KATEDRA INFORMATYKI I AUTOMATYKI

Programowanie Aplikacji Mobilnych

Informatyka

ĆWICZENIE 2

Nawigacja i ViewModel

Instrukcja Laboratoryjna Nr 2

Nawigacja między ekranami, ViewModel i wzorzec MVVM

Przedmiot: Programowanie Aplikacji Mobilnych

Czas realizacji: 90 minut • Forma: indywidualna • Wymaganie wstępne: ukończone ćwiczenie 1

Cel ćwiczenia

Po ukończeniu ćwiczenia student będzie umieć:

- ✓ zdefiniować grafy nawigacji w *Jetpack Compose*
- ✓ przekazywać dane między ekranami przez argumenty tras
- ✓ tworzyć ViewModel i zarządzać stanem UI
- ✓ korzystać ze StateFlow i collectAsStateWithLifecycle
- ✓ zbudować wieloekranową aplikację MVVM
- ✓ obsłużyć przycisk Wstecz i back stack
- ✓ dodać zależności przez Version Catalog (libs.toml)

Wymagania wstępne

Student powinien znać z ćwiczenia 1:

- **@Composable** *i* budowa drzewa *UI Compose,* stan lokalny
- Struktura projektu Android (*Manifest, build.gradle*)
- Uruchamianie aplikacji na emulatorze

Nowe pojęcia w tym ćwiczeniu

- `NavController`, `NavHost`, `composable<Route>`
- `ViewModel`, `viewModels();` Jetpack Lifecycle
- `StateFlow`, `collectAsStateWithLifecycle()`

MVVM *(Model–View–ViewModel)*

Spis treści

1. Wzorzec MVVM: teoria i praktyka

Wzorzec MVVM (Model–View–ViewModel) jest oficjalnie rekomendowaną architekturą aplikacji Android przez Google. Rozdziela odpowiedzialności na trzy warstwy, dzięki czemu kod jest testowalny i utrzymywalny.

1.1. Trzy warstwy MVVM

WARSTWA VIEW (ekrany / @Composable)

```
MainActivity, CounterScreen, ListScreen, DetailScreen
Obserwuje StateFlow z ViewModel przez collectAsStateWithLifecycle()
NIE zawiera logiki biznesowej – tylko renderuje stan i deleguje zdarzenia
Przykład: Button(onClick = { viewModel.addTask(text) })
```

WARSTWA VIEWMODEL (logika prezentacji)

```
Klasa dziedzicząca po ViewModel() – przeżywa rotację ekranu!
Zawiera StateFlow<UiState> jako źródło prawdy (Single Source of Truth)
Wywołuje metody Repository, przetwarza wyniki, emituje nowy stan
Przykład: fun addTask(text: String) { _tasks.update { it + Task(text) } }
```

WARSTWA MODEL (dane i logika domenowa)

```
Repository – abstrahuje źródło danych (Room, API, SharedPreferences)
Data classes – czyste obiekty danych bez logiki Androida
Use Cases (opcjonalne) – enkapsulacja reguł biznesowych
Przykład: data class Task(val id: UUID, val text: String, val done: Boolean)
```

Dlaczego ViewModel przeżywa rotację ekranu?

ViewModel jest przechowywany w ViewModelStore, który jest powiązany z Activity, nie z jej instancją. Gdy ekran się obraca, Android niszczy i tworzy na nowo Activity, ale ViewModelStore pozostaje. ViewModel żyje dopóki Activity nie zostanie faktycznie zakończona (finish() lub Back).

Ważne: nie przechowuj w ViewModel referencji do `Context`, `View` ani `Activity` bo to powoduje memory leaks! Jeśli potrzebujesz Contextu, użyj `AndroidViewModel(application)`.

1.2. Przepływ danych w MVVM

```
// ──────  JEDNOKIERUNKOWY PRZEPŁYW DANYCH (Unidirectional Data Flow – UDF diagram) ──────  //
//   USER ACTION                  EVENT                       UI STATE
//  ┌──────────┐    ──event──>  ┌──────────┐   ──state──>   ┌──────────┐
//  │  Button  │                │ ViewModel │               │  Screen  │
//  │  click   │   <──state──   │ StateFlow │   <──update──  │  @Comp.  │
//  └──────────┘                └──────────┘                └──────────┘  //
//  1. User klika przycisk w View
//  2. View wywołuje metodę ViewModel (event)
//  3. ViewModel aktualizuje StateFlow (state)
//  4. View obserwuje StateFlow i przerysowuje się (recomposition)
//  5. Użytkownik widzi nowy stan UI //
// Zasada: STATE idzie w DÓŁ (ViewModel → View)
//         EVENTY idą w GÓRĘ (View → ViewModel)

// Przykład – sealed class dla stanu UI (zalecany wzorzec)
sealed class TaskUiState {
    object Loading : TaskUiState()
    data class Success(val tasks: List<Task>, val filter: Filter = Filter.ALL) : TaskUiState()
    data class Error(val message: String) : TaskUiState()
}
```

2. Konfiguracja projektu: zależności

Rozpocznij od projektu HelloMobile z Ćwiczenia 1 lub utwórz nowy projekt Empty Activity.

Dodaj wymagane biblioteki Jetpack.

2.1. libs.versions.toml: version catalog

```
TOML – libs.versions.toml
# gradle/libs.versions.toml – dodaj/zweryfikuj te wpisy
[versions]
# Kotlin i Compose (sprawdź aktualne wersje na developer.android.com)
[versions]
kotlin = "2.0.21"
agp = "8.7.3"
compose-bom = "2024.11.00"
lifecycle = "2.8.7"
navigation-compose = "2.8.4"
activity-compose = "1.9.3"
kotlinx-serialization = "1.7.3"

# Compose BOM – zarządza wersjami całego ekosystemu
[libraries.compose-bom]
group = "androidx.compose"
name = "compose-bom"
[libraries.compose-bom.version]
ref = "compose-bom"

[libraries.compose-ui]
group = "androidx.compose.ui"
name = "ui"

[libraries.compose-ui-tooling]
group = "androidx.compose.ui"
name = "ui-tooling"

[libraries.compose-material3]
group = "androidx.compose.material3"
name = "material3"

[libraries.compose-foundation]
group = "androidx.compose.foundation"
name = "foundation"

[libraries.compose-runtime]
group = "androidx.compose.runtime"
name = "runtime"

# Lifecycle + ViewModel
[libraries.lifecycle-viewmodel-compose]
group = "androidx.lifecycle"
name = "lifecycle-viewmodel-compose"

[libraries.lifecycle-viewmodel-compose.version]
ref = "lifecycle"

[libraries.lifecycle-runtime-compose]
group = "androidx.lifecycle"
name = "lifecycle-runtime-compose"
[libraries.lifecycle-runtime-compose.version]
ref = "lifecycle"

# Navigation Compose
[libraries.navigation-compose]
group = "androidx.navigation"
name = "navigation-compose"
[libraries.navigation-compose.version]
```

```
ref = "navigation-compose"

# Activity Compose
[libraries.activity-compose]
group = "androidx.activity"
name = "activity-compose"
[libraries.activity-compose.version]
ref = "activity-compose"

# Kotlin Serialization (type-safe routes)
[libraries.kotlinx-serialization-json]
group = "org.jetbrains.kotlinx"
name = "kotlinx-serialization-json"
[libraries.kotlinx-serialization-json.version]
ref = "kotlinx-serialization"

[plugins.android-application]
id = "com.android.application"
[plugins.android-application.version]
ref = "agp"

[plugins.kotlin-android]
id = "org.jetbrains.kotlin.android"
[plugins.kotlin-android.version]
ref = "kotlin"

[plugins.compose-compiler]
id = "org.jetbrains.kotlin.plugin.compose"
[plugins.compose-compiler.version]
ref = "kotlin"

[plugins.kotlin-serialization]
id = "org.jetbrains.kotlin.plugin.serialization"
[plugins.kotlin-serialization.version]
ref = "kotlin"
```

2.2. build.gradle.kts (app): blok dependencies

Kotlin DSL — build.gradle.kts

```
// app/build.gradle.kts — sprawdź/dodaj poniższe
 plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.compose.compiler)
    alias(libs.plugins.kotlin.serialization)
   type-safe routes // ← nowe
 }
 android {
    compileSdk = 35
    defaultConfig {
        minSdk = 26
        targetSdk = 35
        // ...
    }
}
 dependencies {
    // Compose BOM — importuj PRZED innymi artefaktami
Compose
    implementation(platform(libs.compose.bom))
    implementation(libs.compose.ui)
    implementation(libs.compose.material3)
    implementation(libs.compose.foundation)
    implementation(libs.compose.runtime)
    debugImplementation(libs.compose.ui.tooling)
     // Activity + Lifecycle
    implementation(libs.activity.compose)
    implementation(libs.lifecycle.viewmodel.compose)
    implementation(libs.lifecycle.runtime.compose)
  // collectAsStateWithLifecycle
     // Navigation Compose
    implementation(libs.navigation.compose)
     // Kotlin Serialization (type-safe routes)
    implementation(libs.kotlinx.serialization.json)
}
```

#	Akcja	Synchronizacja Gradle po dodaniu zależności
1	**Sync Now**	Po edycji pliku gradle Android Studio pokazuje baner: "Gradle files have changed" → kliknij **Sync Now**. Czas: 30–120 sekund.
2	**Weryfikacja**	Otwórz Build → Build Output. Brak błędów BUILD SUCCESSFUL = sukces. Jeśli błąd: najczęstszy powód to literówka w nazwie biblioteki w .toml.
3	**Problemy**	Jeśli sync nie zakończy się powodzeniem: File → Invalidate Caches → Invalidate and Restart. Po restarcie: File → Sync Project with Gradle Files.

3. Nawigacja Compose: NavController i NavHost

Navigation Compose to oficjalna biblioteka Google do zarządzania nawigacją w aplikacjach Jetpack Compose. Obsługuje back stack, deep links, animacje przejść i type-safe routes.

3.1. Kluczowe koncepty nawigacji

	Mózg nawigacji. Zarządza back stackiem. Tworzysz przez . Nigdy nie przekazuj do ViewModel — to widok!
	Composable 'kontener' gdzie wyświetlane są ekrany. Definiujesz w nim graf nawigacji — mapowanie trasa → ekran.
	Adnotacja Kotlinx Serialization na klasie trasy. Zastępuje stare stringowe ścieżki ("screen/{id}"). Type-safe: kompilator wykrywa błędy.
	Przejście do nowego ekranu. Opcje: (wyczyszcz stos), (nie duplikuj). Przekazujesz obiekt Route.
	Powrót do poprzedniego ekranu. Odpowiednik systemowego przycisku Wstecz. Bezpieczniejszy niż .
	Przechwytuje systemowy przycisk Wstecz w Compose. Używaj oszczędnie — tylko gdy potrzebujesz niestandardowego zachowania.

3.2. Definiowanie tras: type-safe routes

Utwórz plik Routes.kt w pakiecie aplikacji

KOTLIN - Routes.kt

```
// Routes.kt – definicje tras nawigacji
package pl.edu.prz.taskapp
import kotlinx.serialization.Serializable
// ── TRASY BEZ PARAMETRÓW ──────────────────────────────────────
@Serializable object HomeRoute              // ekran główny (lista zadań)
@Serializable object AddTaskRoute           // ekran dodawania nowego zadania
@Serializable object SettingsRoute          // ekran ustawień
// ── TRASY Z PARAMETRAMI ───────────────────────────────────────
```

```
@Serializable data class TaskDetailRoute(
    val taskId: String                      // przekazujemy ID jako String (UUID.toString())
)
// ── SEALED CLASS dla grupowania tras (opcjonalnie) ──────────────────────────
// Przydatna przy BottomNavigation — mapowanie ikona ↔ trasa
sealed class BottomNavRoute(val label: String) {
    object Home     : BottomNavRoute("Zadania")
    object Settings : BottomNavRoute("Ustawienia") }
// ── PRZYKŁAD użycia w NavHost
composable<HomeRoute>  { HomeScreen(navController) }
composable<TaskDetailRoute> { backStackEntry ->
    val route = backStackEntry.toRoute<TaskDetailRoute>()
    TaskDetailScreen(taskId = route.taskId)  }
```

3.3. NavHost: budowa grafu nawigacji

Kotlin — AppNavigation.kt

```
// AppNavigation.kt — centralny graf nawigacji
package pl.edu.prz.taskapp
import androidx.compose.runtime.Composable
import androidx.navigation.NavHostController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import androidx.navigation.toRoute

object Routes {
  const val HOME = "home"
  const val ADD_TASK = "add_task"
  const val TASK_DETAIL = "task_detail/{taskId}"
    fun taskDetail(taskId: String) = "task_detail/$taskId"
}
  @Composable
  fun AppNavigation(
 val navController = rememberNavController()
     NavHost(
       navController = navController,
       startDestination = Routes.HOME
     ) {
    // ── Ekran główny ──────────────────────────────────────────
  composable(Routes.HOME) {
  HomeScreen(
    onNavigateToAdd = {
      navController.navigate(Routes.ADD_TASK)
  },
    onNavigateToDetail = { id ->
      navController.navigate(
        Routes.taskDetail(id)
      )
    }
  )
  composable(Routes.ADD_TASK) {
    AddTaskScreen(
      onTaskAdded = {
        navController.popBackStack()
      }
    )
 }

composable<HomeRoute> {
            HomeScreen(
                OnNavigateToAdd = { navController.navigate(AddTaskRoute) },
                onNavigateToDetail = { taskId ->
                    navController.navigate(TaskDetailRoute(taskId))
                }
            )
        }
        // ── Dodawanie zadania ─────────────────────────────────────
        composable<AddTaskRoute> {
            AddTaskScreen(
                onTaskAdded = {                   // Po dodaniu: wróć do Home i wyczyść AddTask ze stosu
                    navController.navigate(HomeRoute) {
                        popUpTo(HomeRoute) { inclusive = false }
                    }
```

```
                },
                onNavigateBack = { navController.navigateUp() }
            )
        }
        // ── Szczegóły zadania – z parametrem ──────────────────────
        composable<TaskDetailRoute> { backStackEntry ->
            val route = backStackEntry.toRoute<TaskDetailRoute>()
            TaskDetailScreen(
                taskId = route.taskId,
                onNavigateBack = { navController.navigateUp() }
            )
        }
        // ── Ustawienia ────────────────────────────────────────────
        composable<SettingsRoute> {
            SettingsScreen(onNavigateBack = { navController.navigateUp() })
        }
    }
}
```

> ⚠ **NavController tylko w Composable; nie w ViewModel!**
>
> NavController jest elementem warstwy View i **nigdy** nie powinien być przekazywany do ViewModel. ViewModel nie wie nic o nawigacji.
>
> Poprawny wzorzec: ViewModel emituje zdarzenie (np. NavigateToDetail(id)), ekran obserwuje ten event i sam wywołuje navController.navigate().
>
> Możesz użyć `SharedFlow<UiEffect>` jako kanał zdarzeń jednorazowych (one-shot events), np. nawigacja, snackbar, dialog.

4. ViewModel i StateFlow: zarządzanie stanem

4.1. Tworzenie ViewModel

KOTLIN - TaskViewModel.kt

```
package pl.edu.prz.taskapp
import androidx.lifecycle.ViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.update
import java.util.UUID

// ── MODEL – data class ──────────────────────────────────────────
data class Task(
    val id: String = UUID.randomUUID().toString(),
    val text: String,     val isDone: Boolean = false,
    val priority: Priority = Priority.NORMAL,
)
 enum class Priority { LOW, NORMAL, HIGH }  enum class Filter { ALL, ACTIVE, DONE }

// ── UI STATE – sealed class ─────────────────────────────────────
data class TaskListUiState(
    val tasks: List<Task> = emptyList(),
    val filter: Filter = Filter.ALL,
    val isLoading: Boolean = false,
    val errorMessage: String? = null,
)

// Computed property – przefiltrowane zadania (nie przechowujemy osobno!)
val TaskListUiState.filteredTasks: List<Task>
    get() = when (filter) {
        Filter.ALL    -> tasks
        Filter.ACTIVE -> tasks.filter { !it.isDone }
        Filter.DONE   -> tasks.filter { it.isDone }
    }
```

```
// ── VIEWMODEL ──────────────────────────────────────────────────────
 class TaskViewModel : ViewModel() {
     // _uiState – prywatny, mutowalny (tylko ViewModel może modyfikować)
     private val _uiState = MutableStateFlow(TaskListUiState())
     // uiState – publiczny, read-only (View może tylko obserwować)
     val uiState: StateFlow<TaskListUiState> = _uiState.asStateFlow()

     // ── Akcje (eventy z View) ──────────────────────────────────────
      fun addTask(text: String) {
         if (text.isBlank()) return
           // walidacja
         _uiState.update { current ->
             current.copy(tasks = current.tasks + Task(text = text.trim()))
         }
     }
      fun toggleTask(taskId: String) {
         _uiState.update { current ->
             current.copy(tasks = current.tasks.map { task ->
                 if (task.id == taskId) task.copy(isDone = !task.isDone) else task
             })
         }
     }
      fun deleteTask(taskId: String) {
         _uiState.update { current ->
             current.copy(tasks = current.tasks.filter { it.id != taskId })
         }
     }
      fun setFilter(filter: Filter) {
         _uiState.update { it.copy(filter = filter) }
     }
      fun setPriority(taskId: String, priority: Priority) {
         _uiState.update { current ->
             current.copy(tasks = current.tasks.map { task ->
                 if (task.id == taskId) task.copy(priority = priority) else task
             })
         }
     }

// Dane przykładowe (do testów – usuń w produkcji)
     init {
         _uiState.update { it.copy(tasks = listOf(
             Task(text = "Skonfigurować Android Studio", isDone = true),
             Task(text = "Napisać pierwszą aplikację Compose"),
             Task(text = "Nauczyć się ViewModel i StateFlow"),
             Task(text = "Zbudować aplikację z nawigacją", priority = Priority.HIGH),
         ))}
     }
 }
```

4.2. Podłączenie *ViewModel* do *Composable*

KOTLIN - HomeScreen.kt (fragment)

```
// HomeScreen.kt – obserwacja StateFlow
 package pl.edu.prz.taskapp
 import androidx.compose.runtime.getValue
 import androidx.lifecycle.compose.collectAsStateWithLifecycle
 import androidx.lifecycle.viewmodel.compose.viewModel
  @Composable fun HomeScreen(
     onNavigateToAdd: () -> Unit,
     onNavigateToDetail: (taskId: String) -> Unit,
     // Domyślny ViewModel – tworzony raz na czas życia Activity:
     viewModel: TaskViewModel = viewModel() ) {
     // collectAsStateWithLifecycle – zatrzymuje kolekcję gdy app idzie w tło!
     // Bezpieczniejsze niż collectAsState() – oszczędza baterię.
     val uiState by viewModel.uiState.collectAsStateWithLifecycle()
      // Używamy przefiltrowanych zadań (computed property)
     val tasks = uiState.filteredTasks
      Scaffold(
         topBar = {
             TopAppBar(
                 title = { Text("Moje Zadania (${tasks.size})") },
                 actions = {
                     IconButton(onClick = onNavigateToAdd) {
                         Icon(Icons.Default.Add, contentDescription = "Dodaj")
```

```
                    }
                }
            )
        }
    ) { paddingValues ->
        Column(modifier = Modifier.padding(paddingValues)) {
             // FilterChips – zmiana filtra przez ViewModel
            FilterRow(
                currentFilter = uiState.filter,
                onFilterChange = viewModel::setFilter    // method reference
            )
            // Lista zadań
            LazyColumn(modifier = Modifier.fillMaxSize()) {
                items(
                    items = tasks,
                    key = { it.id }    // key = stabilna identyfikacja dla animacji
                ) { task ->
                    TaskItem(
                        task = task,
                        onToggle = { viewModel.toggleTask(task.id) },
                        onDelete = { viewModel.deleteTask(task.id) },
                        onClick  = { onNavigateToDetail(task.id) }
                    )
                }
            }
        }
    }
}
```

collectAsStateWithLifecycle vs collectAsState

collectAsStateWithLifecycle() (zalecane) — zatrzymuje kolekcję gdy app przechodzi do stanu STOPPED (np. idzie w tło, ekran się wyłącza). Oszczędza baterię i CPU. Wymaga: `lifecycle-runtime-compose`.

collectAsState() — kolekcja jest aktywna nawet gdy app jest w tle. Prostsze w użyciu, ale nieoptymalne. Używaj tylko w preview lub testach.

Reguła: zawsze używaj collectAsStateWithLifecycle() w produkcyjnym kodzie.

5. Budowanie aplikacji Task Manager

Zbudujemy kompletną aplikację zarządzania zadaniami. Nazwa projektu: TaskApp.

Składać się będzie ona z 4 ekranów połączonych nawigacją i współdzielącego ViewModel.

5.1. Struktura projektu

Struktura projektu TaskApp

```
TaskApp/
└── app/src/main/java/pl/edu/prz/taskapp/
    ├── MainActivity.kt          ← setContent { AppNavigation() }
    ├── Routes.kt                ← definicje tras (@Serializable)
    ├── AppNavigation.kt         ← NavHost z grafem nawigacji
    ├── TaskViewModel.kt         ← ViewModel + Task model + UiState
    └── ui/
        ├── HomeScreen.kt        ← lista zadań (LazyColumn)
        ├── AddTaskScreen.kt     ← formularz dodawania
        ├── TaskDetailScreen.kt  ← szczegóły zadania
        └── components/
            ├── TaskItem.kt      ← pojedyncze zadanie (reużywalne)
            └── FilterRow.kt     ← FilterChips do filtrowania
```

5.2. MainActivity.kt

KOTLIN - MainActivity.kt

```
// MainActivity.kt – punkt wejścia, startuje nawigację
package pl.edu.prz.taskapp
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import pl.edu.prz.taskapp.ui.theme.TaskAppTheme
  class MainActivity : ComponentActivity() {
     override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
         enableEdgeToEdge()
         setContent {
             TaskAppTheme {
                 // Cały graf nawigacji – AppNavigation zarządza NavController
                 AppNavigation()
             }
         }
     }
 }
 // WAŻNE: ViewModel jest tworzony przez viewModel() w HomeScreen/AddTaskScreen
 // WAŻNE: Ten sam ViewModel jest współdzielony gdy przekazujesz go w dół
 // Alternatywa dla współdzielenia: hiltViewModel() lub ViewModelProvider z Activity scope
```

5.3. Ekran główny: HomeScreen z LazyColumn

KOTLIN - HomeScreen.kt

```
// ui/HomeScreen.kt – kompletna implementacja
package pl.edu.prz.taskapp.ui
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import androidx.lifecycle.viewmodel.compose.viewModel
import pl.edu.prz.taskapp.*
@OptIn(ExperimentalMaterial3Api::class)
 @Composable fun HomeScreen(
     onNavigateToAdd: () -> Unit,
     onNavigateToDetail: (String) -> Unit,
     viewModel: TaskViewModel = viewModel() ) {
     val uiState by viewModel.uiState.collectAsStateWithLifecycle()
     val tasks = uiState.filteredTasks
      Scaffold(
         topBar = {
             TopAppBar(
                 title = { Text("Zadania") },
                 colors = TopAppBarDefaults.topAppBarColors(
                     containerColor = MaterialTheme.colorScheme.primaryContainer
                 ),
                 actions = {
                     Text("${tasks.count { it.isDone }}/${uiState.tasks.size}",
                          style = MaterialTheme.typography.bodyMedium,
                          modifier = Modifier.padding(end = 8.dp))
                     IconButton(onClick = onNavigateToAdd) {
                         Icon(Icons.Default.Add, "Dodaj zadanie")
                     }
                 }
             )
         }
     ) { innerPadding ->
         Column(modifier = Modifier.fillMaxSize().padding(innerPadding)) {
             // FilterRow – selektor filtra
```

```
            FilterRow(
                currentFilter = uiState.filter,
                onFilterChange = viewModel::setFilter
            )
            // Pusta lista
            if (tasks.isEmpty()) {
                Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
                    Column(horizontalAlignment = Alignment.CenterHorizontally) {
                        Text("Brak zadań", style = MaterialTheme.typography.titleMedium,
                             color = MaterialTheme.colorScheme.onSurfaceVariant)
                        Spacer(Modifier.height(8.dp))
                        Text(if (uiState.filter == Filter.ALL) "Dotknij + aby dodać pierwsze zadanie"
                             else "Brak zadań w tej kategorii",
                             style = MaterialTheme.typography.bodySmall,
                             color = MaterialTheme.colorScheme.onSurfaceVariant)
                    }
                }
            } else {
                LazyColumn(
                    modifier = Modifier.fillMaxSize(),
                    contentPadding = PaddingValues(vertical = 8.dp)
                ) {
                    items(items = tasks, key = { it.id }) { task ->
                        TaskItem(
                            task     = task,
                            onToggle = { viewModel.toggleTask(task.id) },
                            onDelete = { viewModel.deleteTask(task.id) },
                            onClick  = { onNavigateToDetail(task.id) }
                        )
                    }
                }
            }
        }
    }
}
```

5.4. Komponent TaskItem.kt

KOTLIN - TaskItem.kt

```
// ui/components/TaskItem.kt — reużywalny komponent zadania
package pl.edu.prz.taskapp.ui.components
import androidx.compose.animation.animateContentSize
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.style.TextDecoration
import androidx.compose.ui.unit.dp
import pl.edu.prz.taskapp.Priority
import pl.edu.prz.taskapp.Task
  @Composable fun TaskItem(
     task: Task,
     onToggle: () -> Unit,
     onDelete: () -> Unit,
     onClick: () -> Unit,
     modifier: Modifier = Modifier ) {
     val priorityColor = when (task.priority) {
         Priority.HIGH   -> MaterialTheme.colorScheme.error
         Priority.NORMAL -> MaterialTheme.colorScheme.primary
         Priority.LOW    -> MaterialTheme.colorScheme.onSurfaceVariant
     }
      Card(
         modifier = modifier
             .fillMaxWidth()
             .padding(horizontal = 16.dp, vertical = 4.dp)
             .animateContentSize()
       // animuje zmiany rozmiaru karty
             .clickable(onClick = onClick),
colors = CardDefaults.cardColors(
containerColor = if (task.isDone)
                 MaterialTheme.colorScheme.surfaceVariant
```

```
            else MaterialTheme.colorScheme.surface
        ),
        elevation = CardDefaults.cardElevation(defaultElevation = if (task.isDone) 0.dp else 2.dp)
    ) {
        Row(
            modifier = Modifier.padding(horizontal = 8.dp, vertical = 12.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            // Priorytet – kolorowy pasek
            Box(modifier = Modifier.width(4.dp).height(40.dp)
                .padding(end = 0.dp), // handled by Spacer
            )
            Checkbox(
                checked = task.isDone,
                onCheckedChange = { onToggle() },
                colors = CheckboxDefaults.colors(
                    checkedColor = MaterialTheme.colorScheme.primary
                )
            )
            Spacer(Modifier.width(8.dp))
            Text(
                text = task.text,
                modifier = Modifier.weight(1f),
                style = MaterialTheme.typography.bodyLarge.copy(
                    textDecoration = if (task.isDone) TextDecoration.LineThrough else null,
                    color = if (task.isDone) MaterialTheme.colorScheme.onSurfaceVariant
                            else MaterialTheme.colorScheme.onSurface
                )
            )
            // Ikona priorytetu
            if (task.priority == Priority.HIGH) {
                Text("!", color = priorityColor,
                     style = MaterialTheme.typography.titleMedium,
                     modifier = Modifier.padding(horizontal = 4.dp))
            }
            IconButton(onClick = onDelete) {
                Icon(Icons.Default.Delete, "Usuń",
                     tint = MaterialTheme.colorScheme.onSurfaceVariant)
            }
        }
    }
}
```

5.5. Ekran AddTask: formularz z walidacją

AddTaskScreen.kt

```
// ui/AddTaskScreen.kt package pl.edu.prz.taskapp.ui
import androidx.compose.foundation.layout.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material3.*
import androidx.compose.runtime.* import androidx.compose.ui.Modifier
import androidx.compose.ui.focus.FocusRequester
import androidx.compose.ui.focus.focusRequester
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import pl.edu.prz.taskapp.Priority
import pl.edu.prz.taskapp.TaskViewModel
 @OptIn(ExperimentalMaterial3Api::class)
 @Composable fun AddTaskScreen(
     onTaskAdded: () -> Unit,
     onNavigateBack: () -> Unit,
     viewModel: TaskViewModel = viewModel() ) {
     // Stan lokalny – formularz to stan View, nie ViewModel
     var taskText by remember { mutableStateOf("") }
     var selectedPriority by remember { mutableStateOf(Priority.NORMAL) }
     val isValid = taskText.isNotBlank() && taskText.length <= 200
      // Auto-focus na TextField po otwarciu ekranu
     val focusRequester = remember { FocusRequester() }
     LaunchedEffect(Unit) { focusRequester.requestFocus() }
      Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Nowe zadanie") },
                navigationIcon = {
```

```
                    IconButton(onClick = onNavigateBack) {
                        Icon(Icons.AutoMirrored.Filled.ArrowBack, "Wstecz")
                    }
                }
            )
        }
    ) { innerPadding ->
        Column(
            modifier = Modifier.fillMaxSize()
              .padding(innerPadding)
                .padding(horizontal = 24.dp, vertical = 16.dp),
            verticalArrangement = Arrangement.spacedBy(16.dp)
        ) {
            OutlinedTextField(
                value = taskText,
                onValueChange = { if (it.length <= 200) taskText = it },
                label = { Text("Treść zadania") },
                placeholder = { Text("Np. Ukończyć Ćwiczenie 2...") },
                supportingText = {
                    Text("${taskText.length}/200",
                         color = if (taskText.length > 180)
                             MaterialTheme.colorScheme.error
                         else MaterialTheme.colorScheme.onSurfaceVariant)
                },
                isError = taskText.length > 200,
                singleLine = false,
                maxLines = 4,
                modifier = Modifier.fillMaxWidth().focusRequester(focusRequester)
            )
             // Wybór priorytetu — SegmentedButton (Material3)
            Text("Priorytet:", style = MaterialTheme.typography.labelMedium)
            SingleChoiceSegmentedButtonRow(modifier = Modifier.fillMaxWidth()) {
                Priority.entries.forEachIndexed { index, priority ->
                    SegmentedButton(
                        shape = SegmentedButtonDefaults.itemShape(index, Priority.entries.size),
                        onClick = { selectedPriority = priority },
                        selected = selectedPriority == priority,
                        label = { Text(priority.name) }
                    )
                }
            }
             Spacer(Modifier.weight(1f))
             // Przyciski
            Row(modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.spacedBy(8.dp)) {
                OutlinedButton(onClick = onNavigateBack, modifier = Modifier.weight(1f)) {
                    Text("Anuluj")
                }
                Button(
                    onClick = {
                        viewModel.addTask(taskText)
                        onTaskAdded()                    // nawigacja przez callback
                    },
                    enabled = isValid,
                    modifier = Modifier.weight(1f)
                ) {
                    Text("Dodaj zadanie")
                }
            }
        }
    }
}
```

5.6. Ekran szczegółów: TaskDetailScreen

TaskDetailScreen.kt

```
// ui/TaskDetailScreen.kt — szczegóły zadania z parametrem nawigacji
package pl.edu.prz.taskapp.ui
import androidx.compose.foundation.layout.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
```

```
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import androidx.lifecycle.viewmodel.compose.viewModel
import pl.edu.prz.taskapp.Priority
import pl.edu.prz.taskapp.TaskViewModel
  @OptIn(ExperimentalMaterial3Api::class) @Composable fun TaskDetailScreen(
     taskId: String,
            // otrzymany z nawigacji
     onNavigateBack: () -> Unit,
     viewModel: TaskViewModel = viewModel() ) {
     val uiState by viewModel.uiState.collectAsStateWithLifecycle()
      // Znajdź zadanie po ID (może być null jeśli usunięto)
     val task = uiState.tasks.find { it.id == taskId }
      // Jeśli zadanie nie istnieje – wróć automatycznie
     LaunchedEffect(task) {
         if (task == null) onNavigateBack()
     }
      Scaffold(
         topBar = {
             TopAppBar(
                 title = { Text("Szczegóły zadania") },
                 navigationIcon = {
                     IconButton(onClick = onNavigateBack) {
                         Icon(Icons.AutoMirrored.Filled.ArrowBack, "Wstecz")
                     }
                 }
             )
         }
     ) { innerPadding ->
         task?.let { t ->
             Column(
                 modifier = Modifier.fillMaxSize()
                     .padding(innerPadding)
                     .padding(24.dp),
                 verticalArrangement = Arrangement.spacedBy(16.dp)
             ) {
                 // Treść zadania
                 Card(modifier = Modifier.fillMaxWidth()) {
                     Column(modifier = Modifier.padding(16.dp)) {
                         Text("Treść", style = MaterialTheme.typography.labelSmall,
                              color = MaterialTheme.colorScheme.onSurfaceVariant)
                         Spacer(Modifier.height(8.dp))
                         Text(t.text, style = MaterialTheme.typography.bodyLarge)
                     }
                 }
                  // Status + Priorytet
                 Row(horizontalArrangement = Arrangement.spacedBy(8.dp)) {
                     AssistChip(
                         onClick = { viewModel.toggleTask(t.id) },
                         label = { Text(if (t.isDone) "Ukończone ✓" else "W trakcie") },
                         colors = AssistChipDefaults.assistChipColors(
                             containerColor = if (t.isDone)
                                 MaterialTheme.colorScheme.primaryContainer
                             else MaterialTheme.colorScheme.surfaceVariant
                         )
                     )
                     AssistChip(
                         onClick = { },
                         label = { Text("Priorytet: ${t.priority.name}") }
                     )
                 }
                  // Zmiana priorytetu
                 Text("Zmień priorytet:", style = MaterialTheme.typography.labelMedium)
                 Priority.entries.forEach { priority ->
                     OutlinedButton(
                         onClick = { viewModel.setPriority(t.id, priority) },
                         modifier = Modifier.fillMaxWidth()
                     ) {
                         Text(priority.name)
                     }
                 }
                  Spacer(Modifier.weight(1f))
                  // Usuń zadanie
                 Button(
                     onClick = { viewModel.deleteTask(t.id); onNavigateBack() },
                     modifier = Modifier.fillMaxWidth(),
                     colors = ButtonDefaults.buttonColors(
                         containerColor = MaterialTheme.colorScheme.errorContainer,
                         contentColor = MaterialTheme.colorScheme.onErrorContainer
```

```
                    )
                ) { Text("Usuń zadanie") }
            }
        }
    }
}
```

5.7. FilterRow: FilterChips

FilterRow.kt

```
// ui/components/FilterRow.kt
package pl.edu.prz.taskapp.ui.components
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import pl.edu.prz.taskapp.Filter
 @Composable fun FilterRow(
    currentFilter: Filter,
    onFilterChange: (Filter) -> Unit,
    modifier: Modifier = Modifier ) {
    LazyRow(
        modifier = modifier.fillMaxWidth(),
        contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp),
        horizontalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        items(Filter.entries.size) { index ->
            val filter = Filter.entries[index]
            val label = when (filter) {
                Filter.ALL    -> "Wszystkie"
                Filter.ACTIVE -> "W trakcie"
                Filter.DONE   -> "Ukończone"
            }
            FilterChip(
                selected = currentFilter == filter,
                onClick = { onFilterChange(filter) },
                label = { Text(label) }
            )
        }
    }
}
```

6. Zadania do wykonania

Poniższe zadania realizujesz w nowym projekcie TaskApp lub w projekcie HelloMobile z Ćwiczenia 1. Każde zadanie bazuje na poprzednim.

Zadanie 1. Konfiguracja projektu i nawigacja bazowa (20 pkt)

Wymagania

1.1. Utwórz projekt TaskApp lub dodaj zależności do HelloMobile zgodnie z sekcją 2 (libs.toml + build.gradle.kts). Pokaż prowadzącemu Build Output: BUILD SUCCESSFUL.

1.2. Zdefiniuj trasy nawigacji w Routes.kt: HomeRoute, AddTaskRoute, TaskDetailRoute(taskId). Użyj @Serializable.

1.3. Zbuduj AppNavigation.kt z NavHost zawierającym 3 destinations. Na razie ekrany mogą być tymczasowe: Text("Home"), Text("AddTask"), Text("Detail").

1.4. W MainActivity podłącz AppNavigation(). Uruchom na emulatorze i sprawdź czy aplikacja startuje bez crashy.

1.5. Ręcznie wywołaj nawigację: dodaj Button na "Home" który przechodzi do "AddTask". Sprawdź przycisk Wstecz.

Zadanie 2. ViewModel i lista zadań (30 pkt)

Wymagania

2.1. Zaimplementuj TaskViewModel.kt zgodnie z sekcją 4.1: Task, Filter, TaskListUiState, metody: addTask, toggleTask, deleteTask, setFilter.

2.2. Zaimplementuj HomeScreen.kt z LazyColumn wyświetlającym zadania. Każde zadanie jako TaskItem: tekst + Checkbox + przycisk Usuń.

2.3. Podłącz ViewModel do HomeScreen przez viewModel() i obserwuj uiState przez collectAsStateWithLifecycle(). Zmiany przez ViewModel muszą automatycznie aktualizować UI.

2.4. Zaimplementuj FilterRow z FilterChip dla: Wszystkie / W trakcie / Ukończone. Zmiana filtra musi natychmiast filtrować listę.

2.5. Pokaż prowadzącemu: dodaj kilka zadań przez init {} w ViewModel, zaznacz kilka jako done, przełącz filtry.

Zadanie 3. Ekran dodawania i szczegóły (30 pkt)

Wymagania

3.1 Zaimplementuj AddTaskScreen.kt z OutlinedTextField i przyciskiem Dodaj. Walidacja: pole nie może być puste, max 200 znaków. Przycisk Dodaj jest disabled gdy walidacja nie przechodzi.

3.2 Po kliknięciu Dodaj: wywołaj viewModel.addTask(text), wywołaj callback onTaskAdded() (nawigacja wstecz). Nowe zadanie musi pojawić się na liście.

3.3 Zaimplementuj TaskDetailScreen.kt — pokaż szczegóły zadania znalezionego po taskId. Dodaj przycisk toggle (ukończ/cofnij) i usuń.

3.4 Nawigacja: kliknięcie zadania na liście otwiera TaskDetailScreen z właściwym taskId. Wstecz wraca do listy.

3.5 Usuń dane przykładowe z init {} w ViewModel — lista powinna startować pusta.

Zadanie 4. Rozszerzenia (20 pkt)

Wymagania

4.1 Dodaj w AddTaskScreen wybór priorytetu (SegmentedButton lub RadioButton): LOW / NORMAL / HIGH. Priorytet jest przekazywany do viewModel.addTask().

4.2 Na liście zadań HIGH-priority oznacz wyróżnikiem: np. kolorowy pasek z lewej strony karty lub ikona '!' przy tekście.

4.3 Dodaj obsługę pustego stanu (empty state): gdy lista jest pusta, pokaż centralnie tekst z instrukcją i ikoną (ContentUnavailableView lub własny Composable).

4.4 Zaimplementuj Snackbar przy usuwaniu zadania z możliwością cofnięcia (Undo). Wskazówka: użyj SnackbarHostState w Scaffold i zapamiętaj ostatnio usunięte zadanie.

Wskazówka do Zadania 4.4: Snackbar z Undo

Snackbar Undo pattern

```
// Snackbar z Undo – wzorzec
// W ViewModel: przechowaj ostatnio usunięte zadanie
 private var lastDeletedTask: Task? = null  fun deleteTask(taskId: String) {
     val task = _uiState.value.tasks.find { it.id == taskId } ?: return
     lastDeletedTask = task                    // zapamiętaj
     _uiState.update { it.copy(tasks = it.tasks.filter { t -> t.id != taskId }) } }
  fun undoDelete() {
     val task = lastDeletedTask ?: return
     _uiState.update { it.copy(tasks = it.tasks + task) }
     lastDeletedTask = null }  // W HomeScreen – SnackbarHostState
 val snackbarHostState = remember { SnackbarHostState() }
 val scope = rememberCoroutineScope()

  Scaffold(snackbarHost = { SnackbarHost(snackbarHostState) }) { ... }
  // W TaskItem onDelete:
        onDelete = {
     viewModel.deleteTask(task.id)
     scope.launch {
         val result = snackbarHostState.showSnackbar(
             message = "Zadanie usunięte",
             actionLabel = "Cofnij",
             duration = SnackbarDuration.Short
         )
         if (result == SnackbarResult.ActionPerformed) {
             viewModel.undoDelete()
         }
     }
 }
```

7. Kryteria oceniania

Ćwiczenie oceniane jest w skali 0–100 punktów. Maksymalny czas: 90 minut.

Zadanie	Punkty	Uwagi
Zad. 1. Konfiguracja i nawigacja bazowa	**20**	Sync Gradle + działające przejścia między ekranami
Zad. 2. ViewModel i lista zadań	**30**	LazyColumn + filtry + toggle + delete
Zad. 3. AddTask i TaskDetail	**30**	Pełna nawigacja z parametrem taskId
Zad. 4. Priorytet + empty state + Undo	**20**	Kompletna implementacja wszystkich 4 podpunktów
SUMA	**100**	*Minimum do zaliczenia: 50 punktów*

Punkty	Ocena	Opis
90–100	**5.0**	Wszystkie zadania ukończone, kod czysty, MVVM poprawnie zaimplementowany
80–89	**4.5**	Zadania 1–4 ukończone z drobnymi brakami lub niepełnym Zadaniem 4
70–79	**4.0**	Zadania 1–3 w pełni, Zadanie 4 częściowo (min. 2 z 4 podpunktów)
60–69	**3.5**	Zadania 1–3 ukończone, Zadanie 4 pominięte

50–59	**3.0**	Zadania 1–2 + podstawowa nawigacja do AddTask (bez TaskDetail)
0–49	**2.0**	Niezaliczone. Wymagany termin poprawkowy.

8. Najczęstsze błędy i rozwiązania

	Brakuje pluginu w build.gradle.kts. Sprawdź też wpis w libs.versions.toml: plugins → kotlin-serialization. Sync Gradle po każdej zmianie.
	Sprawdź, czy @Serializable jest zaimportowane. Upewnij się, że Route klasa/object jest w tym samym pakiecie lub poprawnie importowana w NavHost.
	viewModel() tworzy ViewModel skojarzony z najbliższym ViewModelStoreOwner (zazwyczaj Activity). Jeśli ViewModel jest tworzony osobno w kilku ekranach, to różne instancje! Rozwiązanie: przekaż ViewModel jako parametr lub użyj activityViewModel().
	Sprawdź, czy nie używasz mutableListOf() zamiast MutableStateFlow. Mutable collections nie powiadamiają o zmianach! Zawsze używaj .update { } lub copy() z niezmienną listą.
	Sprawdź, czy navController.navigateUp() jest poprawnie podłączony. Jeśli używasz BackHandler sprawdź czy enabled=true. Domyślne zachowanie Wstecz jest zarządzane przez NavController automatycznie.
	wymaga żeby ID było unikalne i stabilne. Jeśli używasz indeksu jako key, możesz mieć crashe przy usuwaniu elementów. Używaj UUID jako klucz zadania.

9. Porównanie: stan lokalny vs ViewModel

Ważne pytanie: kiedy używać remember { mutableStateOf() } (stan lokalny w Composable), a kiedy ViewModel? Odpowiedź zależy od zasięgu stanu.

Kryterium	`remember { mutableStateOf() }`	ViewModel + StateFlow
Przeżywa rotację	❌ NIE — traci stan przy obróceniu	☑ TAK — przeżywa rotację ekranu
Zasięg	Tylko w danym Composable + dzieci	Cały ekran (Activity / Fragment)
Kiedy używać	UI state: pole tekstowe, rozwinięty panel	Business state: lista danych, błędy, załadowanie
Przykłady	var expanded, var text, var isVisible	lista tasków, user, zalogowany, error message
Testowalność	Trudna — zintegrowana z UI	Łatwa — ViewModel to czysta klasa Kotlin
Lifecycle	Żyje z Composable	Żyje z ViewModelStoreOwner (Activity)

10. Sprawozdanie i repozytorium

Sprawozdanie nie jest wymagane. Demonstracja kodu prowadzącemu zastępuje pisemne sprawozdanie.

~~**Repozytorium GitHub (wymagane):** *Link do repo na Moodle do końca tygodnia w którym odbywa się ćwiczenie.*~~

#	Akcja	Checklist przed wysłaniem repozytorium
1	Kod kompiluje się	Build→Make Project→0 errors. Aplikacja uruchamia się na emulatorze bez crasha.
2	Struktura plików	Sprawdź czy wszystkie pliki są w repozytorium: Routes.kt, AppNavigation.kt, TaskViewModel.kt, HomeScreen.kt, AddTaskScreen.kt, TaskDetailScreen.kt
3	Commit message	Conventional Commits: feat: add navigation and ViewModel; opisz co zrobiłeś. Unikaj: 'fix', 'update', 'changes' bez opisu.
4	README.md	Dodaj krótki README.md: imię/nazwisko, opis projektu, zrzut ekranu (możesz dodać .png z emulatora).
5	Brak .apk / build/	.gitignore powinien wykluczać: build/, .gradle/, *.apk, .idea/workspace.xml. Sprawdź: git status nie pokazuje tych plików.
6	Link na Moodle	Wklej URL repozytorium (np. https://github.com/twoje-konto/taskapp-lab2) w odpowiednie pole na Moodle.

Instrukcja przygotowana dla przedmiotu Programowanie Aplikacji Mobilnych.
Wersja 1.5 • Rok akademicki 2025/2026 • Kontynuacja Ćwiczenia 1 (Android Studio + Compose).
Następne ćwiczenie: Ćwiczenie 3 — Room Database, persystencja danych i Coroutines IO.