Food>> getUserData(String date) async {
    List<Food> foods = new List();
    return await foodCollection
        .document(uid)
        .collection(date)
        .getDocuments()
        .then((value) {
            value.documents.forEach((doc) {
                foods.add(Food(
                    name: doc.data['name'] ?? '',
                    kcal: doc.data['kcal'] ?? 0,
                    fat: doc.data['fat'] ?? 0,
                    imageUrl: doc.data['image'] ?? null,
                    amount: doc.data['amount'] ?? 0,
                ));
            });
            return foods;
        });
}
```

Rysunek 19: Metody wysyłające, oraz odbierające dane z bazy danych

Działający system komunikacji z bazą danych pozwolił na dodanie do aplikacji kalendarza, umożliwiającego po kliknięciu w dany dzień, sprawdzenie jego podsumowania.

Posiadając już wszystkie najważniejsze funkcjonalności, ostatnim krokiem było dodanie paru mniejszych modułów, jak przykładowo przedstawianie informacji w postaci kołowych "progress barów", rozwinięcie już istniejących modułów o dodatkowe metody, jak i uporządkowanie kodu, oraz przebudowanie interfejsu, aby był on faktycznie przejrzysty i funkcjonalny dla przeciętnego użytkownika.

4.2 Aplikacja od strony użytkownika

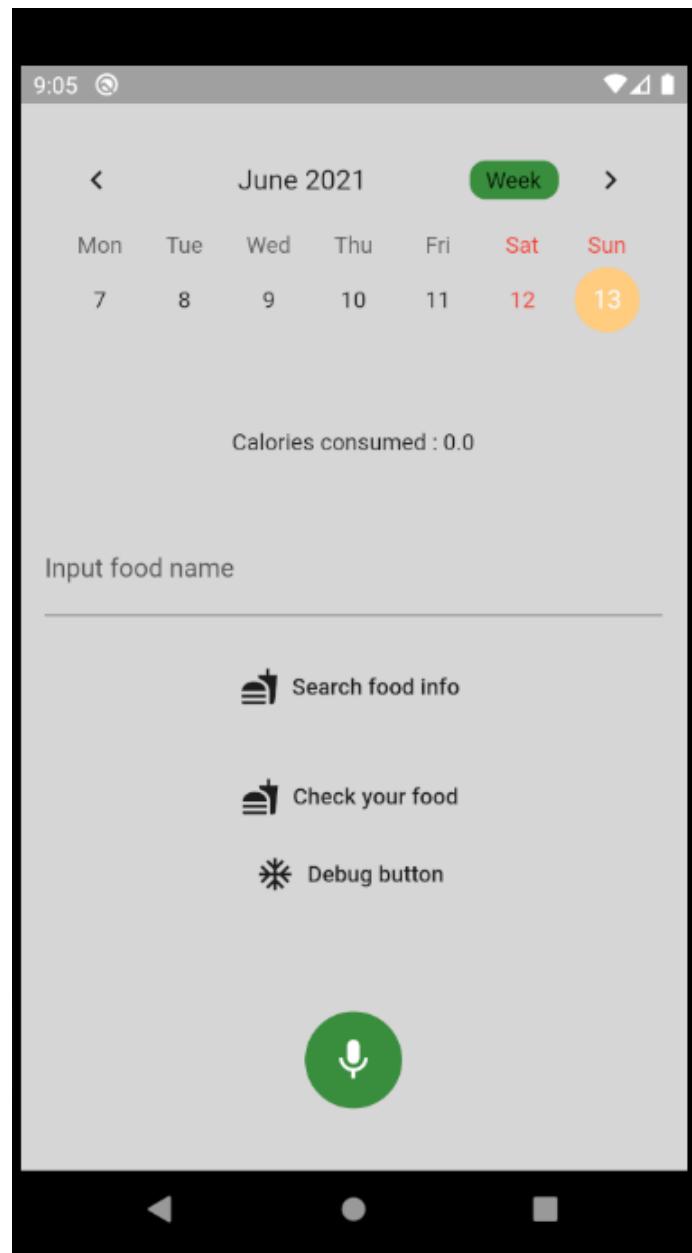
Początkowo, w fazie rozwijania aplikacji jej wygląd służył jedynie testowaniu funkcjonalności, przedstawiając jedynie biały ekran z kilkoma opcjami.



Rysunek 20: Pierwsza wersja ekranu głównego

W tej wersji kalendarz nie był jeszcze działający a cała górná sekcja ekranu służyła wyłącznie w celach podgladowych. Po wpisaniu odpowiedniej frazy w pole tekstowe, a następnie kliknięciu guzika "Search food info", aplikacja łączyła się z API żywieniowym i zwracała informacje o znalezionym jedzeniu. Całą listę można było następnie zobaczyć po kliknięciu przycisku "Check your food", jednak lista ta była ukazywana w formie wierszy zawierających jedynie połączony tekst przedstawiający same wartości liczbowe zjedzonych składników. Informacje te były nieczytelne dla jakiegokolwiek użytkownika. Aplikacja posiadała już jednak

działający moduł "speech to text", jednak nie był on w żaden sposób zabezpieczony i często był powodem zaprzestania działania aplikacji.

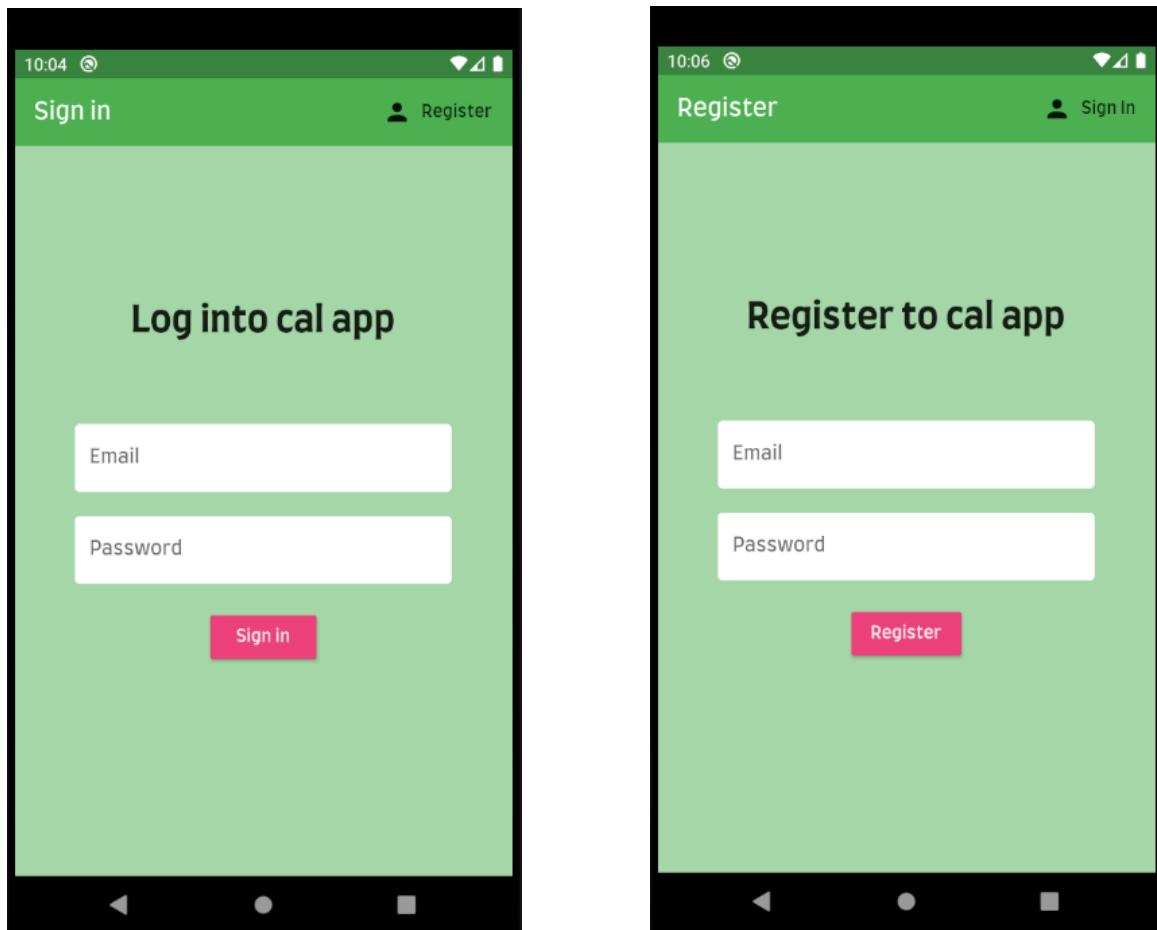


Rysunek 21: Ekran główny po dodaniu kalendarza

Po dodaniu kalendarza ekran zaczął coraz bardziej przypominać finalną formę, chociaż brakowało w nim jeszcze kilku funkcjonalności. Suma zjedzonych danego dnia kalorii przedstawiana była w postaci tekstu i służyła w głównej mierze testowaniu działania kalendarza, brakowało także informacji o innych wartościach odżywcznych. W wersji tej znajdował się także przycisk, którego działanie było stale zmieniane w zależności od potrzeb przetestowania danej funkcjonalności. Moduł "speech to text" był już w pełni działający, a użytkownik poprzez przytrzymanie zielonego przycisku przedstawiającego mikrofon mógł wyszukać interesującą go żywność.

Finalna wersja interfejsu aplikacji

Pierwszym ekranem jaki napotka nowy użytkownik jest ekran logowania.

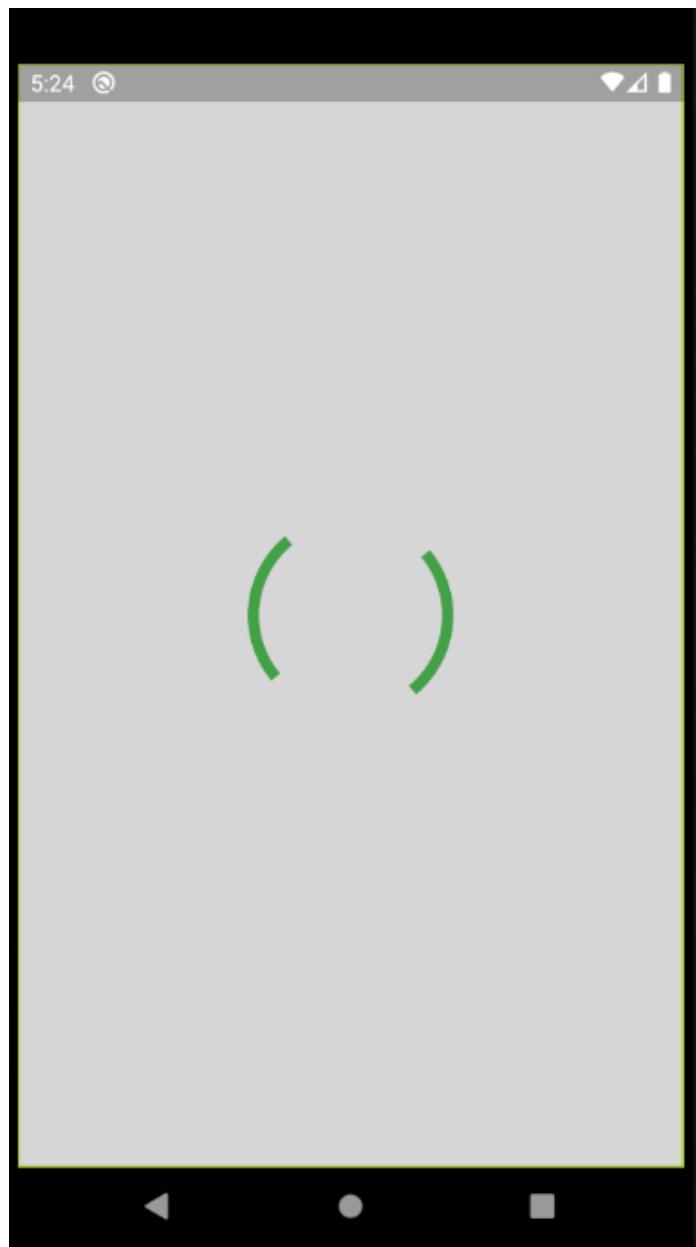


(a) Ekran logowania

(b) Ekran rejestracji

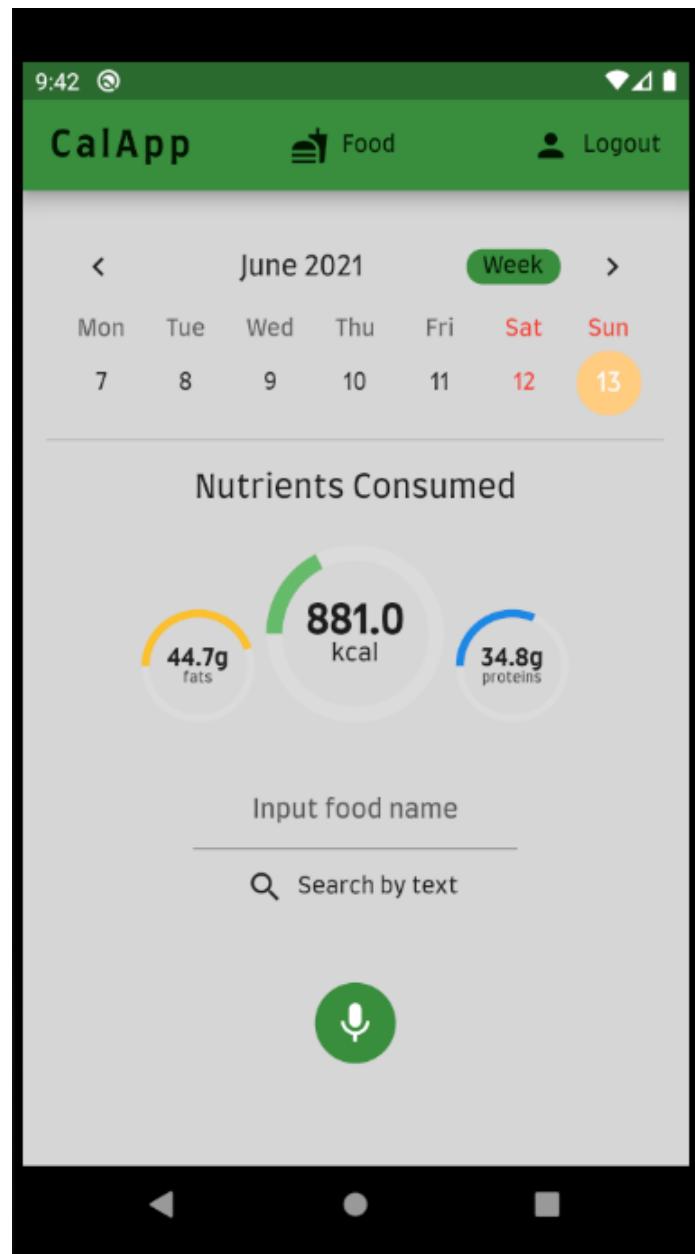
Rysunek 22: Ekrany logowania i rejestracji

W przypadku, gdy użytkownik nie posiada jeszcze konta, ma on możliwość kliknięcia na widoczny w prawym górnym rogu ekranu przycisk, który podmieni wyświetlany ekran na ekran rejestracji. Same ekranы niewiele się od siebie różnią pod kątem wizualnym, jedynie funkcjonalnością, oraz zawartym tekstem.



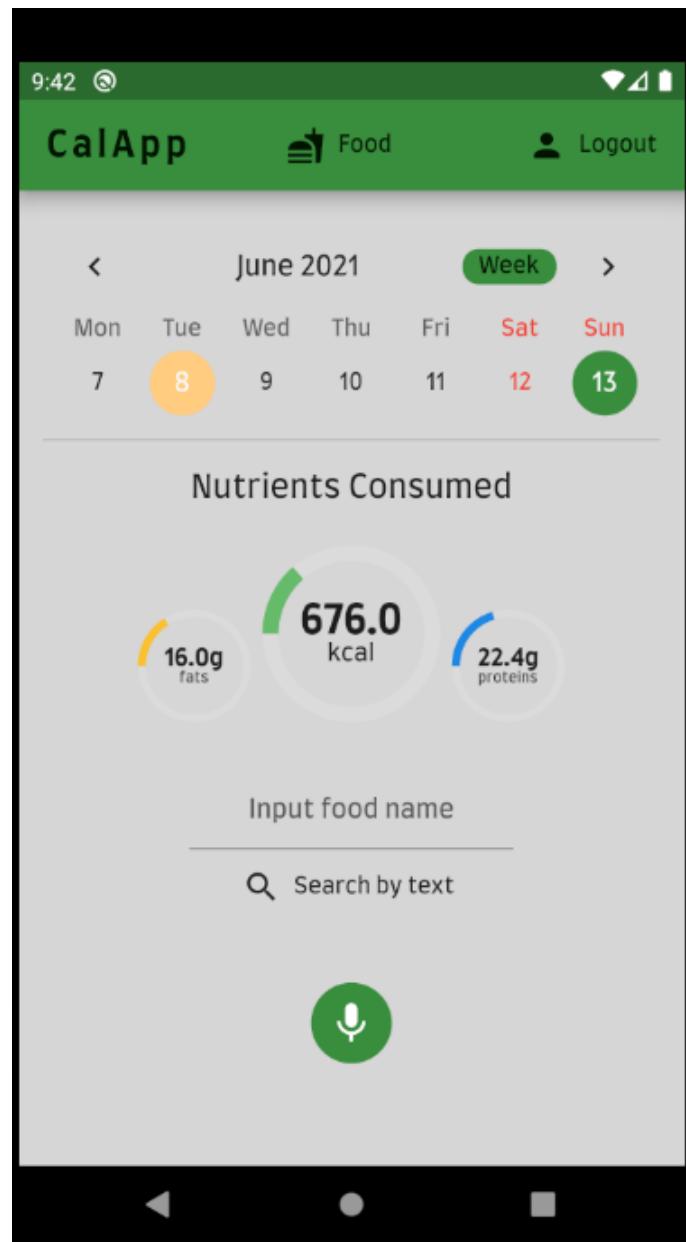
Rysunek 23: Ekran ładowania

W momencie gdy aplikacja potrzebuje załadować potrzebne jej do funkcjonowania informacje, jak przykładowo po logowaniu, lub podczas kontaktowania się z API w celu pobrania informacji o jedzeniu, użytkownikowi ukazywany jest ekran ładowania, przedstawiający animację kręcącego się, przerwanego okręgu.



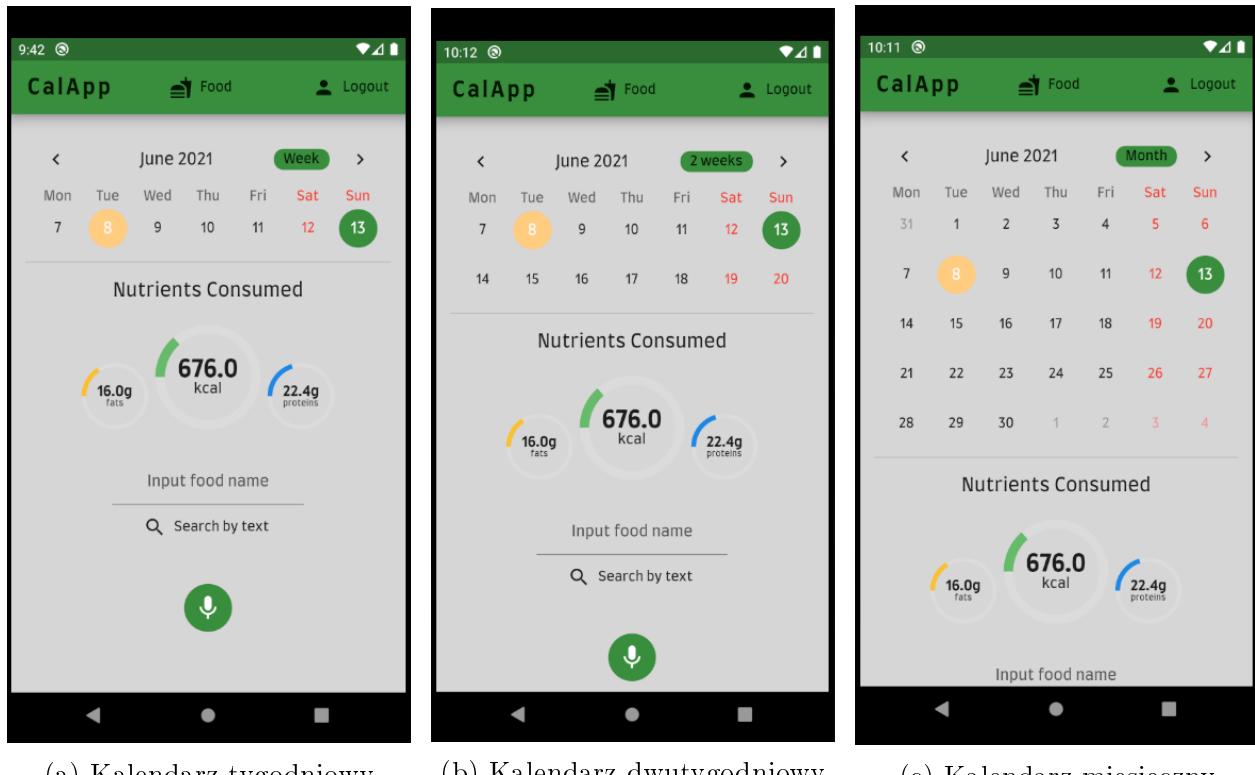
Rysunek 24: Finalny ekran główny

W finalnej wersji aplikacji, strona główna jest przedstawiona w bardziej przejrzysty sposób, ukazując najważniejsze nabycie wartości odżywcze w postaci zapełniających się "progress barów", a także dokładnej informacji tekstowej. Pole tekstowe służące wyszukiwaniu jedzenia zostało zmniejszone, a nawet może zostać ukryte po kliknięciu przycisku "Search by text" w momencie gdy pole jest puste. Aby wejść do zakładki ukazującej spożyte jedzenie, użytkownik musi kliknąć w przycisk "Food" znajdujący się na nagłówku aplikacji. Po prawo od niego możliwa jest opcja wylogowania się z aplikacji, która przeniesie użytkownika z powrotem na ekran logowania.



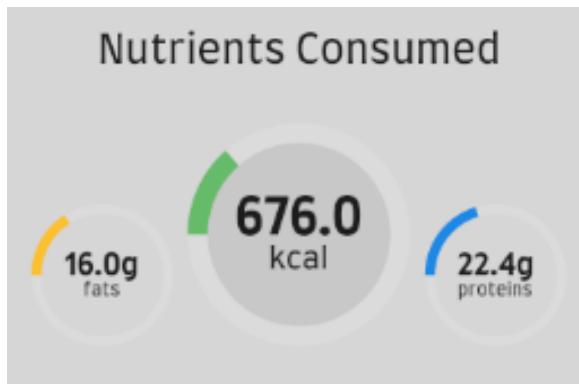
Rysunek 25: Ekran główny z wybranym innym dniem

Po kliknięciu odpowiedniego dnia na kalendarzu wygląd wykresów zmienia się w zależności od informacji jakie posiada w sobie baza danych. Domyślnie, wartości są zerowe.

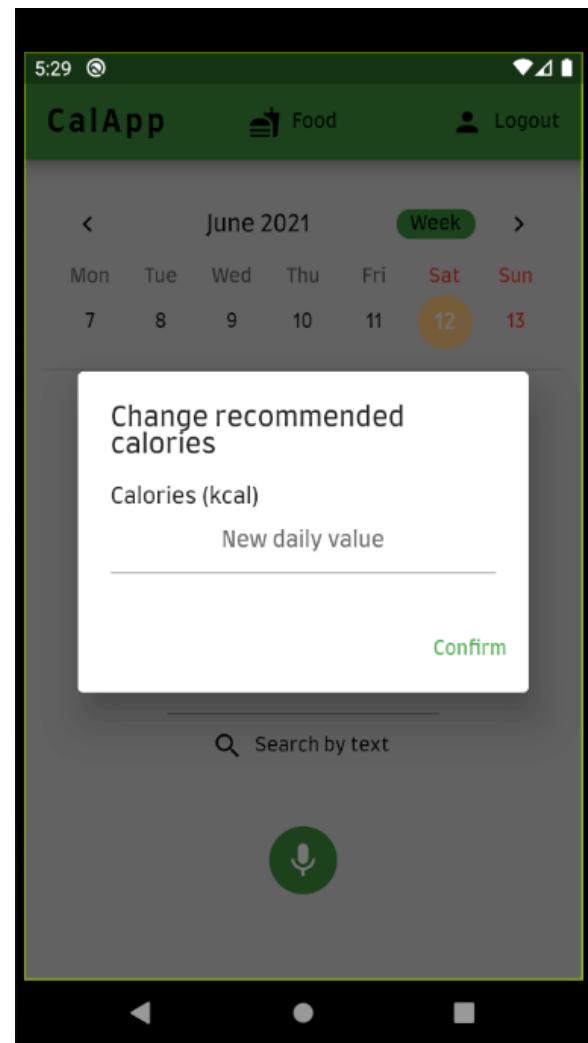


Rysunek 26: Tryby wyświetlania kalendarza

Kalendarz posiada 3 różne tryby wyświetlania zmieniane poprzez kliknięcie zielonego przycisku obok nazwy obecnego miesiąca. Wyświetlane moduły są przedstawione w formie przewijalnej listy, przez co są w stanie dynamicznie dostosować swoją pozycję na ekranie. Sprawia to, że użytkownik nadal ma dostęp do wszystkich funkcjonalności, niezależnie od wyświetlanej formy kalendarza.



(a) Przycisk do zmiany dziennego spożycia



(b) Okienko do zmiany dziennego spożycia

Rysunek 27: Zmiana rekomendowanego dziennego spożycia wartości odżywcznych

Użytkownik jest w stanie zmienić ustawienia rekomendowanego dziennego spożycia dla każdej z wartości odżywcznych, poprzez kliknięcie w odpowiednie dla tej wartości podsumowanie.

Food	Calories	Fat(g)	Protein(g)
2xBanana	178.0 kcal	0.7g	2.2g
1xCauliflo...	25.0 kcal	0.3g	1.9g
1xPizza	268.0 kcal	12.3g	10.4g
1xSalami	378.0 kcal	31.6g	21.1g

Rysunek 28: Ekran spozytego jedzenia

Ostatnim ekranem w aplikacji jest ten przedstawiający podsumowanie wszystkich zjedzonych danego dnia produktów. Informacje przedstawiane są w odpowiedniej kolejności:

- Zdjęcie produktu
- Ilość danego produktu
- Nazwa produktu
- Kalorie
- Tłuszcze

- Białka

Użytkownik ma możliwość usunięcia niechcianego wpisu poprzez kliknięcie przycisku znajdującego się na końcu przedstawiającego produkt wiersza. W przypadku, gdy API nie jest w stanie zwrócić odpowiedniego dla produktu zdjęcia, zastępowane jest ono ogólnym zdjęciem, przedstawiającym grafikę wielu produktów spożywczych.

5 Podsumowanie

Praca miała na celu stworzenie aplikacji mobilnej, mającej za zadanie wspomóc użytkownika w prowadzeniu notatek świadczących o dziennym spożyciu kalorii, ułatwiając cały proces przy pomocy wykorzystania technologii "speech to text". Autorowi udało się osiągnąć wszystkie założone wcześniej cele, jednak aplikacja nadal posiadać będzie możliwość dalszego rozwoju, poprzez dodawanie wielu brakujących, lecz opcjonalnych funkcjonalności, co zwiększy jej atrakcyjność wobec konkurencji. W przypadku chęci wydania jej na rynek, należałoby zainwestować w lepszej jakości API żywieniowe, jako że obecnie wykorzystywane w aplikacji API firmy "Edamam" jest ograniczone pod względem ilości udostępnianych informacji w wersji darmowej, jednak nadal jest znacznie lepszym rozwiązaniem niż darmowe wersje konkurencyjnych API i w obecnej formie aplikacji jego działanie jest zadowalające.

Pierwszy rozdział przedstawia problematykę, oraz definiuje cel, oraz zakres pracy. Drugi rozdział opiera się na przedstawieniu informacji potrzebnych do zrozumienia jej istoty, oraz rozwiązań w niej zastosowanych. Skupia się on nie tylko na teorii dotyczącej samych funkcjonalności aplikacji, ale także ukazuje rolę jaką pełni w dzisiejszych czasach rynek aplikacji mobilnych, oraz zawiera w sobie analizę trzech najpopularniejszych, konkurencyjnych rozwiązań. Trzeci rozdział posiada dokładne opisy technologii, wykorzystanych w procesie tworzenia aplikacji, natomiast rozdział czwarty poświęcony jest jego przebiegowi.

Rynek aplikacji mobilnych rozwija się preżnie już od wielu lat, dlatego kwestią czasu jest powstanie innego rozwiązania wykorzystującego w sobie technologię "speech to text". Zaproponowana aplikacja może jednak stanowić źródło inspiracji dla osób chcących podjąć się takiego zadania i pomóc w jego implementacji. Proces pisania pracy pozwolił pokazać potencjał frameworku Flutter, pomimo faktu, że jest to relatywnie nowa technologia i pozwala stwierdzić, iż będzie on prawdopodobnie zyskiwał na popularności w przeciągu następujących lat. Jest on prosty do opanowania, ale również oferuje szeroki wachlarz możliwości, nie tylko poprzez ciągłe rozwijanie przez firmę Google, ale także dzięki wsparciu dla społeczności, która jest w

stanie tworzyć własne "pluginy" w formie gotowych do zintegrowania z projektem widget'ów, udostępnianych na stronie "<https://pub.dev/>", co bezpośrednio przyspiesza proces tworzenia aplikacji.

6 Indeks rysunków

1	Wykres udziału w rynku poszczególnych mobilnych systemów operacyjnych na przestrzeni 10 lat	6
2	Komputery, a smartphony na przestrzeni lat 2013-2016	7
3	Komputery, a smartphony na przestrzeni lat 2016-2019	7
4	Fitatu - Ekran główny	17
5	Fitatu - Ekrany modyfikacji	18
6	Fitatu - Ekran szczegółowego podsumowania	18
7	Fitatu - Ekran zawierający banery informacyjne	19
8	Yazio - Ekran główny	20
9	Yazio - Ekran z klikniętym przyciskiem "+"	20
10	Yazio - Ekran podsumowujący	21
11	Fatsecret - Ekran główny	22
12	Fatsecret - Ekran podsumowujący	23
13	Fatsecret - Ekran raportu	23
14	Najczęściej wykorzystywane języki do tworzenia aplikacji mobilnych	29
15	Moduł komunikujący się z API żywieniowym	30
16	Pole tekstowe, oraz przycisk zatwierdzający wpisaną frazę	31
17	Metoda wyświetlająca listę zjedzonych produktów w postaci wierszy	32
18	Metoda odpowiedzialna za nasłuchiwanie	33
19	Metody wysyłające, oraz odbierające dane z bazy danych	34
20	Pierwsza wersja ekranu głównego	35
21	Ekran główny po dodaniu kalendarza	36
22	Ekrany logowania i rejestracji	37
23	Ekran ładowania	38

24	Finalny ekran główny	39
25	Ekran główny z wybranym innym dniem	40
26	Tryby wyświetlania kalendarza	41
27	Zmiana rekommendowanego dziennego spożycia wartości odżywczych	42
28	Ekran spożytego jedzenia	43

7 Bibliografia

References

- [1] *A short history of speech recognition.* URL: <https://sonix.ai/history-of-speech-recognition>.
- [2] John Callaham. *The history of Android: The evolution of the biggest mobile OS in the world.* URL: <https://www.androidauthority.com/history-android-os-name-789433/>.
- [3] Edgar F. Codd. “<https://dl.acm.org/doi/10.1145/362384.362685>”. In: (1970).
- [4] *Dart overview.* URL: <https://dart.dev/overview>.
- [5] *Fast Fourier Transformation FFT - Basics.* URL: <https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft>.
- [6] *Flutter architectural overview.* URL: <https://flutter.dev/docs/resources/architectural-overview>.
- [7] Abdallah Ismail. *How many calories should I eat a day?* URL: https://www.researchgate.net/publication/324943266_How_many_calories_should_I_eat_a_day.
- [8] Kevin Jackson. *A brief history of the smartphone.* URL: <https://sciencenode.org/feature/How%20did%20smartphones%20evolve.php>.
- [9] *Klasyfikacja wymagań.* URL: http://zasoby.open.agh.edu.pl/~10sdczerner/page/klasyfikacja_wymagan.html.
- [10] John Loeffler. *The History Behind the Invention of the First Cell Phone.* URL: <https://interestingengineering.com/the-history-behind-the-invention-of-the-first-cell-phone>.

- [11] Microsoft. *Projekt internetowego interfejsu API*. URL: <https://docs.microsoft.com/pl-pl/azure/architecture/best-practices/api-design>.
- [12] *Mobile Operating System Market Share Worldwide*. URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide>.
- [13] *Mobile VS. Desktop Internet Usage*. URL: <https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics>.
- [14] L. R. Rabiner. “An Introduction to Hidden Markov Models”. In: (). URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.957.202&rep=rep1&type=pdf>.
- [15] *The exodus from desktop to mobile: Why it matters and what to do about it*. URL: <https://www.smartinsights.com/mobile-marketing/mobile-marketing-strategy/from-desktop-mobile-why-matters/>.
- [16] *The History of Mobile Apps*. URL: <https://inventionland.com/inventing/the-history-of-mobile-apps/>.
- [17] *Types of databases*. URL: <https://www.oracle.com/database/what-is-database/#link5>.
- [18] *What are microservices?* URL: <https://microservices.io/>.