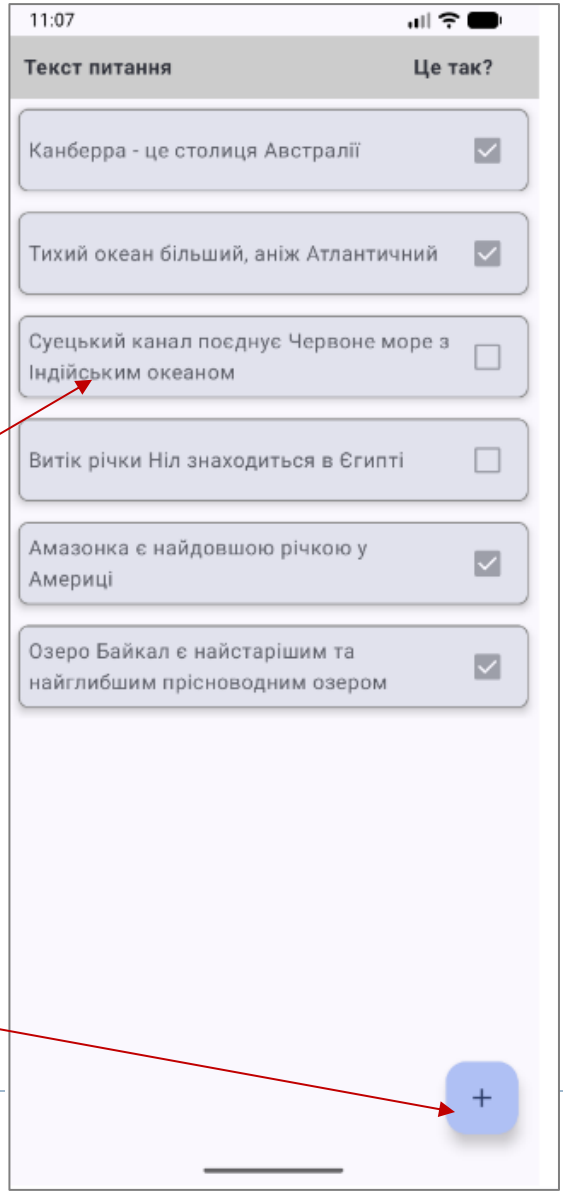
Сучасні технології мобільного програмування

Безпечна за типами навігація та передача даних між екранами

Слайди до лекцій (4 змістовий модуль)

Додання другого екрану

- Другий екран навчального застосунку містить список питань з правильною відповіддю.
- Передбачена можливість додання, видалення питань та переходу до екрану редагування (екран редагування буде доданий пізніше).



Тап по питанню відкриває екран редагування питання

Додання нового питання

Додання другого екрану – додані рядки

values\strings.xml | uk\strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name" translatable="false">GeoQuizCompose</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="next_button">Next</string>
    <string name="correct_toast">Correct!</string>
    <string name="incorrect_toast">Incorrect!</string>
    <string name="question_australia">Canberra is the capital of Australia.</string>
    <string name="question_oceans">The Pacific Ocean is larger than the Atlantic Ocean.</string>
    <string name="question_mideast">The Suez Canal connects the Red Sea and the Indian Ocean.</string>
    <string name="question_africa">The source of the Nile River is in Egypt.</string>
    <string name="question_americas">The Amazon River is the longest river in the Americas.</string>
    <string name="question_asia">Lake Baikal is the world\'s oldest and deepest freshwater lake.</string>
    <string name="questions">Questions</string>
    <string name="question_text">Question text</string>
    <string name="is_true">Is true</string>
    <string name="add_question">Add</string>
    <string name="question">Question</string>
    <string name="is_answer_true">Is answer True?</string>
    <string name="update_question">Update</string>
    <string name="remove_question">Remove</string>
    <string name="home">Quiz</string>
    <string name="settings">Settings</string>
</resources>
```

Додання другого екрану – додані рядки

values\strings.xml | uk\strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="true_button">Так</string>
    <string name="false_button">Ні</string>
    <string name="next_button">Далі</string>
    <string name="correct_toast">Правильно!</string>
    <string name="incorrect_toast">Неправильно!</string>
    <string name="question_australia">Канберра - це столиця Австралії</string>
    <string name="question_oceans">Тихий океан більший, аніж Атлантичний океан</string>
    <string name="question_mideast">Суецький канал поєднує Червоне море та Індійський океан</string>
    <string name="question_africa">Витік річки Ніл знаходиться в Єгипті</string>
    <string name="question_americas">Амазонка є найдовшою річкою у Америці</string>
    <string name="question_asia">Озеро Байкал є найстарішим та найглибшим прісноводним озером</string>
    <string name="questions">Питання</string>
    <string name="question_text">Текст питання</string>
    <string name="is_true">Це так?</string>
    <string name="add_question">Додати</string>
    <string name="question">Питання</string>
    <string name="is_answer_true">Правильна відповідь Так?</string>
    <string name="update_question">Оновити</string>
    <string name="remove_question">Видалити</string>
    <string name="home">Тест</string>
    <string name="settings">Налаштування</string>
</resources>
```

Додання другого екрану – додання залежності для іконок Material 3

libs.versions.toml | build.gradle.kts (:app)

```
1   [versions]
2   agp = "9.4.1"
    ...
12  [libraries]
13  androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
    ...
27  androidx-lifecycle-viewmodel-compose = { group = "androidx.lifecycle", name = "lifecycle-viewmodel-compose" }
28  androidx-compose-material-icons-extended = { group = "androidx.compose.material", name = "material-icons-extended" }
```

libs.versions.toml | build.gradle.kts (:app)

```
8   android {
    ...
40  dependencies {
41      implementation(platform( p0 = libs.androidx.compose.bom)) androidx.compose:compose-bom:2026.09.00
        ...
49      implementation(libs.androidx.lifecycle.viewmodel.compose) androidx.lifecycle:lifecycle-viewmodel-compose
50      implementation(libs.androidx.compose.material.icons.extended) androidx.compose.material:material-icons-extended
51      testImplementation(libs.junit) junit:junit:4.13.2
        ...
```



Додання другого екрану - QuestionListScreen

```
@Composable
fun QuestionListScreen(
    questionList: List<Question>,
    onEditQuestionClick: (Question) -> Unit = {},
    onAdd: () -> Unit = {},
    innerPadding: PaddingValues
) {
    Box(
        modifier = Modifier.fillMaxSize()
                           . padding(innerPadding)
    ) {
        LazyColumn {
            listHeader()
            items(questionList) { question ->
                QuestionCard(
                    question = question,
                    onEditQuestionClick = { onEditQuestionClick(question) }
                )
            }
        }
...
```

Як стан передається список питань та виклики функцій редагування існуючого та додання нового питання

Користувацька функція (див. далі)

Користувацька Composable функція (див. далі)

Додання другого екрану - QuestionListScreen

```
…
        FloatingActionButton(
            onClick = onAdd,
            modifier = Modifier
                .align(Alignment.BottomEnd)
                .padding(16.dp)
        ) {
            Icon(Icons.Filled.Add, contentDescription = stringResource(R.string.add_question))
        }
    }
}

private fun LazyListScope.listHeader() {
    stickyHeader {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .background(Color.LightGray)
                .padding(4.dp),
            horizontalArrangement = Arrangement.Start
        ) { …
```

Функція з material-icons-extended

Функція-розширення на інтерфейсі LazyListScope – заголовок списку

Додання другого екрану - QuestionListScreen

```
…
        Text(
            stringResource(R.string.question_text),
            fontWeight = FontWeight.Bold,
            modifier = Modifier
                .fillMaxWidth(fraction = 0.75f)
                .padding(start = 8.dp)
        )
        Text(
            stringResource(R.string.is_true),
            fontWeight = FontWeight.Bold,
        )
    }
  }
}
…
```



Додання другого екрану - QuestionListScreen

Користувацька Composable функція для елемента списку

```
…
@Composable
fun QuestionCard(
    question: Question,
    onEditQuestionClick: () -> Unit = {}
) {
    Card(
        shape = RoundedCornerShape(8.dp),
        border = BorderStroke(1.dp, Color.Gray),
        elevation = CardDefaults.cardElevation(defaultElevation = 4.dp),
        modifier = Modifier
            .fillMaxWidth()
            .padding(4.dp)
            .clickable(
                onClick = onEditQuestionClick
            )
    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier
                .padding(8.dp),
    ) { …
```

Додання другого екрану - QuestionListScreen

```
...
            Text(
                text = question.questionText,
                modifier = Modifier.fillMaxWidth(0.8f)
            )
            Checkbox(
                checked = question.answer,
                enabled = false,
                onCheckedChange = {},
            )
        }
    }
}
...
```



Додання другого екрану - QuestionListScreen

```
...
@Preview(showBackground = true)
@Composable
fun QuestionListScreenPreview() {
    val sampleQuestions = listOf(
        Question(questionText = stringResource(R.string.question_australia), answer = true),
        Question(questionText = stringResource(R.string.question_oceans), answer = true),
        Question(questionText = stringResource(R.string.question_mideast), answer = false),
    )
    QuestionListScreen(
        questionList = sampleQuestions,
        onEditQuestionClick = {},
        onAdd = {},
        innerPadding = PaddingValues(0.dp)
    )
}
```



Додання другого екрану - рефакторинг QuestionDataSource

- Додаймо до класу QuestionDataSource add, update та delete функції.

```
class QuestionDataSource {
    private val questions = mutableListOf(
        Question(questionText = "Канберра - це столиця Австралії", answer = true),
        ...
        )
    )
    fun getQuestions(): List<Question> {
        return questions
    }
    fun addQuestion(question: Question) {
        questions.add(question)
    }
    fun updateQuestion(question: Question) {
        val index = questions.indexOfFirst { it.id == question.id }
        if (index != -1) {
            questions[index] = question
        }   } ...
```

Додання другого екрану - рефакторинг QuestionDataSource

- Додаймо до класу QuestionDataSource add, update та delete функції.

```
...
    fun deleteQuestion(question: Question) {
        questions.remove(question)
    }
}
```



Додання другого екрану - рефакторинг інтерфейсу QuestionRepository

- Оскільки функціонал застосунку розширюється до можливості створення нових питань тесту та редагування видалення наявних питань, додаймо відповідні методи до інтерфейсу.

```
interface QuestionRepository {
    fun getQuestions(): List<Question>
    fun getQuestionById(id: String): Question?
    fun getQuestionListState(): StateFlow<List<Question>>
    fun addQuestion(question: Question)
    fun updateQuestion(question: Question)
    fun deleteQuestion(question: Question)
}
```

Повертає стан списку питань



Додання другого екрану - рефакторинг QuestionRepositoryImpl

- Враховуючи те, що Composable-функції більш аніж одного екрану використовують QuestionRepository, доцільно організувати роботу з єдиним екземпляром QuestionRepository - зробимо це за допомогою шаблону проєктування Singleton:

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

import ...

class QuestionRepositoryImpl private constructor(private val dataSource: QuestionDataSource) :
    QuestionRepository {
    companion object {
        @Volatile
        private var instance: QuestionRepository? = null

        fun getInstance(): QuestionRepository {
            return instance ?: synchronized(this) {
                instance ?: QuestionRepository(QuestionDataSource()).also {instance = it }
            }
        }
    }
} ...
```

Singleton

Додання другого екрану - рефакторинг QuestionRepositoryImpl

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

    override fun getQuestionById(id: String): Question? {
        return dataSource.getQuestions().find { it.id == id }
    }

    private val questionListMutableState: MutableStateFlow<List<Question>> =
        MutableStateFlow(dataSource.getQuestions())

    override fun getQuestionListState(): StateFlow<List<Question>> {
        return questionsMutableState
    }
...
```

Інкапсульований стан списку питань

Функція для отримання стану іншими компонентами



Додання другого екрану - рефакторинг QuestionRepositoryImpl

```
...
override fun addQuestion(question: Question) {
    dataSource.addQuestion(question)
    questionListMutableState.value = dataSource.getQuestions()
}

override fun updateQuestion(question: Question) {
    dataSource.updateQuestion(question)
    questionListMutableState.value = dataSource.getQuestions()
}

override fun deleteQuestion(question: Question) {
    dataSource.deleteQuestion(question)
    questionListMutableState.value = dataSource.getQuestions()
}
```

Стан списку питань



Додання другого екрану - ListUIState

- QuestionListScreen може мати три стани в залежності від взаємодії з користувачем:
- відображення списку,
- перехід до екрану QuestionScreen для додання нового питання при клікі на FAB,
- перехід до екрану QuestionScreen для редагування питання, по якому виконаний клік.

```
sealed interface ListUiState {
    data object List : ListUiState                  // show QuestionListScreen
    data object Add : ListUiState                   // show QuestionScreen (empty form)
    data class Edit(val questionId: String) : ListUiState  // show QuestionScreen for this question
}
```



Додання другого екрану - QuestionListViewModel

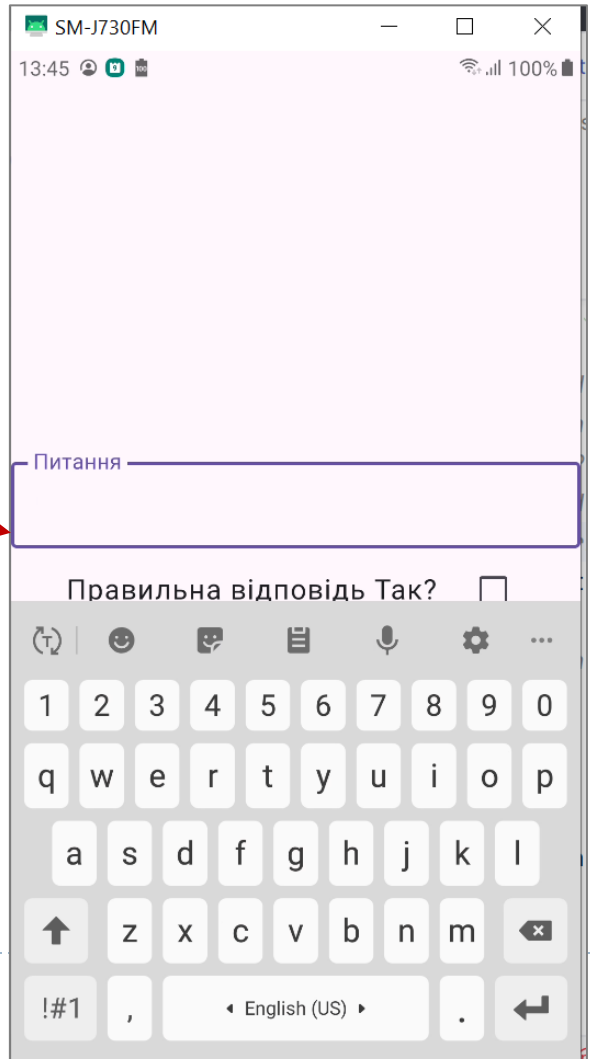
- QuestionListViewModel приймає як аргумент QuestionRepository та містить стан списку питань та стан екрану списку питань з визначенням обробників подій на цьому екрані.

```
class QuestionListViewModel(questionRepository: QuestionRepository) : ViewModel() {
    val questionListFlow = questionRepository.getQuestionListState()
    private val _uiState = MutableStateFlow<ListUiState>(ListUiState.List)
    val uiState = _uiState.asStateFlow()
    fun onAddQuestionClick() {
        _uiState.value = ListUiState.Add
    }
    fun onEditQuestionClick(q: Question) {
        _uiState.value = ListUiState.Edit(q.id)
    }
    fun onBack() {
        _uiState.value = ListUiState.List
    }  // call after Save or Back
}
```

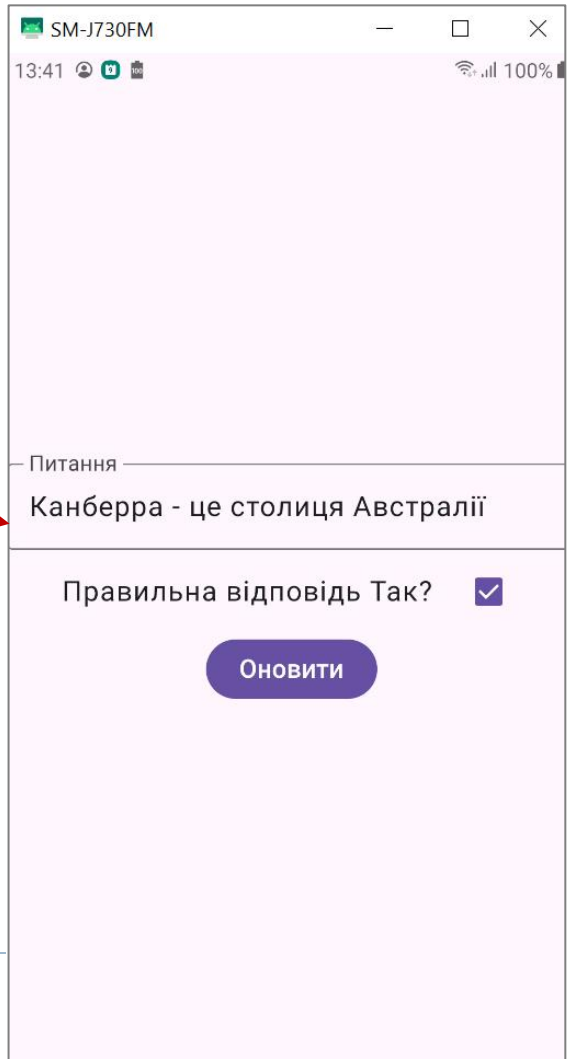
Додання третього екрану

- Третій екран навчального застосунку використовується для додавання нового питання тесту та редагування існуючого питання.

Додання питання



Редагування питання



Додання третього екрану - QuestionScreen

```
@Composable
fun QuestionScreen(
    questionText: String,
    answer: Boolean,
    buttonTextRes: Int,
    onQuestionTextChange: (String) -> Unit,
    onAnswerChange: (Boolean) -> Unit,
    onAddUpdateButtonClick: () -> Unit,
    onRemoveButtonClick: () -> Unit,
    innerPadding: PaddingValues
) {
    val localFocusManager = LocalFocusManager.current
    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
            .fillMaxSize()
            .padding(innerPadding)
            .pointerInput(Unit) {
                detectTapGestures(onTap = {
                    localFocusManager.clearFocus()          })          } …
```

Організація приховання екранної клавіатури при тапі зовні OutlinedTextField

Додання третього екрану - QuestionScreen

```
…
    ) {
        OutlinedTextField(
            value = questionText,
            onValueChange = onQuestionTextChange,
            label = { Text(stringResource(R.string.questions)) },
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(6.dp))
        Row(
            horizontalArrangement = Arrangement.Center,
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier.fillMaxWidth()
        ) {
            Text(text = stringResource(R.string.is_answer_true))
            Spacer(modifier = Modifier.height(6.dp))
            Checkbox(
                checked = answer,
                onCheckedChange = onAnswerChange
            )
    } …
```

Додання третього екрану - QuestionScreen

```
...
        Spacer(modifier = Modifier.height(6.dp))
        Button(
            onClick = onAddUpdateButtonClick
        ) {
            Text(text = stringResource(buttonTextRes))
        }
        Spacer(modifier = Modifier.height(6.dp))
        Button(
            onClick = onRemoveButtonClick
        ) {
            Text(text = stringResource(R.string.remove_question))
        }
    }
}
...
```



Додання третього екрану - QuestionScreen

```
...
@Preview(showBackground = true)
@Composable
fun AddQuestionScreenPreview() {
    QuestionScreen(
        questionText = "",
        answer = false,
        buttonTextRes = R.string.add_question,
        onQuestionTextChange = {},
        onAnswerChange = {},
        onAddUpdateButtonClick = {},
        onRemoveButtonClick = {},
        innerPadding = PaddingValues()
    )
}
...
```



Додання третього екрану - QuestionScreen

```
...
@Preview(showBackground = true)
@Composable
fun EditQuestionScreenPreview() {
    QuestionScreen(
        questionText = "Канберра - це столиця Австралії",
        answer = true,
        buttonTextRes = R.string.update_question,
        onQuestionTextChange = {},
        onAnswerChange = {},
        onAddUpdateButtonClick = {},
        onRemoveButtonClick = {},
        innerPadding = PaddingValues()
    )
}
```



Додання третього екрану - QuestionRepository & QuestionRepositoryImpl

- Екран додання/редагування питання тесту буде приймати конкретне питання, але нам не потрібно на цьому екрані спостерігати **MutableStateFlow<Question>,** оскільки передача інформації йде тільки від цього екрану і не повертаєься до нього.
- В результаті класи QuestionRepository & QuestionRepositoryImpl залишаються без змін.



Додання третього екрану - QuestionViewModel

- QuestionViewModel приймає аргумент QuestionRepository та questionId. У разі questionId = null, екран відкривається у режимі додання нового питання, у іншому разі – в режимі редагування:

```
private val TAG = "QuestionViewModel«

class QuestionViewModel(
    private val questionRepository: QuestionRepository,
    private val questionId: String? = null // null = "add" mode, not null = "edit" mode
) : ViewModel() {

    private val initialQuestion = questionId?.let { questionRepository.getQuestionById(it) }

    private val _questionText = MutableStateFlow(initialQuestion?.questionText ?: "")

    val questionText: StateFlow<String> = _questionText.asStateFlow()

    private val _answer = MutableStateFlow(initialQuestion?.answer ?: false)

    val answer: StateFlow<Boolean> = _answer.asStateFlow()


    fun onQuestionTextChange(newText: String) {
        _questionText.value = newText
    }
```

...

Додання третього екрану - QuestionViewModel

```
...
    fun onAnswerChange(newAnswer: Boolean) {
        _answer.value = newAnswer
    }

    fun saveQuestion() {
        val question = Question(
            id = initialQuestion?.id ?: UUID.randomUUID().toString(),
            questionText = _questionText.value,
            answer = _answer.value
        )
        if (initialQuestion == null) {
            questionRepository.addQuestion(question)
        } else {
            questionRepository.updateQuestion(question)
        }
        Log.d(TAG, "Saved question: $question")
    }

    fun deleteQuestion() {
        initialQuestion?.let { questionRepository.deleteQuestion(it) }
    }
}
```

Перевірка – Main Activity

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            GeoQuizComposeDemoTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    val questionDataSource = QuestionDataSource()
                    val questionRepository = QuestionRepositoryImpl.getInstance(questionDataSource)
                    /* For MainScreen  - закоментовано*/
                    /* For QuestionListScreen */
                    val listViewModel: QuestionListViewModel = viewModel(
                        factory = ViewModelFactory(
                            modelClass = QuestionListViewModel::class.java,
                            creator = { QuestionListViewModel(questionRepository) }
                        )
                    )
                    val questionList by listViewModel.questionListFlow.collectAsStateWithLifecycle()
                    val state by listViewModel.uiState.collectAsStateWithLifecycle()
```

Стан списку питань

Стан екрану списку питань

...

Перевірка – Main Activity

...

```
BackHandler(enabled = state != ListUiState.List) {
    listViewModel.onBack()
}
when (val s = state) {
    ListUiState.List -> QuestionListScreen(
        questionList = questionList,
        onEditQuestionClick = listViewModel::onEditQuestionClick,
        onAdd = listViewModel::onAddQuestionClick,
        innerPadding = innerPadding
    )

    ListUiState.Add -> QuestionScreenHost(
        questionRepository = questionRepository,
        questionId = null,
        buttonTextRes = R.string.add_question,
        onDone = listViewModel::onBack,
        innerPadding = innerPadding
    )
```

Обробник події тапу по системній кнопці Back

Користувацька Composable функція для виведення екрану питання у режимі додання або редагування

...



Перевірка – Main Activity

```
...
                is ListUiState.Edit -> QuestionScreenHost(
                    questionRepository = questionRepository,
                    questionId = s.questionId,
                    buttonTextRes = R.string.update_question,
                    onDone = listViewModel::onBack,
                    innerPadding = innerPadding
                )
            }
        }
    }
}
...
```

Користувацька Composable функція для виведення екрану питання у режимі додання або редагування



Перевірка - Main Activity

```
...
@Composable
private fun QuestionScreenHost(
    questionRepository: QuestionRepository,
    questionId: String?,
    buttonTextRes: Int,
    onDone: () -> Unit,
    innerPadding: PaddingValues
) {
    val questionViewModel: QuestionViewModel = viewModel(
        key = questionId ?: "new",
        factory = ViewModelFactory(
            modelClass = QuestionViewModel::class.java,
            creator = { QuestionViewModel(questionRepository, questionId) }
        )
    )
    val questionText by questionViewModel.questionText.collectAsStateWithLifecycle()
    val answer by questionViewModel.answer.collectAsStateWithLifecycle()

...
```

Користувацька Composable функція для виведення екрану питання у режимі додання або редагування

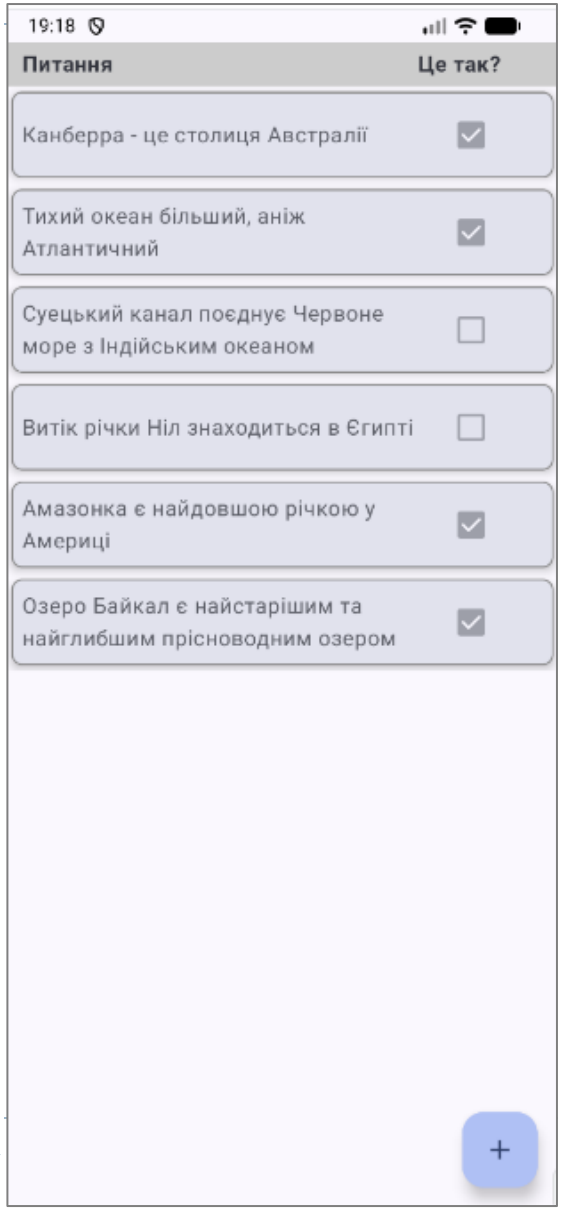


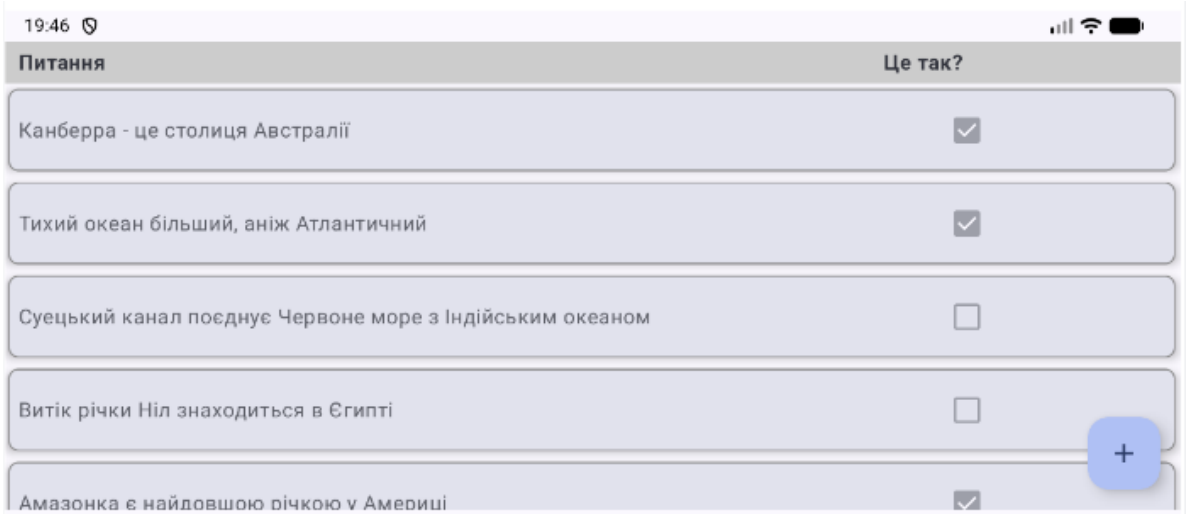
Перевірка - Main Activity

```
...
    QuestionScreen(
        questionText = questionText,
        answer = answer,
        buttonTextRes = buttonTextRes,
        onQuestionTextChange = questionViewModel::onQuestionTextChange,
        onAnswerChange = questionViewModel::onAnswerChange,
        onAddUpdateButtonClick = {
            questionViewModel.saveQuestion()
            onDone()
        },
        onRemoveButtonClick = {
            questionViewModel.deleteQuestion()
            onDone()
        },
        innerPadding = innerPadding
    )
}
```

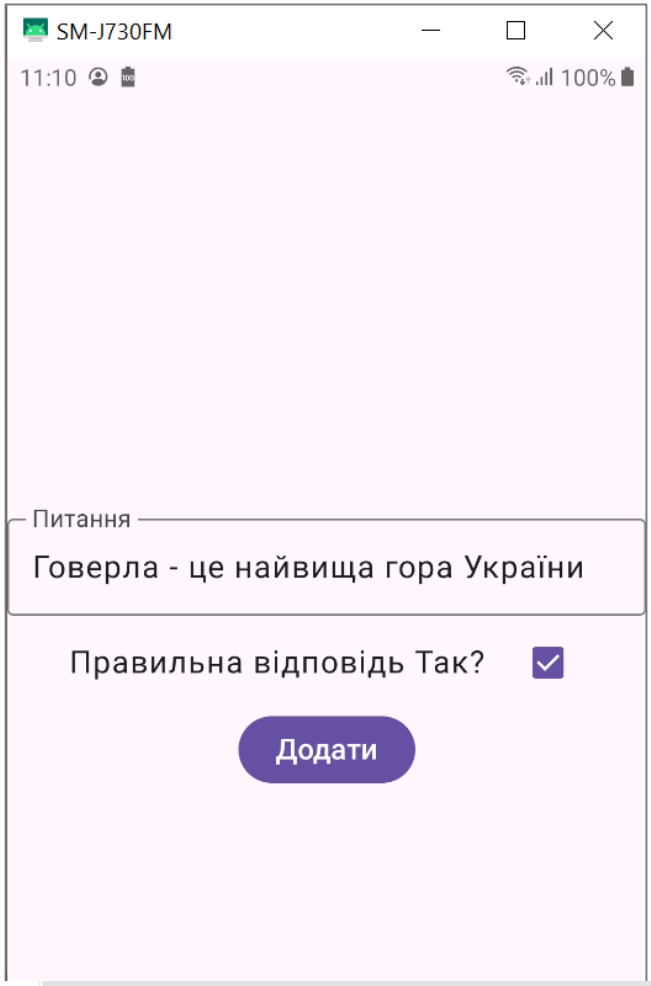


Перевірка – Main Activity





Перевірка додання питання



Logcat Logcat

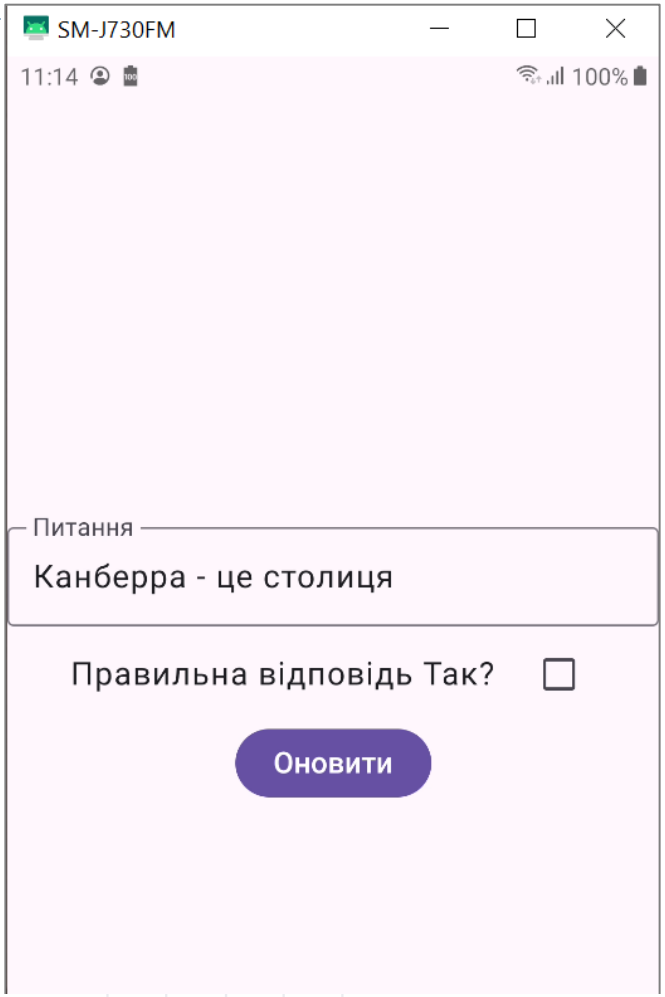
samsung SM-J730FM (52001a65fe66c45d) Android 9

tag=:QuestionScreen

Cc

```
ером, answer=true), Question(id=ac93d5f9-daae-40d1-9041-cc4a1dbae7e6, questionText=Говерла - це найвища гора України, answer=true)]
```

Перевірка редагування питання



Logcat

Logcat

samsung SM-J730FM (52001a65fe66c45d) Android 9

tag=:QuestionScreen

D Question bank: [Question(id=9264ce61-1160-4671-bf25-bda7947241cb, questionText=Канберра - це столиця, answer=false),

Рефакторинг MainScreenViewModel

- Наразі MainScreenViewModel для отримання банку питань використовує нереактивну функцію з репозитарію
  fun getQuestions(): List<Question>
- Для динамічного опрацювання списку питань при їх доданні/редагуванні/видаленні необхідно користуватися функцією
  fun getQuestionListState(): StateFlow<List<Question>>
- Тому видалимо fun getQuestions(): List<Question> з QuestionRepository та QuestionRepositoryImpl, а ініціалізацію стану та отримання банку питань у MainScreenViewModelперепишемо:

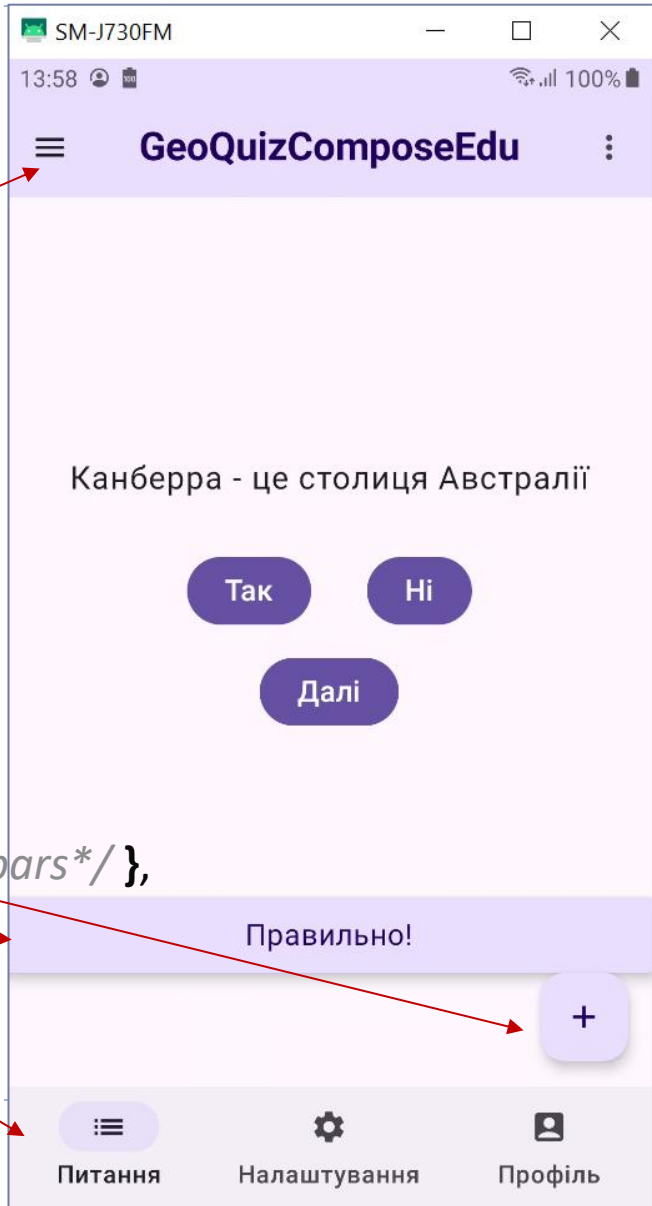
```
private val _mainScreenState = MutableStateFlow(
    MainScreenState(currentQuestion =
                    questionRepository.getQuestionListState().value.first().questionText)
)
val mainScreenState = _mainScreenState.asStateFlow()
```

```
val questionBank = questionRepository.getQuestionListState().value
```

Scaffold

- Scaffold - контейнер з бібліотеки Material3, що забезпечує стандартний шаблон для створення складних UI.
- Scaffold надає слоти для інших стандартних контейнерів.
- Scaffold автоматично читає Material3 тему застосунку.

```
Scaffold(
        topBar = { /* Application top bar (Toolbar)*/ },
        bottomBar = { /* Application bottom bar
                        (usually navigation bar)*/ },
        floatingActionButton = { /* FAB button */ },
        floatingActionButtonPosition = FabPosition.End,
        snackbarHost = {/* Snackbar host to display snack-bars*/ },
        modifier = Modifier.fillMaxSize()
    ) { innerPadding ->
      // main content
    }
}
```



Scaffold - innerPadding

- Значення аргументу innerPadding : PaddingValues Scaffold передається до основного вмісту автоматично, забезпечуючи правильні відступи від панелей:

```
package ua.edu.znu.geoquizcomposeedu

import ...

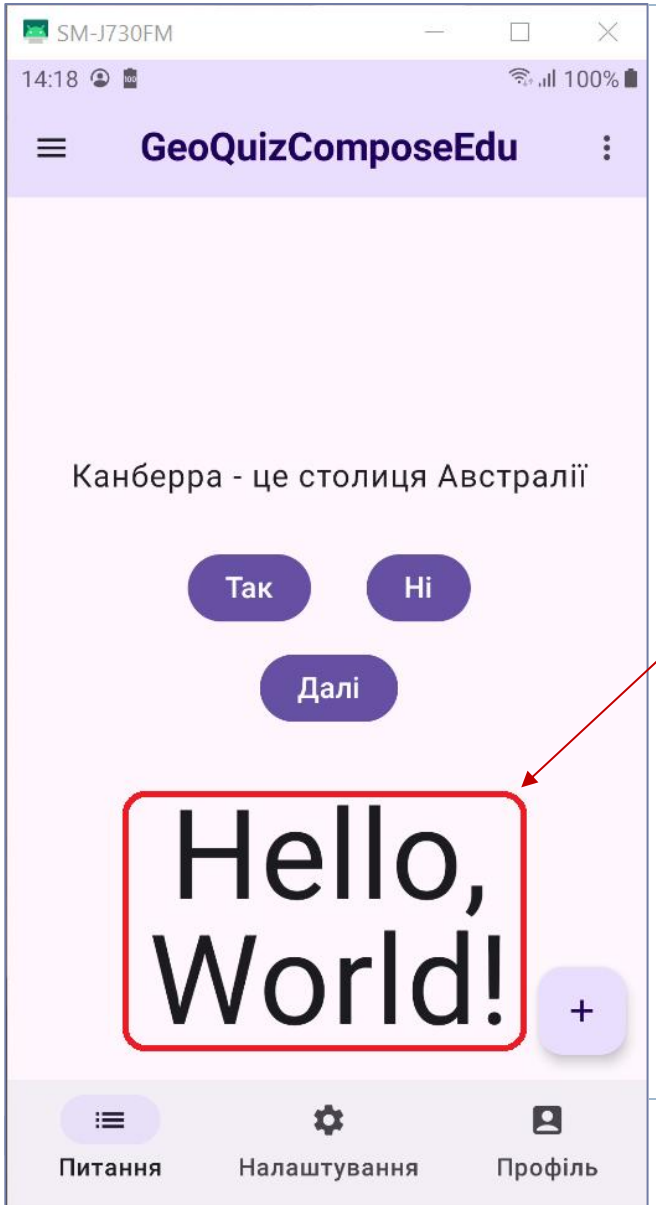
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            GeoQuizComposeEduTheme {
                val snackbarHostState = remember { SnackbarHostState() }
                Scaffold(
                    topBar = { TopAppBar()},
                    floatingActionButton = {
                        AppFloatingActionButton(
                            onFabClick = {/* TODO: Navigate to add question screen */ }
                        )
                    }, ...
```

Scaffold - innerPadding

```
...
        floatingActionButtonPosition = FabPosition.End,
        bottomBar = { BottomAppBar() },
        snackbarHost = {
            AppSnackbar(snackbarHostState = snackbarHostState)
        },
    ) { innerPadding ->
        MainScreen(
            innerPadding = innerPadding,
            snackbarHostState = snackbarHostState
        )
        Text(
            text = "Hello, World!",
            textAlign = TextAlign.Center,
            fontSize = 20.sp,
//              fontSize = 100.sp,
//              lineHeight = 100.sp,
            modifier = Modifier
                .padding(innerPadding)
                .fillMaxSize()
                .wrapContentHeight(Alignment.Bottom)
                .padding(16.dp)                )            }        }    }}
```

Scaffold - innerPadding



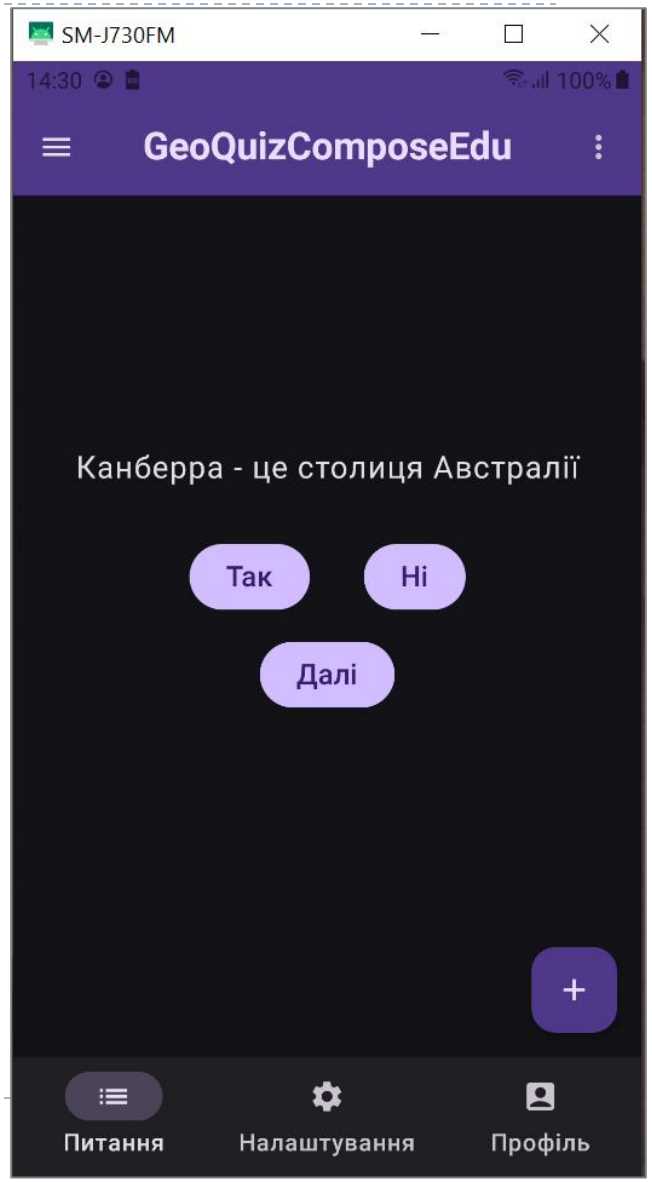
з innerPadding

без innerPadding



Scaffold - використання теми Material3

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            GeoQuizComposeEduTheme(
                darkTheme = true
            ) {
                val snackbarHostState =
                                remember { SnackbarHostState() }
                Scaffold(...){
...
}
```

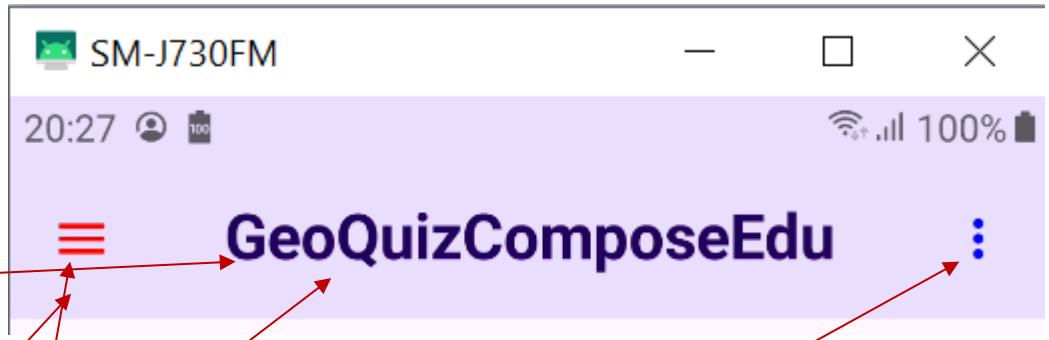


Material3-компоненти Scaffold - TopAppBar

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

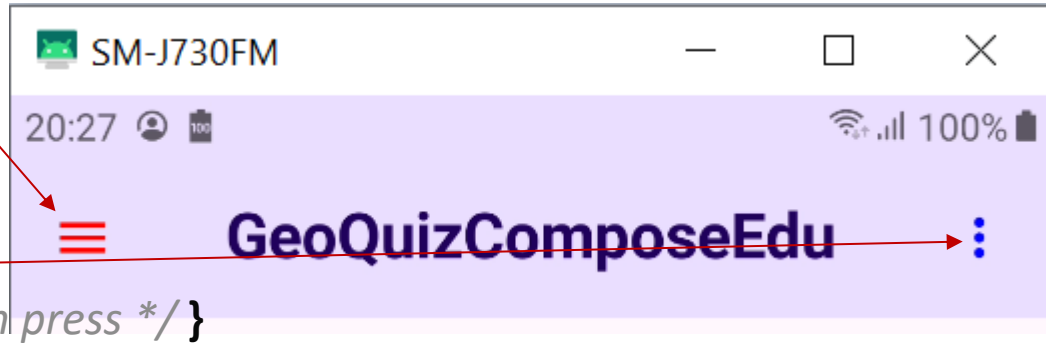
import ...

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun GeiQuizTopAppBar() {
//    TopAppBar(
    CenterAlignedTopAppBar(
//    MediumTopAppBar(
//    LargeTopAppBar(
        title = {
            Text(
                text = stringResource(R.string.app_name),
                fontWeight = FontWeight.Bold
            )       },
        colors = TopAppBarDefaults.topAppBarColors(
            containerColor = MaterialTheme.colorScheme.primaryContainer,
            titleContentColor = MaterialTheme.colorScheme.onPrimaryContainer,
            /* Its possible to set custom color for icons, but its not recommended */
            navigationIconContentColor = Color.Red,
            actionIconContentColor = Color.Blue        ), ...
```



Material3-компоненти Scaffold - TopAppBar

```
...
    navigationIcon = {
        IconButton(
            onClick = { /* Handle navigation icon press */ }
        ) {
            Icon(
                imageVector = Icons.Default.Menu,
                contentDescription = stringResource(R.string.main_menu)
            )
        }
    },
    actions = {
        IconButton(
            onClick = { /* Handle action icon press */ }
        ) {
            Icon(
                imageVector = Icons.Default.MoreVert,
                contentDescription = stringResource(R.string.more_actions)
            )
        }
    }  )}
```

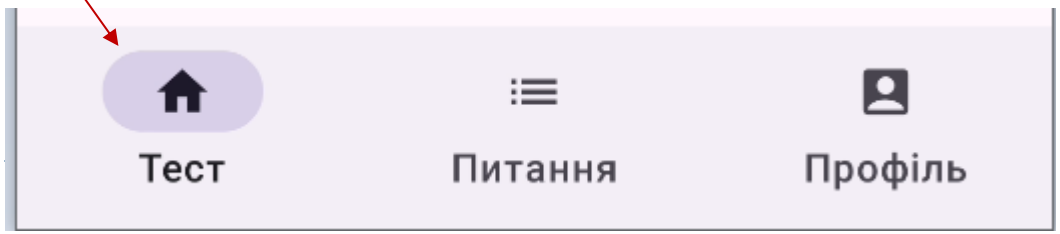


Material3-компоненти Scaffold - NavigationBar

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

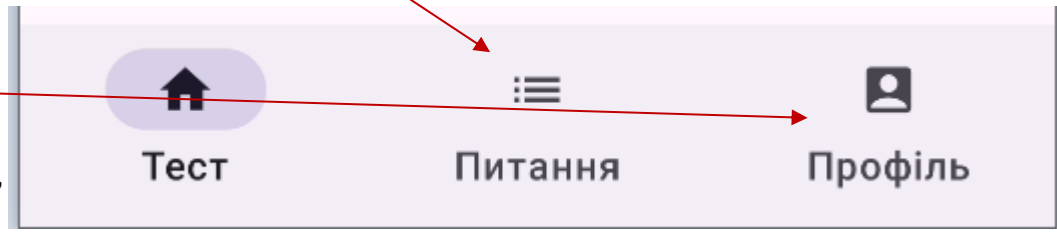
import ...

@Composable
fun BottomAppBar() {
    NavigationBar {
        NavigationBarItem(
            selected = true,
            onClick = { /* Handle menu click */ },
            icon = {
                Icon(
                    imageVector = Icons.Default.Home,
                    contentDescription = stringResource(R.string.home)
                )
            },
            label = { Text(stringResource(R.string.home)) },
        )
...
```



Material3-компоненти Scaffold - NavigationBar

```
NavigationBarItem(
    selected = false,
    onClick = { /* Handle menu click */ },
    icon = {
        Icon(
            imageVector = Icons.AutoMirrored.Filled.List,
            contentDescription = stringResource(R.string.questions)
        )
    },
    label = { Text(stringResource(R.string.questions)) },
)
NavigationBarItem(
    selected = false,
    onClick = { /* Handle menu click */ },
    icon = {
        Icon(
            imageVector = Icons.Default.AccountBox,
            contentDescription = stringResource(R.string.profile)
        )
    },
    label = { Text(stringResource(R.string.profile)) },      )   } }
```



Еволюція навігації в Compose

- До появи типобезпечної (Type-Safe) навігації розробники Jetpack Compose значною мірою покладалися на визначення маршрутів на основі рядків для керування навігацією між екранами. Цей підхід вимагав ручного створення маршрутів, вбудовування параметрів у рядки та розбору цих рядків у цільових функціях Composable. Хоча цей метод був функціональним, він спричиняв кілька проблем:

```
// Navigating to a profile screen with an ID
navController.navigate("profile/123")
```

- У цьому прикладі «profile/123» — це рядок, який кодує маршрут та його параметр. Однак така простота має кілька недоліків:



Проблеми з навігацією на основі рядків:

1. **Відсутність безпеки типів**: параметри передаються як рядки, що вимагає ручного вилучення та перетворення. Якщо очікувалося, що параметр буде цілим числом, але було передано рядок, ця невідповідність не буде виявлена до моменту виконання, що потенційно може спричинити збої.

```
// Extracting the ID from the route string
val profileId = backStackEntry.arguments?
                              .getString("id")?.toIntOrNull() ?: 0
```

Якщо перетворення не вдається (наприклад, через нечисловий рядок), програма може працювати непередбачувано або аварійно завершити роботу.



Проблеми з навігацією на основі рядків:

2. **Ручне конструювання рядків**: Розробникам доводилося вручну об'єднувати та інтерполювати рядки для створення маршрутів, що призводило до помилок, якщо форматування маршруту було неправильним.

```
// Incorrectly formatted route
val userId = "123"
navController.navigate("profile/userId")
                                        // Should be "$userId"
```

Такі проблеми були поширеними, особливо у великих застосунках з багатьма параметрами, що робило логіку навігації схильною до помилок та складнішою в обслуговуванні.



Проблеми з навігацією на основі рядків:

3. **Помилки виконання**: Оскільки маршрути навігації були побудовані як рядки, помилки в іменах маршрутів або типах параметрів не виявлялися, доки програма не запустилася, що ускладнювало налагодження.

```
// Navigation failure due to typo
navController.navigate("profiles/123") // Incorrect route name
```

Ця помилка виявлятиметься лише під час виконання, що ускладнюватиме відстеження та виправлення проблем.

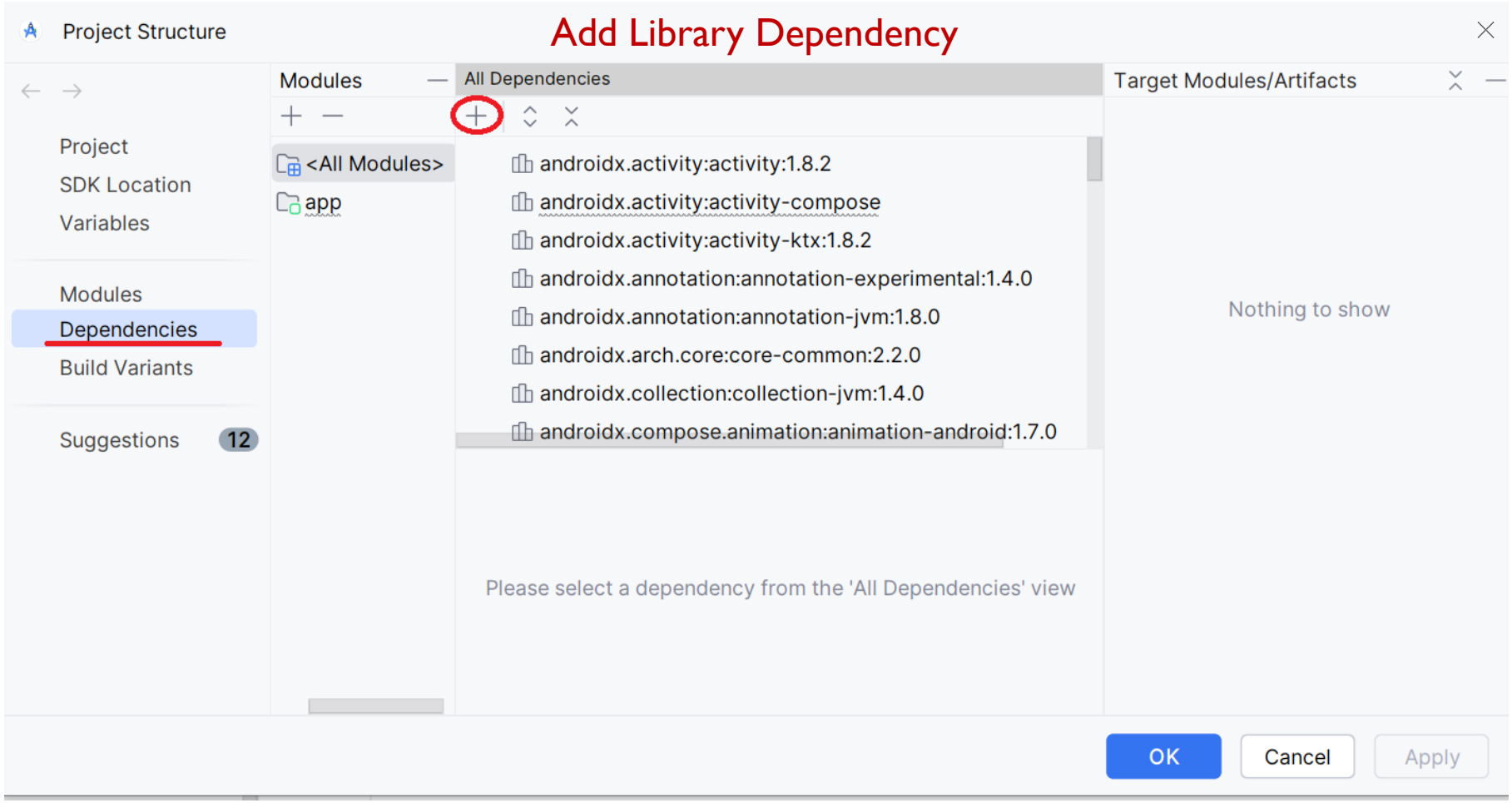


Проблеми з навігацією на основі рядків:

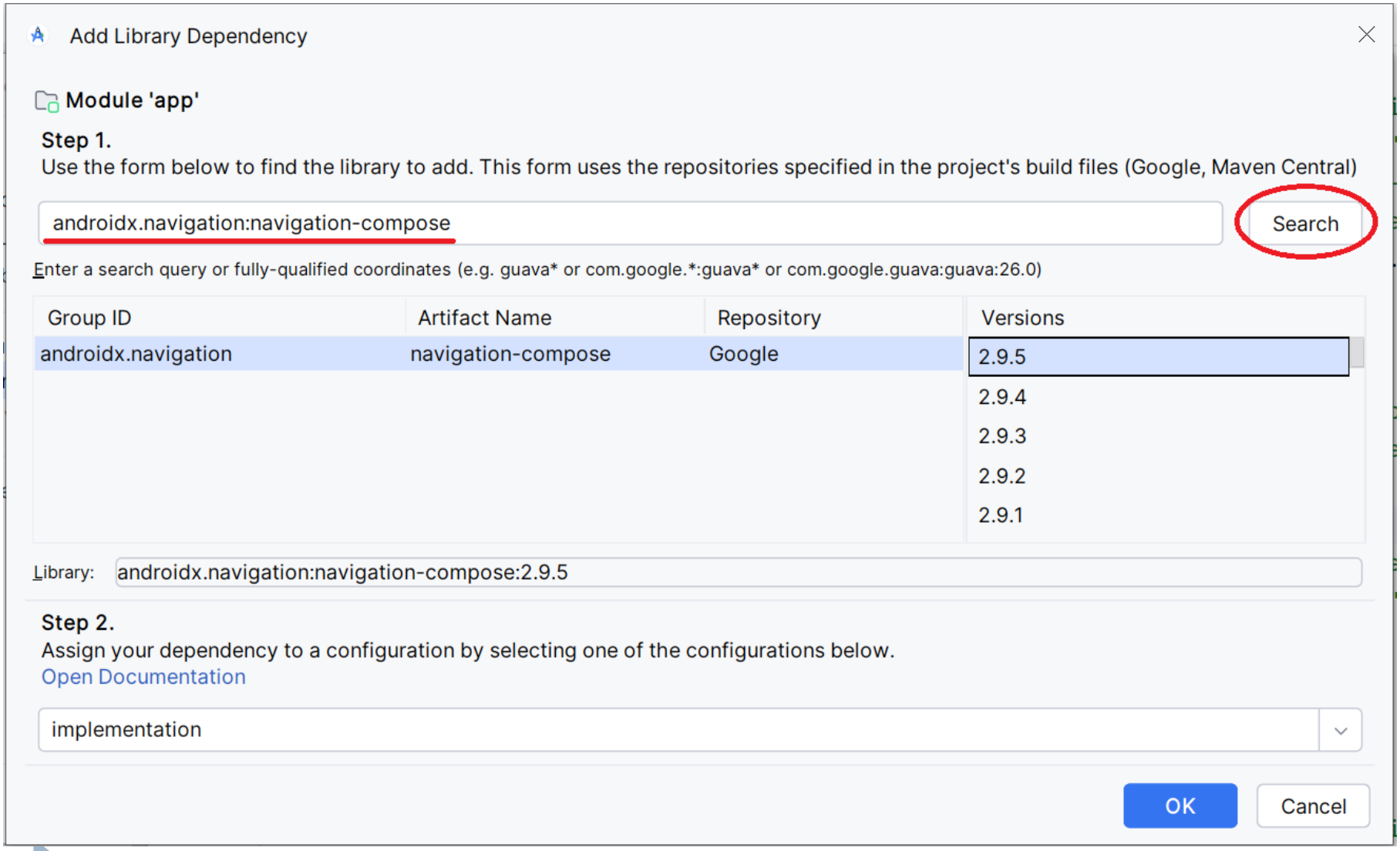
4. **Обмежена масштабованість**: Зі зростанням складності застосунків керування кількома маршрутами та параметрами за допомогою рядків ставало громіздким, а ризик невідповідностей зростав. Це було особливо складно під час підтримки навігації між різними модулями або функціями в застосунку.

Підсумовуючи, хоча рядкова навігація працювала, їй бракувало надійності, необхідної для більших та складніших застосувань. Потреба в більш надійному та зручному в обслуговуванні підході призвела до розробки типобезпечної (Type-Safe) навігації.

Додання залежностей



Додання залежностей



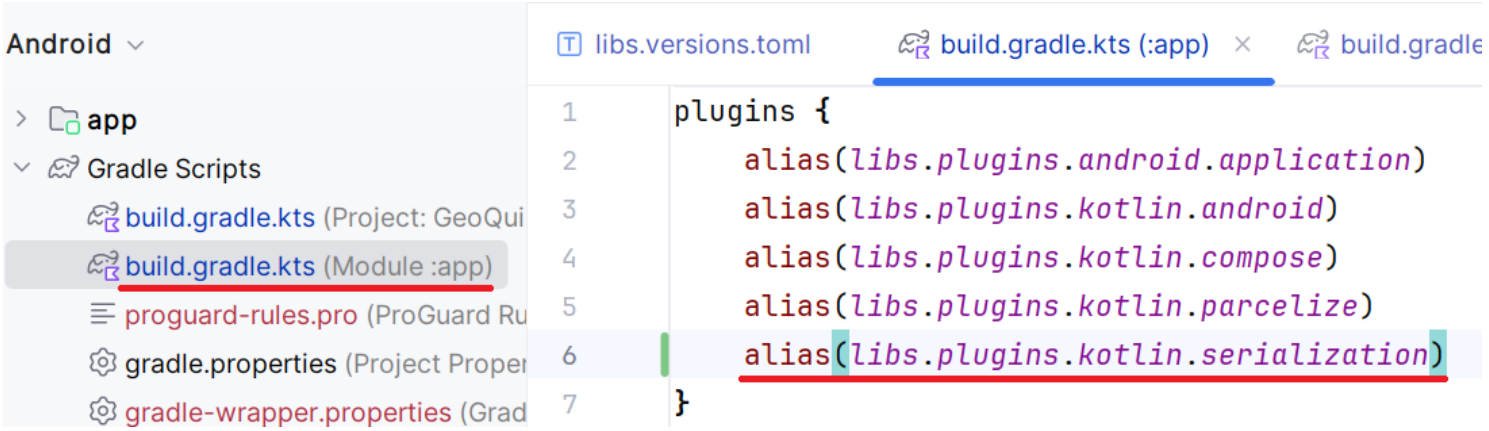
Додання залежностей



Додання плагіну org.jetbrains.kotlin.plugin.serialization

gradle.properties (P
gradle-wrapper.pro
libs.versions.toml (\
local.properties (SD
settings.gradle.kts (

```
37  [plugins]
38  android-application = { id = "com.android.application", version.ref = "agp" }
39  kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
40  kotlin-compose = { id = "org.jetbrains.kotlin.plugin.compose", version.ref = "kotlin" }
41  kotlin-parcelize = { id = "org.jetbrains.kotlin.plugin.parcelize", version.ref = "kotlin" }
42  kotlin-serialization = { id = "org.jetbrains.kotlin.plugin.serialization", version.ref = "kotlin" }
```





Частини навігаційного компоненту

- **Навігаційний граф** – це декларативний спосіб опису зв'язків між різними екранами (**composable-функціями**) застосунку, який використовує компонент **NavHost** для керування переходами. Цей граф дозволяє переміщуватися між екранами за допомогою унікальних ідентифікаторів (**URI/route**) без необхідності знати їхню внутрішню реалізацію.
- **NavHost** - Це основний компонент API навігації, він приймає **NavHostController** (для керування навігацією) та startDestination (початковий екран).
- **NavController** - відстежує стек Composable функцій, що складають екрани застосунку, а також стан кожного екрана.
- **composable()**: Ця функція використовується всередині NavHost для зв'язування конкретного route (URI) з певною composable-функцією, яка буде відображатися при переході за цим route.



Проста навігація між двома екранами - UI

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

import ...

@Composable
fun FirstScreen(
    onNavigateForward: () -> Unit
) {
    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.fillMaxSize()
    ) {
        Text("First Screen")
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick =  onNavigateForward) {
            Text("Go forward")
        }
    }
}
```

callback для навігації на другий екран



Проста навігація між двома екранами - UI

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

import ...

@Composable
fun SecondScreen(
    onNavigateBack: () -> Unit
) {
    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.fillMaxSize()
    ) {
        Text("Second Screen")
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = onNavigateBack) {
            Text("Go back")
        }
    }
}
```

callback для повернення на перший екран



Проста навігація між двома екранами - Routes

- Щоб використовувати типобезпечні маршрути, спочатку потрібно визначити серіалізовані класи або об'єкти, які представляють маршрути (зазвичай це виконується у запечатаному класі Routes.kt, який також повинен бути серіалізованим).
- Використовуйте такі правила при визначенні маршрутів:
  - визначайте об'єкт або об'єкт даних для маршрутів без аргументів;
  - визначайте клас або клас даних для маршрутів з аргументами;

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation.nav

import kotlinx.serialization.Serializable

@Serializable
sealed class Routes {

    @Serializable
    object FirstScreen      // First screen route

    @Serializable
    object SecondScreen     // Second screen route
```

Проста навігація між двома екранами

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation

import ...

@Composable
fun NavAppSimple(innerPadding: PaddingValues) {
    val navController = rememberNavController()
    NavHost(
        navController = navController,
        startDestination = Routes.FirstScreen,
        modifier = Modifier.padding(innerPadding)
    ) {
        composable<Routes.FirstScreen> {
            FirstScreen(
                onNavigateForward = {
                    navController.navigate(Routes.SecondScreen)
                }
            )
        }
...
```

Переходить на другий екран та додає його до стеку navController-а

Проста навігація між двома екранами

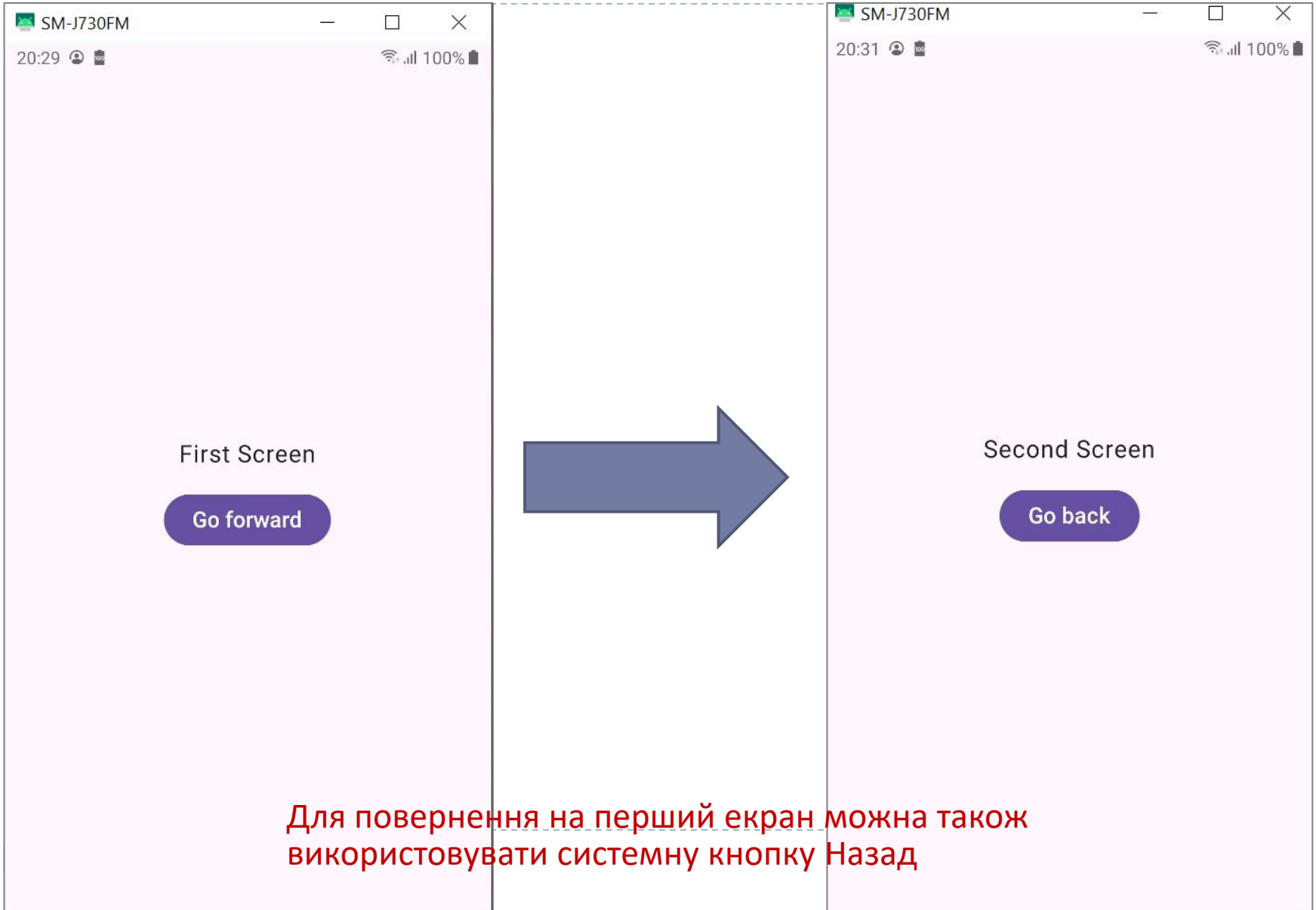
```
...
        composable<Routes.SecondScreen> {
            SecondScreen(
                onNavigateBack = {
                    navController. popBackStack()
                }
            )
        }
    }
}
```

Переходить на попередній екран, видаляючи його зі стеку navController-a, navController.navigate(Routes.SecondScreen) додав би дублікат маршруту до стеку - це не коректно.

```
package ua.edu.znu.geoquizcomposeedu
import ...
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
                NavAppSimple(innerPadding)
...
```



Проста навігація між двома екранами



Для повернення на перший екран можна також використовувати системну кнопку Назад

Навігація з передачею примітивного параметра - UI

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

import ...

@Composable
fun FirstScreen(
    onNavigateForward: (String) -> Unit
) {
    val customPrimitive = remember { mutableStateOf("") }
    val localFocusManager = LocalFocusManager.current

    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.fillMaxSize().
            pointerInput(Unit) {
                detectTapGestures(onTap = {
                    localFocusManager.clearFocus()
                })
            }
...
```

callback для навігації на другий екран приймає параметр

для приховання віртуальної клавіатури

Навігація з передачею примітивного параметра - UI

```
...
    ) {
        Text("First Screen")
        Spacer(modifier = Modifier.height(16.dp))
        TextField(
            value = customPrimitive.value,
            onValueChange = { customPrimitive.value = it },
            label = { Text("Enter custom primitive") }
        )
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = { onNavigateForward(customPrimitive.value) }) {
            Text("Go forward")
        }
    }
}
```



Навігація з передачею примітивного параметра - UI

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

import ...

@Composable
fun SecondScreen(
    customPrimitive: String,
    onNavigateBack: () -> Unit
) {
    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.fillMaxSize()
    ) {
        Text("Second Screen - $customPrimitive")
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = onNavigateBack) {
            Text("Go back")
        }
    }
}
```

параметр передається першим екраном

Навігація з передачею примітивного параметра - Routes

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation.nav

import kotlinx.serialization.Serializable

@Serializable
sealed class Routes {

    @Serializable
    data object FirstScreen : Routes()

    @Serializable
    data class SecondScreen(val customPrimitive: String) : Routes()
}
```

Успадкування від Routes дозволить поліморфно працювати з маршрутами

Використовується клас, оскільки маршрут передає параметр



Навігація з передачею примітивного параметра

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation

import ...

@Composable
fun NavAppPassPrimitive(innerPadding: PaddingValues) {
    val navController = rememberNavController()
    NavHost(
        navController = navController,
        startDestination = Routes.FirstScreen,
        modifier = Modifier.padding(innerPadding)
    ) {
        composable<Routes.FirstScreen> {
            FirstScreen(
                onNavigateForward = {
                    customPrimitive ->
                navController.navigate(Routes.SecondScreen(customPrimitive))
                }
            )
    }
...
```

Переходить на другий екран з передачею параметра та додає другий екран до стеку navController-а

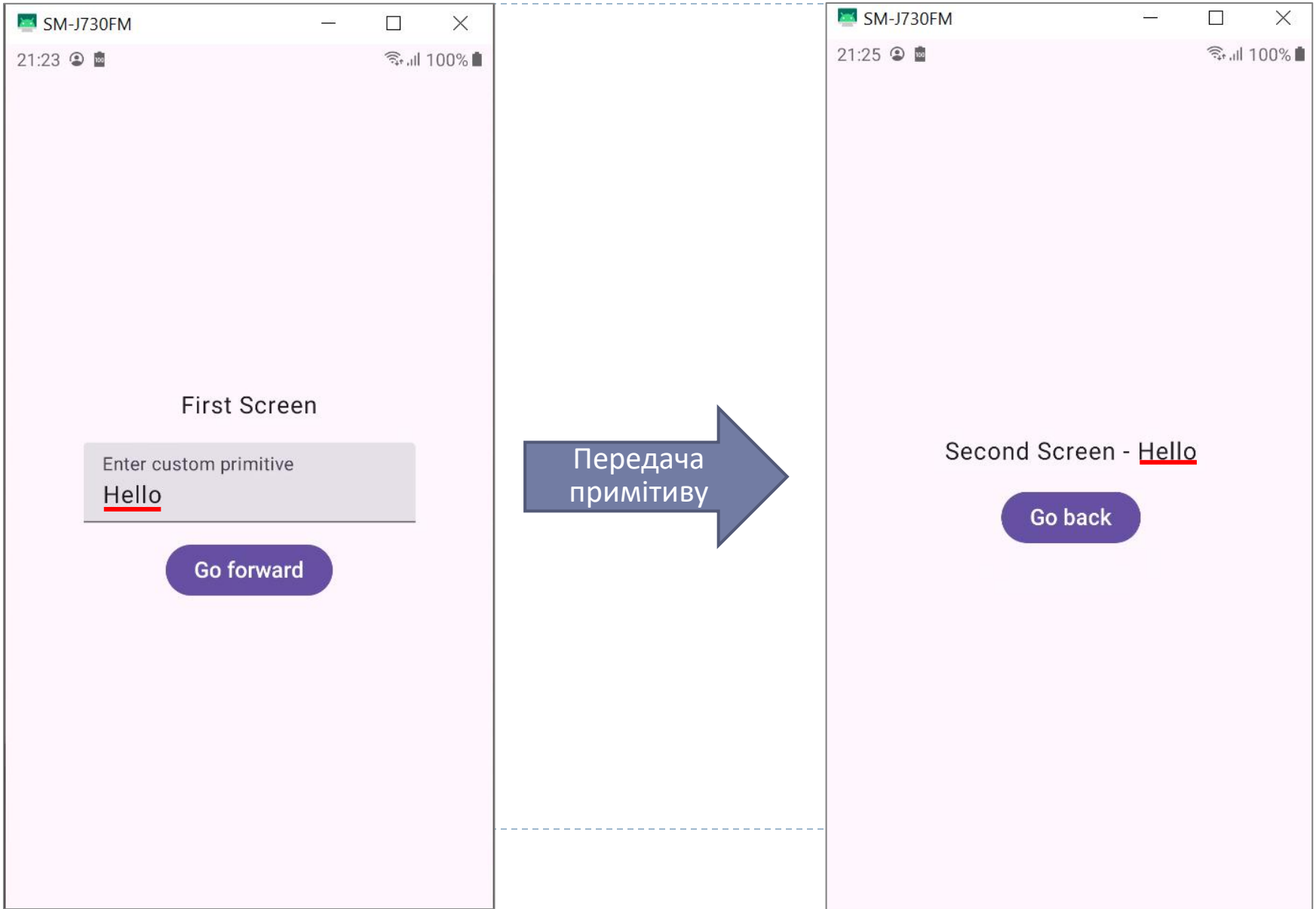
Навігація з передачею примітивного параметра

```
...
        composable<Routes.SecondScreen> { backStackEntry ->
            val route = backStackEntry.toRoute<Routes.SecondScreen>()
            SecondScreen(
                customPrimitive = route.customPrimitive,
                onNavigateBack = {
                    navController. popBackStack()
                }
            )
        }
    }
}
```

Отримуємо маршрут для поточного призначення з запису стеку navCntroller-a
і передаємо його поле як параметр другому екрану

```
package ua.edu.znu.geoquizcomposeedu
import ...
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
            NavAppPassPrimitive(innerPadding)
...
```

Навігація з передачею примітивного параметра



Навігація з передачею параметра користувацького типу - Data

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

import ...

@Serializable
data class Subject(
    val id: String = UUID.randomUUID().toString(),
    val text: String,
    val value: Boolean
)
```

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

object SubjectRepository {
    val subjects = mutableListOf(
        Subject(text = "Subject 1", value = true),
        Subject(text = "Subject 2", value = false),
        Subject(text = "Subject 3", value = true),
        Subject(text = "Subject 4", value = false)
    )
}
```



Навігація з передачею параметра користувацького типу - UI

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

import ...

@Composable
fun FirstScreen(
    onListItemClick: (Subject) -> Unit
) {
    LazyColumn(modifier = Modifier.fillMaxSize()) {
        items(
            items = SubjectRepository.subjects,
            key = { subject -> subject.id }
        ) { subject ->
            Text(
                text = "${subject.text} - ${subject.value}",
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp)
                    .clickable {
                        onListItemClick(subject)
                    },
        )      }   }}
```

callback для навігації на другий екран приймає параметр

Навігація з передачею параметра користувацького типу - UI

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

import ...

@Composable
fun SecondScreen(
    subject: Subject,
    onNavigateBack: () -> Unit
) {
    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.fillMaxSize()
) {
        Text(text = "${subject.text} - ${subject.value}")
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = onNavigateBack) {
            Text("Go Back")
        }
    }
}
```

параметр передається першим екраном

Навігація з передачею параметра користувацького типу - Routes

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation.nav

import ...

@Serializable
sealed class Routes {

    @Serializable
    data object FirstScreen : Routes()

    @Serializable
    data class SecondScreen(val subject: Subject) : Routes()
}
```

Використовується клас, оскільки маршрут передає параметр



Навігація з передачею параметра користувацького типу

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation

import ...

@Composable
fun NavAppPassCustomType(innerPadding: PaddingValues) {
    val navController = rememberNavController()
    NavHost(
        navController = navController,
        startDestination = Routes.FirstScreen,
        modifier = Modifier.padding(innerPadding)
    ) {
        composable<Routes.FirstScreen> {
            FirstScreen(
                onListItemClick = {
                        subject ->
                    navController.navigate(Routes.SecondScreen(subject))
                }
            )
        }
```

Переходить на другий екран з передачею параметра та додає другий екран до стеку navController-а

..

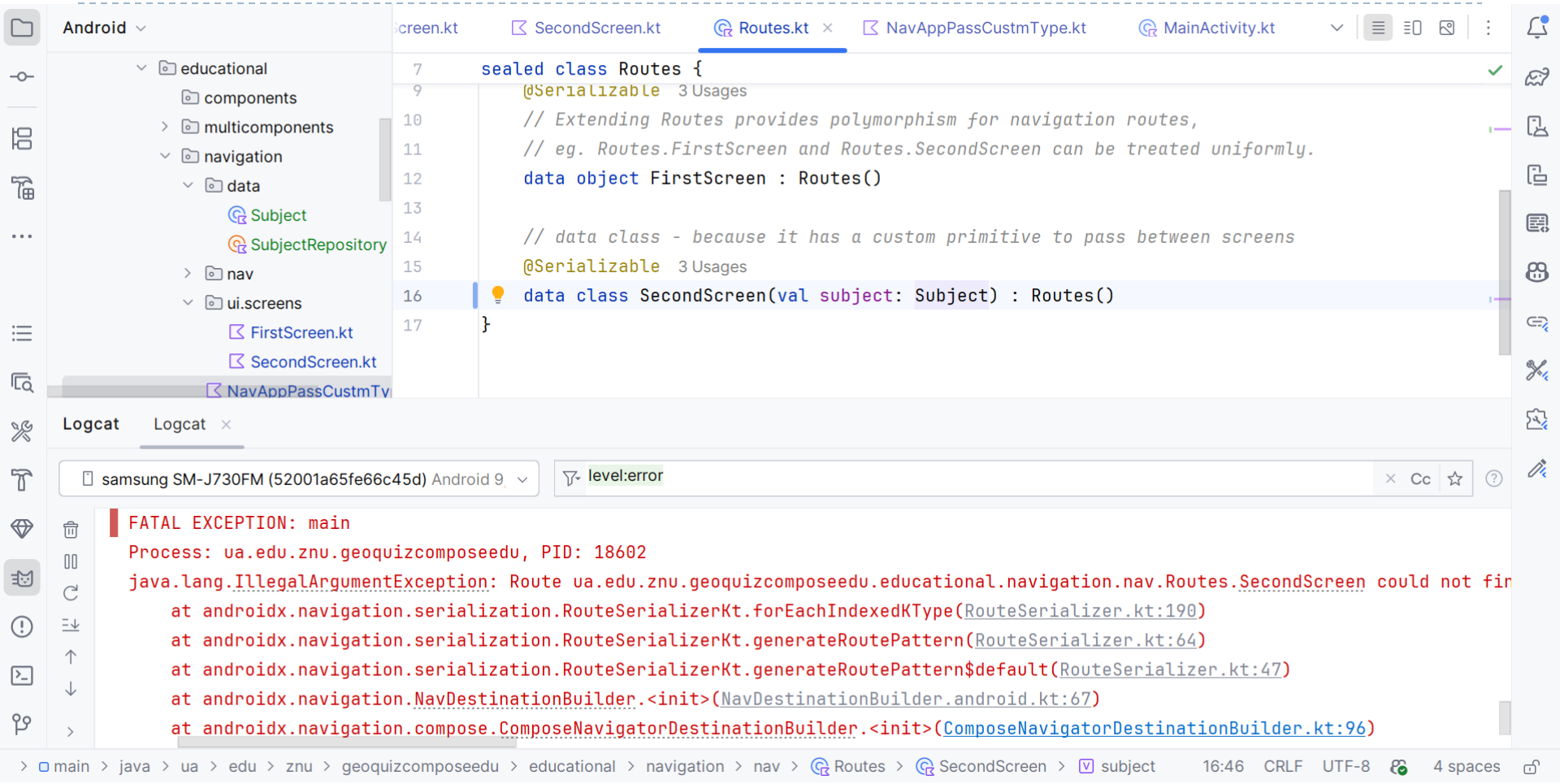
Навігація з передачею параметра користувацького типу

```
..
        composable<Routes.SecondScreen> { backStackEntry ->
            val route = backStackEntry.toRoute<Routes.SecondScreen>()
            SecondScreen(
                subject = route.subject,
                onNavigateBack = {
                    navController. popBackStack()
                }
            )
        }
    }
}
```

Отримуємо маршрут для поточного призначення з запису стеку navCntroller-a
і передаємо його поле як параметр другому екрану

```
package ua.edu.znu.geoquizcomposeedu
import ...
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
            NavAppPassCustmType(innerPadding)
...
```

Навігація з передачею параметра користувацького типу - без визначення NavType



java.lang.IllegalArgumentException: Route SecondScreen could not find any NavType for argument subject of type Subject

- typeMap received was {}

Навігація з передачею параметра користувацького типу - NavType

- Необхідно визначити екземпляр androidx.navigation.NavType<Subject> з реалізацією правил серіалізації та десеріалізації користувацького типу.
- Також ми реалізуємо методи для копіювання/ до та отримання з серіалізованого користувацького типу в Bundle. Екземпляр NavType <Question> буде використовуватися compose внутрішньо для його введення в екземпляр Bundle та подальшого його отримання.
- Існують вбудовані NavTypes для примітивних типів, таких як Int, Long, Float, Boolean, String, посилань на ресурси, користувацьких енумераторів.
- Інші типи, що є Serializable або Parcelable, підтримують тільки значення за-замовчуванням "@null".



Навігація з передачею параметра користувацького типу - NavType

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation.nav

import ...

object SubjectNavType {
    val subjectType = object : NavType<Subject>(
        isNullableAllowed = false
    ) {
        override fun put(
            bundle: SavedState,
            key: String,
            value: Subject
        ) {
            bundle.putString(key, Json.encodeToString(value))
        }
        override fun get(
            bundle: SavedState,
            key: String
        ): Subject? {
            return Json.decodeFromString(bundle.getString(key) ?: return null)
        }
...
```

Сінглетон користувацького типу, що автоматично зберігається та отримується з об'єкту bundle типу SavedState при навігації з передачею параметру користувацького типу

Успадковується від androidx.navigation.NavType<T extends Object> - абстрактний клас

Серіалізує об'єкт Subject у рядок та зберігає його у bundle

Отримує серіалізований у рядок об'єкт Subject з bundle та десеріалізує його у об'єкт Subject

Навігація з передачею параметра користувацького типу - NavType

```
...
    override fun parseValue(value: String): Subject {
        return Json.decodeFromString(Uri.decode(value))
    }
    override fun serializeAsValue(value: Subject): String {
        return Uri.encode(Json.encodeToString(value))
    }
}
```

Десеріалізує серіалізований у рядок користувацький тип до об'єкту користувацького типу.

Серіалізує об'єкт користувацького типу у рядок
Перевизначення неабстрактного методу класу android.navigation.NavType:
public actual open fun serializeAsValue(value: T): String = value.*toString*()



Навігація з передачею параметра користувацького типу - NavType

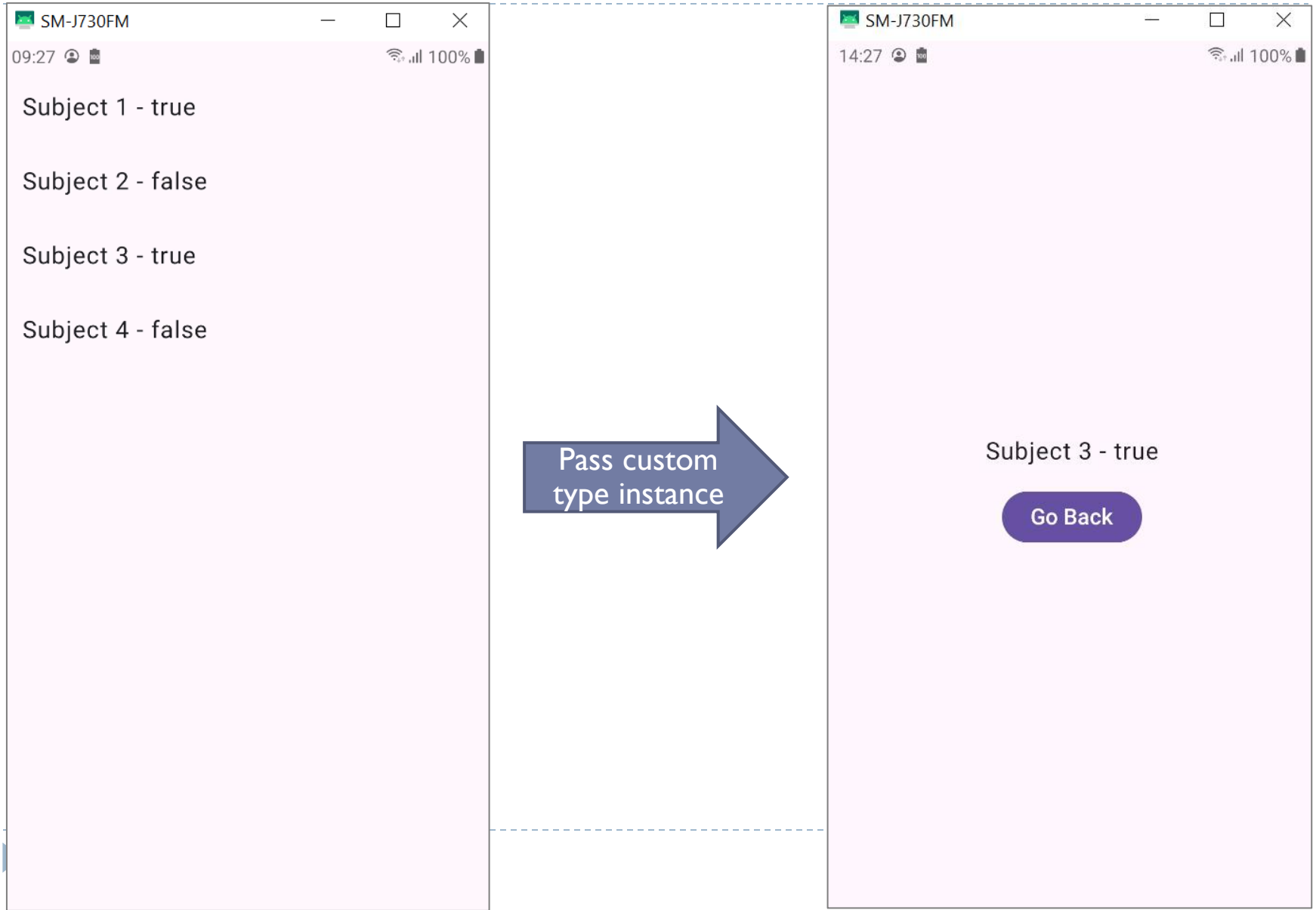
```
package ua.edu.znu.geoquizcomposeedu.educational.navigation

import ...

@Composable
fun NavAppPassCustomType(innerPadding: PaddingValues) {
    val navController = rememberNavController()
    NavHost(
        ...
    ) {
...
        composable<Routes.SecondScreen>(
            typeMap = mapOf(
                typeOf<Subject>() to SubjectNavType.subjectType,
            )
        ) { backStackEntry ->
            val route = backStackEntry.toRoute<Routes.SecondScreen>()
            SecondScreen(
                subject = route.subject,
                onNavigateBack = {
                    navController.popBackStack(
)                   }           )       }   }}
```

Передаємо параметр користувацького типу на другий екран

Навігація з передачею параметра користувацького типу



Навігація з передачею складного Serializable параметра користувацького типу - Data

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

import ...

@Serializable
data class Subject(
    val id: String = UUID.randomUUID().toString(),
    val text: String,
    val value: Boolean
    val category: Category
)


package ua.edu.znu.geoquizcomposeedu.educational.navigation.data

import ...

@Serializable
data class Category(
    val id: String = UUID.randomUUID().toString(),
    val name: String
)
```



Навігація з передачею складного Serializable параметра користувацького типу - Data

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

object SubjectRepository {
    val subjects = mutableListOf(
        Subject(text = "Subject 1", value = true, category = CategoryRepository.categories[0]),
        Subject(text = "Subject 2", value = false, category = CategoryRepository.categories[0]),
        Subject(text = "Subject 3", value = true, category = CategoryRepository.categories[1]),
        Subject(text = "Subject 4", value = false, category = CategoryRepository.categories[2]),
    )
}
```

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

object CategoryRepository {
    val categories = mutableListOf(
        Category(name = "Category 1"),
        Category(name = "Category 2"),
        Category(name = "Category 3")
    )
}
```



Навігація з передачею складного Serializable параметра користувацького типу - UI

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

import ...

@Composable
fun FirstScreen(
    onListItemClick: (Subject) -> Unit
) {
    LazyColumn(
        modifier = Modifier.fillMaxSize()
    ) {
        items(
            items = SubjectRepository.subjects,
            key = { subject -> subject.id }
        ) { subject ->
            Text(
                text = "${subject.text} - ${subject.value} - ${subject.category.name}",
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp)
                    .clickable {
                        onListItemClick(subject)                    },          )       }    }}
```

Навігація з передачею складного Serializable параметра користувацького типу - UI

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

import ...

@Composable
fun SecondScreen(
    subject: Subject,
    onNavigateBack: () -> Unit
) {
    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
            .fillMaxSize()
    ) {
        Text(text = "${subject.text} - ${subject.value} - ${subject.category.name}")
        Spacer(modifier = Modifier.height(16.dp))
        Button(
            onNavigateBack
        ) {
            Text("Go Back")
    }   }}
```

Навігація з передачею складного Serializable параметра користувацького типу - Routes

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation.nav

import ...

@Serializable
sealed class Routes {

    @Serializable
    data object FirstScreen : Routes()

    @Serializable
    data class SecondScreen(val subject: Subject) : Routes()
}
```

Використовується клас, оскільки маршрут передає параметр



Навігація з передачею складного Serializable параметра користувацького типу - NavType

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation.nav

import ...

object SubjectNavType {
    val subjectType = object : NavType<Subject>(
        isNullableAllowed = false
    ) {
        override fun put(
            bundle: SavedState,
            key: String,
            value: Subject
        ) {
            bundle.putString(key, Json.encodeToString(value))
        }
        override fun get(
            bundle: SavedState,
            key: String
        ): Subject? {
            return Json.decodeFromString(bundle.getString(key) ?: return null)
        }
...
```

Сінглетон користувацького типу, що автоматично зберігається та отримується з об'єкту bundle типу SavedState при навігації з передачею параметру користувацького типу

Успадковується від androidx.navigation.NavType<T extends Object> - абстрактний клас

Серіалізує об'єкт Subject у рядок та зберігає його у bundle

Отримує серіалізований у рядок об'єкт Subject з bundle та десеріалізує його у об'єкт Subject

SubjectNavType серіалізує/десериалізує Subject включно з його Category, яка повинна мати анотацію @Serializable

Навігація з передачею параметра складного Serializable користувацького типу - NavType

```
...
    override fun parseValue(value: String): Subject {
        return Json.decodeFromString(Uri.decode(value))
    }
    override fun serializeAsValue(value: Subject): String {
        return Uri.encode(Json.encodeToString(value))
    }
}
```

Десеріалізує серіалізований у рядок користувацький тип до об'єкту користувацького типу.

Серіалізує об'єкт користувацького типу у рядок
Перевизначення неабстрактного методу класу android.navigation.NavType:
public actual open fun serializeAsValue(value: T): String = value.*toString*()



Навігація з передачею параметра складного Serializable користувацького типу

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation

import ...

@Composable
fun NavAppPassCustomType(innerPadding: PaddingValues) {
    val navController = rememberNavController()
    NavHost(
        navController = navController,
        startDestination = Routes.FirstScreen,
        modifier = Modifier.padding(innerPadding)
    ) {
        composable<Routes.FirstScreen> {
            FirstScreen(
                onListItemClick = { subject ->
                    val durationMs = measureTimeMillis {
                        navController.navigate(Routes.SecondScreen(subject))
                    }
                    Log.d("NavTiming", "navigate took $durationMs ms")
                }
            )
    } ...
```

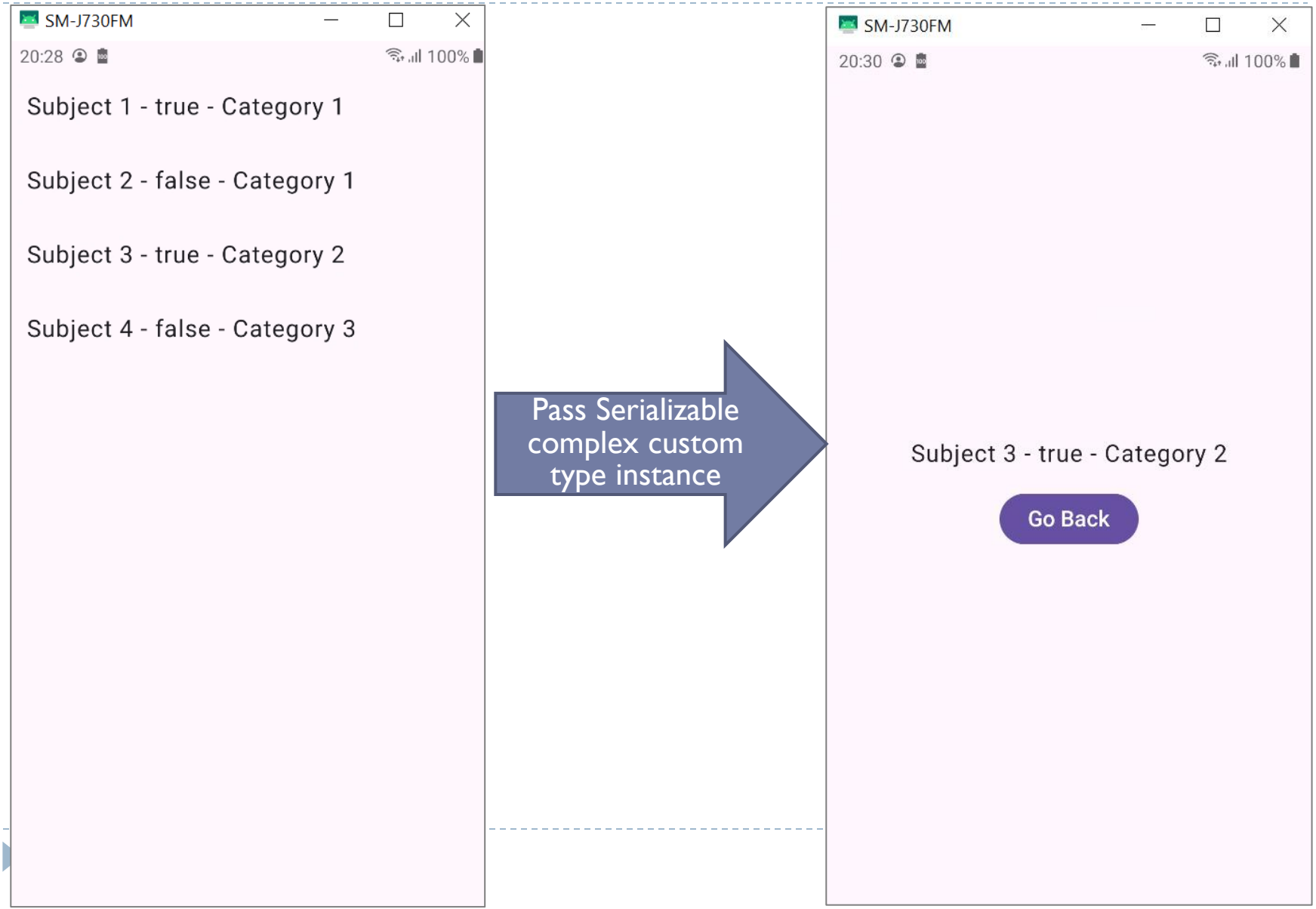
Навігація з передачею параметра складного Serializable користувацького типу

```
...
        composable<Routes.SecondScreen>(
            typeMap = mapOf(
                typeOf<Subject>() to SubjectNavType.subjectType,
            )
        ) { backStackEntry ->
            // unpacking the back stack entry - current navigation destination
            // to obtain the route
            val route = backStackEntry.toRoute<Routes.SecondScreen>()
            SecondScreen(
                subject = route.subject,
                onNavigateBack = {
                    navController.popBackStack()
                }
            )
        }
    }
}
```

Передаємо параметр користувацького типу на другий екран



Навігація з передачею параметра складного Serializable користувацького типу



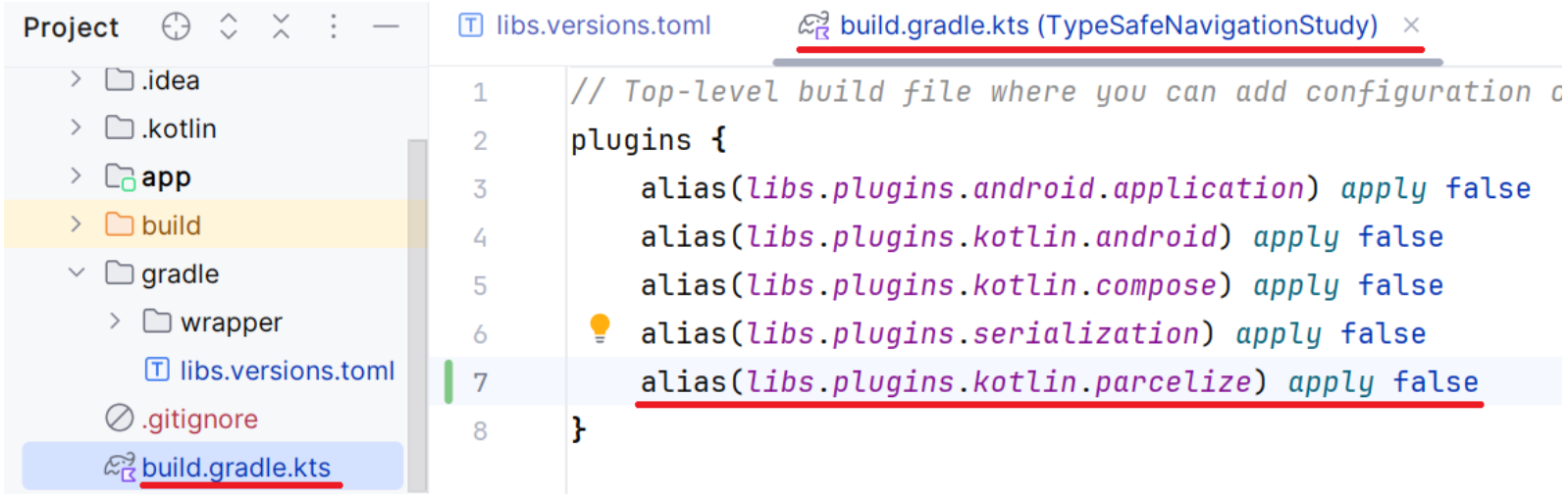
Serializable vs Parcelable об'єкти, що передаються навігацією як аргументи

- **Serializable** — це стандартний інтерфейс Java, який дозволяє серіалізувати об'єкт, тобто перетворити його на потік байтів і зберегти у файлі, передати через мережу або передавати між компонентами Android (Activities, Fragments, Bundle, Composable) (зазвичай як серіалізований рядок).
- **Parcelable** — це специфічний Android інтерфейс, який дозволяє передавати об'єкт як параметр від одного компонента Android до іншого. Це ефективніший метод порівняно із серіалізацією, оскільки він не вимагає перетворення об'єкта на потік байтів (або у рядок для Android). Коли об'єкт передається за допомогою Parcelable, він передається безпосередньо від одного компонента до іншого (наприклад, від Composable до Bundle і навпаки).



Add plugin org.jetbrains.kotlin.plugin.parcelize







Навігація з передачею складного Parcelable параметра користувацького типу - Data

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

import ...

@Serializable
@Parcelize
data class Subject(
    val id: String = UUID.randomUUID().toString(),
    val text: String,
    val value: Boolean
    val category: Category
): Parcelable
```

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

import ...

@Serializable
@Parcelize
data class Category(
    val id: String = UUID.randomUUID().toString(),
    val name: String
) : Parcelable
```

Навігація з передачею складного Parcelable параметра користувацького типу - Routes

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation.nav

import ...

@Serializable
sealed class Routes {

    @Serializable
    data object FirstScreen : Routes()

    @Serializable
    data class SecondScreen(val subject: Subject) : Routes()
}
```

Використовується клас, оскільки маршрут передає параметр



Навігація з передачею складного Parcelable параметра користувацького типу - NavType

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation.nav

import ...

private const val TAG = "SubjectNavType"

object SubjectNavType {
    val subjectType = object : NavType<Subject>(
        isNullableAllowed = false
    ) {
        override fun put(
            bundle: SavedState,
            key: String,
            value: Subject
        ) {
            Log.d(TAG, "put: bundle = $bundle, key = $key, value = $value")
            bundle.putParcelable(key, value)
        }

        override fun get(
            bundle: SavedState,
            key: String
        ): Subject? {
...
```

Вставляє значення Parcelable у Bundle (без серіалізації JSON), замінюючи будь-яке існуюче значення для заданого ключа. Як ключ, так і значення можуть бути null.

Повертає значення, пов'язане з заданим ключем (без JSON-серіалізації), або null, якщо:
- Для заданого ключа не існує відповідності потрібного типу.
- З ключем явно пов'язане значення null.
- Об'єкт не є типу clazz.

Навігація з передачею складного Parcelable параметра користувацького типу - NavType

```
...
        Log.d(TAG, "get: bundle = $bundle, key = $key")
        return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            bundle.getParcelable(key, Subject::class.java)
        } else {
            @Suppress("DEPRECATION") // for backwards compatibility
            bundle.getParcelable(key)
        }
    }

    override fun parseValue(value: String): Subject {
        Log.d(TAG, "parseValue: value = $value")
        return Json.decodeFromString(value)
    }

    override fun serializeAsValue(value: Subject): String {
        Log.d(TAG, "serializeAsValue: value = $value")
        return Json.encodeToString(value)
    }
}
}
```

Для зворотньої сумісності

Декодує та десеріалізує заданий рядок JSON до значення типу Subject

Серіалізує значення типу Subject в еквівалентний JSON

Навігація з передачею параметра складного Parcelable користувацького типу

```
package ua.edu.znu.geoquizcomposeedu.educational.navigation

import ...

@Composable
fun NavAppPassCustomType(innerPadding: PaddingValues) {
    val navController = rememberNavController()
    NavHost(
        navController = navController,
        startDestination = Routes.FirstScreen,
        modifier = Modifier.padding(innerPadding)
    ) {
        composable<Routes.FirstScreen> {
            FirstScreen(
                onListItemClick = { subject ->
                    val durationMs = measureTimeMillis {
                        navController.navigate(Routes.SecondScreen(subject))
                    }
                    Log.d("NavTiming", "navigate took $durationMs ms")
                }
            )
    }}...
```

Навігація з передачею параметра складного Parcelable користувацького типу

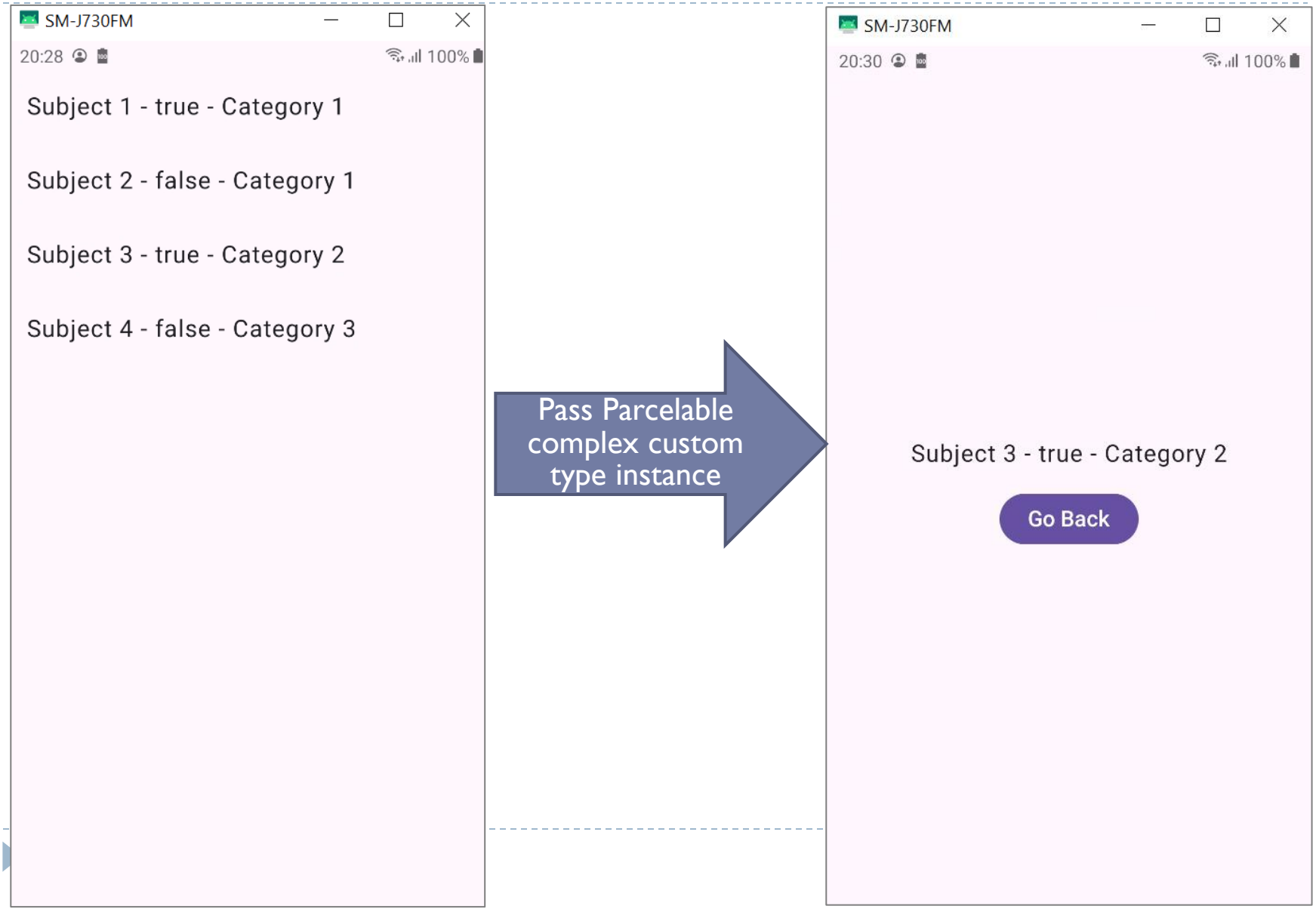
```
...
        composable<Routes.SecondScreen>(
            typeMap = mapOf(
                typeOf<Subject>() to SubjectNavType.subjectType,
            )
        ) { backStackEntry ->
            // unpacking the back stack entry - current navigation destination
            // to obtain the route
            val route = backStackEntry.toRoute<Routes.SecondScreen>()
            SecondScreen(
                subject = route.subject,
                onNavigateBack = {
                    navController.popBackStack()
                }
            )
        }
    }
}
```

Використовуємо користувацький екземпляр NavType з Navigation Graph як елемент Map<KType, NavType<*>> аргументу typeMap NavGraphBuilder.composable

Передаємо параметр користувацького типу на другий екран



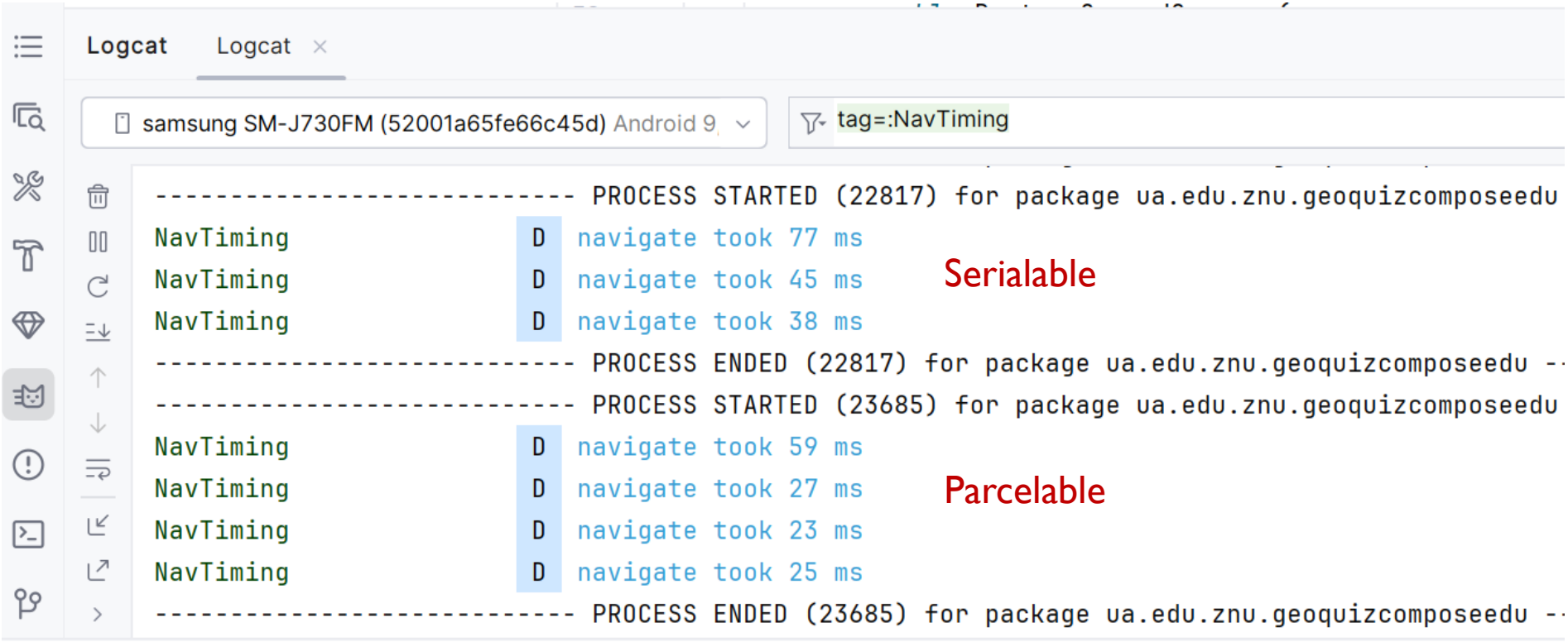
Навігація з передачею параметра складного Parcelable користувацького типу



Навігація з передачею параметра складного Parcelable користувацького типу - Logcat



Навігація з передачею параметра складного Serializable vs Parcelable користувацького типу





Навігація з передачею параметра складного Serializable vs Parcelable користувацького типу

Factor	Serializable	Parcelable
Overview	Serializable is the standard Java interface for persistence.	Parcelable is the Android-specific interface for persistence.
Serialization	Objects are serialized using the Java Serialization API.	Objects are serialized using the Android Parcelable API.
Memory Usage	Serializable objects are stored in memory and can be retrieved quickly.	Parcelable objects are stored in an Android application bundle and require more time to access.
Speed	Serializable is slower than Parcelable.	Parcelable is faster than Serializable.
Size	Serializable objects are larger than Parcelable objects.	Parcelable objects are smaller than Serializable objects.
Implementation	Serializable objects are implemented by implementing the Serializable interface.	Parcelable objects are implemented by extending the Parcelable class.

Навігація з передачею параметра складного Serializable vs Parcelable користувацького типу

Factor	Serializable	Parcelable
Hierarchy	Serializable supports class hierarchy.	Parcelable does not support class hierarchy.
Reflection	Serializable objects can be accessed using Java's reflection API.	Parcelable objects cannot be accessed using Java's reflection API.
Thread Safety	Serializable objects are not thread-safe	Parcelable objects are thread-safe



Підготовка до додання навігації до навчального застосунку - QuestionDataSource

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

class QuestionDataSource {
    private val questions = mutableListOf( ... )
    fun getQuestions(): List<Question> = questions
    fun addQuestion(question: Question) {
        if (!questions.contains(question)) {
            questions.add(question)
        }
    }
    fun updateQuestion(updatedQuestion: Question) {
        val index = questions.indexOfFirst { it.id == updatedQuestion.id }
        if (index != -1) {
            questions[index] = updatedQuestion
        }
    }
    fun removeQuestion(question: Question) {
        questions.removeIf { it.id == question.id }
    }
}
```

Додані функції для змін джерела даних

Підготовка до додання навігації до навчального застосунку - QuestionRepositoryImpl

```
package ua.edu.znu.geoquizcomposeedu.data
import ...
class QuestionRepositoryImpl private constructor(private val questionDataSource:
                                                    QuestionDataSource):QuestionRepository {
    companion object {
        @Volatile
        private var INSTANCE: QuestionRepositoryImpl? = null

        fun getInstance(): QuestionRepositoryImpl {
            return INSTANCE ?: synchronized(this) {
                INSTANCE ?: QuestionRepositoryImpl(QuestionDataSource()).also { INSTANCE = it }
            }
        }
    }
    private fun getQuestions(): List<Question> {
        return questionDataSource.getQuestions()
    }
    override fun getQuestionByIndex(index: Int): Question {
        return getQuestions()[index]
    }
    ...
```

Підготовка до додання навігації до навчального застосунку - QuestionRepositoryImpl

```
...
override fun getQuestionBankSize() = getQuestions().size
private val questionsMutableStateFlow: MutableStateFlow<List<Question>> =
                                                    MutableStateFlow(getQuestions())
override fun getQuestionListState(): StateFlow<List<Question>> {
    return questionsMutableStateFlow
}
override fun addQuestion(question: Question) {
    questionDataSource.addQuestion(question)
    questionsMutableStateFlow.update { oldQuestions ->
        if (oldQuestions.contains(question)) oldQuestions else oldQuestions + question
    }
}
override fun updateQuestion(updatedQuestion: Question) {
    questionDataSource.updateQuestion(updatedQuestion)
    questionsMutableStateFlow.update { oldQuestions ->
        oldQuestions.map {
            if (it.id == updatedQuestion.id) updatedQuestion else it
        }
    }
}...
```

Додані оператори для зміни джерела даних

Підготовка до додання навігації до навчального застосунку - QuestionRepositoryImpl

```
...
    override fun removeQuestion(question: Question) {
        questionDataSource.removeQuestion(question)
        questionsMutableStateFlow.update { oldQuestions -> oldQuestions - question }
    }
}
```

Додані оператори для зміни джерела даних



Підготовка до додання навігації до навчального застосунку - MainScreen

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

import ...

private const val TAG = "MainScreen"

data class MainScreenState(
    val currentIndex: Int = 0,
    val questionList: List<Question> = emptyList()
)

@Composable
fun MainScreen(
//    innerPadding: PaddingValues,
    snackbarHostState: SnackbarHostState,
) {
    val questionRepository = QuestionRepositoryImpl.getInstance()
    val mainViewModel: MainViewModel = viewModel(
        factory = ViewModelFactory(MainViewModel::class.java) {
            MainViewModel(questionRepository)
        }
    )
    val context = LocalContext.current      ...
```

Тепер також буде відслідковуватися стан списку питань

Буде переведений до функції NavGeoQuiz, яка викликатиметься з MainActivity замість MainScreen

Для виведення повідомлень у SnackBar

Підготовка до додання навігації до навчального застосунку - MainScreen

```
...
    val mainScreenState by mainViewModel.mainScreenState.collectAsStateWithLifecycle()
    val scrollState = rememberScrollState()
    Column(
        modifier = Modifier
            .fillMaxSize()
            .verticalScroll(scrollState),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text =
mainViewModel.getQuestionByIndex(mainScreenState.currentIndex).questionText,
            textAlign = TextAlign.Center,
            modifier = Modifier
                .padding(16.dp)
        )
...
```

Для скролінгу у горизонтальній орієнтації



Підготовка до додання навігації до навчального застосунку - MainScreen

```
...
    Row(
        modifier = Modifier.size(height = 80.dp, width = 200.dp),
        horizontalArrangement = Arrangement.SpaceEvenly,
        verticalAlignment = Alignment.CenterVertically
    ) {
        val scope = rememberCoroutineScope()
        Button(
            onClick = {
                val messageId = mainViewModel.onAnswerButtonClick(true)
//                  Toast.makeText(context, messageId, Toast.LENGTH_SHORT).show()
                scope.launch {
                    snackbarHostState.showSnackbar(
                        context.getString(messageId)
                    )
                }
            },
        ) {
            Text(stringResource(id = R.string.true_button))
        }
        Spacer(modifier = Modifier.width(16.dp))
...
```

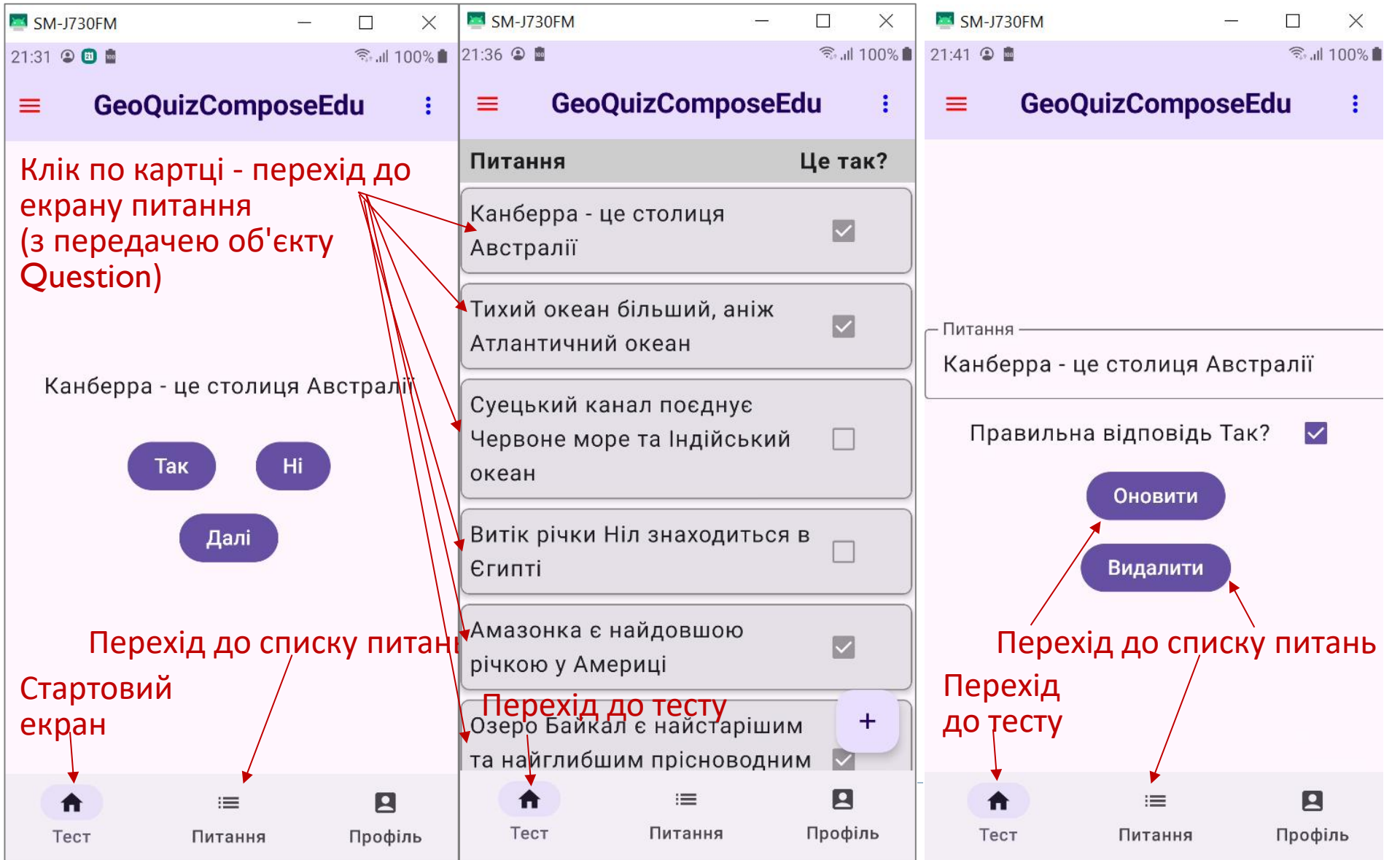
Замінений на SnackBar

Підготовка до додання навігації до навчального застосунку - MainScreen

```
...
        Button(
            onClick = {
                val messageId = mainViewModel.onAnswerButtonClick(false)
//                  Toast.makeText(context, messageId, Toast.LENGTH_SHORT).show()
                scope.launch {
                    snackbarHostState.showSnackbar(
                        context.getString(messageId),
                    )
                }
            },
        ) {
            Text(stringResource(id = R.string.false_button))
        }
    }
    Button(onClick = {
        mainViewModel.onNextQuestionButtonClick()
        Log.d(TAG, "MainScreen: currentIndex = ${mainScreenState.currentIndex}")
    }) {
        Text(stringResource(id = R.string.next_button))
    }
} }
```

Замінений на SnackBar

Навігація у навчальному застосунку



Навігація у навчальному застосунку - Routes

```
package ua.edu.znu.geoquizcomposeedu.nav

import kotlinx.serialization.Serializable
import ua.edu.znu.geoquizcomposeedu.data.Question

@Serializable
sealed class Routes {

    @Serializable
    data object Home : Routes()

    @Serializable
    data object QuestionList : Routes()

    @Serializable
    data class QuestionEdit(val question: Question) : Routes()

    @Serializable
    data object QuestionAdd : Routes()
}
```

Маршрут до екрану тесту

Маршрут до списку питань

Маршрут до екрану питання-режим редагування

Маршрут до екрану питання-режим додання

Навігація у навчальному застосунку - QuestionNavType

```
package ua.edu.znu.geoquizcomposeedu.nav

import ...

object QuestionNavType {
    val questionType = object : NavType<Question>(
        isNullableAllowed = false
    ) {
        override fun put(
            bundle: SavedState,
            key: String,
            value: Question
        ) { bundle.putParcelable(key, value) }

        override fun get(
            bundle: SavedState,
            key: String
        ): Question? {
            return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
                bundle.getParcelable(key, Question::class.java)
            } else {
                @Suppress("DEPRECATION") // for backwards compatibility
                bundle.getParcelable(key)          }       } ...
```

Навігація у навчальному застосунку - QuestionNavType

```
...
    override fun parseValue(value: String): Question {
        Log.d(TAG, "parseValue: value = $value")
        return Json.decodeFromString(value)
    }

    override fun serializeAsValue(value: Question): String {
        Log.d(TAG, "serializeAsValue: value = $value")
        return Json.encodeToString(value)
    }
  }
}
```



Навігація у навчальному застосунку - NavGeoQuiz

```
package ua.edu.znu.geoquizcomposeedu.nav

import ...

@Composable
fun NavPassQuestion(
    innerPadding: PaddingValues,
    snackbarHostState: SnackbarHostState,
    navController: NavHostController
) {
    NavHost(
        navController = navController,
        startDestination = Routes.Home,
        modifier = Modifier.padding(innerPadding)
    ) {
        composable<Routes.Home> {
            MainScreen(snackbarHostState)
        }
...
```

Визначення стартового екрану

Побудова навігаційного графу

Перехід до екрану тесту



Навігація у навчальному застосунку - NavGeoQuiz

```
...
composable<Routes.QuestionList> {
    QuestionListScreen(
        navController,
        onEditQuestionClick = { question ->
            navController.navigate(Routes.QuestionEdit(question))
        }
    )
}
composable<Routes.QuestionEdit>(
    typeMap = mapOf(
        typeOf<Question>() to QuestionNavType.questionType,
    )
) { backStackEntry ->
    val route = backStackEntry.toRoute<Routes.QuestionEdit>()
    QuestionScreen(
        initialQuestion = route.question,
        buttonTextRes = R.string.update_question,
        onNavigateBack = {
            navController.popBackStack()
        }        )      } ...
```

Перехід до екрану списку питань

Перехід до екрану питання в режимі редагування

Навігація у навчальному застосунку - NavGeoQuiz

```
...
        composable<Routes.QuestionAdd> {
            QuestionScreen(
                initialQuestion = null,
                buttonTextRes = R.string.add_question,
                onNavigateBack = {
                    navController.popBackStack()
                }
            )
        }
    }
}
```

Перехід до екрану питання в режимі додання питання



Навігація у навчальному застосунку - ViewModels

```
package ua.edu.znu.geoquizcomposeedu.viewmodel

import ...

class QuestionListViewModel(questionRepository: QuestionRepository) : ViewModel() {
    val questionListFlow = questionRepository.getQuestionListState()
}
```

Відстежується тільки стан списку питань

```
package ua.edu.znu.geoquizcomposeedu.viewmodel

import ...

class QuestionViewModel(private val questionRepository: QuestionRepository) :
                                                                    ViewModel() {
    fun onAddQuestionClick(question: Question) = questionRepository.addQuestion(question)
    fun onUpdateQuestionClick(question: Question) =
                                        questionRepository.updateQuestion(question)
    fun onRemoveQuestionClick(question: Question) =
                                        questionRepository.removeQuestion(question)
}
```



Функції модифікації питання та списку питань

Навігація у навчальному застосунку - QuestionListScreen

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

import ...

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun QuestionListScreen(
    navController: NavController,
//    innerPadding: PaddingValues,
    onEditQuestionClick: (Question) -> Unit = {},
) {
    val questionRepository = QuestionRepositoryImpl.getInstance()
    val questionListViewModel: QuestionListViewModel = viewModel(
        factory = ViewModelFactory(QuestionListViewModel::class.java) {
            QuestionListViewModel(questionRepository)
        }
    )
    val questionList: List<Question> by
                            questionListViewModel.questionListFlow.collectAsStateWithLifecycle()
...
```

Перенесено до NavHost

Навігація у навчальному застосунку - QuestionListScreen

```
...
Box(modifier = Modifier.fillMaxSize()) {
    LazyColumn {
        listHeader()
        items(
            items = questionList,
            key = { question -> question.id }
        ) { question ->
            QuestionCard(
                question,
                onEditQuestionClick = {
                    onEditQuestionClick(question)
                }            )        }      }
    FloatingActionButton(
        onClick = { navController.navigate(Routes.QuestionAdd) },
        modifier = Modifier
            .align(Alignment.BottomEnd)
            .padding(16.dp)
    ) {
        Icon(Icons.Filled.Add, contentDescription = stringResource(R.string.add_question))
    }   }} ...
```

Навігація у навчальному застосунку - QuestionListScreen

...

```
@OptIn(ExperimentalFoundationApi::class)
fun LazyListScope.listHeader() {
    stickyHeader {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .background(Color.LightGray)
                .padding(4.dp),
            horizontalArrangement = Arrangement.Start
        ) {
            Text(
                text = stringResource(R.string.question),
                fontWeight = FontWeight.Bold,
                modifier = Modifier
                    .fillMaxWidth(0.75f)
                    .padding(start = 8.dp)
            )
            Text(
                text = stringResource(R.string.is_true),
                fontWeight = FontWeight.Bold,
            )        }    }} ...
```

Навігація у навчальному застосунку - QuestionListScreen

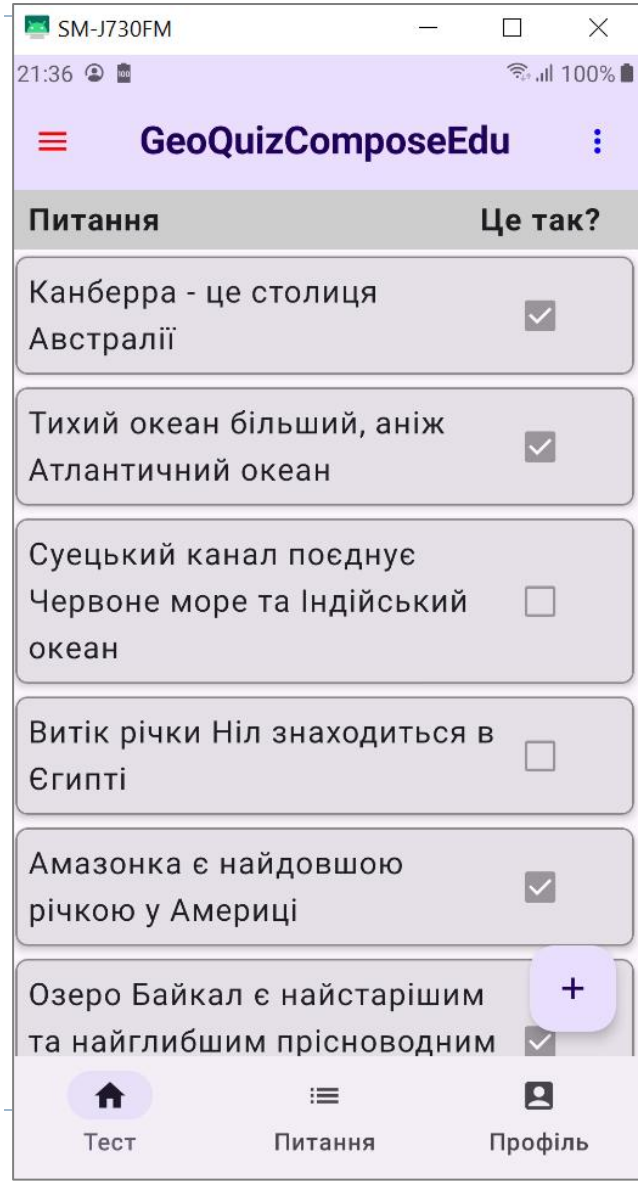
...

```
@Composable
fun QuestionCard(
    question: Question,
    onEditQuestionClick: () -> Unit = {},
) {
    Card(
        shape = RoundedCornerShape(8.dp),
        border = BorderStroke(1.dp, Color.Gray),
        elevation = CardDefaults.cardElevation(4.dp),
        modifier = Modifier
            .fillMaxWidth()
            .padding(4.dp)
            .clickable(
                onClick = onEditQuestionClick
            )
    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier
                .padding(8.dp)
        ) { ...
```

Тепер картка клікабельна

Навігація у навчальному застосунку - QuestionListScreen

```
...
        Text(
            text = question.questionText,
            modifier = Modifier.fillMaxWidth(0.8f)
        )
        Checkbox(
            checked = question.answer,
            enabled = false,
            onCheckedChange = {},
        )
    }
  }
}
```



Навігація у навчальному застосунку - QuestionScreen

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

import ...

@Composable
fun QuestionScreen(
//    innerPadding: PaddingValues,
    initialQuestion: Question?,
    buttonTextRes: Int,
    onNavigateBack: () -> Unit = {}
) {
    val questionRepository = QuestionRepositoryImpl.getInstance()
    val questionViewModel: QuestionViewModel = viewModel(
        factory = ViewModelFactory(QuestionViewModel::class.java) {
            QuestionViewModel(questionRepository)
        }
    )
    var textValue by rememberSaveable { mutableStateOf(initialQuestion?.questionText ?: "") }
    var state by rememberSaveable { mutableStateOf(initialQuestion?.answer ?: false) }
    val localFocusManager = LocalFocusManager.current
...
```

Перенесено до NavHost

Навігація у навчальному застосунку - QuestionScreen

```
...
    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
            .fillMaxSize()
            .pointerInput(Unit) {
                detectTapGestures(onTap = {
                    localFocusManager.clearFocus()
                })
            }
    ) {
        OutlinedTextField(
            value = textValue,
            onValueChange = { updatedText ->
                textValue = updatedText
            },
            label = { Text(text = stringResource(R.string.questions)) },
            modifier = Modifier.fillMaxWidth()
        )
...
```

Для приховання віртуальної клавіатури

Навігація у навчальному застосунку - QuestionScreen

```
...
    Spacer(modifier = Modifier.height(6.dp))
    Row(
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.Center,
        modifier = Modifier.fillMaxWidth()
    ) {
        Text(text = stringResource(R.string.is_answer_true))
        Spacer(modifier = Modifier.padding(8.dp))
        Checkbox(
            checked = state,
            onCheckedChange = { isChecked ->
                state = isChecked
            },
        )
    }
    Spacer(modifier = Modifier.height(8.dp))
...
```



Навігація у навчальному застосунку - QuestionScreen

...

```
Button(onClick = {
    val question = initialQuestion?.copy(
        questionText = textValue,
        answer = state
    ) ?: Question(
        questionText = textValue,
        answer = state
    )
    if (initialQuestion != null) {
        questionViewModel.onUpdateQuestionClick(question)
    } else {
        questionViewModel.onAddQuestionClick(question)
    }
    onNavigateBack()
    Log.d(
        TAG,
        "Question bank: ${questionRepository.getQuestionListState().value}"
    )
}) {
    Text(stringResource(buttonTextRes))
} ...
```

Кнопка оновлення або додання питання

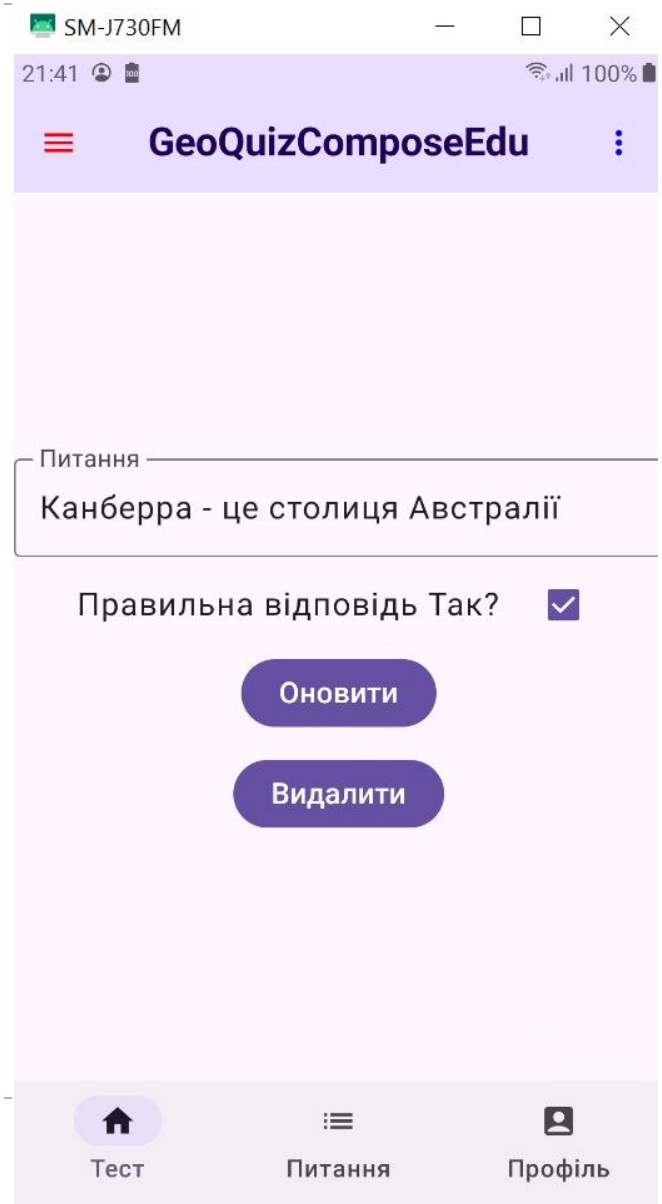
Навігація у навчальному застосунку - QuestionScreen

```
...
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = {
            questionViewModel.onRemoveQuestionClick(initialQuestion!!)
            onNavigateBack()
            Log.d(
                TAG,
                "Question bank: ${questionRepository.getQuestionListState().value}"
            )
        }) {
            Text(stringResource(R.string.remove_question))
        }
    }
}

@Preview(showSystemUi = true)
@Composable
fun QuestionScreenPreview() {
    val sampleQuestion = QuestionRepositoryImpl.getInstance().getQuestionByIndex(0)
    QuestionScreen(sampleQuestion, R.string.update_question)
}
```

Кнопка видалення питання

Навігація у навчальному застосунку - QuestionScreen



Додання другого екрану - QuestionListScreen

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

import ...

private const val TAG = "QuestionListScreen"

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun QuestionListScreen(
    innerPadding: PaddingValues,
) {
    val questionRepository = QuestionRepository.getInstance()

    // Using a generic ViewModelFactory to reduce boilerplate code
    val questionListViewModel: QuestionListViewModel = viewModel(
        factory = ViewModelFactory(QuestionListViewModel::class.java) {
            QuestionListViewModel(questionRepository)
        }
    )
  val questionList: List<Question> by
                    questionListViewModel.questionListFlow.collectAsStateWithLifecycle()

    Log.d(TAG, "QuestionListScreen: questionList.size = ${questionList.size}")
...
```

Отримання сінглетону репозиторія

ViewModel створюється через фабрику, оскільки приймає аргумент

Підписка екрану на зміни стану огорненого MutableStateFlow списку питань

Додання другого екрану - QuestionListScreen

```
...
Box(modifier = Modifier.fillMaxSize()) {
    LazyColumn(
        contentPadding = innerPadding,
    ) {
        listHeader()
        items(
            items = questionList,
            key = { question -> question.id }
        ) { question ->
            QuestionCard(
                question,
                onQuestionDeleted = {
                    questionListViewModel.onRemoveQuestionClick(question)
                }
            )
        }
    }
}
...
```

Користувацька LazyListScope.listHeader() функція - виклик

Додання другого екрану - QuestionListScreen

Material 3 компонент

```
...
        FloatingActionButton(
            onClick = { /* TODO: Navigate to add question screen */ },
            modifier = Modifier
                .align(Alignment.BottomEnd)
                .padding(16.dp)
        ) {
            Icon(Icons.Filled.Add, contentDescription = stringResource(R.string.add_question))
        }
    }  // End of Box
} // End of QuestionListScreen
...
```



Додання другого екрану - QuestionListScreen

```
...
@OptIn(ExperimentalFoundationApi::class)
fun LazyListScope.listHeader() {
    stickyHeader {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .background(Color.LightGray)
                .padding(4.dp),
            horizontalArrangement = Arrangement.Start
        ) {
            Text(
                text = "Question",
                fontWeight = FontWeight.Bold,
                modifier = Modifier
                    .fillMaxWidth(0.65f)
                    .padding(start = 8.dp)
            )
            Text(
                text = "Is true",
                fontWeight = FontWeight.Bold,
            )      }   }} ...
```

Користувацька LazyListScope.listHeader() функція

Додання другого екрану - QuestionListScreen

```
...
@Composable
fun QuestionCard(
    question: Question,
    onQuestionDeleted: () -> Unit = {},
    onEditQuestionClick: () -> Unit = {},
) {
    Card(
        shape = RoundedCornerShape(8.dp),
        border = BorderStroke(1.dp, Color.Gray),
        elevation = CardDefaults.cardElevation(4.dp),
        modifier = Modifier
            .fillMaxWidth()
            .padding(4.dp)
    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier
                .padding(8.dp)
        ) {
...
```

Composable для елемента списку

Додання другого екрану - QuestionListScreen

```
...
    Text(
        text = question.questionText,
        modifier = Modifier.fillMaxWidth(0.65f)
    )
    Checkbox(
        checked = question.answer,
        enabled = false,
        onCheckedChange = {},
    )
    Spacer(modifier = Modifier.width(8.dp))
    IconButton(onClick = onEditQuestionClick) {
        Icon(
            imageVector = Icons.Filled.Edit,
            contentDescription = "Edit question",
        )
    }
    IconButton(
        onClick = onQuestionDeleted
    ) {
...
```

Checkbox не передбачає взаємодії з користувачем

Буде реалізована пізніше

Додання другого екрану - QuestionListScreen

```
...
            Icon(
                imageVector = Icons.Filled.Delete,
                contentDescription = "Edit question",
            )
        }
    }
}

@Preview(showBackground = true)
@Composable
fun QuestionCardPreview() {
    QuestionCard(
        question = Question(
            id = 1,
            questionText = "Канберра - це столиця Австралії",
            answer = true
        )
    )
}
```

Додання другого екрану - QuestionListScreen

