Aplikacje Mobilne - Aplikacja - Sprawozdanie

Grupa laboratoryjna: L9	Student:	Prowadzący zajęcia:
	Wiktor Jordeczka (151 785)	mgr. inż. Mateusz Leszek

Link do repozytorium: https://github.com/Wiktor-Jordeczka/Trails

Aplikacja składa się z czterech aktywności:

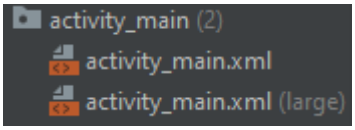
1. Główna, wyświetlająca fragmenty ze szlakami i zdjęciami
2. Szczegółów szlaku, wyświetlająca fragment szczegółów, tylko w układzie smartfona
3. Animacji, wyświetlająca animację powitalną
4. Zdjęcia, wyświetlająca zdjęcie pełnoekranowo

[DODATKOWE] : aplikacja napisana w kotlinie

Wszystkie fragmenty i aktywności działają prawidłowo po obrocie ekranu, korzystając z SaveInstanceState. Przykład:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    if (savedInstanceState != null){
        trailID = savedInstanceState.getInt("trailID")
        speed = savedInstanceState.getFloat("speed")
        trailLengthKm = savedInstanceState.getFloat("trailLengthKm")
    }else{
        val stoper = StoperFragment()
        stoper.trailID = trailID
        val ft: FragmentTransaction = childFragmentManager.beginTransaction()
        ft.add(R.id.stoper_container, stoper)
        ft.addToBackStack(null)
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
        ft.commit()
    }
}

override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    trailID?.let { outState.putInt("trailID", it) }
    speed.let { outState.putFloat("speed", it) }
    trailLengthKm.let { outState.putFloat("trailLengthKm", it) }
}
```

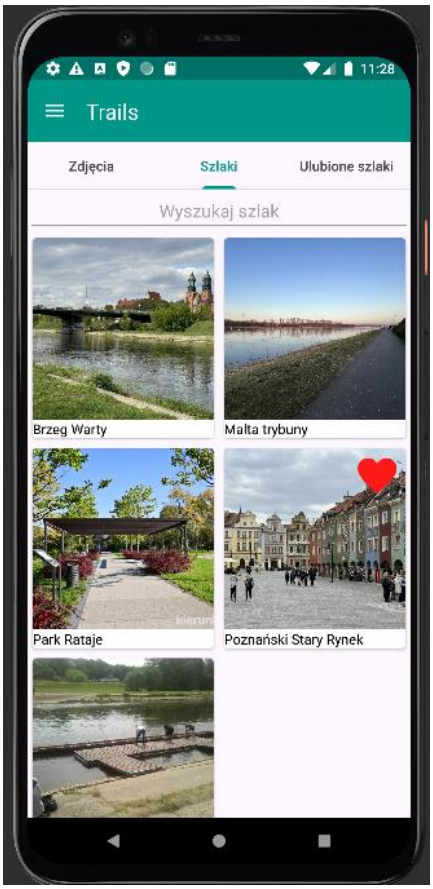
Aplikacja posiada osobne wersje układu dla smartfonów i tabletów.

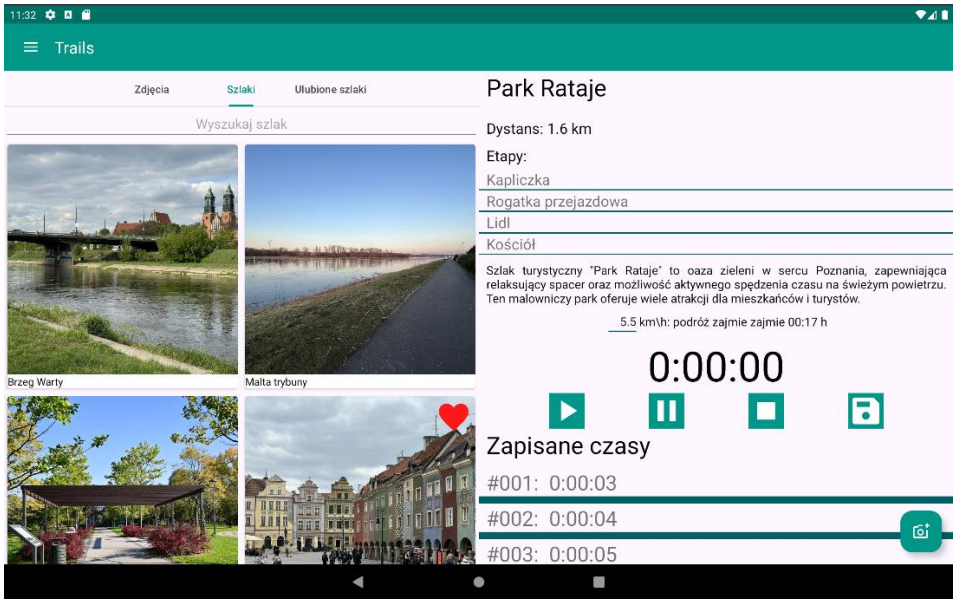


Przykład ustawiania listenera, dla elementu, który występuje tylko w układzie tabletowym:

```
// Tablet only - przycisk fab
binding.tabletFab?.setOnClickListener {
    dispatchTakePictureIntent()
}
```

Porównanie wyglądu układów aktywności głównej, fragmentu szlaków (oraz fragmentu szczegółów w wersji tabletowej)





[DODATKOWE] : aplikacja wykorzystuje bazę danych

- Przykładowa tabela

```
object DBContract {
    const val DATABASE_VERSION = 8
    const val DATABASE_NAME = "trails_database.db"
    private const val TEXT_TYPE = " TEXT"
    private const val INT_TYPE = " INTEGER"
    private const val FLOAT_TYPE = " REAL"
    private const val COMMA_SEP = ","

    //Tabela szlaków
    object TrailsTable : BaseColumns {
        const val TABLE_NAME = "trails"
        const val COLUMN_NAME_TRAIL_NAME = "trail_name"
        const val COLUMN_NAME_TRAIL_DESC = "trail_description"
        const val COLUMN_NAME_TRAIL_IMG = "trail_img_id"
        const val COLUMN_NAME_TRAIL_LENGTH = "trail_length"
        const val COLUMN_NAME_TRAIL_FAV = "is_favorite"
        const val CREATE_TABLE = "CREATE TABLE " +
                TABLE_NAME + " (" +
                BaseColumns._ID + " INTEGER PRIMARY KEY," +
                COLUMN_NAME_TRAIL_NAME + TEXT_TYPE + COMMA_SEP +
                COLUMN_NAME_TRAIL_DESC + TEXT_TYPE + COMMA_SEP +
                COLUMN_NAME_TRAIL_IMG + INT_TYPE + COMMA_SEP +
                COLUMN_NAME_TRAIL_FAV + INT_TYPE + COMMA_SEP +
                COLUMN_NAME_TRAIL_LENGTH + FLOAT_TYPE + " );"
        const val DELETE_TABLE = "DROP TABLE IF EXISTS $TABLE_NAME;"
    }
```

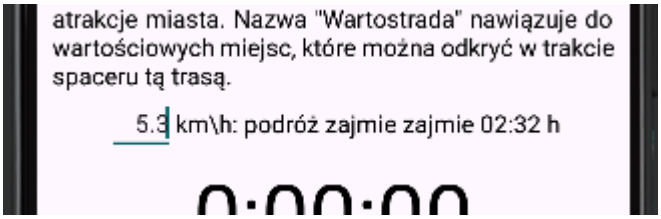
- Przykładowe zapytanie

```
// db connection
dbHelper = context?.let { TrailsDBHelper(it) }!! // create helper
db = dbHelper.writableDatabase // get the db
val table = "${DBContract.TrailsTable.TABLE_NAME} trails INNER JOIN
${DBContract.StagesTable.TABLE_NAME} stages on trails.${BaseColumns._ID} =
stages.${DBContract.StagesTable.COLUMN_NAME_TRAIL_ID_FK}"
val projection = arrayOf(DBContract.TrailsTable.COLUMN_NAME_TRAIL_NAME,
DBContract.TrailsTable.COLUMN_NAME_TRAIL_DESC,
DBContract.TrailsTable.COLUMN_NAME_TRAIL_IMG,
DBContract.TrailsTable.COLUMN_NAME_TRAIL_LENGTH,
DBContract.TrailsTable.COLUMN_NAME_TRAIL_FAV,
DBContract.StagesTable.COLUMN_NAME_STAGE_NAME)
val selection = "trails.${BaseColumns._ID} = ?"
val selectionArgs = arrayOf(id.toString())
val groupBy = null
val having = null
val sortOrder = null
val cursor = db.query(
    table,   // The table to query
    projection,             // The array of columns to return (pass null to get all)
    selection,              // The columns for the WHERE clause
    selectionArgs,          // The values for the WHERE clause
    groupBy,                // don't group the rows
    having,                 // don't filter by row groups
    sortOrder               // The sort order
)
var trailName = ""
var trailDescription = ""
val trailStages: MutableList<String> = mutableListOf()
with(cursor){
    while(moveToNext()){
        trailName =
getString(getColumnIndexOrThrow(DBContract.TrailsTable.COLUMN_NAME_TRAIL_NAME))
        trailDescription =
getString(getColumnIndexOrThrow(DBContract.TrailsTable.COLUMN_NAME_TRAIL_DESC))
        trailLengthKm =
getFloat(getColumnIndexOrThrow(DBContract.TrailsTable.COLUMN_NAME_TRAIL_LENGTH))
        val stage =
getString(getColumnIndexOrThrow(DBContract.StagesTable.COLUMN_NAME_STAGE_NAME))
        trailStages.add(stage)
    }
}
cursor.close()
dbHelper.close()
```

[DODATKOWE] : fragment szczegółów wyświetla informację o czasie przejścia szlaku na podstawie wpisanej przez użytkownika prędkości

```
// input kalkulatora czasu podróży
binding.speedEdit.addTextChangedListener(object : TextWatcher {
    override fun onTextChanged(s: CharSequence, start: Int, before: Int, count: Int) {
        if (s.isNotEmpty()){ // czy cokolwiek jest wpisane
            try { // czy możemy odczytać input jako float
                val tmp = (s.toString()).toFloat()
                if(tmp>0){ // czy prędkość dodatnia
                    speed = tmp
                    calculateTime()
                }
            } catch (ex: NumberFormatException) {
                // Not a float
            }
        }
    }
```

```
// obliczamy czas podróży i wyświetlamy go
private fun calculateTime(){
    val time = trailLengthKm/speed
    val hrsString = "%02d".format(time.toInt())
    val minString = "%02.0f".format((time % 1)*60)
    val text = "km\\h: podróż zajmie zajmie ${hrsString}:${minString} h"
    binding.speedtext.text = text
}
```



Aplikacja zawiera fragment dynamiczny ze stoperem, stoper działa poprawnie po zmianie orientacji oraz po przejściu aplikacji do tła.

- Działanie stopera

```
// handler stopera
private fun runStoper() {
    val timeView: TextView = binding.timeView
    handler.post(object : Runnable {
        private var prevSeconds = 0
        override fun run() {
            if (seconds != prevSeconds){
                // Bugfix dla układu tabletowego, gdy odpalimy kilka stoperów naraz
                handler.removeCallbacksAndMessages(null)
            }
            val hours = seconds / 3600
            val minutes = seconds % 3600 / 60
            val secs = seconds % 60
            val time = String.format("%d:%02d:%02d", hours, minutes, secs)
            timeView.text = time
            if (running) {
                seconds++
                prevSeconds = seconds
            }
            handler.postDelayed(this, 1000)
        }
    })
}
```

- Utrzymanie liczenia czasu w tle

```
override fun onPause() {
    wasRunning = running
    running = false
    stopTime = SystemClock.elapsedRealtime()/1000
    super.onPause()
}

override fun onResume() {
    super.onResume()
    if (wasRunning) {
        running = true
        seconds += ((SystemClock.elapsedRealtime()/1000) - stopTime).toInt()
    }
}
```

[DODATKOWE] : można również zapisać wyniki w rankingu, przechowywanym w bazie danych.

```
// zapisujemy czas do bazy danych
private fun onClickSave() {
    dbHelper = context?.let { TrailsDBHelper(it) }!! // create helper
    db = dbHelper.writableDatabase // get the db
    val values = ContentValues().apply {
        put(DBContract.TimesTable.COLUMN_NAME_TRAIL_TIME, seconds)
        put(DBContract.TimesTable.COLUMN_NAME_TRAIL_ID_FK, trailID)
    }
    db.insert(
        DBContract.TimesTable.TABLE_NAME,
        null,
        values
    )
    dbHelper.close()
    updateAdapter() // odświeżamy adapter
}
```

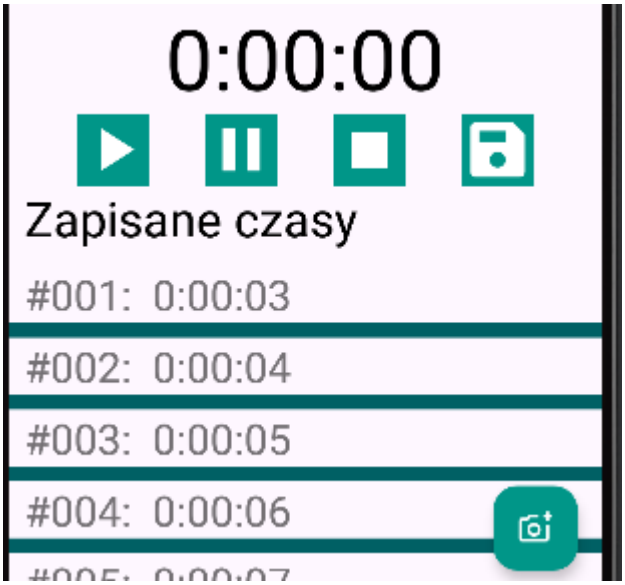
[DODATKOWE] : można obejrzeć zapisane wyniki poniżej stopera, osobne dla każdego szlaku

[DODATKOWE] : ikony zamiast napisów na przyciskach

[DODATKOWE] : zapisane czasy można usunąć przytrzymując wybrany wiersz

```
binding.timesListView.setOnItemLongClickListener
{ _, _, position, _ ->
    deleteTime(timesIds[position]) // usuwamy
czas z tabeli
    updateAdapter() // odświeżamy adapter
    true// konsumujemy zdarzenie dla poprawności
}
```

```
// usuwamy czas z bazy danych
private fun deleteTime(id: Int) {
    // db connection
    dbHelper = context?.let{TrailsDBHelper(it)}!!
    db = dbHelper.writableDatabase // get the db
    val table = DBContract.TimesTable.TABLE_NAME
    val selection = "${BaseColumns._ID} = ?"
    val selectionArgs = arrayOf(id.toString())
    db.delete(table, selection, selectionArgs)
    dbHelper.close()
}
```



Aplikacja wykorzystuje ViewPager, zawierający Fragmenty, które korzystają z RecyclerView, z układem siatki, dla których poszczególnych pozycji użyto CardView. Przechodzenie pomiędzy kartami odbywa się także za pomocą gestu przeciągnięcia. Obraz wyżej.

```
// ustawiamy adapter do ViewPagera
mPagerAdapter = SectionsPagerAdapter(supportFragmentManager, this)
val pager = binding.pager
pager.adapter = mPagerAdapter
pager.setCurrentItem(startingFragmentPosition)
```

```
// pobieramy dane szlaków do adaptera RecyclerView
getDataFromDB((activity as MainActivity).db)
val adapter = CaptionedImagesAdapter(trailNames, trailImgIds, favTrails)
binding.tab1recycler.adapter = adapter
val layoutManager = GridLayoutManager(activity, 2)
binding.tab1recycler.layoutManager = layoutManager
```

```
class CaptionedImagesAdapter{
...
// ustawiamy wygląd karty
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val cardView = holder.cardView
    val imageView = cardView.findViewById<ImageView>(R.id.card_image)
    val drawable = ContextCompat.getDrawable(cardView.context, imageIds[position])
    imageView.setImageDrawable(drawable)
    imageView.contentDescription = captions[position]
    val textView = cardView.findViewById<TextView>(R.id.card_text)
    textView.text = captions[position]
    // jeżeli szlak należy do ulubionych to wyświetlamy serduszko
    val heart = cardView.findViewById<ImageView>(R.id.card_heart)
    if (favTrails[position] == 0){
        heart.visibility = View.INVISIBLE
    }else{
        heart.visibility = View.VISIBLE
    }

    cardView.setOnClickListener{
        recListener.onClick(position)
    }

    cardView.setOnLongClickListener{
        recListener.onLongClick(position)
        return@setOnLongClickListener true // konsumujemy zdarzenie, żeby nie odpalić
również OnClick
    }
}
```

Na ekranie szczegółów jest przycisk FAB, uruchamiający aparat.

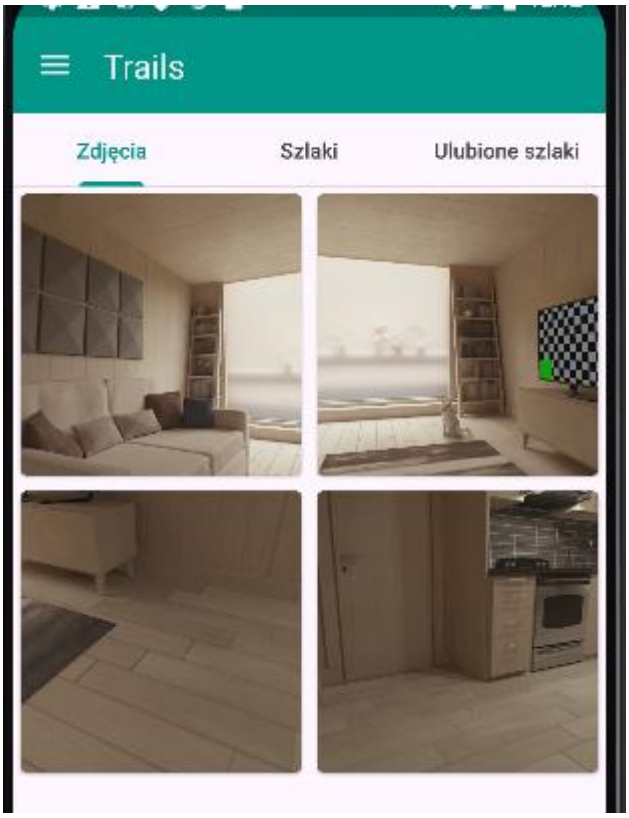
```
// do wywołania kamery i zapisania zdjęcia
val fab: FloatingActionButton = binding.fab
fab.setOnClickListener {
    dispatchTakePictureIntent()
}
```

[DODATKOWE] : zdjęcia są zapisane w bazie danych, możemy je wyświetlić we fragmencie Zdjęć, oraz po kliknięciu wyświetlić je pełnoekranowo z możliwością przybliżenia i obracania. Można je też usunąć przytrzymując wybrane zdjęcie.

```
// jeśli zdjęcie zostało zapisane do pliku, to zapiszmy ścieżkę do pliku w bazie
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == Activity.RESULT_OK){
        val values = ContentValues().apply {
            put(DBContract.PicturesTable.COLUMN_NAME_IMG_PATH, currentPhotoPath)
        }
        db.insert(
            DBContract.PicturesTable.TABLE_NAME,
            null,
            values
        )
    }
    super.onActivityResult(requestCode, resultCode, data)
}
```

```
// pobieramy zdjęcia z bazy i ustawiamy adapter
getImagesFromDB((activity as MainActivity).db)
val adapter = ImagesAdapter(imgPaths, ids)
binding.imagesTabRecycler.adapter = adapter
val layoutManager = GridLayoutManager(activity, 2)
binding.imagesTabRecycler.layoutManager = layoutManager
```

```
// wyświetlamy zdjęcie w fullscreenie
override fun onClick(position: Int) {
    val intent = Intent(activity, FullscreenImageActivity::class.java)
    intent.putExtra("path",imgPaths[position])
    startActivity(intent)
}

// usuwamy zdjęcie (plik oraz wpis w bd)
override fun onLongClick(position: Int) {
    val db = (activity as MainActivity).db
    val selection = "${BaseColumns._ID} = ?"
    val selectionArgs = arrayOf(ids[position].toString())
    db.delete(
        DBContract.PicturesTable.TABLE_NAME,
        selection,
        selectionArgs
    )
    // usuwamy plik
    val file = File(imgPaths[position])
    file.delete()
    refreshAdapter(db) // odświeżamy dane adaptera
}
```



Każda aktywność ma pasek aplikacji

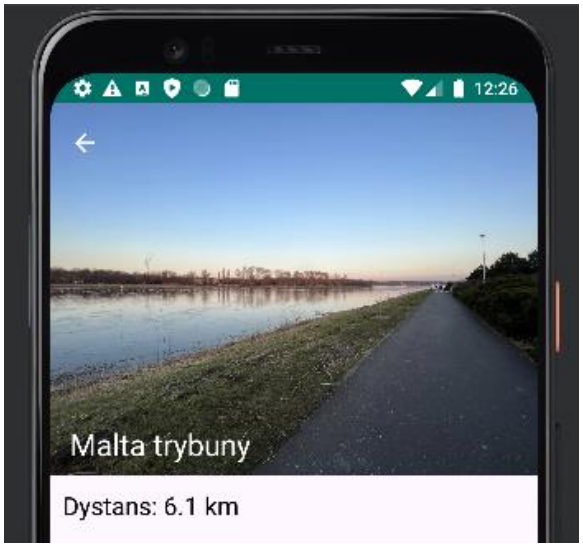
```
// ustawiamy toolbar
val toolbar = binding.toolbar.toolbar
setSupportActionBar(toolbar)
```

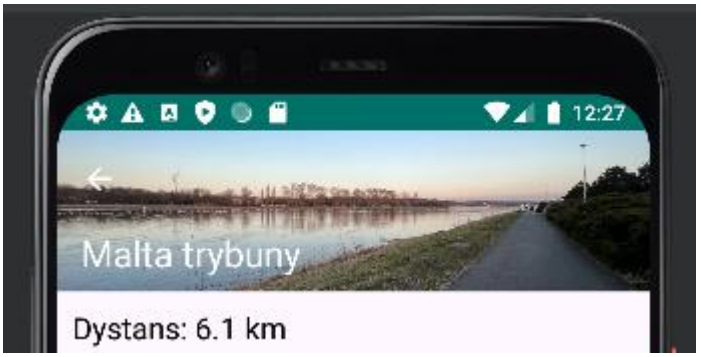
Ekran szczegółów jest przewijany w pionie razem z paskiem aplikacji

Na ekranie szczegółów pojawia się obrazek na pasku aplikacji i zwija się razem z nim.

- Wycinek XML:

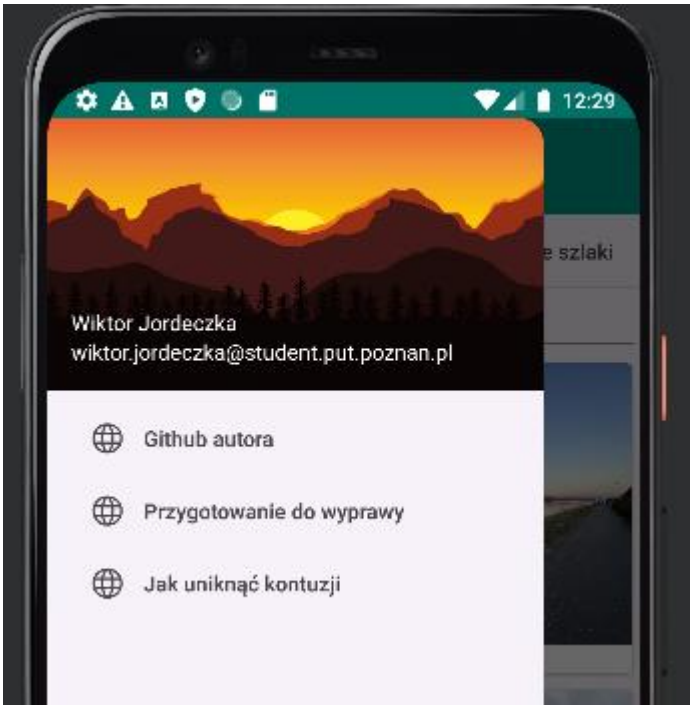
```
<com.google.android.material.appbar.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="300dp"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    android:fitsSystemWindows="true">

    <com.google.android.material.appbar.CollapsingToolbarLayout
        android:id="@+id/collapsing_toolbar_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_scrollFlags="scroll|exitUntilCollapsed"
        app:contentScrim="?attr/colorPrimary"
        app:collapsedTitleTextColor="@color/white"
        app:expandedTitleTextColor="@color/white">

        <ImageView
            android:id="@+id/trail_image"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scaleType="centerCrop"
            android:src="@drawable/drawer_image_maybe"
            android:contentDescription="@string/top_image"
            app:layout_collapseMode="parallax" />

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:layout_collapseMode="pin" />
    </com.google.android.material.appbar.CollapsingToolbarLayout>
</com.google.android.material.appbar.AppBarLayout>

<androidx.core.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fillViewport="true"
    app:layout_behavior="@string/appbar_scrolling_view_behavior" >
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/detailFrag"
        android:name="edu.put.inf151785.TrailDetailFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp" />
</androidx.core.widget.NestedScrollView>
```





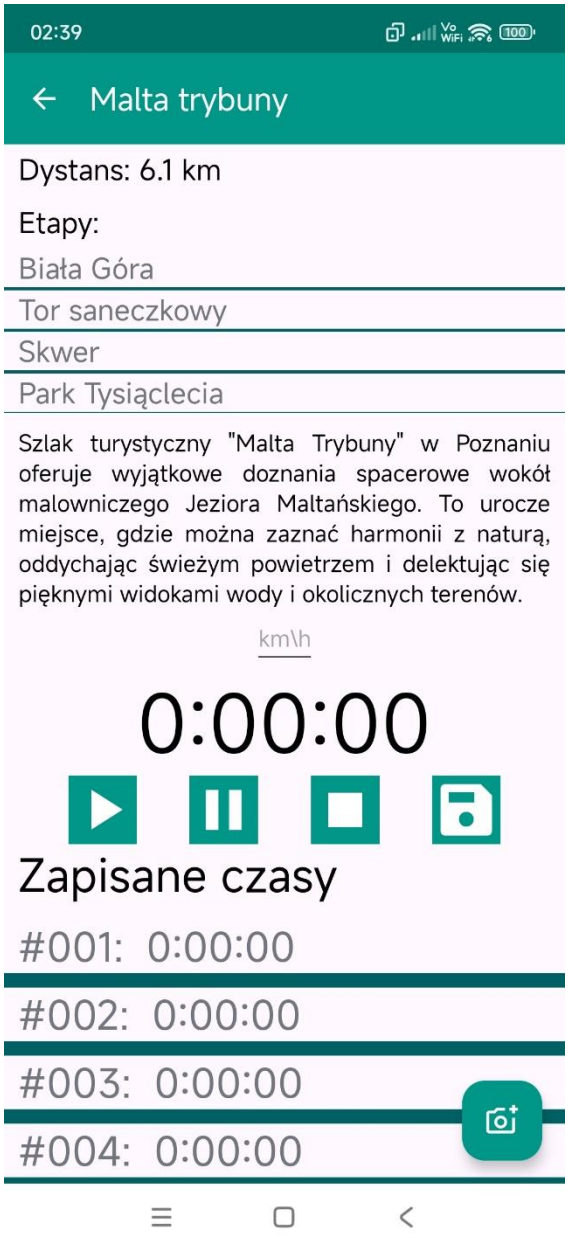
Do aplikacji dodana została szuflada nawigacyjna

```
// hamburger szuflady nawigacyjnej
val drawer = binding.drawerLayout
val toggle = ActionBarDrawerToggle(
    this,
    drawer,
    toolbar,
    R.string.open_drawer,
    R.string.close_drawer
)
drawer.addDrawerListener(toggle)
toggle.syncState()

// elementy szuflady nawigacyjnej
val navigationView = binding.navView
navigationView.setNavigationItemSelectedListener { item ->
    lateinit var url: String
    val id = item.itemId
    when (id) {
        R.id.nav_link1 -> url = "https://github.com/Wiktor-Jordeczka"
        R.id.nav_link2 -> url =
            "https://www.c-and-a.com/pl/pl/shop/przygotowania-do-wedrowki"

        R.id.nav_link3 -> url =
            "https://blog.gopass.travel/pl/bezpieczne-wedrowki-jak-cieszyc-sie-
wycieczka-bez-stresu/"
    }
    val defaultBrowser =
        Intent.makeMainSelectorActivity(Intent.ACTION_MAIN,
Intent.CATEGORY_APP_BROWSER)
    defaultBrowser.setData(Uri.parse(url))
    startActivity(defaultBrowser)
    true
}
```



Aplikacja korzysta z motywów

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Base.Theme.Trails_151785"
parent="Theme.Material3.DayNight.NoActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
        <item name="android:textColor">@color/black</item>
    </style>

    <style name="Theme.Trails_151785" parent="Base.Theme.Trails_151785" />

    <style name="FullscreenImage" parent="Base.Theme.Trails_151785">
        <item name="background">@color/black</item>
    </style>

    <style name="Animation" parent="Base.Theme.Trails_151785">
        <item name="android:textColor">@color/white</item>
    </style>
</resources>
```

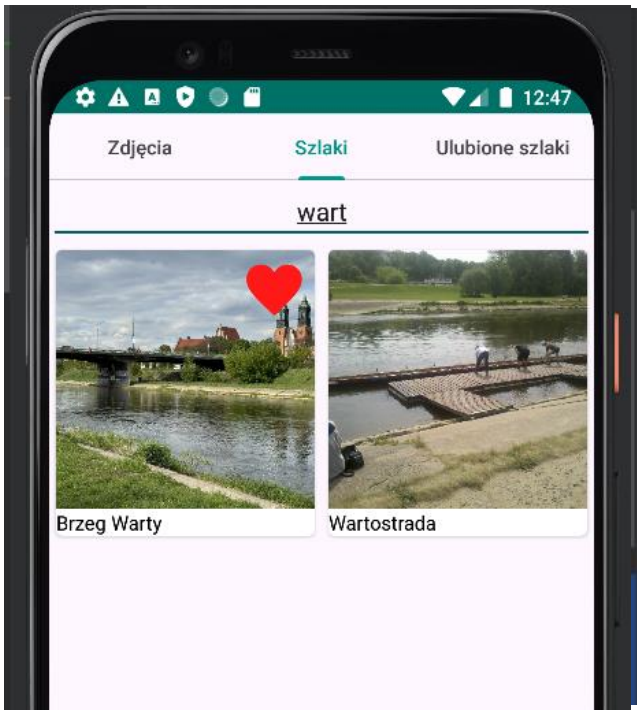


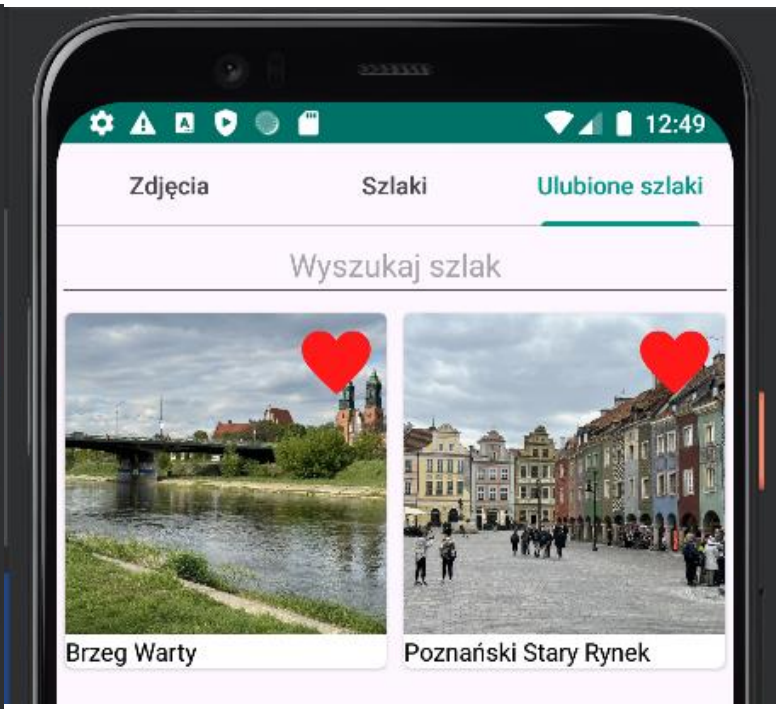
[DODATKOWE] : można dodawać szlaki do ulubionych poprzez ich długie przytrzymanie. Wizualizowane serduszkiem nałożonym na CardView.

```
// dodaj / usuń szlak z ulubionych w bd
override fun onLongClick(position: Int) {
    val db = (activity as MainActivity).db
    var isFav = favTrails[position]
    isFav = if (isFav == 0){
        1
    }else{
        0
    }
    val values = ContentValues().apply {
        put(DBContract.TrailsTable.COLUMN_NAME_TRAIL_FAV, isFav)
    }
    val selection = "${BaseColumns._ID} = ?"
    val selectionArgs = arrayOf(ids[position].toString())
    db.update(
        DBContract.TrailsTable.TABLE_NAME,
        values,
        selection,
        selectionArgs
    )
    refreshAdapter(db)
}
```

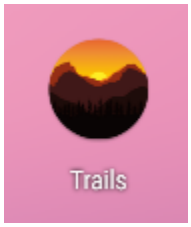
[DODATKOWE] : możliwość wyszukiwania szlaku

```
//search box
binding.searchBox.addTextChangedListener(object : TextWatcher {
    override fun beforeTextChanged(s: CharSequence, start: Int, count: Int, after: Int)
{}
    override fun afterTextChanged(s: Editable) {}
    override fun onTextChanged(s: CharSequence, start: Int, before: Int, count: Int) {
        searchedTrail = s.toString()
        refreshAdapter((activity as MainActivity).db)
    }
})
```





[DODATKOWE] : własna ikona aplikacji




```xml
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon
xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@color/ic_launcher_background"/>
    <foreground android:drawable="@mipmap/ic_launcher_foreground"/>
</adaptive-icon>
```

Aplikacja posiada animację powitalną

[DODATKOWE] : animacja korzysta z kilku animatorów jednocześnie

```
// animacja tekstu
val animatedText = binding.animationTitle
animatedText.measure(0,0)
val xDelta = ((getScreenWidth(this)/2)-(animatedText.measuredWidth/2)).toFloat()
val textAnimator = ObjectAnimator.ofFloat(animatedText, "translationX", xDelta)
textAnimator.setDuration(animationLength)
val textAlphaAnimator = ObjectAnimator.ofFloat(animatedText, "alpha", 0.0f, 1.0f)
textAlphaAnimator.setDuration(animationLength)

// animacja logo
val animatedLogo = binding.animationIcon
animatedLogo.measure(0,0)
val yDelta = (-((getScreenHeight(this))/*-
(animatedLogo.drawable.intrinsicHeight)*/)).toFloat()
val logoAnimator = ObjectAnimator.ofFloat(animatedLogo, "translationY", yDelta)
logoAnimator.setDuration(animationLength)

// kod do wykonania po zakończeniu animacji
logoAnimator.addListener(object : AnimatorListenerAdapter() {
    override fun onAnimationEnd(animation: Animator) {
        super.onAnimationEnd(animation)
        // uruchamiamy aktywność główną z flagą informującą, że nie wrócimy już do
obecnej aktywności
        val intent = Intent(this@AnimationActivity, MainActivity::class.java)
        intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
        startActivity(intent)
        finish()
    }
})

// łączymy animacje w set i uruchamiamy
val animatorSet = AnimatorSet()
animatorSet.play(textAnimator).with(logoAnimator).with(textAlphaAnimator)
animatorSet.start()
```