Сучасні технології мобільного програмування

Архітектура Jetpack Compose застосунку

Слайди до лекцій (3 змістовий модуль)

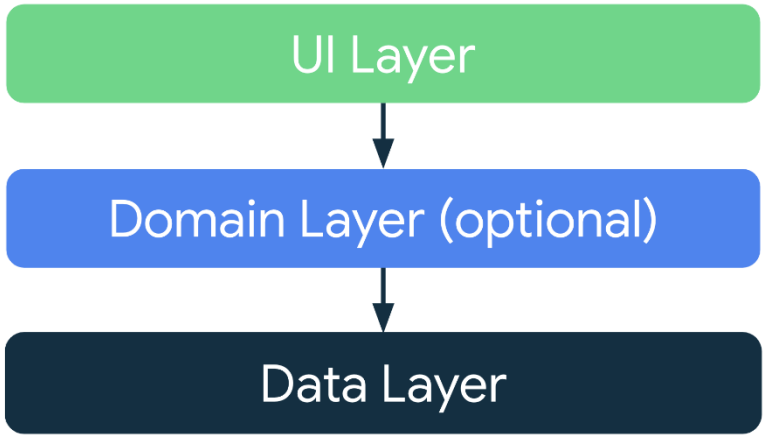
Архітектура програми

- **Архітектура програми** передбачає розподіл обов'язків між частинами програми. Добре продумана архітектура програми допомагає масштабувати програму, та розширювати її додатковими функціями. Архітектура також може спростити співпрацю в команді.
- Архітектура програми реалізується розподілом функцій програми, реалізованих програмними частинами, за різними шарами, які часто виділяють як окремі пакети.

Архітектура програми - шари

Рекомендована архітектура програми передбачає щонайменше два шари:

- **Шар інтерфейсу користувача (UI Layer)**: шар, який відображає дані програми на екрані, але є незалежним від даних.
- **Шар даних (Data Layer)**: шар, який зберігає, отримує та надає доступ до даних програми.
- Також можна додати ще один рівень, який називається **шаром домену (Domain Layer)**, щоб спростити та повторно використовувати взаємодію між рівнями інтерфейсу користувача та даних.

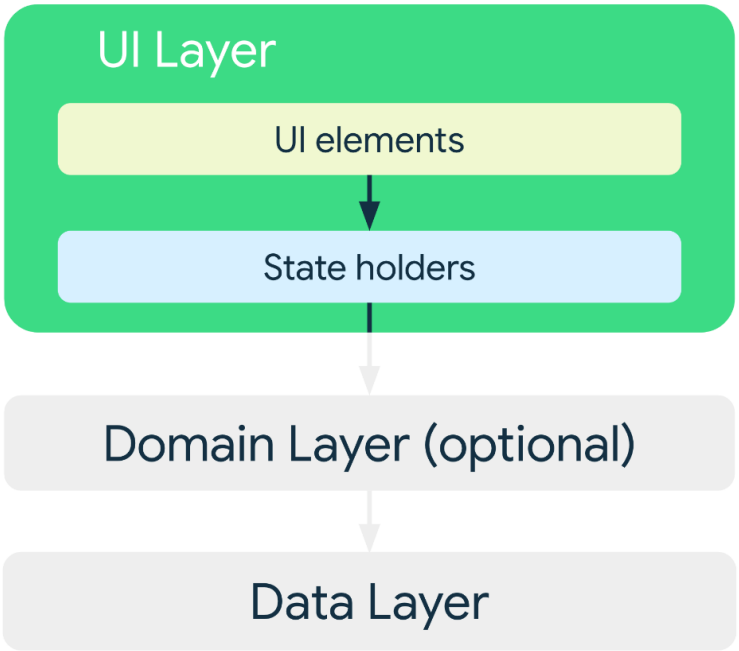


Стрілки визначають залежності між класами. Наприклад, рівень домену залежить від класів рівня даних.

Архітектура програми - UI Layer

Шар інтерфейсу користувача складається з таких компонентів:

- **Елементи інтерфейсу користувача (UI Elements)**: компоненти, які відображають дані на екрані (створюються за допомогою Jetpack Compose).
- **Власники станів (State Holders):** компоненти, які зберігають дані, надають їх інтерфейсу користувача та обробляють **логіку інтерфейсу користувача** (**UI logic**).

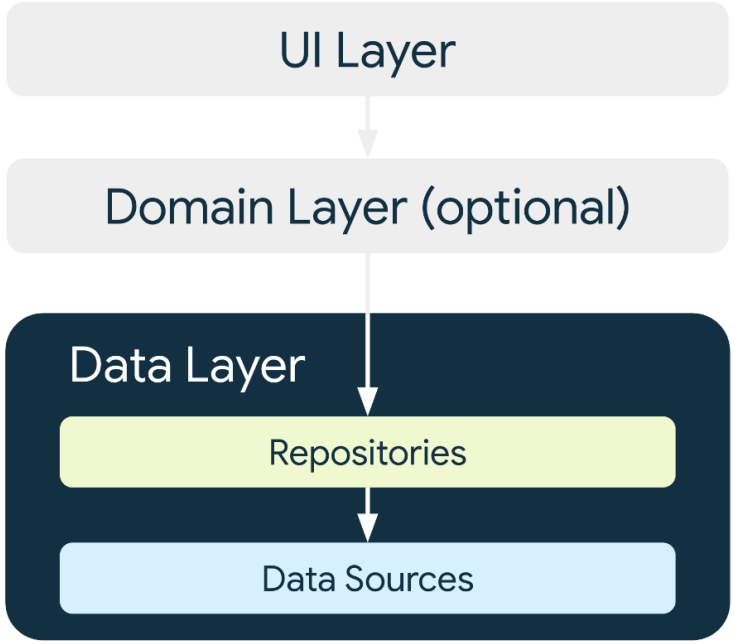


Приклади UI-логіки:

- отримання тексту, який повинен з'явитися на екрані, з використанням рядкових ресурсів,
- виведення повідомлень за допомогою Toast/Snackbar,
- навігація до нового екрану після кліку на кнопці
- тощо.

Архітектура програми - Data Layer

- **Шар даних** містить **дані програми (application data)** та **бізнес-логіку (Buisiness logic)**, які визначають, як дані мають створюватися, змінюватися, зберігатися та видалятися. (фактично це реалізація вимог до програми).
- **Шар даних** складається з **репозиторіїв (Repositories)**, кожен з яких може містити від нуля до багатьох **джерел даних (Data Source)**. Необхідно створювати репозиторії для кожного типу даних, які обробляться програмою.

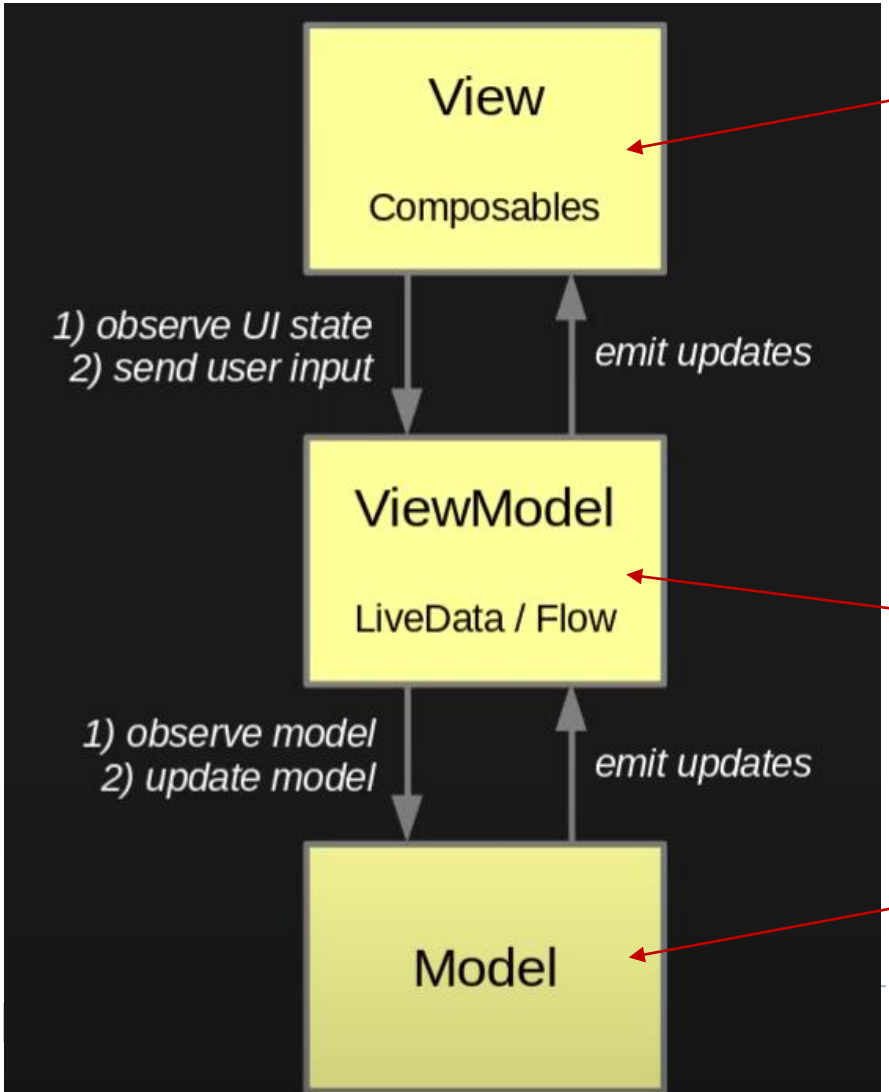


Приклади бізнес-логіки:

- додання/редагування/видалення питання тесту до банку питань,
- підрахунок правильних відповідей з розрахунком загальної оцінки за тест,
- тощо.

Архітектура MVVM

- **MVVM - Model View ViewModel** - архітектурний шаблон

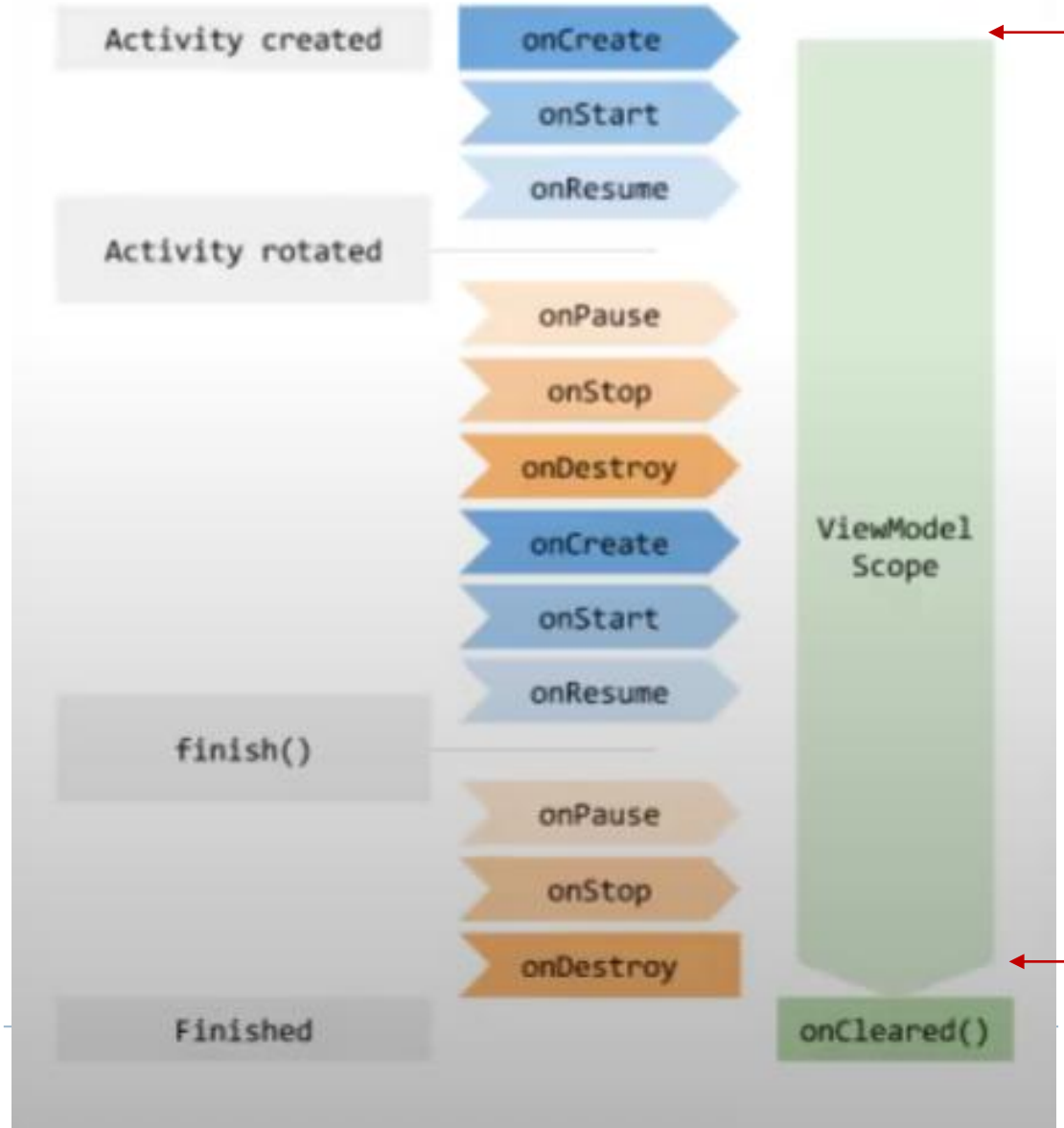


Відображає екран на основі даних від ViewModel

Отримує дані від Model і перетворює їх до вигляду, зручного для використання у View, переживає зміни конфігурації, має спрощений життєвий цикл (див. наст. слайд)

Не регламентується патерном. Часто - репозиторій з даними

Життєвий цикл активності та ViewModel

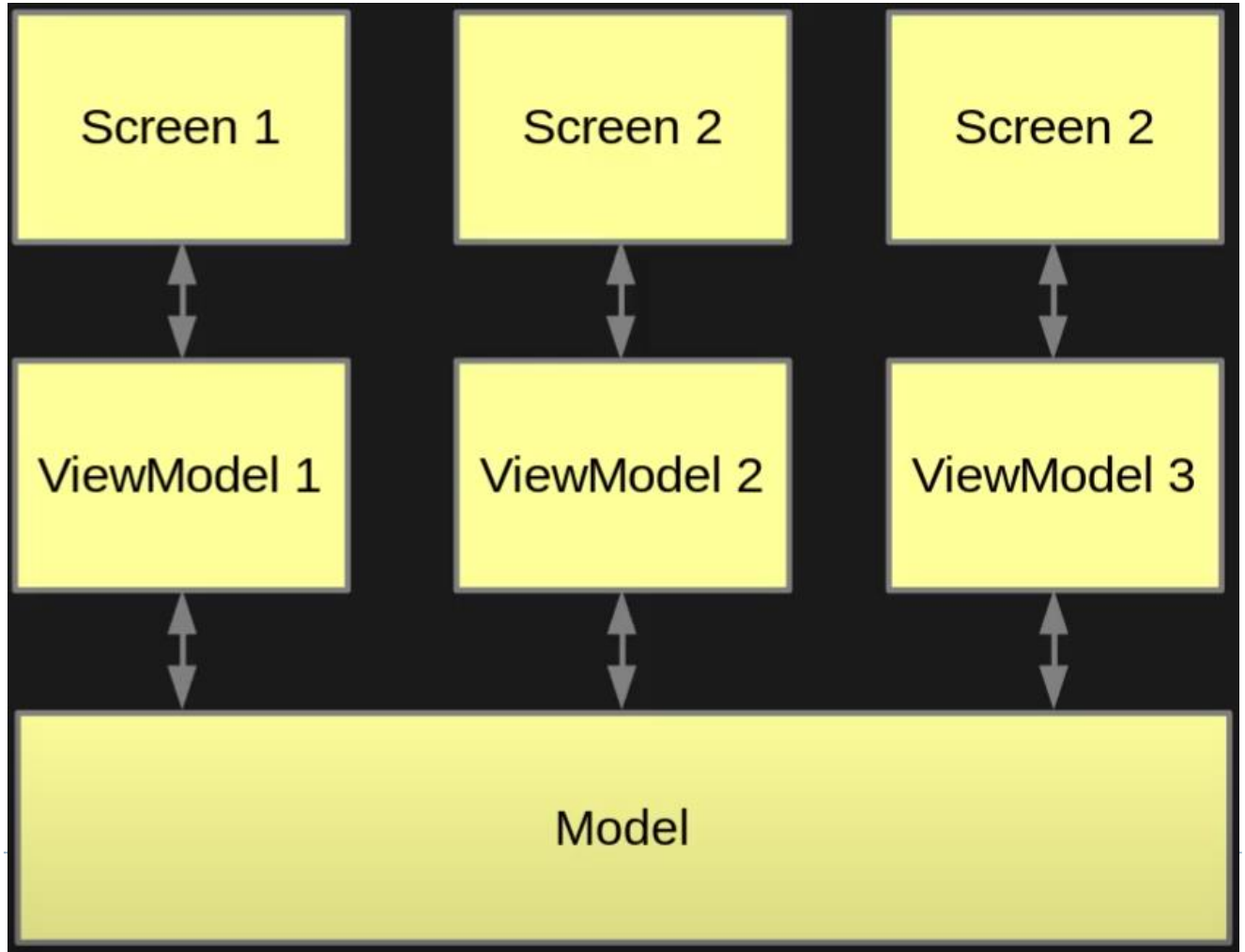


ViewModel created

ViewModel destroyed

ViewModel

- Зазвичай кожна ViewModel асоціюється з одним екраном



Add dependency

androidx.lifecycle.lifecycle-viewmodel-compose

Android

- app
  - manifests
  - ua.edu.znu.geo
- Gradle Scripts
  - build.gradle.kts (P
  - build.gradle.kts (M
  - proguard-rules.pr
  - gradle.properties
  - gradle-wrapper.pr
  - libs.versions.toml
  - local.properties (S
  - settings.gradle.kts

MainScreen.kt | Question.kt | LogLifecycle.kt | MainActivity.kt | build.gradle.kts (:app) | libs.versions.toml

```
12  [libraries]
13  androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
17  androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version.ref
20  androidx-ui = { group = "androidx.compose.ui", name = "ui" }
21  androidx-ui-graphics = { group = "androidx.compose.ui", name = "ui-graphics" }
22  androidx-ui-tooling = { group = "androidx.compose.ui", name = "ui-tooling" }
23  androidx-ui-tooling-preview = { group = "androidx.compose.ui", name = "ui-tooling-preview" }
24  androidx-ui-test-manifest = { group = "androidx.compose.ui", name = "ui-test-manifest" }
25  androidx-ui-test-junit4 = { group = "androidx.compose.ui", name = "ui-test-junit4" }
26  androidx-material3 = { group = "androidx.compose.material3", name = "material3" }
27  androidx-lifecycle-viewmodel-compose = { group = "androidx.lifecycle", name = "lifecycle-viewmodel-compose",
28                                          version.ref = "lifecycleRuntimeKtx" }
29  [plugins]
```

Android

- app
  - manifests
  - kotlin+java
    - ua.edu.znu.geoquizcompose
    - ua.edu.znu.geoquizcompose
    - ua.edu.znu.geoquizcompose
  - res
  - res (generated)
- Gradle Scripts
  - build.gradle.kts (Project: GeoQui
  - build.gradle.kts (Module :app)
  - proguard-rules.pro (ProGuard Ru

MainScreen.kt | Question.kt | LogLifecycle.kt | MainActivity.kt | build.gradle.kts (:app)

You can use the Project Structure dialog to view and edit your project configuration Open (Ctrl+Alt+Shift+

```
43  dependencies {
44
45      implementation(libs.androidx.core.ktx)
46      implementation(libs.androidx.lifecycle.runtime.ktx)
47      implementation(libs.androidx.activity.compose)
48      implementation(platform( dependencyProvider = libs.androidx.compose.bom))
49      implementation(libs.androidx.ui)
50      implementation(libs.androidx.ui.graphics)
51      implementation(libs.androidx.ui.tooling.preview)
52      implementation(libs.androidx.material3)
53      implementation(libs.androidx.lifecycle.viewmodel.compose)
```

MainViewModel

```
package ua.edu.znu.geoquizcomposeedu.viewmodel
import ...
private const val TAG = "MainViewModel"
class MainViewModel : ViewModel() {
  private val _mainScreenState = MutableStateFlow(MainScreenState())
  val mainScreenState: StateFlow<MainScreenState> = _mainScreenState.asStateFlow()
  private val questionBank = listOf(  ...    )
  var currentIndex = _mainScreenState.value.currentIndex
//   fun getCurrentQuestionId() =
                        questionBank[_mainScreenState.value.currentIndex].textResId
  fun getQuestionIdByIndex(index: Int): Int {
    return questionBank[index].textResId
  }
...
```

успадковує androidx.lifecycle.ViewModel

змінюваний стан для оновлення у ViewModel

незмінюваний стан, що надається Composable

Працює некоректно, оскільки стан ViewModel ще не оновлюється, коли у Composable відбувається рекомпозиція (це відбувається внаслідок асинхронної природи оновлення стану у Compose.

Необхідно передавати поточний індекс з Composable, оскільки там він оновився, а у ViewModel - ні

Це шаблон інкапсуляції: єдине джерело достовірної інформації, запобігає зовнішнім мутаціям, безпечніший односторонній потік даних.

MainViewModel

```
...
  fun onAnswerButtonClick(isTrue: Boolean): Int {
    val currentQuestion = questionBank[currentIndex]
    val messageId = if (isTrue == currentQuestion.answer) {
      R.string.correct_toast
    } else {
      R.string.incorrect_toast
    }
    return messageId
  }

  fun onNextQuestionButtonClick() {
    _mainScreenState.value = _mainScreenState.value.copy(
      currentIndex = (currentIndex + 1) % questionBank.size
    )
    currentIndex = _mainScreenState.value.currentIndex
    Log.d(TAG, "MainViewModel.onNextQuestionButtonClick currentIndex: $currentIndex")
  }

  fun isLastQuestion(): Boolean {
    return currentIndex == questionBank.size - 1
  }
```

Створення нового стану

Оновлення локальної змінної

Працює правильно, оскільки це чиста функція, яка завжди відображає поточний стан.

MainScreen

```
package ua.edu.znu.geoquizcomposeedu.ui.screens
import ...

data class MainScreenState(
   val currentIndex: Int = 0
)
@Composable
fun MainScreen(
   innerPadding: PaddingValues,
) {
   val mainViewModel: MainViewModel = viewModel()
   val mainScreenState by mainViewModel.mainScreenState.collectAsStateWithLifecycle()
   val context = LocalContext.current
...
```

Видалені анотація @Parcelize та успадкування від Parcelable, оскільки тепер стан створюється та зберігається у ViewModel

Створює або використовує існуючу ViewModel з областю видимості - Activity, що викликає цю Composable (або NaBackStackEntry при використанні Jetpack Navigation)

Збирає значення з Flow та конвертує їх у Compose State. Це виконується на етапі життєвого циклу активності, що запускає Composable, між onStart() та onStop().

MainScreen

```
...
  Column(
    modifier = Modifier
      .fillMaxSize()
      .padding(innerPadding),
    verticalArrangement = Arrangement.Center,
    horizontalAlignment = Alignment.CenterHorizontally
  ) {
    Text(
//        text = stringResource(id = mainViewModel.getCurrentQuestionId()),
      text = stringResource(id =
                    mainViewModel.getQuestionIdByIndex(mainScreenState.currentIndex)),
      textAlign = TextAlign.Center,
      modifier = Modifier
        .padding(innerPadding)
        .padding(16.dp)
    )
...
```

Працює некоректно, оскільки стан ViewModel ще не оновлюється, коли у Composable відбувається рекомпозиція (це відбувається внаслідок асинхронної природи оновлення стану у Compose.

Необхідно передавати поточний індекс з Composable, оскільки тут він оновився, а у ViewModel - ні

MainScreen

…

```
Row(
    modifier = Modifier.size(height = 90.dp, width = 200.dp),
    horizontalArrangement = Arrangement.SpaceEvenly,
    verticalAlignment = Alignment.CenterVertically
) {
    Button(
        onClick = {
            val messageId = mainViewModel.onAnswerButtonClick(true)
            Toast.makeText(context, messageId, Toast.LENGTH_SHORT).show()
        }
    ) {
        Text(stringResource(id = R.string.true_button))
    }
    Spacer(modifier = Modifier.width(16.dp))
    Button(
        onClick = {
            val messageId = mainViewModel.onAnswerButtonClick(false)
            Toast.makeText(context, messageId, Toast.LENGTH_SHORT).show()          }) {
        Text(stringResource(id = R.string.false_button))
    }      } …
```

MainScreen

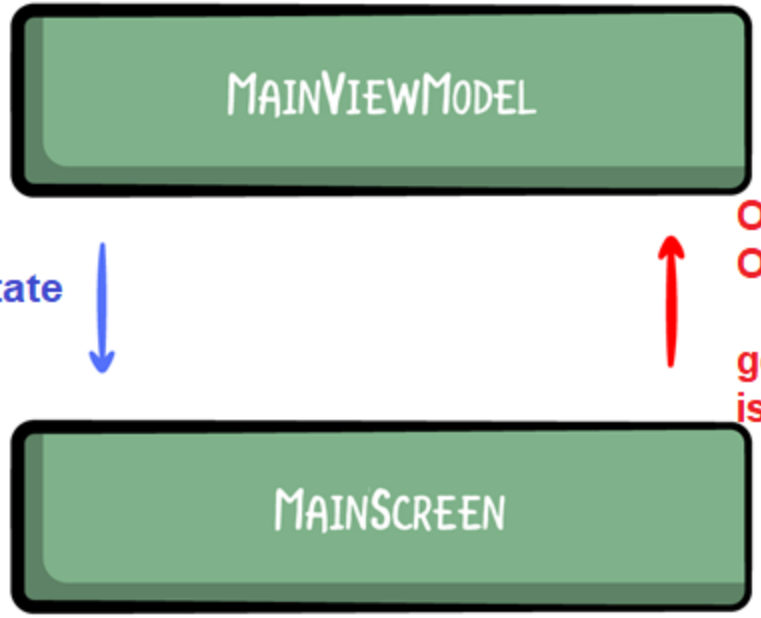
```
...
    Spacer(modifier = Modifier.width(16.dp))
    Button(onClick = {
        mainViewModel.onNextQuestionButtonClick()
    }) {
        Text(stringResource(id = R.string.next_button))
    }
    Box(
        modifier = Modifier.height(100.dp)
    ) {
        if (mainViewModel.isLastQuestion()) {
            logCompositionLifecycle("LastQuestionText")
            Text(
                text = "This is the last question"
            )
        }
    }
  }
}
```

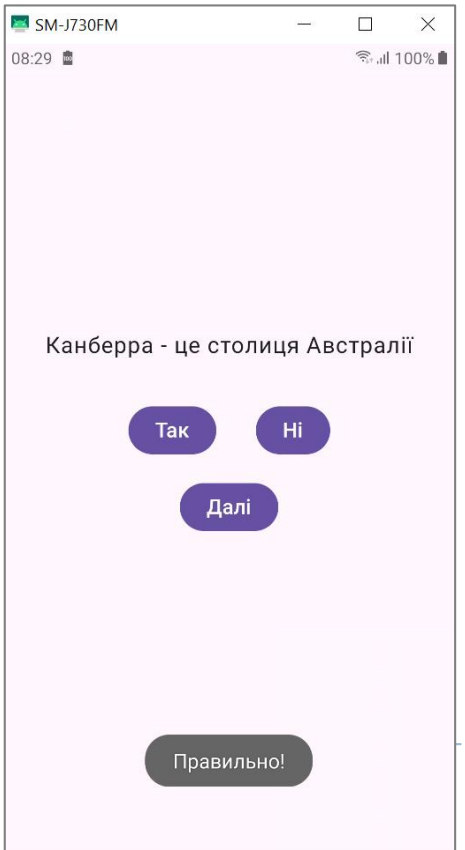
```
@Preview(showBackground = true)
@Composable
fun MainScreenPreview() {
    val innerPadding = PaddingValues(16.dp)
    MainScreen(innerPadding)
}
```

MainActivity

```
package ua.edu.znu.geoquizcomposeedu
import ...
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            GeoQuizComposeEduTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    MainScreen(
                        innerPadding = innerPadding
                    )
                }
            }
        }
    }
}
```

Обробка стану з односпрямованим потоком даних - MainViewModel & MainScreen





ViewModel & State Hoisting

- **Підняття стану (State Hoisting)** – це переміщення стану Composable компонента UI до його викликаючої сторони. Це робить Composable компонент UI **"без стану" (stateless)**.
- Composable компонент UI отримує стан як параметр та надає події (наприклад, через лямбда-функції) для свого викликаючого об'єкта для обробки змін стану.
- ViewModel діє як власник стану, надаючи стан інтерфейсу компонованим об'єктам та обробляючи логіку та події, пов'язані з інтерфейсом користувача.
- Викликаючою стороною може бути, наприклад, активність, у якій створюється об'єкт ViewModel та отримується з ViewModel стан для передачі його як параметру Composable компоненту UI.
- Composable компонент UI "без стану" легше тестується та легше повторно використовується у різних контекстах.

ViewModel & State Hoisting - MainScreen

- MainViewModel залишилася без змін.
- З MainScreen Composable видалений стан - stateless Composable.

```
package ua.edu.znu.geoquizcomposeedu.ui.screens

import ...

data class MainScreenState(
    val currentIndex: Int = 0
)
@Composable
fun MainScreen(
    mainScreenState: MainScreenState,
    getQuestionText: @Composable (Int) -> String,
    onAnswerButtonClick: (Boolean) -> Int,
    onNextQuestionButtonClick: () -> Unit,
    innerPadding: PaddingValues,
) {
    val context = LocalContext.current
```

Тепер стан приймається як параметр, що буде передаватися викликаючою стороною

Callback функції, будуть задані викликаючою стороною

...

ViewModel & State Hoisting - MainScreen

…

```
Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(innerPadding),
    verticalArrangement = Arrangement.Center,
    horizontalAlignment = Alignment.CenterHorizontally
) {
    Text(
        text = getQuestionText(mainScreenState.currentIndex),
        textAlign = TextAlign.Center,
        modifier = Modifier
            .padding(innerPadding)
            .padding(16.dp)
    )
    Row(
        modifier = Modifier.size(height = 90.dp, width = 200.dp),
        horizontalArrangement = Arrangement.SpaceEvenly,
        verticalAlignment = Alignment.CenterVertically
    ) { …
```

Отримання тексту питання за індексом зі стану через callback-функцію

ViewModel & State Hoisting - MainScreen

…

```
    Button(
        onClick = {
            val messageId = onAnswerButtonClick(true)
            Toast.makeText(context, messageId, Toast.LENGTH_SHORT).show()
        }
    ) {  Text(stringResource(id = R.string.true_button))  }
    Spacer(modifier = Modifier.width(16.dp))
    Button(
        onClick = {
            val messageId = onAnswerButtonClick(false)
            Toast.makeText(context, messageId, Toast.LENGTH_SHORT).show()
        }) {Text(stringResource(id = R.string.false_button))  }
  }
  Spacer(modifier = Modifier.width(16.dp))
  Button(onClick = {
    onNextQuestionButtonClick()
  }) {
    Text(stringResource(id = R.string.next_button))
  }
}
}
```

Отримання ідентифікатора відповіді через callback-функцію

Виклик callback-функції

ViewModel & State Hoisting - MainActivity

ВИЛИКАЮЧА СТОРОНА

```
package ua.edu.znu.geoquizcomposeedu

import ...

class MainActivity : ComponentActivity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContent {
      GeoQuizComposeEduTheme {
        Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
          val mainViewModel: MainViewModel = viewModel()
          val mainScreenState by
                        mainViewModel.mainScreenState.collectAsStateWithLifecycle()
          MainScreen(
            mainScreenState = mainScreenState,
            getQuestionText = { index ->
              stringResource(mainViewModel.getQuestionByIndex(index))
            },
            onAnswerButtonClick = { isTrue ->
              mainViewModel.onAnswerButtonClick(isTrue)
            }, ...
```

створення стану, що зберігається у ViewModel та підписка на його зміни

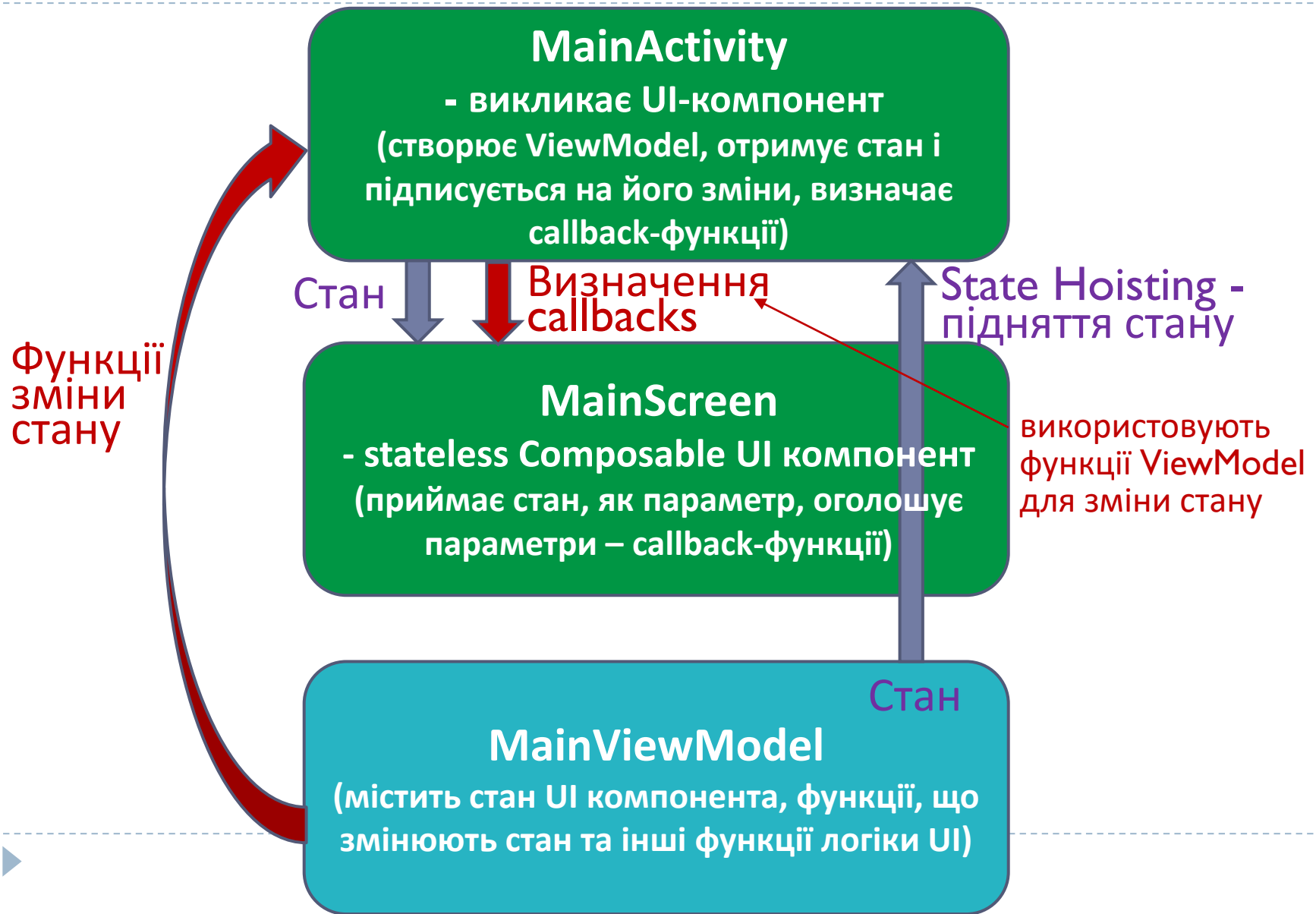
передача стану і визначень callback-функцій до Composable

ViewModel & State Hoisting - MainActivity

```
...
                onNextQuestionButtonClick = {
                    mainViewModel.onNextQuestionButtonClick()
                },
                innerPadding = innerPadding
            )
        }
    }
  }
}
```

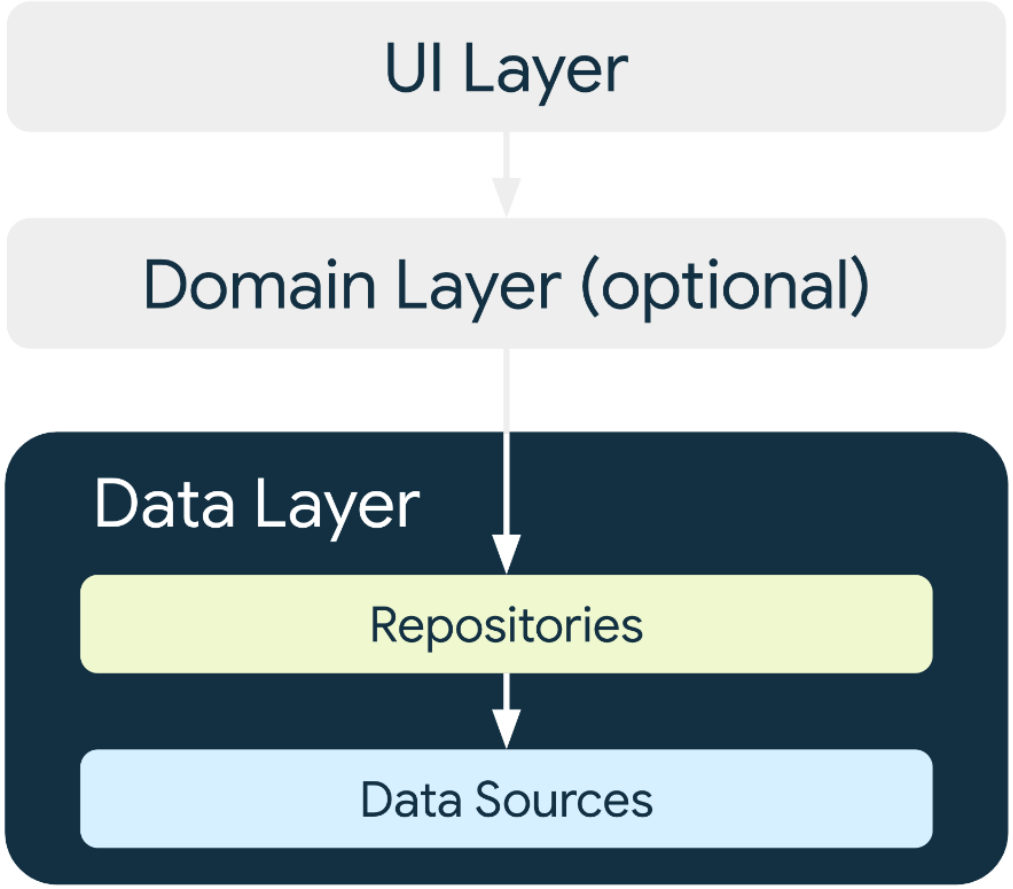
передача визначенянь callback-функції до Composable

ViewModel & State Hoisting



Додання Data Layer

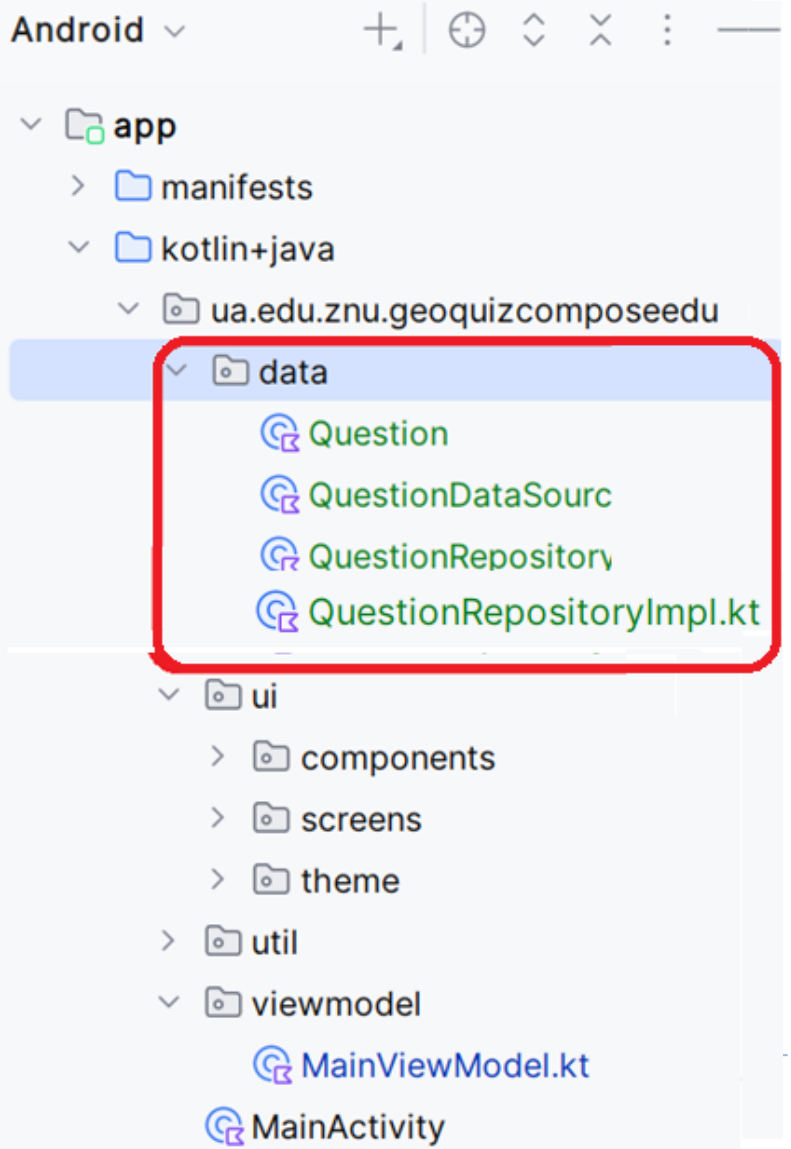
- Рекомендована архітектура рівня даних у додатку Jetpack Compose, згідно з офіційними рекомендаціями Android, зосереджена навколо **репозиторіїв** та **джерел даних**.

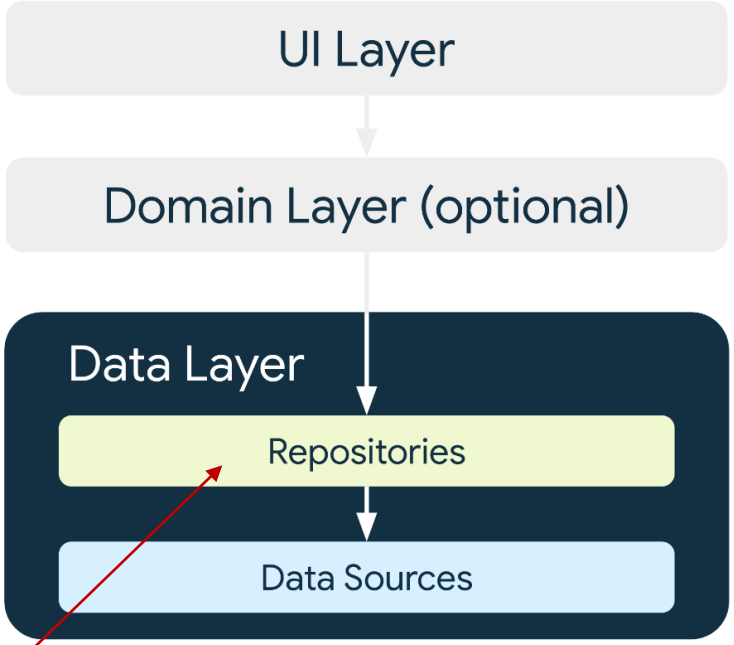


Додання Data Layer

- **Репозиторії -** виступають єдиним джерелом достовірної інформації для певних типів даних. Вони надають дані рівню інтерфейсу користувача (та додатковому рівню домену), а також абстрагують базові джерела даних. Класи репозиторію створюють для кожного окремого типу даних. Наприклад, UserRepository, ProductRepository, OrderRepository. Репозиторії містять бізнес-логіку, пов'язану з отриманням та маніпулюванням даними. Рекомендується організовувати репозиторій як інтерфейс з абстрактними методами та його реалізацію.
- **Джерела даних** - обробляють специфіку зберігання та отримання даних з різних джерел, наприклад, локальної БД, результатів запитів до API тощо. Репозиторій може використовувати одне або кілька джерел даних. Повинна існувати можливість зміни джерела даних без змін у репозиторії.

Додання Data Layer





Репозиторій реалізований як інтерфейс з абстрактними методами та клас-реалізація

Додання Data Layer - рефакторинг data-класу

```
package ua.edu.znu.geoquizcomposeedu.data

import java.util.UUID

data class Question(
    val id: String = UUID.randomUUID().toString(),
    val questionText: String,
    val answer: Boolean
)
```

Доданий ідентифікатор з унікальним випадковим значенням, який дозволяє шукати конкретний екземпляр при його оновленні.

Ідентифікатор рядкового ресурсу замінений на рядок для можливості додання нових та редагування старих питань

Додання Data Layer - DataSource

```
package ua.edu.znu.geoquizcomposeedu.data

class QuestionDataSource {
   private val questions = mutableListOf(
      Question(questionText="Канберра - це столиця Австралії", answer = true),
      Question(questionText="Тихий океан більший, аніж Атлантичний океан", answer =
true),
      Question(questionText="Суецький канал поєднує Червоне море та Індійський океан",
answer = false),
      Question(questionText="Витік річки Ніл знаходиться в Єгипті", answer = false),
      Question(questionText="Амазонка є найдовшою річкою у Америці", answer = true),
      Question(questionText="Озеро Байкал є найстарішим та найглибшим прісноводним
озером", answer = true),
   )

   fun getQuestions(): List<Question> = questions
}
```

Тепер список мутабельний для можливості додання нових та редагуванняі видалення старих питань

функція отримання даних

QuestionDataSource - постачальник початкового списку питань. Усе керування станом та оновлення знаходяться в QuestionRepository, що забезпечує єдине джерело достовірної інформації (a single source of truth) для інтерфейсу користувача.

Додання Data Layer - інтерфейс Repository

```
package ua.edu.znu.geoquizcomposeedu.data

interface QuestionRepository {
    fun getQuestionByIndex(index: Int): Question
    fun getQuestionBankSize(): Int
}
```

Функції, що використовуватимуться інтерфейсом користувача (екземплярами ViewModel)

Додання Data Layer - реалізація RepositoryImpl

```
package ua.edu.znu.geoquizcomposeedu.data

import ...

class QuestionRepositoryImpl (val questionDataSource:
                                        QuestionDataSource):QuestionRepository {

    private fun getQuestions(): List<Question> {
        return questionDataSource.getQuestions()
    }

    override fun getQuestionByIndex(index: Int): Question {
        return getQuestions()[index]
    }

    override fun getQuestionBankSize() = getQuestions().size
}
```

Приватна функція для зменшення коду у інших функціях

Функції, що використовуватимуться інтерфейсом користувача (екземплярами ViewModel)

Додання Data Layer - MainViewModel refactoring

```
package ua.edu.znu.geoquizcomposeedu.viewmodel
import ...
private const val TAG = "MainViewModel"
class MainViewModel(val questionRepository: QuestionRepository) : ViewModel() {
   private val _mainScreenState = MutableStateFlow(MainScreenState())
   val mainScreenState: StateFlow<MainScreenState> = _mainScreenState.asStateFlow()
//    private val questionBank = listOf(
//        Question(textResId = R.string.question_australia, answer = true),
//        Question(textResId = R.string.question_oceans, answer = true),
//        Question(textResId = R.string.question_mideast, answer = false),
//        Question(textResId = R.string.question_africa, answer = false),
//        Question(textResId = R.string.question_americas, answer = true),
//        Question(textResId = R.string.question_asia, answer = true)
//    )
//    private val questionBank: List<Question>
                                = questionRepository.getQuestionBank()

   var currentIndex = _mainScreenState.value.currentIndex
...
```

Репозиторій передається як параметр, як тип зазначається інтерфейс

Видалено

Додання Data Layer - MainViewModel refactoring

```
...
fun getQuestionIdByIndex(index: Int): Question {
    return questionRepository.getQuestionByIndex(index)
}

fun onAnswerButtonClick(isTrue: Boolean): Int {
    val currentQuestion = getQuestionByIndex(currentIndex)
    val messageId = if (isTrue == currentQuestion.answer) {
        R.string.correct_toast
    } else {
        R.string.incorrect_toast
    }
    return messageId
}

fun onNextQuestionButtonClick() {
    _mainScreenState.value = _mainScreenState.value.copy(
        currentIndex = (currentIndex + 1) % questionRepository.getQuestionBankSize()
    )

    currentIndex = _mainScreenState.value.currentIndex
    Log.d(TAG, "onNextQuestionButtonClick: $currentIndex")
} }
```

Додання Data Layer - MainScreen refactoring

```
package ua.edu.znu.geoquizcomposeedu.ui.screens

import ...


data class MainScreenState(
    val currentIndex: Int = 0
)

@Composable
fun MainScreen(
    mainScreenState: MainScreenState,
    getQuestionText: @Composable (Int) -> String,
    onAnswerButtonClick: (Boolean) -> Int,
    onNextQuestionButtonClick: () -> Unit,
    innerPadding: PaddingValues,
) {
    val context = LocalContext.current

...
```

Тепер стан зберігається у ViewModel, а не в Bundle, тому серіалізація або парселізація не потрібні

Тепер стан приймається як параметр, що буде передаватися викликаючою стороною

Callback функції, будуть задані викликаючою стороною

Додання Data Layer - MainScreen refactoring

```
...
  Column(
    modifier = Modifier
      .fillMaxSize()
      .padding(innerPadding),
    verticalArrangement = Arrangement.Center,
    horizontalAlignment = Alignment.CenterHorizontally
  ) {
    Text(
      text = getQuestionText(mainScreenState.currentIndex),
      textAlign = TextAlign.Center,
      modifier = Modifier
        .padding(innerPadding)
        .padding(16.dp)
    )
    Row(
      modifier = Modifier.size(height = 90.dp, width = 200.dp),
      horizontalArrangement = Arrangement.SpaceEvenly,
      verticalAlignment = Alignment.CenterVertically
    ) { ...
```

Отримання тексту питання за індексом зі стану через callback-функцію

… Додання Data Layer - MainScreen refactoring

Отримання ідентифікатора відповіді через callback-функцію

```
Button(
    onClick = {
        val messageId = onAnswerButtonClick(true)
        Toast.makeText(
            context,
            messageId,
            Toast.LENGTH_SHORT
        ).show()
    }
) { Text(stringResource(id = R.string.true_button)) }
Spacer(modifier = Modifier.width(16.dp))
Button(
    onClick = {
        val messageId = onAnswerButtonClick(false)
        Toast.makeText(
            context,
            messageId,
            Toast.LENGTH_SHORT
        ).show()
    }) { Text(stringResource(id = R.string.false_button))}
} …
```

… Додання Data Layer - MainScreen refactoring

```
        Spacer(modifier = Modifier.width(16.dp))
        Button(onClick = {
            onNextQuestionButtonClick()
        }) {
            Text(stringResource(id = R.string.next_button))
        }
    }
}

@Preview(showBackground = true)
@Composable
fun MainScreenPreview() {
    val previewState = MainScreenState(
        currentIndex = 0,
    )
    val innerPadding = PaddingValues(16.dp)
    MainScreen(
        mainScreenState = previewState,
        getQuestionText = { index -> "Канбера - столиця Австралії" },
        onAnswerButtonClick = { isTrue -> R.string.correct_toast },
        onNextQuestionButtonClick = { },
        innerPadding = innerPadding    )}
```

Виклик callback-функції

Додання Data Layer - MainActivity refactoring

ВИЛИКАЮЧА СТОРОНА

Функція viewModel() за замовчуванням працює тільки для ViewModels без параметрів. Для ViewModel з параметрами використовується ViewModelProvider.Factory

Створення репозиторію

```
package ua.edu.znu.geoquizcomposeedu

import ...

class MainActivity : ComponentActivity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContent {
      GeoQuizComposeEduTheme {
        Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
          val questionRepository: QuestionRepository = remember {
            QuestionRepositoryImpl(QuestionDataSource())
          }
          val mainViewModel: MainViewModel = viewModel(
            factory = object : ViewModelProvider.Factory {
              override fun <T : ViewModel> create(modelClass: Class<T>): T {
                if (modelClass.isAssignableFrom(MainViewModel::class.java)) {
                  @Suppress("UNCHECKED_CAST")
                  return MainViewModel(questionRepository) as T
                }
                throw IllegalArgumentException("Unknown ViewModel class") } } )  ...
```

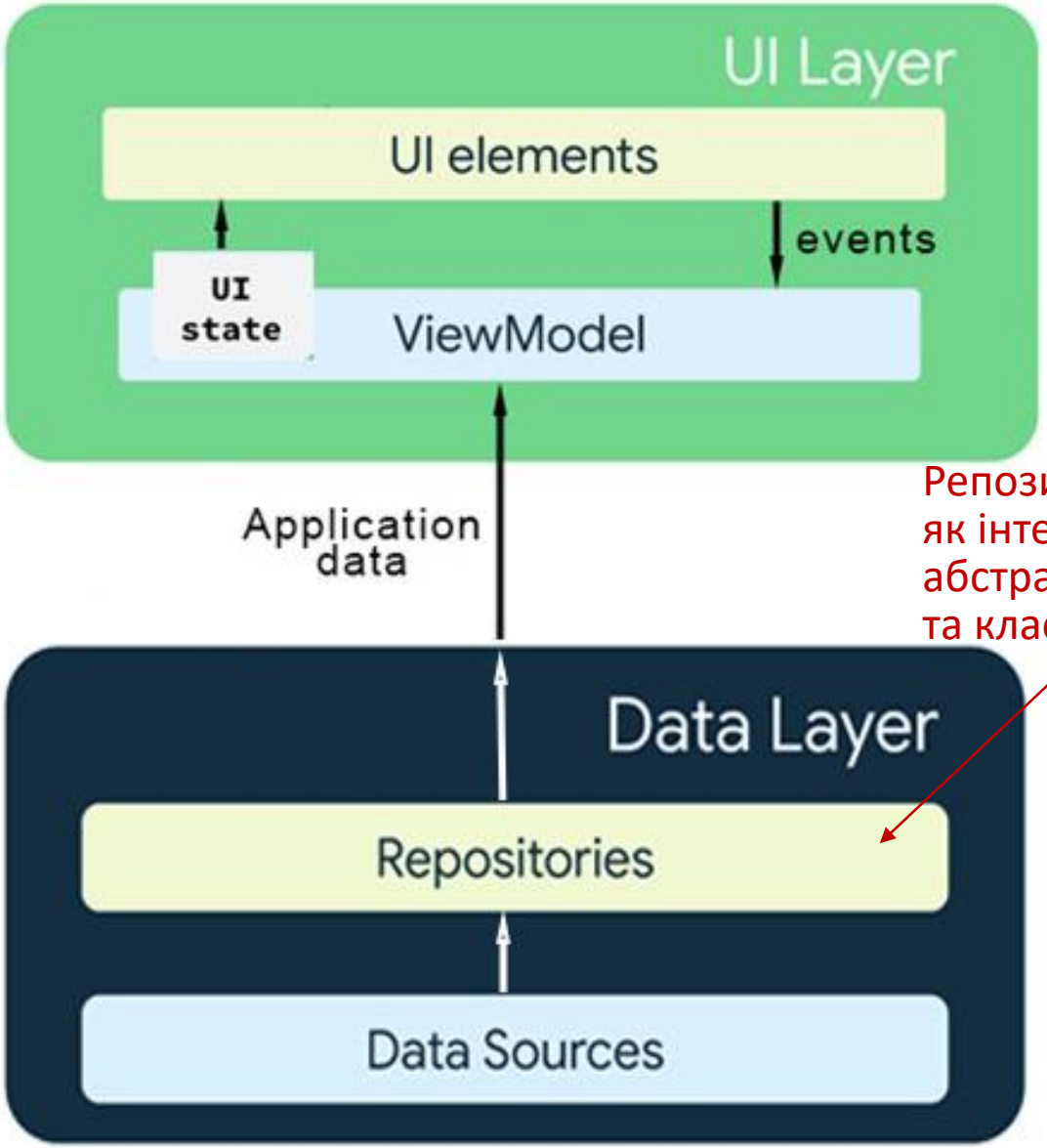
Додання Data Layer - MainActivity refactoring

створення стану, що зберігається у ViewModel та підписка на його зміни

```
...
            val mainScreenState by
                        mainViewModel.mainScreenState.collectAsStateWithLifecycle()
            MainScreen(
                mainScreenState = mainScreenState,
                getQuestionText = { index ->
                    mainViewModel.getQuestionByIndex(index).questionText
                },
                onAnswerButtonClick = { isTrue ->
                    mainViewModel.onAnswerButtonClick(isTrue)
                },
                onNextQuestionButtonClick = {
                    mainViewModel.onNextQuestionButtonClick()
                },
                innerPadding = innerPadding
            )
        }
    }
}
}
}
```

передача стану і визначень callback-функцій до Composable

Архітектура Jetpack Compose застосунку



Репозиторій реалізований як інтерфейс з абстрактними методами та клас-реалізація