**Кросплатформенне програмування**

## **Лабораторна робота №8**

«Розробка API enterprise системи»

*Виконав Заяць Ілля, група ІПЗ-41*

**Мета:**

Відповідно до завдання лабораторної роботи №1, реалізувати виклик стороннього API для створенного сервісу.

Хід роботи

Відповідно до завдання лабораторної роботи №1, реалізуйте виклик стороннього API для вашого сервісу. Використовуйте клієнт Ktor (https://ktor.io/docs/client-create-and-configure.html).

У лабораторній роботі необхідно реалізувати:

- запити із використанням різних HTTP методів (PUT, GET, DELETE, ...)

- запити із використанням різних headers, queryParams тощо

- маппінг для тіла запиту та відповіді відповідно до формату даних, наприклад JSON чи XML

- спільний конфіг для різних ендпоінтів

Вибрати можна будь-який сервіс для інтеграції (або https://httpbin.org/#/ чи http://jsonplaceholder.typicode.com/). Результати виконання роботи завантажити у github. У README.md створити документацію щодо сервісу, який ви інтегрували та які ендпоінти використані (у загальному).

Посилання на гіт-хаб репозиторій:

<https://github.com/db-user/kpLR8>

## *Завдання*

## 1.)- Пояснення структури і роботи, опис домену.

***!!!Оскільки об’єм роботи дуже великий, то в текст були додані заголовки, за допомогою яких можна переміщуватись по документу!!!***

Програма, яка інтегрує JSONPlaceholder в систему управління бібліотекою, включає кілька основних компонентів, які відповідають за керування бібліотечними ресурсами та забезпеченням зручного API для взаємодії з системою. Це зокрема використання стороннього API через клієнт Ktor, що дозволяє отримувати, створювати, оновлювати та видаляти пости.

*1.)- Ключові учасники:*

Читачі (Member) — тепер можуть не лише позичати книги, але й взаємодіяти з JSONPlaceholder API, що дозволяє їм створювати пости та взаємодіяти з іншими функціями API.

Бібліотекар (Librarian) — має можливість не лише керувати книгами в бібліотеці, а й взаємодіяти з API, наприклад, для оновлення постів або видалення їх через HTTP-запити.

Система (Library) — центральний організатор усіх операцій з книгами та постами, а також управління бібліотечними ресурсами через API.

*2.)- Ключові сутності та їх функції:*

Книга (Book) — без змін від початкової версії програми. Програма зберігає книги в бібліотеці, але додано нове поняття — взаємодія з JSONPlaceholder API для роботи з постами. Книга може бути позичена або повернена, а додатково можна здійснювати API запити для публікації постів.

Читач бібліотеки (Member) — цей клас тепер не тільки працює з бібліотекою, а й має функції для взаємодії з JSONPlaceholder API. Читач може створювати пости через API, а також отримувати й оновлювати їх.

Бібліотека (Library) — центральний компонент для керування книжковим фондом, але також забезпечує логіку для взаємодії з API. Програма дозволяє через бібліотеку робити запити до зовнішнього сервісу (JSONPlaceholder).

Пост (Post) — новий тип сутності, що з’явився через інтеграцію API. Пости мають заголовок і текст, і взаємодія з ними відбувається через API-запити.

*3.)- Основні операції:*

Інтеграція з API: Програма тепер використовує HTTP-методи для створення, оновлення, видалення та отримання постів через JSONPlaceholder.

GET /posts — отримання всіх постів.

GET /posts/{id} — отримання посту за ID.

POST /posts — створення нового поста.

PUT /posts/{id} — оновлення поста.

DELETE /posts/{id} — видалення поста.

Операції з книгами: Взаємодія з книгами залишається такою ж, як і в початковій версії — бібліотекар може додавати або видаляти книги, а читач може позичати і повертати їх.

Логування дій читачів: Логування тепер здійснюється не лише для операцій з книгами, а й для взаємодії з API (створення постів, оновлення тощо).

*4.)- Підрахунок та статистика:*

Кількість постів — додано можливість отримувати статистику по постах через API (наприклад, кількість всіх постів або постів за певним ID).

Кількість доступних книг — залишається незмінною, однак додано ще одне джерело інформації через API — кількість постів або інформація про них.

*5.)- Інтерфейси взаємодії:*

Читачі (Members) — можуть не лише взаємодіяти з книгами (позичати та повертати їх), а й виконувати запити до API для створення чи оновлення постів.

Бібліотекарі (Librarians) — мають доступ до розширених функцій: додавання книг у бібліотеку та взаємодія з API для оновлення чи видалення постів.

Система (Library) — як і раніше, займається управлінням всіма ресурсами бібліотеки, однак тепер вона також забезпечує логіку взаємодії з JSONPlaceholder через клієнт Ktor для API-запитів.

**Порівняння з початковою версією програми "Бібліотека":**

*1.)- Додавання стороннього API:*

У порівнянні з початковою версією програми, яка була лише внутрішньою системою для управління книгами, ця версія тепер інтегрує зовнішній API (JSONPlaceholder) для роботи з постами. Це дозволяє не тільки працювати з бібліотечними ресурсами, а й взаємодіяти з віддаленим сервером для отримання та обробки постів.

*2.)- Нова сутність — пост (Post):*

Початкова версія програми працювала лише з книгами та читачами, а тепер додається ще одна сутність — пост, з яким можуть працювати читачі та бібліотекарі через API. Це розширює функціональність системи та додає можливість інтеграції з зовнішніми даними.

*3.)- Оновлення логіки взаємодії:*

Логіка взаємодії тепер включає не лише бібліотечні операції (позичання книг), а й роботу з JSONPlaceholder, зокрема створення, оновлення, видалення постів. Читачі можуть використовувати ці функції через інтегрований API.

*4.)- Кодова структура:*

Раніше програма мала лише клас бібліотеки, читача та книги, тепер вона додала API-інтерфейси для обробки запитів через Ktor-клієнт, що дозволяє працювати з JSON-даними, здійснювати запити та отримувати відповіді через HTTP-протокол.

Таким чином, програма значно розширила свою функціональність, додавши можливість взаємодії з віддаленими сервісами, що дозволяє робити її більш гнучкою та потужною.

## 2.)- Код:

**build.gradle.kts – файл залежностей**

dependencies {

implementation("io.ktor:ktor-client-core:2.3.3")

implementation("io.ktor:ktor-client-cio:2.3.3")

implementation("io.ktor:ktor-client-content-negotiation:2.3.3")

implementation("io.ktor:ktor-serialization-kotlinx-json:2.3.3")

implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.0")

}

**LibraryApiIntegration.kt — Клієнт для роботи з API**

import io.ktor.client.\*

import io.ktor.client.call.\*

import io.ktor.client.plugins.contentnegotiation.\*

import io.ktor.client.request.\*

import io.ktor.http.\*

import io.ktor.serialization.kotlinx.json.\*

import kotlinx.coroutines.runBlocking

import kotlinx.serialization.Serializable

import kotlinx.serialization.json.Json

// ==========================

// Ktor Client конфігурація

// ==========================

val client = HttpClient {

install(ContentNegotiation) {

json(Json {

prettyPrint = true

isLenient = true

ignoreUnknownKeys = true

})

}

}

// ==========================

// Модель для JSONPlaceholder

// ==========================

@Serializable

data class LogPost(

val userId: Int,

val id: Int = 0,

val title: String,

val body: String

)

// ==========================

// Функція для відправки логу в API

// ==========================

suspend fun sendLog(title: String, body: String) {

val log = LogPost(userId = 1, title = title, body = body)

client.post("https://jsonplaceholder.typicode.com/posts") {

contentType(ContentType.Application.Json)

setBody(log)

}

}

**Library.kt — Основні класи бібліотеки**

data class Book(

val title: String,

val author: String,

val isbn: String,

var available: Boolean = true,

var borrower: Member? = null

) {

fun isAvailable(): Boolean = available

fun borrow(member: Member) {

if (available) {

available = false

borrower = member

member.borrow\_book(this)

} else {

println("Book is not available.")

}

}

fun returnBook() {

available = true

borrower?.return\_book(this)

borrower = null

}

}

class Member(

val name: String,

val memberId: String,

val maxBooks: Int = 3

) {

val borrowedBooks: MutableList<Book> = mutableListOf()

fun borrow\_book(book: Book) {

if (can\_borrow()) {

borrowedBooks.add(book)

// Логування дії позичання

runBlocking {

sendLog(

title = "Book Borrowed",

body = "Member $name borrowed '${book.title}'."

)

}

} else {

println("$name can't borrow more books.")

// Логування неуспішного позичання

runBlocking {

sendLog(

title = "Borrow Failed",

body = "Member $name tried to borrow '${book.title}' but exceeded the limit."

)

}

}

}

fun return\_book(book: Book) {

if (borrowedBooks.contains(book)) {

borrowedBooks.remove(book)

// Логування дії повернення книги

runBlocking {

sendLog(

title = "Book Returned",

body = "Member $name returned '${book.title}'."

)

}

} else {

println("$name didn't borrow this book.")

// Логування неуспішного повернення

runBlocking {

sendLog(

title = "Return Failed",

body = "Member $name tried to return '${book.title}', which wasn't borrowed."

)

}

}

}

fun can\_borrow(): Boolean = borrowedBooks.size < maxBooks

}

class Library {

val books: MutableList<Book> = mutableListOf()

val members: MutableList<Member> = mutableListOf()

fun add\_book(book: Book) {

books.add(book)

}

fun remove\_book(book: Book) {

books.remove(book)

}

fun find\_book\_by\_title(title: String): Book? {

return books.find { it.title == title }

}

fun register\_member(member: Member) {

members.add(member)

}

fun unregister\_member(member: Member) {

members.remove(member)

}

fun borrow\_book(member: Member, book: Book) {

if (book.isAvailable() && member.can\_borrow()) {

book.borrow(member)

} else {

println("Cannot borrow book.")

}

}

fun return\_book(member: Member, book: Book) {

book.returnBook()

}

}

class Librarian(

val name: String,

val employeeId: String

) {

fun add\_book\_to\_library(book: Book, library: Library) {

library.add\_book(book)

// Логування додавання книги

runBlocking {

sendLog(

title = "Book Added",

body = "Librarian $name added '${book.title}' by ${book.author}."

)

}

}

fun remove\_book\_from\_library(book: Book, library: Library) {

library.remove\_book(book)

// Логування видалення книги

runBlocking {

sendLog(

title = "Book Removed",

body = "Librarian $name removed '${book.title}'."

)

}

}

fun register\_new\_member(member: Member, library: Library) {

library.register\_member(member)

// Логування реєстрації нового члена

runBlocking {

sendLog(

title = "New Member",

body = "Librarian $name registered member '${member.name}' with ID ${member.memberId}."

)

}

}

}

**Main.kt — Головна програма для запуску**

fun main() {

val library = Library()

val librarian = Librarian("John Doe", "L123")

val member = Member("Alice", "M001")

// Реєстрація члена

librarian.register\_new\_member(member, library)

val book1 = Book("Kotlin Programming", "JetBrains", "123-456-789")

val book2 = Book("Advanced Kotlin", "JetBrains", "987-654-321")

// Додавання книг до бібліотеки

librarian.add\_book\_to\_library(book1, library)

librarian.add\_book\_to\_library(book2, library)

// Позичання книги

library.borrow\_book(member, book1)

// Повернення книги

library.return\_book(member, book1)

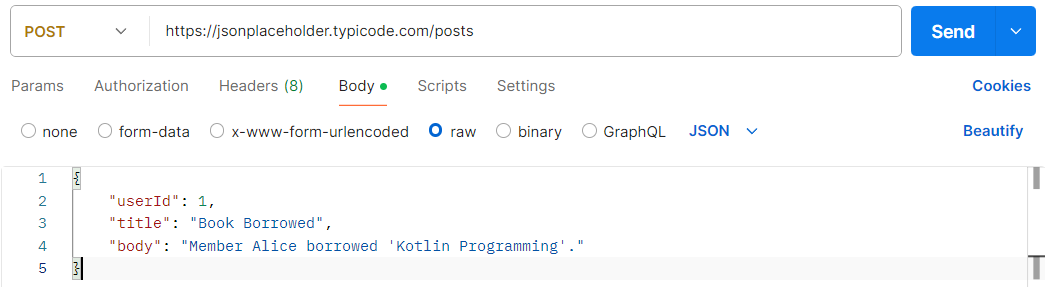
// Видалення книги з бібліотеки

librarian.remove\_book\_from\_library(book2, library)

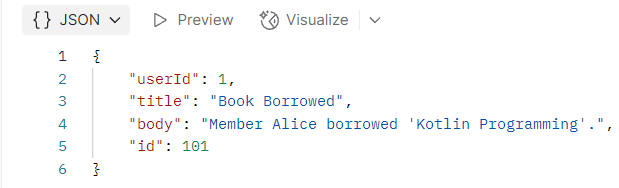
}

## 3.)- Приклад виконання (програма Postman, метод POST, URL: https://jsonplaceholder.typicode.com/posts):

Параметри тестування:



Результат:



## **Висновок:**

Проект "Бібліотека з інтеграцією API JSONPlaceholder" — це розширена система управління бібліотечними ресурсами, яка не тільки реалізує основні операції з книгами та читачами, а й інтегрує зовнішнє API для роботи з постами. Система дозволяє бібліотекарям та читачам виконувати операції не лише з книгами (додавання, видалення, позичання, повернення), а й взаємодіяти з віддаленим сервером для створення та оновлення постів через JSONPlaceholder.

Вона включає кілька типів учасників, таких як читачі, бібліотекарі та бібліотека як центральну точку для взаємодії. Додана можливість отримувати та обробляти дані через API значно розширює функціональність програми, забезпечуючи додаткові можливості для роботи з даними поза межами лише бібліотеки.

Цей проект є гарним прикладом інтеграції сторонніх API в систему управління ресурсами, що дає можливість автоматизувати й спростити багато процесів, одночасно надаючи користувачам доступ до більшої кількості функцій, таких як створення постів та їх оновлення в межах бібліотечної системи.