



PROGRAMOWANIE URZĄDZEŃ MOBILNYCH 2

WYKŁAD 1

- PODSTAWOWE INFORMACJE
- TREŚCI PROGRAMOWE
- ZASADY ZALICZENIA
- JETPACK NAVIGATION 3

Rafał Lewandków

pokój 075

rafal.lewandkow2@uwr.edu.pl

Forma zajęć i liczba godzin:

Wykład 15 godz. /Laboratorium 30 godz.

Materiały do zajęć: <https://github.com/RafLew84/ProgUM>

Literatura obowiązkowa i zalecana:

- <https://kotlinlang.org/docs/home.html>
- <https://developer.android.com/courses>

Nakład pracy studenta:

Praca własna studenta: 30 godz.

Łączna liczba godzin 75

Liczba punktów ECTS: 3

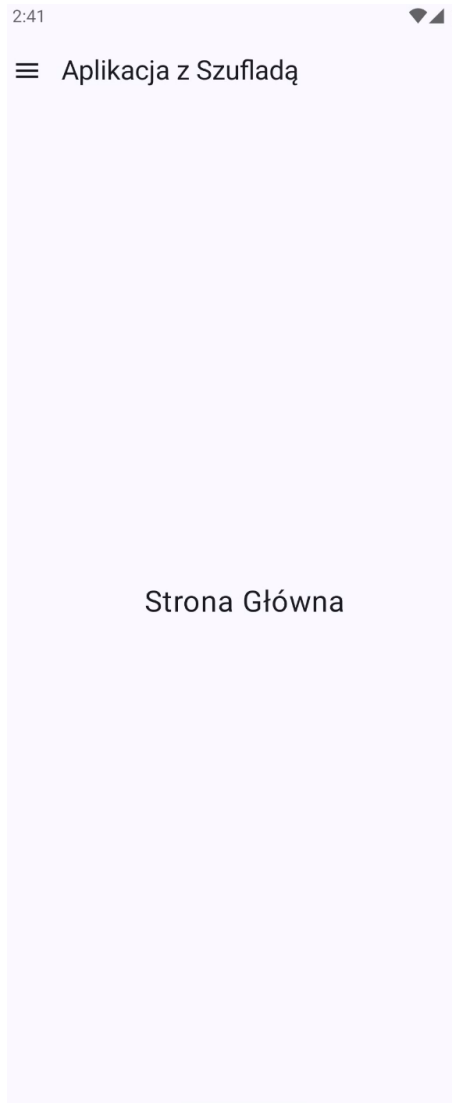
ZASADY ZALICZENIA

Warunkiem zaliczenia laboratorium jest uzyskanie pozytywnej oceny z list zadań.

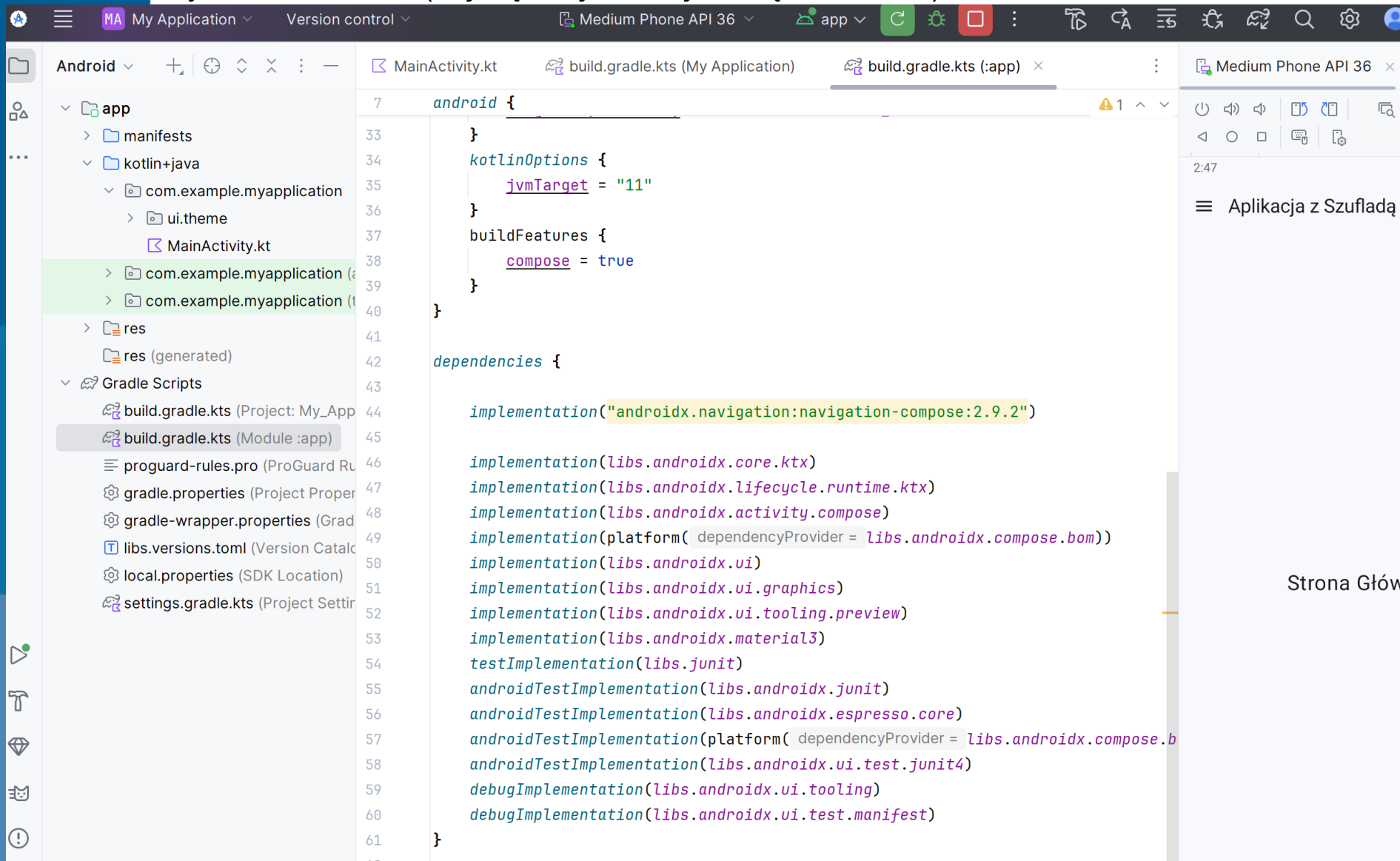
- Na zajęcia przewidzianych jest **5 list zadań**.
- Z każdej listy wystawiana jest **osobna ocena**.
- **Nie jest konieczne** zaliczenie wszystkich list aby otrzymać ocenę pozytywną z laboratorium. Dopuszczalne jest nieoddanie/niezaliczenie jednej listy.
- Każda lista posiada informację o **liczbie punktów** wymaganych na konkretną ocenę
- Każda lista posiada **termin zwrotu**.
- Za każdy tydzień opóźnienia otrzymana ocena jest **obniżana o 1**.
- Listy oddawane są **podczas zajęć** laboratoryjnych.
- Do każdej listy prowadzący **zadaje 4 pytania**.
- Liczba punktów za listę jest przyznawana na podstawie **poprawności wykonania zadań oraz odpowiedzi ustnej**.
- Ocena końcowa jest **średnią arytmetyczną ze wszystkich ocen** z list.
- Na ocenę 3,0 **wymagana jest** średnia co najmniej 3,0.
- Na zajęciach laboratoryjnych dopuszczalne są **trzy nieobecności**.

1. Zasady zaliczenia, Treści Programowe, Zaawansowana Nawigacja.
2. Wprowadzenie do Wielowątkowości: Coroutines. Wątek główny.
3. Coroutines. Współbieżność, Równoległość, Asynchroniczność.
4. Podstawy Architektury Aplikacji: Wzorce MVx (MVC, MVP, MVVM).
5. Reaktywne Zarządzanie Stanem: Flow, StateFlow, SharedFlow.
6. Zaawansowane Zarządzanie Stanem: withContext, StateIn, ShareIn, FlowOn, combine.
7. Coroutines: Kanały - Asynchroniczna Wymiana Danych Między Coroutines.
8. Zapis Danych do Pliku: SharedPreferences, DataStore.
9. Baza Danych SQLite + ROOM: Entity, Dao, Database, CRUD, Operacje Asynchroniczne.
10. Praca z Zewnętrznymi Źródłami Danych: Retrofit2, Operacje Asynchroniczne.
11. Wstrzykiwanie Zależności: Dagger, Hilt.
12. Czysta Architektura - Warstwa Domeny i Wzorzec Use Case.
13. Wzorzec Single Source of Truth – Strategia Offline Caching.
14. Backend w Chmurze: Wprowadzenie do Firebase i Firestore

Drawer (szuflada nawigacyjna) to element interfejsu użytkownika, który działa jak wysuwane z boku (najczęściej z lewej krawędzi ekranu) menu.



Drawer (szuflada nawigacyjna) to element interfejsu użytkownika, który działa jak wysuwane z boku (najczęściej z lewej krawędzi ekranu) menu.



```
7  android {
33      }
34      kotlinOptions {
35          jvmTarget = "11"
36      }
37      buildFeatures {
38          compose = true
39      }
40  }
41
42  dependencies {
43
44      implementation("androidx.navigation:navigation-compose:2.9.2")
45
46      implementation(libs.androidx.core.ktx)
47      implementation(libs.androidx.lifecycle.runtime.ktx)
48      implementation(libs.androidx.activity.compose)
49      implementation(platform(dependencyProvider = libs.androidx.compose.bom))
50      implementation(libs.androidx.ui)
51      implementation(libs.androidx.ui.graphics)
52      implementation(libs.androidx.ui.tooling.preview)
53      implementation(libs.androidx.material3)
54      testImplementation(libs.junit)
55      androidTestImplementation(libs.androidx.junit)
56      androidTestImplementation(libs.androidx.espresso.core)
57      androidTestImplementation(platform(dependencyProvider = libs.androidx.compose.bom))
58      androidTestImplementation(libs.androidx.ui.test.junit4)
59      debugImplementation(libs.androidx.ui.tooling)
60      debugImplementation(libs.androidx.ui.test.manifest)
61  }
```

2:47

≡ Aplikacja z Szufladą

Strona Główna

Drawer (szuflada nawigacyjna) to element interfejsu użytkownika, który działa jak wysuwane z boku (najczęściej z lewej krawędzi ekranu) menu.

Plik build.gradle na
poziomie **Modułu**

```
7  android {
33      }
34      kotlinOptions {
35          jvmTarget = "11"
36      }
37      buildFeatures {
38          compose = true
39      }
40  }
41
42  dependencies {
43
44      implementation("androidx.navigation:navigation-compose:2.9.2")
45
46      implementation(libs.androidx.core.ktx)
47      implementation(libs.androidx.lifecycle.runtime.ktx)
48      implementation(libs.androidx.activity.compose)
49      implementation(platform(dependencyProvider = libs.androidx.compose.bom))
50      implementation(libs.androidx.ui)
51      implementation(libs.androidx.ui.graphics)
52      implementation(libs.androidx.ui.tooling.preview)
53      implementation(libs.androidx.material3)
54      testImplementation(libs.junit)
55      androidTestImplementation(libs.androidx.junit)
56      androidTestImplementation(libs.androidx.espresso.core)
57      androidTestImplementation(platform(dependencyProvider = libs.androidx.compose.bom))
58      androidTestImplementation(libs.androidx.ui.test.junit4)
59      debugImplementation(libs.androidx.ui.tooling)
60      debugImplementation(libs.androidx.ui.test.manifest)
61  }
```

2:47

≡ Aplikacja z Szufladą

Strona Główna

Drawer (szuflada nawigacyjna) to element interfejsu użytkownika, który działa jak wysuwane z boku (najczęściej z lewej krawędzi ekranu) menu.

Plik build.gradle na
poziomie **Modułu**

Blok dependencies

Dodajemy zależność

```
7  android {
33  }
34  kotlinOptions {
35      jvmTarget = "1.8"
36  }
37  buildFeatures {
38      compose = true
39  }
40  }
41
42  dependencies {
43
44      implementation("androidx.navigation:navigation-compose:2.9.2")
45
46      implementation(libs.androidx.core.ktx)
47      implementation(libs.androidx.lifecycle.runtime.ktx)
48      implementation(libs.androidx.activity.compose)
49      implementation(platform(dependencyProvider = libs.androidx.compose.bom))
50      implementation(libs.androidx.ui)
51      implementation(libs.androidx.ui.graphics)
52      implementation(libs.androidx.ui.tooling.preview)
53      implementation(libs.androidx.material3)
54      testImplementation(libs.junit)
55      androidTestImplementation(libs.androidx.junit)
56      androidTestImplementation(libs.androidx.espresso.core)
57      androidTestImplementation(platform(dependencyProvider = libs.androidx.compose.bom))
58      androidTestImplementation(libs.androidx.ui.test.junit4)
59      debugImplementation(libs.androidx.ui.tooling)
60      debugImplementation(libs.androidx.ui.test.manifest)
61  }
```

Medium Phone API 36

2:47

≡ Aplikacja z Szufladą

Drawer (szuflada nawigacyjna) to element interfejsu użytkownika, który działa jak wysuwane z boku (najczęściej z lewej krawędzi ekranu) menu.

Plik build.gradle na poziomie **Modułu**

Blok dependencies

Dodajemy zależność

Po dodaniu zależności należy **zsynchronizować projekt**

Strona Główna


```
7  android {
33  }
34  kotlinOptions {
35      jvmTarget = "1.8"
36  }
37  buildFeatures {
38      compose = true
39  }
40  }
41
42  dependencies {
43
44      implementation("androidx.navigation:navigation-compose:2.9.2")
45
46      implementation(libs.androidx.core.ktx)
47      implementation(libs.androidx.lifecycle.runtime.ktx)
48      implementation(libs.androidx.activity.compose)
49      implementation(platform(libs.androidx.compose.bom))
50      implementation(libs.androidx.ui)
51      implementation(libs.androidx.ui.graphics)
52      implementation(libs.androidx.ui.tooling.preview)
53      implementation(libs.androidx.material3)
54      testImplementation(libs.junit)
55      androidTestImplementation(libs.androidx.junit)
56      androidTestImplementation(libs.androidx.espresso.core)
57      androidTestImplementation(platform(libs.androidx.compose.bom))
58      androidTestImplementation(libs.androidx.ui.test.junit4)
59      debugImplementation(libs.androidx.ui.tooling)
60      debugImplementation(libs.androidx.ui.test.manifest)
61  }
```

Drawer (szuflada nawigacyjna) to element interfejsu użytkownika, który działa jak wysuwane z boku (najczęściej z lewej krawędzi ekranu) menu.

Plik `libs.versions.toml`

```
1 [versions]
2   app = "8.11.1"
3   kotlin = "2.0.21"
4   coreKtx = "1.16.0"
5   junit = "4.13.2"
6   junitVersion = "1.2.1"
7   espressoCore = "3.6.1"
8   lifecycleRuntimeKtx = "2.9.2"
9   activityCompose = "1.10.1"
10  composeBom = "2024.09.00"
11  navigationCompose = "2.9.2"
12
13 [libraries]
14 androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
15 androidx-navigation-compose = { module = "androidx.navigation:navigation-compose", version.ref = "navigationCompose" }
16 junit = { group = "junit", name = "junit", version.ref = "junit" }
17 androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
18 androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "espressoCore" }
19 androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version.ref = "lifecycleRuntimeKtx" }
20 androidx-activity-compose = { group = "androidx.activity", name = "activity-compose", version.ref = "activityCompose" }
21 androidx-compose-bom = { group = "androidx.compose", name = "compose-bom", version.ref = "composeBom" }
22 androidx-ui = { group = "androidx.compose.ui", name = "ui" }
23 androidx-ui-graphics = { group = "androidx.compose.ui", name = "ui-graphics" }
24 androidx-ui-tooling = { group = "androidx.compose.ui", name = "ui-tooling" }
25 androidx-ui-tooling-preview = { group = "androidx.compose.ui", name = "ui-tooling-preview" }
26 androidx-ui-test-manifest = { group = "androidx.compose.ui", name = "ui-test-manifest" }
27 androidx-ui-test-junit4 = { group = "androidx.compose.ui", name = "ui-test-junit4" }
28 androidx-material3 = { group = "androidx.compose.material3", name = "material3" }
29
30 [plugins]
31 android-application = { id = "com.android.application", version.ref = "app" }
32 kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
33 kotlin-compose = { id = "org.jetbrains.kotlin.plugin.compose", version.ref = "kotlin" }
```

Drawer (szuflada nawigacyjna) to element interfejsu użytkownika, który działa jak wysuwane z boku (najczęściej z lewej krawędzi ekranu) menu.



The image shows a screenshot of an IDE with the `libs.versions.toml` file open. The file is divided into sections: `[versions]`, `[libraries]`, and `[plugins]`. The `[versions]` section lists various Android libraries and their versions. The `[libraries]` section lists the libraries used in the project, each with a version reference. The `[plugins]` section lists the plugins used in the project.

Annotations highlight specific parts of the file:

- Plik libs.versions.toml**: Points to the file name in the project explorer.
- Dodaj numer wersji**: Points to the version number `"2.0.21"` for `kotlin` in the `[versions]` section.
- Dodaj bibliotekę**: Points to the `androidx-core-ktx` library entry in the `[libraries]` section.

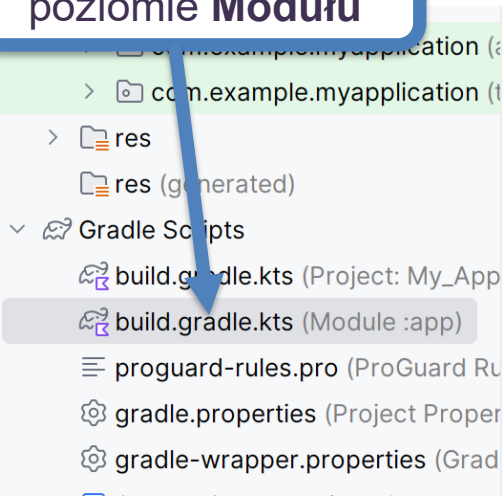
```
[versions]
1  app = "8.11.1"
2  kotlin = "2.0.21"
3  coreKtx = "1.16.0"
4  junit = "4.13.2"
5  junitVersion = "1.2.1"
6  espressoCore = "3.6.1"
7  lifecycleRuntimeKtx = "2.9.2"
8  activityCompose = "1.10.1"
9  composeBom = "2024.09.06"
10 navigationCompose = "2.9.2"

[libraries]
11 androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
12 androidx-navigation-compose = { module = "androidx.navigation:navigation-compose", version.ref = "navigationCompose" }
13 junit = { group = "junit", name = "junit", version.ref = "junit" }
14 androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
15 androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "espressoCore" }
16 androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version.ref = "lifecycleRuntimeKtx" }
17 androidx-activity-compose = { group = "androidx.activity", name = "activity-compose", version.ref = "activityCompose" }
18 androidx-compose-bom = { group = "androidx.compose", name = "compose-bom", version.ref = "composeBom" }
19 androidx-ui = { group = "androidx.compose.ui", name = "ui" }
20 androidx-ui-graphics = { group = "androidx.compose.ui", name = "ui-graphics" }
21 androidx-ui-tooling = { group = "androidx.compose.ui", name = "ui-tooling" }
22 androidx-ui-tooling-preview = { group = "androidx.compose.ui", name = "ui-tooling-preview" }
23 androidx-ui-test-manifest = { group = "androidx.compose.ui", name = "ui-test-manifest" }
24 androidx-ui-test-junit4 = { group = "androidx.compose.ui", name = "ui-test-junit4" }
25 androidx-material3 = { group = "androidx.compose.material3", name = "material3" }

[plugins]
26 android-application = { id = "com.android.application", version.ref = "app" }
27 kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
28 kotlin-compose = { id = "org.jetbrains.kotlin.plugin.compose", version.ref = "kotlin" }
```

Drawer (szuflada nawigacyjna) to element interfejsu użytkownika, który działa jak wysuwane z boku (najczęściej z lewej krawędzi ekranu) menu.

Plik build.gradle na poziomie **Modułu**




Blok **dependencies**

```
dependencies {  
  
    implementation(libs.androidx.navigation.compose)  
  
    implementation(libs.androidx.core.ktx)  
    implementation(libs.androidx.lifecycle.runtime.ktx)  
    implementation(libs.androidx.activity.compose)  
    implementation(platform(dependencyProvider = libs.androidx.compose.bom))  
    implementation(libs.androidx.ui)  
    implementation(libs.androidx.ui.graphics)  
    implementation(libs.androidx.ui.toolina.preview)  
}
```

Dodajemy **zależność**

```
data object AppDestinations {  
    3 Usages  
    const val HOME = "home"  
    2 Usages  
    const val PROFILE = "profile"  
    2 Usages  
    const val SETTINGS = "settings"  
}
```



centralne miejsce do
przechowywania
unikalnych identyfikatorów
naszych ekranów

```
@Composable
fun HomeScreen() {
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        Text( text = "Strona Główna", fontSize = 24.sp)
    }
}
```

1 Usage

```
@Composable
fun ProfileScreen() {
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        Text( text = "Profil Użytkownika", fontSize = 24.sp)
    }
}
```

1 Usage

```
@Composable
fun SettingsScreen() {
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        Text( text = "Ustawienia", fontSize = 24.sp)
    }
}
```

```
data object AppDestinations {
    3 Usages
    const val HOME = "home"
    2 Usages
    const val PROFILE = "profile"
    2 Usages
    const val SETTINGS = "settings"
}
```

centralne miejsce do
przechowywania
unikalnych identyfikatorów
naszych ekranów

Ekrany aplikacji

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun MainApp() {
    val navController = rememberNavController()
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()

    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            DrawerContent(navController = navController, drawerState = drawerState)
        }
    ) {
        Scaffold(...) { paddingValues ->
            // Kontener na ekrany, które będą się zmieniać
            NavHost(...) {
                composable( route = AppDestinations.HOME) { HomeScreen() }
                composable( route = AppDestinations.PROFILE) { ProfileScreen() }
                composable( route = AppDestinations.SETTINGS) { SettingsScreen() }
            }
        }
    }
}
```


Tworzy i "zapamiętuje" instancję **NavController**. Funkcja **remember** zapewnia, że **ten sam** kontroler jest używany podczas **rekompozycji**, co utrzymuje stan nawigacji (np. historię przeglądanych ekranów).

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun MainApp() {
    val navController = rememberNavController()
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()

    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            DrawerContent(navController = navController, drawerState = drawerState)
        }
    ) {
        Scaffold(...) { paddingValues ->
            // Kontener na ekrany, które będą się zmieniać
            NavHost(...) {
                composable( route = AppDestinations.HOME) { HomeScreen() }
                composable( route = AppDestinations.PROFILE) { ProfileScreen() }
                composable( route = AppDestinations.SETTINGS) { SettingsScreen() }
            }
        }
    }
}
```

Tworzy i "zapamiętuje" instancję **NavController**. Funkcja **remember** zapewnia, że **ten sam** kontroler jest używany podczas **rekompozycji**, co utrzymuje stan nawigacji (np. historię przeglądanych ekranów).

Tworzy i „zapamiętuje” **stan szuflady**. **DrawerValue.Closed** ustawia jej początkowy stan na zamknięty.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun MainApp() {
    val navController = rememberNavController()
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()

    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            DrawerContent(navController = navController, drawerState = drawerState)
        }
    ) {
        Scaffold(...) { paddingValues ->
            // Kontener na ekrany, które będą się zmieniać
            NavHost(...) {
                composable( route = AppDestinations.HOME) { HomeScreen() }
                composable( route = AppDestinations.PROFILE) { ProfileScreen() }
                composable( route = AppDestinations.SETTINGS) { SettingsScreen() }
            }
        }
    }
}
```

Tworzy i "zapamiętuje" instancję **NavController**. Funkcja **remember** zapewnia, że **ten sam** kontroler jest używany podczas **rekompozycji**, co utrzymuje stan nawigacji (np. historię przeglądanych ekranów).

Tworzy i „zapamiętuje” **stan szuflady**. **DrawerValue.Closed** ustawia jej początkowy stan na zamknięty.

Uzyskujemy dostęp do **zasięgu korutyn**, który jest powiązany z **cyklem życia kompozycji**. Jest on **niezbędny** do wywoływania **asynchronicznych** funkcji, takich jak **drawerState.open()** i **drawerState.close()**, w odpowiedzi na zdarzenia UI.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun MainApp() {
    val navController = rememberNavController()
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()

    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            DrawerContent(navController = navController, drawerState = drawerState)
        }
    ) {
        Scaffold(...) { paddingValues ->
            // Kontener na ekrany, które będą się zmieniać
            NavHost(...) {
                composable( route = AppDestinations.HOME) { HomeScreen() }
                composable( route = AppDestinations.PROFILE) { ProfileScreen() }
                composable( route = AppDestinations.SETTINGS) { SettingsScreen() }
            }
        }
    }
}
```

Główny kontener implementujący wzorzec szuflady.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun MainApp() {
    val navController = rememberNavController()
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()

    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            DrawerContent(navController = navController, drawerState = drawerState)
        }
    ) {
        Scaffold(...) { paddingValues ->
            // Kontener na ekrany, które będą się zmieniać
            NavHost(...) {
                composable( route = AppDestinations.HOME) { HomeScreen() }
                composable( route = AppDestinations.PROFILE) { ProfileScreen() }
                composable( route = AppDestinations.SETTINGS) { SettingsScreen() }
            }
        }
    }
}
```

Główny kontener implementujący wzorzec szuflady.

Łączy komponent z jego stanem.
Zmiana drawerState (np. na DrawerValue.Open) automatycznie spowoduje **rekompozycję** i wizualne otwarcie szuflady.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun MainApp() {
    val navController = rememberNavController()
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()

    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            DrawerContent(navController = navController, drawerState = drawerState)
        }
    ) {
        Scaffold(...) { paddingValues ->
            // Kontener na ekrany, które będą się zmieniać
            NavHost(...) {
                composable( route = AppDestinations.HOME) { HomeScreen() }
                composable( route = AppDestinations.PROFILE) { ProfileScreen() }
                composable( route = AppDestinations.SETTINGS) { SettingsScreen() }
            }
        }
    }
}
```

Główny kontener implementujący wzorzec szuflady.

Łączy **komponent** z jego **stanem**.
Zmiana drawerState (np. na DrawerValue.Open) automatycznie spowoduje **rekompozycję** i wizualne otwarcie szuflady.

Slot (**@Composable () -> Unit**), w którym definiujemy wygląd i logikę samej szuflady.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun MainApp() {
    val navController = rememberNavController()
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()

    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            DrawerContent(navController = navController, drawerState = drawerState)
        }
    ) {
        Scaffold(...) { paddingValues ->
            // Kontener na ekrany, które będą się zmieniać
            NavHost(...) {
                composable( route = AppDestinations.HOME) { HomeScreen() }
                composable( route = AppDestinations.PROFILE) { ProfileScreen() }
                composable( route = AppDestinations.SETTINGS) { SettingsScreen() }
            }
        }
    }
}
```

Główny kontener implementujący wzorzec szuflady.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun MainApp() {
    val navController = rememberNavController()
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()

    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            DrawerContent(navController = navController, drawerState = drawerState)
        }
    ) {
        Scaffold(...) { paddingValues ->
            // Kontener na ekrany, które będą się zmieniać
            NavHost(...) {
                composable( route = AppDestinations.HOME) { HomeScreen() }
                composable( route = AppDestinations.PROFILE) { ProfileScreen() }
                composable( route = AppDestinations.SETTINGS) { SettingsScreen() }
            }
        }
    }
}
```

Łączy **komponent** z jego **stanem**.
Zmiana drawerState (np. na DrawerValue.Open) automatycznie spowoduje **rekompozycję** i wizualne otwarcie szuflady.

Slot (@Composable () -> Unit), w którym definiujemy wygląd i logikę samej szuflady.

Akcje Nawigacyjne zdefiniowane w innej funkcji

Zmiana stanu szuflady

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun MainApp() {
    val navController = rememberNavController()
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()

    ModalNavigationDrawer(...) {
        Scaffold(
            topBar = {
                TopAppBar(
                    title = { Text( text = "Aplikacja z Szufladą") },
                    navigationIcon = {
                        IconButton(onClick = {
                            scope.launch { drawerState.apply { if (isClosed) open() else close() } }
                        }) { Icon( imageVector = Icons.Filled.Menu, contentDescription = "Menu") }
                    }
                )
            }
        ) { paddingValues ->
            NavHost(
                navController = navController,
                startDestination = AppDestinations.HOME,
                modifier = Modifier.padding(paddingValues)
            ) {
                composable( route = AppDestinations.HOME) { HomeScreen() }
                composable( route = AppDestinations.PROFILE) { ProfileScreen() }
                composable( route = AppDestinations.SETTINGS) { SettingsScreen() }
            }
        }
    }
}
```

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun DrawerContent(navController: NavController, drawerState: DrawerState) {
    val scope = rememberCoroutineScope()

    ModalDrawerSheet {
        Column(modifier = Modifier.padding( all = 16.dp)) {
            Text( text = "Menu", style = MaterialTheme.typography.headlineSmall)
            Spacer(modifier = Modifier.height( height = 16.dp))

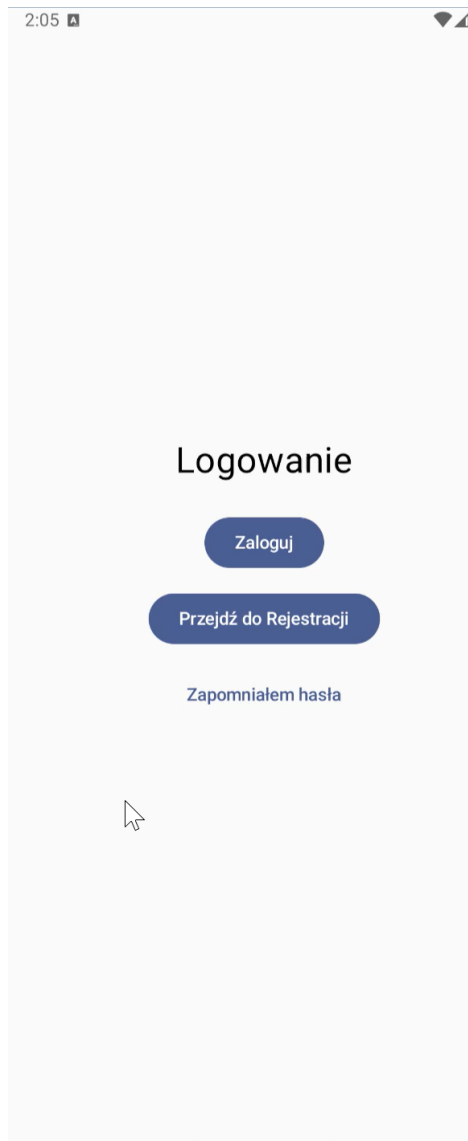
            NavigationDrawerItem(...)
            NavigationDrawerItem(...)
            NavigationDrawerItem(...)
        }
    }
}
```

funkcja kompozycyjna
(@Composable) z biblioteki
Material 3 w Jetpack Compose,
która definiuje wygląd i kontener
samej wysuwanej szuflady.

Komponent UI reprezentujący
pojedynczy element w
szufladzie.

```
NavigationDrawerItem(  
    icon = { Icon( imageVector = Icons.Default.Home, contentDescription = "Strona główna") },  
    label = { Text( text = "Strona główna") },  
    selected = false,  
    onClick = {  
        navController.navigate( route = AppDestinations.HOME)  
        scope.launch { drawerState.close() }  
    }  
)
```

Zagnieżdżona nawigacja jest używana do organizowania i **izolowania** powiązanych ze sobą ekranów w samodzielne **grafy**.



Zagnieżdżona nawigacja jest używana do organizowania i **izolowania** powiązanych ze sobą ekranów w samodzielne **grafy**.

Identyfikatory dla grafów.
Traktujemy jak zwykłe ekrany

Ścieżki do ekranów
powiązanych z **grafem**
AUTH_GRAPH.

Ścieżki do ekranów
powiązanych z **grafem**
MAIN_APP_GRAPH.

```
data object AppDestinations {  
    // Grafy  
    3 Usages  
    const val AUTH_GRAPH = "auth_graph"  
    2 Usages  
    const val MAIN_APP_GRAPH = "main_app_graph"  
  
    // Ekran autentykacji  
    2 Usages  
    const val LOGIN = "login"  
    2 Usages  
    const val REGISTER = "register"  
    2 Usages  
    const val FORGOT_PASSWORD = "forgot_password"  
  
    // Główne ekrany aplikacji  
    2 Usages  
    const val WELCOME = "welcome"  
    2 Usages  
    const val PROFILE = "profile"  
}
```

Główny NavHost. Jego zadaniem **nie jest zarządzanie** pojedynczymi ekranami, ale decydowanie, który **zagnieżdżony graf** jest aktualnie aktywny

```
@Composable
```

```
fun SimpleNestedNavApp() {
```

```
    val navController = rememberNavController()
```

```
    NavHost(
```

```
        navController = navController,
```

```
        startDestination = AppDestinations.AUTH_GRAPH
```

```
    ) {
```

```
        // Graf autentykacji (logowanie, rejestracja, itp.)
```

```
        authGraph(navController)
```

```
        // Główny graf aplikacji po zalogowaniu
```

```
        mainAppGraph(navController)
```

```
    }
```

```
}
```

Główny NavHost. Jego zadaniem **nie jest zarządzanie** pojedynczymi ekranami, ale decydowanie, który **zagnieżdżony graf** jest aktualnie aktywny

```
@Composable
fun SimpleNestedNavApp() {
    val navController = rememberNavController()
    NavHost(
        navController = navController,
        startDestination = AppDestinations.AUTH_GRAPH
    ) {
        // Graf autentykacji (logowanie, rejestracja, itp.)
        authGraph(navController)

        // Główny graf aplikacji po zalogowaniu
        mainAppGraph(navController)
    }
}
```

Punkt startowy **aplikacji.** Po włączeniu aplikacji użytkownik trafia do **przepływu logowania**

Główny NavHost. Jego zadaniem **nie jest zarządzanie** pojedynczymi ekranami, ale decydowanie, który **zagnieżdżony graf** jest aktualnie aktywny

Punkt startowy **aplikacji.** Po włączeniu aplikacji użytkownik trafia do **przepływu logowania**

```
@Composable
fun SimpleNestedNavApp() {
    val navController = rememberNavController()

    NavHost(
        navController = navController,
        startDestination = AppDestinations.AUTH_GRAPH
    ) {
        // Graf autentykacji (logowanie, rejestracja, itp.)
        authGraph(navController)

        // Główny graf aplikacji po zalogowaniu
        mainAppGraph(navController)
    }
}
```

Mamy **jeden główny** NavHost, który zawiera **oba grafy** (authGraph i mainAppGraph).

NavHost jest jak **cała mapa** aplikacji.

NavController to **GPS**, który porusza się po tej mapie.

Główny NavHost. Jego zadaniem **nie jest zarządzanie** pojedynczymi ekranami, ale decydowanie, który **zagnieżdżony graf** jest aktualnie aktywny

```
@Composable
fun SimpleNestedNavApp() {
    val navController = rememberNavController()
    NavHost(
        navController = navController,
        startDestination = AppDestinations.AUTH_GRAPH
    ) {
        // Graf autentykacji (logowanie, rejestracja, itp.)
        authGraph(navController)
        // Główny graf aplikacji po zalogowaniu
        mainAppGraph(navController)
    }
}
```

Punkt startowy **aplikacji.** Po włączeniu aplikacji użytkownik trafia do **przepływu logowania**

Metody tworzące grafy

Mamy **jeden główny** NavHost, który zawiera **oba grafy** (authGraph i mainAppGraph).

NavHost jest jak **cała mapa** aplikacji.

NavController to **GPS**, który porusza się po tej mapie.

Funkcja rozszerzająca

Punkt startowy grafu

Ścieżka do grafu

Definicja **wszystkich**
ekranów obecnych w
tym grafie

```
fun NavGraphBuilder.authGraph(navController: NavController) {  
    navigation(  
        startDestination = AppDestinations.LOGIN,  
        route = AppDestinations.AUTH_GRAPH  
    ) {  
        composable( route = AppDestinations.LOGIN) {  
            LoginScreen(navController) }  
        composable( route = AppDestinations.REGISTER) {  
            RegisterScreen(navController) }  
        composable( route = AppDestinations.FORGOT_PASSWORD) {  
            ForgotPasswordScreen(navController) }  
    }  
}
```

Funkcja rozszerzająca

Punkt startowy grafu

Ścieżka do grafu

Definicja **wszystkich**
ekranów obecnych w
tym grafie

```
fun NavGraphBuilder.mainAppGraph(navController: NavController) {  
    navigation(  
        startDestination = AppDestinations.WELCOME,  
        route = AppDestinations.MAIN_APP_GRAPH  
    ) {  
        composable( route = AppDestinations.WELCOME) {  
            WelcomeScreen(navController) }  
        composable( route = AppDestinations.PROFILE) {  
            ProfileScreen(navController) }  
    }  
}
```

@Composable

```
fun LoginScreen(navController: NavController) {  
    Column(  
        modifier = Modifier.fillMaxSize().padding(all = 16.dp),  
        verticalArrangement = Arrangement.Center,  
        horizontalAlignment = Alignment.CenterHorizontally  
    ) {  
        Text(text = "Logowanie", fontSize = 28.sp)  
        Spacer(Modifier.height(height = 24.dp))  
        Button(onClick = { navController.navigateToMainApp() }) {  
            Text(text = "Zaloguj")  
        }  
        Spacer(Modifier.height(height = 12.dp))  
        Button(onClick = { navController.navigate(route = AppDestinations.REGISTER) }) {  
            Text(text = "Przejdź do Rejestracji")  
        }  
        Spacer(Modifier.height(height = 12.dp))  
        TextButton(onClick = { navController.navigate(route = AppDestinations.FORGOT_PASSWORD) }) {  
            Text(text = "Zapomniałem hasła")  
        }  
    }  
}
```

Funkcja rozszerzająca
definiująca nawigację
przez grafy.

Funkcja rozszerzająca

Przenieś do głównego grafu aplikacji

```
fun NavController.navigateToMainApp() {  
    this.navigate( route = AppDestinations.MAIN_APP_GRAPH) {  
        popUpTo( route = AppDestinations.AUTH_GRAPH) {  
            inclusive = true  
        }  
    }  
}
```

Powrót na stosie nawigacji **do początku** AUTH_GRAPH. Usuń go.

Usuń graf **z całą zawartością** – czyli **wszystkimi** ekranami przez które przechodziliśmy poruszając się po AUTH_GRAPH.