Kurs programowania aplikacji mobilnych
Sprawozdanie: Aplikacja wspierająca pielęgnację roślin

Data: 05/06/2022
Ewa Namysł
Informatyka stosowana, III rok
link do repozytorium: https://github.com/namysl/domowy-ogrodnik-android-app

1. Założenia:

Aplikacja ma na celu wesprzeć użytkownika w utrzymaniu i pielęgnacji roślin doniczkowych dzięki przypomnieniom o najważniejszych czynnościach, np. podlewaniu i przesadzaniu.

Podstawowymi funkcjonalnościami aplikacji są:

A) dodawanie roślin do bazy danych wraz z nazwą, zdjęciem i opcjonalnym opisem,
B) galeria dodanych roślin i powiększanie ich zdjęć,
C) usuwanie roślin z galerii,
D) dodawanie przypomnień jednorazowych,
E) dodawanie przypomnień cyklicznych,
F) wyświetlanie listy przypomnień,
G) usuwanie przypomnień,
H) ostrzeżenia typu popup o możliwym przelaniu rośliny w przypadku ustawienia codziennych powiadomień o podlewaniu,
I) wyświetlanie ile dodano roślin na głównym ekranie.

Aplikacja będzie tworzona w języku Kotlin na platformę Android.

2. Funkcjonalności obecne w tej wersji aplikacji:

Ad A) Użytkownik dodaje zdjęcie rośliny z galerii urządzenia lub robi zdjęcie bezpośrednio z poziomu aplikacji. Dostępne są dwa pola tekstowe. Pierwsze pole *Nazwa* musi zostać wypełnione przez użytkownika, natomiast drugie pole *Opis* jest opcjonalne. Aplikacja została zabezpieczona przed zapisaniem pustego pola *Nazwa* i brakiem zdjęcia. W takim przypadku wyskakuje stosowna informacja.

Ad B) Użytkownik może przeglądać dodane rośliny w wertykalnej, przewijalnej galerii (zakładka *Twoje rośliny)*. Przy każdej roślinie znajdują się dwa guziki – *Usuń* oraz *Przypomnij*, które pozwalają kolejno na usuwanie oraz dodawanie przypomnień dla konkretnej rośliny. Ponadto klikając w miniaturkę zdjęcia, możemy zobaczyć jej powiększenie.

Ad C) Użytkownik może usuwać rośliny w galerii klikając przycisk *Usuń.*

Ad D/E) Po kliknięciu guzika *Przypomnij* otwierana jest nowa aktywność, w której użytkownik precyzuje datę, godzinę oraz czynność, o której ma przypominać powiadomienie.
Klikając na pole *Data* otwierany jest kalendarz, natomiast po kliknięciu *Czas* otwarty zostanie zegar dwudziestoczterogodzinny. Czynność wybierana jest z listy rozwijalnej z predefiniowanymi czynnościami (np. podlewanie, przesadzanie etc). W aktywności występuje też lista rozwijalna z której użytkownik może wybrać czy przypomnienie ma być cykliczne. Jeśli użytkownik chce powiadomienie jednorazowe – wybiera *powtarzanie wyłączone*. W przypadku powiadomień

cyklicznych do wyboru jest kilka opcji: *powtarzaj codziennie, powtarzaj co tydzień, powtarzaj co dwa tygodnie* etc.

Ad F/G) Użytkownik może zobaczyć dodane powiadomienia w zakładce *Twoje przypomnienia.* Znajduje się tam zdjęcie i nazwa rośliny, której dotyczy przypomnienie, oraz czynność, data, godzina oraz częstotliwość powiadomień. Obok każdego z nich znajduje się przycisk *Usuń,* który usuwa powiadomienie.

Ad H) Przy dodawaniu przypomnień o podlewaniu i wyborze *powtarzaj codziennie*, wyskakuje popup z ostrzeżeniem o szkodliwości zbyt częstego podlewania i pytaniem czy na pewno dodać tego typu przypomnienie. Po kliknięciu *Anuluj*, użytkownik może zmienić wprowadzane dane, natomiast po kliknięciu *Tak* powiadomienie zostaje dodane.

Ad I) Na głównym ekranie informacji widnieje animowany licznik, który informuje użytkownika ile roślin dodał już do aplikacji.

3. Harmonogram prac:

Tydzień 1	• Wstępny projekt GUI aplikacji, • pierwsza tabela w bazie danych dla roślin.
Tydzień 2	• Funkcja dodawania roślin do bazy danych, • galeria roślin i wczytywanie danych z bazy.
Tydzień 3	• Rozbudowa tabeli roślin o dodatkowe pole, • funkcja usuwania roślin, • powiększanie zdjęć w galerii, • licznik roślin na ekranie powitalnym aplikacji, • poprawki GUI, • dodanie animacji w interfejsie.
Tydzień 4	• Stworzenie nowej tabeli w bazie danych dla przypomnień, • dodawanie i usuwanie przypomnień, • GUI zakładki *Twoje przypomnienia* oraz aktywności dodawania przypomnień
Tydzień 5	• Wywoływanie przypomnień jednorazowych, • wywoływanie przypomnień cyklicznych, • anulowanie przypomnień, • ostrzeżenia o możliwym przelaniu roślin,

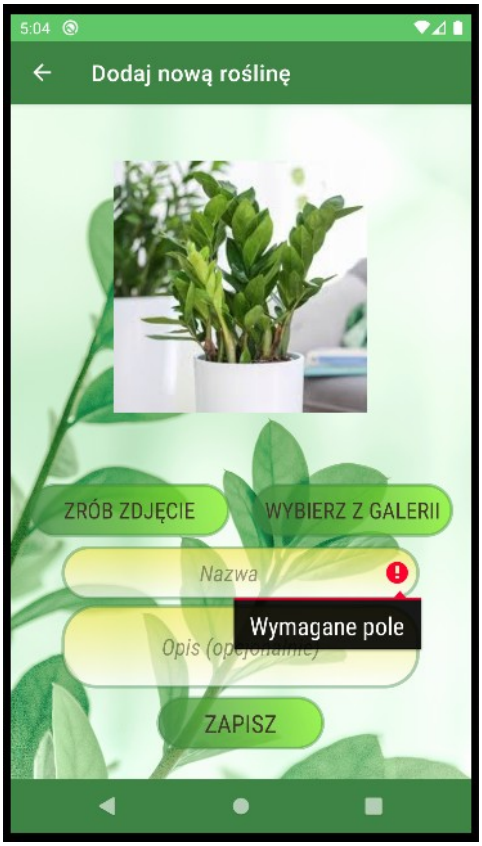
4. Wygląd i GUI aplikacji:

Strona główna aplikacji

Galeria roślin (zakładka *Twoje rośliny*)

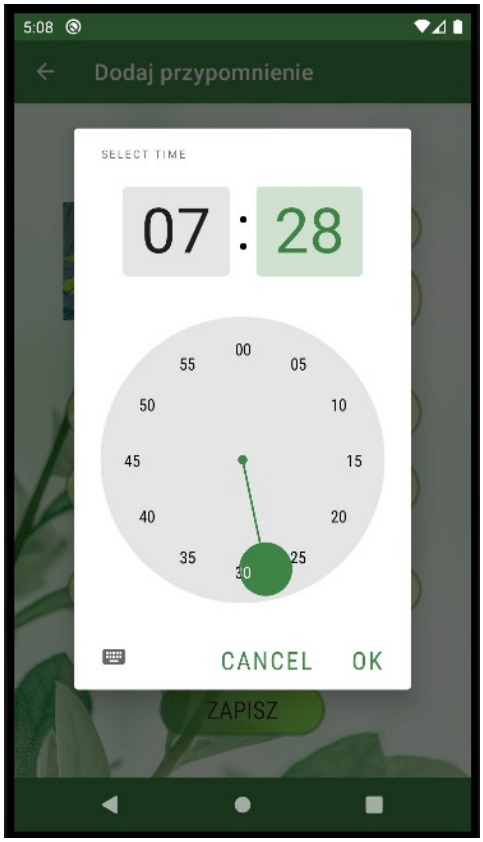
Powiększanie zdjęć

Po usunięciu wybranej rośliny



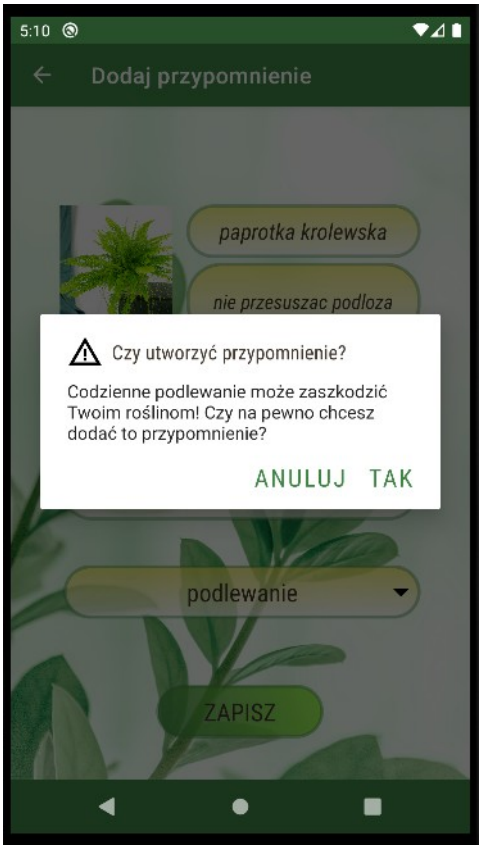
Dodawanie rośliny - zabezpieczenie przed pustym polem

Dodawanie przypomnień

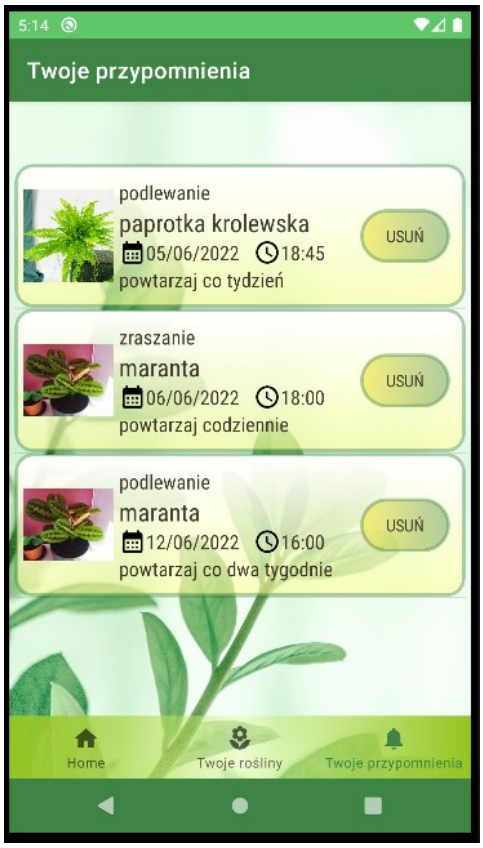


Dodawanie przypomnień – zegar dwudziestoczterogodzinny

Dodawanie przypomnień – lista rozwijalna czynności



Dodawanie przypomnień – ostrzeżenia przed przelaniem



Lista przypomnień (zakładka *Twoje przypomnienia)*

Powiadomienie popup

Nieodczytane powiadomienie

Ponadto elementy GUI są animowane – animacje oraz działanie aplikacji można zobaczyć na filmie załączonym do sprawozdania na Classroomie (*aplikacja_demo.mp4*).

5. Język aplikacji oraz dodatkowe biblioteki:

Aplikacja napisana została na platformę Android w Kotlinie. Do stworzenia lokalnej bazy danych dla roślin i przypomnień wykorzystano bibliotekę Room. Do tworzenia i wyświetlania powiadomień użyto funkcji Androida, m.in. AlarmManager oraz PendingIntent.

6. Opis działania najważniejszych funkcjonalności oraz fragmenty kodu:

Każda zakładka z paska nawigacyjnego ma własny plik xml ze zdefiniowanymi polami tekstowymi, guzikami i innymi niezbędnymi elementami GUI.

Następnie zakładki definiowane są w pliku *bottom_nav_menu.xml* oraz dodawane do *AppBarConfiguration* w *MainActivity.kt* jak poniżej:

```
val appBarConfiguration = AppBarConfiguration(
    setOf(R.id.navigation_home, R.id.navigation_plants, R.id.navigation_reminders)
)
```

Każda z zakładek musi wywołać metodę *onCreateView*, która rozpoczyna proces tworzenia widoku. Tam wywołujemy i operujemy na elementach, które zdefiniowaliśmy w plikach xml.

Poza standardową aktywnością *MainAcivity.kt*, aplikacja posiada również dwie inne aktywności – *AddPlantActivity.kt* oraz *AddReminderActivity.kt.* Pierwsza odpowiada za dodawanie roślin, druga za dodawanie nowych przypomnień. Należy je dodać do *AndroidManifest.xml:*

```
<activity android:name=".AddPlantActivity"
    android:parentActivityName=".MainActivity"
    android:label="Dodaj nową roślinę" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
<activity android:name=".AddReminderActivity"
    android:parentActivityName=".MainActivity"
    android:label="Dodaj przypomnienie" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
<receiver
    android:name=".AlarmReceiver"
    android:enabled="true"
    android:exported="false" />
```

W manifeście deklarujemy też *AlarmReceiver.kt*, który odpowiadać będzie za rejestrowanie powiadomień, które stworzymy.

6.1 Baza danych*:*

W aplikacji wykorzystano bibliotekę Room, która umożliwia stworzenie lokalnej bazy danych SQLite. Baza ma dwie tabele – *PlantsDB* oraz *RemindersDB.*

Pierwsza służy do zapisywania nowych roślin, posiada kolumny *path*, *name*, *description* oraz autogenerowane *id*. *Path* to ścieżka do zdjęcia w pamięci wewnętrznej.

W drugiej tabeli *RemindersDB* zapisywane są przypomnienia:

```
@Entity
class RemindersDB: Serializable {
    //getters & setters
    @PrimaryKey(autoGenerate = true)
    var id = 0

    @ColumnInfo(name = "date")
    var date: String? = null

    @ColumnInfo(name = "time")
    var time: String? = null

    @ColumnInfo(name = "chore")
    var chore: String? = null

    @ColumnInfo(name = "frequency")
    var frequency: String? = null

    @ColumnInfo(name = "plantName")
    var plantName: String? = null

    @ColumnInfo(name = "plantPhoto")
    var plantPhoto: String? = null
}
```

Dwa ostatnie pola, *plantName* oraz *plantPhoto*, odnoszą się do rośliny, której dotyczy przypomnienie.

W DAO dla obu tabel zapisane są zapytania SQL oraz podstawowe metody (insert, delete, update).

Sama baza danych wraz z tabelami zdefiniowana jest w *AppDB*, a obsługiwana jest przez instancję *ClientDB.*

6.2. Zakładka *Twoje rośliny*:

Galeria zbudowana jest z kilku plików. *PlantsFragment.kt* definiuje główny widok i pozwala na stworzenie scrollowalnej listy wertykalnej z pojedynczym guzikiem *Dodaj nową roślinę* na samym dole. Dzięki wykorzystaniu adaptera *PlantAdapter.kt* oraz modelu *PlantModel.kt* można utworzyć szablon dla każdej rośliny, a następnie wypełnić go danymi z bazy danych.

Dzięki *PlantAdapter.kt* możemy stworzyć dla każdej rośliny guzik, który odpowiada np. za przesłanie informacji do aktywności dodającej przypomnienia:

```
buttonAddReminder.setOnClickListener{ it: View!
    val intent = Intent(context, AddReminderActivity::class.java)
    intent.putExtra( name: "plant_reminder", plant)
    context.startActivities(arrayOf(intent))
}
```

przesłanie dodatkowych informacji (obiekt *plant*) i uruchomienie aktywności AddReminderActivity

W adapterze wczytywane są także zdjęcia z pamięci wewnętrznej.

Wczytywanie z bazy jest wykonywane asynchronicznie w *PlantsFragment.kt* i dodawane do adaptera, bazując na modelu danych *PlantModel.kt* i projekcie GUI z *single_item_plant.xml:*

```
class LoadFromDB : AsyncTask<Void?, Void?, List<PlantsDB>?>(){
    override fun doInBackground(vararg p0: Void?): List<PlantsDB>?{
        //retrieve data from DB
        return ClientDB.getInstance(requireContext())?.appDatabase?.plantsDAO()?.allDesc()
    }

    override fun onPostExecute(db: List<PlantsDB>?){
        super.onPostExecute(db)

        if (db != null) {
            for (element in db){
                list.add(PlantModel(element.name, element.description, element.path!!, element))
            }

            listView.adapter = PlantAdapter(requireContext(), R.layout.single_item_plant, list)
        }
    }
}

LoadFromDB().execute()
```

W *PlantAdapter.kt* jest także zaimplementowane powiększanie zdjęć. Wykorzystano do tego AlertDialog, w którym wczytujemy nazwę i zdjęcie rośliny z pamięci:

```
imageViewPhoto.setOnClickListener{ it: View!
    val info = AlertDialog.Builder(context)
    val factory = LayoutInflater.from(context)
    val dialogView: View = factory.inflate(R.layout.picture_popup, root: null)

    val infoImageView: ImageView = dialogView.findViewById(R.id.dialog_imageview)

    loadImageFromStorage(plant.photo, infoImageView)

    info.setView(dialogView)
    info.setIcon(R.drawable.ic_plants)
    info.setTitle(plant.name)
    //info.setMessage(plant.description)
    info.setNeutralButton( text: "Wróć") { _, _ -> }
    info.show()
}
```

6.3 Zakładka *Twoje przypomnienia:*

Lista przypomnień działa podobnie do galerii roślin. Tutaj także jest model oraz adapter – *ReminderModel.kt* oraz *ReminderAdapter.kt,* które wykorzystywane są do przedstawienia przypomnień w uporządkowany sposób według projektu GUI z *single_item_reminder.xml* oraz *fragment_reminders.xml.*

6.4. *Dodaj nową roślinę:*

Dodawanie nowej rośliny jest nową aktywnością, którą wywołujemy klikając guzik z poziomu galerii:

```
buttonAddNew?.setOnClickListener{ it: View!
    val intent = Intent (activity, AddPlantActivity::class.java)
    activity?.startActivity(intent)
}
```

Zasada działania tej funkcjonalności jest podobna do funkcjonalności z poprzedniego projektu (aplikacja modowa). Żeby wybrać zdjęcie z galerii lub je wykonać wymagane było stworzenie metod *checkAndRequestPermission*, *onActivityResult* oraz *onRequestPermissionsResult,* sprawdzających uprawnienia aplikacji. Jeśli aplikacja jest uruchamiana po raz pierwszy wyskakuje pop up i prosi o przyznanie uprawnień.

Po dodaniu zdjęcia zapisujemy plik do pamięci wewnętrznej, a następnie dodajemy roślinę do bazy, tym razem używając kotlinowych Coroutine, które działają podobnie co Async w Javie.

```
val newPlant = PlantsDB()
newPlant.path = picture
newPlant.name = editTextName!!.text.toString()
newPlant.description = editTextDescription!!.text.toString()

CoroutineScope(IO).launch { this: CoroutineScope
    ClientDB.getInstance(applicationContext)?.appDatabase?.plantsDAO()?.insert(newPlant)
}

Toast.makeText(view.context, text: "Zapisano", Toast.LENGTH_SHORT).show()

this.finish() //closes fragment
startActivity(Intent( packageContext: this, MainActivity::class.java)) //moves to homepage
```

stworzenie obiektu bazy, dodanie go w Coroutine i powrót do ekranu głównego

6.5. Dodawanie przypomnień:

W galerii klikamy przycisk *Przypomnij* przy wybranej roślinie. Wybieramy datę, czas oraz czynność.

Na tym etapie można rozróżnić przypomnienia na dwa przypadki – jednorazowe i cykliczne.

Jeśli zostawimy *powtarzanie wyłączone*, wówczas w funkcji *startAlarm* wywołane zostanie *alarmManager.setExact*, który odpowiada za jednokrotne powiadomienia.
W innym przypadku wywołane zostanie *alarmManager.setInexactRepeating* z odpowiednią opcją, którą wybraliśmy (*powtarzaj co miesiąc* etc.), a powiadomienie zostanie ustawione z daną częstotliwością.
Należy dodać, że metoda *setInexactRepeating* nie będzie wyświetlać powiadomień z najwyższą dokładnością (tak jak *setExactRepeating)*, lecz system operacyjny będzie mógł wyświetlać powiadomienia z małym opóźnieniem w celu optymalizacji zarządzania baterią urządzenia.

```
var repeat = AlarmManager.INTERVAL_DAY  // daily
when (spinnerFrequency?.selectedItem.toString()){
    "powtarzaj co tydzień" -> repeat *= 7
    "powtarzaj co dwa tygodnie" -> repeat *= 14
    "powtarzaj co miesiąc" -> repeat *= 30
    "powtarzaj co trzy miesiące" -> repeat *= 90
}
alarmManager.setInexactRepeating(AlarmManager.RTC_WAKEUP, calendar.timeInMillis, repeat, pendingIntent)
```

powiadomienia cykliczne

Ponadto dzięki automatycznie generowanym kluczom id z bazy danych możemy wysyłać wiele powiadomień:

```
val alarmManager = getSystemService(Context.ALARM_SERVICE) as AlarmManager
val intent = Intent( packageContext: this, AlarmReceiver::class.java)

intent.putExtra( name: "plantPhoto", plantInfo[0])
intent.putExtra( name: "plantName", plantInfo[1])
intent.putExtra( name: "plantChore", plantInfo[2])

// requestCode should be unique to allow multiple notifications,
// so we take autogenerated key id from db
val pendingIntent = PendingIntent.getBroadcast( context: this, requestID.toInt(),
                                                intent, FLAG_UPDATE_CURRENT)
```

Wartość *requestID* to właśnie klucz, który otrzymujemy po utworzeniu obiektu przypomnienia w bazie, który przesyłamy do wywoływanej funkcji *startAlarm*. Aby wysyłać wiele różnych przypomnień wartość ta powinna być zawsze różna.

Wracając do *AlarmReceiver.kt* zdefiniowanego w *AndroidManifest.xml* – odpowiada on za odbieranie powiadomienia i wyświetlanie go zgodnie z szablonem w *AlarmNotificationBuilder.kt:*

```
val photoBitmap = BitmapFactory.decodeFile( pathName: "$plantPhoto/profile.jpg")
return NotificationCompat.Builder(applicationContext, channelID)
    .setContentTitle("Czas na <strong>${plantChore}</strong>!".toSpanned())
    .setContentText("Twoja roślina <strong>${plantName}</strong> wymaga uwagi.".toSpanned())
    .setSmallIcon(R.drawable.ic_plants)
    .setLargeIcon(photoBitmap)
    .setStyle(NotificationCompat.BigPictureStyle().bigPicture(photoBitmap).bigLargeIcon( b: null))
    .setContentIntent(pendingIntent)
    .setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION))
    .setAutoCancel(true)
```

szablon przypomnień popup

6.6. Ostrzeżenia przed przelaniem:

Ostrzeżenie to prosty *AlertDialog*. Sprawdzając pola powtarzalności przypomnień i czynności sprawdzane jest czy równocześnie użytkownik ustawił podlewanie i codzienne powiadomienia:

```
if(newReminder.frequency == "powtarzaj codziennie" && newReminder.chore == "podlewanie") {
    val alertDialog = AlertDialog.Builder( context: this)
        .setIcon(R.drawable.ic_warning)
        .setTitle("Czy utworzyć przypomnienie?")
        .setMessage("Codzienne podlewanie może zaszkodzić Twoim roślinom! " +
                "Czy na pewno chcesz dodać to przypomnienie?")

    alertDialog.setPositiveButton( text: "Tak") { _, _ ->
        addAlarmAndFinish(newReminder)
    }

    alertDialog.setNegativeButton( text: "Anuluj") { _, _ -> }

    alertDialog.show()
}
else{
    addAlarmAndFinish(newReminder)
}
```

W tym przypadku aplikacja czeka na potwierdzenie czy użytkownik na pewno chce kontynuować przed dodanie powiadomienia.