

# POLITECHNIKA KOSZALIŃSKA



## WYDZIAŁ ELEKTRONIKI I INFORMATYKI

INFORMATYKA  
TECHNOLOGIE INTERNETOWE I MOBILNE

Katarzyna Aleksandra Mural  
U-9517

Manager urody – aplikacja mobilna wspierająca  
planowanie zabiegów kosmetycznych

Beauty planner – a mobile application that supports  
planning of beauty treatments

Praca inżynierska wykonana pod kierunkiem  
dr inż. Grzegorz Górska

Koszalin 2019

*Niniejszą pracę dedykuję  
Narzeczonemu Łukaszowi i rodzicom*

## **Streszczenie**

Niniejsza praca przedstawia opis utworzonej aplikacji mobilnej w języku Swift na system iOS dla urządzeń typu iPhone 6 o nazwie „Beauty Planner”.

W części pierwszej dokumentacji zawarto krótkie porównania do istniejących już aplikacji mobilnych o zbliżonym zakresie funkcjonalnym.

Druga część opisuje architekturę aplikacji, w której szczegółowo omówiono budowę wszystkich klas z podziałem na odpowiednie grupy, a także dodatkowo przedstawiono budowę encji w bazie danych i opis połączeń pomiędzy widokami dla wszystkich storyboards.

W trzeciej części zaprezentowano funkcjonalności aplikacji wraz z wybranymi fragmentami kodu. Dla lepszego przedstawienia zaimplementowanych funkcji dołączono zrzuty ekranów z poszczególnych działów.

Ostatnia część zawiera podsumowanie, w którym uzasadniono, dlaczego stworzona aplikacja wspiera planowanie zabiegów kosmetycznych.

**Słowa kluczowe:** *Swift, iOS, iPhone, Beauty Planner, architektura aplikacji, storyboards, zabiegi kosmetyczne.*

## **Summary**

This thesis presents description of mobile application named “Beauty Planner” created in Swift programming language for iOS operation system under iPhone 6 device.

The first part of documentation described short comparison between different mobile applications with similar functionalities.

The second part describes architecture of application where explained in details structure of all classes divided to specific groups. Additionally, presented structure of entities in databases and description of connections between views for all storyboards.

The third part presents functionalities of application with examples of codes. Implemented functions has added screenshots of specific sections for better understanding them.

The last part contains summary which explains why created application supports planning of beauty treatments.

**Key words:** *Swift, iOS, iPhone, Beauty Planner, architecture of application, storyboards, beauty treatments.*

# Spis Treści

<b>1.</b>	<b>Wstęp.....</b>	<b>8</b>
<b>1.1.</b>	<b>Cele pracy .....</b>	<b>8</b>
<b>1.2.</b>	<b>Organizacja pracy .....</b>	<b>9</b>
<b>2.</b>	<b>Prezentacja wybranych pojęć dotyczących kosmetologii .....</b>	<b>11</b>
<b>3.</b>	<b>Wybrane zagadnienia dotyczące technologii mobilnych.....</b>	<b>13</b>
<b>4.</b>	<b>Prezentacja aplikacji systemów o zblżonym zakresie funkcjonalnym .....</b>	<b>14</b>
<b>5.</b>	<b>Założenia i architektura aplikacji mobilnej.....</b>	<b>17</b>
<b>5.1.</b>	<b>Diagram klas.....</b>	<b>18</b>
<b>5.2.</b>	<b>Diagram przypadków użycia.....</b>	<b>19</b>
<b>5.3.</b>	<b>Wybór narzędzi i technologii programistycznych.....</b>	<b>22</b>
<b>5.4.</b>	<b>Budowa dla storyboards .....</b>	<b>23</b>
<b>5.4.1.</b>	<b>LaunchScreen storyboard .....</b>	<b>24</b>
<b>5.4.2.</b>	<b>Main storyboard .....</b>	<b>25</b>
<b>5.4.3.</b>	<b>Planner storyboard.....</b>	<b>27</b>
<b>5.4.4.</b>	<b>Tips storyboard .....</b>	<b>30</b>
<b>5.4.5.</b>	<b>Calendar storyboard .....</b>	<b>32</b>
<b>5.4.6.</b>	<b>Measurements storyboard.....</b>	<b>33</b>
<b>5.5.</b>	<b>Budowa klas aplikacji .....</b>	<b>35</b>
<b>5.5.1.</b>	<b>Grupa Main .....</b>	<b>36</b>
<b>5.5.1.1.</b>	<b>Klasa ShortTips.....</b>	<b>36</b>
<b>5.5.1.2.</b>	<b>Klasa ViewController .....</b>	<b>37</b>
<b>5.5.1.3.</b>	<b>Klasa UITabBarController .....</b>	<b>38</b>
<b>5.5.1.4.</b>	<b>Klasa ShortTipsCoreData .....</b>	<b>39</b>
<b>5.5.1.5.</b>	<b>Klasa AppDelegate .....</b>	<b>40</b>
<b>5.5.2.</b>	<b>Grupa Planner .....</b>	<b>41</b>
<b>5.5.2.1.</b>	<b>Klasa NewAppointmentVC .....</b>	<b>41</b>
<b>5.5.2.2.</b>	<b>Klasa PlannerViewController .....</b>	<b>43</b>
<b>5.5.2.3.</b>	<b>Klasa EditAppointmentVC .....</b>	<b>44</b>
<b>5.5.2.4.</b>	<b>Klasa Appointment.....</b>	<b>46</b>
<b>5.5.2.5.</b>	<b>Klasa CoreDataOperations .....</b>	<b>47</b>

<b>5.5.3. Grupa Tips.....</b>	48
<b>5.5.3.1. Klasa Tip .....</b>	48
<b>5.5.3.2. Klasa TipsTableVC .....</b>	49
<b>5.5.3.3. Klasa TipsCoreData .....</b>	50
<b>5.5.3.4. Klasa TipsDetailsVC .....</b>	51
<b>5.5.3.5. Klasa TipsViewController.....</b>	52
<b>5.5.4. Grupa Calendar.....</b>	53
<b>5.5.4.1. Klasa DatePickerController.....</b>	53
<b>5.5.4.2. Klasa CalendarKit.....</b>	54
<b>5.5.5. Grupa Measurements.....</b>	55
<b>5.5.5.1. Klasa MeasurementsMenuVC .....</b>	55
<b>5.5.5.2. Klasa BMIVViewController .....</b>	56
<b>5.5.5.3. Klasa ChartData.....</b>	58
<b>5.5.5.4. Klasa Measurement.....</b>	59
<b>5.5.5.5. Klasa ExamplesDefaults i Env .....</b>	59
<b>5.5.5.6. Klasa MeasurementsCoreData.....</b>	60
<b>5.5.5.7. Klasa ChartSettingsVC .....</b>	61
<b>5.5.6. Grupa Contacts .....</b>	63
<b>5.5.6.1. Klasa Contact .....</b>	63
<b>5.5.6.2. Klasa ContactsListVC .....</b>	64
<b>5.5.6.3. Klasa NewContactVC.....</b>	65
<b>5.5.6.4. Klasa EditContactVC.....</b>	67
<b>5.5.6.5. Klasa ContactsCoreData .....</b>	69
<b>5.5.7. Grupa Reminder.....</b>	70
<b>5.5.7.1. Klasa ReminderVC .....</b>	70
<b>5.6. Budowa encji w CoreData .....</b>	72
<b>5.6.1. Encja AppointmentEntity .....</b>	72
<b>5.6.2. Encja ContactsEntity.....</b>	73
<b>5.6.3. Encja MeasurementsEntity .....</b>	73
<b>5.6.4. Encja ShortTipEntity .....</b>	74
<b>5.6.5. Encja TipEntity .....</b>	74
<b>5.7. Struktura zasobów typu Bundle.....</b>	75
<b>5.8. Struktura zasobów typu Assets.....</b>	76
<b>5.9. Opis zastosowanych bibliotek CocoaPods.....</b>	77
<b>5.9.1. Biblioteka SwiftCharts .....</b>	78
<b>5.9.2. Biblioteka CalendarKit .....</b>	78
<b>5.9.3. Biblioteka Toast-Swift .....</b>	79
<b>5.9.4. Biblioteka PopupDialog .....</b>	80

<b>6.</b>	<b>Prezentacja funkcjonalności i wybranych fragmentów kodu.....</b>	<b>81</b>
<b>6.1.</b>	<b>Krótkie porady.....</b>	<b>82</b>
<b>6.2.</b>	<b>Wyświetlanie wizyt na liście.....</b>	<b>84</b>
<b>6.3.</b>	<b>Dodawanie nowej wizyty do listy.....</b>	<b>85</b>
<b>6.4.</b>	<b>Edycja istniejącej wizyty.....</b>	<b>88</b>
<b>6.5.</b>	<b>Usunięcie istniejącej wizyty .....</b>	<b>90</b>
<b>6.6.</b>	<b>Wysyłanie wiadomości typu SMS.....</b>	<b>91</b>
<b>6.7.</b>	<b>Wyświetlanie listy kontaktów .....</b>	<b>94</b>
<b>6.8.</b>	<b>Dodawanie nowego kontaktu do listy.....</b>	<b>95</b>
<b>6.9.</b>	<b>Edycja istniejącego kontaktu .....</b>	<b>97</b>
<b>6.10.</b>	<b>Usunięcie istniejącego kontaktu .....</b>	<b>99</b>
<b>6.11.</b>	<b>Ustawienia powiadomień o zbliżającym się terminie.....</b>	<b>100</b>
<b>6.12.</b>	<b>Ustawienia i przegląd kalendarza .....</b>	<b>102</b>
<b>6.13.</b>	<b>Przeglądanie porad po wybranej kategorii.....</b>	<b>106</b>
<b>6.14.</b>	<b>Wyliczanie indeksu BMI .....</b>	<b>111</b>
<b>6.15.</b>	<b>Prezentacja przedziałów klasyfikacji BMI .....</b>	<b>113</b>
<b>6.16.</b>	<b>Zapisywanie wartości BMI z wybraną datą.....</b>	<b>117</b>
<b>6.17.</b>	<b>Usunięcie wszystkich zapisanych wartości BMI.....</b>	<b>118</b>
<b>6.18.</b>	<b>Tworzenie wykresu dla wybranej jednostki.....</b>	<b>119</b>
<b>6.19.</b>	<b>Okno wyboru typu Popup.....</b>	<b>122</b>
<b>6.20.</b>	<b>Powiadamienia typu Toast.....</b>	<b>123</b>
<b>7.</b>	<b>Podsumowanie pracy .....</b>	<b>124</b>
<b>8.</b>	<b>Bibliografia.....</b>	<b>126</b>
<b>9.</b>	<b>Spis rysunków .....</b>	<b>128</b>
<b>10.</b>	<b>Spis kodów źródłowych .....</b>	<b>131</b>
<b>Dodatek D1: Zawartość płyty CD .....</b>		<b>132</b>

## **1. Wstęp**

Główym celem niniejszej pracy dyplomowej było stworzenie aplikacji mobilnej, która wspierałaby planowanie zabiegów kosmetycznych. Skierowana jest zarówno dla kobiet jak i mężczyzn, ponieważ zastosowano odpowiedni dobór kolorystyczny dla szaty graficznej oraz ikon. Interfejs graficzny został opracowany z uwzględnieniem języka angielskiego, ze względu na szerszy krąg potencjalnych odbiorców. Większość funkcjonalności została przedstawiona na licznie zamieszczonych zrzutach ekranu w pracy dyplomowej.

Zdecydowano się na urządzenia mobilne, ponieważ w obecnych czasach praktycznie każda osoba posiada telefon przy sobie. Umożliwia to łatwy i szybki dostęp do aplikacji, która zawiera w sobie wszystkie niezbędne informacje odnośnie zaplanowanych wizyt użytkownika.

Mnogość różnych funkcjonalności ułatwia efektywne zarządzanie umówionymi wizytami oraz listą kontaktów do gabinetów kosmetycznych lub innych niezbędnych kontaktów. Niniejsza praca w sposób szczegółowy wyjaśnia architekturę aplikacji i zaimplementowanych funkcji.

### **1.1. Cele pracy**

Celem pracy było stworzenie aplikacji mobilnej posiadającej takie funkcjonalności jak personalny dzienniczek zabiegów kosmetycznych, który pomaga w zarządzaniu wszystkimi zaplanowanymi wizytami znajdującymi się na liście.

Funkcja wcześniejszego powiadamiania o zbliżającym się terminie informuje użytkownika o zaplanowanych spotkaniach, tylko jeśli wyraził na to zgodę i ustawił owe przypomnienie.

Poradnik kosmetyczny z możliwością wyszukiwania porad za pomocą słów kluczowych, jest niezwykle przydatną funkcjonalnością, zwłaszcza jeśli baza porad będzie bardziej rozbudowana. Użytkownik jest w stanie szybko znaleźć interesujące go wpisy dzięki zaimplementowanemu mechanizmowi wyszukiwania.

Lista kontaktów do gabinetów kosmetycznych służy do gromadzenia przez użytkownika takich i podobnych kontaktów w jednym miejscu, aby móc szybciej odnaleźć dany salon bądź gabinet, z którym ma zamiar się skontaktować.

Ostatnią z wymaganych funkcjonalności było stworzenie wysyłania wiadomości z prośbą o rejestrację terminu zabiegu. Tutaj użytkownik może zaoszczędzić trochę swojego czasu, dzięki gotowym szablonom wiadomości i synchronizacji danych. Przekazywany jest numer telefonu, pod który ma zostać wysłana wiadomość, a w samej wiadomości tekowej jest już utworzona wcześniej cała treść wiadomości wraz z konkretną datą zabiegu i godziną.

## 1.2. Organizacja pracy

W poniższych działach omówiono całą budowę aplikacji, w jakim języku i środowisku została stworzona, a także dla jakiej dziedziny. Przedstawiono diagram klas oraz cztery diagramy przypadków użycia. Utworzono także krótkie porównanie z już istniejącymi na rynku podobnymi aplikacjami tego typu.

W dziale niżej opisano niektóre pojęcia kosmetologiczne i kosmetyczne, aby wprowadzić czytelnika i użytkownika w dziedzinę jaką jest kosmetologia, aby każdy mógł zrozumieć jej podstawy. Scharakteryzowano tutaj zastosowanie poszczególnych stron aplikacji, jak i do czego je wykorzystać, co można w nich znaleźć, a także przedstawiono trochę wiedzy teoretycznej dla danej dziedziny jaką jest kosmetologia, a kosmetyka.

Następnie przedstawiono dział, który dotyczy wyboru narzędzi i technologii programistycznych. W tym miejscu po krótce wymieniono z jakich narzędzi i środowisk korzystano, aby powstała cała aplikacja. Między innymi bazowano na języku programistycznym jakim jest Swift w środowisku Xcode dla iOS.

Kolejny, czwarty dział dotyczy wybranych zagadnień dotyczących technologii mobilnych, w którym szczegółowo uzasadniono wybrany język i środowisko programistyczne.

W piątym dziale zaprezentowano aplikacje systemów o zbliżonym zakresie funkcjonalnym. Stworzono porównanie głównych działów aplikacji. Takich jak kalendarz, który porównano z aplikacją o nazwie „*Calendars*”. Następny dział dotyczący listy wizyt, przedstawia widok edycji, przyrównano z aplikacją „*Wunderlist*”. Ostatnim wpisem w tym dziale jest kalkulator BMI, w odniesieniu do aplikacji o nazwie „*Calculator Free*”.

W dziale szóstym opisano założenia aplikacji mobilnej i jej architekturę. Przedstawiono diagram klas dla zobrazowania budowy programistycznej. Zawarto również opis budowy elementów typu storyboards dla każdej grupy klas, a także szczegółowo przedstawiono ich budowę. Dodatkowo opisano budowę encji w Core Data dla każdej z grupy oraz użyte

biblioteki, którymi się posłużyono w tworzeniu aplikacji. Na koniec przedstawiono zasoby typu Bundle. W tym miejscu przechowywane są pliki tekstowe do wczytywania porad dla zakładki „*Tip*” oraz zasoby typu Assets, które odpowiedzialne są za przechowanie wszystkich załączonych zdjęć, loga i różnego rodzaju ikon do aplikacji.

W dziale siódmym zaprezentowano wszystkie funkcjonalności i wybrane fragmenty kodów. Opisano każdy widok aplikacji, wraz z załączonymi zrzutami ekranu i z uwzględnieniem najciekawszych fragmentów kodu dla tych widoków. Dzięki temu opis jest spójny i współgra ze sobą, a połączenie tych elementów lepiej obrazuje jego działanie dla danej funkcjonalności.

Na koniec zawarto podsumowanie pracy, w którym opisano, w jakim stopniu udało się zrealizować wszystkie postawione cele i zadania. Zamieszczone informacje o tym, co należy poprawić lub rozbudować w przyszłości, a także czy efekt końcowy pracy wyszedł zadowalający. Po podsumowaniu załączono bibliografię, która zawiera odnośniki do źródeł, na których się wspierano podczas tworzenia całej pracy dyplomowej, zarówno części praktycznej jak i pisemnej.

## **2. Prezentacja wybranych pojęć dotyczących kosmetologii**

W celu lepszego zrozumienia dalszego opisu funkcjonalności aplikacji, wstępnie wyjaśniono główne pojęcia dotyczące dziedziny jaką jest kosmetologia.

Pierwszym z nich jest pojęcie kosmetyczka. To osoba, która posiada podstawową wiedzę w zakresie różnych zabiegów kosmetycznych, problemów skórnych czy sposobów ich leczenia. Takie osoby zazwyczaj ukończyły specjalistyczne kursy lub szkoły (wraz z egzaminem końcowym). Kosmetyczka może wykonywać szereg zabiegów pielęgnacyjnych lub upiększających, między innymi są to różnego rodzaju zabiegi pielęgnacyjne skóry twarzy i ciała. Takie jak naturalna metoda barwienia włosów henną wraz z regulacją brwi, maseczki nawilżające lub oczyszczające skórę oraz peelingi na twarz i ciało. Zabiegi upiększające takie jak manicure, pedicure, stylistyka rzęs, depilacja, piercing, lifting rzęs, stylizację włosów itp. Przede wszystkim są to zabiegi dotyczące włosów, skóry i paznokci. Zatem kosmetyczka zajmuje się kosmetyką, która ma na celu wykorzystanie środków lub podjęcie zabiegów będących efektem pielęgnacji ciała bądź włosów. Może również wprowadzać małe zmiany mające na celu upiększenie wizerunku danej osoby. Informacje zaczerpnięto ze źródeł zewnętrznych, które można znaleźć załączone w bibliografii [11][18][19].

Kosmetologia natomiast jest znacznie szerszym pojęciem, ponieważ łączy w sobie nie tylko kosmetykę, ale i inne pokrewne temu kierunkowi dziedziny. Między innymi obejmują takie obszary jak higiena, dermatologia, ziołolecznictwo, SPA i wellness, dietetykę, aromaterapię, kosmetykę pielęgnacyjną wraz z upiększającą, a także medycynę estetyczną czy farmakologię. Zatem kosmetolog jest osobą, z bardzo obszerną wiedzą, po ukończonych studiach licencjackich, niekiedy rozszerzonych o studia magisterskie i dodatkowe szkolenia.

Osoba z wiedzą kosmetologiczną często łączy różne dziedziny, takie jak dietetyka, dermatologia. Dzięki temu jest w stanie jeszcze lepiej pomóc ludziom borykającym się z problemami skórnymi. Stara się zadbać o ich estetykę od wewnętrz i zewnętrz, przede wszystkim doradzając ludziom w sprawie doboru kosmetyków oraz różnych zabiegów kosmetycznych. Kosmetolog powinien również tłumaczyć swoim pacjentom czym jest zdrowy tryb życia i dlaczego warto, aby dbali o to, żeby na stałe zagościł w ich życiu.

Zdrowy tryb jest nieodzownym elementem leczenia wielu przypadłości skórnych, chociażby takich jak łuszczyca, trądzik, czy atopowe zapalenie skóry (powszechnie zwane AZS). Pacjent stosując się do zaleceń pielęgnacyjnych, zdrowego trybu życia, a przede wszystkim racjonalnego odżywiania jest w stanie załagodzić wszelkie objawy chorób skórnych. Przypadłości, które często dotyczą estetyki pacjenta mają wpływ na jego

samopoczucie psychiczne, a zatem tak ważne jest, aby kosmetolog posiadał szeroki zakres wiedzy i potrafił umiejętnie ją wykorzystać. Zazwyczaj we współpracy z innym fachowym specjalistą z pokrewnej dziedziny, aby jak najkorzystniej pomóc pacjentowi. Informacje zaczerpnięto ze źródeł zewnętrznych, które można znaleźć załączone w bibliografii [11][14][18][19].

Z biegiem lat, każdy z nas zauważa, że wszystko się powoli, a niekiedy diametralnie zmienia. W związku z dynamicznie rozwijającą się dziedziną kosmetologii zaczęło powstawać coraz więcej firm specjalizujących się w różnego typu usługach na rzecz poprawy urody i zdrowia. To natomiast spowodowało zwiększone zainteresowanie kosmetologią wśród ludzi, niezależnie od płci, ponieważ każdy chce dobrze wyglądać, a także dbać o siebie zarówno fizycznie, jak i pod względem estetycznym.

Po dokładniejszym wyjaśnieniu zagadnień z dziedziny kosmetologii opisano zawartość aplikacji. Jak sama nazwa wskazuje manager urody (Beauty Planner) wspiera proces zarządzania różnymi czynnikami związanymi z urodą. Takie jak planowanie wizyt, a także samodzielne dbanie o własną urodę i ciało. Dodatkowo zawarto możliwość zbierania pomiarów ciała (waga i indeks BMI).

Przykładowymi wizytami, które można nanosić do listy w aplikacji mogą być np. wizyta u manikiurystki, stylistki rzęs, makijażystki, fryzjera, a także zabieg masażu bądź spa, itp.

Poprzez samodzielne dbanie o ciało należy rozumieć szereg dostępnych porad z podziałem na kategorie. Do pierwszej podkategorii zaliczono twarz, w której umieszczono takie porady jak „regulacja brwi” bądź maseczki na twarz, które użytkownik może przygotować samodzielnie w domu. W kolejnej części umieszczono rady dotyczące włosów, w której zawarto między innymi domowe sposoby na pielęgnację włosów, a także różnego rodzaju maski do samodzielnego przyrządzenia. Trzecia podkategoria przedstawia dział odnoszący się do pielęgnacji paznokci. Tutaj użytkownik może dowiedzieć się jak wzmacnić swoje paznokcie, poprawnie je piłować, a nawet jak samemu je odżywić. Ostatnia, czwarta część dotyczy dbania o ciało. Między innymi można znaleźć tutaj informacje jak samemu wykonać cukrowy peeling, nawilżać skórę lub jak ją chronić przed słońcem.

Możliwość zbierania pomiarów należy rozumieć jako wyliczanie przez użytkownika indeksu BMI, a także zbieranie tych informacji wraz z wagą i nanoszenie otrzymanych wartości na wykres, który umożliwia kontrolowanie wagi wraz z indeksem BMI poprzez widoczne trendy spadkowe, rosnące lub stałe na przestrzeni wybranego miesiąca.

### **3. Wybrane zagadnienia dotyczące technologii mobilnych**

Wybrano platformę iOS (system operacyjny zaprojektowany przez firmę Apple) z paru względów. Jednym z powodów była chęć lepszego poznania języka Swift oraz środowiska programistycznego Xcode.

Język Swift został również zaprojektowany przez firmę Apple. Należy do grupy obiektowych języków programowania, czyli wspierających dziedziczenie, polimorfizm oraz enkapsulacje (wydzielenie fragmentów kodu do niezależnych funkcji lub klas). Został stworzony jako następca dla sprawzonego już Objective-C. Mimo stosunkowo młodego wieku (2014r.) jest szeroko wykorzystywany przez różne platformy (system operacyjny macOS, watchOS i tvOS).

Odnośnie środowiska Xcode, jest ono uniwersalne ze względu na możliwość tworzenia aplikacji w co najmniej kilku różnych językach programowania (C++, Java, Swift, C, a także Objective-C) i ponad to na różne platformy sprzętowe (komputery typu Mac, tablety iPad, telefony iPhone oraz zegarki Apple Watch). Jego ogromną zaletą jest wyjątkowa stabilność pracy oraz możliwość korzystania z edytora graficznego podczas tworzenia interfejsów dla aplikacji.

Zdecydowano się na utworzenie takiej aplikacji mobilnej, ze względu na nisze rynkową, która prawdopodobnie ma w sobie przyszłościowy potencjał dla dalszego rozwoju. Natomiast aplikacje na system Android w obecnej sytuacji górują na rynku, co powoduje trudności z wyróżnieniem się w tego typu sklepach (np. sklep Google Play).

Android jest systemem operacyjnym dla urządzeń przenośnych. Jego największą zaletą jest dostępność całego kodu źródłowego, jednakże zabezpieczenie aplikacji przed kradzieżą kodu pozostawia wiele do życzenia. Wiele z tych aplikacji nie jest tak rygorystycznie sprawdzanych przez producenta przed ich umieszczeniem w sklepie Google Play, co tworzy potencjalne ryzyko dotyczące bezpieczeństwa ich użytkowania. W przypadku sklepu App Store wymagania odnośnie prawidłowego działania i niższej konsumpcji pamięci są znacznie wyższe. Odnośnie Google Play warto jest wiedzieć, że jest to największy sklep z aplikacjami na urządzenia mobilne. Aktualnie należy on do firmy Google, znanej szczególnie w branży informatycznej.

Ze względu na powyższe niedoskonałości systemu Android, zdecydowano się na stworzenie aplikacji dla systemu iOS. Przy wykorzystaniu nowoczesnego języka Swift, a wszystko zaprojektowano przy użyciu środowiska Xcode. Informacje zaczerpnięto ze źródeł zewnętrznych, które można znaleźć załączone w bibliografii [2][3][5][11][19].

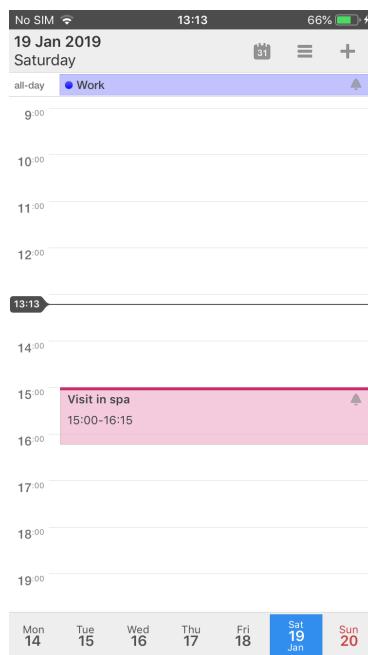
#### 4. Prezentacja aplikacji systemów o zbliżonym zakresie funkcjonalnym

W pierwszej aplikacji porównano kalendarz pomiędzy aplikacją „Calendars”, a „Beauty Planner”. W obu przypadkach można nanosić wydarzenia całodniowe, zmieniać ich kolor, dodawać je w przedziałach czasowych. Można zmieniać dni na listwie czasowej, pokazana jest aktualna godzina na stronie w obu przypadkach.

Jednakże w aplikacji „Beauty Planner” można łatwo rozpoznać aktualny dzień, który jest oznaczony kolorem czerwonym na listwie czasowej, jeśli przyciśnie się w inny dzień to będzie w kolorze czarnym, natomiast w aplikacji „Calendars” wszystkie dni na jakie się przyciska są oznaczone kolorem niebieskim.

Na rysunku po lewej stronie przedstawiono aplikację „Calendars” (*Rysunek 1*), a po prawej stronie zakładkę dotyczącą kalendarza w utworzonej aplikacji „Beauty Planner” (*Rysunek 2*). Informacje zaczerpnięto ze źródeł zewnętrznych, które można znaleźć załączone w bibliografii [20].

Rysunek 1 - Zrzut ekranu aplikacji "Calendars"



Rysunek 2 - Zrzut ekranu widoku kalendarza z aplikacji "Beauty Planner"



Źródło: Opracowanie własne.

Źródło: Opracowanie własne.

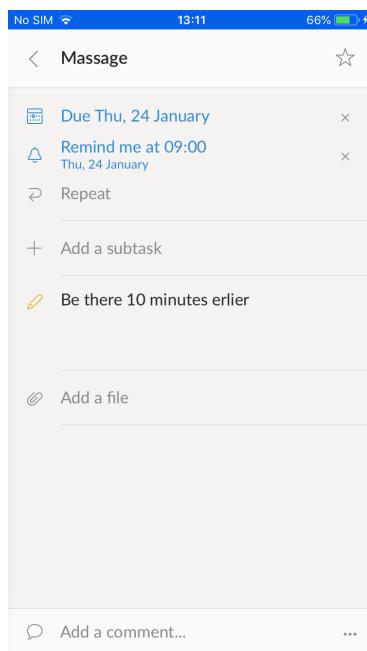
W drugiej aplikacji porównano edycję dla jednego wydarzenia, które zawarte jest w liście, pomiędzy aplikacją „Wunderlist” (*Rysunek 3*), a „Beauty Planner” (*Rysunek 4*).

W obu przypadkach można określić nazwę i datę dla danego zdarzenia, ustawić przypomnienie i dodać notatkę.

Atutem w aplikacji „Wunderlist” jest to, że można załączyć dodatkowo pliki, dodać różne podzadania, a także dodać komentarz i oznaczyć wydarzenie jako ważne (wtedy jest wyróżnione na liście). Natomiast w aplikacji „Beauty Planner” do wydarzenia można dodać numer telefonu z listy kontaktów, wprowadzić adres, a także dodatkowo ustawić czas trwania zdarzenia i przypisać mu odpowiedni kolor, który jest później nanoszony jako blok czasowy w kalendarzu.

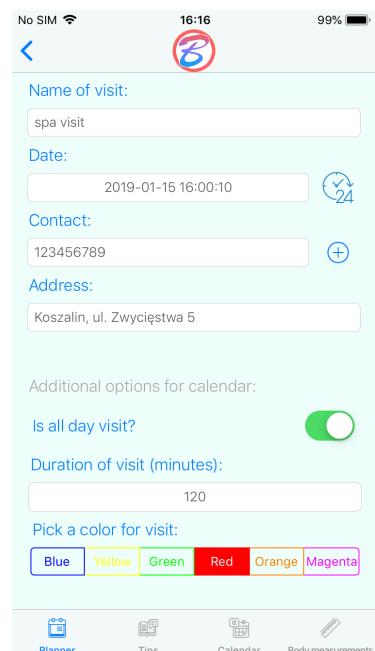
Na rysunku po lewej stronie przedstawiono aplikację „Wunderlist” (*Rysunek 3*), a po prawej stronie zakładkę dotyczącą edycji wpisu w utworzonej aplikacji „Beauty Planner” (*Rysunek 4*). Informacje zaczerpnięto ze źródeł zewnętrznych, które można znaleźć załączone w bibliografii [20].

Rysunek 3 - Zrzut ekranu aplikacji "Wunderlist"



Źródło: Opracowanie własne.

Rysunek 4 - Zrzut ekranu edycji wizyty w aplikacji "Beauty Planner"



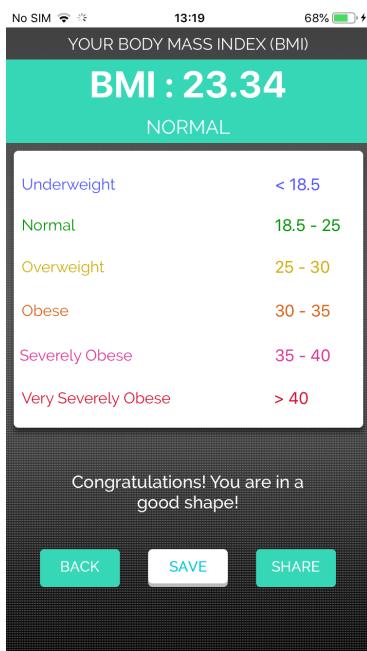
Źródło: Opracowanie własne.

W trzeciej aplikacji porównano kalkulator BMI pomiędzy aplikacją „BMI Calculator Free”, a „Beauty Planner”. W obu przypadkach wyliczany jest indeks BMI, wyświetlany rezultat wraz z zakresem, a także możliwość zapisania.

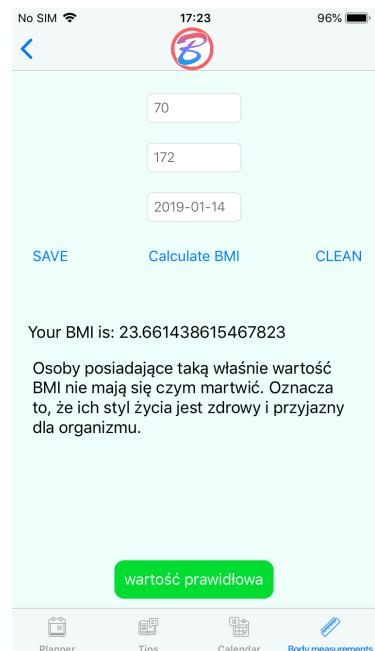
W aplikacji „Calculator Free” przy wyliczaniu indeksu dodatkowo użytkownik może podać swój wiek oraz płeć. Natomiast w aplikacji „Beauty Planner” jest wybór daty, którą użytkownik może wybrać przy zapisie danych, możliwość czyszczenia wszystkich wpisów, a także wyświetlany jest szerszy opis dla każdego zakresu BMI wraz z informacją kolorystyczną.

Na rysunku przedstawiono po lewej stronie aplikację „Calculator Free” (Rysunek 5), a po prawej stronie zakładkę dotyczącą wyliczania indeksu BMI w utworzonej aplikacji „Beauty Planner” (Rysunek 6). Informacje zaczerpnięto ze źródeł zewnętrznych, które można znaleźć załączone w bibliografii [20].

Rysunek 5 - Zrzut ekranu aplikacji "Calculator Free"



Rysunek 6 - Zrzut ekranu zakładki BMI w aplikacji "Beauty Planner"



Źródło: Opracowanie własne.

Źródło: Opracowanie własne.

## **5. Założenia i architektura aplikacji mobilnej**

Na początku tego działu przedstawiono diagram budowy klas oraz cztery różnie diagramy przypadków użycia, które obrazują przykładowe przebiegi użytkowania aplikacji.

Następnie przedstawiono rozmieszczenie wszystkich okien w aplikacji oraz ich połączenie między sobą. Jest to dobrze widoczne na załączonych rysunkach elementów typu storyboard, które pokazują zależności pomiędzy widokami.

W tym dziale można również znaleźć dokładny opis budowy stworzonych klas, zgrupowanych pod względem obszaru przynależności. Opisy rozszerzono o informacje dotyczące zaimportowanych bibliotek oraz dokładnie wyjaśniono ich zastosowanie w aplikacji.

Zamieszczone opisy poszczególnych encji w przygotowanej bazie danych zawierają spis pól, a także ich użycie.

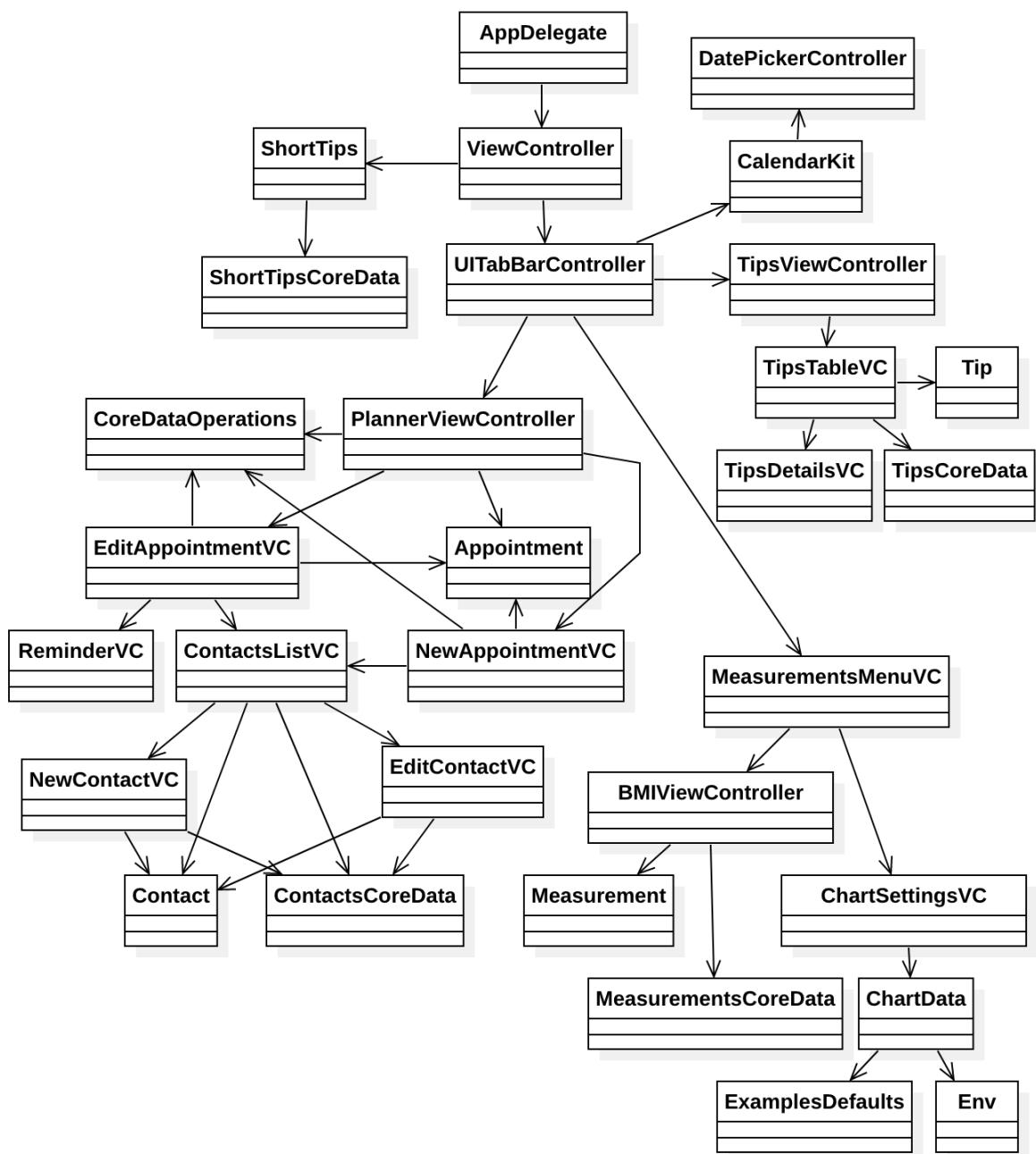
W dalszej części opisano budowę struktury typu Bundle i Assets, które przechowują niezbędne pliki (ikony, zdjęcia, porady w formie plików tekstowych) do działania aplikacji.

Ostatnia grupa reprezentuje zaimportowane biblioteki CocoaPods.

## 5.1. Diagram klas

Poniżej przedstawiono diagram statyczny klas. Ze względu na jego wielkość i szczegółowy opis wszystkich klas w dziale siódmym zdecydowano się go zaprezentować tylko z nazwami klas, bez zawarcia funkcji i zmiennych (Rysunek 7).

Rysunek 7 - Diagram klas

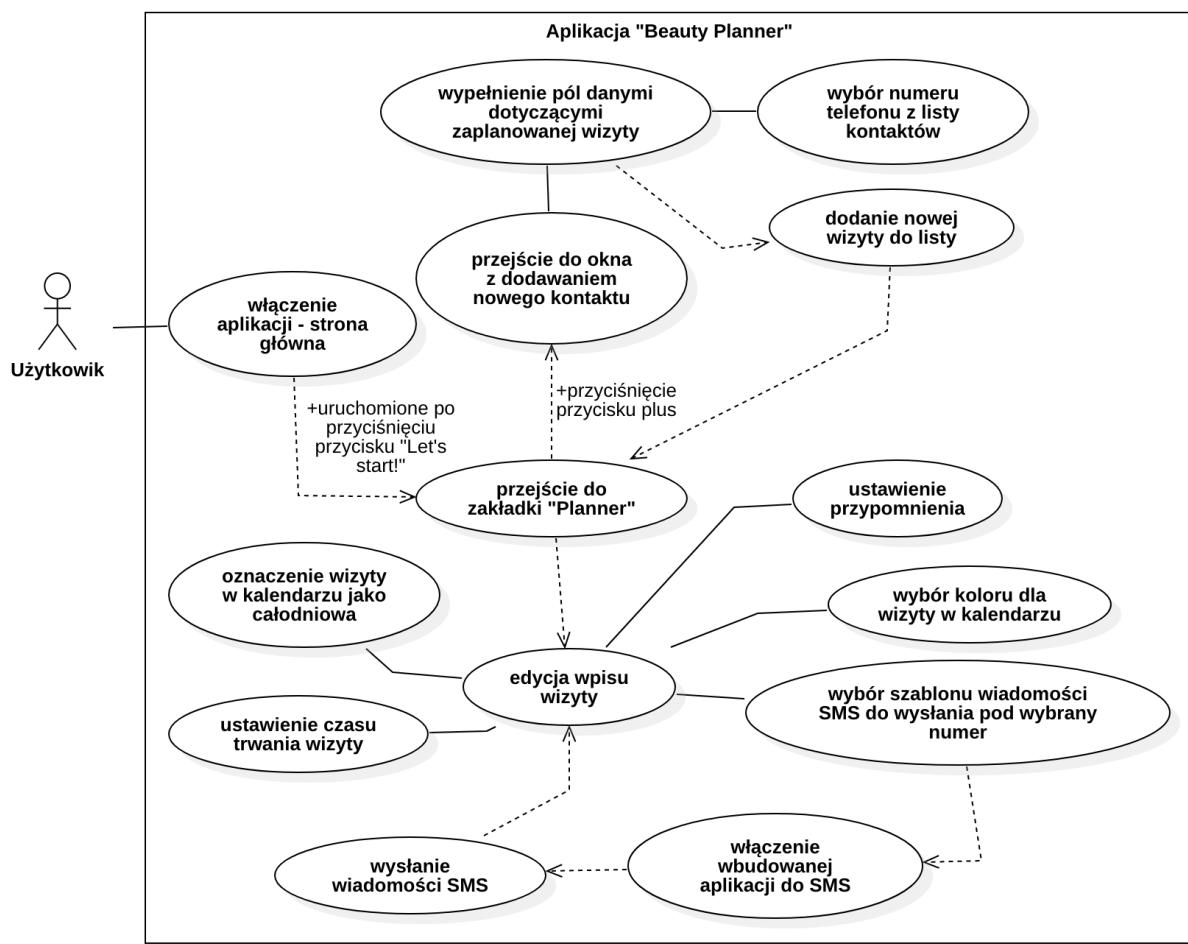


Źródło: Opracowanie własne.

## 5.2. Diagram przypadków użycia

Na poniższych diagramach przedstawiono cztery przypadki użycia. W pierwszym z nich zaprezentowano przegląd listy wizyt, które znajdują się w zakładce „Planner” w aplikacji. Następnie pokazano możliwość dodania nowej wizyty do listy jak również edycję wpisu już istniejącego. W tym miejscu użytkownik może z wybranego szablonu wiadomości wysłać szybko SMS z gotową treścią, wybraną datą jak i godziną wizyty (Rysunek 8).

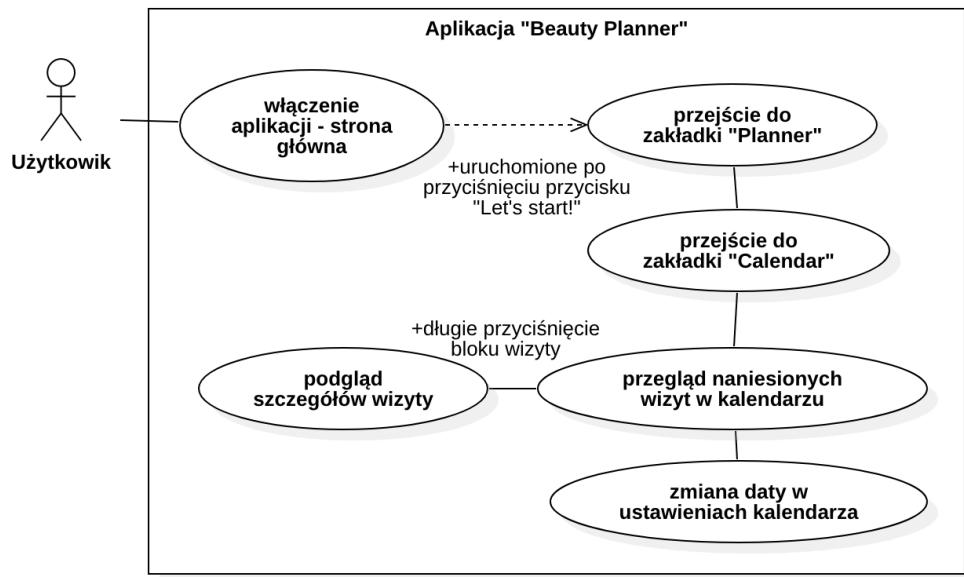
Rysunek 8 - Diagram przypadków użycia dla przeglądu listy wizyt, dodawania i ich edycji



Źródło: Opracowanie własne.

Drugi diagram przedstawia przypadek użycia dla przeglądu krótkich porad jakimi są ciekawostki wyświetlane losowo, tuż po włączeniu aplikacji. Wymaga to interakcji użytkownika z aplikacją, gdyż musi nacisnąć przycisk odpowiedzialny za wylosowanie takiej krótkiej porady (Rysunek 9).

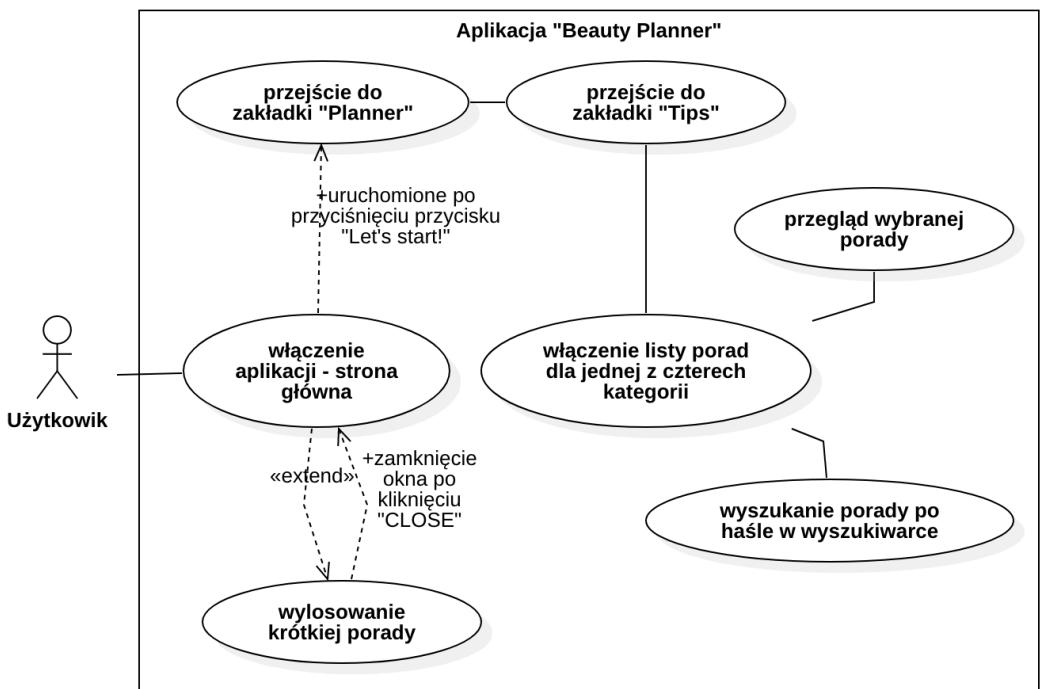
Rysunek 9 - Diagram przypadków użycia dla przeglądu krótkich porad i porad



Źródło: Opracowanie własne.

Trzeci diagram przedstawia przypadek użycia dla przeglądu kalendarza. Użytkownik może nie tylko ustawać wybrany przedział czasowy, ale także swobodnie przeglądać wszystkie naniesione wydarzenia w poszczególnych dniach (Rysunek 10).

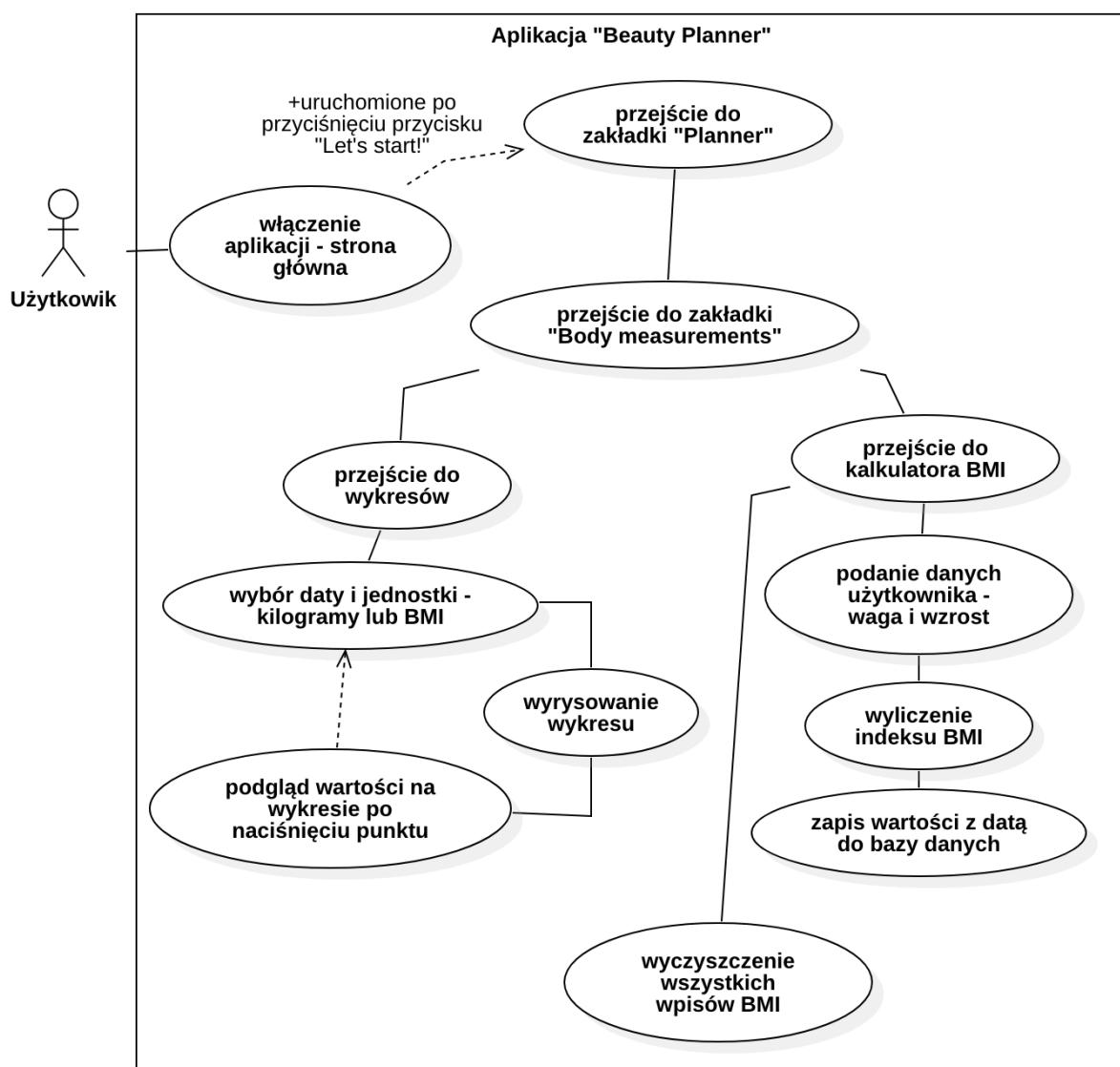
Rysunek 10 - Diagram przypadków użycia dla przeglądu kalendarza



Źródło: Opracowanie własne.

Ostatni czwarty diagram przedstawia przypadek użycia kalkulatora BMI oraz wyrysowanie wykresów. W aplikacji na zakładce „Body measurements” znajdują się dwa przyciski do wyboru. Pierwszy z nich umożliwia przejście do wyliczenia indeksu BMI wraz z zapisem tych wartości i datą do bazy danych. Można również wykasować wszystkie dotychczas zapisane wartości użytkownika. Kolejnym przyciskiem jest przejście do wyrysowania wykresów dla wartości wyliczonych indeksów BMI lub wagi użytkownika (Rysunek 11).

Rysunek 11 - Diagram przypadków użycia dla kalkulatora BMI i wykresów



Źródło: Opracowanie własne.

### **5.3. Wybór narzędzi i technologii programistycznych**

Do narzędzi, które posłużyły do wykonania aplikacji mobilnej należą:

- MacOS High Sierra – wersja 10.13.6
- Xcode – wersja 10.0
- CocoaPods – wersja 1.5.2
- Adobe Photoshop CC 2018
- Sublime Text – wersja 3.1.1



Technologie programistyczne, które wykorzystano w trakcie tworzenia aplikacji:

- Swift – wersja 4.0
- Zewnętrzne biblioteki CocoaPods



## **5.4. Budowa dla storyboards**

Dany dział prezentuje elementy typu storyboards (przedstawia połączenia wybranych widoków pomiędzy sobą). Każdy podrozdział opisuje odrębny storyboard z przedstawieniem zawartych tam widoków w postaci załączonego tam rysunku. Dodatkowo zawarto rysunki odzwierciedlające dokładną budowę struktury każdego z widoków (dokładna hierarchia użytych komponentów).

Lista obiektów typu storyboard:

- LaunchScreen storyboard – zawiera jeden widok, okno widoczne podczas wczytywania
- Main storyboard – łącznie zawiera sześć różnych widoków. Pierwszy z nich jest stroną główną, a następne reprezentują główny podział aplikacji na cztery odrębne strefy.
- Planner storyboard – łącznie zawiera osiem różnych widoków, których celem jest wyświetlanie danych odnośnie wizyt z bazy danych oraz okna odpowiedzialne za edycję tych informacji.
- Tips storyboard – łącznie zawiera cztery różne widoki, które umożliwiają wygodne i intuicyjne przeglądanie porad.
- Calendar storyboard – zawarty jest jeden widok (wyświetlający kalendarz).
- Measurements storyboard – łącznie zawarte jest pięć widoków, które umożliwiają dodawanie pomiarów oraz ich wyświetlanie na wykresie.

### 5.4.1. LaunchScreen storyboard

Widok wyświetlony podczas wczytywania aplikacji zamieszczono na poniższym rysunku (*Rysunek 12*). Można tutaj znaleźć zaprojektowane logo oraz tło, które zostało zbudowane z darmowych ikon.

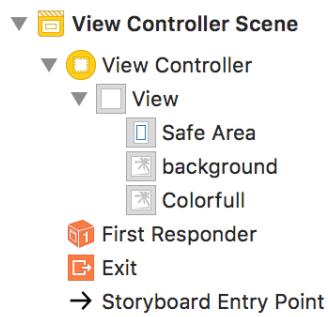
Rysunek 12 - LaunchScreen storyboard



Źródło: *Opracowanie własne.*

Poniższa struktura opisuje połączenie zastosowanych komponentów we wspomnianym widoku (*Rysunek 13*). Elementy graficzne zostały dodane do widoku typu View.

Rysunek 13 - Struktura komponentów widoku LaunchScreen

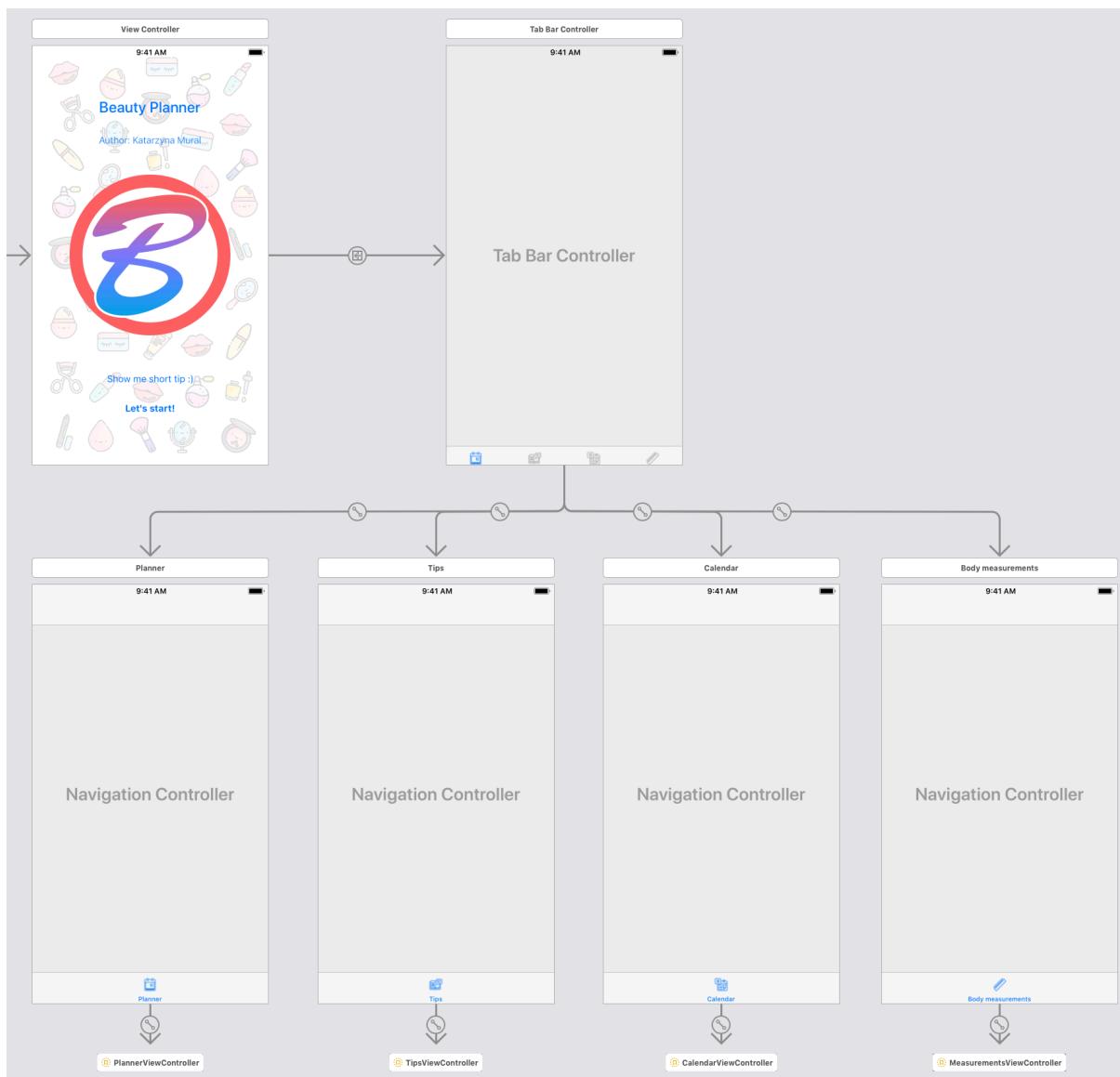


Źródło: *Opracowanie własne.*

### 5.4.2. Main storyboard

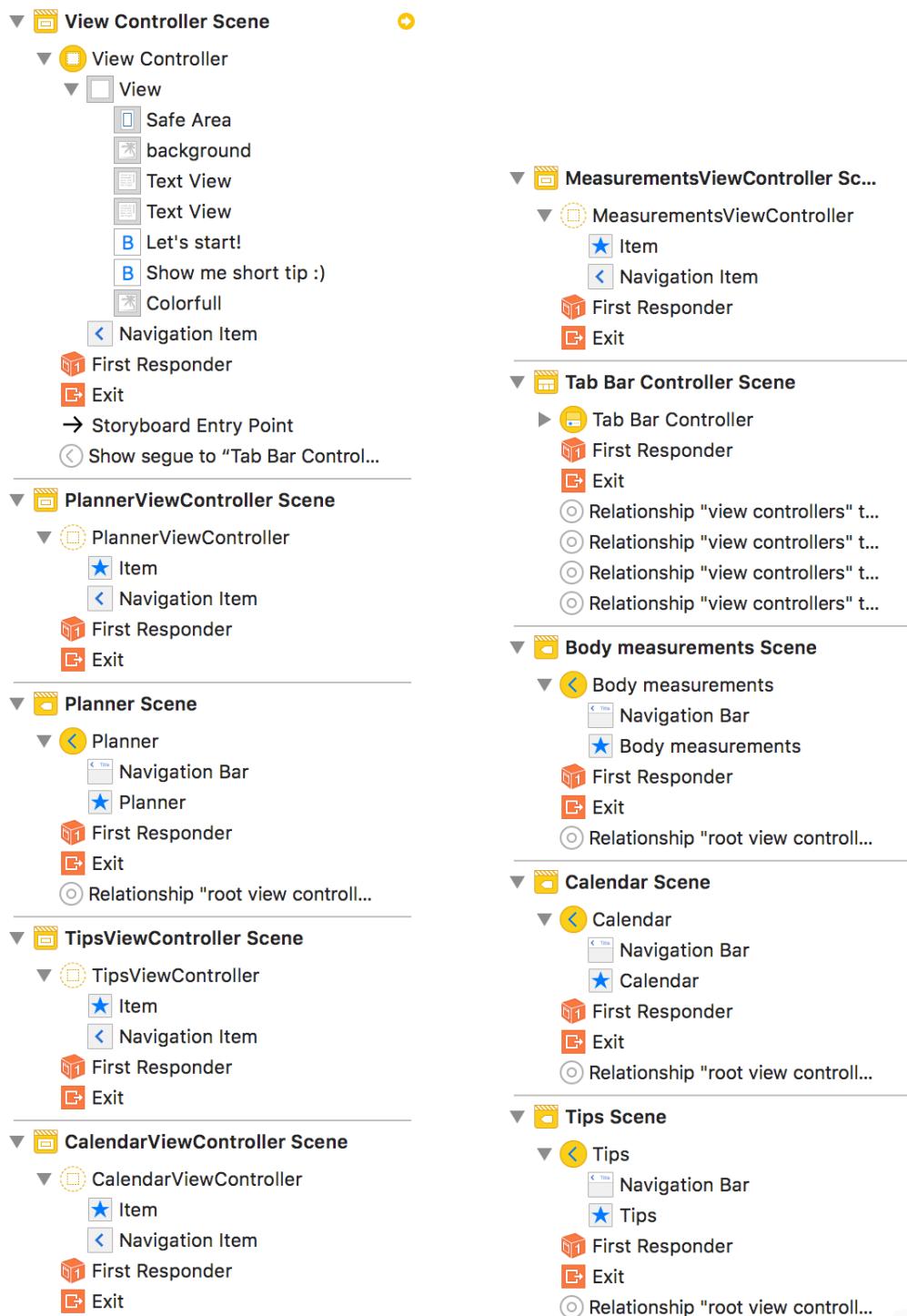
Główny widok oznaczony jest strzałką po lewej stronie widoku. W tym miejscu użytkownik zaczyna pracę z aplikacją. W tym momencie możliwe jest włączenie krótkich porad lub przejście do pierwszej z czterech głównych sekcji. Menu, które realizuje przejście do poszczególnych działów przedstawia cztery różne ikony w zależności od zawartości danego działu. Po kliknięciu na wybrany z nich użytkownik jest przenoszony do odpowiedniego storyboard. Schemat przedstawiono na poniższym rysunku (*Rysunek 14*).

Rysunek 14 - Main storyboard



Źródło: Opracowanie własne.

Rysunek 15 - Struktura widoków dla Main storyboard



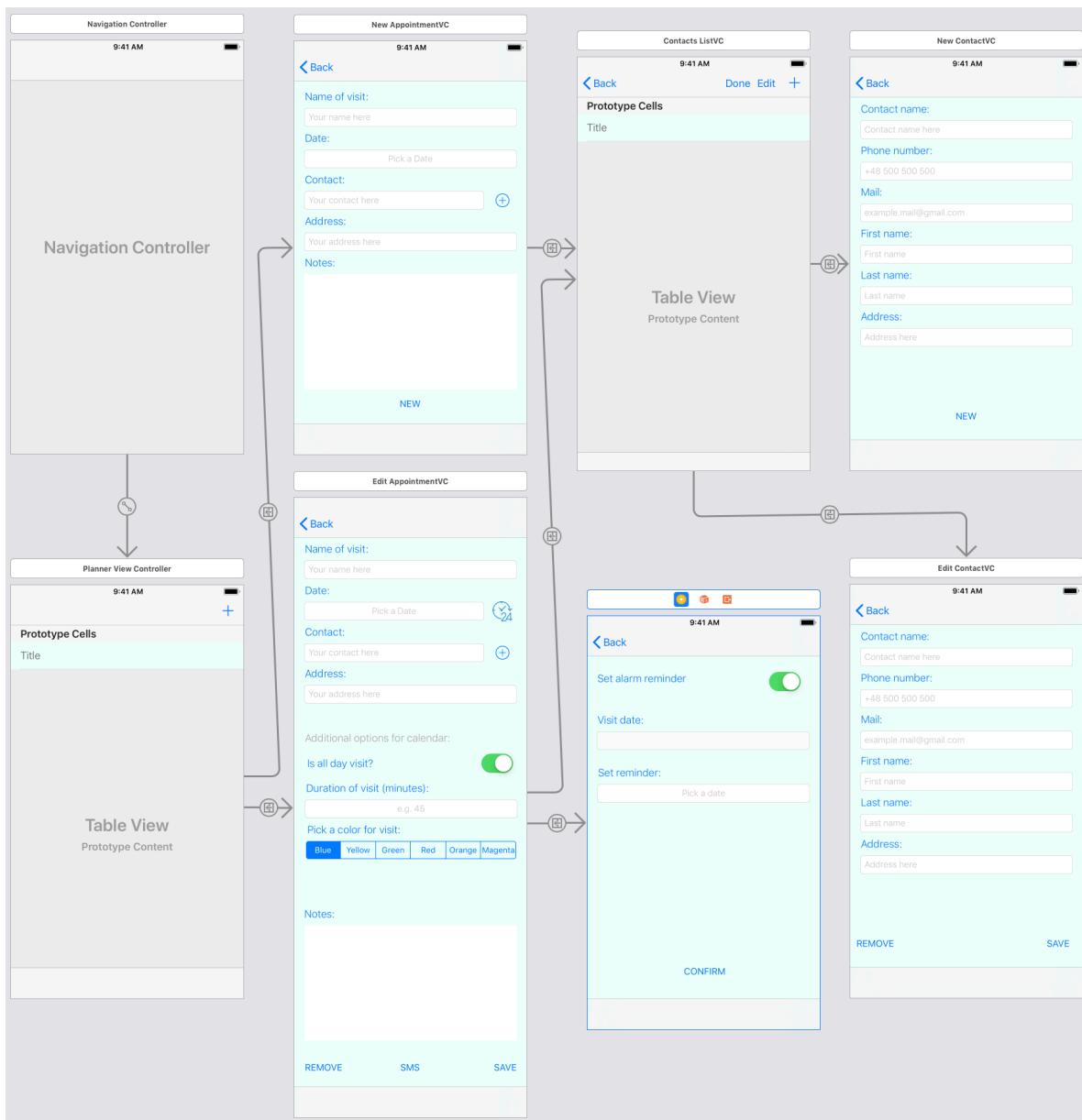
Źródło: Opracowanie własne.

Powyższa struktura opisuje połączenie zastosowanych komponentów dla wszystkich z wymienionych powyżej widoków (Rysunek 15) w storyboard o nazwie Main.

### 5.4.3. Planner storyboard

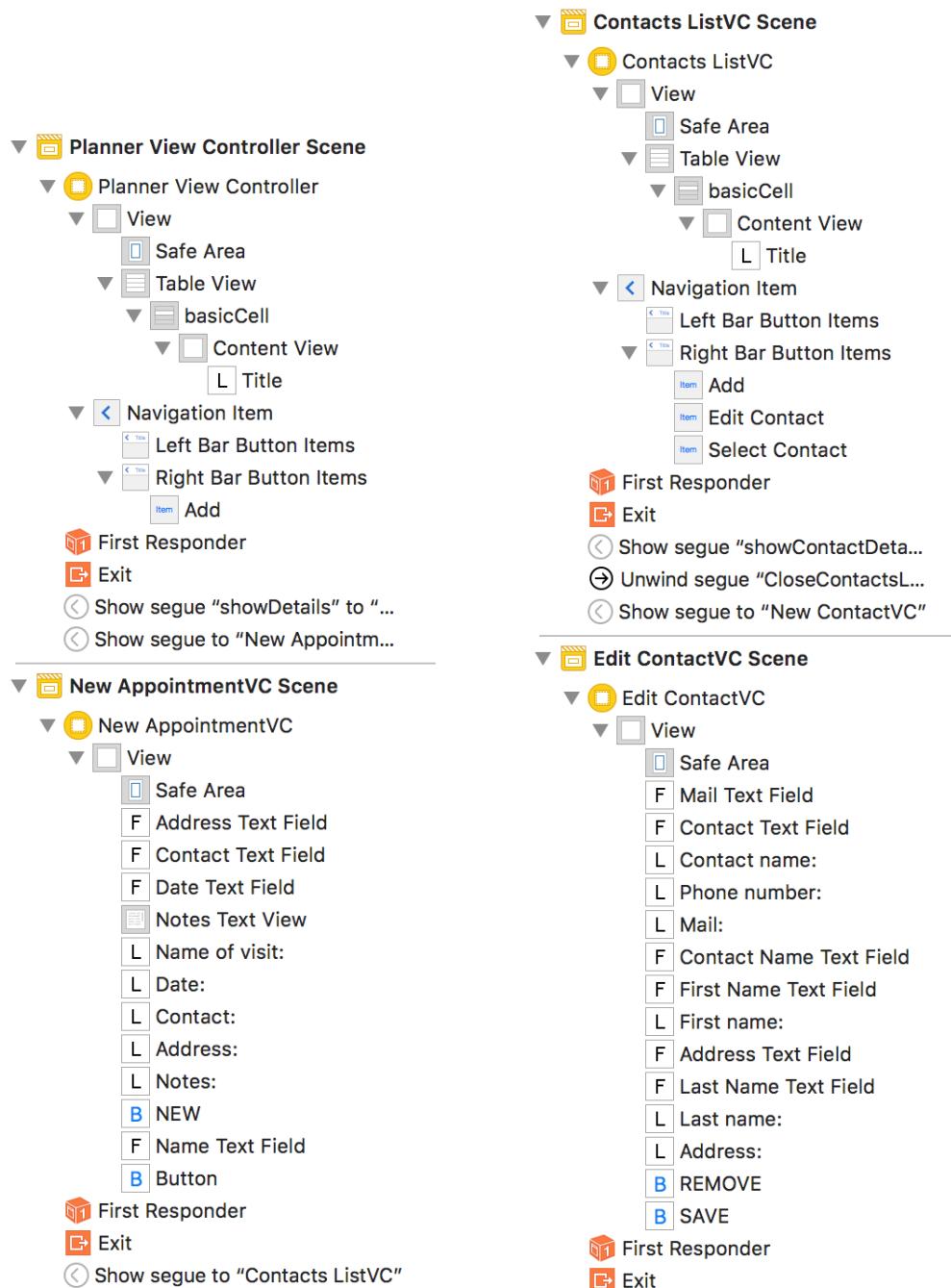
Pierwszą z czterech głównych sekcji aplikacji jest zestaw widoków odpowiedzialny za wyświetlanie zapisanych wizyt, lecz również pozwalający na ich modyfikację. Pozostałe okna pozwalają na wyświetlanie i edycję listy kontaktów oraz ustawienie przypomnienia systemowego o nadchodzącym wydarzeniu. Schemat przedstawiono na poniższym rysunku (*Rysunek 16*).

Rysunek 16 - Planner storyboard



Źródło: Opracowanie własne.

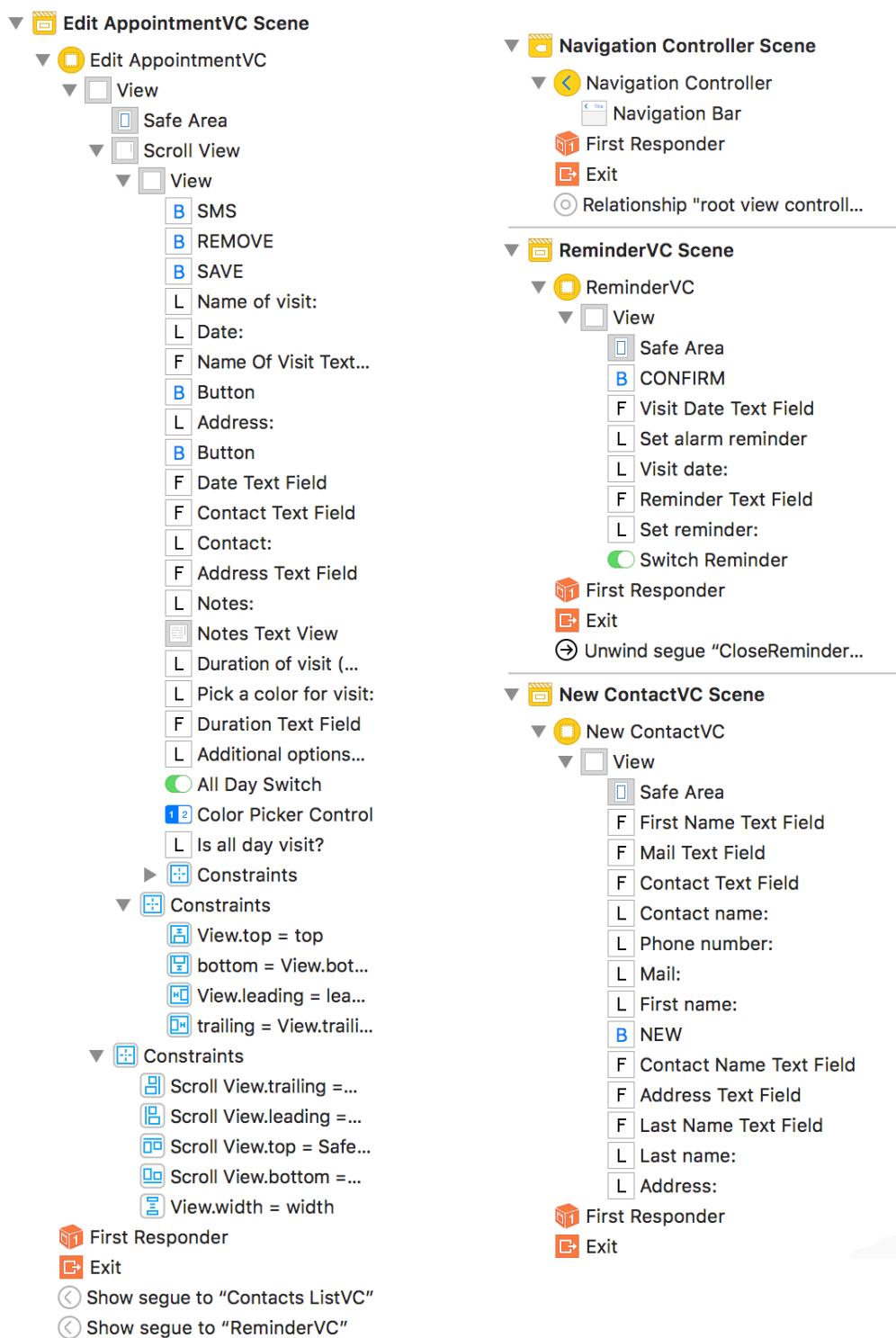
Rysunek 17 - Struktura widoków Planner storyboard cz.1



Źródło: Opracowanie własne.

Powyższa struktura opisuje połączenie zastosowanych komponentów dla wszystkich z wymienionych powyżej widoków (Rysunek 17) w storyboard o nazwie Planner.

Rysunek 18 - Struktura widoków Planner storyboard cz.2



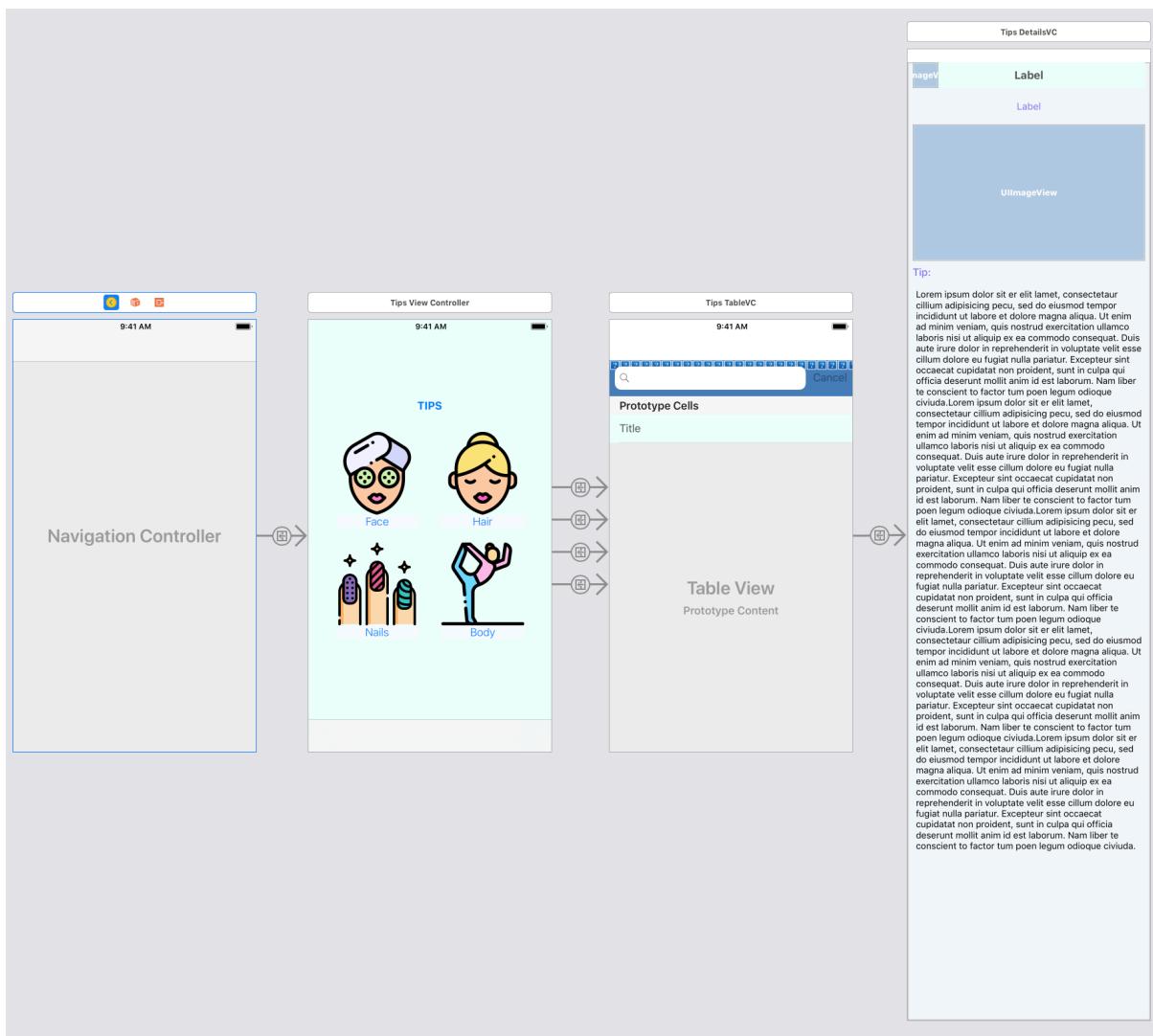
Źródło: Opracowanie własne.

Powyższa struktura opisuje połączenie zastosowanych komponentów dla wszystkich z wymienionych powyżej widoków (*Rysunek 18*) w storyboard o nazwie Planner.

#### 5.4.4. Tips storyboard

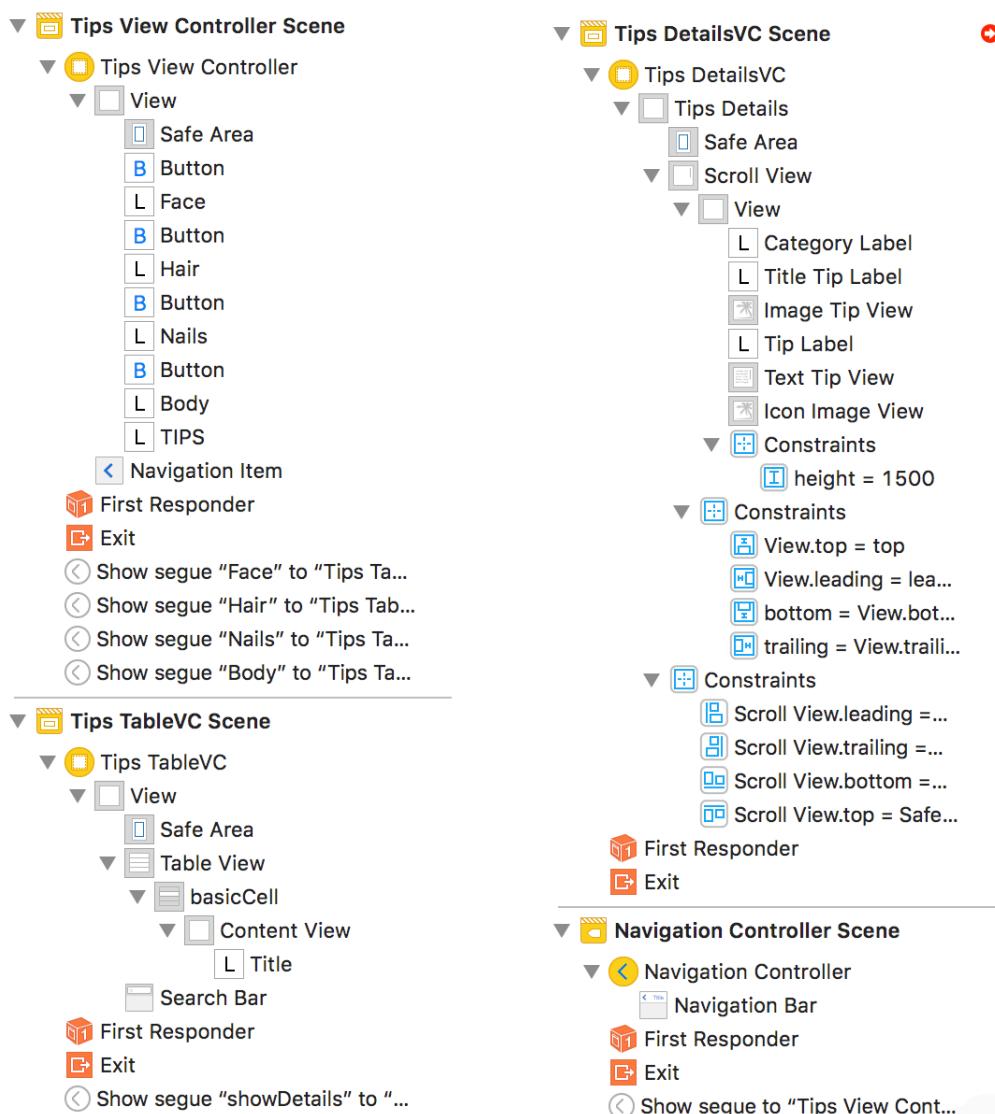
Drugą z czterech głównych sekcji aplikacji jest przeglądarka porad kosmetycznych. Po wybraniu odpowiedniego z działów następuje włączenie spersonalizowanego działu z listą przepisów. Najważniejszą częścią jest widok odpowiedzialny za przedstawienie całej porady, wraz z dedykowanym zdjęciem, nazwą i ikoną kategorii. Schemat przedstawiono na poniższym rysunku (*Rysunek 19*).

Rysunek 19 - Tips storyboard



Źródło: Opracowanie własne.

Rysunek 20 - Struktura widoków Tips storyboard



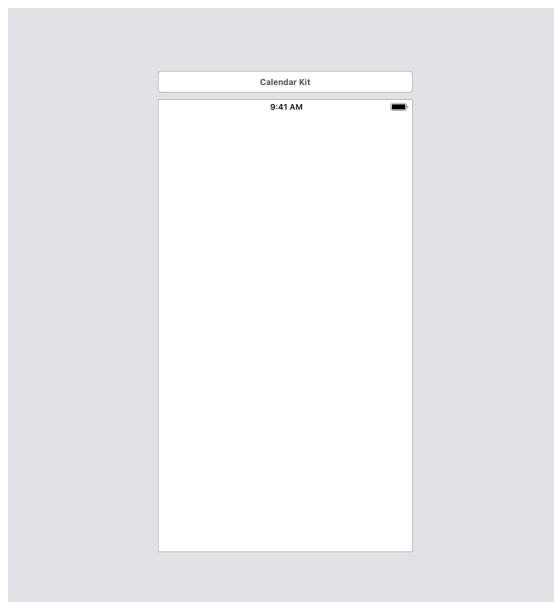
Źródło: Opracowanie własne.

Powyższa struktura opisuje połączenie zastosowanych komponentów dla wszystkich z wymienionych powyżej widoków (*Rysunek 20*) w storyboard o nazwie Tips.

### **5.4.5. Calendar storyboard**

Trzecią z czterech głównych sekcji aplikacji jest widok odpowiedzialny za wyświetlenie wykresu, wraz z naniesionymi wydarzeniami z pierwszego działu wizyt o nazwie „Planner”. Proces wyrysowania kalendarza opiera się o zainportowaną bibliotekę i wprowadzone modyfikacje. Schemat przedstawiono na poniższym rysunku (*Rysunek 21*).

Rysunek 21 - Calendar storyboard

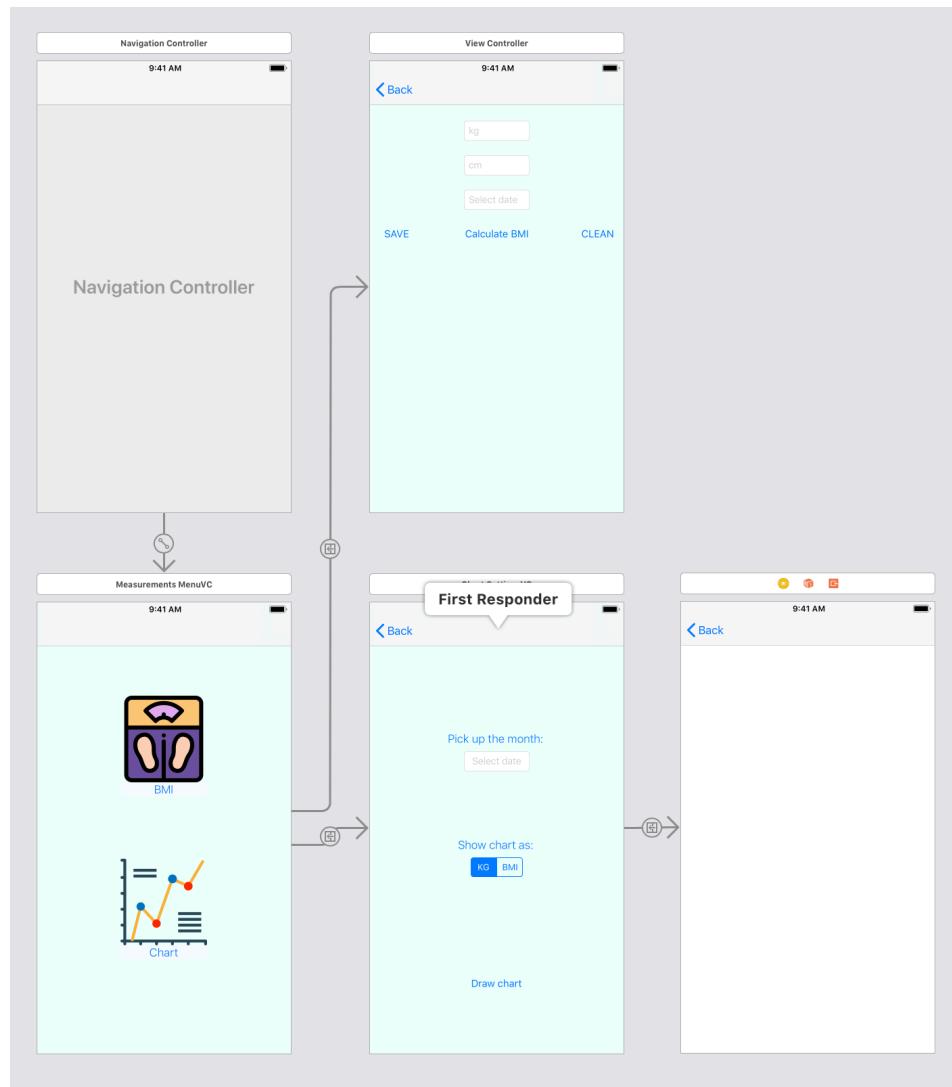


*Źródło:* Opracowanie własne.

#### 5.4.6. Measurements storyboard

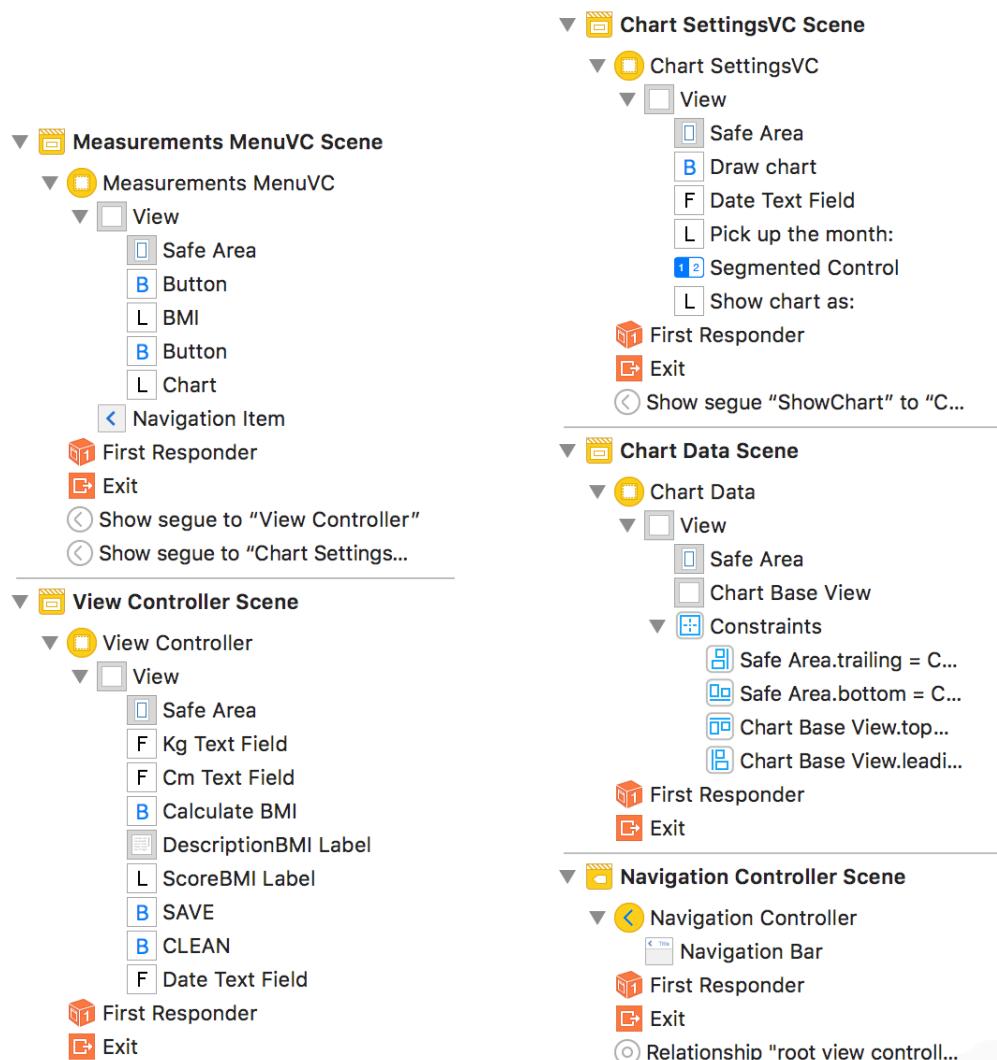
Ostatnią z czterech głównych sekcji aplikacji jest widok odpowiedzialny za wyświetlenie okienek umożliwiających wyliczenie indeksu BMI, a także wyrysowanie wykresu dla zapisanych wartości (waga w kilogramach i rezultat BMI). Schemat przedstawiono na poniższym rysunku (*Rysunek 22*).

Rysunek 22 - Measurements storyboard



Źródło: Opracowanie własne.

Rysunek 23 - Struktura widoków Measurements storyboard



*Źródło: Opracowanie własne.*

Powyższa struktura opisuje połączenie zastosowanych komponentów dla wszystkich z wymienionych powyżej widoków (*Rysunek 23*) w storyboard o nazwie Measurements.

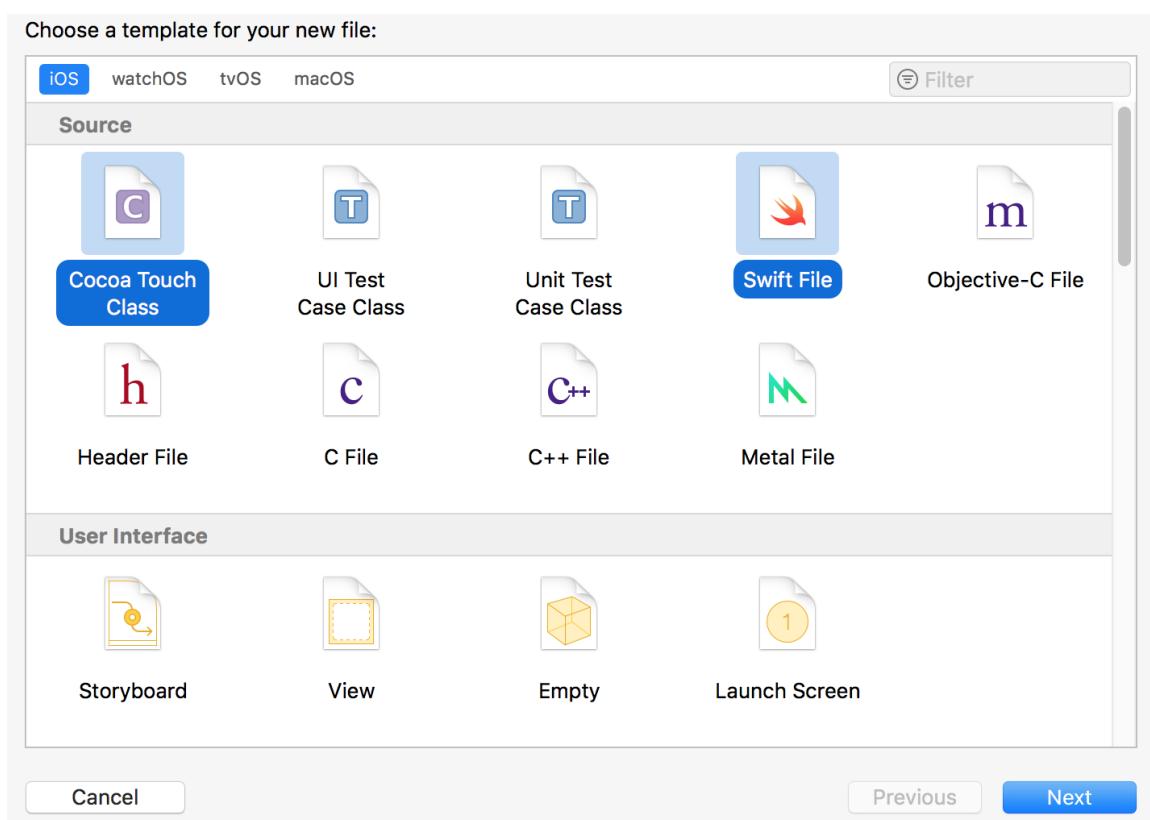
## 5.5. Budowa klas aplikacji

Klasy aplikacji zostały podzielone w grupy, które odzwierciedlają ich przynależność logiczną do konkretnych działań.

Na potrzeby realizacji aplikacji mobilnej posłużono się dwoma rodzajami plików – plik typu Swift file oraz klasa Cocoa Touch Class (*Rysunek 24*).

Szczegółowy opis każdej z zaimplementowanych klas znajduje się w dalszych podrozdziałach.

Rysunek 24 - Rodzaje użytych plików na potrzebę budowy klas

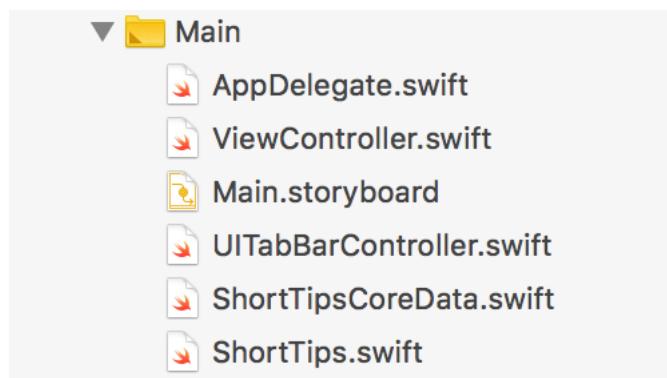


Źródło: *Opracowanie własne*.

### 5.5.1. Grupa Main

Wyodrębniono grupę Main, aby wyróżnić obiekty odnoszące się do głównego widoku całej aplikacji, czyli podziału na cztery sekcje. Zawarto również w tej grupie klasy należące do strony głównej. Schemat przedstawiono na poniższym rysunku (*Rysunek 25*).

Rysunek 25 - Grupa Main



Źródło: Opracowanie własne.

#### 5.5.1.1. Klasa ShortTips

Klasa ShortTips reprezentuje budowę encji ShortTipEntity w postaci obiektowej. Jest wykorzystywana w klasie ViewController w Main storyboard. Umożliwia przechowanie poszczególnych wpisów z bazy danych, w celu ich późniejszego wyświetlania w oknie Popup Dialog.

Lista zaimplementowanych pól:

- **id** – przechowuje wartość liczbową, która znajduje się w encji ShortTipEntity dla wybranego wpisu.
- **shortTip** – przechowuje wartość tekstową (treść krótkiej porady), która znajduje się w encji ShortTipEntity dla wybranego wpisu.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

### 5.5.1.2. Klasa ViewController

Klasa ViewController odpowiedzialna jest za wyświetlanie strony startowej i pobieranie informacji z bazy danych odnośnie krótkich porad. Umożliwia przeglądanie ciekawostek w losowej kolejności. Przedstawia logo aplikacji z danymi o autorze oraz przenosi do następnych podstron.

Wykorzystywana jest w Main storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Wczytuje krótkie porady z bazy danych i przy pierwszym uruchomieniu aplikacji prosi użytkownika o pozwolenie na wyświetlanie powiadomień.
- **viewWillAppear()** – kontroler widoku zostaje poinformowany, że jego widok ma się pojawić. Animuje przejście z LaunchScreen do strony głównej.
- **shouldAutorotate()** – blokada obracania ekranu dla pierwszej strony (dozwolony pionowy tryb przeglądania).
- **didReceiveMemoryWarning()** – standardowa funkcja wywołująca się w przypadku, gdy aplikacja zarejestruje ostrzeżenie odnośnie pamięci.
- **loadShortTipDataFromDB()** – wczytanie danych z bazy danych przy użyciu Core Data. Pobieranie wartości dla krótkich porad z encji ShortTipEntity.

Lista zainportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **UserNotifications** – przesyłanie powiadomień na urządzenie użytkownika, wygenerowane lokalnie z poziomu aplikacji.
- **PopupDialog** – darmowa i zewnętrzna biblioteka pobrała z Cocoa Pods, w celu zaimplementowania tzw. wyskakujących okienek.

Lista zastosowanych obiektów typu @IBAction:

- **showMeShortTip** – wyświetlenie losowej porady w okienku PopupDialog.

### 5.5.1.3. Klasa UITabBarController

Klasa UITabBarController odpowiedzialna jest za wyświetlanie głównego widoku aplikacji, podzielonego na cztery widoki. Każdy z nich prezentuje różne storyboards (Planner, Tips, Calendar, Measurements).

Wykorzystywana jest w Main storyboard, jako definicja działania jednego z kontrolerów typu Tab Bar Controller. Klasa ta jest rozszerzeniem innej (UINavigationController), a dziedziczy po następujących obiektach – UINavigationController.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci.
- **viewWillAppear()** – kontroler widoku zostaje poinformowany, że jego widok ma się pojawić. Animuje przejście ze strony głównej, do kolejnych widoków.
- **shouldAutorotate()** – blokada obracania ekranu dla pierwszej strony (dozwolony pionowy tryb przeglądania).
- **didReceiveMemoryWarning()** – standardowa funkcja wywołująca się w przypadku, gdy aplikacja zarejestruje ostrzeżenie odnośnie pamięci.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

#### **5.5.1.4. Klasa ShortTipsCoreData**

Klasa ShortTipsCoreData odpowiedzialna jest za modyfikację (dodawanie, usuwanie, edycja/nadpisanie) krótkich porad. Wszystkie dane wyświetlane są w oknie PopupDialog. Jest podpięta pod klasę AppDelegate w Main storyboard, jako klasa pomocnicza.

Lista zaimplementowanych funkcji:

- **addShortTip()** – dodaje krótką poradę tekstową do encji ShortTipEntity w bazie danych przy użyciu Core Data.
- **removeShortTip()** – usuwa krótką poradę tekstową z encji ShortTipEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **modifyTip()** – edytuje krótką poradę tekstową w encji ShortTipEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **findMaximumID()** – funkcja wykorzystywana przy dodawaniu krótkiej porady, pozwala na znalezienie największego indeksu ID w celu określenia następnego dla nowej ciekawostki.
- **removeAllShortTips()** – usuwa wszystkie krótkie porady tekstowe z encji ShortTipEntity w bazie danych przy użyciu Core Data.

Lista zimportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.

### **5.5.1.5. Klasa AppDelegate**

Klasa AppDelegate odpowiedzialna jest za dodawanie informacji do bazy danych odnośnie porad oraz krótkich porad (short tips). Wszystkie dane wyświetlane są w odpowiednich działach aplikacji. Wykorzystywana jest w Main storyboard. Klasa dziedziczy po następujących obiektach – UIResponder, UIApplicationDelegate.

Lista zaimplementowanych funkcji:

- **addTipEntries()** – wypełnianie encji TipEntity w bazie danych o porady dla każdej z kategorii („Face”, „Hair”, „Nails”, „Body”).
- **addShortTipsEntries()** – wypełnianie encji ShortTipEntity w bazie danych o krótkie porady dla strony startowej w aplikacji.
- **application()** – odpowiedzialne za jednokrotne załączenie wszystkich porad do bazy danych przy pierwszym uruchomieniu aplikacji.

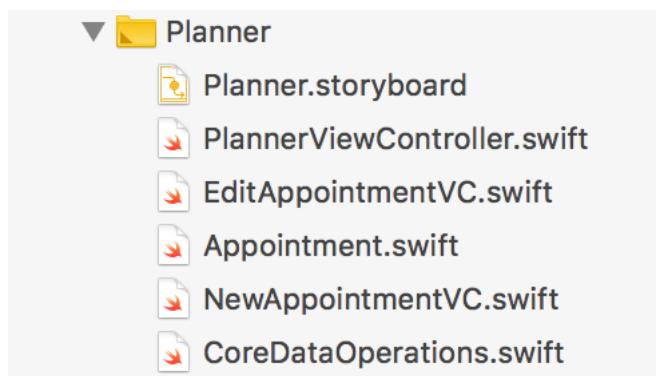
Lista zimportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.

## 5.5.2. Grupa Planner

Wyodrębniono grupę Planner, aby wyróżnić obiekty odnoszące się do wszystkich widoków przetwarzających dane odnośnie wizyt. Schemat przedstawiono na poniższym rysunku (*Rysunek 26*).

Rysunek 26 - Grupa Planner



Źródło: *Opracowanie własne.*

### 5.5.2.1. Klasa NewAppointmentVC

Klasa NewAppointmentVC odpowiedzialna jest za zapisywanie informacji pobranych z pól tekstowych do bazy danych, odnośnie zaplanowanej wizyty. Wszystkie dane wyświetlane są w polach tekstowych przy użyciu komponentów UITextField.

Wykorzystywana jest w Planner storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITextFieldDelegate, UITextViewDelegate.

Lista zastosowanych obiektów typu @IBAction:

- **myUnwindAction()** – funkcja odpowiedzialna za przesyłanie danych z zamkniętego widoku ContactsListVC do aktualnej klasy. Przenoszone są informacje odnośnie numeru telefonu z listy kontaktów.
- **addNewAppointment()** – dodanie nowej wizyty do bazy danych przy użyciu funkcji z klasy CoreDataOperations. Wartości pobierane są z pól tekstowych.

Lista zaimplementowanych funkcji:

- **navigateToPreviousView()** – powrót do poprzednio włączonego widoku.
- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Zawiera główne ustawienia tabeli oraz wyłącza powiadomienia z aplikacji po przejściu do tej strony.
- **textField()** – zabezpieczenie przez wpisaniem innej wartości niż pobrana z DatePicker.
- **viewTapped()** – ukrycie klawiatury i DatePicker w reakcji na gest dotknięcia („tap”).
- **dateChanged()** – określenie formatu pobranej daty z DatePicker i jej wpisanie do pola tekstowego.
- **textFieldShouldReturn()** – ukrycie klawiatury dla pola tekstowego w wyniku kliknięcia w przycisk „Return”.
- **didReceiveMemoryWarning()** – standardowa funkcja wywoływająca się w przypadku, gdy aplikacja zarejestruje ostrzeżenie odnośnie pamięci.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.

Lista zimportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **Toast\_Swift** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania do wyświetlania powiadomień w aplikacji.

Lista zastosowanych obiektów typu @IBOutlet:

- **nameTextField** – obiekt typu UITextField, wyświetla nazwę wizyty.
- **dateTextField** – obiekt typu UITextField, wyświetla datę wizyty.
- **contactTextField** – obiekt typu UITextField, wyświetla numer telefonu.
- **addressTextField** – obiekt typu UITextField, wyświetla adres.
- **notesTextView** – obiekt typu UITextView, miejsce na notatkę użytkownika.

### 5.5.2.2. Klasa PlannerViewController

Klasa PlannerViewController odpowiedzialna jest za pobieranie informacji z bazy danych odnośnie zaplanowanych wizyt. Wszystkie dane wyświetlane są w tabeli UITableView.

Wykorzystywana jest w Planner storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITableViewDelegate, UITableViewDataSource.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Zawiera główne ustawienia tabeli oraz wyłącza powiadomienia z aplikacji.
- **viewWillAppear()** – kontroler widoku zostaje poinformowany, że jego widok ma się pojawić. Informacje odnośnie wizyt zostają załadowane z bazy danych oraz przeładowane w tabeli.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.
- **loadPlannerDataFromDB()** – wczytanie danych z bazy danych przy użyciu Core Data. Pobieranie wartości dla wizyt z encji AppointmentEntity.
- **tableView()** – trzy funkcje odpowiedzialne za zliczanie ilości elementów w tabeli, wyświetlanie wartości oraz wywołanie akcji jeśli element został kliknięty.
- **prepare()** – w tej funkcji następuje wysyłanie elementów z tablicy appointments dla zaznaczonego wiersza do klasy EditAppointmentVC.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **UserNotifications** – przesyłanie powiadomień na urządzenie użytkownika, wygenerowane lokalnie z poziomu aplikacji.

Lista zastosowanych obiektów typu @IBOutlet:

- **tableView** – obiekt typu UITableView

### 5.5.2.3. Klasa EditAppointmentVC

Klasa EditAppointmentVC odpowiedzialna jest za edytowanie i usuwanie informacji pobranych z bazy danych odnośnie zaplanowanych wizyt oraz wysyłanie wiadomości typu SMS (przygotowano trzy szablony do wyboru). Wszystkie dane wyświetlane są w polach tekstowych przy użyciu komponentu UITextField.

Wykorzystywana jest w Planner storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITextFieldDelegate, UITextViewDelegate.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Ustawia właściwości komponentu UIDatePicker, przypisuje kolory dla przycisków odpowiedzialnych za nadanie koloru dla wizyt naniesionych w kalendarzu. Wypełnia pola tekstowe wartościami danej wizyty.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.
- **prepare()** – w tej funkcji następuje wysyłanie daty wizyty, datę przypomnienia dla zaznaczonego wiersza do klasy ReminderVC.
- **turnOffReminder()** – wyłączenie powiadomienia o konkretnej nazwie.
- **turnOnReminder()** – włączenie powiadomienia z nadaniem konkretnej nazwy.
- **navigateToPreviousView()** – powrót do poprzednio włączonego widoku.
- **textField()** – zabezpieczenie przez wpisaniem innej wartości niż pobrana z DatePicker.
- **viewTapped()** – ukrycie klawiatury i DatePicker w reakcji na gest dotknięcia („tap”).
- **dateChanged()** – określenie formatu pobranej daty z DatePicker i jej wpisanie do pola tekowego.
- **textFieldShouldReturn()** – ukrycie klawiatury dla pola tekowego w wyniku kliknięcia w przycisk „Return”.
- **didReceiveMemoryWarning()** – standardowa funkcja wywołująca się w przypadku, gdy aplikacja zarejestruje ostrzeżenie odnośnie pamięci.

Lista zainportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **Toast\_Swift** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania do wyświetlania powiadomień w aplikacji.
- **PopupDialog** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania tzw. wyskakujących okienek.
- **UserNotifications** – przesyłanie powiadomień na urządzenie użytkownika, wygenerowane lokalnie z poziomu aplikacji.

Lista zastosowanych obiektów typu @IBOutlet:

- **nameOfVisitTextField** – obiekt typu UITextField, wyświetla nazwę wizyty.
- **dateTextField** – obiekt typu UITextField, wyświetla datę wizyty.
- **contactTextField** – obiekt typu UITextField, wyświetla numer telefonu.
- **addressTextField** – obiekt typu UITextField, wyświetla adres.
- **notesTextView** – obiekt typu UITextView, miejsce na notatkę użytkownika.
- **allDaySwitch** – obiekt typu UISwitch, przełącznik dla wydarzenia całodniowego.
- **durationTextField** – obiekt typu UITextField, czas trwania wydarzenia.
- **colorPickerControl** – obiekt UISegmentedControl, kolor wydarzenia w kalendarzu.

Lista zastosowanych obiektów typu @IBAction:

- **myUnwindAction()** – funkcja odpowiedzialna za przesłanie danych z zamkniętych widoków (ContactsListVC, ReminderVC) do aktualnej klasy. Przenoszone są informacje odnośnie numeru telefonu z listy kontaktów, data przypomnienia dla konkretnego wydarzenia, a także czy powiadomienie jest włączone.
- **sendSMS()** – ta funkcja wywołuje aplikację do wysyłania SMS-ów, poprzez okno dialogowe Popup Dialog. Udostępniona jest możliwość wyboru szablonu wiadomości (zarezerwowanie wizyty, potwierdzenie wizyty, anulowanie wizyty).
- **removeAppointment()** – usunięcie wpisu z bazy danych przy użyciu funkcji z klasy CoreDataOperations.
- **modifyAppointment()** – modyfikacja wpisu w bazie danych przy użyciu funkcji z klasy CoreDataOperations. Zawiera również walidację dla pustych pól oraz sprawdza poprawność użytych znaków dla nazwy wizyty.

#### 5.5.2.4. Klasa Appointment

Klasa Appointment reprezentuje budowę encji AppointmentEntity w postaci obiektowej. Jest wykorzystywana w klasie PlannerViewController, EditAppointmentVC i NewAppointmentVC w Planner storyboard. Umożliwia przechowanie poszczególnych wpisów z bazy danych, w celu ich późniejszego wyświetlania w polach tekstowych i tabelkach.

Lista zaimplementowanych pól:

- **name** – przechowuje nazwę wizyty w postaci tekstowej.
- **date** – przechowuje datę wizyty w postaci tekstowej.
- **contact** – przechowuje numer telefonu w postaci tekstowej.
- **address** – przechowuje adres w postaci tekstowej.
- **notes** – przechowuje notatkę użytkownika w postaci tekstowej
- **id** – przechowuje wartość liczbową, która znajduje się w encji AppointmentEntity dla wybranego wpisu. Służy do unikalnej numeracji wizyt w bazie danych.
- **reminder** – przechowuje informację czy powiadomienie jest włączone czy nie.
- **reminderDate** – przechowuje datę przypomnienia o wizycie.
- **duration** – przechowuje czas trwania danego wydarzenia.
- **colorNumber** – przechowuje warianty kolorystyczne dla wydarzenia.
- **isAllDay** – przechowuje informację czy wydarzenie jest całodniowe.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

### **5.5.2.5. Klasa CoreDataOperations**

Klasa CoreDataOperations odpowiedzialna jest za modyfikację (dodawanie, usuwanie, edycja/nadpisanie) wizyt. Wszystkie dane wyświetlane są w polach tekstowych. Jest podpięta pod klasę PlannerViewController, EditAppointmentVC i NewAppointmentVC w Planner storyboard, jako klasa pomocnicza.

Lista zaimplementowanych funkcji:

- **addAppointment()** – dodaje wizytę do encji AppointmentEntity w bazie danych przy użyciu Core Data.
- **removeAppointment()** – usuwa wizytę z encji AppointmentEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **modifyAppointment()** – edytuje wizytę w encji AppointmentEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **findMaximumID()** – funkcja wykorzystywana przy dodawaniu wizyty, pozwala na znalezienie największego indeksu ID w celu określenia następnego dla nowej wizyty.
- **removeAllAppointments()** – usuwa wszystkie wizyty z encji AppointmentEntity w bazie danych przy użyciu Core Data.

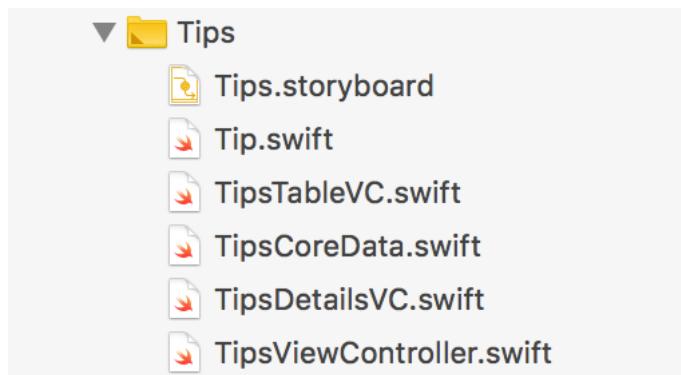
Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.

### 5.5.3. Grupa Tips

Wyodrębniono grupę Tips, aby wyróżnić obiekty odwołujące się do przeglądania i wyszukiwania porad kosmetycznych. Schemat ukazano na poniższym rysunku (*Rysunek 27*).

Rysunek 27 - Grupa Tips



Źródło: Opracowanie własne.

#### 5.5.3.1. Klasa Tip

Klasa Tip reprezentuje budowę encji TipEntity w postaci obiektowej. Jest wykorzystywana w klasie TipsTableVC i TipsDetailsVC w Tips storyboard. Umożliwia przechowywanie poszczególnych wpisów z bazy danych, w celu ich późniejszego wyświetlania w polach tekstowych.

Lista zaimplementowanych pól:

- **id** – przechowuje wartość liczbową dla wybranej porady.
- **category** – przechowuje wartość tekstową kategorii, do której należy porada.
- **pictureName** – przechowuje wartość tekstową nazwy obrazka do wczytania.
- **tip** – przechowuje wartość tekstową (treść porady), która znajduje się w encji TipEntity dla wybranej porady.
- **title** – przechowuje wartość tekstową dla tytułu porady.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

### 5.5.3.2. Klasa TipsTableVC

Klasa TipsTableVC wyświetla porady, które zostały wczytane z bazy danych. Wszystkie dane wyświetlane są w tabeli przy użyciu komponentu TableView.

Wykorzystywana jest w Tips storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITableViewDelegate, UITableViewDataSource, UISearchBarDelegate.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Zawiera cztery kategorie z podziałem kolorystycznym, poprzez które użytkownik nawiguje do kolejnych stron z poradami.
- **viewWillAppear()** – kontroler widoku zostaje poinformowany, że jego widok ma się pojawić. Informacje odnośnie porad zostają załadowane z bazy danych.
- **loadTipsDataFromDB()** – wczytanie danych z bazy danych przy użyciu Core Data. Pobieranie wartości dla porad z encji TipEntity.
- **tableView()** – cztery funkcje odpowiedzialne za zliczanie ilości elementów w tabeli, wyświetlanie wartości, wywołanie akcji po kliknięciu oraz ustawienie koloru wiersza.
- **searchBar()** – funkcja odpowiedzialna za wyszukiwanie porady po wpisanym haśle.
- **searchBarCancelButtonClicked()** – anuluje wyszukiwanie porad.
- **prepare()** – w tej funkcji następuje wysyłanie elementów z tablicy TipEntity dla zaznaczonego wiersza do klasy TipsDetailsVC.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji do paska nawigacyjnego typu Navigation Bar.

Lista zimportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.

Lista zastosowanych obiektów typu @IBOutlet:

- **tableView** – obiekt typu UITableView, wyświetlanie danych.
- **searchBar** – obiekt typu UISearchBar, służy do wyszukiwania.

### 5.5.3.3. Klasa TipsCoreData

Klasa TipsCoreData odpowiedzialna jest za modyfikację (dodawanie, usuwanie, edycja/nadpisanie) porad. Wszystkie dane wyświetlane są w polach tekstowych. Jest podpięta pod klasę TipsTableVC w Tips storyboard, jako klasa pomocnicza.

Lista zaimplementowanych funkcji:

- **addTip()** – dodaje poradę tekstową do encji TipEntity w bazie danych przy użyciu Core Data.
- **removeTip()** – usuwa poradę tekstową z encji TipEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **modifyTip()** – edytuje poradę tekstową w encji TipEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **findMaximumID()** – funkcja wykorzystywana przy dodawaniu porady, pozwala na znalezienie największego indeksu ID w celu określenia następnego dla nowej porady.
- **removeAllTips()** – usuwa wszystkie porady tekstowe z encji TipEntity w bazie danych przy użyciu Core Data.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.

#### **5.5.3.4. Klasa TipsDetailsVC**

Klasa TipsDetailsVC odpowiedzialna jest za wyświetlenie jednej porady z bazy danych. Porada wyświetlana jest przy użyciu komponentów UILabel, UIImageView, UITextView.

Wykorzystywana jest w Tips storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITextFieldDelegate, UITextViewDelegate.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Wczytanie pliku tekstowego z poradą oraz przypisanie wariantu kolorystycznego dla danej kategorii.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.

Lista zimportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

Lista zastosowanych obiektów typu @IBOutlet:

- **categoryLabel** – obiekt typu UILabel, wyświetla kategorię porady.
- **titleTipLabel** – obiekt typu UILabel, wyświetla tytuł porady.
- **imageTipView** – obiekt typu UIImageView, wyświetla obrazek do porady,
- **textTipView** – obiekt typu UITextView, wyświetla treść porady.
- **iconImageView** – obiekt typu UIImageView, wyświetla ikonę danej kategorii.
- **tipLabel** – obiekt typu UILabel, wyświetla napis „porada” („tip”).

### **5.5.3.5. Klasa TipsViewController**

Klasa TipsViewController odpowiedzialna jest za wybór kategorii i jej przesłanie do następnego widoku.

Wykorzystywana jest w Tips storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci.
- **prepare()** – rozpoznaje które przejście (segue) zostało kliknięte i wysyła nazwę kategorii do kontrolera typu TipsTableVC.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.

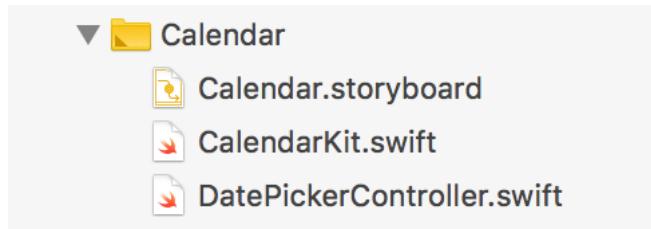
Lista zainportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

#### **5.5.4. Grupa Calendar**

Wyodrębniono grupę Calendar, aby wyróżnić obiekty reprezentujące kod odnoszący się do obsługi i wyświetlania kalendarza. Schemat przedstawiono na poniższym rysunku (*Rysunek 28*).

Rysunek 28 - Grupa Calendar



*Źródło: Opracowanie własne.*

##### **5.5.4.1. Klasa DatePickerController**

Klasa DatePickerController odpowiedzialna jest za włączenie DatePicker na całą stronę w celu lepszego nawigowania do wybranej daty. Zarządza rozmieszczeniem przycisków wyboru „done” i anulowania „cancel”.

Wykorzystywana jest w Calendar storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController.

Lista zaimplementowanych funkcji:

- **loadView()** – załadowanie DatePicker na całym widoku okna.
- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Odpowiedzialny za rozmieszczenie przycisków dla DatePicker.
- **doneButtonDidTap()** – przycisk akceptujący wybraną datę z DatePicker.
- **cancelButtonDidTap()** – przycisk anulujący wybieranie daty z DatePicker.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji do paska nawigacyjnego typu Navigation Bar.

Lista zainportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

#### 5.5.4.2. Klasa CalendarKit

Klasa CalendarKit odpowiedzialna jest za pobieranie informacji z bazy danych odnośnie zaplanowanych wizyt, które są nanoszone w blokach czasowych na kalendarz. Wszystkie zaplanowane wizyty wyświetlane są w kalendarzu z podziałem kolorystycznym.

Wykorzystywana jest w Calendar storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – DayViewController, DatePickerControllerDelegate.

Lista zaimplementowanych funkcji:

- **loadCalendarDataFromDB()** – wczytanie danych odnośnie wizyt z bazy danych.
- **viewWillAppear()** – kontroler widoku zostaje poinformowany, że jego widok ma się pojawić. Informacje odnośnie wizyt zostają załadowane z bazy danych.
- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Zawiera ustawienia kalendarza do zmiany daty oraz przeladowuje dane w kalendarzu.
- **presentDatePicker()** – wyświetlanie DatePicker oraz ustawianie aktualnej daty.
- **datePicker()** – odpowiedzialne za pobieranie daty, gdy zostaje zaznaczone.
- **eventsForDate()** – pobiera wszystkie wizyty i dla każdej z nich ustawia wszystkie parametry (długość trwania, kolor).
- **dayViewDidLongPressEventView()** – po naciśnięciu i długim przytrzymaniu na wybranym wydarzeniu w kalendarzu ukazuje się okno dialogowe PopupDialog, ze szczegółowym opisem i akcjami do wyboru (pokaż więcej, anuluj).
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego.

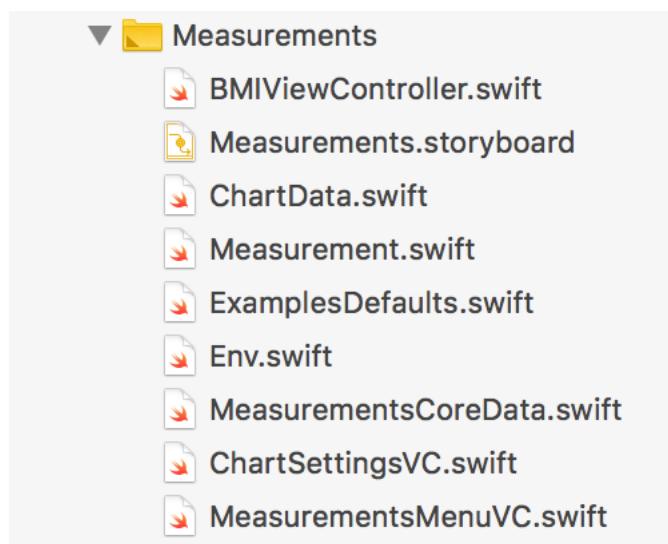
Lista zimportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CalendarKit** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, służy do zaimplementowania kalendarza.
- **DateToolsSwift** – biblioteka pobrana z Cocoa Pods, wymagana do użycia kalendarza.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **PopupDialog** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania tzw. wyskakujących okienek.

### **5.5.5. Grupa Measurements**

Wyodrębniono grupę Measurements, aby wyróżnić obiekty realizujące zbieranie i wyświetlanie pomiarów BMI i wagi. Schemat ukazano na poniższym rysunku (*Rysunek 29*).

Rysunek 29 - Grupa Measurements



*Źródło:* Opracowanie własne.

#### **5.5.5.1. Klasa MeasurementsMenuVC**

Klasa MeasurementsMenuVC jest połączona do widoku, gdzie następuje wybór działu dla obliczania BMI lub przejście do strony z wykresami.

Wykorzystywana jest w Measurements storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.

Lista zainportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

### 5.5.5.2. Klasa BMIVViewController

Klasa BMIVViewController odpowiedzialna jest za wyliczanie indeksu BMI użytkownika z możliwością wyboru daty z DatePicker i zapisania uzyskanego wyniku do bazy danych. W celu usunięcia wszystkich zapisanych wpisów użytkownika wystarczy przycisnąć przycisk „clean”.

Wykorzystywana jest w Measurements storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITextFieldDelegate.

Lista zaimplementowanych funkcji:

- **hideKeyboardWhenTappedAround()** – jeśli klawiatura jest aktywna i kliknięto gdziekolwiek poza nią, ukryje się.
- **dismissKeyboard()** – pominięcie klawiatury, możliwość jej zamknięcia przy użyciu jej samej.
- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Jest odpowiedzialny za ustawienie dzisiejszej daty w polu tekstowym poprzez DatePicker i podłączenie DatePicker do pola dateTextField.
- **getCurrentDate()** – pobieranie aktualnej daty o formacie rok, miesiąc i dzień.
- **textFieldShouldReturn()** – kończy edycję poprzez przyciśnięcie przycisku return.
- **didReceiveMemoryWarning()** – standardowa funkcja wywołująca się w przypadku, gdy aplikacja zarejestruje ostrzeżenie odnośnie pamięci.
- **textField()** – zabezpieczenie DatePicker przed wklejaniem i dodawaniem innej treści i znaków niż te podane przez DatePicker.
- **viewTapped()** – ukrycie klawiatury w reakcji na gest dotknięcia („tap”).
- **dateChanged()** – określenie formatu pobranej daty z DatePicker i jej wpisanie do pola tekstowego.
- **savingBMIDataToDB()** – modyfikuje dane (waga w kilogramach, wynik BMI, data pomiaru) jeśli istnieje jakikolwiek wpis dla danej daty. Natomiast jeśli nie ma wpisu to zapisuje dane dla wybranej daty do bazy danych.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.

Lista zainportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **Toast\_Swift** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania do wyświetlania powiadomień w aplikacji.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **PopupDialog** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania tzw. wyskakujących okienek.

Lista zastosowanych obiektów typu @IBOutlet:

- **kgTextField** – obiekt typu UITextField, pole tekstowe do wpisania wagi w kilogramach.
- **cmTextField** – obiekt typu UITextField, pole tekstowe do wpisania wzrostu w centymetrach.
- **scoreBMILabel** – obiekt typu UILabel, wyświetla wyliczony indeksu BMI.
- **descriptionBMILabel** – obiekt typu UITextView, wyświetla dokładny opis dla wyliczonego indeksu BMI (wygłodzenie, wychudzenie, niedowaga, wartość prawidłowa, nadwaga, I stopień otyłości, II stopień otyłości, otyłość skrajna).
- **dateTextField** – obiekt typu UITextField, wyświetla date w DatePicker i umożliwia jej zmianę.

Lista zastosowanych obiektów typu @IBAction:

- **saveDataBMI()** – wyświetla okno dialogowe PopupDialog z decyzją o zapisaniu pomiarów. Jeśli użytkownik wybierze zapis owych danych to następuje wywołanie funkcji savingBMIDataToDB().
- **cleanDataBMI()** – wyświetla okno dialogowe PopupDialog z decyzją o wyczyszczeniu wszystkich wcześniej zapisanych pomiarów użytkownika z bazy danych.
- **calculateBMI()** – oblicza wartości indeksu BMI. Wyświetla opis dla każdego wyliczenia wraz z krótką informacją kolorystyczną typu toast.

### 5.5.5.3. Klasa ChartData

Klasa ChartData odbiera dane z klasy ChartSettingsVC odnośnie wagi w kilogramach, wyniku BMI i daty dla tych rezultatów. Odpowiedzialna jest za rysowanie wykresu dla dwóch wariantów (kilogramy lub BMI). Wszystkie dane nanoszone są na wykres w postaci punktów połączonych liniami, co umożliwia śledzenie aktualnego trendu (malejący lub rosnący).

Wykorzystywana jest w Measurements storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Konfiguruje cały wykres (opisy osi, jakie odstępy podziałek, zakres na osiach, rodzaj wykresu).
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.

Lista zimportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **SwiftCharts** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania wykresu.

Lista zastosowanych obiektów typu @IBOutlet:

- **chartBaseView** – obiekt typu UIView, służy do wyświetlania wykresu.

#### **5.5.5.4. Klasa Measurement**

Klasa Measurement reprezentuje budowę encji MeasurementsEntity w postaci obiektowej. Jest wykorzystywana w klasie BMIVViewController, MeasurementsCoreData i ChartSettingsVC w Measurements storyboard. Umożliwia przechowanie poszczególnych wpisów z bazy danych, w celu ich późniejszego wyświetlania na wykresie.

Lista zaimplementowanych pól:

- **id** – przechowuje wartość liczbową, która znajduje się w encji MeasurementsEntity dla wybranego pomiaru.
- **resultBMI** – przechowuje wartość liczbową wyniku BMI.
- **weight** – przechowuje wartość liczbową wpisanej wagi w kilogramach.
- **date** – przechowuje datę, kiedy pomiar został pobrany.

Lista zimportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

#### **5.5.5.5. Klasa ExamplesDefaults i Env**

Klasy ExamplesDefaults i Env zostały zainportowane z biblioteki zewnętrznej SwiftCharts. Są one wymagane do utworzenia wykresu na potrzeby przedstawienia zebranych pomiarów (wartości BMI oraz dane odnośnie wagi). Ich główną rolą jest konfiguracja podstawowych parametrów wykresu.

Wykorzystywane są w Measurements storyboard. Klasy importowane są przez inne obiekty – klasa ChartData.

### 5.5.5.6. Klasa MeasurementsCoreData

Klasa MeasurementsCoreData odpowiedzialna jest za modyfikację (dodawanie, usuwanie, edycja/nadpisanie) danych odnośnie pomiarów. Wszystkie dane wyświetlane są w polach tekstowych. Jest podpięta pod klasę BMIVViewController w Measurements storyboard, jako klasa pomocnicza.

Lista zaimplementowanych funkcji:

- **addMeasurement()** – dodaje pomiar do encji MeasurementsEntity w bazie danych przy użyciu Core Data.
- **removeMeasurement()** – usuwa pomiar z encji MeasurementsEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **modifyMeasurement()** – edytuje pomiar w encji MeasurementsEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **findMaximumID()** – funkcja wykorzystywana przy dodawaniu pomiaru, pozwala na znalezienie największego indeksu ID w celu określenia następnego dla nowego wpisu pomiaru.
- **removeAllMeasurement()** – usuwa wszystkie wcześniej zapisane pomiary z encji MeasurementsEntity w bazie danych przy użyciu Core Data.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.

### 5.5.5.7. Klasa ChartSettingsVC

Klasa ChartSettingsVC odpowiedzialna jest za pobieranie informacji z bazy danych odnośnie zebranych pomiarów. Umożliwia otworzenie okna z wyrysowanym wykresem, ale również przesyła informacje odnośnie daty i rodzaju wykresu do widoku prezentującego.

Wykorzystywana jest w Measurements storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITextFieldDelegate.

Lista zaimplementowanych funkcji:

- **calculateNumbersOfDays()** – zlicza ile dni ma wybrany miesiąc danego roku, aby dopasować podziałkę na osi X na wykresie.
- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Ustawia DatePicker dla pola tekstowego wraz z aktualną datą i wczytuje pomiary z bazy danych.
- **getCurrentDate()** – pobranie aktualnej daty w formacie rok i miesiąc.
- **loadMeasurementsDataFromDB()** – pobranie pomiarów z bazy danych dla wartości typu waga, BMI i data wpisu.
- **textFieldShouldReturn()** – kończy edycje poprzez przyciśnięcie przycisku return.
- **textField()** – zabezpieczenie DatePicker przed wklejaniem i dodawaniem innej treści i znaków niż te podane przez DatePicker.
- **viewTapped()** – ukrycie klawiatury w reakcji na gest dotknięcia („tap”).
- **dateChanged()** – określenie formatu pobranej daty z DatePicker i jej wpisanie do pola tekstowego.
- **prepare()** – w tej funkcji następuje wysyłanie typu wykresu, wybranej daty, pomiarów dla tej daty, maksymalnej wartości na osi X do klasy ChartData.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.

Lista zainportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **Toast\_Swift** – darmowa i zewnętrzna biblioteka pobrała z Cocoa Pods, w celu zaimplementowania do wyświetlania powiadomień w aplikacji.

Lista zastosowanych obiektów typu @IBOutlet:

- **dateTextField** – obiekt typu UITextField, wyświetla datę.
- **segmentedControl** – obiekt typu UISegmentedControl, zawiera dwa przyciski wyboru jednostki (wykres dla kilogramów lub BMI).

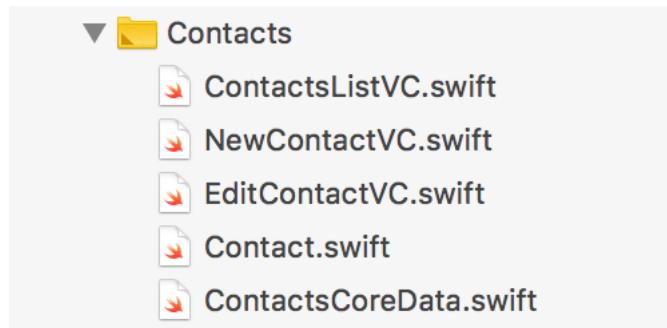
Lista zastosowanych obiektów typu @IBAction:

- **chooseChartType()** – funkcja w zależności od wybranego przycisku ustawia zmienną globalną, która określa rodzaj wykresu.
- **drawChart()** – przygotowanie wykresu dla wybranej daty z DatePicker.

## 5.5.6. Grupa Contacts

Wyodrębniono grupę Contacts, aby wyróżnić obiekty odnoszące się do wyświetlania i edycji listy kontaktów. Schemat przedstawiono na poniższym rysunku (*Rysunek 30*).

Rysunek 30 - Grupa Contacts



Źródło: *Opracowanie własne.*

### 5.5.6.1. Klasa Contact

Klasa Contact reprezentuje budowę encji ContactsEntity w postaci obiektowej. Jest wykorzystywana w klasie ContactsListVC, NewContactVC i EditContactVC w Planner storyboard. Umożliwia przechowanie poszczególnych wpisów z bazy danych, w celu ich późniejszego wyświetlania w polach tekstowych i tabelkach.

Lista zaimplementowanych pól:

- **id** – przechowuje wartość liczbową, która znajduje się w encji ContactsEntity dla wybranego kontaktu. Służy do unikalnej numeracji kontaktów w bazie danych.
- **address** – przechowuje adres w postaci tekstu.
- **contact** – przechowuje numer telefonu w postaci tekstu.
- **contactName** – przechowuje nazwę kontaktu w postaci tekstu.
- **mail** – przechowuje adres e-mail w postaci tekstu.
- **firstName** – przechowuje imię osoby dla danego kontaktu w postaci tekstu.
- **lastName** – przechowuje nazwisko osoby dla danego kontaktu w postaci tekstu.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.

### 5.5.6.2. Klasa ContactsListVC

Klasa ContactsListVC odpowiedzialna jest za pobieranie informacji z bazy danych odnośnie listy kontaktów. Wszystkie dane wyświetlane są w tabeli przy użyciu komponentu TableView. Wykorzystywana jest w Planner storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITableViewDelegate, UITableViewDataSource.

Lista zaimplementowanych funkcji:

- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Blokuje przyciski wyboru dla wierszy w tabeli kontaktów, dopóki żaden nie jest zaznaczony.
- **viewWillAppear()** – kontroler widoku zostaje poinformowany, że jego widok ma się pojawić. Informacje odnośnie kontaktów zostają załadowane z bazy danych oraz przeładowane w tabeli. Blokuje przyciski wyboru dla wierszy w tabeli kontaktów, dopóki żaden nie jest zaznaczony.
- **loadContactsDataFromDB()** – wczytanie danych z bazy danych przy użyciu Core Data. Pobieranie wartości dla kontaktów z encji ContactsEntity.
- **tableView()** – trzy funkcje odpowiedzialne za zliczanie ilości elementów w tabeli, wyświetlanie wartości oraz wywołanie akcji jeśli element został kliknięty.
- **prepare()** – w tej funkcji następuje wysyłanie numeru telefonu zazmaczonego kontaktu z tablicy contactsArray do klasy EditContactVC.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.

Lista zastosowanych obiektów typu @IBOutlet:

- **selectContact** – obiekt typu UIBarButtonItem, przycisk wybierający dany kontakt z listy.
- **editContact** – obiekt typu UIBarButtonItem, przenoszenie do edycji danych tego kontaktu.
- **tableView** – obiekt typu UITableView, wyświetla listę kontaktów z bazy danych.

Lista zastosowanych obiektów typu @IBAction:

- **editContact()** – przenosi do okna z edycją kontaktu.
- **selectContact()** – wybranie danego kontaktu z listy (konkretnie jego numer) i przenosi do okna z edycją wizyty.

### 5.5.6.3. Klasa NewContactVC

Klasa NewContactVC odpowiedzialna jest za zapisywania informacji pobranych z pól tekstowych do bazy danych, odnośnie nowego kontaktu. Wszystkie dane wyświetlane są w polach tekstowych przy użyciu komponentów UITextField.

Wykorzystywana jest w Planner storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITextFieldDelegate, UITextViewDelegate.

Lista zaimplementowanych funkcji:

- **navigateToContactsListView()** – powrót do poprzednio włączonego widoku.
- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci.
- **viewTapped()** – ukrycie klawiatury i DatePicker w reakcji na gest dotknięcia („tap”).
- **textFieldShouldReturn()** – ukrycie klawiatury dla pola tekstowego w wyniku kliknięcia w przycisk „Return”.
- **didReceiveMemoryWarning()** – standardowa funkcja wywoływająca się w przypadku, gdy aplikacja zarejestruje ostrzeżenie odnośnie pamięci.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.

Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **Toast\_Swift** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania do wyświetlanego powiadomień w aplikacji.

Lista zastosowanych obiektów typu @IBOutlet:

- **contactNameTextField** – obiekt typu UITextField, przedstawia nazwę kontaktu.
- **contactTextField** – obiekt typu UITextField, przedstawia numer telefonu.
- **mailTextField** – obiekt typu UITextField, przedstawia e-mail.
- **firstNameTextField** – obiekt typu UITextField, przedstawia imię osoby dla danego kontaktu.
- **lastNameTextField** – obiekt typu UITextField, przedstawia nazwisko osoby dla danego kontaktu.
- **addressTextField** – obiekt typu UITextField, przedstawia adres dla danego kontaktu.

Lista zastosowanych obiektów typu @IBAction:

- **createNewContact()** – wywołanie funkcji, która dodaje kontakt do bazy danych przy użyciu Core Data. Dane pobierane są z pól tekstowych zamieszczonych w widoku kontrolera.

#### 5.5.6.4. Klasa EditContactVC

Klasa EditContactVC odpowiedzialna jest za edytowanie i usuwanie informacji pobranych z bazy danych odnośnie danego kontaktu. Wszystkie dane wyświetlane są w polach tekstowych przy użyciu komponentu UITextField.

Wykorzystywana jest w Planner storyboard, jako definicja działania jednego z kontrolerów typu View Controller. Klasa dziedziczy po następujących obiektach – UIViewController, UITextFieldDelegate, UITextViewDelegate.

Lista zaimplementowanych funkcji:

- **navigateToContactsListView()** – powrót do poprzednio włączonego widoku.
- **viewDidLoad()** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. W polach tekstowych umieszcza dane dla danego kontaktu.
- **viewTapped()** – ukrycie klawiatury i DatePicker w reakcji na gest dotknięcia („tap”).
- **textFieldShouldReturn()** – ukrycie klawiatury dla pola tekstowego w wyniku kliknięcia w przycisk „Return”.
- **didReceiveMemoryWarning()** – standardowa funkcja wywoływająca się w przypadku, gdy aplikacja zarejestruje ostrzeżenie odnośnie pamięci.
- **viewDidAppear()** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu Navigation Bar.

Lista zainportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **Toast\_Swift** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania do wyświetlania powiadomień w aplikacji.
- **PopupDialog** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania tzw. wyskakujących okienek.

Lista zastosowanych obiektów typu @IBOutlet:

- **contactNameTextField** – obiekt typu UITextField, przedstawia nazwę kontaktu.
- **contactTextField** – obiekt typu UITextField, przedstawia numer telefonu.
- **mailTextField** – obiekt typu UITextField, przedstawia e-mail.
- **firstNameTextField** – obiekt typu UITextField, przedstawia imię osoby dla danego kontaktu.
- **lastNameTextField** – obiekt typu UITextField, przedstawia nazwisko osoby dla danego kontaktu.
- **addressTextField** – obiekt typu UITextField, przedstawia adres dla danego kontaktu.

Lista zastosowanych obiektów typu @IBAction:

- **removeContact()** – po przyciśnięciu wyświetlane jest okno dialogowe PopupDialog. Jeśli zatwierdzono usunięcie, to po numerze ID dany kontakt zostaje usunięty z bazy danych.
- **modifyContact()** – po przyciśnięciu wyświetlane jest okno dialogowe PopupDialog. Jeśli zatwierdzono, to wprowadzone dane nadpisują już istniejące dane w bazie danych dla danego kontaktu.

### 5.5.6.5. Klasa ContactsCoreData

Klasa ContactsCoreData odpowiedzialna jest za modyfikację (dodawanie, usuwanie, edycja/nadpisanie) kontaktów. Wszystkie dane wyświetlane są w polach tekstowych. Jest podpięta pod klasę NewContactVC i EditContactVC w Planner storyboard, jako klasa pomocnicza.

Lista zaimplementowanych funkcji:

- **addContact()** – dodaje kontakt do encji ContactsEntity w bazie danych przy użyciu Core Data.
- **removeContact()** – usuwa kontakt z encji ContactsEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **modifyContact()** – edytuje kontakt w encji ContactsEntity w bazie danych przy użyciu Core Data, po wybranym numerze ID.
- **findMaximumID()** – funkcja wykorzystywana przy dodawaniu kontaktu, pozwala na znalezienie największego indeksu ID w celu określenia następnego dla nowego kontaktu.
- **removeAllContacts()** – usuwa wszystkie kontakty z encji ContactsEntity w bazie danych przy użyciu Core Data.

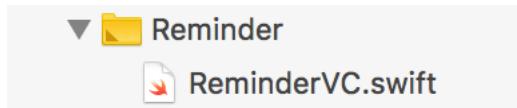
Lista zainportowanych bibliotek:

- **Foundation** – umożliwia dostęp do podstawowych funkcjonalności języka Swift.
- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.

### 5.5.7. Grupa Reminder

Wyodrębniono grupę Reminder, aby wyróżnić obiekt, który służy do zarządzania ustawieniami funkcji przypominającej o wydarzeniu. Schemat przedstawiono na poniższym rysunku (*Rysunek 31*).

Rysunek 31 - Grupa Reminder



Źródło: *Opracowanie własne.*

#### 5.5.7.1. Klasa ReminderVC

Klasa `ReminderVC` odpowiedzialna jest za włączanie i wyłączanie powiadomień o zaplanowanej wizycie. Możliwość wyboru daty na ustawienie przypomnienia przy użyciu `DatePicker`.

Wykorzystywana jest w `Planner` storyboard, jako definicja działania jednego z kontrolerów typu `View Controller`. Klasa dziedziczy po następujących obiektach – `UIViewController`, `UITextFieldDelegate`, `UITextViewDelegate`.

Lista zaimplementowanych funkcji:

- **`navigateToAppointmentView()`** – powrót do poprzedniego okna z edycją wizyty.
- **`viewDidLoad()`** – jest wywoływanie po załadowaniu widoku kontrolera do pamięci. Wpisuje datę wizyty i datę przypomnienia oraz ustawia przełącznik.
- **`textField()`** – zabezpieczenie przez wpisaniem innej wartości niż pobrana z `DatePicker`.
- **`viewTapped()`** – ukrycie klawiatury i `DatePicker` w reakcji na gest dotknięcia („tap”).
- **`dateChanged()`** – określenie formatu pobranej daty z `DatePicker` i wpisanie jej do pola.
- **`textFieldShouldReturn()`** – ukrycie klawiatury dla pola tekstowego w wyniku kliknięcia w przycisk „Return”.
- **`viewDidAppear()`** – kontroler widoku zostaje poinformowany, że jego widok się już pojawił. Dodanie logo aplikacji w postaci ikony do paska nawigacyjnego typu `Navigation Bar`.

Lista zaimportowanych bibliotek:

- **UIKit** – udostępnia wymagane komponenty do implementacji interfejsu graficznego.
- **CoreData** – zarządzanie obiektami w bazie danych.
- **Toast\_Swift** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania do wyświetlania powiadomień w aplikacji.
- **PopupDialog** – darmowa i zewnętrzna biblioteka pobrana z Cocoa Pods, w celu zaimplementowania tzw. wyskakujących okienek.

Lista zastosowanych obiektów typu @IBOutlet:

- **visitTextField** – obiekt typu UITextField, pole tekstowe z data wizyty.
- **reminderTextField** – obiekt typu UITextField, pole tekstowe z datą przypomnienia o wizycie.
- **switchReminder** – obiekt typu UISwitch, przełącznik wskazujący czy przypomnienie zostało włączone lub wyłączone.

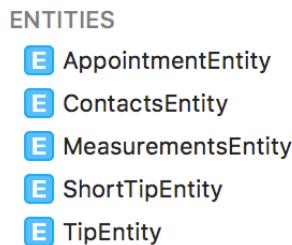
Lista zastosowanych obiektów typu @IBAction:

- **confirmReminder()** – sprawdza czy przełącznik typu switch jest włączony lub wyłączony. W przypadku, gdy jest włączony to ustawia zmienną globalną i zamyka okno, cofając się tym samym do okna z edycją wizyt.

## 5.6. Budowa encji w CoreData

Na potrzebę aplikacji opracowano encje rozdzielone według głównych kategorii aplikacji. Przedstawiono je na poniższym rysunku (*Rysunek 32*). Dokładniejszy opis znajduje się w dalszych podrozdziałach.

Rysunek 32 - Zaimplementowane encje w CoreData



Źródło: *Opracowanie własne*.

### 5.6.1. Encja AppointmentEntity

Wszystkie wymagane pola dla zarządzania wizytami zostały zawarte w encji AppointmentEntity (*Rysunek 33*).

Rysunek 33 - Budowa encji AppointmentEntity

▼ Attributes		
Attribute	Type	^
N colorNumber	Integer 64	◊
N id	Integer 64	◊
S address	String	◊
S contact	String	◊
S duration	String	◊
S name	String	◊
S notes	String	◊
B isAllDay	Boolean	◊
B reminder	Boolean	◊
D date	Date	◊
D reminderDate	Date	◊

Źródło: *Opracowanie własne*.

## 5.6.2. Encja ContactsEntity

Wszystkie wymagane pola dla zarządzania kontaktami zostały zawarte encji ContactsEntity (*Rysunek 34*).

Rysunek 34 - Budowa encji ContactsEntity

▼ Attributes		
Attribute	Type	^
N id	Integer 64	◊
S address	String	◊
S contact	String	◊
S contactName	String	◊
S firstName	String	◊
S lastName	String	◊
S mail	String	◊

Źródło: Opracowanie własne.

## 5.6.3. Encja MeasurementsEntity

Wszystkie wymagane pola dla zarządzania pomiarami zostały zawarte encji MeasurementsEntity (*Rysunek 35*).

Rysunek 35 - Budowa encji MeasurementsEntity

▼ Attributes		
Attribute	Type	^
N id	Integer 64	◊
N resultBMI	Double	◊
N weight	Double	◊
D date	Date	◊

Źródło: Opracowanie własne.

#### 5.6.4. Encja ShortTipEntity

Wszystkie wymagane pola dla zarządzania krótkimi ciekawostkami zostały zawarte encji ShortTipsEntity (*Rysunek 36*).

Rysunek 36 - Budowa encji ShortTipsEntity

▼ Attributes		
Attribute	Type	^
N id	Integer 64	◊
S shortTip	String	◊
<hr/>		
+	-	

Źródło: *Opracowanie własne.*

#### 5.6.5. Encja TipEntity

Wszystkie wymagane pola dla zarządzania poradami zostały zawarte encji TipEntity (*Rysunek 37*).

Rysunek 37 - Budowa encji TipEntity

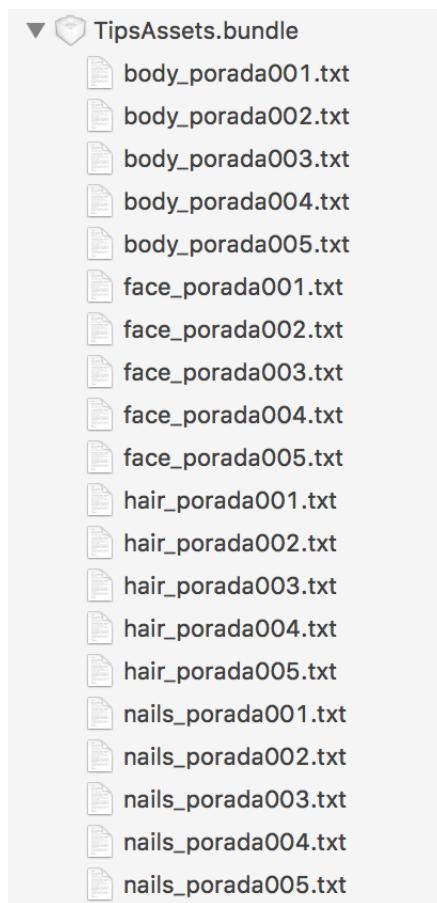
▼ Attributes		
Attribute	Type	^
N id	Integer 64	◊
S category	String	◊
S pictureName	String	◊
S tip	String	◊
S title	String	◊
<hr/>		
+	-	

Źródło: *Opracowanie własne.*

## 5.7. Struktura zasobów typu Bundle

W celu realizacji funkcjonalności do przeglądania porad wymagane było stworzenie plików tekstowych, które są w stanie przechowywać dłuższą zawartość dla porady. Dodatkową zaletą jest łatwość edycji porady poprzez edycję pliku tekstopowego. Porady dla wszystkich kategorii zostały przedstawione na poniższym rysunku (*Rysunek 38*). Znajdują się one w dedykowanej przestrzeni zwanej Bundle, która umożliwia przechowywanie dodatkowych zasobów dla aplikacji.

Rysunek 38 - Struktura zasobów typu Bundle

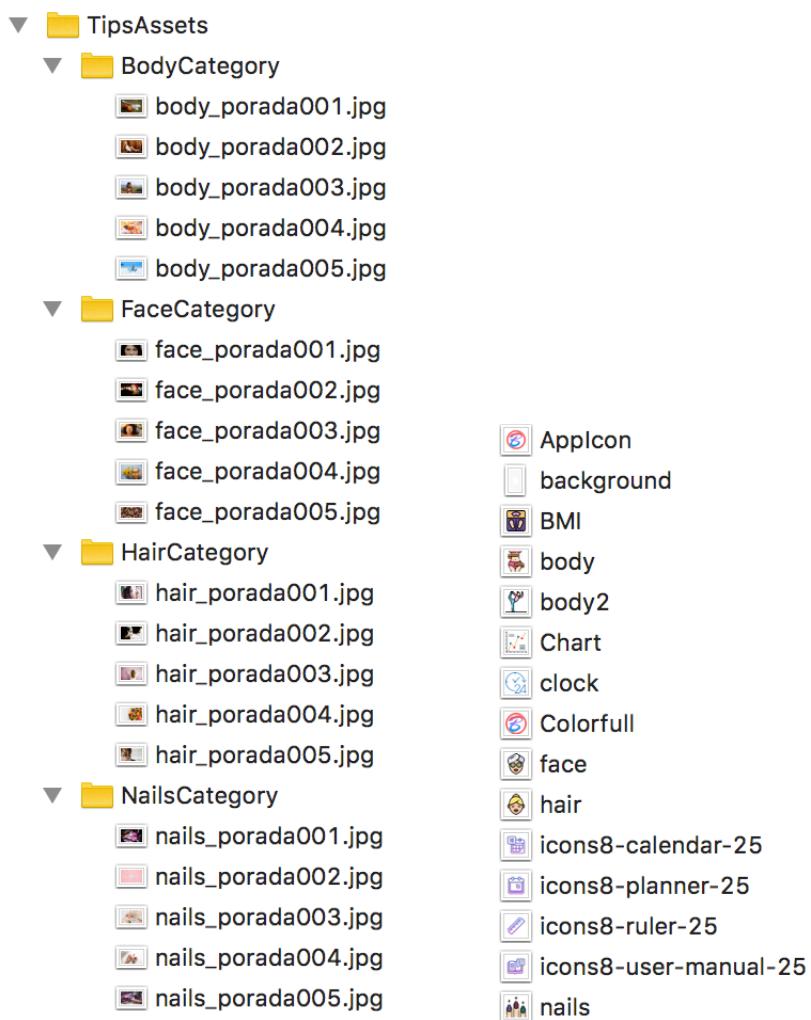


Źródło: *Opracowanie własne.*

## 5.8. Struktura zasobów typu Assets

Zasoby w postaci ikon oraz obrazków zostały załączone w dedykowanym miejscu o nazwie Assets. Zdjęcia wykorzystane w dziale porad podzielono na cztery kategorie widoczne na poniższym rysunku (*Rysunek 39*). Pozostałe ikony nie wymagały pogrupowania.

Rysunek 39 - Struktura zasobów typu Assets

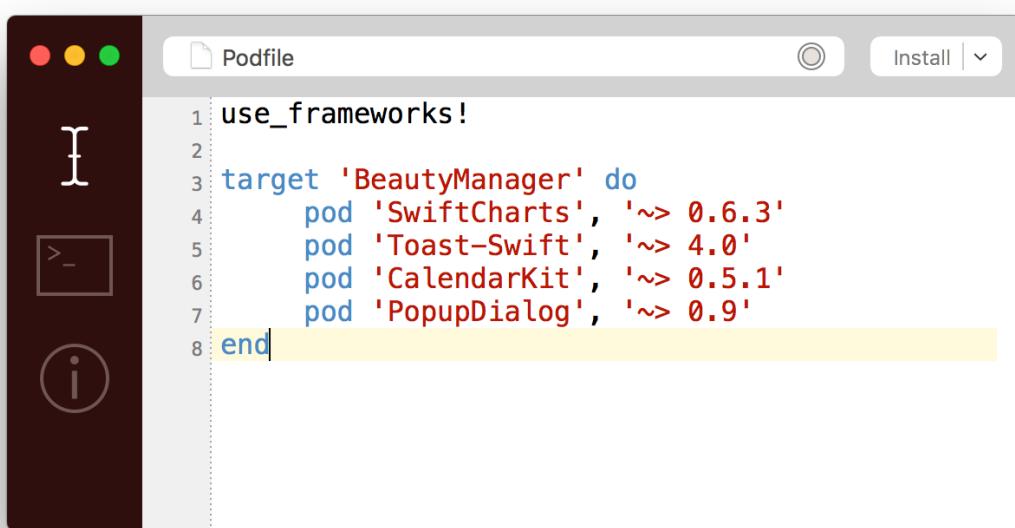


Źródło: Opracowanie własne.

## 5.9. Opis zastosowanych bibliotek CocoaPods

W tym dziale zawarte są opisy wszystkich zastosowanych bibliotek. Do znalezienia darmowych bibliotek posłużył serwis internetowy <https://cocoapods.org/>. W celu poprawnego zainstalowania wymaganych zależności zainstalowano oprogramowanie CocoaPods, przedstawione na poniższym rysunku (Rysunek 40). Informacje zaczerpnięto ze źródeł zewnętrznych, które można znaleźć załączone w bibliografii [1].

Rysunek 40 - Zaimportowane biblioteki CocoaPods

A screenshot of the Xcode interface showing a Podfile editor window. The window title is "Podfile". The code in the editor is:

```
use_frameworks!
target 'BeautyManager' do
    pod 'SwiftCharts', '~> 0.6.3'
    pod 'Toast-Swift', '~> 4.0'
    pod 'CalendarKit', '~> 0.5.1'
    pod 'PopupDialog', '~> 0.9'
end
```

The "Install" button is visible in the top right corner of the window.

Źródło: Opracowanie własne.

Manager zależności wspiera język Swift oraz Objective-C. W przypadku aplikacji Beauty Planner wybrano jedynie komponenty stworzone jedynie przy użyciu języka Swift. Określono również dokładne wersje poszczególnych bibliotek, co dokładnie opisano w następnych podrozdziałach.

### **5.9.1. Biblioteka SwiftCharts**

Biblioteka SwiftCharts umożliwia tworzenie różnego typu wykresów i diagramów. Wybrano najnowszą z dostępnych wersji – 0.6.3, która działa wystarczająco stabilnie. Na potrzeby niniejszej pracy dyplomowej wykorzystano ją do przedstawienia zebranych pomiarów w dziale „Body measurements”. Wprowadzono modyfikacje umożliwiające kontrolę podziałek dla osi pionowej i poziomej, jak również tekstowy opis dla osi X i Y. Ze względu na specyfikę pomiarów zastosowano wykres punktowy z naniesioną linią symbolizującą aktualny trend (rosnący lub malejący).

Klasy w których zaimplementowano użycie powyższej biblioteki:

- ChartData.swift

Kod źródłowy pobrano z poniższej strony:

- <https://cocoapods.org/pods/SwiftCharts>

### **5.9.2. Biblioteka CalendarKit**

Biblioteka CalendarKit umożliwia prezentowanie zdarzeń na darmowym kalendarzu wbudowanym w aplikację. Wybrano wersję – 0.5.1. Zdecydowano się na wybór tego komponentu ze względu na łatwość obsługi głównych funkcjonalności oraz możliwość łatwego wprowadzania w nich zmian. Między innymi wykorzystano w aplikacji nanoszenie zdarzeń na kalendarz, ustawienia kolorów dla konkretnych wydarzeń, ich długość trwania, jak również obsłużono gest przytrzymania bloku danego wydarzenia w celu zaimplementowania dodatkowej akcji.

Klasy w których zaimplementowano użycie powyższej biblioteki:

- CalendarKit.swift

Kod źródłowy pobrano z poniższej strony:

- <https://cocoapods.org/pods/CalendarKit>

### **5.9.3. Biblioteka Toast-Swift**

Biblioteka Toast-Swift umożliwia tworzenie różnego powiadomień typu Toast. Wybrano wersję – 4.0, ponieważ okazała się ona wystarczająca na potrzeby niniejszej aplikacji. Głównym zastosowaniem tej biblioteki są krótkie powiadomienia, zazwyczaj zlokalizowane u dołu ekranu. Dla każdego rodzaju informacji przypisano odpowiedni kolor. Czas wyświetlania dla wszystkich powiadomień ustalono na trzy sekundy. Wykorzystano do walidacji pól tekstowych, wyświetlania kategorii dla wybranego zakresu wyniku BMI.

Klasy w których zaimplementowano użycie powyższej biblioteki:

- ReminderVC.swift
- NewContactVC.swift
- EditContactVC.swift
- BMIVViewController.swift
- ChartSettingsVC.swift
- EditAppointmentVC.swift
- NewAppointmentVC.swift

Kod źródłowy pobrano z poniższej strony:

- <https://cocoapods.org/pods/Toast-Swift>

#### **5.9.4. Biblioteka PopupDialog**

Biblioteka PopupDialog pozwala na animowane włączenie okna dialogowego nad widokiem. Dodatkowo wykorzystano możliwość użycia opcji wyboru w formie przycisków. Zazwyczaj pełnią funkcję decyzyjną lub wyświetlającą więcej informacji.

Posłużyono się wersją – 0.9.

Klasy w których zaimplementowano użycie powyższej biblioteki:

- ReminderVC.swift
- EditContactVC.swift
- BMIVViewController.swift
- CalendarKit.swift
- ViewController.swift
- EditAppointmentVC.swift

Kod źródłowy pobrano z poniższej strony:

- <https://cocoapods.org/pods/PopupDialog>

## **6. Prezentacja funkcjonalności i wybranych fragmentów kodu**

W tym dziale przedstawione zostały główne funkcjonalności aplikacji mobilnej. Niektóre informacje zaczerpnięto ze źródeł zewnętrznych, które można znaleźć załączone w bibliografii [12-18] [21-65].

Należą do nich:

- Krótkie porady, dotyczące ciekawostek kosmetycznych.
- Wyświetlanie wizyt na liście.
- Dodawanie, edycja i usunięcie wizyty.
- Wysyłanie wiadomości typu SMS.
- Wyświetlanie listy kontaktów.
- Dodawanie, edycja i usunięcie kontaktu.
- Ustawienia powiadomień o zbliżającym się terminie.
- Ustawienia i przegląd kalendarza.
- Przeglądanie porad po wybranej kategorii.
- Wyliczanie indeksu BMI.
- Prezentacja przedziałów klasyfikacji BMI.
- Zapisywanie wartości BMI z wybraną datą.
- Usunięcie wszystkich zapisanych wartości BMI.
- Tworzenie wykresu dla wybranej jednostki.
- Okno wyboru typu PopupDialog.
- Powiadomienia typu Toast.

Załączono również wybrane fragmenty kodów programistycznych wykonane w języku Swift.

## 6.1. Krótkie porady

Po uruchomieniu aplikacji włącza się okno startowe, na którym znajduje się nazwa aplikacji „Beauty Planner”, imię i nazwisko autora, logo aplikacji umiejscowione na samym środku okna, poniżej niego zostało umieszczony przycisk do krótkich ciekawostek, natomiast na samym dole przycisk, który przenosi użytkownika do wnętrza aplikacji (*Rysunek 41*).

Logo aplikacji zostało zaprojektowane i wykonane w aplikacji Adobe Photoshop. Stworzono w nim dwie litery połączone w jedną, z różnymi kolorami dla lepszego odróżnienia.

Tło dla strony startowej również wykonano w aplikacji Adobe Photoshop. Wykorzystano gotowe ikony pobrane ze strony internetowej z darmowymi obrazkami, ikonami i wektorami.

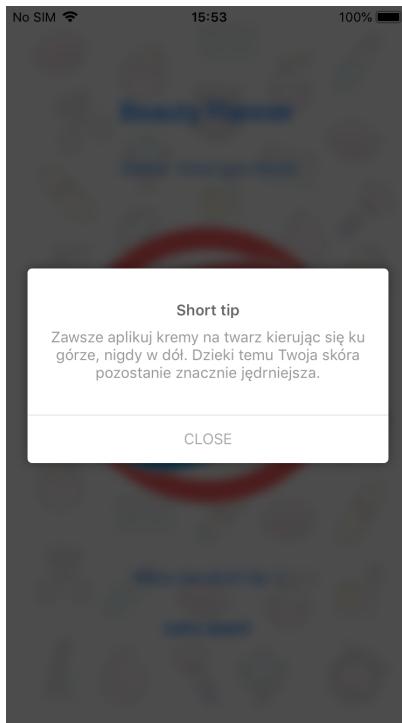
Rysunek 41 - Strona startowa aplikacji



Źródło: Opracowanie własne.

Po kliknięciu w przycisk o nazwie „Show me short tip :)” ukazuje się okno dialogowe (PopupDialog) z krótką ciekawostką, która jest wyświetlana losowo ze zbioru porad dla strony startowej (*Rysunek 42*).

Rysunek 42 - Okno wyświetlające krótką poradę na stronie startowej



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 1*) przedstawia wyświetlenie okna dialogowego typu PopupDialog. W tej funkcji zliczany jest rozmiar tablicy „shortTipsArray” zawierający krótkie porady. Następnie wybierany jest losowo numer krótkiej porady na podstawie ilości elementów z tablicy, który zostaje przypisany do zmiennej „numberOfShortTip”.

W oknie dialogowym na samej górze umieszczono na stałe tytuł „Short tip”. Zaraz pod nim znajduje się treść ciekawostki (w kodzie przypisana do zmiennej „shortTipMessage”), która wcześniej została wybrana losowo. Na dole okna jest umieszczony przycisk akcji o nazwie „closeButton” odpowiedzialny za zamknięcie okienka.

```
@IBAction func showMeShortTip(_ sender: Any) {
    var sizeOfArray: Int = shortTipsArray.count
    let numberOfShortTip = Int.random(in: 0...(sizeOfArray - 1))
    let mainTitle = "Short tip"
    let shortTipMessage = shortTipsArray[numberOfShortTip].shortTip
    let popupDialog = PopupDialog(title: mainTitle, message:
shortTipMessage)
    let closeButton = CancelButton(title: "CLOSE") {}
    popupDialog.addButton([closeButton])
    self.present(popupDialog, animated: true, completion: nil)
}
```

Kod 1 - Losowanie i wyświetlanie krótkiej porady

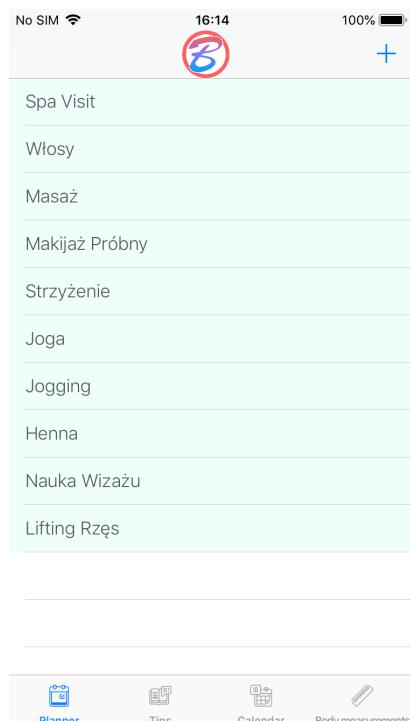
## 6.2. Wyświetlanie wizyt na liście

Po przejściu ze strony startowej aplikacji, użytkownik zostaje przekierowany na stronę z listą zaplanowanych wizyt. Zakładka ta ma nazwę „Planner” i jest główną stroną dla tej grupy. W środku górnego paska nawigacyjnego umieszczono miniaturę logo, natomiast po prawej stronie znajduje się przycisk w kształcie plusa, który służy do dodania nowej wizyty (*Rysunek 43*).

W poniższym rysunku znajduje się przykładowa lista wizyt. Pusta lista jest cała biała z widocznymi liniami odstępu. Przy naniesieniu wydarzenia do listy, kolor wiersza zmienia się z białego na jasny miętowy. Cała lista jest przewijana przy zwiększonej ilości wpisów.

W dolnym pasku nawigacyjnym znajdują się cztery ikony, każda z nich jest podpisana w celu lepszego nawigowania po aplikacji. Strona, na której znajduje się użytkownik podświetla ikonę na dolnym pasku nawigacyjnym na kolor niebieski, natomiast pozostałe, są w kolorach szarych. W poniższych rysunkach podświetlona jest pierwsza ikona dotycząca aktualnej strony o nazwie „Planner”.

Rysunek 43 - Wyświetlanie wizyt na liście



Źródło: Opracowanie własne.

### **6.3. Dodawanie nowej wizyty do listy**

Na głównej stronie grupy „Planner”, w górnym pasku nawigacyjnym po prawej stronie, po przyciśnięciu przycisku w kształcie plusa, użytkownik zostaje przekierowany do okna z dodaniem do listy nowej wizyty (*Rysunek 44*). Jest to jedna z podstron grupy „Planner”.

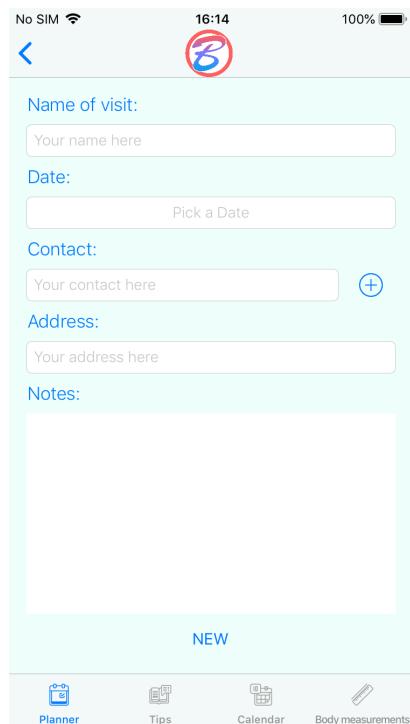
W środku górnego paska nawigacyjnego umieszczono miniaturę logo, natomiast po lewej stronie znajduje się przycisk w kształcie mniejszości, który służy do powrotu na poprzednią stronę.

W poniższym rysunku znajdują się podpisane pola tekstowe (nazwa wizyty, data wizyty, kontakt, adres i miejsce na notatkę użytkownika). Przy polu z kontaktem został umieszczony przycisk, który włącza kolejną stronę z listą kontaktów (*Rysunek 44*).

Aby użytkownik mógł dodać wpis do listy musi koniecznie wypełnić dwa pola, które są wymagane, takie jak nazwa wizyty oraz data wizyty (*Rysunek 45 b*)).

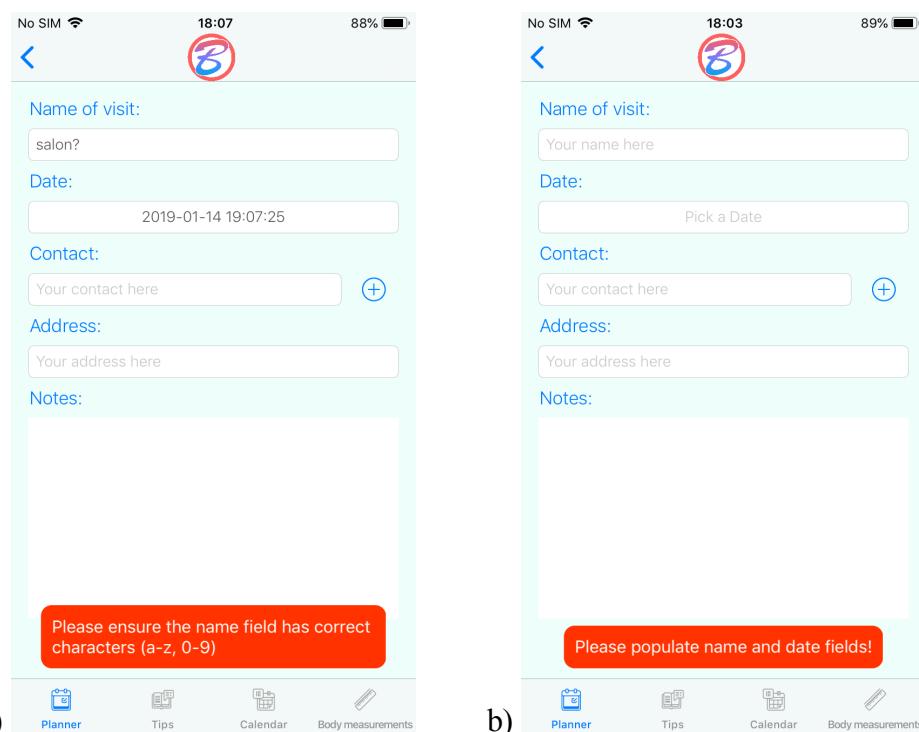
Dla poprawnego zapisu wydarzenia do bazy danych, stworzono zabezpieczenia przed wpisywaniem przez użytkownika innych znaków niż te które są zalecane. Jeśli użytkownik się pomyli lub przypadkowo wpisze niechciany znak, nie będzie mógł zapisać wpisu i dostanie powiadomienie typu toast na dole okna w kolorze czerwonym, które zawiera informację w jakiego zakresu znaków lub liter można użyć (*Rysunek 45 a*)).

Rysunek 44 - Widok z dodawaniem nowej wizyty do listy



Źródło: Opracowanie własne.

Rysunek 45 – Widok z walidacją pól tekstowych w dodawaniu nowej wizyty



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 2*) przedstawia tablicę znaków, która jest wykorzystana w zabezpieczeniu pól tekstowych. Inne znaki poza umieszczonymi w tablicy nie są dopuszczalne.

W tej funkcji przedstawiono kod realizujący dodanie nowej wizyty do bazy danych. Wzbogacony o sprawdzenie czy pola z nazwą wizyty i datą nie są puste.

```
let alphabetRule: CharacterSet = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
"a", "ą", "b", "c", "ć", "d", "e", "ę", "f", "g", "h", "i", "j", "k", "ł",
"ł", "m", "n", "ń", "o", "ó", "p", "q", "r", "s", "ś", "t", "u", "v", "w",
"x", "y", "z", "ź", "ż", " ", "-"]

@IBAction func addNewAppointment(_ sender: Any) {
    style.backgroundColor = UIColor(
        red: 255.0/255.0,
        green: 50.0/255.0,
        blue: 0.0/255.0,
        alpha: 1.0)
    style.messageColor = .white
    let finalSet = CharacterSet.letters.union(alphabetRule)

    if (nameTextField.text != "" && dateTextField.text != "") {
        if (finalSet.isSuperset(of: CharacterSet(charactersIn:
nameTextField.text!)) == true) {
            _ = CoreDataOperations().addAppointment(
                nameValue: nameTextField.text!,
                dateValue: dateTextField.text!,
                contactValue: contactTextField.text!,
                addressValue: addressTextField.text!,
                notesValue: notesTextView.text!,
                reminderValue: isReminderEnabled,
                reminderDateValue: remindDate,
                durationValue: "60",
                colorNumberValue: 0,
                isAllDayValue: false)
            navigateToPreviousView()
        } else {
            self.view.makeToast("Please ensure the name field has
correct characters (a-z, 0-9)", duration: 3.0, position: .bottom, style:
style)
        }
    } else {
        self.view.makeToast("Please populate name and date fields!", duration: 3.0, position: .bottom, style: style)
    }
}
```

Kod 2 - Dodanie nowej wizyty do bazy danych wraz z walidacją pól tekstowych

## **6.4. Edycja istniejącej wizyty**

Na głównej stronie grupy „Planner”, po wybraniu kliknięciem w wydarzenie z listy, użytkownik zostaje przekierowany do okna z edycją wizyty (*Rysunek 46*). Jest to kolejna z podstron grupy „Planner”.

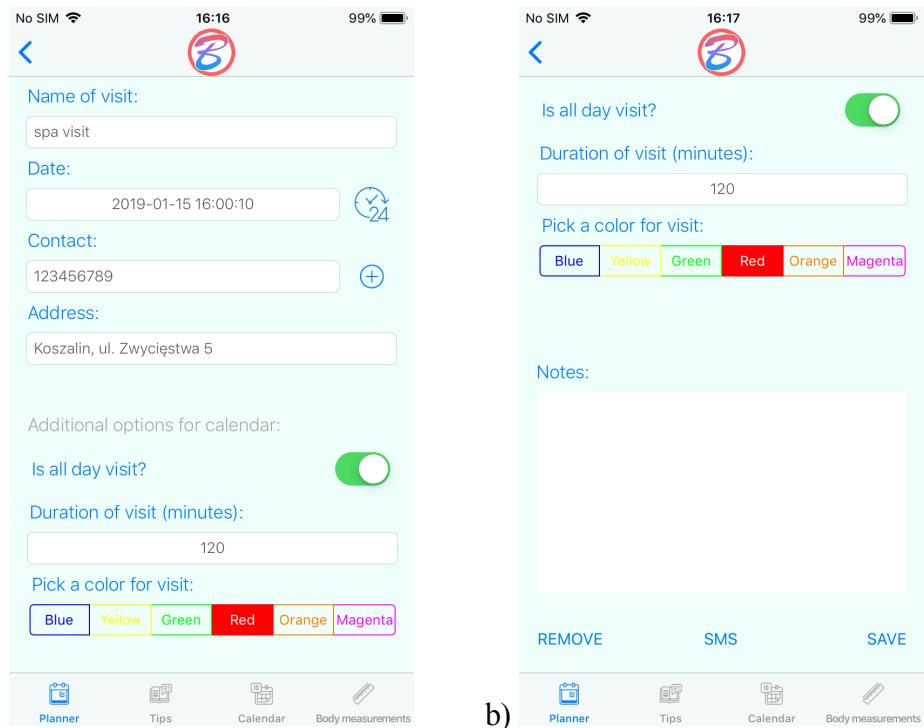
W środku górnego paska nawigacyjnego umieszczono miniaturę logo, natomiast po lewej stronie znajduje się przycisk w kształcie mniejszości, który służy do powrotu na poprzednią stronę.

W poniższym rysunku znajdują się podpisane pola tekstowe (nazwa wizyty, data wizyty, kontakt, adres, czas trwania wizyty i miejsce na notatkę użytkownika), niektóre już wypełnione przez użytkownika przy dodaniu wizyty. Przy polu z datą został umieszczony przycisk, który otwiera kolejną stronę z ustawieniem przypomnienia (*Rysunek 55*), natomiast przy polu z kontaktem został umieszczony przycisk, który włącza kolejną stronę z listą kontaktów (*Rysunek 50*).

W oknie edycji dodano nową funkcjonalność jaką jest zaznaczenie czy wizyta jest całodniowa, poniżej wpisanie przedziału czasowego dla wydarzenia i wybór koloru dla wizyty, która jest nanoszona w kalendarzu (*Rysunek 46 a) i b)*).

Na samym dole strony zostały umieszczone trzy przyciski (od lewej: usuń, sms, zapisz). Po naciśnięciu przycisku o nazwie „REMOVE” zostaje wyświetlane okno dialogowe dla potwierdzenia usunięcia wydarzenia, jeśli zostanie to potwierdzone to wizyta zostaje usunięta z bazy danych po numerze ID. Przycisk o nazwie „SMS” po kliknięciu wyświetla okno dialogowe z wyborem szablonu wiadomości. Ostatni przycisk o nazwie „SAVE” nadpisuje dane wydarzenie.

Rysunek 46 - Widok z edycją istniejącej wizyty



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 3*) zostaje uruchomiony po kliknięciu w przycisk „SAVE” i pobiera wybrane wartości z wymienionych niżej pól, a następnie zapisuje je w bazie danych. Dodatkowo sprawdza czy przełącznik ze strony z ustawieniem przypomnienia dla wizyty został włączony lub wyłączony i zgodnie z tym włącza lub wyłącza alert przypominający.

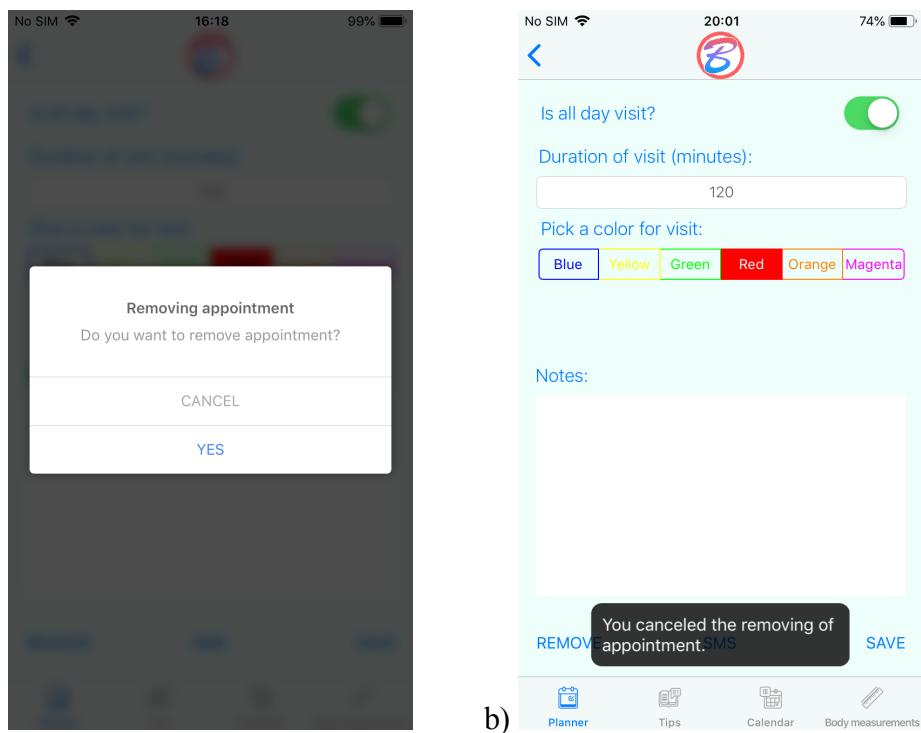
```
CoreDataOperations().modifyAppointment(
    id: self.currentAppointmentID,
    nameValue: self.nameOfVisitTextField.text!,
    dateValue: self.dateTextField.text!,
    contactValue: self.contactTextField.text!,
    addressValue: self.addressTextField.text!,
    notesValue: self.notesTextView.text!,
    reminderValue: self.isReminderEnabled,
    reminderDateValue: self.remindDate,
    durationValue: self.durationTextField.text!,
    colorNumberValue:
Int64(self.colorPickerController!.selectedSegmentIndex),
    isAllDayValue: self.allDaySwitch.isOn)
if (self.isReminderEnabled == true) {
    self.turnOnReminder()
} else {
    self.turnOffReminder() }
self.navigateToPreviousView()
```

Kod 3 - Realizacja edycji wizyty

## 6.5. Usunięcie istniejącej wizyty

Po przyciśnięciu w przycisk o nazwie „REMOVE” ukazuje się okno dialogowe typu PopupDialog do podjęcia akcji usunięcia danej wizyty. Na jednym z rysunków (*Rysunek 47 a)*) został pokazany przykład dla anulowania usunięcia wizyty. Po anulowaniu okno dialogowe znika i zostaje wyświetlona informacja typu Toast na dole strony w kolorze czarnym (*Rysunek 47 b)*) potwierdzająca, że użytkownik anulował usunięcie wizyty. Jeśli natomiast wizyta została usunięta widok w aplikacji zostaje przekierowany do głównej strony grupy „Planner”, czyli do listy z zaplanowanymi wizytami.

Rysunek 47 - Widok dla usunięcia wybranej wizyty



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 4*) przedstawia wywołanie funkcji usuwającej wizytę z bazy danych po konkretnym ID.

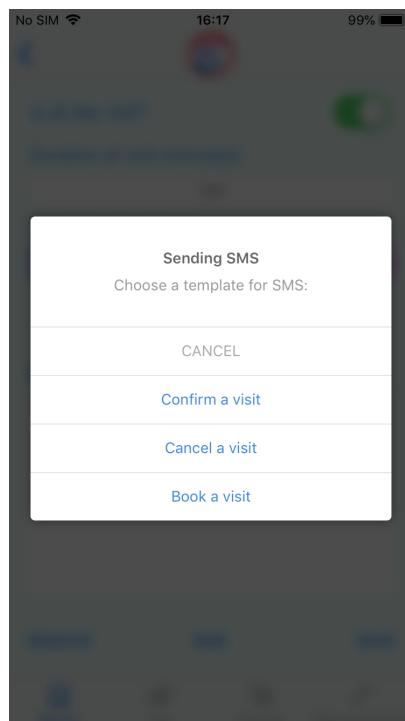
```
CoreDataOperations().removeAppointment(  
    id: self.currentAppointmentID)
```

Kod 4 - Wywołanie funkcji usuwającej wizytę

## 6.6. Wysyłanie wiadomości typu SMS

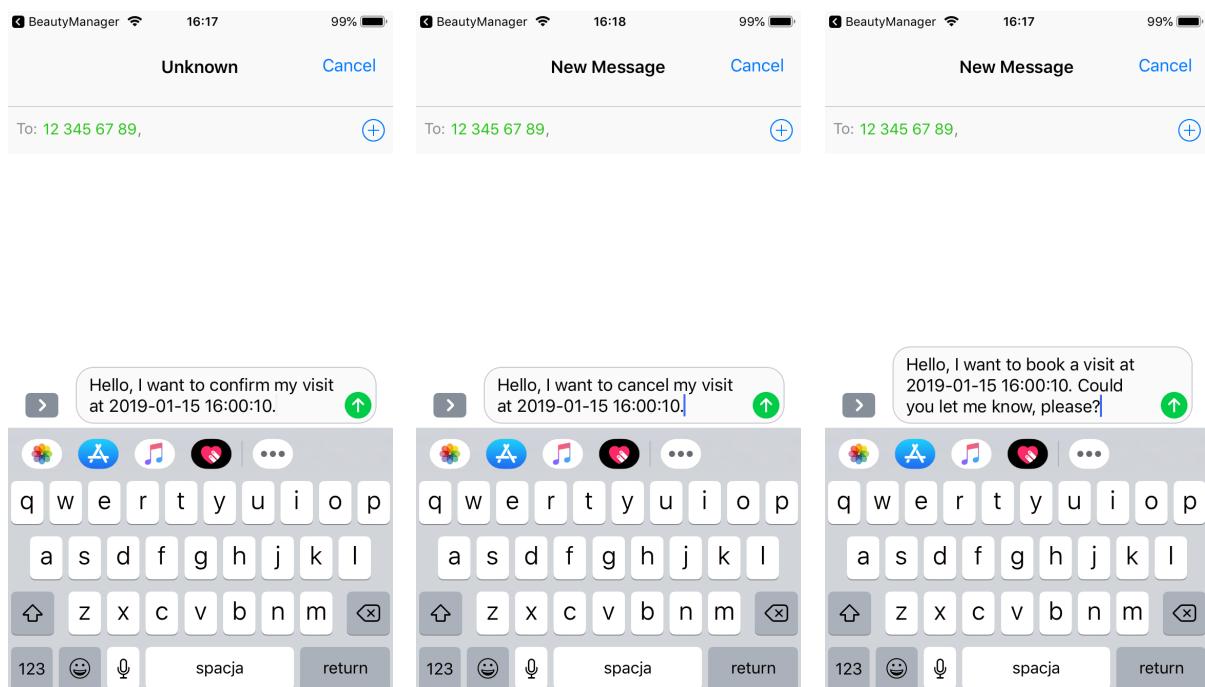
Na stronie edycji wizyty po przyciśnięciu w przycisk o nazwie „SMS” ukazuje się okno dialogowe typu PopupDialog z czterema przyciskami do wyboru (*Rysunek 48*). Pierwszy z nich to „CANCEL”, który anuluje przejście do aplikacji umożliwiającej wysyłanie SMS-ów. Następne trzy przyciski to szablony wiadomości (*Rysunek 49*) jakie może wysłać użytkownik na podany wcześniej numer telefonu lub wybrany z listy kontaktów. Pierwszy szablon z listy dotyczy potwierdzenia wiadomości na daną datę i godzinę, które wprowadził użytkownik przy zapisie wizyty. Drugi szablon z wiadomością dotyczy anulowania wizyty z konkretnego dnia i godziny. Trzeci i ostatni szablon zawiera prośbę o umówienie wizyty na podany dzień i wyznaczoną godzinę.

Rysunek 48 – Widok z wyświetleniem okna dialogowego z wyborem szablonu wiadomości typu SMS



Źródło: Opracowanie własne.

Rysunek 49 - Gotowe szablony wiadomości



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 5*) przedstawia funkcję odpowiedzialną za wysyłanie wiadomości typu SMS przy użyciu domyślnej aplikacji wbudowanej w system. Okno dialogowe typu PopupDialog umożliwia wybór gotowego szablonu wiadomości. Zawarto tekst odnośnie potwierdzenia wizyty, jej anulowania i prośbę o umówienie spotkania na dany termin i konkretną godzinę. Wszystkie wiadomości pobierają wcześniej wybraną datę. Wiadomości nie wysyłają się automatycznie, ale umożliwiają ich wcześniejszy podgląd. Wysłanie następuje dopiero po zatwierdzeniu przez użytkownika, a cała wiadomość zostanie zapisana w domyślnym komunikatorze. Samo wywołanie odpowiedniej aplikacji do wysyłania wiadomości typu SMS następuje po otworzeniu specjalnego ciągu znaków (zawierającego przedrostek „sms:”). Numer telefonu, na który ma zostać wysłana wiadomość jest automatycznie kopiowany pomiędzy aplikacją Beauty Manager, a domyślnie wbudowaną aplikacją do SMS-ów. Takie rozwiążanie umożliwia szybsze i wygodniejsze przygotowanie poprawnej wiadomości. Użytkownik może anulować proces wysyłania wiadomości SMS w dowolnym momencie, a nawet powrócić do głównej aplikacji.

```

@IBAction func sendSMS(_ sender: Any) {
    // POPUP DIALOG
    let mainTitle = "Sending SMS"
    let question = "Choose a template for SMS:"
    let popupDialog = PopupDialog(title: mainTitle, message: question)
    let cancelButton = CancelButton(title: "CANCEL") {
        self.view.makeToast("You canceled the sending a SMS.",
duration: 3.0, position: .bottom)
    }
    let confirmSmsButton = DefaultButton(title: "Confirm a visit",
dismissOnTap: true) {
        let sms: String = "sms:" + self.contactTextField.text! +
"&body=Hello, I want to confirm my visit at " + self.dateTextField.text! +
"."
        let strURL: String =
sms.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)!
UIApplication.shared.open(URL.init(string: strURL)!, options:
[], completionHandler: nil)
    }

    let cancelSmsButton = DefaultButton(title: "Cancel a visit",
dismissOnTap: true) {
        let sms: String = "sms:" + self.contactTextField.text! +
"&body=Hello, I want to cancel my visit at " + self.dateTextField.text! +
"."
        let strURL: String =
sms.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)!
UIApplication.shared.open(URL.init(string: strURL)!, options:
[], completionHandler: nil)
    }

    let questionSmsButton = DefaultButton(title: "Book a visit",
dismissOnTap: true) {
        let sms: String = "sms:" + self.contactTextField.text! +
"&body=Hello, I want to book a visit at " + self.dateTextField.text! +
". Could you let me know, please?"
        let strURL: String =
sms.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)!
UIApplication.shared.open(URL.init(string: strURL)!, options:
[], completionHandler: nil)
    }
    popupDialog.addButton([cancelButton, confirmSmsButton,
cancelSmsButton, questionSmsButton])
    self.present(popupDialog, animated: true, completion: nil)
}

```

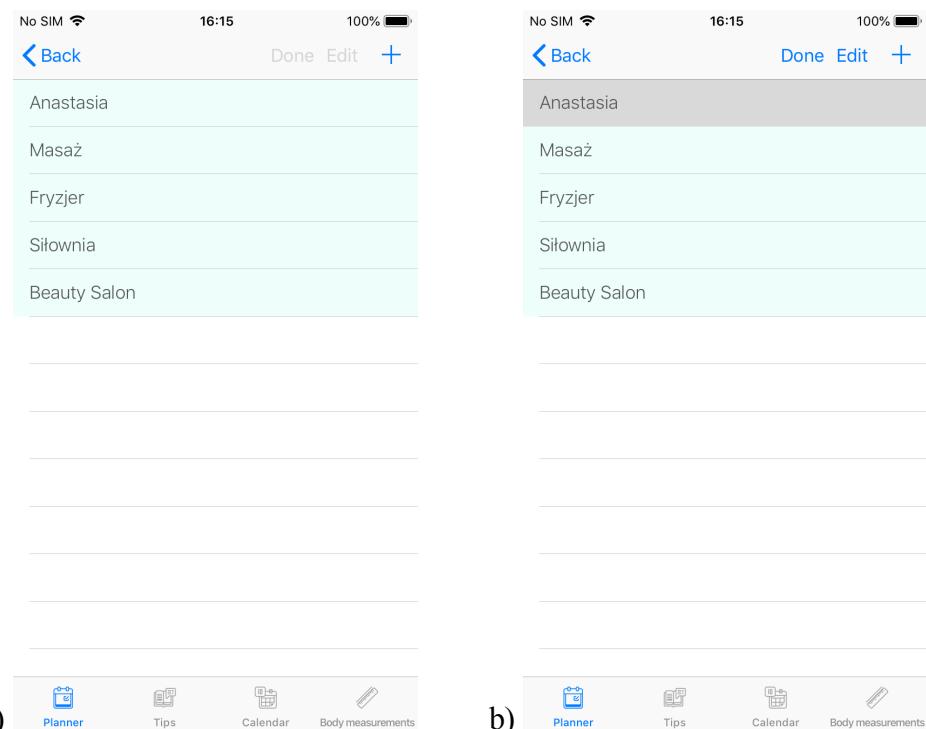
Kod 5 - Realizacja wyboru szablonu wiadomości SMS

## 6.7. Wyświetlanie listy kontaktów

Po przejściu z głównej strony zakładki „Planner”, do widoku edycji przy polu z kontaktem znajduje się przycisk, który przekierowuje użytkownika do listy kontaktów aplikacji (*Rysunek 50*). Zakładka ta jest kolejną podstroną dla grupy „Planner”.

W górnym pasku nawigacyjnym umieszczono cztery przyciski, po lewej stronie znajduje się przycisk, który służy do powrotu na poprzednią stronę. Po prawej stronie przycisk w kształcie plusa, dzięki niemu można dodać kolejny kontakt do listy. Dwa pozostałe przyciski aktywują się w momencie kliknięcia na dany wiersz z kontaktem (*Rysunek 50 b)*). Przycisk o nazwie „Done” jest odpowiedzialny za wybranie kontaktu, zamknięcie aktualnego okna i przejście do poprzednio otwartego (nowa wizyta lub jej edycja), a następnie umieszczenie numeru telefonu w polu odnośnie kontaktu. Ostatni przycisk „Edit” służy do edycji zaznaczonego kontaktu w nowym oknie.

Rysunek 50 - Widok z listą kontaktów w aplikacji



Źródło: Opracowanie własne.

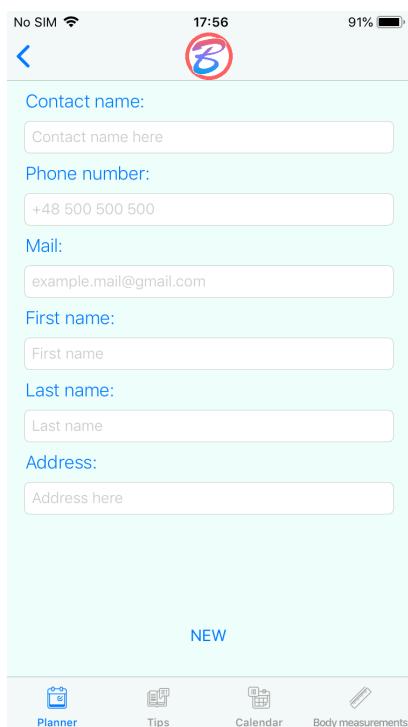
## 6.8. Dodawanie nowego kontaktu do listy

Poniższy rysunek (*Rysunek 51*) przedstawia szablon dla dodania nowego kontaktu. W tym oknie znajdują się podpisane pola tekstowe (nazwa kontaktu, numer telefonu, e-mail, imię, nazwisko i adres).

Aby użytkownik mógł dodać wpis do listy musi koniecznie wypełnić dwa pola, które są wymagane, takie jak nazwa kontaktu oraz numer telefonu (*Rysunek 52 b*).

Dla poprawnego zapisu wpisu do bazy danych, stworzono zabezpieczenia przed wpisywaniem przez użytkownika innych znaków niż te które są zalecane. Jeśli użytkownik się pomyli lub przypadkowo wpisze niechciany znak, nie będzie mógł zapisać kontaktu i dostanie powiadomienie typu toast na dole okna w kolorze czerwonym, które zawiera informację w jakiego zakresu znaków, liter bądź cyfr można użyć (*Rysunek 52 a*).

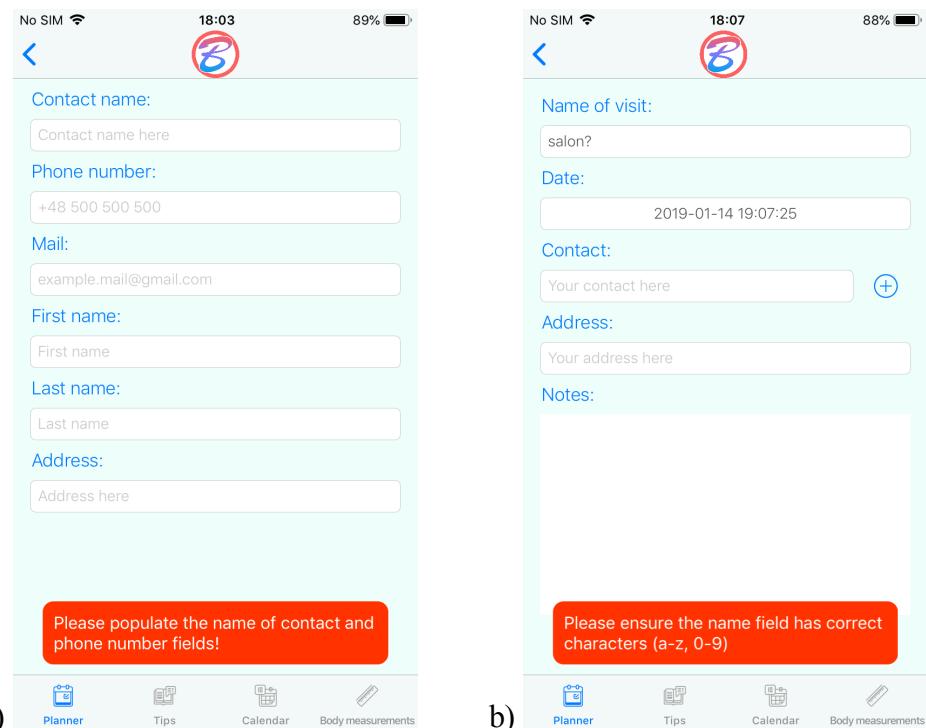
Rysunek 51 - Widok z dodaniem nowego kontaktu do listy



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 6*) przedstawia funkcję, która realizuje dodanie nowego kontaktu do bazy danych. Wzbogacony o sprawdzenie czy pola z nazwą kontaktu i numerem telefonu nie są puste.

Rysunek 52 - Widok z walidacją pól tekstowych w dodawaniu nowego kontaktu



Źródło: Opracowanie własne.

```
@IBAction func createNewContact(_ sender: Any) {
    style.backgroundColor = UIColor(red: 255.0/255.0, green: 50.0/255.0,
blue: 0.0/255.0, alpha: 1.0)
    style.messageColor = .white
    let finalSet = CharacterSet.letters.union(alphabetRule)
    if (contactNameTextField.text != "" && contactTextField.text != "") {
        if (finalSet.isSuperset(of: CharacterSet(charactersIn:
contactNameTextField.text!)) == true) {
            _ = ContactsCoreData().addContact(
                addressValue: addressTextField.text!,
                contactValue: contactTextField.text!,
                contactNameValue: contactNameTextField.text!,
                mailValue: mailTextField.text!,
                firstNameValue: firstNameTextField.text!,
                lastNameValue: lastNameTextField.text!)
            navigateToContactsListView()
        } else {
            self.view.makeToast("Please ensure the name of contact field has correct
characters (a-z, 0-9)", duration: 3.0, position: .bottom, style: style)
        }
    } else {
        self.view.makeToast("Please populate the name of contact and
phone number fields!", duration: 3.0, position: .bottom, style: style)
    }
}
```

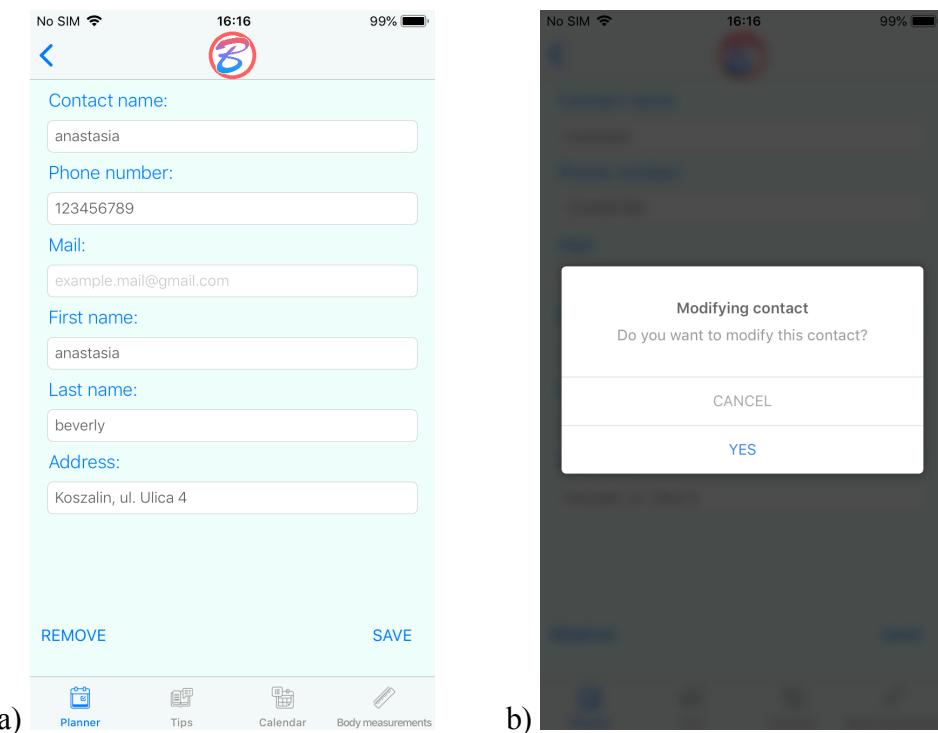
Kod 6 - Dodanie nowego kontaktu do bazy danych

## 6.9. Edycja istniejącego kontaktu

Poniższy rysunek przedstawia edycję kontaktu (*Rysunek 53 a*). W tym widoku znajdują się podpisane pola tekstowe (nazwa kontaktu, numer telefonu, e-mail, imię, nazwisko i adres), niektóre już wypełnione przez użytkownika przy dodaniu kontaktu.

Na samym dole strony zostały umieszczone dwa przyciski (usuń i zapisz). Po naciśnięciu przycisku o nazwie „REMOVE” zostaje wyświetlone okno dialogowe dla potwierdzenia usunięcia wpisu, jeśli zostanie to potwierdzone to kontakt zostaje usunięty z bazy danych po numerze ID. Przycisk o nazwie „SAVE” nadpisuje dany wpis (*Rysunek 53 b*).

Rysunek 53 - Widok z edycją istniejącego kontaktu



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 7*) zostaje uruchomiony po kliknięciu w przycisk „SAVE” i pobiera wybrane wartości z wymienionych niżej pól, a następnie zapisuje je w bazie danych.

```
ContactsCoreData().modifyContact(id: self.currentContactID,  
                                 addressValue: self.addressTextField.text!,  
                                 contactValue: self.contactTextField.text!,  
                                 contactNameValue: self.contactNameTextField.text!,  
                                 mailValue: self.mailTextField.text!,  
                                 firstNameValue: self.firstNameTextField.text!,  
                                 lastNameValue: self.lastNameTextField.text!)  
self.navigateToContactsListView()
```

Kod 7 - Uruchomienie kodu po przyciśnięciu przycisku "SAVE" i zapis do danych

Poniższy fragment kodu (*Kod 8*) szuka wpisu po numerze ID, następnie podmienia wartości pól (address, contact, contactName, mail, firstName i lastName) w encji „ContactEntity” i zapisuje je w bazie danych.

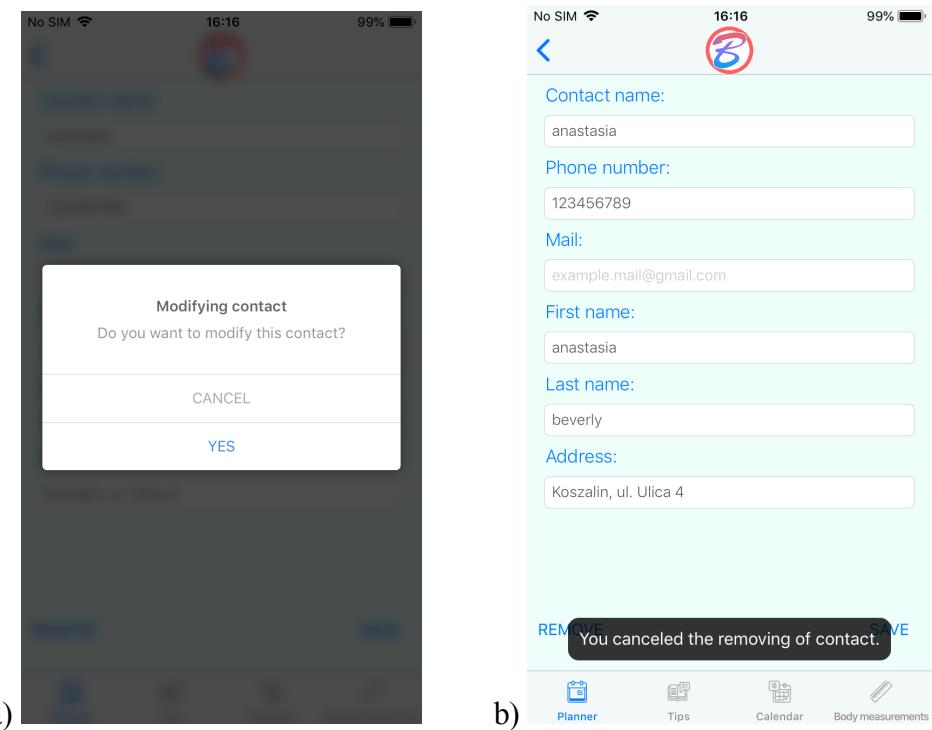
```
func modifyContact(id: Int64, addressValue: String, contactValue: String,  
contactNameValue: String, mailValue: String, firstNameValue: String,  
lastNameValue: String) {  
    let appDelegate = UIApplication.shared.delegate as! AppDelegate  
    let context = appDelegate.persistentContainer.viewContext  
    let request = NSFetchedRequest<NSFetchRequestResult>(entityName:  
"ContactsEntity")  
    request.returnsObjectsAsFaults = false  
    let predicate = NSPredicate(format: "id == \(\(id)")")  
    request.predicate = predicate  
    let result = try? context.fetch(request)  
    let resultData = result as! [ContactsEntity]  
    if resultData.count != 0 {  
        resultData[0].setValue(addressValue, forKey: "address")  
        resultData[0].setValue(contactValue, forKey: "contact")  
        resultData[0].setValue(contactNameValue, forKey: "contactName")  
        resultData[0].setValue(mailValue, forKey: "mail")  
        resultData[0].setValue(firstNameValue, forKey: "firstName")  
        resultData[0].setValue(lastNameValue, forKey: "lastName")  
        do {  
            try context.save()  
        } catch let error as NSError {  
            print("Could not save \(error), \(error.userInfo)")  
        }  
    }  
}
```

Kod 8 - Modyfikacja kontaktu i zapis w bazie danych

## 6.10. Usunięcie istniejącego kontaktu

Po przyciśnięciu w przycisk o nazwie „REMOVE” ukazuje się okno dialogowe typu PopupDialog do podjęcia akcji usunięcia danego wpisu kontaktu. Na jednym z rysunków (*Rysunek 54 a)*) został pokazany przykład dla anulowania usunięcia kontaktu. Po anulowaniu okno dialogowe znika i zostaje wyświetlona informacja typu Toast na dole strony w kolorze czarnym (*Rysunek 54 b)*) potwierdzająca, że użytkownik anulował usunięcie danego kontaktu. Jeśli natomiast wizyta została usunięta widok w aplikacji zostaje przekierowany do poprzedniej strony, w tym przypadku z listą kontaktów.

Rysunek 54 - Widok z usunięciem danego kontaktu



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 9*) przedstawia wywołanie funkcji usuwającej kontakt z bazy danych po konkretnym ID.

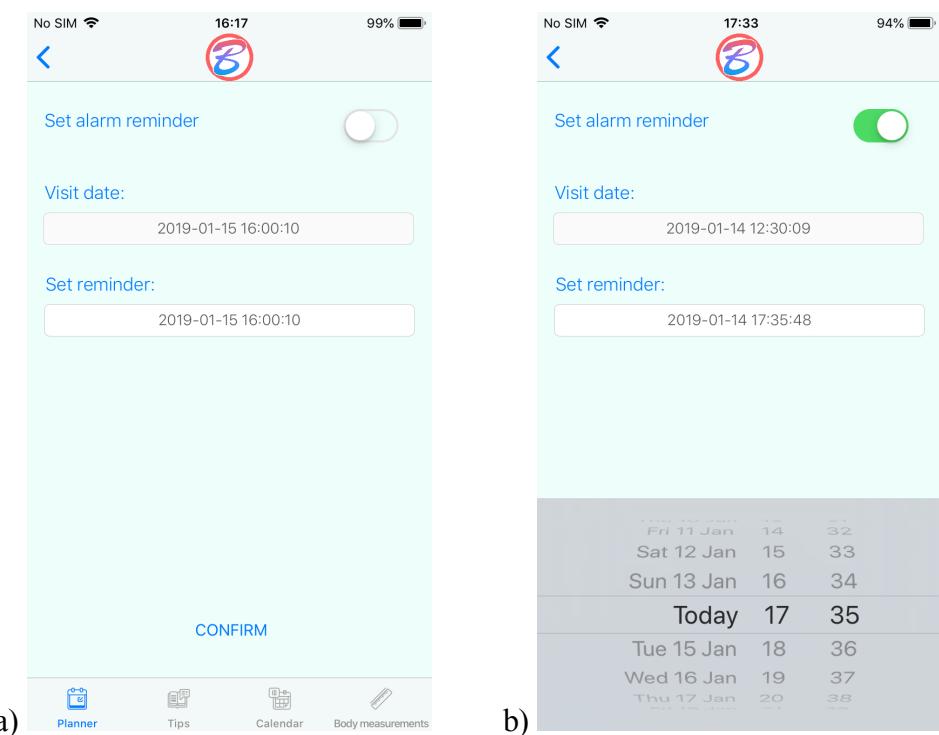
```
ContactsCoreData().removeContact(id: self.currentContactID)
```

Kod 9 - Wywołanie funkcji usuwającej kontakt z bazy danych

## 6.11. Ustawienia powiadomień o zbliżającym się terminie

Poniższe rysunki (*Rysunek 55 a) i b)*) przedstawiają okno z ustawieniem przypomnienia dla konkretnej wizyty. Przyciskiem typu switch użytkownik uruchamia alert dla przypomnienia bądź go wyłącza. W polu „Visit date.” jest umieszczona data wizyty i jej godzina, to miejsce nie jest edytowalne w tym miejscu. Poniżej znajduje się pole o nazwie „Set reminder:”, w którym użytkownik może ustawić poprzez DatePicker dla jakiego dnia i jakiej godziny, powiadomienie przypominające o danej wizycie ma się uruchomić.

Rysunek 55 - Widok przedstawiający ustawienie powiadomienia



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 10*) sprawdza czy przełącznik jest włączony czy też nie, jeśli jest włączony to zamyka okno i wraca do poprzedniego widoku. Przesyłając przy tym informacje o tym, że powiadomienie ma zostać włączone.

```
@IBAction func confirmReminder(_ sender: Any) {
    if (switchReminder.isOn == true) {
        if (reminderTextField.text != "") {
            isReminderEnabled = switchReminder.isOn
            performSegue(withIdentifier: "CloseReminderWindow", sender: self)
        } else {
            self.view.makeToast("Choose the date if you want turn ON reminder.",
                duration: 3.0, position: .bottom, style: style)
        }
    } else {
        isReminderEnabled = switchReminder.isOn
        performSegue(withIdentifier: "CloseReminderWindow", sender: self)
    }
}
```

Kod 10 - Włączenie lub wyłączenie powiadomienia

Poniższe rysunki przedstawiają, jak wygląda alert dla wydarzenia na smartfonie. Na zablokowanej stronie telefonu (*Rysunek 56 a)*), wyświetlany jest tytuł powiadomienia (nazwa wizyty), data i godzina, a na końcu adres, jeśli jest załączony. Na kolejnym rysunku (*Rysunek 56 b)*), przedstawiono widok z ikoną aplikacji i aktywnym przypomnieniem.

Rysunek 56 - Zrzut ekranu form powiadomień na telefonie

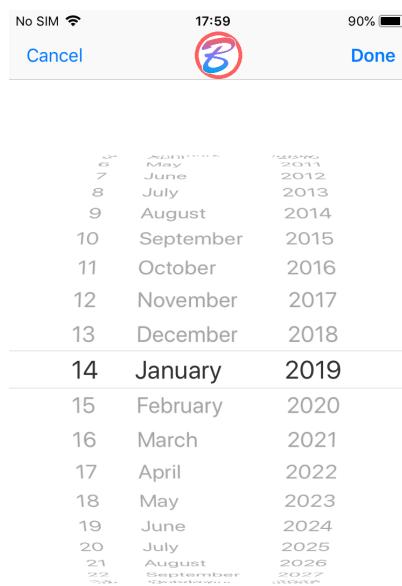


Źródło: *Opracowanie własne*.

## 6.12. Ustawienia i przegląd kalendarza

Poniższy rysunek (*Rysunek 57*) przedstawia ustawienie dla kalendarza, odpowiedzialne za zmianę daty w widoku. Wybór daty jest bardzo łatwy w obsłudze, dzięki przewijanej liście. Każda kolumna (dni, miesiące, lata) jest przewijana osobno.

Rysunek 57 - Widok ustawień kalendarza dla zmiany daty

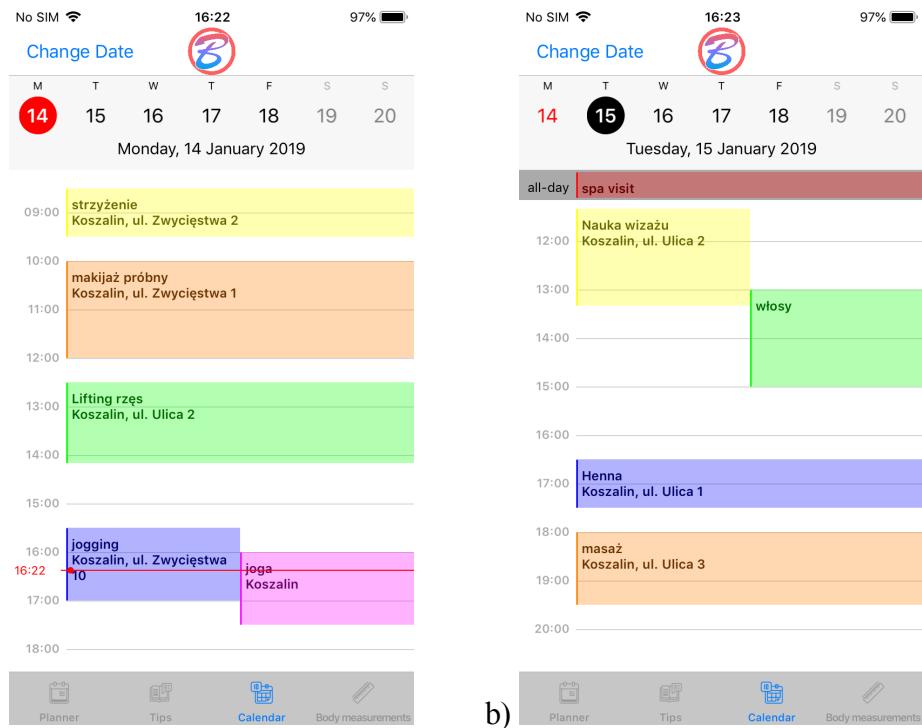


Źródło: Opracowanie własne.

Na kolejnym widoku (*Rysunek 58 a*) przedstawiono przykładowe wypełnienie kalendarza zaplanowanymi wydarzeniami. Dzięki zróżnicowanej kolorystyce łatwo rozróżnić dane bloki i ich przedział czasowy, użytkownik może sobie skategoryzować różne wpisy poprzez kolory. W aktualnym dniu na liście dniowej kalendarza czerwoną linią oznaczono bieżącą godzinę.

Natomiast w górnym pasku kalendarza znajduje się listwa czasowa, wyświetlająca dni tygodnia, a także nazwę konkretnego dnia, dzień, miesiąc i rok. Aktualny dzień zaznaczony jest na tym pasku kółkiem koloru czerwonego, pozostałe dni po przyciśnięciu są oznaczone kółkiem w kolorze czarnym (*Rysunek 58 b*). Wizyta całodniowa jest oznaczana pod listwą czasową kalendarza o nazwie „all-day”, bez względu czy użytkownik przewinie widok dzienny w dół czy też górę, wydarzenie całodniowe zawsze pozostanie w tym samym miejscu, dobrze widoczne.

Rysunek 58 - Widok kalendarza w aplikacji



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 11*) wczytuje nowe wydarzenia czy nic się nie zmieniło w bazie danych. Konfiguruje każde zdarzenie, wizytę nadając im odpowiedni kolor, czas trwania, tekst. Sprawdza, kiedy się zaczyna i kończy wydarzenie, a także sprawdza wydarzenia czy mają włączony tryb dla całodniowego bloku w kalendarzu.

Poniższy rysunek (*Rysunek 59*) przedstawia okno dialogowe dla konkretnej wizyty. Można je uruchomić poprzez przyciśnięcie i dłuższe przytrzymanie na wybrany blok z wydarzeniem w kalendarzu.

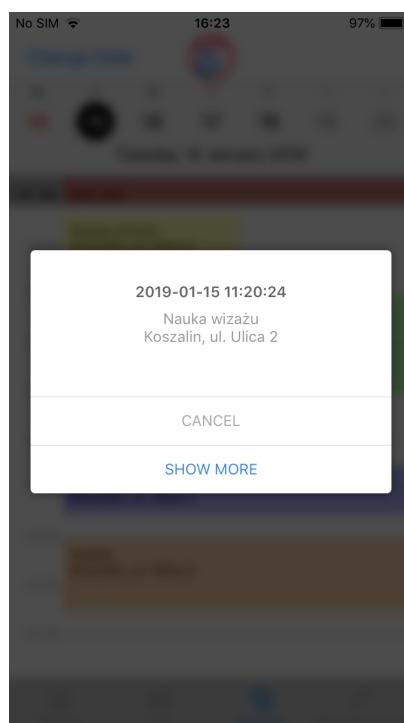
```

var colors = [UIColor.blue, UIColor.yellow, UIColor.green, UIColor.red,
UIColor.orange, UIColor.magenta]
override func eventsForDate(_ date: Date) -> [EventDescriptor] {
    visitsArray.removeAll()
    dateArray.removeAll()
    loadCalendarDataFromDB()
    var events = [Event]()
    for i in 0...visitsArray.count-1 {
        let event = Event()
        var visitDate = dateArray[i]
        let duration: Int = Int(appointments[i].duration)!
        var chunk = TimeChunk.dateComponents(minutes: duration)
        let datePeriod = TimePeriod(beginning: visitDate, chunk: chunk)
        event.startDate = datePeriod.beginning!
        event.endDate = datePeriod.end!
        var info = visitsArray[i]
        event.text = info.reduce("", {$0 + $1 + "\n"})
        event.color = colors[Int(appointments[i].colorNumber)]
        event.isAllDay = appointments[i].isAllDay
        events.append(event)
        event.userInfo = Int(appointments[i].id) }
    return events
}

```

Kod 11 - Przygotowanie danych dla kalendarza

Rysunek 59 - Widok podglądu wydarzenia w kalendarzu



*Źródło: Opracowanie własne.*

Poniższy fragment kodu (*Kod 12*) jest włączony poprzez długie przytrzymanie bloku wydarzenia w kalendarzu, a w wyniku tego uruchamia okno dialogowe typu PopupDialog. Dzięki tej funkcji wyświetlane są następujące informacje:

- Data wydarzenia wraz z godziną
- Tytuł wizyty
- Adres (zostanie pobrany i wyświetlony, tylko wtedy kiedy użytkownik wypełni to pole)
- Przycisk „CANCEL”, anulujący wyświetlanie tej informacji
- Przycisk „SHOW MORE”, który przenosi użytkownika do widoku z listą wszystkich wydarzeń

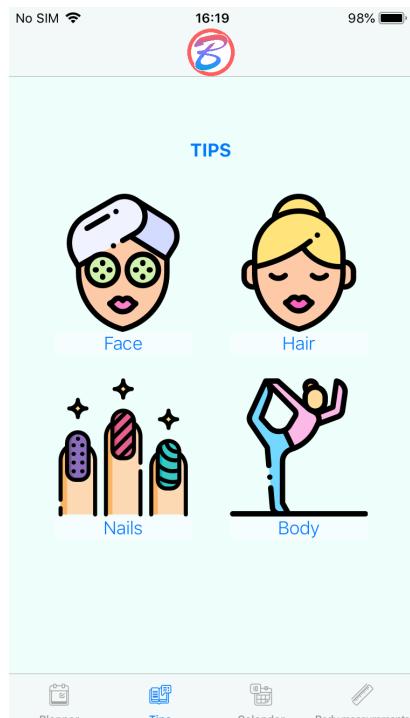
```
override func dayViewDidLongPressEvent(_ eventView: EventView) {  
    guard let descriptor = eventView.descriptor as? Event else {  
        return  
    }  
    dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss"  
    let dateFromString = dateFormatter.string(from:  
descriptor.startDate)  
    let mainTitle = "\(dateFromString)"  
    let question = descriptor.text  
    let popupDialog = PopupDialog(title: mainTitle, message: question)  
  
    let cancelButton = CancelButton(title: "CANCEL") {  
    }  
    let yesButton = DefaultButton(title: "SHOW MORE",  
dismissOnTap: true) {  
        self.tabBarController?.selectedIndex = 0  
    }  
    popupDialog.addButton([cancelButton, yesButton])  
    self.present(popupDialog, animated: true, completion: nil)  
}
```

Kod 12 - Wyświetlenie okna dialogowego ze szczegółową informacją dla wydarzenia w kalendarzu

### 6.13. Przeglądanie porad po wybranej kategorii

Następna strona dotyczy różnego rodzaju porad. Na załączonym rysunku (*Rysunek 60*) przedstawiono cztery ikony, które są przyciskami do poszczególnych kategorii z poradami („Face”, „Hair”, „Nails”, „Body”).

Rysunek 60 - Widok głównej strony porad z ich kategoriami



Źródło: Opracowanie własne.

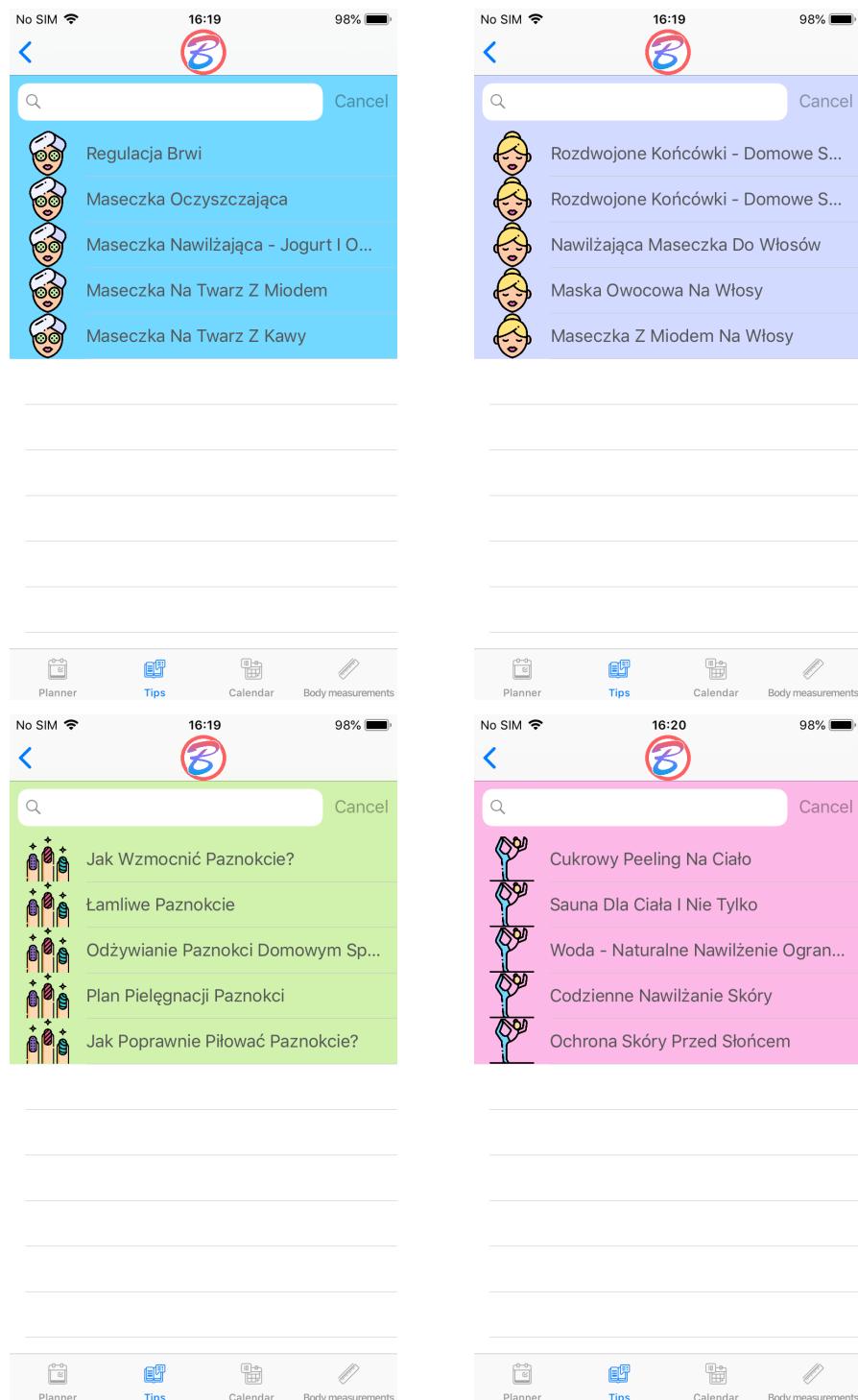
Poniższy fragment kodu (*Kod 13*) przedstawia sprawdzenie połączenia segoe, które zostało kliknięte i odpowiednio przesyła do następnego widoku nazwę wybranej kategorii.

```
if segue.identifier == "Face" {
    if let vc = segue.destination as? TipsTableVC {
        vc.selectedCategory = "Face"
    } else if segue.identifier == "Hair" {
        if let vc = segue.destination as? TipsTableVC {
            vc.selectedCategory = "Hair"
        }
    } else if segue.identifier == "Nails" {
        if let vc = segue.destination as? TipsTableVC {
            vc.selectedCategory = "Nails"
        }
    } else if segue.identifier == "Body" {
        if let vc = segue.destination as? TipsTableVC {
            vc.selectedCategory = "Body"
        }
    }
}
```

Kod 13 - Sprawdzenie, które połączenie (segue) zostało wywołane

Rysunek (Rysunek 61) poniżej przedstawia listę z poradami dla czterech kategorii. Każdej z nich przypisano wersję kolorystyczną oraz odpowiednie ikony dla łatwiejszego rozróżnienia w jakiej sekcji znajduje się użytkownik.

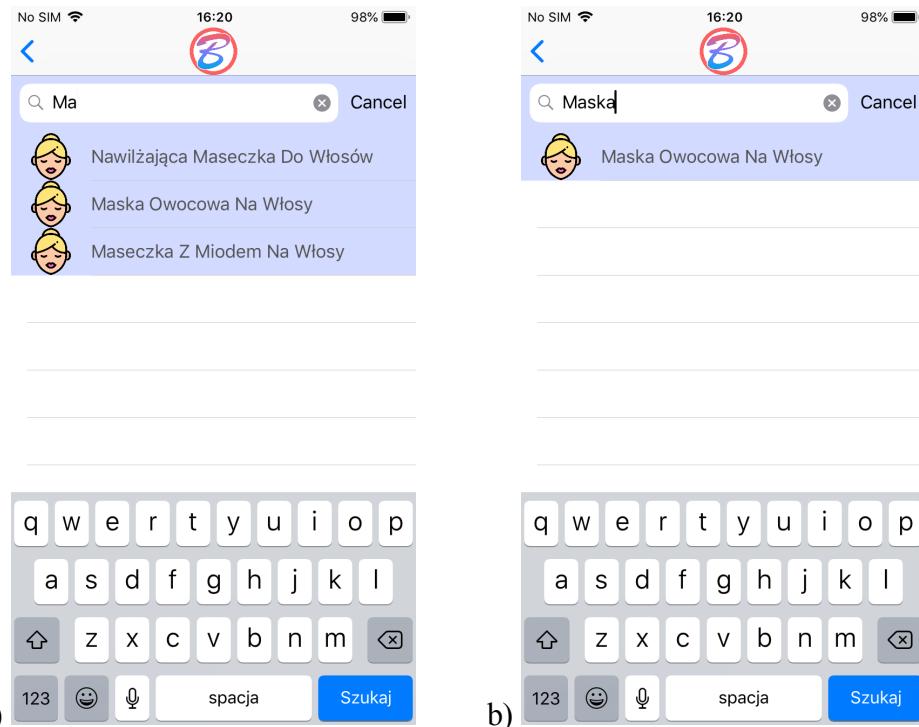
Rysunek 61 - Widok czterech kategorii porad



Źródło: Opracowanie własne.

Dodatkowo nad całą listą z poradami znajduje się miejsce do wpisania szukanej rady. Dana fraza jest wyszukiwana w całym tytule porady (*Rysunek 61 a) i b)*), dzięki temu użytkownik łatwiej może znaleźć interesujący go wpis.

Rysunek 62 - Widok opcji wyszukiwania porady po wpisanym haśle w wyszukiwarce



Źródło: *Opracowanie własne.*

Poniższy fragment kodu (*Kod 14*) przedstawia mechanizm, który pokazuje, jak dokonano wyszukiwania po wpisanych słowach kluczowych. Najpierw zostaje przefiltrowana tablica z poradami tylko odnośnie ich tytułów, a następnie sprawdza czy fraza tekstowa tam się znajduje. Wynik filtra zostaje zapisany do dodatkowej tablicy, której zawartość jest wyświetlona w tabelce. W przypadku kliknięcia na przycisk „Cancel” ponownie wczytuje się oryginalna tablica z poradami, a w pasku wyszukiwania czyszczone jest pole tekstowe.

```

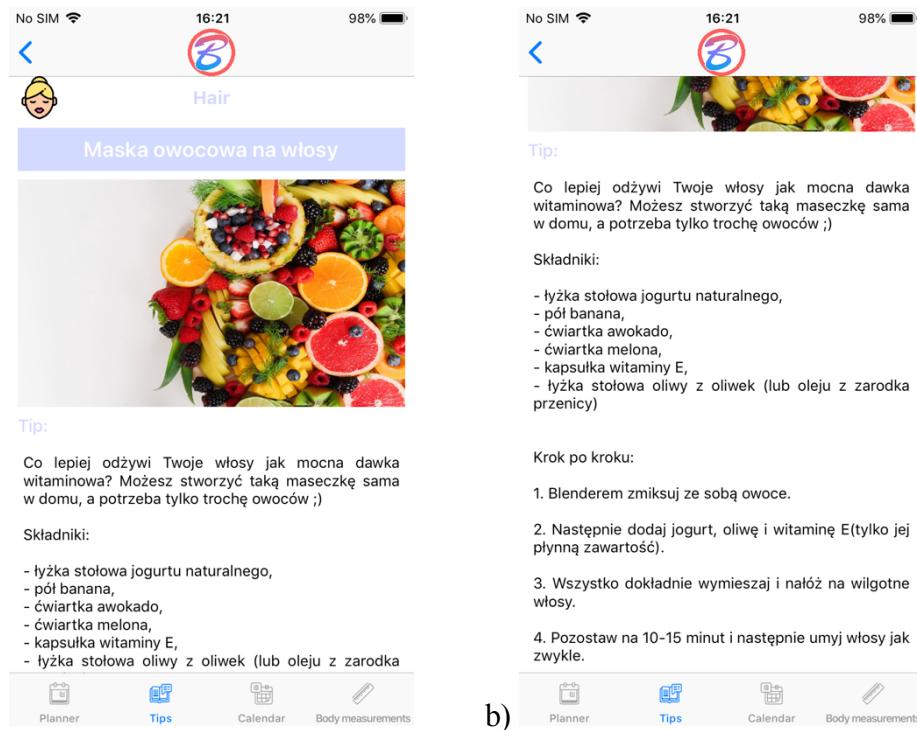
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String){
    if (searchBar.text != "") { filteredTipsArray =
tips.filter({$0.title.lowercased().contains(searchBar.text!.lowercased())})
        isSearch = true
    } else {
        filteredTipsArray = tips
        isSearch = false}
    tableView.reloadData() }
func searchBarCancelButtonClicked(_ searchBar: UISearchBar) {
    isSearch = false
    searchBar.text = ""
    tableView.reloadData()
}

```

Kod 14 - Wyszukiwanie po wpisanych słowach kluczowych

Na stronie z wyświetlением konkretnej porady (*Rysunek 63 a) i b)*) zaraz pod górnym paskiem nawigacyjnym została umieszczona ikona dla danej kategorii w lewym rogu, natomiast na środku zlokalizowano jej główny tytuł, który jest w tej samej kolorystyce gatunkowej. W wierszu poniżej osadzono tytuł dla porady, tutaj z kolei napis jest biały, a ramka kolorystycznie odpowiada przypisanej kategorii. Umieszczone zdjęcie zostało pobrane ze strony z darmowymi grafikami, aby urozmaicić kolorystykę aplikacji. Pod nim znajduje się docelowa porada. Cała strona jest przewijana.

Rysunek 63 - Widok przykładowej porady



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 15*) przedstawia wyświetlanie porad, wraz z rozróżnieniem dla jakiej wersji kolorystycznej są przypisane.

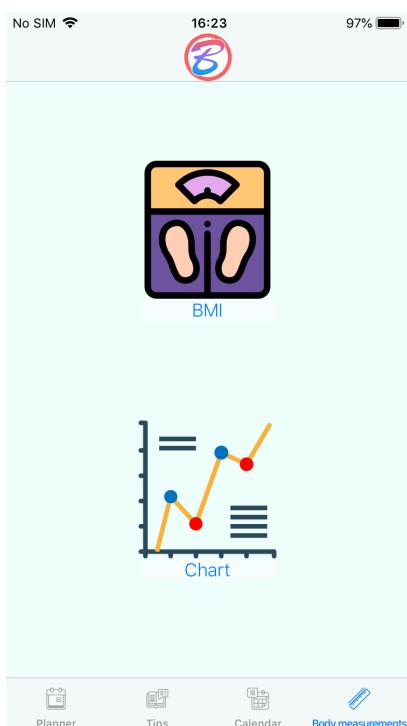
```
let file = tipDetailsArray!.tip
let bundlePath = Bundle.main.path(forResource: "TipsAssets", ofType:
"bundle")
let resourceBundle = Bundle.init(path: bundlePath!)
if let filepath = resourceBundle!.path(forResource: file, ofType: "txt") {
    do {
        let zm = try String(contentsOfFile: filepath, encoding: .utf8)
        textTipView.text = zm
    } catch {
        print("Error")
    } else {
        print("File not found")
    }
    if(selectedCategory == "Face"){
        // blue
        icon = UIImage(named: "face")
        cellColor = UIColor.init(red: 114.0/255.0, green: 216.0/255.0,
blue: 255.0/255.0, alpha: 1.0)
    } else if(selectedCategory == "Hair"){
        icon = UIImage(named: "hair")
        // violet
        cellColor = UIColor.init(red: 211.0/255.0, green: 218.0/255.0,
blue: 255.0/255.0, alpha: 1.0)
    } else if(selectedCategory == "Nails"){
        icon = UIImage(named: "nails")
        // green
        cellColor = UIColor.init(red: 208.0/255.0, green: 242.0/255.0,
blue: 173.0/255.0, alpha: 1.0)
    } else if(selectedCategory == "Body"){
        icon = UIImage(named: "body2")
        // pink
        cellColor = UIColor.init(red: 252.0/255.0, green: 184.0/255.0,
blue: 231.0/255.0, alpha: 1.0)
    }
    iconImageView.image = icon
    categoryLabel.backgroundColor = UIColor.clear
    categoryLabel.textColor = cellColor
    titleTipLabel.backgroundColor = cellColor
    titleTipLabel.textColor = UIColor.white
    tipLabel.textColor = cellColor
```

Kod 15 - Wyświetlenie porad dla wybranej kategorii w wersji kolorystycznej

## 6.14. Wyliczanie indeksu BMI

Ten widok okna przedstawia główną stronę o nazwie „Body measurements” (Rysunek 64). Znajdują się tutaj dwie ikony, które są odpowiedzialne za przejście do kolejnych podstron. Każda z nich jest podpisana, aby ułatwić użytkownikowi rozróżnienie działów. Po kliknięciu w pierwszą ikonę ukazuje się okno z kalkulatorem do wyliczenia indeksu BMI, natomiast po przyciśnięciu w drugą ikonę, użytkownik zostaje przekierowany na podstronę odnoszącą się do wyrysowania wykresów.

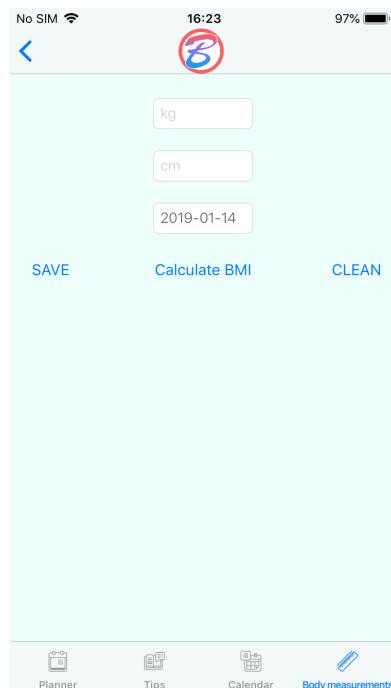
Rysunek 64 - Widok głównej strony pomiarów ciała



Źródło: Opracowanie własne.

Poniższy rysunek (Rysunek 65) przedstawia stronę z kalkulatorem do wyliczenia indeksu BMI. Umieszczone są tutaj dwa pola tekstowe w których użytkownik podaje swoją wagę w kilogramach, poniżej podaje swój wzrost w centymetrach, następnie może przycisnąć przycisk „Calculate BMI”, aby wyliczyć i sprawdzić jaki ma indeks BMI. Jeśli chce zapisać ten wynik, to z trzeciej tabelki, w której znajduje się data, musi ją wybrać lub pozostawić aktualną i otrzymany rezultat zostanie zapisany do bazy danych razem z wybraną datą, po przyciśnięciu przycisku „SAVE”. Trzeci przycisk „CLEAN” służy do wyczyszczenia całej bazy danych z wcześniejszymi zapisanymi pomiarami BMI.

Rysunek 65 - Widok kalkulatora BMI



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 16*) przedstawia algorytm do wyliczenia indeksu BMI oraz wyświetla obliczoną wartość.

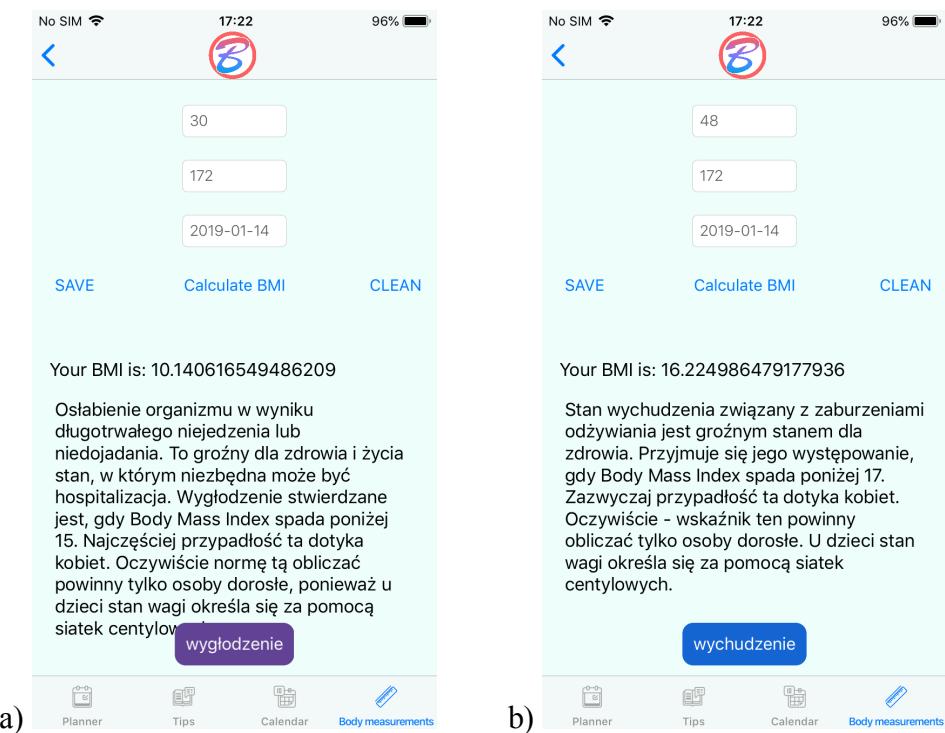
```
kgValue = Double(kgTextField.text!)!
let cmValue = Double(cmTextField.text!)
let mValue = Double(cmValue!/100.0)
resultBmi = Double(kgValue / (mValue * mValue))
scoreBMILabel.text = "Your BMI is: \(resultBmi)"
var result = "toast"
var style = ToastStyle()
```

Kod 16 - Algorytm wyliczający indeks BMI

## 6.15. Prezentacja przedziałów klasyfikacji BMI

Poniższe rysunki (*Rysunek 66 a) i b)*) przedstawiają przypadki testowe (konkretnie dane dla wagi i wzrostu), które wywołują opisy dla dwóch kategorii. Równocześnie wyświetla się wiadomość typu Toast z kategorią, do której przynależy wyliczony indeks BMI – „wygłodzenie” w kolorze fioletowym oraz „wychudzenie” w kolorze niebieskim.

Rysunek 66 - Widok wyliczonych wartości BMI (wygłodzenie i wychudzenie)



Źródło: Opracowanie własne.

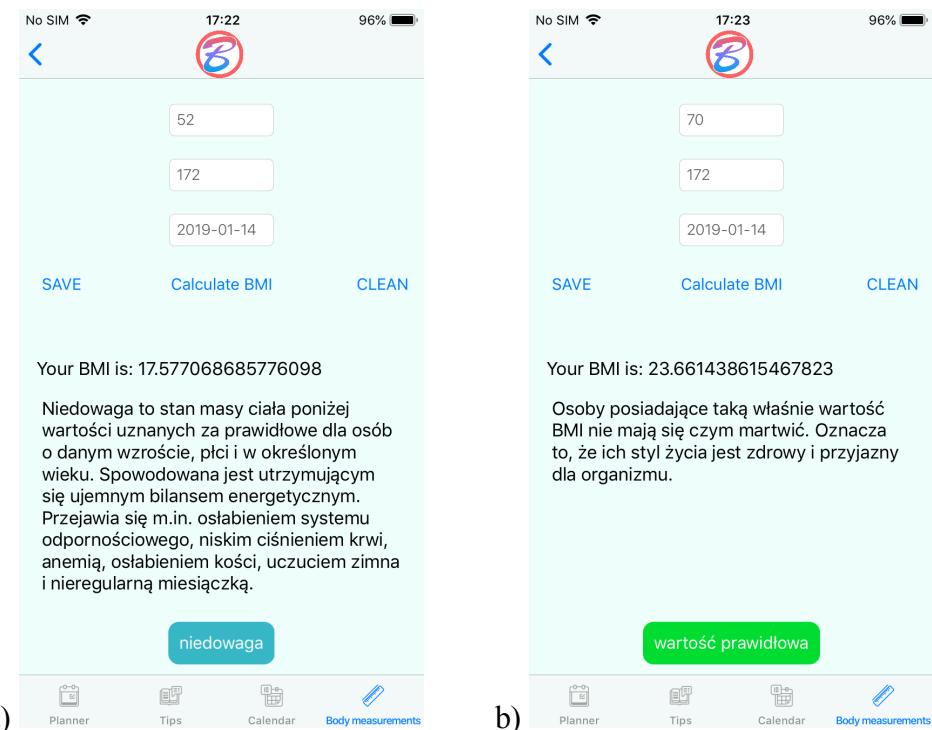
W poniższym fragmencie kodu (*Kod 17*) przedstawiono przedziały wartości dla powyższych dwóch kategorii. „Wygłodzenie” to przedział wartości poniżej 16.0, a „wychudzenie” to indeks wyższy niż 16.0 i jednocześnie niższy niż 16.99.

```
if (resultBmi < 16.0) {
    result = "wygłodzenie"
    style.backgroundColor = UIColor(red: 97.0/255.0, green:
66.0/255.0, blue: 148.0/255.0, alpha: 1.0)
} else if (resultBmi > 16.0 && resultBmi < 16.99) {
    result = "wychudzenie"
    style.backgroundColor = UIColor(red: 20.0/255.0, green:
100.0/255.0, blue: 210.0/255.0, alpha: 1.0) }
```

Kod 17 - Przedział wartości dla dwóch kategorii (wygłodzenie i wychudzenie)

Poniższe rysunki (*Rysunek 67 a) i b)*) przedstawiają przypadki testowe (konkretnie dane dla wagi i wzrostu), które wywołują opisy dla dwóch kategorii. Równocześnie wyświetla się wiadomość typu Toast z kategorią, do której przynależy wyliczony indeks BMI – „niedowaga” w kolorze turkusowym oraz „wartość prawidłowa” w kolorze zielonym.

Rysunek 67 - Widok wyliczonych wartości BMI (niedowaga i wartość prawidłowa)



Źródło: *Opracowanie własne.*

W poniższym fragmencie kodu (*Kod 18*) przedstawiono przedziały wartości dla powyższych dwóch kategorii. „Niedowaga” to przedział wartości wyższy niż 17.0 i jednocześnie niższy niż 18.49, a „wartość prawidłowa” to indeks wyższy niż 18.50 i jednocześnie niższy niż 24.99.

```

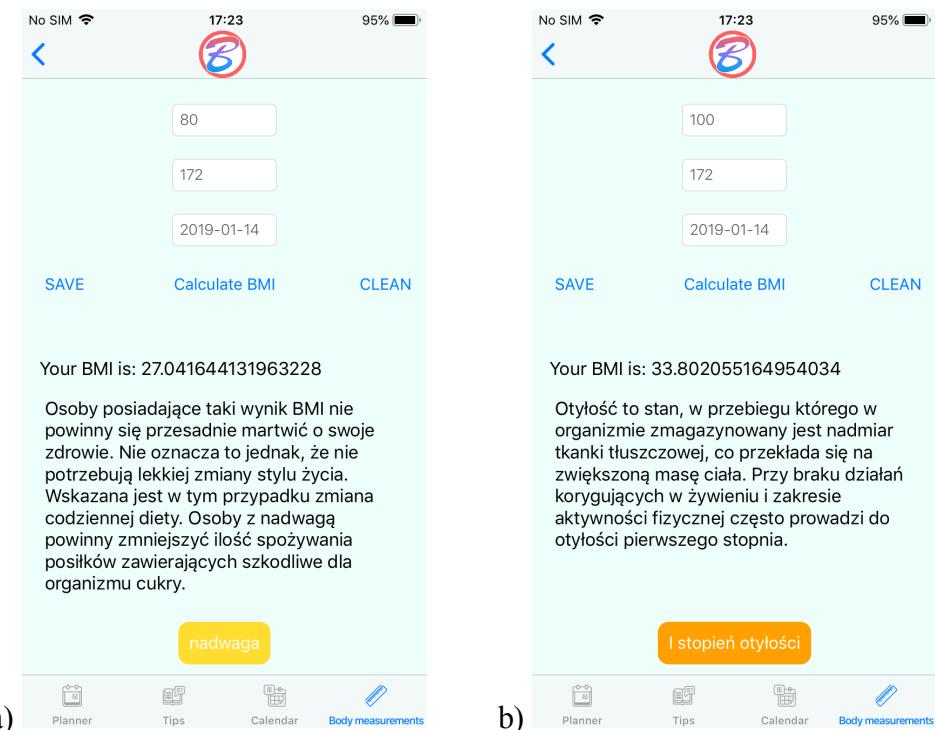
} else if (resultBmi > 17.0 && resultBmi < 18.49) {
    result = "niedowaga"
    style.backgroundColor = UIColor(red: 55.0/255.0, green:
182.0/255.0, blue: 197.0/255.0, alpha: 1.0)
} else if (resultBmi > 18.50 && resultBmi < 24.99) {
    result = "wartość prawidłowa"
    style.backgroundColor = UIColor(red: 0.0/255.0, green:
221.0/255.0, blue: 48.0/255.0, alpha: 1.0)
}

```

Kod 18 - Przedział wartości dla dwóch kategorii (niedowaga i wartość prawidłowa)

Poniższe rysunki (*Rysunek 68 a) i b)*) przedstawiają przypadki testowe (konkretnie dane dla wagi i wzrostu), które wywołują opisy dla dwóch kategorii. Równocześnie wyświetla się wiadomość typu Toast z kategorią, do której przynależy wyliczony indeks BMI – „nadwaga” w kolorze żółtym oraz „I stopień otyłości” w kolorze pomarańczowym.

Rysunek 68 - Widok wyliczonych wartości BMI (nadwaga i I stopień otyłości)



*Źródło:* Opracowanie własne.

W poniższym fragmencie kodu (*Kod 19*) przedstawiono przedziały wartości dla powyższych dwóch kategorii. „Nadwaga” to przedział wartości wyższy niż 25.0 i jednocześnie niższy niż 29.99, a wartość „I stopień otyłości” to indeks wyższy niż 30.0 i jednocześnie niższy niż 34.99.

```

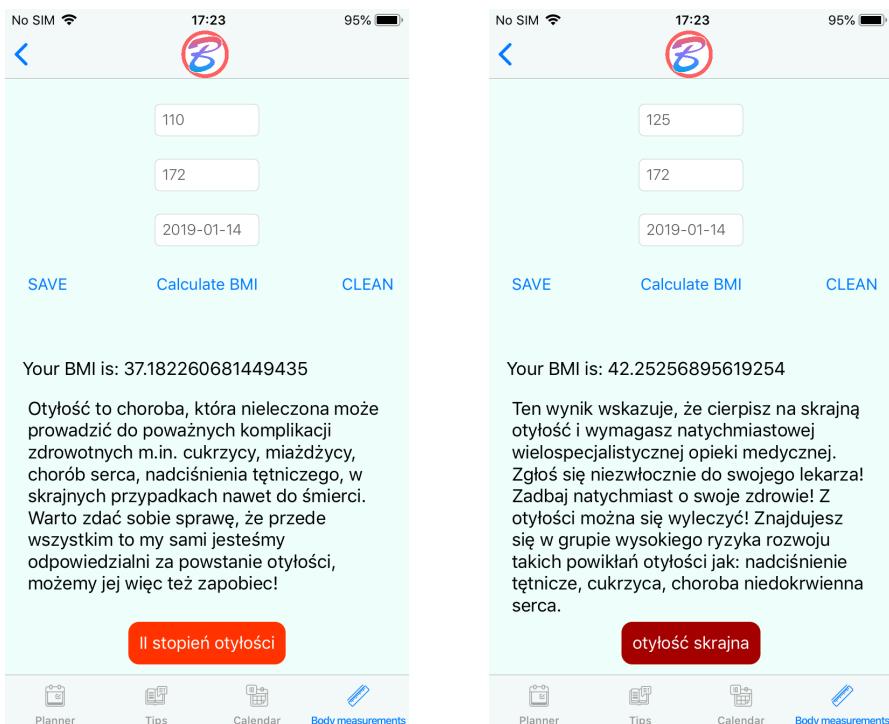
} else if (resultBmi > 25.0 && resultBmi < 29.99) {
    result = "nadwaga"
    style.backgroundColor = UIColor(red: 284.0/255.0,
green: 221.0/255.0, blue: 48.0/255.0, alpha: 1.0)
} else if (resultBmi > 30.0 && resultBmi < 34.99) {
    result = "I stopień otyłości"
    style.backgroundColor = UIColor(red: 255.0/255.0,
green: 160.0/255.0, blue: 0.0/255.0, alpha: 1.0)
}

```

Kod 19 - Przedział wartości dla dwóch kategorii (nadwaga i I stopień otyłości)

Poniższe rysunki (*Rysunek 69 a) i b)*) przedstawiają przypadki testowe (konkretnie dane dla wagi i wzrostu), które wywołują opisy dla dwóch kategorii. Równocześnie wyświetla się wiadomość typu Toast z kategorią, do której przynależy wyliczony indeks BMI – „II stopień otyłości” w kolorze czerwonym oraz „otyłość skrajna” w kolorze bordowym.

Rysunek 69 - Widok wyliczonych wartości BMI (II stopień otyłości i otyłość skrajna)



Źródło: Opracowanie własne.

W poniższym fragmencie kodu (*Kod 20*) przedstawiono przedziały wartości dla powyższych dwóch kategorii. „II stopień otyłości” to przedział wartości wyższy niż 35.0 i jednocześnie niższy niż 39.99, a wartość „otyłość skrajna” to indeks wyższy niż 40.0.

```

} else if (resultBmi > 35.0 && resultBmi < 39.99) {
    result = "II stopień otyłości"
    style.backgroundColor = UIColor(red: 255.0/255.0,
green: 50.0/255.0, blue: 0.0/255.0, alpha: 1.0)
} else if (resultBmi > 40.0) {
    result = "otyłość skrajna"
    style.backgroundColor = UIColor(red: 165.0/255.0,
green: 0.0/255.0, blue: 0.0/255.0, alpha: 1.0)
}

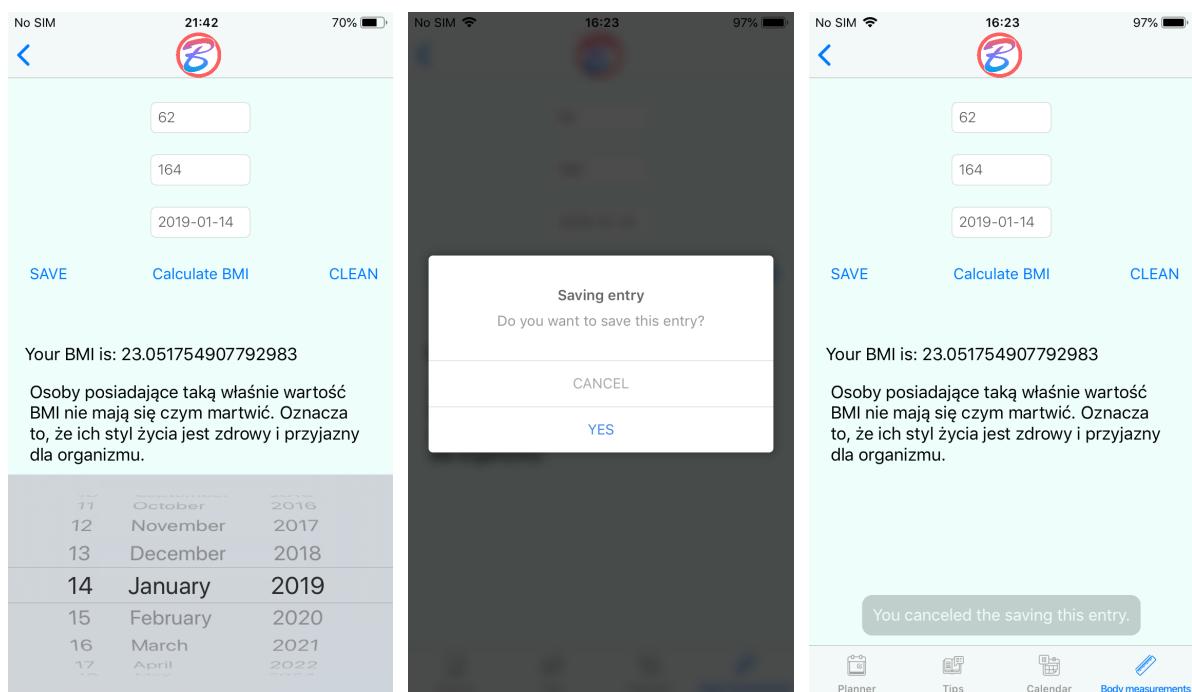
```

Kod 20 - Przedział wartości dla dwóch kategorii (II stopień otyłości i otyłość skrajna)

## 6.16. Zapisywanie wartości BMI z wybraną datą

Chcąc zapisać wyliczony indeks BMI, użytkownik musi wybrać datę z danego pola, a następnie przycisnąć przycisk „SAVE”. Po wykonaniu tych czynności pojawia się okno dialogowe typu PopupDialog z pytaniem potwierdzającym, czy na pewno chce się zapisać dany wpis. Przycisk „CANCEL” anuluje wykonanie czynności zapisującej i zamyka okno, tym samym wyświetlając na dole strony wiadomość typu Toast. Natomiast przycisk „SAVE” zapisuje wyliczone dane z datą do bazy danych (*Rysunek 70*).

Rysunek 70 - Widok z zapisem wartości BMI z wybraną datą



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 21*) przedstawia dodanie nowego wpisu do bazy danych, jeśli istnieje już w wpis z wybraną datą to go nadpisuje.

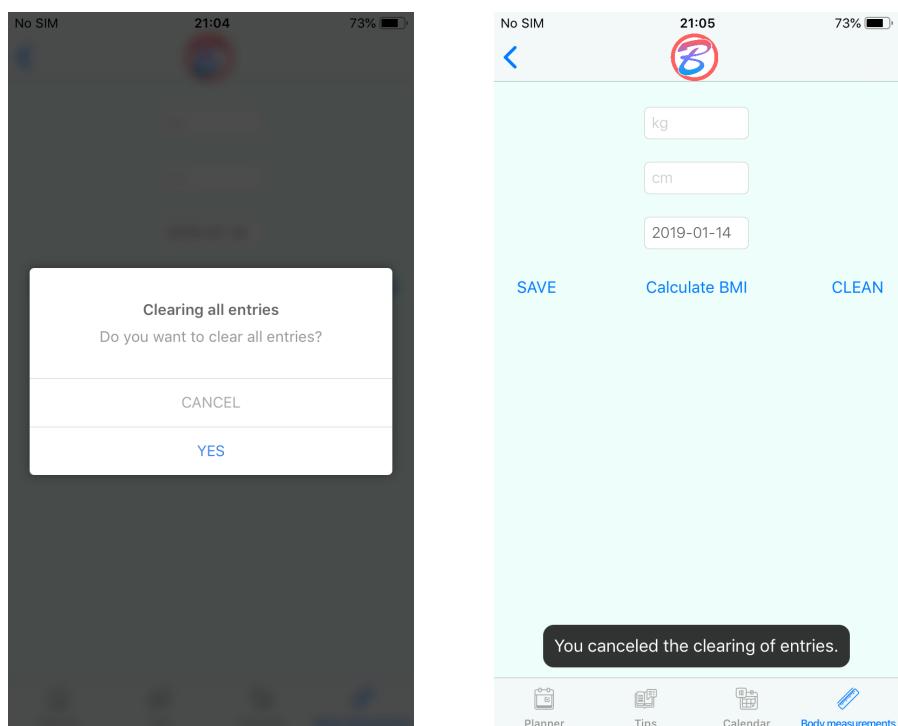
```
if (isFind == true){  
    _ = MeasurementsCoreData().modifyMeasurement(id: foundID, resultBMIValue:  
resultBmi, weightValue: kgValue, dateValue: currentDate)  
    self.view.makeToast("Entry modyfied and saved.", ...)  
} else {  
    _ = MeasurementsCoreData().addMeasurement(resultBMIValue: resultBmi,  
weightValue: kgValue, dateValue: currentDate)  
    self.view.makeToast("Entry added and saved.", ...) }
```

Kod 21 - Dodanie nowego wpisu do bazy danych

## 6.17. Usunięcie wszystkich zapisanych wartości BMI

Chcąc usunąć wszystkie wpisy z bazy danych wystarczy przycisnąć przycisk „CLEAN”, następnie wyświetli się okno dialogowe typu PopupDialog z pytaniem, czy na pewno użytkownik chce usunąć wszystkie wpisy. Przycisk „CANCEL” anuluje wykonanie czynności usuwającej i zamknie okno, tym samym wyświetlając na dole strony wiadomość typu Toast. Natomiast przycisk „SAVE” usuwa wszystkie wcześniej zapisane wpisy z bazy danych (*Rysunek 71*).

Rysunek 71 - Widok przedstawiający usunięcie wszystkich zapisanych wartości BMI



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 22*) jest odpowiedzialny za wywołanie funkcji usuwającej z bazy danych wszystkie wartości dotyczące pomiarów BMI.

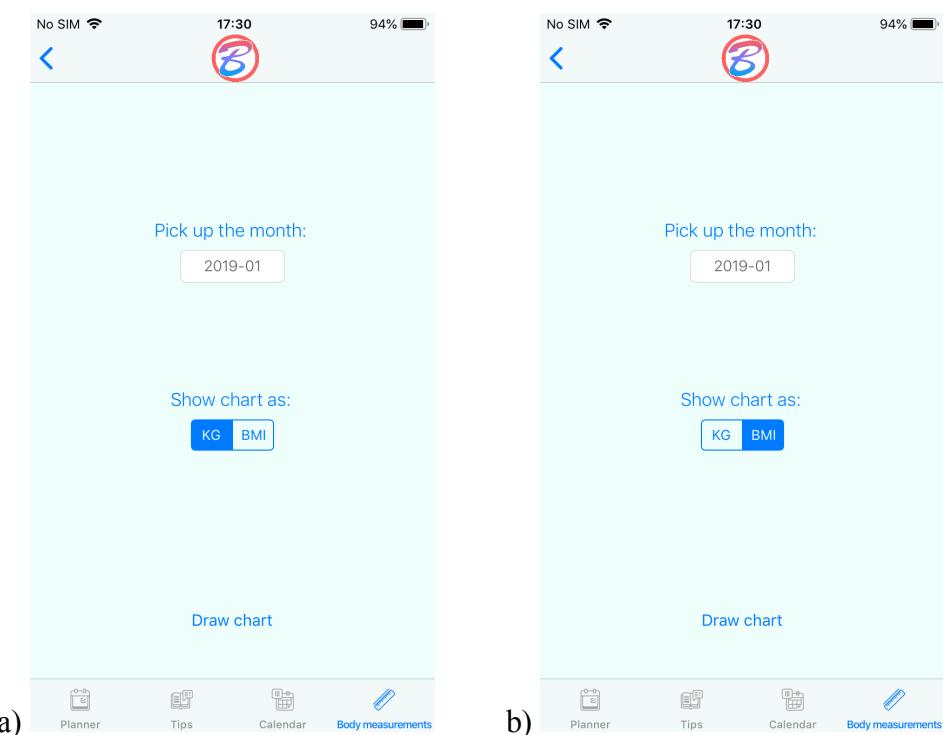
\_ = MeasurementsCoreData().removeAllMeasurements()

Kod 22 - Wywołanie funkcji usuwającej wpisy BMI z bazy danych

## 6.18. Tworzenie wykresu dla wybranej jednostki

Kolejna podstrona zakładki „Body measurements” to wyrysowanie wykresów dla wyliczonych wartości BMI lub wagi użytkownika dla wybranych przedziałów czasowych. Poniższe rysunki przedstawiają pierwszą stronę dla tych diagramów (*Rysunek 72 a) i b)*). Najpierw użytkownik musi wybrać zakres czasu dla jakiego chce otrzymać wykres, do wyboru ma zakres lat i miesięcy, następnie wybiera jednostkę dla jakiej ma dany graf powstać (kilogramy lub BMI). Na koniec przyciska przycisk „Draw chart” i zostaje przekierowany do kolejnej podstrony z gotowym wykresem (*Rysunek 73 i 74*). Tam są naniesione dane wcześniejszych wyliczeń ze strony BMI (waga i indeks BMI), na wykresie oznaczone w postaci punktów i połączone linią. Po przyciśnięciu w dany punkt zostaje wyświetlona przy nim wartość wagi lub BMI i dzień miesiąca (*Rysunek 73 a) i b)*) (*Rysunek 74 a) i b)*).

Rysunek 72 - Widok z wyborem jednostki dla wyrysowania wykresu



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 23*) przedstawia przycisk wyboru odnośnie konkretnej jednostki dla jakiej ma zostać wyrysowany wykres (kilogramy lub BMI). A także sprawdza, czy są jakieś dane dla wybranego miesiąca, a jeśli nie to informuje o tym użytkownika, przy pomocy powiadomienia typu Toast. W przypadku, gdy istnieją jakiekolwiek wpisy dla danego miesiąca, to przechodzi do nowego okna z wyrysowanym diagramem.

```
@IBAction func chooseChartType(_ sender: Any) {
    switch segmentedControl.selectedSegmentIndex {
        case 0:
            // KG
            chartType = 0
        case 1:
            // BMI
            chartType = 1
        default:
            break } }

@IBAction func drawChart(_ sender: Any) {
    if (dateTextField.text != "") {
        let date = dateTextField.text
        year = String(date![0...3])
        month = String(date![5...6])
        let numday = calculateNumbersOfDays(yearValue: year, monthValue: month)
        maxValue = Double(numday)
        array.removeAll()
        for i in 0...numday-1 {
            array.append((0, 0)) }
        loadMeasurementsDataFromDB()
        array = array.filter { $0 != (0, 0.0) }
        if (array.isEmpty) {
            self.view.makeToast("No records for that date.", duration: 3.0,
position: .bottom)
        } else {
            performSegue(withIdentifier: "ShowChart", sender: sender) }
    } else {
        self.view.makeToast("Please, select month and year.", duration: 3.0,
position: .bottom) } }
```

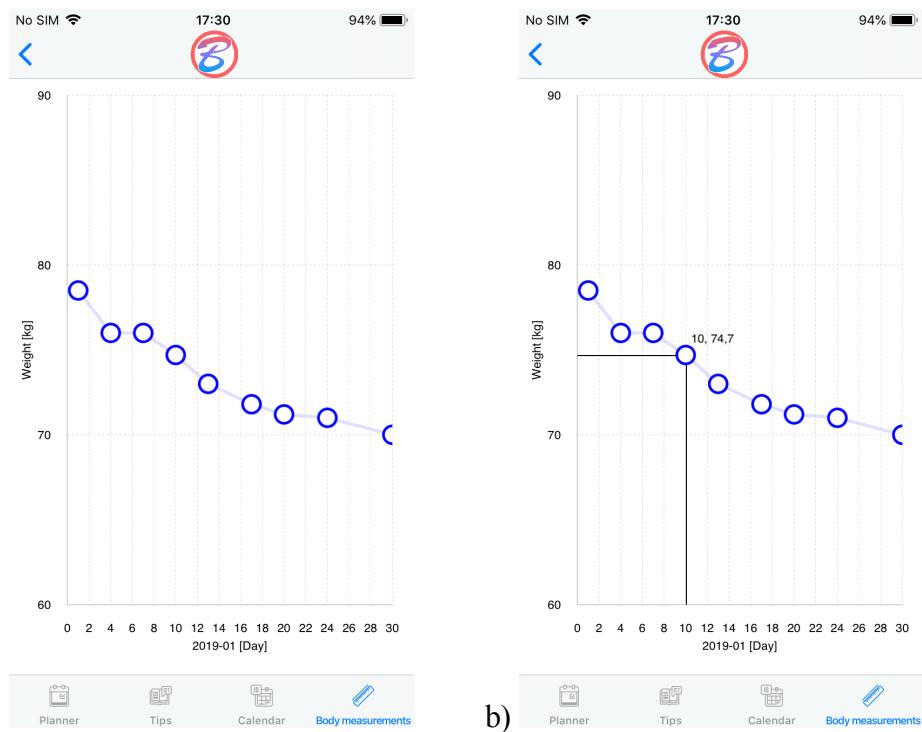
Kod 23 - Wybór jednostki dla wykresu

Poniższy fragment kodu (*Kod 24*) decyduje o tym jaka jednostka pojawi się na osi pionowej (kilogramy lub BMI) i co ile będzie wyrysowana ta przedziałka (10.0 dla kilogramów, 2.0 dla indeksu BMI).

```
if (chartType == 0) {
    xibName = "Weight [kg]"
    multiplyUnit = 10.0
} else {
    xibName = "BMI"
    multiplyUnit = 2.0 }
```

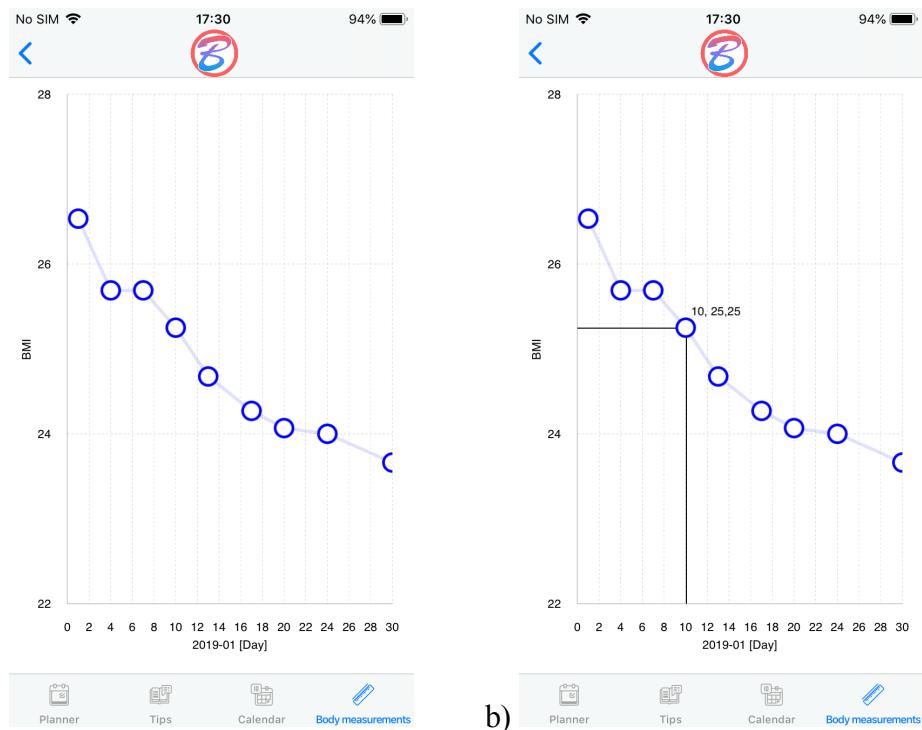
Kod 24 - Podpisanie osi wykresu

Rysunek 73 - Widok wykresu dla kilogramów



Źródło: Opracowanie własne.

Rysunek 74 - Widok wykresu dla BMI

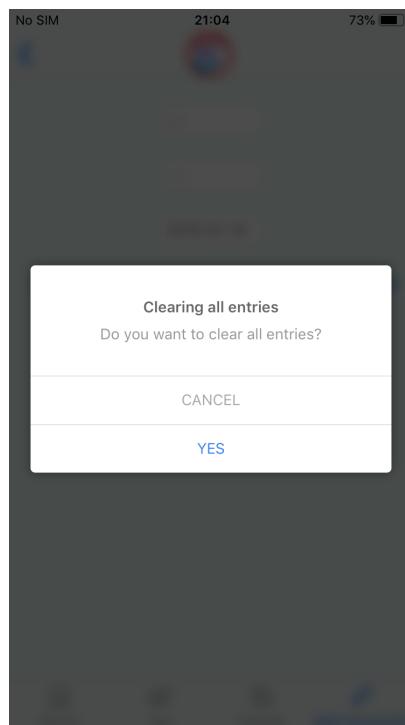


Źródło: Opracowanie własne.

## 6.19. Okno wyboru typu Popup

Okno dialogowe typu PopupDialog jest używane do interakcji z użytkownikiem. Daje to różne możliwości wyboru. Takie okno składa się przeważnie z tytułu danej wiadomości, pytania, które jest skierowane do użytkownika i objaśniające realizację danej akcji oraz przycisków, które dane akcje realizują lub anulują. W rysunku poniżej (Rysunek 75) przedstawiono przykład dla usunięcia wszystkich wcześniej zapisanych wpisów odnośnie rezultatów BMI. Użytkownik chcąc usunąć te wpisy przyciśnie przycisk „YES”, natomiast jeśli chce anulować wykonanie danej czynności, przyciśnie przycisk „CANCEL”.

Rysunek 75 - Widok przedstawiający okno dialogowe typu PopupDialog



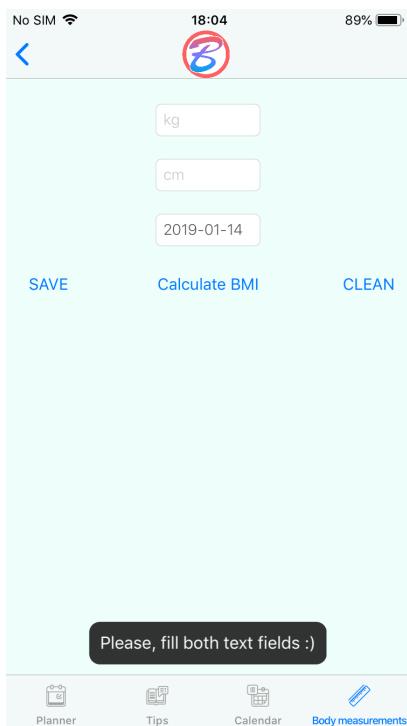
Źródło: *Opracowanie własne.*

## 6.20. Powiadomienia typu Toast

Wiadomość typu Toast, jest wykorzystana do informowania użytkownika o:

- Poprawnym wypełnieniu pól tekstowych
- Jakich znaków ma użyć przy wypełnianiu danego pola
- Informacje przy BMI jaki jest jego zakres (w jakiej grupie się znajduje) w wersjach kolorystycznych
- Przy zapisie danych, czy użytkownik wypełnił wszystkie wymagane pola
- Jako informacja przy anulowaniu akcji

Rysunek 76 - Widok przedstawiający okno informacyjne typu Toast



Źródło: Opracowanie własne.

Poniższy fragment kodu (*Kod 25*) przedstawia wyświetlenie wiadomości typu Toast, ponieważ użytkownik nie wypełnił pól, aby móc obliczyć wartość BMI (*Rysunek 76*).

```
self.view.makeToast("Please, fill both text fields :)", duration: 3.0,  
position: .bottom)
```

Kod 25 - Wyświetlenie wiadomości typu Toast

## 7. Podsumowanie pracy

Zgodnie z założeniami udało się stworzyć aplikację mobilną na iPhone 6 ze wszystkimi przewidzianymi funkcjonalnościami. Aplikacja spełnia funkcję personalnego dzienniczka z powodzeniem ze względu na ilość zaimplementowanych funkcji. Jedną z nich jest przechowywanie i zarządzanie wszystkimi wizytami, które są umieszczone w zakładce Planner.

Następnie zrealizowano funkcję dla wcześniejszego powiadamiania o zbliżającym się terminie. Jest to niezwykle przydatne, ponieważ nie zawsze mamy przy sobie notatki i łatwo można o czymś zapomnieć, a przypomnienie na telefonie jest cały czas aktywne, dopóki się go nie wyłączy.

Kolejną funkcją zrealizowaną w aplikacji było stworzenie poradnika kosmetycznego. Został on podzielony na cztery kategorie wraz z przypisanym kolorem do każdej z nich. Użytkownik ma możliwość wyszukiwania porad w każdej grupie za pomocą słów kluczowych.

Nastecną funkcjonalnością jaką udało się stworzyć to lista kontaktów do gabinetów kosmetycznych, którą użytkownik może sam edytować i rozbudowywać o indywidualny spis połączeń.

Ostatnią z wymaganych funkcji było zrealizowanie wysyłania wiadomości z prośbą o rejestrację terminu zabiegu. Pomyślnie stworzono tą funkcjonalność rozbudowując ją o gotowe szablony wiadomości do wyboru, takie jak umówienie, potwierdzenie i anulowanie wizyty. Po wyborze odpowiedniego szablonu wiadomości użytkownik zostaje przeniesiony do aplikacji systemowej, która jest odpowiedzialna za wysyłanie SMS, tam jest już wpisana wiadomość wraz z konkretną datą dla zabiegu i podaną godziną oraz numerem telefonu.

Wykorzystanie bazy danych z użyciem CoreData wydaje się znacznie łatwiejsze w użyciu niż tradycyjna i powszechna baza danych. Po połączeniu CoreData do projektu wszystko było realizowane praktycznie w jednym miejscu, np. tworzenie encji, załączanie obrazków czy plików tekstowych nie wymagało użycia zapytań SQL tak jak w tradycyjnej bazie danych.

Użycie zewnętrznych bibliotek CocoaPods znacznie ułatwiło i przyspieszyło realizację projektu. Ich baza jest szeroka, dzięki temu programista ma bardzo duży wybór w doborze odpowiedniej biblioteki do swojego programu i języka, w którym programuje.

Interfejs zaprojektowano tak, aby był jak najbardziej intuicyjny i prosty w obsłudze. Menu w aplikacji jest cały czas dostępne dla użytkownika na dole ekranu w postaci paska i zawiera czytelne ikony z nazwami. Nawigacja w każdej zakładce jest wygodna, ponieważ każda kolejna strona jest zagłębieniem poprzedniej, dzięki temu można łatwo i szybko wrócić na wcześniejszą stronę. Dodatkowym atutem podczas budowania interfejsu graficznego była łatwość

w zarządzaniu komponentami (ich rozmieszczenie, nadanie rozmiaru, łatwość podpięcia z kodem).

Można wyróżnić obszary, które należało by usprawnić lub rozbudować. Jednym z potencjalnych zakresów są porady, które na chwilę obecną zostały udostępnione jedynie w języku polskim, jednakże istnieje możliwość rozbudowy porad w przyszłości o język angielski. Aplikacja mogłaby zawierać w sobie dodatkowy dział dotyczący głównych ustawień programu, np. ustawienie języka (polski, angielski), zmiana koloru tła w aplikacji. Kalkulator BMI również mógłby zostać rozbudowany, chociażby o bardziej zaawansowany algorytm do wyliczenia indeksu BMI osobno dla kobiet i mężczyzn, a także z podziałem na odpowiednie grupy wiekowe. Kolejna rzecz, która mogłaby ulec poprawie to nadanie dźwięku dla alertów przypominających lub wiadomości typu toast. Ostatni obszar, który w przyszłości może być rozbudowany, to ilość porad dla każdej z kategorii, np. przy każdej kolejnej aktualizacji aplikacji możliwe jest dodanie kilku nowych porad dla urozmaicenia bazy danych o nowinki urodowe i porady kosmetyczne.

Istotnym elementem, który można byłoby usprawnić jest kompatybilność aplikacji z większą liczbą urządzeń (różne rodzaje iPhone oraz iPad). W chwili obecnej aplikacja jest stworzona tylko pod iPhone 6, co ogranicza potencjalny rynek odbiorców.

Po wprowadzeniu drobnych poprawek w aplikacji można by umieścić „Beauty Planner” w internetowym sklepie „App Store” w wersji darmowej oraz płatnej.

## 8. Bibliografia

1. <https://cocoapods.org/>
2. <https://www.hackingwithswift.com/>
3. <https://www.raywenderlich.com/>
4. <https://stackoverflow.com/>
5. <https://developer.apple.com/>
6. <https://github.com/>
7. <https://youtube.com/>
8. <https://www.ioscreator.com/>
9. <https://www.iphonelife.com/>
10. <https://dev.to/>
11. <https://pl.wikipedia.org/wiki/>
12. <https://zdrowie.tvn.pl/>
13. <https://potrafiszschudnac.pl/>
14. <http://www.centrumdobrejterapii.pl/>
15. <https://www.chcemybycrodzicami.pl/>
16. <https://www.bmi-kalkulator.pl/>
17. <https://polki.pl/>
18. <http://kcmaleksandrow.pl/>
19. <https://en.wikipedia.org/wiki/>
20. <https://www.apple.com/pl/ios/app-store/>
21. [https://www.flaticon.com/free-icon/lips\\_1140518](https://www.flaticon.com/free-icon/lips_1140518)
22. [https://www.flaticon.com/free-icon/lipstick\\_1140523](https://www.flaticon.com/free-icon/lipstick_1140523)
23. [https://www.flaticon.com/free-icon/eyelash\\_1140528](https://www.flaticon.com/free-icon/eyelash_1140528)
24. [https://www.flaticon.com/free-icon/sponge\\_1140556](https://www.flaticon.com/free-icon/sponge_1140556)
25. [https://www.flaticon.com/free-icon/blusher\\_1140531](https://www.flaticon.com/free-icon/blusher_1140531)
26. [https://www.flaticon.com/free-icon/brush\\_1024560](https://www.flaticon.com/free-icon/brush_1024560)
27. [https://www.flaticon.com/free-icon/mascara\\_1024553](https://www.flaticon.com/free-icon/mascara_1024553)
28. [https://www.flaticon.com/free-icon/perfume\\_1024585](https://www.flaticon.com/free-icon/perfume_1024585)
29. [https://www.flaticon.com/free-icon/mirror\\_1140546](https://www.flaticon.com/free-icon/mirror_1140546)
30. [https://www.flaticon.com/free-icon/eyelash\\_1140550](https://www.flaticon.com/free-icon/eyelash_1140550)
31. [https://www.flaticon.com/free-icon/lip-gloss\\_1140535](https://www.flaticon.com/free-icon/lip-gloss_1140535)
32. [https://www.flaticon.com/free-icon/mirror\\_1140517](https://www.flaticon.com/free-icon/mirror_1140517)

33. [https://www.flaticon.com/free-icon/cream\\_1140530](https://www.flaticon.com/free-icon/cream_1140530)
34. [https://www.flaticon.com/free-icon/oil\\_1140543](https://www.flaticon.com/free-icon/oil_1140543)
35. [https://www.flaticon.com/free-icon/eyeliner\\_1140527](https://www.flaticon.com/free-icon/eyeliner_1140527)
36. <https://icons8.com/icon/49496/user-manual>
37. <https://icons8.com/icon/48196/planner>
38. <https://icons8.com/icon/43978/calendar>
39. <https://icons8.com/icon/52572/ruler>
40. [https://www.flaticon.com/free-icon/24-hours\\_899061](https://www.flaticon.com/free-icon/24-hours_899061)
41. [https://www.flaticon.com/free-icon/nail\\_452706#term=beauty%20nails](https://www.flaticon.com/free-icon/nail_452706#term=beauty%20nails)
42. [https://www.flaticon.com/free-icon/yoga\\_1187206](https://www.flaticon.com/free-icon/yoga_1187206)
43. [https://www.flaticon.com/free-icon/facial-mask\\_1024576](https://www.flaticon.com/free-icon/facial-mask_1024576)
44. [https://www.flaticon.com/free-icon/woman\\_1024573](https://www.flaticon.com/free-icon/woman_1024573)
45. [https://www.flaticon.com/free-icon/bar-chart\\_858695](https://www.flaticon.com/free-icon/bar-chart_858695)
46. <https://www.pexels.com/photo/sea-fashion-beach-sand-33622/>
47. <https://www.pexels.com/photo/woman-relaxing-relax-spa-56884/>
48. <https://www.pexels.com/photo/photography-of-a-woman-wearing-black-bikini-drinking-water-1024389/>
49. <https://pixabay.com/en/women-sauna-spa-wellness-model-936549/>
50. <https://pixabay.com/en/berry-close-up-cooking-delicious-1851349/>
51. <https://pixabay.com/en/people-hands-manicure-cuticle-2587157/>
52. <https://www.pexels.com/photo/woman-s-pink-pedicure-973405/>
53. <https://www.pexels.com/photo/photo-of-woman-with-lotion-on-hand-1368692/>
54. <https://www.pexels.com/photo/spilled-white-nail-polish-on-pink-surface-1029894/>
55. <https://pixabay.com/en/people-hands-manicure-cuticle-2583493/>
56. <https://www.pexels.com/photo/close-up-photography-of-a-woman-792994/>
57. <https://www.pexels.com/photo/adult-art-beautiful-beauty-413880/>
58. <https://www.pexels.com/photo/woman-girl-beauty-mask-3192/>
59. <https://www.pexels.com/photo/clear-glass-bowl-beside-yellow-flower-1638280/>
60. <https://www.pexels.com/photo/beans-coffee-morning-espresso-9186/>
61. <https://pixabay.com/en/blur-braided-hair-brunette-close-up-1853957/>
62. <https://www.pexels.com/photo/assorted-sliced-fruits-1128678/>
63. <https://www.pexels.com/photo/woman-wearing-white-long-sleeved-shirt-973401/>
64. <https://pixabay.com/en/haircut-hair-cut-beauty-salon-combs-834280/>
65. <https://www.pexels.com/photo/woman-holding-white-plumeria-flower-1234901/>

## **9. Spis rysunków**

Rysunek 1 - Zrzut ekranu aplikacji "Calendars" .....	14
Rysunek 2 - Zrzut ekranu widoku kalendarza z aplikacji "Beauty Planner" .....	14
Rysunek 3 - Zrzut ekranu aplikacji "Wunderlist" .....	15
Rysunek 4 - Zrzut ekranu edycji wizyty w aplikacji "Beauty Planner" .....	15
Rysunek 5 - Zrzut ekranu aplikacji "Calculator Free" .....	16
Rysunek 6 - Zrzut ekranu zakładki BMI w aplikacji "Beauty Planner" .....	16
Rysunek 7 - Diagram klas.....	18
Rysunek 8 - Diagram przypadków użycia dla przeglądu listy wizyt, dodawania i ich edycji .....	19
Rysunek 9 - Diagram przypadków użycia dla przeglądu krótkich porad i porad.....	20
Rysunek 10 - Diagram przypadków użycia dla przeglądu kalendarza.....	20
Rysunek 11 - Diagram przypadków użycia dla kalkulatora BMI i wykresów .....	21
Rysunek 12 - LaunchScreen storyboard.....	24
Rysunek 13 - Struktura komponentów widoku LaunchScreen.....	24
Rysunek 14 - Main storyboard.....	25
Rysunek 15 - Struktura widoków dla Main storyboard.....	26
Rysunek 16 - Planner storyboard .....	27
Rysunek 17 - Struktura widoków Planner storyboard cz.1 .....	28
Rysunek 18 - Struktura widoków Planner storyboard cz.2 .....	29
Rysunek 19 - Tips storyboard .....	30
Rysunek 20 - Struktura widoków Tips storyboard.....	31
Rysunek 21 - Calendar storyboard.....	32
Rysunek 22 - Measurements storyboard .....	33
Rysunek 23 - Struktura widoków Measurements storyboard.....	34
Rysunek 24 - Rodzaje użytych plików na potrzebę budowy klas.....	35
Rysunek 25 - Grupa Main.....	36
Rysunek 26 - Grupa Planner .....	41
Rysunek 27 - Grupa Tips.....	48
Rysunek 28 - Grupa Calendar .....	53
Rysunek 29 - Grupa Measurements .....	55
Rysunek 30 - Grupa Contacts .....	63
Rysunek 31 - Grupa Reminder.....	70

Rysunek 32 - Zaimplementowane encje w CoreData .....	72
Rysunek 33 - Budowa encji AppointmentEntity .....	72
Rysunek 34 - Budowa encji ContactsEntity .....	73
Rysunek 35 - Budowa encji MeasurementsEntity .....	73
Rysunek 36 - Budowa encji ShortTipsEntity.....	74
Rysunek 37 - Budowa encji TipEntity .....	74
Rysunek 38 - Struktura zasobów typu Bundle.....	75
Rysunek 39 - Struktura zasobów typu Assets.....	76
Rysunek 40 - Zimportowane biblioteki CocoaPods .....	77
Rysunek 41 - Strona startowa aplikacji.....	82
Rysunek 42 - Okno wyświetlające krótką poradę na stronie startowej.....	83
Rysunek 43 - Wyświetlanie wizyt na liście .....	84
Rysunek 44 - Widok z dodawaniem nowej wizyty do listy .....	86
Rysunek 45 – Widok z walidacją pól tekstowych w dodawaniu nowej wizyty .....	86
Rysunek 46 - Widok z edycją istniejącej wizyty .....	89
Rysunek 47 - Widok dla usunięcia wybranej wizyty .....	90
Rysunek 48 – Widok z wyświetleniem okna dialogowego z wyborem szablonu wiadomości typu SMS .....	91
Rysunek 49 - Gotowe szablony wiadomości .....	92
Rysunek 50 - Widok z listą kontaktów w aplikacji.....	94
Rysunek 51 - Widok z dodaniem nowego kontaktu do listy .....	95
Rysunek 52 - Widok z walidacją pól tekstowych w dodawaniu nowego kontaktu .....	96
Rysunek 53 - Widok z edycją istniejącego kontaktu .....	97
Rysunek 54 - Widok z usunięciem danego kontaktu .....	99
Rysunek 55 - Widok przedstawiający ustawienie powiadomienia .....	100
Rysunek 56 - Zrzut ekranu form powiadomień na telefonie .....	101
Rysunek 57 - Widok ustawień kalendarza dla zmiany daty .....	102
Rysunek 58 - Widok kalendarza w aplikacji .....	103
Rysunek 59 - Widok podglądu wydarzenia w kalendarzu .....	104
Rysunek 60 - Widok głównej strony porad z ich kategoriami.....	106
Rysunek 61 - Widok czterech kategorii porad.....	107
Rysunek 62 - Widok opcji wyszukiwania porady po wpisanym haśle w wyszukiwarce .....	108
Rysunek 63 - Widok przykładowej porady .....	109
Rysunek 64 - Widok głównej strony pomiarów ciała .....	111

Rysunek 65 - Widok kalkulatora BMI .....	112
Rysunek 66 - Widok wyliczonych wartości BMI (wygłodzenie i wychudzenie).....	113
Rysunek 67 - Widok wyliczonych wartości BMI (niedowaga i wartość prawidłowa).....	114
Rysunek 68 - Widok wyliczonych wartości BMI (nadwaga i I stopień otyłości) .....	115
Rysunek 69 - Widok wyliczonych wartości BMI (II stopień otyłości i otyłość skrajna).....	116
Rysunek 70 - Widok z zapisem wartości BMI z wybraną datą .....	117
Rysunek 71 - Widok przedstawiający usunięcie wszystkich zapisanych wartości BMI .....	118
Rysunek 72 - Widok z wyborem jednostki dla wyrysowania wykresu .....	119
Rysunek 73 - Widok wykresu dla kilogramów.....	121
Rysunek 74 - Widok wykresu dla BMI.....	121
Rysunek 75 - Widok przedstawiający okno dialogowe typu PopupDialog .....	122
Rysunek 76 - Widok przedstawiający okno informacyjne typu Toast.....	123

## **10. Spis kodów źródłowych**

Kod 1 - Losowanie i wyświetlanie krótkiej porady .....	83
Kod 2 - Dodanie nowej wizyty do bazy danych wraz z walidacją pól tekstowych .....	87
Kod 3 - Realizacja edycji wizyty .....	89
Kod 4 - Wywołanie funkcji usuwającej wizytę .....	90
Kod 5 - Realizacja wyboru szablonu wiadomości SMS .....	93
Kod 6 - Dodanie nowego kontaktu do bazy danych .....	96
Kod 7 - Uruchomienie kodu po przyciśnięciu przycisku "SAVE" i zapis do danych .....	98
Kod 8 - Modyfikacja kontaktu i zapis w bazie danych .....	98
Kod 9 - Wywołanie funkcji usuwającej kontakt z bazy danych .....	99
Kod 10 - Włączenie lub wyłączenie powiadomienia .....	101
Kod 11 - Przygotowanie danych dla kalendarza.....	104
Kod 12 - Wyświetlenie okna dialogowego ze szczegółową informacją dla wydarzenia w kalendarzu .....	105
Kod 13 - Sprawdzenie, które połączenie (segoe) zostało wywołane .....	106
Kod 14 - Wyszukiwanie po wpisanych słowach kluczowych.....	109
Kod 15 - Wyświetlenie porad dla wybranej kategorii w wersji kolorystycznej .....	110
Kod 16 - Algorytm wyliczający indeks BMI.....	112
Kod 17 - Przedział wartości dla dwóch kategorii (wygłodzenie i wychudzenie).....	113
Kod 18 - Przedział wartości dla dwóch kategorii (niedowaga i wartość prawidłowa) .....	114
Kod 19 - Przedział wartości dla dwóch kategorii (nadwaga i I stopień otyłości).....	115
Kod 20 - Przedział wartości dla dwóch kategorii (II stopień otyłości i otyłość skrajna).....	116
Kod 21 - Dodanie nowego wpisu do bazy danych.....	117
Kod 22 - Wywołanie funkcji usuwającej wpisy BMI z bazy danych .....	118
Kod 23 - Wybór jednostki dla wykresu.....	120
Kod 24 - Podpisanie osi wykresu.....	120
Kod 25 - Wyświetlenie wiadomości typu Toast .....	123

## **Dodatek D1: Zawartość płyty CD**

Do niniejszej pracy załączono płytę CD wraz z poniższymi plikami:

- praca dyplomowa w wersji pisemnej (.docx, .pdf)
- aplikacja moblina



O Ś W I A D C Z E N I E  
(STUDENTA)

.....  
Imię i Nazwisko studenta

.....  
nr albumu

Oświadczam, że moja praca pt.: .....

- .....
- 1) została przygotowana przeze mnie samodzielnie\*,
  - 2) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (j.t. Dz. U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem,
  - 3) nie zawiera danych i informacji, które uzyskałem w sposób niedozwolony,
  - 4) nie była podstawą nadania dyplomu uczelni wyższej lub tytułu zawodowego ani mnie ani innej osobie.

Ponadto oświadczam, że treść pracy przedstawionej przez mnie do obrony, zawarta na przekazywanym nośniku elektronicznym, jest identyczna z jej wersją drukowaną.

.....  
data

.....  
podpis studenta

\*Uwzględniając merytoryczny wkład promotora (w ramach prowadzonego seminarium dyplomowego)

