

Android – Uprawnienia i Powiadomienia

1. Uprawnienia

Za pomocą uprawnień użytkownicy decydują, które aplikacje mogą korzystać z określonych funkcji, takich jak dostęp do aparatu, lokalizacji, kontaktów czy powiadomień.

W systemie android możemy podzielić uprawnienia na

-**normalne(Normal Permissions)**: nie stanowiące ryzyka dla prywatności.

-**niebezpieczne(Dangerous Permissions)**: mające wpływ na prywatność użytkownika lub integralność systemu, aplikacje muszą jawnie żądać dostępu do takich uprawnień w trakcie działania aplikacji.

-**Signature Permissions**: Uprawnienia do podpisu w systemie Android są używane do udzielania uprawnień tylko wtedy, gdy aplikacja żądająca jest podpisana przy użyciu tego samego certyfikatu, co aplikacja, która zadeklarowała uprawnienie

-**Uprawnienia Systemowe (System Permissions)**: przyznawane aplikacjom systemowym.

Każda aplikacja musi deklarować uprawnienia, których wymaga, w pliku AndroidManifest.xml. Deklaracja ta informuje system i użytkownika o wymaganych uprawnieniach.

Normalne uprawnienie:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Niebezpieczne uprawnienie:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Od Androida 6.0 (Marshmallow, API poziom 23), uprawnienia niebezpieczne muszą być obsługiwane w czasie działania. Aplikacje muszą sprawdzić, czy mają odpowiednie uprawnienia przed wykonaniem operacji, które ich wymagają, i jeśli nie, żądać ich od użytkownika.

Np. jeżeli chcielibyśmy napisać aplikację umożliwiającą wysłanie powiadomień do użytkownika z akcją ustawienia alarmu potrzebowalibyśmy następujących uprawnień:

normalne:

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM"/>
```

niebezpieczne:

```
<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
```

2. Powiadomienia

2.1 Dokumentacja Powiadomień

<https://developer.android.com/develop/ui/views/notifications>

Powiadomienia (ang. Notifications) to wiadomości wysyłane przez aplikacje, które informują użytkownika o ważnych zdarzeniach, nawet gdy aplikacja nie jest aktywnie używana.

Powiadomienia pojawiają się w pasku statusu, na ekranie blokady, a także mogą być widoczne w rozwijanym obszarze powiadomień.

2.2 Kanały powiadomień

Od Androida 8.0 (Oreo, API 26), Google wprowadziło kanały powiadomień jako nową funkcję zarządzania powiadomieniami, aplikacja może mieć osobne kanały dla wiadomości, przypomnień, aktualizacji, itp. a użytkownicy mogą zarządzać ustawieniami powiadomień dla każdego kanału osobno. Kanał tworzymy raz, np. podczas tworzenia aplikacji (onCreate).

```
private fun createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val name = "default"
        val descriptionText = "Kanał domyślny"
        val importance = NotificationManager.IMPORTANCE_HIGH
        val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
            description = descriptionText
        }

        val notificationManager: NotificationManager =
            getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        notificationManager.createNotificationChannel(channel)}}

```

NotificationChannel: Klasa reprezentująca kanał powiadomień.

CHANNEL_ID: Unikalny identyfikator kanału. Powinien być stałą wartością, np. w klasie aktywności:

```
private val CHANNEL_ID = "default_channel_id"
```

Użycie funkcji w onCreate:

```
companion object {
    const val CHANNEL_ID = "default_channel"
    const val NOTIFICATION_ID = 1
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    createNotificationChannel()
    setContent {
        NotificationApp()
    }
}

```

2.3 Dokumentacja Powiadomień - szczegóły

name: Nazwa kanału.

description: opis kanału.

importance: Poziom ważności powiadomień w kanale.

getSystemService(Context.NOTIFICATION_SERVICE): Pobiera systemowy serwis zarządzający powiadomieniami.

notificationManager.createNotificationChannel(channel): Rejestruje kanał w systemie. Jeśli kanał o tym CHANNEL_ID już istnieje, system go nie nadpisuje, a jedynie aktualizuje jego ustawienia, jeśli zostały zmienione.

Możemy określić poziom ważności powiadomień, co wpływa na ich wygląd i sposób prezentacji:

IMPORTANCE_NONE (0): Powiadomienia nie są wyświetlane.

IMPORTANCE_MIN (1): Powiadomienia nie generują dźwięku ani nie pojawiają się na ekranie blokady.

IMPORTANCE_LOW (2): Powiadomienia nie generują dźwięku, ale pojawiają się w obszarze powiadomień.

IMPORTANCE_DEFAULT (3): Powiadomienia generują dźwięk, ale nie wyświetlają się na ekranie blokady.

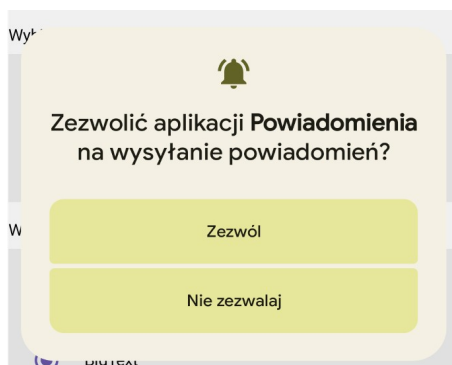
IMPORTANCE_HIGH (4): Powiadomienia generują dźwięk i wyświetlają się na ekranie blokady oraz w formie heads-up (krótkie powiadomienia, które pojawiają się na górze ekranu).

3. Sprawdzenie uprawnień

Od Androida 13 (API poziom 33), aplikacje muszą uzyskać specjalne uprawnienie (POST_NOTIFICATIONS), aby móc wysyłać powiadomienia.

```
var permissionGranted by remember { mutableStateOf(checkNotificationPermission()) }
```

Gdy użytkownik skorzysta z funkcji (wysłanie powiadomienia) aplikacja za pierwszym razem powinna poprosić o odpowiednie uprawnienie.



3.1 Sprawdzenie uprawnień- szczegóły

Funkcja checkNotificationPermission() będzie sprawdzała czy przyznano uprawnienie:

```
private fun checkNotificationPermission(): Boolean {  
    return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {  
        ContextCompat.checkSelfPermission(  
            this,  
            Manifest.permission.POST_NOTIFICATIONS  
        ) == PackageManager.PERMISSION_GRANTED  
    } else {  
        true  
    }  
}
```

Pozwolenie POST_NOTIFICATIONS wymaga sprawdzenia od wersji androida13 (Tiramisu) jeżeli na urządzeniu jest starsza wersja funkcja zwraca true.

Funkcja kompozycyjna będzie wyświetlała i zarządzała komponentami przez Jetpack Compose.

```
@Composable  
fun NotificationApp(){  
    var permissionGranted by remember { mutableStateOf(checkNotificationPermission()) }  
}
```

3.2 Launcher

Do zarządzania wynikami aktywności należy użyć **Activity Result API**.

RememberLauncherForActivityResult to funkcja w Jetpack Compose, która tworzy **launcher** do uruchamiania działań związanych z wynikami aktywności, takich jak żądanie uprawnień, wybieranie plików, umieszczamy w funkcji kompozycyjnej NotificationApp():

```
val requestPermissionLauncher = rememberLauncherForActivityResult(  
    ActivityResultContracts.RequestPermission()  
) { isGranted ->
```

```

        permissionGranted = isGranted
    if (!isGranted) {
        Toast.makeText(
            this,
            "Brak zezwolenia ! Nie można wyświetlić powiadomienia.",
            Toast.LENGTH_SHORT
        ).show()
    }
}

```

Laucher może zostać użyty np. gdy użytkownik kliknie przycisk do wysłania powiadomienia:

```

    Button(onClick = {
        if (permissionGranted) {
            sendSimpleNotification(this, "alarm", "ustaw alarm")
        } else {
            requestPermissionLauncher.launch(Manifest.permission.POST_NOTIFICATIONS)
        }
    }) {
        Text("Wyślij Powiadomienie")
    }
}

```

Używamy `ActivityResultApi` zamiast starej metody `onRequestPermissionsResult()`.

4. Wysyłanie powiadomień

`sendSimpleNotification` jest funkcją, w której trzeba zbudować powiadomienie:

`NotificationCompat.Builder` potrzebuje kontekstu i id kanału.

Za pomocą buildera umieszczamy ikonę, tytuł, wiadomość oraz priorytet powiadomienia.

4.1 Builder

```

@RequiresPermission(Manifest.permission.POST_NOTIFICATIONS)
private fun sendSimpleNotification(context: Context, title: String, message: String) {
    val builder = NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.clock1)
        .setContentTitle(title)
        .setContentText(message)
        .setPriority(NotificationCompat.PRIORITY_HIGH)
    with(NotificationManagerCompat.from(context)) {
        notify(NOTIFICATION_ID, builder.build())
    }
}

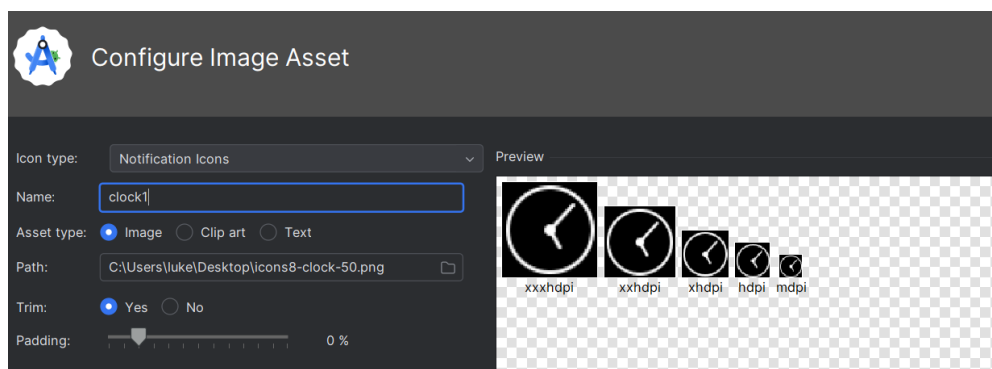
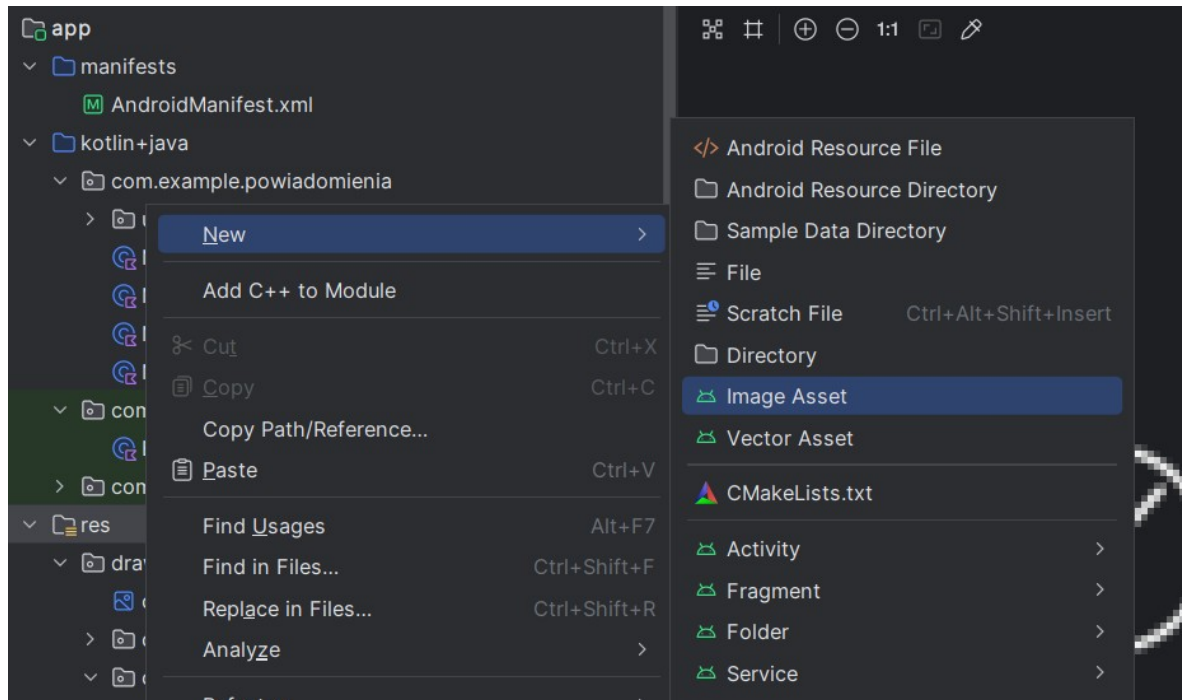
```

`NotificationManagerCompat.from(context)`: Pobiera instancję `NotificationManagerCompat` dla danego kontekstu, zapewniając kompatybilność z różnymi wersjami Androida.

`notify(NOTIFICATION_ID, builder.build())`: Wyświetla powiadomienie z unikalnym identyfikatorem. `NOTIFICATION_ID` zbudowanym za pomocą buildera.

4.1 Obrazki

Ikony do powiadomień można utworzyć za pomocą Image Asset:

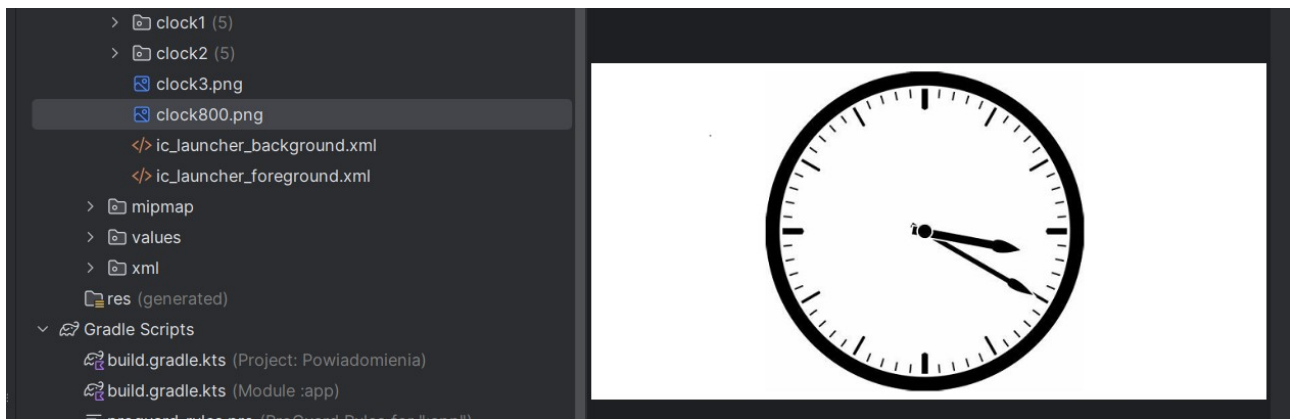


Z tak przygotowaną ikoną jesteśmy w stanie wyświetlić proste powiadomienie.



Drugą opcją jest umieszczenie obrazka w katalogu res/Drawable.

Jeżeli chodzi o ikony powiadomień to Android i tak przerabia je do odpowiedniej odcieni szarości. Jeżeli chcemy użyć dużego obrazka w ustawieniu stylu powiadomienia wtedy najlepiej przygotować obrazek samemu (sprawdza się rozmiar 800x400)



4.2 Intencja

Dodanie intencji ustawienia alarmu w `sendSimpleNotification`:

```
val alarmIntent = Intent(AlarmClock.ACTION_SET_ALARM).apply {
    putExtra(AlarmClock.EXTRA_MESSAGE, "Budzik")
    putExtra(AlarmClock.EXTRA_HOUR, 7)
    putExtra(AlarmClock.EXTRA_MINUTES, 0)
}

val alarmPendingIntent = PendingIntent.getActivity(
    this,
    0,
    alarmIntent,
    PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE
)
```

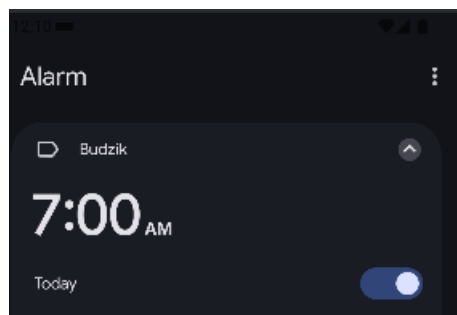
`PendingIntent` pozwala na uruchomienie intencji w przyszłości, nawet jeśli aplikacja nie jest aktywna.

`FLAG_UPDATE_CURRENT`: Jeśli `PendingIntent` już istnieje, aktualizuje jego dane (extras) nowymi wartościami z `alarmIntent`.

`FLAG_IMMUTABLE`: Oznacza, że `PendingIntent` jest niezmienny – nie można go modyfikować po jego utworzeniu.

Możemy dodać akcję w builderze powiadomienia:

```
.addAction(
    R.drawable.clock1,
    "Ustaw nowy Alarm",
    alarmPendingIntent
)
```



Klikając w ustaw nowy alarm w powiadomieniu spowoduje dodanie nowego alarmu z podaną w aplikacji godziną.

5 Wybór obrazków

Wybór ikon/obrazków które możemy pokazać w powiadomieniu możemy umożliwić za pomocą wyświetlenia obrazków na ekranie z rolą RadioButton, oraz dodać prawdziwe RadioButtons.

Należy utworzyć listę dostępnych obrazków:

```
val icons = listOf(
    R.drawable.clock800,
    R.drawable.clock1
)

var selectedIcon by remember { mutableStateOf<Int>(icons[0]) }
```

W kolumnie dla każdego obrazka z listy należy utworzyć wiersze korzystając z listy obrazków:

```
Column(
    modifier = Modifier
        .padding(16.dp)
        .background(Color(0xFF0F0F0F))
) {

    icons.forEach { iconEl ->
        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier
                .fillMaxWidth()
                .background(Color(0xFFE3E3E3))
                .selectable(
                    selected = selectedIcon == iconEl,
                    onClick = { selectedIcon = iconEl },
                    role = Role.RadioButton
                )
            .padding(8.dp)
        ) {
            RadioButton(
                selected = selectedIcon == iconEl,
                onClick = { selectedIcon = iconEl }
            )
            Spacer(modifier = Modifier.width(8.dp))
            Image(
                painter = painterResource(id = iconEl),
                contentDescription = null,
                modifier = Modifier.size(48.dp)
            )
        }
    }
}
```

Cały Wiersz możemy potraktować jako radio button ustawiając role na Role.RadioButton, oraz sprawdzeniu czy wybrany obrazek jest równy elementowi w wierszu, oraz implementując zdarzenie – zmianie selectionIcon na zaznaczoną w onClick:

```
.selectable(
    selected = selectedIcon == iconEl,
    onClick = { selectedIcon = iconEl },
    role = Role.RadioButton
)
```

Możemy też dodać RadioButton na takiej samej zasadzie:

```
RadioButton(  
    selected = selectedIcon == iconEl,  
    onClick = { selectedIcon = iconEl }  
)
```

W kolumnie należy też dodać wiersz z przyciskiem do wysłania powiadomienia – tak aby był pod spodem obrazków lub nad.

Wybrany obraz można by przekazać do funkcji show:

```
private fun sendSimpleNotification(context: Context, title: String, message: String, @DrawableRes selectedIconId: Int)
```

Na przyciśnięcie przycisku:

```
sendSimpleNotification(context, "alarm", "ustaw alarm",selectedIcon)
```

W builderze powiadomienia:

```
.setSmallIcon(iconResId)
```

Jeżeli chcielibyśmy użyć dużej ikony, należałoby przekonwertować obraz na Bitmapę:

```
val bitMapa= BitmapFactory.decodeResource(resources, selectedIconId)
```

i użyć w builderze:

```
.setLargeIcon(bitMapa)
```

<https://developer.android.com/develop/ui/views/notifications/expanded>

6.Style powiadomień

W Androidzie powiadomienia mogą mieć różne style, które pozwalają na dostosowanie wyglądu i funkcji.

-**BigTextStyle:**Umożliwia wyświetlanie dłuższej wiadomości tekstowej

-**BigPictureStyle:**Umożliwia wyświetlenie dużego obrazu

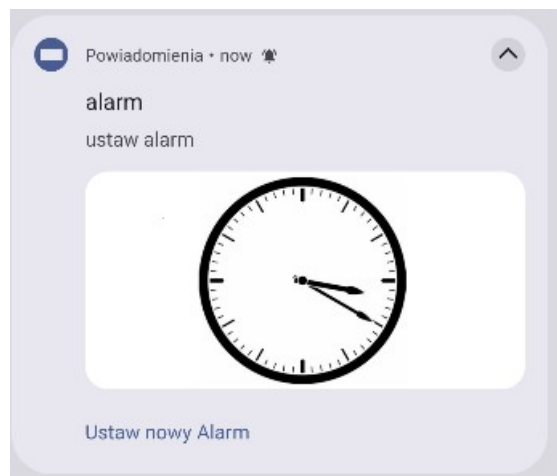
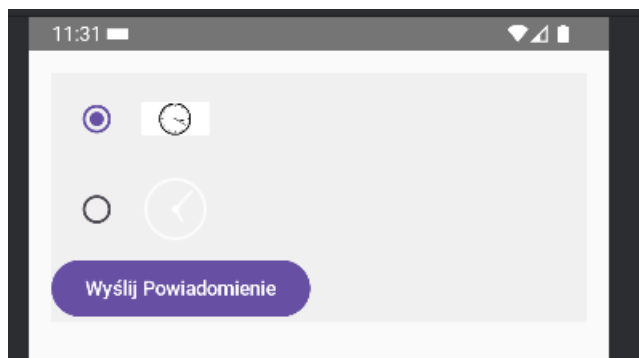
-**InboxStyle:**-Wyświetla listę kilku krótkich wiadomości lub informacji, która jest widoczna po rozszerzeniu powiadomienia.

-**MessagingStyle:**Pozwala na wyświetlanie powiadomień w formie wątku wiadomości z rozmówcą, z obsługą avatarów.

-**MediaStyle:**Dodaje kontrolki odtwarzania multimediiów (np. Play, Pause, Next).

Przykład stylu z dużym obrazkiem w builderze powiadomienia:

```
.setStyle(  
    NotificationCompat.BigPictureStyle()  
        .bigPicture(bitMapa)  
)
```



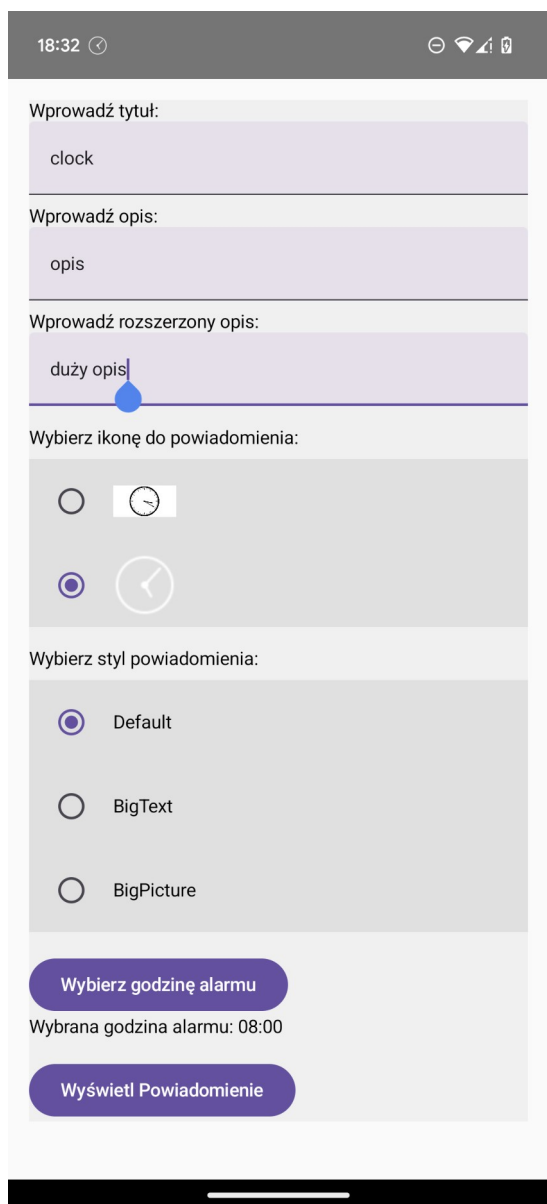
Zadanie:

Stwórz aplikację do tworzenia powiadomień.

- wymagaj tytułu
- pozwól umieścić opis
- pozwól umieścić poszerzony opis
- pozwól zamieścić obraz (wybór spośród dwóch z resources)
- pozwól zmienić styl (domyślny – tytuł i opis, bigText – duży opis dodany do powiadomienia, bigPicture – styl z dużym obrazkiem jak w przykładzie.
- w powiadomieniu zamieść akcję ustawienia alarmu, dane do ustawienia alarmu pobierz od użytkownika w aplikacji.

Można zastosować: <https://developer.android.com/develop/ui/compose/components/time-pickers>

- finalne powiadomienie powinno pojawić się po przyciśnięciu przycisku.



18:32

Wprowadź tytuł:

clock



Wprowadź opis:

opis

Wprowadź rozszerzony opis:

duży opis

Wybierz ikonę do powiadomienia:

☐  ☒ 

Wybierz styl powiadomienia:

☒ Default

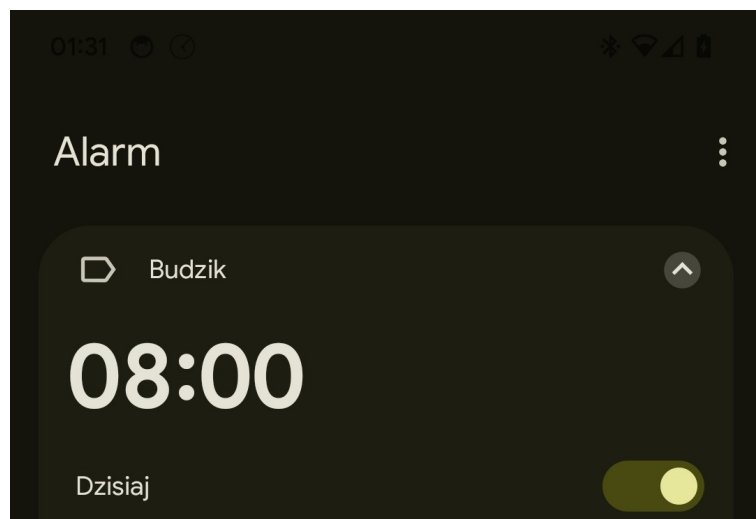
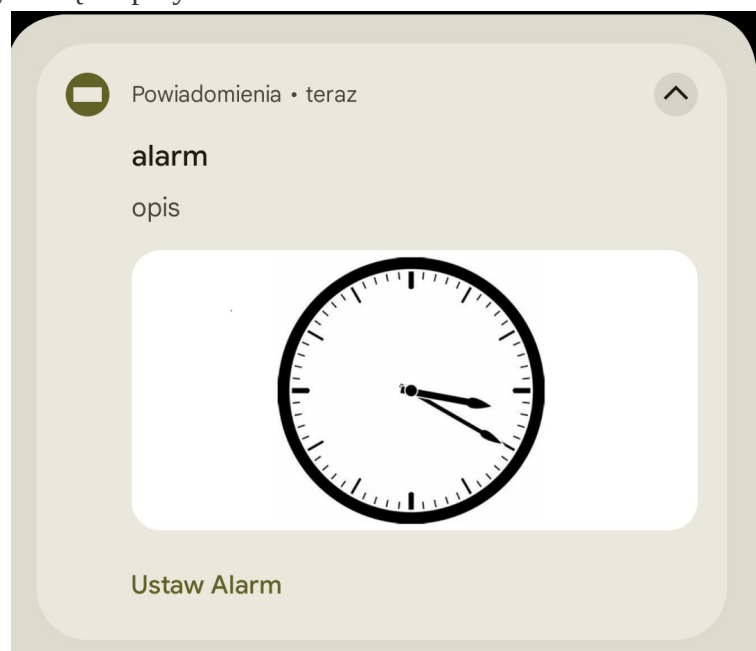
☐ BigText

☐ BigPicture

Wybierz godzinę alarmu

Wybrana godzina alarmu: 08:00

Wyświetl Powiadomienie



Importy z przykładowej aplikacji:

```
import android.Manifest
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.app.TimePickerDialog
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.os.Build
import android.os.Bundle
import android.provider.AlarmClock
import android.util.Log
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.compose.setContent
import androidx.activity.result.contract.ActivityResultContracts
import androidx.annotation.DrawableRes
import androidx.annotation.RequiresPermission
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.selection.selectable
import androidx.compose.material3.Button
import androidx.compose.material3.RadioButton
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.semantics.Role
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.input.TextFieldValue
import androidx.compose.ui.unit.dp
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import androidx.core.content.ContextCompat
import java.util.Calendar
```