Kolegium Nauk Przyrodniczych
Uniwersytet Rzeszowski

Przedmiot:

Programowanie Urządzeń Mobilnych

Nazwa projektu:

Budzik, stoper, minutnik

Wykonał:

Kamil Filar, Informatyka rok III, lab1

Prowadzący: dr inż. Piotr Lasek

Rzeszów 2020

Spis treści:

1. Opis aplikacji

Aplikacja nazywa się: „Projekt PUM” - pod taką nazwą jest zapisana w urządzeniu. Nazwa i zakres projektu to: „Budzik, stoper, minutnik”. Aplikacja służy do odmierzania czasu. Jej główne funkcjonalności to:

1. Budzik
    a. Dodanie nowego alarmu
    b. Edycja istniejącego alarmu
    c. Możliwość dezaktywacji alarmu
    d. Usunięcie alarmu
2. Stoper
    a. Wystartowanie licznika
    b. Zatrzymanie licznika
    c. Reset licznika
    d. Dokonanie pomiarów czasu i wyświetlenie ich
3. Minutnik
    a. Wybranie przez użytkownika odpowiedniego czasu
    b. Uruchomienie odmierzania czasu
    c. Zatrzymanie odmierzania czasu

Do stworzenia aplikacji wykorzystałem język Dart z frameworkiem Flutter. Wybrałem to środowisko ze względu na intrygujące, proste środowisko tworzenia graficznego interfejsu użytkownika. Szczegółowy opis funkcjonalności znajduje się w punkcie **2**. Aplikacja umożliwia min. precyzyjne mierzenie czasu w stoperze z dokładnością do 1 milisekundy, a także dokonanie dziesięciu pomiarów przez użytkownika. Minutnik umożliwia dokładne odmierzanie czasu z dokładnością do 1 sekundy. Budzik umożliwia funkcje ustawiania nowego alarmu, edytowania go. W odróżnieniu od innych aplikacji zrobiłem notyfikację jako formę powiadomienia

użytkownika o tym, że alarm działa. To rozwiązanie jest moim zdaniem niestandardowe i lepsze od klasycznego, w którym mamy możliwość dokładania drzemki w nieskończoność (w tej aplikacji również jest opcja drzemki ale zrealizowana w inny sposób).

2. Opis funkcjonalności

Poniżej zostaną zaprezentowane funkcjonalności wraz z opisem wszystkich kluczowych plików stworzonych lub modyfikowanych przeze mnie w projekcie. Są to:

1. main.dart
2. Budzik.dart
3. NewAlarm.dart
4. EditAlarm.dart
5. Stoper.dart
6. Minutnik.dart

2.1 main.dart

Klasa main.dart jest główną klasą w projekcie. To w niej znajduje się konstruktor dzięki któremu działa budzik, a także w tej klasie inicjowana jest lokalna baza danych oraz wywoływane są odpowiednie klasy (Budzik, Stoper, Minutnik). W tej klasie znajduje się element AppBar, który służy w całej aplikacji do nawigacji po niej. Wygląd tego elementu znajduje się na rysunku Rysmain.2a:

Rysmain.2a

Kod tego elementu został przedstawiony na rysunku Rysmain.2b.

```dart
@override
Widget build(BuildContext context) {
  return DefaultTabController(

    initialIndex: widget.selectedPage,
    length: 3,
    child: Scaffold(
      appBar: AppBar(
        title: Text(
          'Projekt PUM',
          style: GoogleFonts.comicNeue(
              fontWeight: FontWeight.w700,
              fontSize: 30,
              color: Colors.lightGreenAccent.shade400),
        ),  // Text
        centerTitle: true,
        flexibleSpace: Container(
          decoration: BoxDecoration(
            gradient: LinearGradient(
                begin: Alignment.topLeft,
                end: Alignment.bottomRight,
                colors: <Color>[
                  Colors.grey.shade900,
                  Colors.green.shade900,
                  Colors.green.shade700,
                  Colors.grey.shade900,
                ]),  // <Color>[], LinearGradient
            border: Border.all(
              width: 1,
              color: Colors.green.shade900,
            ),  // Border.all
            boxShadow: [
              BoxShadow(
                color: Colors.lightGreenAccent.shade700.withOpacity(0.8),
                blurRadius: 8,
                spreadRadius: 2,
                offset: Offset(4, 4),
              ),  // BoxShadow
            ],
```

Rysmain.2b

W swoim ciele AppBar posiada TabBar który jest odpowiedzialny za podzielenie widoku na trzy części, w których odpowiednio znajdują się Budzik, Stoper, Minutnik. W swoim ciele TabBar.view posiada wywołania 3 metod, które przyczyniają się do wyświetlania odpowiednich widoków w aplikacji.

Poniższy rysunek (Rysmain.2c) przedstawia sposób inicjalizacji bazy danych oraz wywołania odpowiednich widoków w metodach, które są wywołane we wcześniej wspomnianym elemencie TabBar.view.

```
class _MyHomePageState extends State<MyHomePage> with TickerProviderStateMixin {
  var db;
  initDb() async {
    print("Zainicjalizowano bazę danych");
    Database db = await openDatabase(join(await getDatabasesPath(), 'AlarmDB.db'));
    // await db.query('alarms').then((value) => print(value));
    print("Inicjalizacja zakończona");
  }

  //------------------------ Budzik ---------------------------\\
  Widget alarm() {
    return Budzik();
  }

  //------------------------ Minutnik ------------------------\\
  Widget timer() {
    return Minutnik();
  }

  //------------------------ Stoper---------------------------\\
  Widget stopwach() {
    List<String> x = new List(10);
    x[0] = null;
    x[1] = x[0];
    x[2] = x[1];
    x[3] = x[2];
    x[4] = x[3];
    x[5] = x[4];
    x[6] = x[5];
    x[7] = x[6];
    x[8] = x[7];
    x[9] = x[8];
    return Stoper(x);
  }
```

Rysmian.2c

Klasa mian posiada również element, który jest odpowiedzialny za rozruch budzika tzn. za wystąpienie notyfikacji (oraz ikony aplikacji), która informuje użytkownika o tym, że dany alarm jest aktywny (posiada również opcje dla iOS). Znajduje się na rysunku Rysmian.2d:

```dart
final FlutterLocalNotificationsPlugin flutterLocalNotificationsPlugin =
    FlutterLocalNotificationsPlugin();

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  var initializationSettingsAndroid = AndroidInitializationSettings('clock');
  var initializationSettingsIOS = IOSInitializationSettings(
      requestAlertPermission: true,
      requestBadgePermission: true,
      requestSoundPermission: true,
      onDidReceiveLocalNotification:
          (int id, String title, String body, String payload) async {});
  var initializationSettings = InitializationSettings(
      android: initializationSettingsAndroid, iOS: initializationSettingsIOS);
  await flutterLocalNotificationsPlugin.initialize(initializationSettings,
      onSelectNotification: (String payload) async {
    if (payload != null) {
      debugPrint('notification payload: ' + payload);
    }
  });

  runApp(MyApp());
}
```

Rysmain.2d

Kolejnym ważnym elementem pliku main.dart jest klasa *Alarm()*, która zawiera w sobie konstruktor oraz funkcje potrzebne do tworzenia nowych alarmów. Wygląda następująco:

```dart
class Alarm {
  int ID_Alarm;
  DateTime Alarm_DateTime;
  int Alarm_Vibration;
  int Alarm_Drzemka;
  int Alarm_isActive;
  int Monday;
  int Tuesday;
  int Wednesday;
  int Thursday;
  int Friday;
  int Saturday;
  int Sunday;
```

Rysmain.2e

```dart
Alarm(
    {this.ID_Alarm,
    this.Alarm_DateTime,
    this.Alarm_Vibration,
    this.Alarm_Drzemka,
    this.Alarm_isActive,
    this.Monday,
    this.Tuesday,
    this.Wednesday,
    this.Thursday,
    this.Friday,
    this.Saturday,
    this.Sunday});
```

Rysmain.2f

```
Alarm.fromMap(Map<String, dynamic> map) {
  this.ID_Alarm = map['ID_Alarm'];
  this.Alarm_DateTime = DateTime.parse(map['Alarm_DateTime']);
  this.Alarm_Vibration = map['Alarm_Vibration'];
  this.Alarm_Drzemka = map['Alarm_Drzemka'];
  this.Alarm_isActive = map['Alarm_isActive'];
  this.Monday = map['Monday'];
  this.Tuesday = map['Tuesday'];
  this.Wednesday = map['Wednesday'];
  this.Thursday = map['Thursday'];
  this.Friday = map['Friday'];
  this.Saturday = map['Saturday'];
  this.Sunday = map['Sunday'];
}
```

Rysmain.2g

Powyższy rysunek (Rysmain.2g) przedstawia metodę która umożliwia „odczyt z mapy” wartości zdefiniowanych w konstruktorze, analogicznie metoda *toMap()* mapuje wartości z konstruktora (Rysmain.2h).

```
Map<String, dynamic> toMap() {
  return {
    'ID_Alarm': ID_Alarm,
    'Alarm_DateTime': Alarm_DateTime.toString(),
    'Alarm_Vibration': Alarm_Vibration,
    'Alarm_Drzemka': Alarm_Drzemka,
    'Alarm_isActive': Alarm_isActive,
    'Monday': Monday,
    'Tuesday': Tuesday,
    'Wednesday': Wednesday,
    'Thursday': Thursday,
    'Friday': Friday,
    'Saturday': Saturday,
    'Sunday': Sunday,
  };
}
```

Rysmain.2h

2.2 Budzik.dart

Jest to plik, w którym znajduje się zawartość budzika. Budzi (z dodanym jednym alarmem) prezentuje się następująco:

1. Pod tym numerem znajduje się element, który jest odpowiedzialny za wygląd poszczególnego alarmu. Po jego lewej stronie mamy pobrane dane z bazy odnośnie alarmu (RysBu.2a), a elementy:

a – jest to przycisk służący do usuwania danego alarmu. Jego działanie zostało zaprezentowane na RysBu.2b oraz wygląd kodu na RysBu.2.c

RysBu.2b

RysBu.2c

b – przycisk, który przenosi nas do panelu edycji. W nim jest pobrana informacja o danym alarmie i przekazana do edycji (jego działanie zostało zaprezentowane na RysBu.2d).

RysBu.2d

c – jest to element switch, który jest odpowiedzialny za to czy dany alarm jest aktywny. Jest zrobiony po to żeby użytkownik mógł dobrowolnie zrezygnować z użycia budzika w danym dniu. Jego działanie zostało pokazane na RysBu.2e:

RysBu.2e

2. Jest to przycisk, który przenosi nas do panelu, gdzie użytkownik ma możliwość dodania nowego alarmu. Działanie (RysBu.2f) oraz wygląd tego elementu (RysBu.2h) został zaprezentowany poniżej:

RysBu.2f

RysBu.2h

Ważnymi funkcjami zapewniającymi poprawne działanie budzika są 4 funkcje:

- *autoUpdateDates()*
- *checkForAlarms()*
- *scheduleAlarm();*
- *getAlarms();*

Metoda *autoUpdateDates()* jest wywoływana raz dziennie (RysBu.2i) i jest odpowiedzialna za to żeby aktualizowała odpowiednio dni. Sprawdza

jaki jest jutro dzień i czy dla danego dnia jest ustawiony budzik na wartość 1, jeżeli tak to do daty dodaje jeden dzień i aktualizuje alarm, jeżeli nie to sprawdza czy następny dzień to np. wotrek itd. Funkcja znajduje się na RysBu.2j.

```
@override
void initState() {
  super.initState();
  initializeDateFormatting();
  dateFormat = new DateFormat.MMMMd('pl');
  timer = Timer.periodic(Duration(seconds:1), (Timer t) => checkForAlarms());
  timer2 = Timer.periodic(Duration(days: 1), (Timer t) => autoUdateDates());
  _alarms = getAlarms();
```

RysBu.2i

```
62    autoUdateDates() async {
63      List<Alarm> alarms = await getAlarms();
64      var pon=1;
65      var wt=2;
66      var sr=3;
67      var czw=4;
68      var pt=5;
69      var sb=6;
70      var nd=7;
71      DateTime tomorrow = new DateTime.now().add(Duration(days: 1));
72      for(var alarm in alarms){
73        if(tomorrow.weekday==pon){...}
110       else if(tomorrow.weekday==wt){...}
147       else if(tomorrow.weekday==sr){...}
184       else if(tomorrow.weekday==czw){...}
221       else if(tomorrow.weekday==pt){...}
258       else if(tomorrow.weekday==sb){...}
295       else if(tomorrow.weekday==nd){...}
332     }
333   }
```

RysBu.2j

Metoda *checkForAlarms()* sprawdza czy jakiś alarm powinien dzwonić. Metoda jest wywoływana co sekundę (RysBu.2i). W sowim ciele sprawdza czy dany alarm ma ustawioną wibrację oraz to czy jest ustawiona drzemka. Jeżeli tak to jest sprawdzana dostępność wibracji na danym urządzeniu i je wywołuje. Podobnie jest z drzemką. Drzemka jest pojedyncza – oznacza to, że jeżeli alarm ma ustawioną drzemką to drzemka po

zadzwonieniu alarmu doda się automatycznie o 5min do przodu po czym jak zadzwoni alarm cofnie z powrotem się do stanu pierwotnego (RysBu.2k).

```
checkForAlarms() async {
  List<Alarm> alarms = await getAlarms();
  DateTime now =new DateTime.now();
  now = new DateTime(now.year,now.month,now.day,now.hour,now.minute,now.second,0,0).add(Duration(hours: 1));
  for(var alarm in alarms)
    {
      if(alarm.Alarm_DateTime==now && alarm.Alarm_isActive==1)
        {
          scheduleAlarm();
```

```
if(alarm.Alarm_Vibration==1){
  if (await Vibration.hasVibrator()) {
    if (await Vibration.hasCustomVibrationsSupport()) {
      Vibration.vibrate(pattern: [500, 1000, 500, 1000, 500, 1000, 5
    }
    else{
      Vibration.vibrate();
      await Future.delayed(Duration(milliseconds: 500));
      Vibration.vibrate();
    }
  }
```

```
if(alarm.Alarm_Drzemka==1){
  Alarm copyOfAlarm = Alarm.copy2(alarm,alarm.Alarm_DateTime.add(new Duration(minutes: 4)),0);
  updateAlarm(copyOfAlarm);
}
if(alarm.Alarm_Drzemka==0){
    Alarm copyOfAlarm = Alarm.copy2(alarm,alarm.Alarm_DateTime.add(new Duration(minutes: -4)),1);
    updateAlarm(copyOfAlarm);
}
```

RysBu.2k

Metoda *scheduleAlarm()* (RysBu.2l) jest odpowiedzialna za wywołanie notyfikacji, która poinformuje użytkownika o tym, że alarm działa. Nie chciałem w projekcie umieszczać dodatkowego okna na cały ekran z informacją, że budzik jest uruchomiony ponieważ moim zdaniem jest to co najmniej zbyt „ofensywne” w stosunku do użytkownika. W zamian notyfikacja wyświetla pełną informację o budziku i w łatwy sposób można wyłączyć budzik nie szukając przycisku do ustawienia drzemki czy wyłączenia alarmu – wystarczy ją przesunąć i wszystko podzieje się automatycznie.

```dart
void scheduleAlarm() async {
  var scheduledNotificationDateTime =
      DateTime.now();
  var androidPlatformChannelSpecifics = AndroidNotificationDetails(
    'alarm_notif',
    'alarm_notif',
    'Channel for Alarm notification',
    icon: 'clock',
    sound: RawResourceAndroidNotificationSound('song'),
    largeIcon: DrawableResourceAndroidBitmap('clock'),
  );  // AndroidNotificationDetails

  var iOSPlatformChannelSpecifics = IOSNotificationDetails(
      sound: 'song.wav',
      presentAlert: true,
      presentBadge: true,
      presentSound: true);
  var platformChannelSpecifics = NotificationDetails(
      android: androidPlatformChannelSpecifics,
      iOS: iOSPlatformChannelSpecifics);
  // ignore: deprecated_member_use
  await flutterLocalNotificationsPlugin.schedule(
      0,
      'Alarm uruchomiony!',
      'Czas na działanie :)',
      scheduledNotificationDateTime,
      platformChannelSpecifics);
}
```

(RysBu.2l)

Do pobrania danych alarmów użyłem metody *Future<List<Alarm>> getAlarms()*, gdzie pobierana jest lista wszystkich alarmów (RysBu.2m).

```dart
Future<List<Alarm>> getAlarms() async {
  List<Map<String, dynamic>> alarmsDB;
  Database db =
  await openDatabase(join(await getDatabasesPath(), 'AlarmDB.db'));
  await db.query('BudzikEntity').then((value) => alarmsDB = value);

  List<Alarm> alarms=List.generate(alarmsDB.length, (index) {
    Alarm alarm = Alarm.fromMap(alarmsDB[index]);
    return alarm;
  });  // List.generate
  // print(alarms);
  return alarms;
}
```

RysBu.2m

2.2.1 NewAlarm.dart

Nowy alarm jest widokiem w którym możemy dodać nowe dane do budzika. Podgląd na panel dodawania nowego alarmu:

1. Pod tym numerem znajduje się element, który przenosi nas do poprzedniego widoku w momencie kiedy zrezygnujemy z dodawania nowego alarmu.

2. Numberpicker – jest to element często wykorzystywany w moim projekcie, pozwala on na wybranie w intuicyjny sposób użytkownikowi interesującej go godziny. Domyślnie ustawiłem wartości na 6:30. Wygląd kodu, jest prawie identyczny we wszystkich przypadkach i został on opisany szczegółowo w punkcie 2.4 Minutnik.dart. W tym przypadku jedyną różnicą jest fakt, że został pozbawiony kolumny z sekundami (z przyczyn użyteczności).

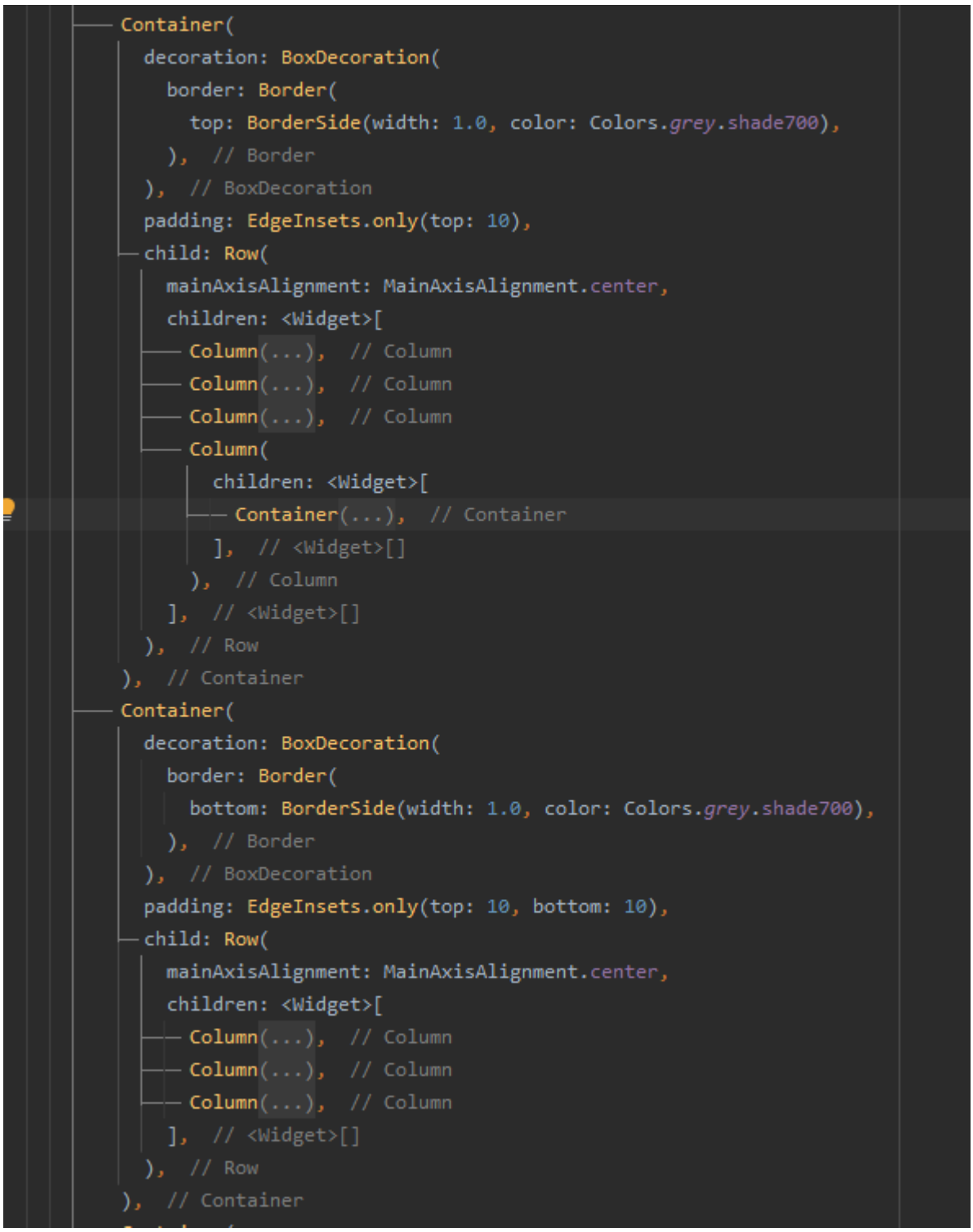
3. Jest to panel z wyborem dni. Jego działanie polega na wybraniu w jakie dni tygodnia dany alarm ma być aktywny. Dni, które nie są wybrane (domyślnie wybrane są wszystkie) nie posiadają charakterystycznego, zielonego lub niebieskiego koloru wypełnienia (patrz RysNA.2a):

RysNA.2a

Przycisk *pon*, *śr*, *nd* – **nie** są zaznaczone

Przycisk *wt*, *czw*, *pt*, *sb* – są zaznaczone

Wygląd kodu dla pojedynczego przycisku został przedstawiony na rysuku RysNA.2b:



RysBu.2b (w każdej kolumnie jest kod dla pojedynczego przycisku)

4. Jest to panel z wyborem czy do danego alarmu ma zostać dodana drzemka i wibracja, rozwiązanie zostało wdrożone przy pomocy elementu switch (RysNA.2c). Domyślne wartości są ustawione na 1.

```
Switch(
  value: isSwitched_drzemka,
  onChanged: (value){
    setState(() {
      isSwitched_drzemka=value;
      if(isSwitched_drzemka==true){
        drzemka_to_add=1;
      }
      else{
        drzemka_to_add=0;
      }
      print('Drzemka: '+drzemka_to_add.toString());
    });
  },
  activeTrackColor: Colors.green.shade700,
  activeColor: Colors.white,
)  // Switch
```

RysNA.2c

5. Jest to przycisk, który służy do zapisania danych. To w nim znajdują się dwa zabezpieczenia dot. tego, że nie ma możliwości dodania alarmu jeżeli nie został wybrany ani jeden dzień (wówczas wartości są ustawiane automatycznie dla wszystkich dni RysNA.2d) a także zabezpieczenie sprawdzające czy już podany alarm istnieje (nie ma możliwości dodania dwóch takich samych RysNa.2e). Kolejnym etapem jest dodanie do bazy danych informacji o podanym budziku i przejście na stronę główną budzika (RysNA.2f1 i RysNA.2f2). W momencie gdy taki sam budzik już wystąpi po prostu użytkownik zostanie przeniesiony na stronę główną. Jest za to odpowiedzialna funkcja z obrazka RysNA.2g

```
if(pon==0 && wt==0 && sr==0 && czw==0 && pt==0 && sb==0 && nd==0){
  pon = wt = sr = czw = pt = sb = nd = 1;
  print("Ustawiono wartości domyślne dla dni ponieważ nie wybrano żadnego!");
}
```

RysNA.2d

```dart
bool compare(Alarm alarmA, Alarm alarmB){
    if(alarmA.Alarm_DateTime==alarmB.Alarm_DateTime)
      if(alarmA.Alarm_Drzemka==alarmB.Alarm_Drzemka)
        if(alarmA.Alarm_Vibration==alarmB.Alarm_Vibration)
          if(alarmA.Alarm_isActive==alarmB.Alarm_isActive)
            if(alarmA.Monday==alarmB.Monday)
              if(alarmA.Tuesday==alarmB.Tuesday)
                if(alarmA.Wednesday==alarmB.Wednesday)
                  if(alarmA.Thursday==alarmB.Thursday)
                    if(alarmA.Friday==alarmB.Friday)
                      if(alarmA.Saturday==alarmB.Saturday)
                        if(alarmA.Sunday==alarmB.Sunday)
                          return true;
    return false;
}
```

RysNA.2e – ta funkcja znajduje się w pliku main.dart

```dart
checkAlarmExist(Alarm A1) async{
  List<Alarm> alarms = await getAlarms();
  for(var alarm in alarms)
    {
      if(alarm.compare(alarm, A1)){
        return true;
      }
    }
  return false;
}
```

RysNA.2f2

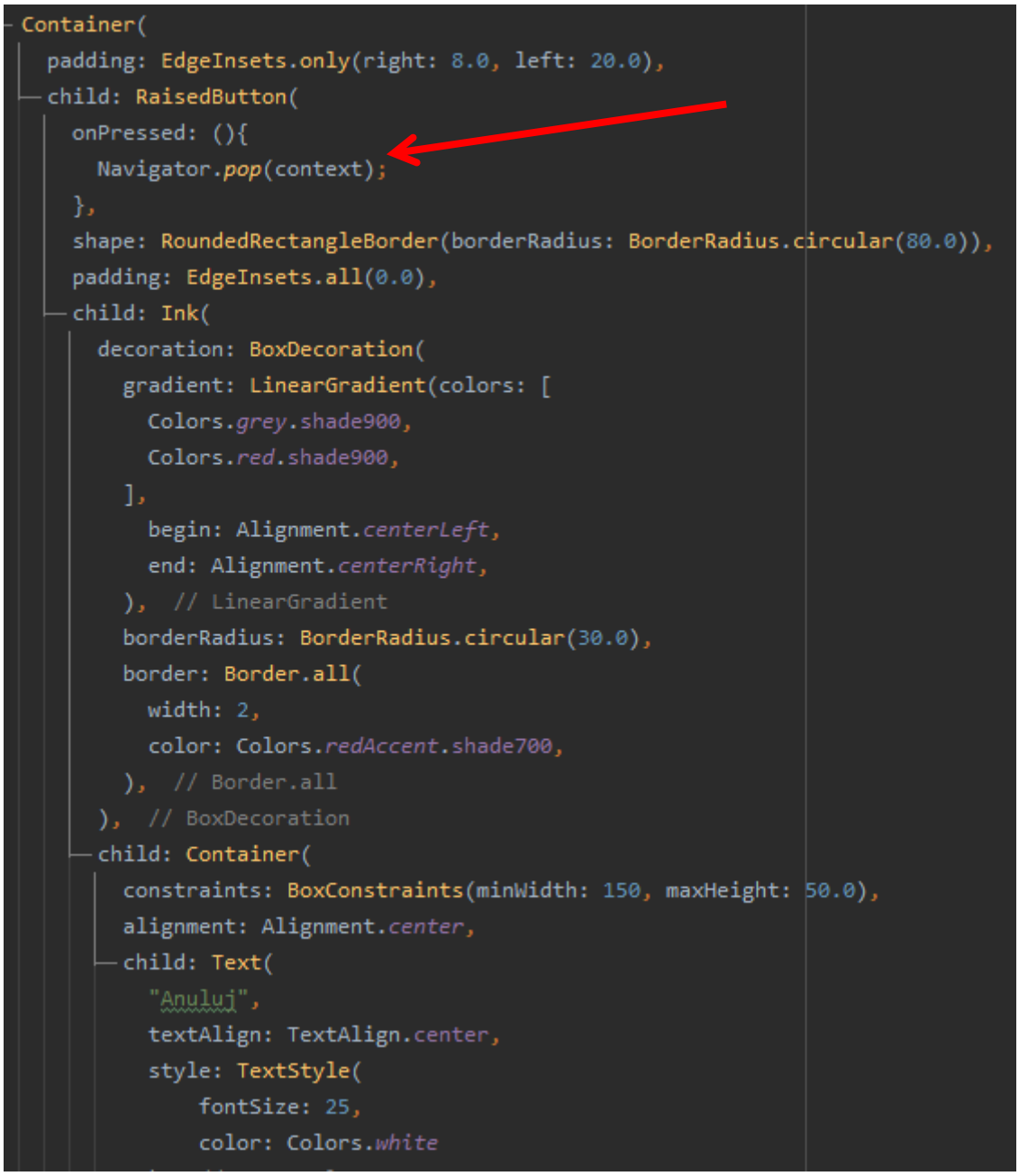
```dart
bool test = await checkAlarmExist(Alarm(Alarm_DateTime: date_to_add, Alarm_Vibration: vibration_to_add, Alarm_Drzemka: drzemka_to_add, Alarm_isActive: 1,
    Monday: pon, Tuesday: wt, Wednesday: sr, Thursday: czw, Friday: pt, Saturday: sb, Sunday: nd));  // Alarm
print("Bool: "+test.toString());
if(test){
  Navigator.push(context, MaterialPageRoute(builder: (context)=>MyHomePage(0)));
}
else{
  insert_to_DB(Alarm(Alarm_DateTime: date_to_add, Alarm_Vibration: vibration_to_add, Alarm_Drzemka: drzemka_to_add, Alarm_isActive: 1,
      Monday: pon, Tuesday: wt, Wednesday: sr, Thursday: czw, Friday: pt, Saturday: sb, Sunday: nd));  // Alarm

  Navigator.push(context, MaterialPageRoute(builder: (context)=>MyHomePage(0)));
}
```

RysNA.2f1

6. Przycisk służący do przejścia z powrotem na stronę główną. Jego działanie zostało zaznaczone czerwoną strzałką, reszta obrazka RysNA.2h przedstawia

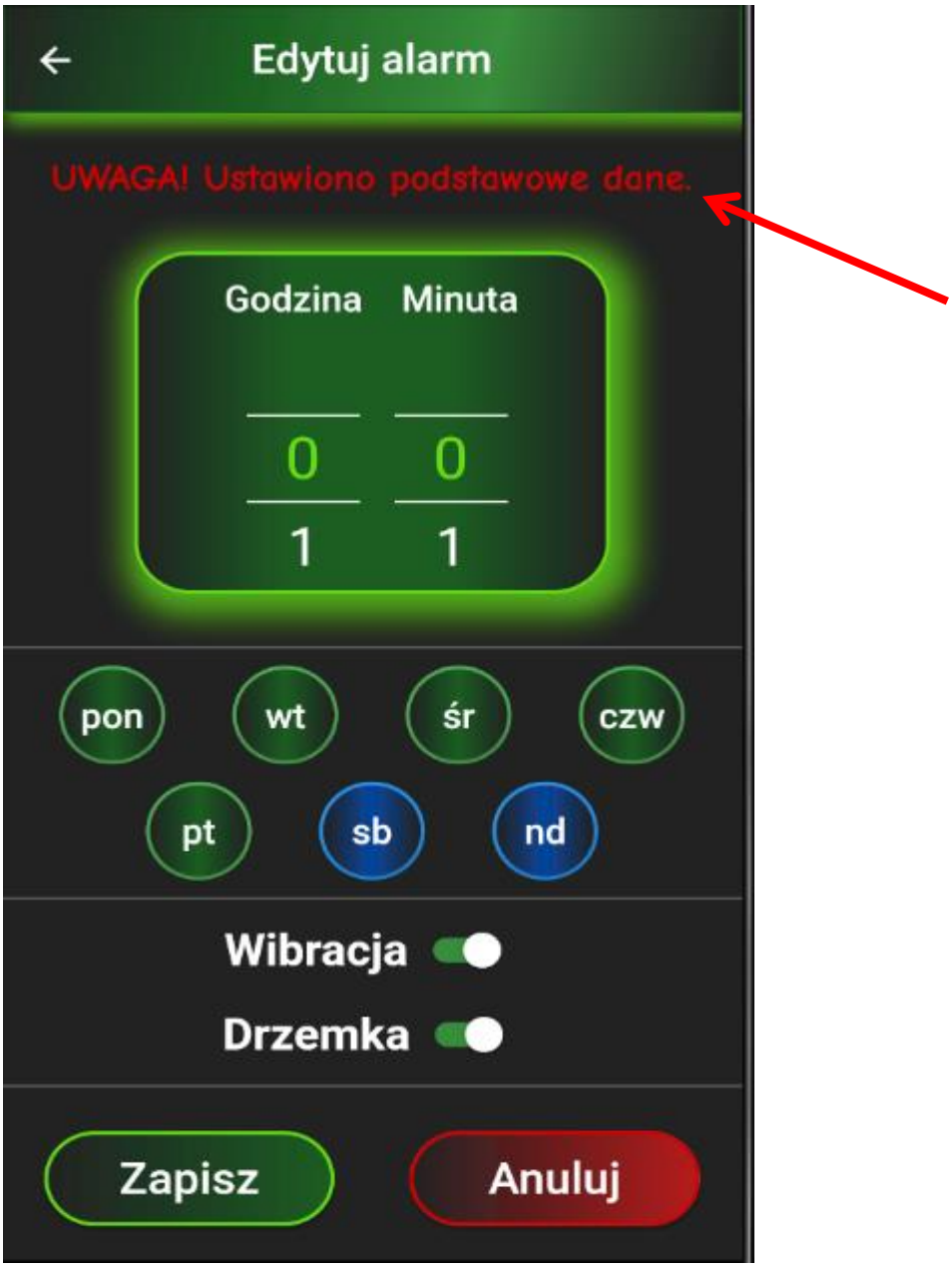
wygląd kodu. Jego działanie jest takie same jak elementu znajdującego się pod numerem 1.



RysNA.2h

2.2.2 EditAlarm.dart

Plik EditAlarm.dart nie różni się wiele od klasy z dodaniem nowego alarmu. Jedyna różnica jest taka, że jego wartości zostały ustawione na podstawowe tzn. jeżeli użytkownik, kliknie zapisz bez wybrania nowych wartości do aktualizacji wczytane zostaną bazowe. Na obrazku RysEA.2a został zaznaczony strzałką wygląd komunikatu ostrzegającego użytkownika przed niechcianym zapisaniem niepoprawnych danych. Przycisk *„anuluj”* działa na zasadzie powrotu do strony głównej.



RysEA.2a

2.3 Stoper.dart

Stoper znajduje się w klasie Stoper.dart i jest wywołany w klasie main.dart.

Podgląd na cały stoper:

1. Panel wyświetlający odmierzany czas. Jego wartość to początkowa to 00:00:00 zainicjowana ziemnną StoperTime (RysSto.2a):

```
20    String StoperTime = "00:00:00";
```

RysSto.2a

W momencie uruchomienia stopera pojawia się dodatkowa para zer reprezentująca milisekundy (RysSto.2b):

RysSto.2b

Do prawidłowego wyświetlania czasu użyłem funkcji *keepruning()* oraz *starttimer()*, które zostały zaprezentowane na rysunku RysSto.2c.

```
26  void keepruning() {
27    if (swatch.isRunning) {
28      starttimer();
29    }
30    if (this.mounted){
31    setState(() {
32      StoperTime = swatch.elapsed.inHours.toString().padLeft(2, "0") +
33          ":" +
34          (swatch.elapsed.inMinutes % 60).toString().padLeft(2, "0") +
35          ":" +
36          (swatch.elapsed.inSeconds % 60).toString().padLeft(2, "0")+
37          ":" +
38          (swatch.elapsed.inMilliseconds % 100).toString().padLeft(2, "0");
39    });
40    }
41  }
42
43  void starttimer() {
44    Timer(dur, keepruning);
45  }
```

RysSto.2c.

Czas odmierzania został ustawiony na **1 milisekundę**.

2. Panel z pomiarami. Jest to obiekt, który przechowuje 10 wczytanych przez użytkownika pomiarów za pomocą przycisku „Pomiar”. Funkcja służąca do wyświetlania pomiarów to *showPartTime()* zaprezentowana na RysSto.2d.

```
showPartTime() {
  widget.x[9] = widget.x[8];
  widget.x[8] = widget.x[7];
  widget.x[7] = widget.x[6];
  widget.x[6] = widget.x[5];
  widget.x[5] = widget.x[4];
  widget.x[4] = widget.x[3];

  widget.x[3] = widget.x[2];

  widget.x[2] = widget.x[1];

  widget.x[1] = widget.x[0];

  widget.x[0] = swatch.elapsed.inHours.toString().padLeft(2, "0") +
      ":" +
      (swatch.elapsed.inMinutes % 60).toString().padLeft(2, "0") +
      ":" +
      (swatch.elapsed.inSeconds % 60).toString().padLeft(2, "0")+
      ":" +
      (swatch.elapsed.inMilliseconds % 100).toString().padLeft(2, "0");

  print("Pomiar: "+widget.x[0]);

  if(widget.x[9]!=null) isButtonBlocked=true;
  setState(() {});

}
```

RysSto.2d

W momencie kiedy jest już 10 wczytanych pomiarów, przycisk służący do ich wczytywania zostaje zablokowany w celu zapobiegnięcia przepełnienia listy. Wygląd został zaimplementowany przy pomocy generowanej listy. Widok kodu dla listy pomiarów (RysSto.2e):

```
children: List.generate(
  10,
      (index) => Text(
    (widget.x.elementAt(index) == null ||
        widget.x.elementAt(index) == ""
        ? "Pomiar "+(index+1).toString()+'\n'+"----------------------"
        : "t"+(10-index).toString()+" - "+widget.x.elementAt(index)+'\n'
        textAlign: TextAlign.center,

        style: GoogleFonts.comicNeue(
          fontWeight: FontWeight.w600,
          color: Colors.white,
          fontSize: 20,
```

RysSto.2e

3. Przycisk „Pomiar” – ten element służy do zrobienie pomiarów w stoperze. Działa, gdy stoper jest uruchomiony lub lista pomiarów jest niezapełniona (w innym wypadku zwróci błąd niewidoczny dla użytkownika). Ponownie kod każdego guzika wygląda niemal identycznie. Przykładowy kod guzika (RysSto.2f):

```
RaisedButton(
  onPressed: (){isrunning ? {isButtonBlocked ? print("Osiagnieto limit pomiarów!")
      :{stop2 ? null : showPartTime()}} : print("Stoper nie pracuje!"); },
  shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(80.0)),
  padding: EdgeInsets.all(3.0),
  color: isButtonBlocked ? Colors.black : Colors.green.shade900,
  child: Ink(
    decoration: BoxDecoration(
      gradient: LinearGradient(colors: [Colors.grey.shade900,
        Colors.green.shade900,],
        begin: Alignment.centerLeft,
        end: Alignment.centerRight,
      ),  // LinearGradient
      borderRadius: BorderRadius.circular(30.0),
      border: Border.all(
        width: 2,
        color: Colors.lightGreenAccent.shade700,
      ),  // Border.all
    ),  // BoxDecoration
    child: Container(
      constraints: BoxConstraints(minWidth: 130, maxHeight: 45.0),
      alignment: Alignment.center,
      child: Text(
        "Pomiar",
        textAlign: TextAlign.center,
        style: GoogleFonts.comicNeue(
          fontWeight: FontWeight.w600,
          color: Colors.white,
          fontSize: 25,
        ),
      ),  // Text
```

RysSto.2f

W guziku parametr onPressed wygląda ciekawie ze względu na to, że zastosowałem w nim podwójnie zagnieżdżoną funkcję warunkową, która działa na zasadzie: **x ? {akcja gdy x = *true* } : {akcja gdy x = *false*}**. Pozostałe guziki różnią się jedynie kolorem oraz funkcją onPressed.

4. Przycisk „Stop” – jest odpowiedzialny za zatrzymanie działania stopera. W momencie kiedy jest naciśnięty możemy zresetować stoper przyciskiem „Reset”. Gdy chcemy go wystartować z poziomu zatrzymanego czasu, wystarczy wcisnąć ponownie przycisk „Start”.

5. Przycisk „Reset” – w swoim ciele ma zaimplementowaną funkcję *resetStoper()* (RysSto.2g)

```
void resetStoper() {
  setState(() {
    stop2 = false;
    reset2 = true;
  });
  StoperTime = "00:00:00";
  isButtonBlocked=false;
  for(int i=0;i<10;i++){
    widget.x[i] = null;
  }
  swatch.reset();
}
```

RysSto.2g

Funkcja resetuje cały stan stopera i listę pomiarów. Drugą różnicą (podobnie jak w przypadku przycisku „Stop”) jest kolor przycisku. Kod znajduje się na RysSto.2h.

```
RaisedButton(
  onPressed: stop2 ? null : stopStoper,
  shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(80.0)),
  padding: EdgeInsets.all(2.0),
  color: Colors.red,
  child: Ink(
    decoration: BoxDecoration(
      gradient: LinearGradient(colors: [Colors.grey.shade900,
        Colors.red.shade900,],
        begin: Alignment.centerLeft,
        end: Alignment.centerRight,
      ),  // LinearGradient
```

RysSto.2h

6. Przycisk „Start” – służy do uruchomienia lub wznowienia działania stopera. Zabezpieczenie przeciwko kilkukrotnym startowaniu tego przycisku wygląda następująco (RysSto.2i i RysSto.2j):

```
onPressed: (){isrunning ? print("Stoper jest uruchomiony!"): startStoper();},
```

RysSto.2i

Przycisk w swoim ciele posiada funkcję, która jest odpowiedzialna za uruchomienie odliczania czasu. Jest to funkcja *void startStoper()* (RysSto.2j):

```
void startStoper() {
  isrunning = true;
  if (this.mounted){
    setState(() {
      stop2 = false;
      reset2 = true;
    });
  }
  swatch.start();
  starttimer();
}
```

RysSto.2j

Podsumowując jeżeli chcemy zresetować stoper do wartości początkowych, należy najpierw zastopować stoper. Pozostałe akcje powinny być intuicyjne dla użytkownika. W momencie przełączenia się na inną kartę (Budzik.dart lub Minutnik.dart) stoper przestanie działać gdyż w ten sposób uniknąłem błędów w aplikacji. Nieoczekiwane akcje na przyciskach również zostały zabezpieczone aby aplikacja działała prawidłowo.

2.4 Minutnik.dart

Minutnik znajduje się w klasie Minutnik.dart i jest wywołany w klasie main.dart.

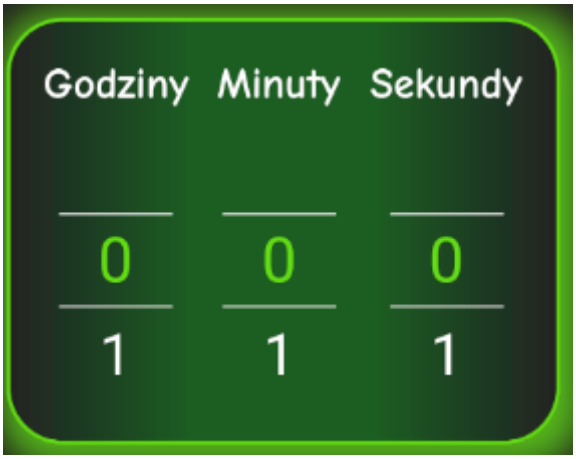
Podgląd na cały minutnik:

1. Panel z możliwością wyboru godziny. Został stworzony przy pomocy gotowego elementu numberpicker, który zaimportowałem z (RysMinu.2a):

```
6    import 'package:numberpicker/numberpicker.dart';
```

RysMinu.2a

który wcześniej dodałem do pliku pubspec.yaml (wersja numberpickera: ^1.2.1). Panel umożliwia użytkownikowi wybranie interesującego go czasu, gdzie godziny są w zakresie od 0 do 23, minuty od 0 do 59, sekundy od 0 do 59. Aktualne wybrane wartości mają kolor zielony (zostało to zaprezentowane poniżej na RysMinu.2b):



RysMinu.2b

Każdy kolumna w tym elemencie różni tylko się danymi tekstowymi lub zakresem. Żeby sztucznie nie wydłużać dokumentacji przedstawię kod dla numberpickera dla kolumny „Minuty” wygląda następująco (RysMinu.2c):

```
NumberPicker.integer(
    decoration: BoxDecoration(
      border: Border(
        top: BorderSide(width: 1.0, color: Color(0xFFFFFFFFFF)),
        bottom: BorderSide(width: 1.0, color: Color(0xFFFFFFFFFF)),
      ),  // Border
    ),  // BoxDecoration
  selectedTextStyle: TextStyle(
      color: Colors.lightGreenAccent.shade700,
      fontSize: 33
  ),  // TextStyle
    textStyle: TextStyle(
        color: Colors.white,
        fontSize: 30
    ),  // TextStyle
    initialValue: min,
    minValue: 0,
    maxValue: 59,
    listViewWidth: 60.0,
    onChanged: (val) {
      setState(() {
        min = val;
      });
    }
    )  // NumberPicker.integer
```

RysMinu.2c

2. Jest to miejsce gdzie wyświetla się odmierzany czas. Odpowiednio od lewej strony są wyświetlane godziny, minuty i sekundy.

```
Container(
  padding: EdgeInsets.only(bottom: 25),
  child: Container(
    child: Text(
      timeToDisplay,
        style: GoogleFonts.comicNeue(
          fontWeight: FontWeight.w900,
          color: Colors.white,
          fontSize: 50,
        )
    ),  // Text
  ),  // Container
),  // Container
```

RysMinu.2d

Rysunek RysMinu.2d przedstawia wygląd źródła tego elementu (zmienna timeToDisplay początkowo jest zainicjowana wartością „00:00:00”). W kodzie do poprawnego wyświetlania i startowania minutnika napisałem funkcję *void start()* (znajduje się od 31 do 186 linii w pliku Minutnik.dart). Funkcja *void resume()* znajduje się w kodzie w liniach od 196 do 347, działa na podobnej zasadzie co funkcja void start() z tym, że służy do wznawiania działania minutnika.

3. Przycisk „wznów” – ten element służy do wznawiania działania minutnika, w momencie kiedy zostanie zatrzymany. Kod każdego guzika wygląda niemal identycznie. Przykładowy kod guzika (RysMinu.2e):

```
Container(
  padding: EdgeInsets.only(top: 10),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: <Widget>[
      Container(
        child: RaisedButton(
          onPressed: stopped ? resume : null,
          shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(80.0)),
          padding: EdgeInsets.all(3.0),
          color: Colors.green.shade900,
          child: Ink(
            decoration: BoxDecoration(
              gradient: LinearGradient(colors: [Colors.grey.shade900,
                Colors.green.shade900,],
                begin: Alignment.centerLeft,
                end: Alignment.centerRight,
              ),  // LinearGradient
              borderRadius: BorderRadius.circular(30.0),
              border: Border.all(
                width: 2,
                color: Colors.lightGreenAccent.shade700,
              ),  // Border.all
            ),  // BoxDecoration
            child: Container(
              constraints: BoxConstraints(minWidth: 150, maxHeight: 45.0),
              alignment: Alignment.center,
              child: Text(
                "Wznów",
                textAlign: TextAlign.center,
                style: GoogleFonts.comicNeue(
                  fontWeight: FontWeight.w600,
                  color: Colors.white,
                  fontSize: 26,
                ),
```

RysMinu.2e

4. Przycisk „start” – służy jak sama nazwa wskazuje do startowania odliczania czasu. W momencie kiedy czas jest 00:00:00 przycisk również zadziała, ale ze względu na to, że nie ma co odliczać sam się zatrzyma. Różni się zastosowaną funkcją onPressed od pozostałych guzików (RysMinu.2f)

```
588                                    onPressed: started ? start : null,
```

RysMinu.2f

5. Przycisk „stop” – służy do zastopowania działającego minutnika, w momencie kiedy minutnik „nie pracuje” nie możemy go użyć (jest wyłączony). Różni się kolorem i zastosowaną metodą od powyższych przycisków (RysMinu.2g).

```
onPressed: stopped ? null : stop,
shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(80.0)),
padding: EdgeInsets.all(1.0),
color: Colors.red,
child: Ink(
  decoration: BoxDecoration(
    gradient: LinearGradient(colors: [Colors.grey.shade900,
      Colors.red.shade900,],
      begin: Alignment.centerLeft,
      end: Alignment.centerRight,
    ),  // LinearGradient
```

RysMinu.2g

Podsumowując jeżeli chcemy zresetować minutnik do wartości początkowych, należy ustawić wszystkie wartości numberpickera na 0. W momencie przełączenia się na inną kartę (Budzik.dart lub Stoper.dart) minutnik przestanie działać gdyż w ten sposób uniknąłem błędów w aplikacji. Nieoczekiwane akcje na przyciskach również zostały zabezpieczone aby aplikacja działała prawidłowo.