Крос-платформне програмування

ЛАБОРАТОРНА РОБОТА № 7

Розробка REST-клієнта з використанням Kotlin-фреймворку ktor. Створення візуального UI за допомогою Compose Desktop.

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторної роботи
.

Львів

2024

Мета роботи: Ознайомлення з елементами розробки додатків з використанням фреймворку Kotlin Multiplatform, фреймворком ktor для організації клієнт-серверної взаємодії на основі REST, сценарієм збирання Java-проєктів Graddle та бібліотекою візуальних компонентів JetPack Compose.

Завдання до лабораторної роботи.

На основі даних, що надаються вказаним у варіанті REST-сервісом реалізувати невелику ***інформаційну систему*** з візуальним інтерфейсом користувача на основі шаблону Kotlin-проєкту Compose for Desktop. Забезпечити відображення усієї колекції даних, що постачається відповідним REST-джерелом, вивід детальних даних про обраний екземпляр, та пошук дпаних за певним критерієм.

Оформити письмовий звіт про виконання роботи.

Звіт повинен містити:

- титульний аркуш:
- умову задачі відповідного варіанту;
- структуру JSON – коду, що повертається вказаним джерелом, та пояснення її елементів;
- код класів моделей, що використовуються програмою для отримання даних з JSON – коду;
- код та налаштування REST-клієнта для підключення до сервісу та отримання даних;
- код функцій, що відображають вікно програми та його елементи;
- знімки екрану з результатами роботи створеної системи.

Короткі теоретичні відомості

Compose Multiplatform

Варіанти завдань.

1. Створити програму для відображення даних про температуру повітря у 10-ти містах Львівщини протягом минулого тижня за допомогою REST-сервісу https://www.weatherapi.com/. Реалізувати відображення температури за кожен день тижня у кожному з обраних міст, перехід до перегляду усіх погодних параметрів для обраної дати у вказаному місті, а також відображення динаміки зміни в цьому місті вибораного погодного параметра (температура, тиск, кількість опадів) протягом попередніх двох тижнів.

2. Створити інформаційну систему про породи собак за допомогою REST-сервісу https://dog.ceo/dog-api/ . Реалізувати можливість пошуку та перегляду світлин собак обраної породи.

3. За допомогою REST-сервісу https://dictionaryapi.dev/ створити програму для відображення детального опису вказаного користувачем слова англійської мови. Програма повинна запам'ятовувати істроію пошуку та відображати визначення для всіх слів з цього списку. Реалізувати відображення опису та синонімів для вибраного зі історії пошуку слова.

4. На основі сервісу https://restcountries.com/ розробити інформаційну систему про країни Південної Америки. Головний екран програми повинен відображати список усіх країн континенту з короткою інформацію про кожну, наприклад, назву, столиця, зображення прапора, тощо. Реалізувати перехід від списку до детальної інформації про країну, відобразити лише дані, які мають до економіки вибраної країни (грошова одиниця, кількість населеннння і. т. д.).

5. Розробити програму для відображення новин з вибраної тематики на обраній користувачем мові. Реалізувати відображення усіх новин та перехід до перегляду обраної замітки із відображенням графічних зображень та посиланням на джерело інформації. Дані для програми отримати за допомогою REST-сервісу https://newsapi.org/..

6. Створити інформаційну систему про футбол в Аргентині за допомогою REST-сервісу https://www.thesportsdb.com/api.php. Головне вікно повинно відображати перелік футбольних ліг країни, та коротку інформацію про кожну лігу. Реалізувати перегляду списку команд, при виборі ліги на головному вікні, а також перегляд детальної інформації про окрему команду.

7. За допомогою REST-сервісу https://www.thesportsdb.com/api.php розробити інформаційну систему про футбольні матчі української прем'єр-ліги Надати користувачеві можливість вибирати зі списків назви команд-суперниць, та сезон, в якому відбулася гра.

8. На основі сервісу https://restcountries.com/ розробити інформаційну систему про країни Африки. Головний екран програми повинен відображати список усіх країн континенту з короткою інформацію про кожну. Наприклад, назву, столицю, зображення прапора, тощо. Реалізувати перехід від списку країн до детальної інформації про країну, де відобразити лише дані суспільно політичного характеру (незалежність, державна мова і т. п.).

9. Створити програму для пошуку та відображення даних про погоду сьогодні у вказаному користувачем населеному пункті за допомогою REST-сервісу https://www.weatherapi.com/. Пошук здійснюється за географічними координатами населеного пункту, які повинен надати користувач, крім цього, програма повинна запам'ятовувати історію пошуку та відображати дані про погоду за сьогодні в усіх населених пунктах з цього списку. Реалізувати відображення загальних параметрів (температури, тиску, вологості та кількості опадів) за минулі два тижні для обраного зі списку раніше вказаних населених пунктів місця.

10. На основі REST-сервісу https://www.thesportsdb.com/api.php створити інформаційну систему про футбол в Німеччині. Головне вікно програми повинно відображати перелік команд German Bundesliga з короткою інформацією про них, а також інструмент для пошуку інформації матчі між обраними командами ліги у вказаному сезоні.

11. Розробити ІС з інформацією про країни Північної Америки. Відобразити список усіх країн з короткою інформацію про кожну країну. Для отримання даних скористатися REST-сервісом https://restcountries.com/. Реалізувати перехід від списку країн до детальної інформації про вибрану країну. Яку саме інформацію з наданих сервісом даних відображати на головному екрані, а яку в деталях, – обрати самостійно.

12. Створити інформаційну систему про баскетбол в Італії за допомогою REST-сервісу https://www.thesportsdb.com/api.php. Головне вікно повинно відображати перелік ліг з баскетболу в країні, та коротку інформацію про кожну лігу. Реалізувати перегляду списку команд при виборі ліги на головному вікні, а також перегляд списку гравців вибраної команди.

13. На основі сервісу https://restcountries.com/ розробити інформаційну систему про країни Азії. Головний екран програми повинен відображати список усіх країн континенту з короткою інформацію про кожну. Реалізувати перехід від списку країн до детальної інформації про країну, де відобразити лише дані, які відображають стан економіки країни (грошова одиниця, кількість населенння і. т. п.).

14. На основі REST-сервісу https://www.thesportsdb.com/api.php створити інформаційну систему про американський футбол в США. Головне вікно програми повинно відображати перелік команд усіх ліг країни з короткою інформацією про них, а також вікно для пошуку гравця за прізвищем. У випадку успішного пошуку гравця система має відображати коротку інформацію про самого гравця та команду, в якій він зараз грає.

15. Створити програму для відображення даних про погоду у Львові протягом минулих двох тижнів за допомогою REST-сервісу https://www.weatherapi.com/. Реалізувати відображення загальних параметрів (температури, тиску, вологості та кількості опадів) за кожен день протягом зазначеного періоду, перехід до перегляду усіх погодних параметрів для обраної

дати, а також відображення динаміки зміни вказаного погодного параметра протягом попередніх двох тижнів.

16. Користуючись даними REST-сервісу https://www.thesportsdb.com/api.php розробити інформаційну систему про ігри канадської хокейної ліги Canadian WHL. Надати користувачеві можливість вибирати зі списку назви команд-суперниць, та сезон, в якому відбулася гра.

17. За даними REST-сервісу https://restcountries.com/ розробити інформаційну систему про країни Південної Америки. Головний екран програми повинен відображати список усіх країн континенту з короткою інформацію про кожну. Реалізувати перехід від списку країн до детальної інформації про країну, де відобразити лише дані суспільно політичного характеру (незалежність, державна мова і т. п.).

18. Створити інформаційну систему про футбольні клуби Іспанії за допомогою REST-сервісу https://www.thesportsdb.com/api.php . Реалізувати можливість перегляду списку футбольних ліг країни, загальної інформації про окрему команду та списку гравців команди.

19. Розробити ІС з інформацією про країни Азії. Відобразити список усіх європейських країн з короткою інформацією про кожну. Для отримання даних скористатися REST-сервісом https://restcountries.com/. Реалізувати перехід від списку країн до детальної інформації про країну. Яку саме інформацію з наданих сервісом даних відображати на горловному екрані, а яку в детальному описі країни – вибрати самостійно.

20. Створити програму для відображення даних про погоду в столицях країн Європеського Союзу на основі REST-сервісу https://www.weatherapi.com/. Реалізувати відображення загальних параметрів (температури, тиску, вологості та кількості опадів) для всіх міст, перехід до перегляду температури за попередні 10 днів для обраного міста а також відображення усіх даних про погоду в обраному місті за вказану дату в межах попередніх двох тижнів.

21. Створити інформаційну систему про хокей в Канаді за допомогою REST-сервісу https://www.thesportsdb.com/api.php. Головне вікно має відображати перелік хокейних ліг країни, та коротку інформацію про кожну лігу. Реалізувати перегляду списку команд при виборі ліги на головному вікні, а також перегляд детальної інформації про вибрану команду.

22. За допомогою REST-сервісу https://dictionaryapi.dev/ створити програму для пошуку та відображення опису заданого користувачем слова англійської мови. Забезпечити можливість подальшого переходу до визначення слів, використаних при описі вказаного слова (якщо таке слово доступне у вказаному сервісі) та повернення назад до попереднього слова.

23. Створити програму для відображення даних про погоду в обласних центрах України на основі REST-сервісу https://www.weatherapi.com/. Реалізувати відображення загальних параметрів (температури, тиску вологості та опадів) для всіх міст, перехід до перегляду детальних параметрів для обраного

міста а також відображення погоди в обраному місті за вказану дату протягом попередніх двох тижнів.

24. За допомогою REST-сервісу https://restcountries.com/ розробити інформаційну систему про поширення мов у світі. Надати користувачеві можливість вибрати мову з готового списку поширених мов, або ввести назву мови самостійно. Для кожної країни у результаті пошуку відобразити назву, столицю, прапор і т. п.

25. На основі сервісу http://universities.hipolabs.com/search розробити ІС з інформацією про університети у різних країнах світу. Реалізувати перегляд списку країн, університети яких зареєстровані в системі, можливість перегляду списку університетів обраної країни, перегляду інформації про вибраний університет з переходом на офіційний сайт закладу.

26. Розробити ІС з інформацією про країни Европи. Відобразити список усіх европейських країн з короткою інформацією про кожну. Наприклад, назву, столицю, зображення прапора, тощо. Для отримання даних скористатися REST-сервісом https://restcountries.com/. Реалізувати перехід від списку країн до детальної інформації про країну, яку саме інформацію з наданих сервісом даних відображати, – вибрати самостійно.

27. За допомогою REST-сервісу https://www.thesportsdb.com/api.php розробити інформаційну систему про баскетбольні матчі FIBA AmeriCup. Надати користувачеві можливість вибирати зі списку назви команд-суперниць, та сезон, в якому відбулася гра.

28. На основі сервісу https://restcountries.com/ розробити інформаційну систему про країни Азії. Головний екран програми повинен відображати список усіх країн континенту з короткою інформацію про кожну. Наприклад, назву, столицю, зображення прапора, тощо. Реалізувати перехід від списку країн до детальної інформації про країну, де відобразити лише дані суспільно політичного характеру (незалежність, державна мова і т. п.).

29. Створити інформаційну систему про футбольні клуби Англії за допомогою REST-сервісу https://www.thesportsdb.com/api.php . Реалізувати можливість перегляду загальної інформації про команду та списку гравців команди.

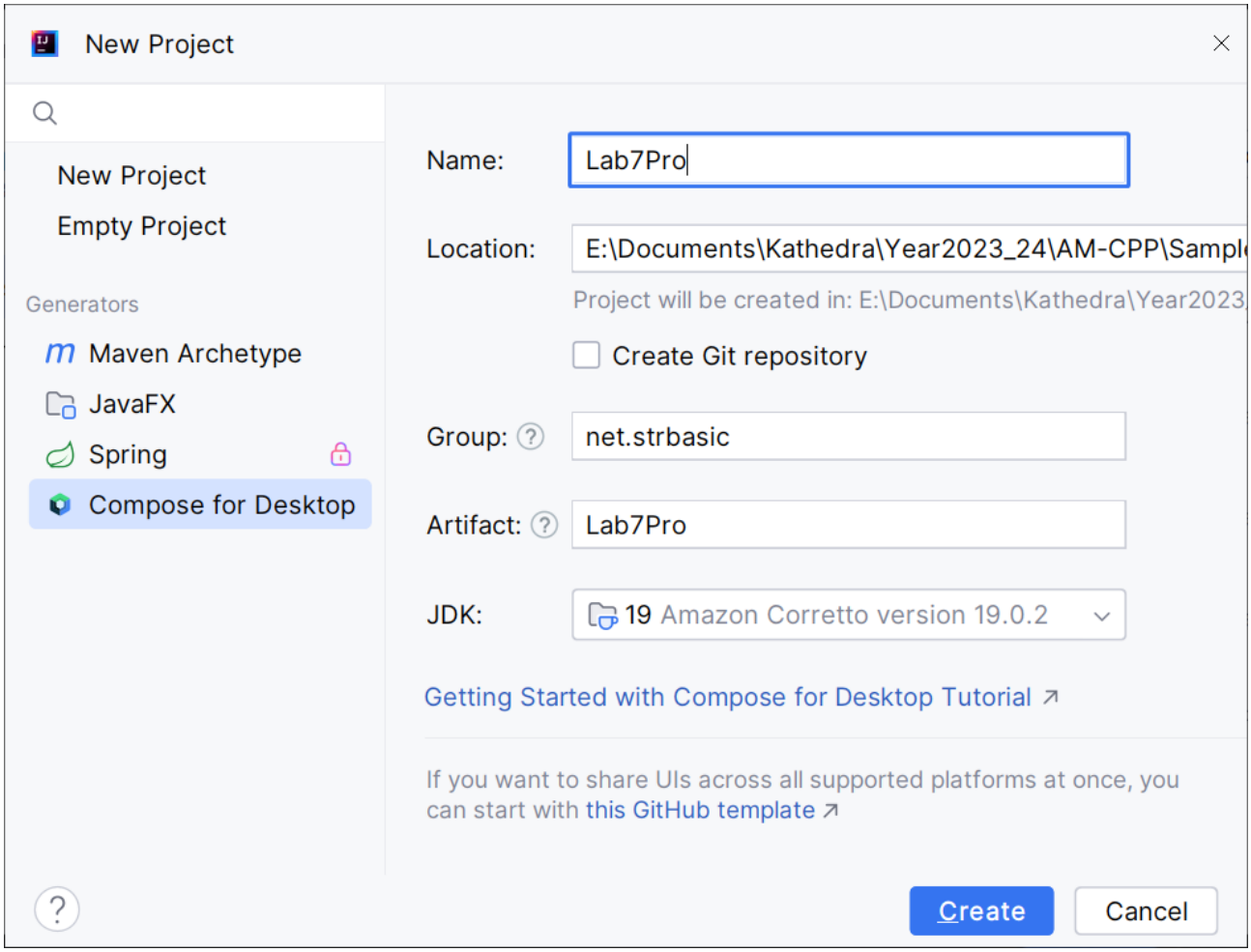
30. Створити програму для відображення новин на основі REST-сервісу https://newsapi.org/. Реалізувати відображення усіх новин, що стосуються вказаної країни, перехід до перегляду обраної замітки і пошук та відображення постів за вказаним фрагментом назви або вмісту.

Приклад розв'язування задачі 30

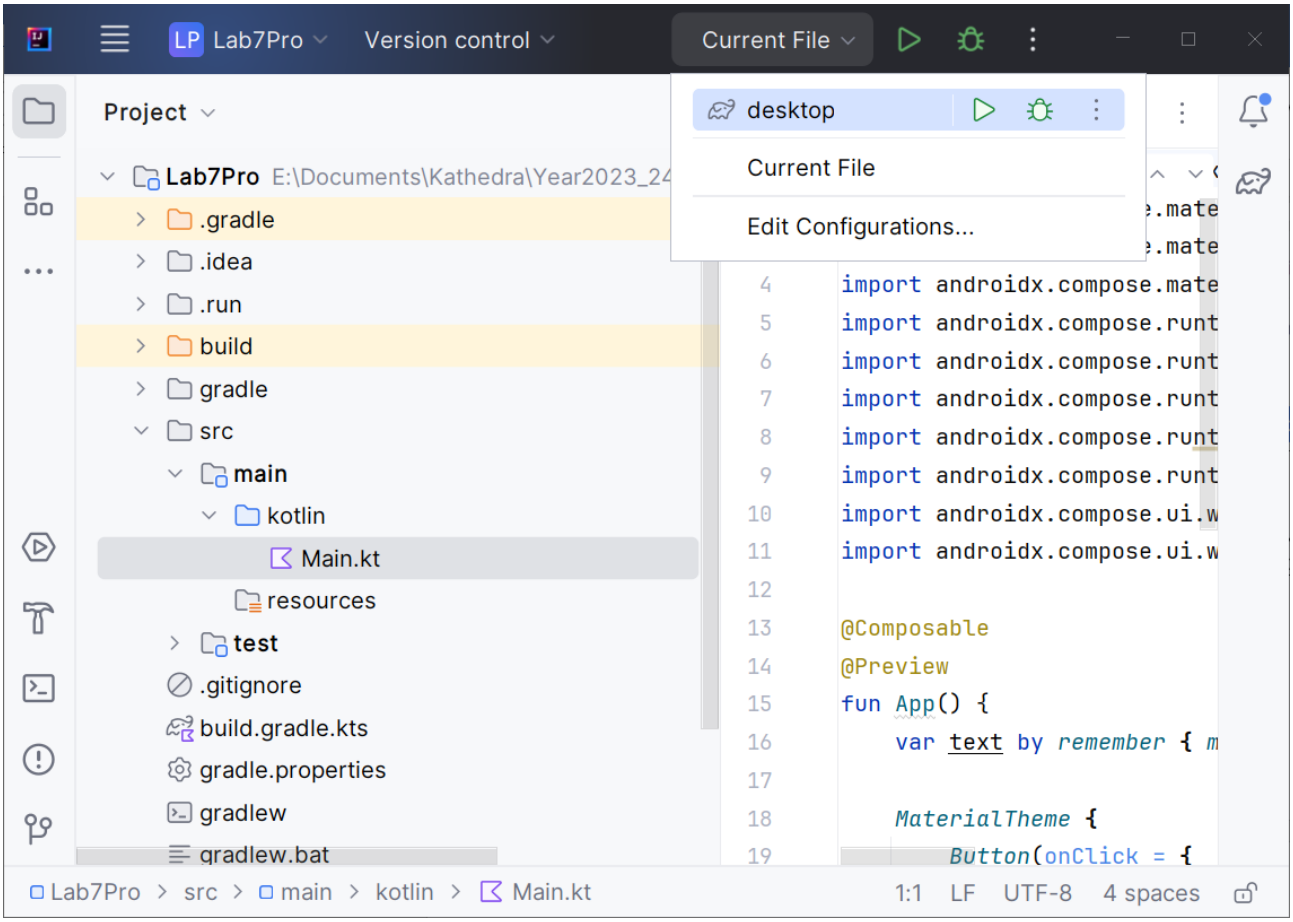
30. Створити програму для відображення новин на основі REST-сервісу https://newsapi.org/. Реалізувати відображення усіх новин, що стосуються вказаної країни, перехід до перегляду обраної замітки і пошук та відображення постів за вказаним фрагментом назви або вмісту.

Частина 1. Створення проєкту

Запускаємо ***IntelliJ IDEA Community Edition***. Створюємо новий проєкт ***Compose for Desktop***. Цей шаблон проекту використовує ***Kotlin*** в якості мови програмування за замовчуванням та сценарій збирання ***Graddle***.



Ортимаємо готовий до запуску проєкт зі зразком коду простого вікна. Перевіримо його працездатність, вибравши desktop в випадаючому меню конфігурації запуску проєкту (див малюнок далі).



Для розробки REST-клієнта використовуватимемо бібліотеки ktor, та gson, їх потрібно додати в розділ залежностей у файлі ***build.gradle.kts***:

```
implementation("io.ktor:ktor-client-cio-jvm:2.3.2")
implementation(
    "io.ktor:ktor-client-content-negotiation:2.3.2")
implementation(
    "io.ktor:ktor-serialization-kotlinx-json:2.3.2")
implementation("com.google.code.gson:gson:2.8.9")
```

Актуальні версії потрібних біліотек та код залежностей для відповідного сценарію завжди можна знайти в центральному репозиторії Maven (mvnrepository.com).

У цьому файлі також потрібно додати плагін для серіалізації

```
kotlin("plugin.serialization") version "1.8.20"
```

у відповідному розділі (***plugins {}***)

Частина 2. Створення моделей для роботи з джерелом даних

Отримуємо API-ключі для роботи з REST-сервісом за вказаною у варіанті роботи адресою (якщо це передбачено сервісом). В коді прикладу ключ зберігається в окремому класі Constants.

Виконавши запити до передбачених сервісом кінцевих точок (endpoints), на основі отриманого JSON в пакеті data.models створюємо класи моделей:

```
package data.model
import com.google.gson.annotations.SerializedName
import kotlinx.serialization.Serializable
@Serializable
data class Article(
    @SerializedName("author")
    val author: String?,
    @SerializedName("content")
    val content: String?,
    @SerializedName("description")
    val description: String?,
    @SerializedName("publishedAt")
    val publishedAt: String,
    @SerializedName("source")
    val source: Source,
    @SerializedName("title")
    val title: String?,
    @SerializedName("url")
    val url: String?,
    @SerializedName("urlToImage")
    val urlToImage: String?
)
```

```
package data.model
import com.google.gson.annotations.SerializedName
import kotlinx.serialization.Serializable
```

```
@Serializable
data class Source(
    @SerializedName("id")
    val id: String?,
    @SerializedName("name")
    val name: String
)
```

```
package data.model
import com.google.gson.annotations.SerializedName
import kotlinx.serialization.Serializable
@Serializable
data class News(
    @SerializedName("articles")
    val articles: List<Article>,
    @SerializedName("status")
    val status: String,
    @SerializedName("totalResults")
    val totalResults: Int
)
```

IntelliJ IDEA дозволяє також створювати Kotlin класи на основі JSON автоматично, за допомогою додаткового плагіну JSON to Kotlin Class.

Для роботи з колекціями створених моделей та JSON-кодом, отроиманим з сервера потрібно реалізувати ktor-клієнт. Збережемо ці налаштування в окремому класі в пакеті data:

```
package data

import data.model.News
import io.ktor.client.*
import io.ktor.client.call.*
```

```
import io.ktor.client.engine.cio.*
import io.ktor.client.plugins.contentnegotiation.*
import io.ktor.client.request.*
import io.ktor.serialization.kotlinx.json.*
import utils.Constants

object NewsApiClient {
    private val client = HttpClient(CIO) {
        install(ContentNegotiation) {
            json()
        }
    }

    suspend fun getTopHeadlines(): News {
        val url = "https://newsapi.org/v2/top-
headlines?country=ua&apiKey=${Constants.API_KEY}"
        return client.get(url).body()
    }

    suspend fun getSearchedNews(searchedText: String): News {
        val url =
"https://newsapi.org/v2/everything?q=$searchedText&apiKey=${C
onstants.API_KEY}"
        return client.get(url).body()
    }
}
```

Частина 3. Створення графічного UI на моделей для роботи з джерелом даних

Для графічного інтерфейсу користувача створимо три Composable-об’єкти (функції):

MainContent – для відображення таблиці (Grid) новин з отриманої в JSON колекції;

SidePanel – бічна панеь міститиме логотип, поле для пошуку та кнопку для відображення усієї колекції;

MainScreen – головне вікно, на якому розмістимо бічну панель та таблицю новин.

```
package ui
import androidx.compose.foundation.layout.Row
import androidx.compose.runtime.*
import data.NewsApiClient
import data.model.Article
import io.ktor.client.plugins.*
import kotlinx.coroutines.launch

@Composable
fun MainScreen() {
  var articles by remember
{ mutableStateOf(emptyList<Article>()) }
  var headerTitle by remember{ mutableStateOf("Headlines") }
  var searchedText by remember { mutableStateOf("") }

  val scope = rememberCoroutineScope()

  LaunchedEffect(searchedText) {
    scope.launch{
      try {
        val newsData = if(searchedText.isNotEmpty()) {
          NewsApiClient.getSearchedNews(searchedText)
```

```
        } else {
          NewsApiClient.getTopHeadlines()
        }
        articles = newsData.articles
      }catch (e: ClientRequestException) {
        println("Error fetching data: ${e.message}")
      }
    }
  }

  Row {
    //SidePanel
    SidePanel(onMenuSelected = {
      headerTitle = it
      searchedText = ""
      articles = emptyList()
    }, onNewsSearched = {_searchedText, _headerTitle ->
      searchedText = _searchedText
      headerTitle = _headerTitle
      articles = emptyList()
    })
    //MainContent
    MainContent(headerTitle, articles)
  }

}
```

```
package ui
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Search
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.input.pointer.pointerHoverIcon
import androidx.compose.ui.res.loadImageBitmap
import androidx.compose.ui.res.useResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import handCursor

@Composable
fun SidePanel(onMenuSelected: (header:String) -> Unit,
onNewsSearched: (searchedText: String,header: String) ->
Unit) {
  var searchedText by remember { mutableStateOf("") }
  Column(
    modifier = Modifier.fillMaxWidth(0.15f).fillMaxHeight()
              .padding(12.dp).background(Color.White),
    horizontalAlignment = Alignment.CenterHorizontally
  ) {
    val bitmap = useResource("logo_image.png") {
      loadImageBitmap(it)
    }
```

```
Image(bitmap,"Logo", modifier = Modifier.width(100.dp))
Spacer(modifier = Modifier.padding(18.dp))
OutlinedTextField(
  modifier = Modifier.fillMaxWidth().height(50.dp),
  singleLine = true,
  placeholder = {
    Text("Search")
  },
  value = searchedText,
  onValueChange = {
    searchedText = it
  },
  trailingIcon = {
    IconButton(
      onClick = {
        onNewsSearched(
            searchedText,"Results for '$searchedText'")
      },
      modifier = Modifier.size(40.dp)
                  .pointerHoverIcon(handCursor())
    ) {
      Icon(
        imageVector = Icons.Default.Search,
        contentDescription = "Search button",
        tint = Color.Black
      )
    }
  }
)
TextButton(
```

```
      onClick = {
        searchedText = ""
        onMenuSelected("Headlines")
      }
    ) {
      Text(
        "Headlines",
        fontWeight = FontWeight.Bold,
        color = Color.Black,
        modifier = Modifier.pointerHoverIcon(handCursor())
      )
    }
  }
}

package ui
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.foundation.lazy.grid.items
import androidx.compose.material.Card
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.input.pointer.pointerHoverIcon
import androidx.compose.ui.layout.ContentScale
```

```
import androidx.compose.ui.res.loadImageBitmap
import androidx.compose.ui.res.useResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import data.model.Article
import handCursor
import loadPicture
import openURL
import java.net.URI

@Composable
fun MainContent(headerTitle: String, articles: List<Article>)
{
  if(articles.isNotEmpty()) {
    Column(
      modifier = Modifier.fillMaxSize()
            .background(Color.White).padding(8.dp)
    ) {
      Text(text = headerTitle, fontSize = 24.sp,
                fontWeight = FontWeight.ExtraBold)
      Spacer(Modifier.padding(4.dp))
      LazyVerticalGrid(
        columns = GridCells.Adaptive(minSize = 300.dp),
        verticalArrangement = Arrangement.spacedBy(4.dp)
      ) {
        items(articles) {
          Card(
            modifier = Modifier.width(400.dp).height(200.dp)
               .padding(4.dp).pointerHoverIcon(handCursor())
```

```
      .clickable {
        if(!it.url.isNullOrEmpty()) {
          openURL(URI(it.url))
        }
      }
  ) {
    Box {
      val bitmap = useResource("no_image.png") {
        loadImageBitmap(it)
      }
      if(it.urlToImage.isNullOrEmpty()) {
        Image(
          bitmap,
          "no image available",
          modifier = Modifier.size(100.dp)
                  .align(Alignment.TopCenter),
          contentScale = ContentScale.Crop
        )
      } else {
        Image(
          loadPicture(it.urlToImage),
          "news thumbnail",
          contentScale = ContentScale.Crop
        )
      }
      Column(
        modifier = Modifier
                  .align(Alignment.BottomStart)
          .background(Color.White).padding(4.dp)
      ) {
        Text(
```

```
                    it.title ?: "",
                    color = Color.Black,
                    fontWeight = FontWeight.Bold
                  )
                  Spacer(modifier = Modifier.padding(2.dp))
                  Text(
                    it.content ?: "",
                    color = Color.Black,
                    fontWeight = FontWeight.ExtraLight,
                    maxLines = 2,
                    overflow = TextOverflow.Ellipsis
                  )
                }
              }
            }
          }
        }
      } else {
        Column(
          modifier = Modifier.fillMaxSize()
                             .background(Color.White),
          verticalArrangement = Arrangement.Center,
          horizontalAlignment = Alignment.CenterHorizontally
        ) {
          Text("Loading...")
        }
      }
    }
```

В головному файлі розмістимо функцію main, та декфлька допоміжних функцій:

```
import androidx.compose.ui.graphics.toComposeImageBitmap
import androidx.compose.ui.input.pointer.PointerIcon
import androidx.compose.ui.window.Window
import androidx.compose.ui.window.application
import ui.MainScreen
import java.awt.Cursor
import java.awt.Desktop
import org.jetbrains.skia.Image
import java.net.URI
import java.net.URL
fun main() = application {
    Window(onCloseRequest = ::exitApplication,
                            title = "YourNews") {
        MainScreen()
    }
}

fun handCursor() = PointerIcon(Cursor
            .getPredefinedCursor(Cursor.HAND_CURSOR))

fun openURL(uri: URI) {
    val desktop = Desktop.getDesktop()
    desktop.browse(uri)
}

fun loadPicture(url: String) =
    Image.makeFromEncoded(URL(url).readBytes())
        .toComposeImageBitmap()
```

Після відлагодження поданого вище коду отримаємо робочий застосунок