**Міністерство освіти і науки України**

**Національний університет «Чернігівська політехніка»**

**Навчально-науковий інститут електронних та інформаційних технологій**

**Кафедра інформаційних та комп'ютерних систем**

**Розрахунково-графічна робота**

**з дисципліни «Програмування мобільних пристроїв»**

**на тему:**

“ Інтеграція з віддаленими сервісами за допомогою HTTP-методів ”

Виконав

студент групи КІт-211

Оленченко Ілля Романович

Перевірила

асистент Чорноног О. А.

оцінка \_\_\_\_\_\_\_\_\_ балів

Підпис викладача \_\_\_\_\_\_\_\_\_\_\_

Чернігів 2022 р.

**1.2 Короткі теоретичні відомості**

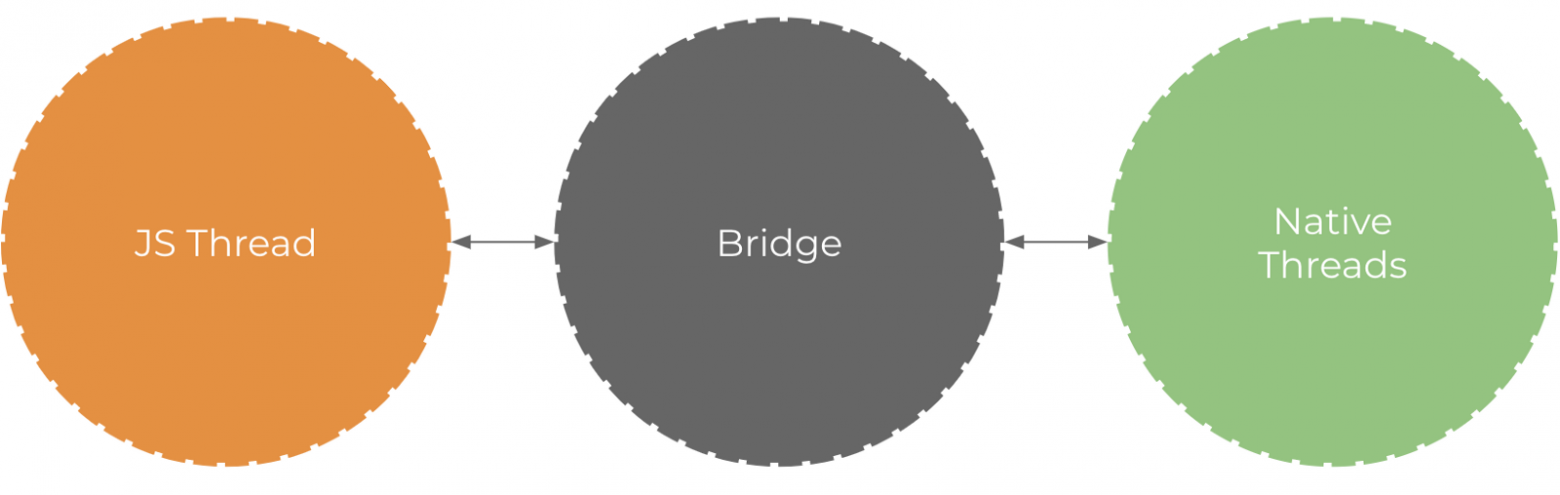
**React Native з Expo**

**React Native** – це кросплатформений фреймворк з відкритим вихідним кодом, написаний на JavaScript. React Native використовується для створення додатків, як для мобільних систем Android та iOS, так і для десктоп, серед яких Windows та macOS.

React Native був вперше випущений Facebook як проект з відкритим вихідним кодом у 2015 році. Усього за пару років він став одним із найкращих рішень, що використовуються для мобільної розробки. Розробка React Native використовується для підтримки деяких провідних мобільних програм у світі, включаючи Instagram, Facebook і Skype.

React Native написано з використанням суміші JavaScript та JXL, спеціального коду розмітки, схожого на XML. Платформа має можливість взаємодіяти як із областями – потоками на основі JavaScript, так і з існуючими потоками нативних програм. Як працює це спілкування? React Native використовує так званий міст. У той час як JavaScript і нативні потоки написані зовсім різними мовами, це функція мосту, яка робить можливим двонаправлений зв'язок.

Відмінна візуалізація концепції мосту:



Це означає, що - якщо ви вже маєте власний додаток для iOS або Android - ви все одно можете використовувати його компоненти або перейти до розробки на React Native.

Різниця між React Native та іншими кросплатформовими рішеннями для розробки полягає в тому, що React Native не відображає WebView у своєму коді. Він працює на реальних, нативних уявленнях та компонентах. Це одна з причин вражаючого успіху React Native.

**Expo** - це набір інструментів, за допомогою якого ви можете написати програму React Native за лічені хвилини. Він включає готові інструменти, такі як конфігурації Android Studio / XCode, управління сертифікатами в Apple & Google і push-сповіщення, і це лише деякі з них.

**GraphQL** — це мова запитів і маніпуляції даними з відкритим кодом для API і середовище виконання для обслуговування запитів з наявних даних.

GraphQL надає підхід розробки веб API і його можна порівнювати і протиставляти REST та іншим архітектурам вебсервісів. Він дозволяє клієнтам визначати структуру потрібних даних і таку саму структуру повертає сервер, таким чином запобігаючи передачі надлишкових даних, але це впливає на дієвість вебкешування результатів запитів. Гнучкість і багатість мови запитів, що може бути не потрібна для простих API. Він складається з системи типів, мови запитів і семантики виконання, статичної валідації і інтроспекції.

GraphQL підтримує читання, писання (змінювання) і підписування на зміни даних.

**StepZen** - це сервер GraphQL з унікальною архітектурою. Це допомагає розробникам швидко створювати API, і, використовуючи декларативні конфігурації, їм потрібно писати менше коду. API працюють у движку GraphQL на основі Golang, розгорнутому в Kubernetes, тому API дуже реагують на потреби додатків.

Ви пишете кілька рядків декларативного коду, ми приносимо мільйон рядків, щоб зробити важкі речі. Незалежно від того, чи є бекенди REST, базами даних або GraphQL, ваша команда може працювати, забезпечуючи гнучкий рівень GraphQL за дні, а не тижні чи місяці. Ваші API є стислими та декларативними та працюють у хмарі StepZen або приватній хмарі з вбудованою оптимізацією продуктивності, вартості та надійност

**Apollo Client** - це найкращий спосіб використовувати GraphQL для створення клієнтських додатків. Клієнт розроблений, щоб допомогти розробнику швидко створити користувальницький інтерфейс, який отримує дані за допомогою GraphQL і може бути використаний з будь-яким інтерфейсом JavaScript.

Зробити вміст книг доступнішим для пошуку в Інтернеті. За допомогою Google Books API та Open Library API додаток може виконувати повнотекстовий пошук і отримувати інформацію про книги, можливості перегляду та доступність електронних книг. Ви також можете керувати своїми особистими книжковими полицями.

**1.3 Хід роботи**

Мобільний додаток Бібліотека дозволяє користувачу шукати та відстежувати книги які вони прочитали.

Для мобільний додаток реалізовано на мові програмування JS з використанням фреймворку Rect Native в сумісності з Expo.

В Google Books API та Open Library API будемо виконувати пошук книг.

StepZen та Apollo використовується для зручності пошуку книжок на вище загаданих ресурсах.

AsyncStorage для зебереження данних в пам’яті телефону.

Для роботи з React Native використовується середовище розробки Visual Studio Code

Робота з 2 різними API в одному додатку є досить складним завданням. Ось чому я використовував StepZen для легкого запиту API GraphQL з додатка.

Було створено безкоштовний обліковий запис на сайті stepzen.com для отримання можливості створення API GraphQL.

На рисунку 1.1 зображено успішний запуск програми StepZen та створення localhost адресси де ми маємо доступ до всього функціоналу програми для створення потрібного нам API GraphQL де було використано API Google та API Open Library.

Готовий API GraphQL прописано в лістингу 1.1.

Пенель StepZen зображено на рисунку 1.2.

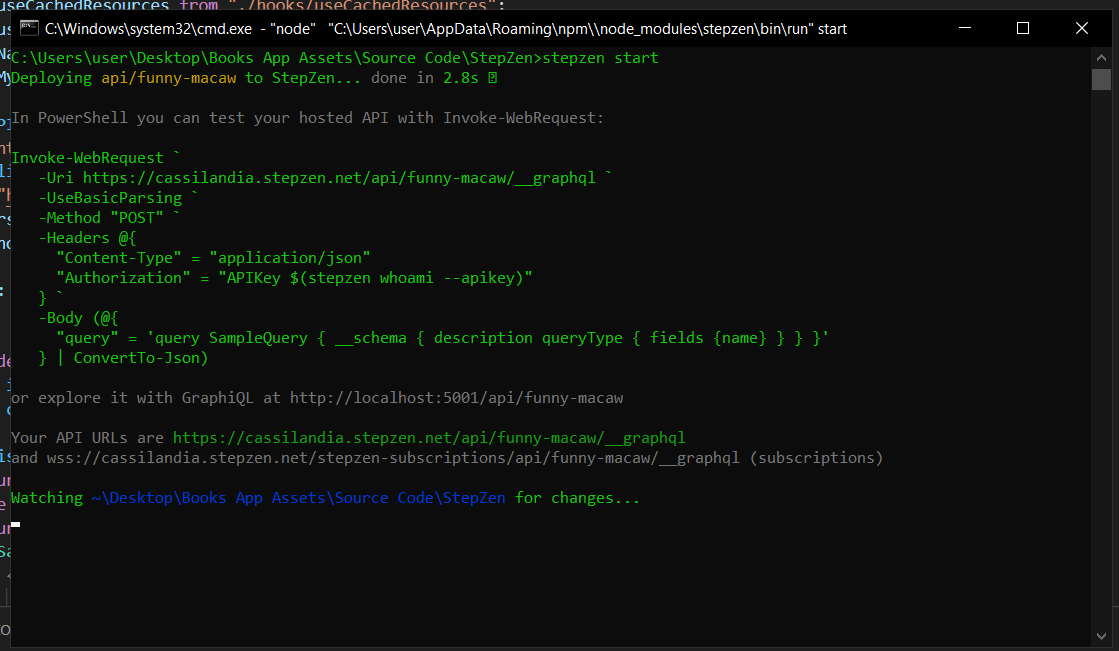


Рисунок 1.1 – Запуск StepZen через консоль

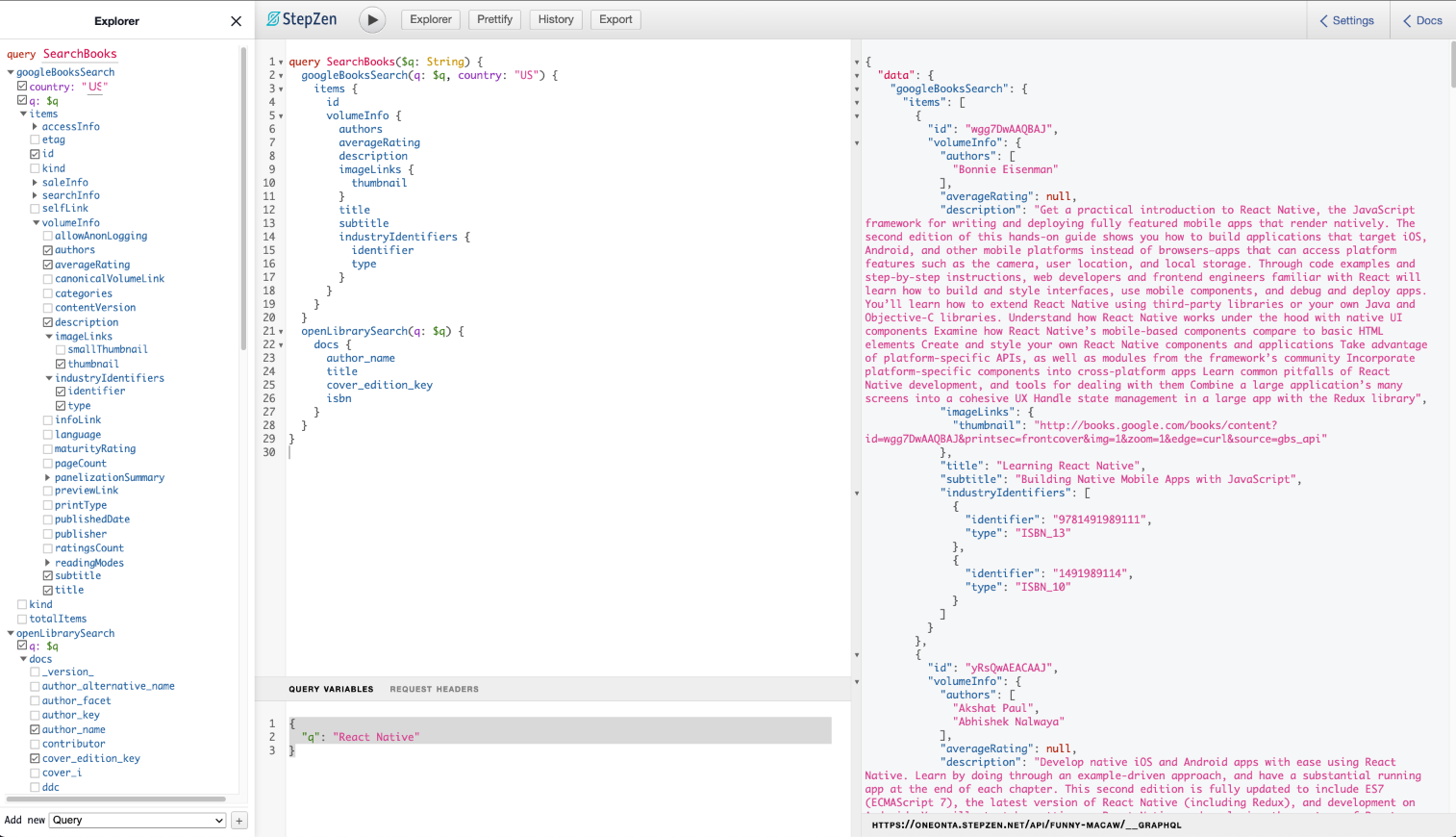


Рисунок 1.1 – Пенель StepZen

Лістинг 1.1

*query* SearchBooks($q: String) {

googleBooksSearch(q: $q, country: "US") {

items {

id

volumeInfo {

authors

averageRating

description

imageLinks {

thumbnail

}

title

subtitle

industryIdentifiers {

identifier

*type*

}

}

}

}

openLibrarySearch(q: $q) {

docs {

author\_name

title

cover\_edition\_key

isbn

}

}

}

Далі було створено додаток за допомогою Expo та підключено потрібні для роботи бібліотеки.

Список бібліотек зображено в лістингу 1.2.

Лістинг 1.2

"@apollo/client": "^3.7.1",

    "@expo/vector-icons": "^13.0.0",

    "@react-navigation/bottom-tabs": "^6.0.5",

    "@react-navigation/native": "^6.0.2",

    "@react-navigation/native-stack": "^6.1.0",

    "expo": "~47.0.6",

    "expo-asset": "~8.6.2",

    "expo-constants": "~14.0.2",

    "expo-font": "~11.0.1",

    "expo-linking": "~3.2.3",

    "expo-splash-screen": "~0.17.5",

    "expo-status-bar": "~1.4.2",

    "expo-system-ui": "~2.0.1",

    "expo-web-browser": "~12.0.0",

    "graphql": "^15.8.0",

    "react": "18.1.0",

    "react-dom": "18.1.0",

    "react-native": "0.70.5",

    "react-native-safe-area-context": "4.4.1",

    "react-native-screens": "~3.18.0",

    "react-native-web": "~0.18.9",

    "@react-native-async-storage/async-storage": "~1.17.3"

За допомогою функцій бібліотеки **Apollo Client** було реалізовану клієнську частину де ми авторизувались в stepzen по нашому особистому api ключу.

Підключеняя відбувається в файлі App.tsx, файл зображено в літингу 1.3.

Лістинг 1.3

import { StatusBar } from "expo-status-bar";

import { SafeAreaProvider } from "react-native-safe-area-context";

import { ApolloClient, InMemoryCache, ApolloProvider } from "@apollo/client";

import useCachedResources from "./hooks/useCachedResources";

import useColorScheme from "./hooks/useColorScheme";

import Navigation from "./navigation";

import MyBooksProvider from "./context/MyBooksProvider";

const API\_KEY =

  "oneonta::stepzen.net+1000::1c894b653ee8a3d59cf2e0a87d22803af367cb5f23ebb009ed5b840b258586c4";

const client = new ApolloClient({

  uri: "https://oneonta.stepzen.net/api/good-deer/\_\_graphql",

  headers: {

    Authorization: `Apikey ${API\_KEY}`,

  },

  cache: new InMemoryCache(),

});

export default function App() {

  const isLoadingComplete = useCachedResources();

  const colorScheme = useColorScheme();

  if (!isLoadingComplete) {

    return null;

  } else {

    return (

      <SafeAreaProvider>

        <ApolloProvider client={client}>

          <MyBooksProvider>

            <Navigation colorScheme={colorScheme} />

          </MyBooksProvider>

        </ApolloProvider>

        <StatusBar />

      </SafeAreaProvider>

    );

  }

}

Пошук книжок в наших в API Google та Open Library реалізвано в файлі SearchScreen.tsx за допомогою пошукової строки.

Файл SearchScreen.tsx зображено в лістингу 1.4

import { gql, useLazyQuery } from "@apollo/client";

import {

  ActivityIndicator,

  StyleSheet,

  FlatList,

  TextInput,

  Button,

} from "react-native";

import { Text, View } from "../components/Themed";

import { RootTabScreenProps } from "../types";

import React, { useState } from "react";

import BookItem from "../components/BookItem";

import { SafeAreaView } from "react-native-safe-area-context";

const query = gql`

  query SearchBooks($q: String) {

    googleBooksSearch(q: $q, country: "US") {

      items {

        id

        volumeInfo {

          authors

          averageRating

          description

          imageLinks {

            thumbnail

          }

          title

          subtitle

          industryIdentifiers {

            identifier

            type

          }

        }

      }

    }

    openLibrarySearch(q: $q) {

      docs {

        author\_name

        title

        cover\_edition\_key

        isbn

      }

    }

  }

`;

export default function SearchScreen() {

  const [search, setSearch] = useState("");

  const [provider, setProvider] = useState<

    "googleBooksSearch" | "openLibrarySearch"

  >("googleBooksSearch");

  const [runQuery, { data, loading, error }] = useLazyQuery(query);

  const parseBook = (item) => {

    if (provider === "googleBooksSearch") {

      const isbn = item.volumeInfo.industryIdentifiers?.find(

        (id) => id.type === "ISBN\_13"

      )?.identifier;

      return {

        title: item.volumeInfo.title,

        image: item.volumeInfo.imageLinks?.thumbnail,

        authors: item.volumeInfo.authors,

        isbn: isbn || item.volumeInfo.industryIdentifiers?.[0]?.identifier,

      };

    } else {

      return {

        title: item.title,

        authors: item.author\_name,

        image: `https://covers.openlibrary.org/b/olid/${item.cover\_edition\_key}-M.jpg`,

        isbn: item.isbn?.[0],

      };

    }

  };

  return (

    <SafeAreaView edges={["top"]} style={styles.container}>

      <View style={styles.header}>

        <TextInput

          value={search}

          onChangeText={setSearch}

          placeholder="Пошук"

          style={styles.input}

        />

        <Button

          title="Пошук"

          onPress={() => runQuery({ variables: { q: search } })}

        />

      </View>

      <View style={styles.tabs}>

        <Text

          style={

            provider === "googleBooksSearch"

              ? { fontWeight: "bold", color: "royalblue" }

              : {}

          }

          onPress={() => setProvider("googleBooksSearch")}

        >

          Google Books

        </Text>

        <Text

          style={

            provider === "openLibrarySearch"

              ? { fontWeight: "bold", color: "royalblue" }

              : {}

          }

          onPress={() => setProvider("openLibrarySearch")}

        >

          Open Library

        </Text>

      </View>

      {loading && <ActivityIndicator />}

      {error && (

        <View style={styles.container}>

          <Text style={styles.title}>Error fetching books</Text>

          <Text>{error.message}</Text>

        </View>

      )}

      <FlatList

        data={

          provider === "googleBooksSearch"

            ? data?.googleBooksSearch?.items

            : data?.openLibrarySearch?.docs || []

        }

        renderItem={({ item }) => <BookItem book={parseBook(item)} />}

        showsVerticalScrollIndicator={false}

      />

    </SafeAreaView>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    padding: 10,

    backgroundColor: "white",

  },

  header: {

    flexDirection: "row",

    alignItems: "center",

  },

  title: {

    fontSize: 20,

    fontWeight: "bold",

  },

  separator: {

    marginVertical: 30,

    height: 1,

    width: "80%",

  },

  input: {

    flex: 1,

    padding: 10,

    borderColor: "gainsboro",

    borderWidth: 1,

    borderRadius: 5,

    marginVertical: 5,

  },

  tabs: {

    flexDirection: "row",

    justifyContent: "space-around",

    alignItems: "center",

    height: 50,

  },

});

Відображення списку книг після запиту в рядку пошуку реалізовано в файлах MyBooksScreen.tsx та BookItem.tsx. Зміст файлів зображено в лістингах 1.5 та 1.6.

Лістинг 1.5

import { FlatList, StyleSheet } from "react-native";

import BookItem from "../components/BookItem";

import { View } from "../components/Themed";

import { useMyBooks } from "../context/MyBooksProvider";

export default function MyBooksScreen() {

  const { savedBooks } = useMyBooks();

  return (

    <View style={styles.container}>

      <FlatList

        data={savedBooks}

        renderItem={({ item }) => <BookItem book={item} />}

      />

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    padding: 10,

  },

  title: {

    fontSize: 20,

    fontWeight: "bold",

  },

  separator: {

    marginVertical: 30,

    height: 1,

    width: "80%",

  },

});

Лістинг 1.6

import { View, Text, StyleSheet, Image, Pressable } from "react-native";

import React from "react";

import Colors from "../constants/Colors";

import { useMyBooks } from "../context/MyBooksProvider";

type BookItemProps = {

  book: Book;

};

const BookItem = ({ book }: BookItemProps) => {

  const { onToggleSaved, isBookSaved } = useMyBooks();

  const saved = isBookSaved(book);

  return (

    <View style={styles.container}>

      <Image source={{ uri: book.image }} style={styles.image} />

      <View style={styles.contentContainer}>

        <Text style={styles.title}>{book.title}</Text>

        <Text>{book.authors?.join(", ")}</Text>

        <Pressable

          style={[styles.button, saved ? { backgroundColor: "lightgray" } : {}]}

          onPress={() => onToggleSaved(book)}

        >

          <Text style={styles.buttonText}>

            {saved ? "Прочитано" : "Хочу прочитати"}

          </Text>

        </Pressable>

      </View>

    </View>

  );

};

const styles = StyleSheet.create({

  container: {

    flexDirection: "row",

    marginVertical: 10,

  },

  image: {

    flex: 1,

    aspectRatio: 2 / 3,

    marginRight: 10,

  },

  contentContainer: {

    flex: 4,

    borderColor: "lightgray",

    borderBottomWidth: 0.5,

  },

  title: {

    fontSize: 16,

    fontWeight: "500",

  },

  button: {

    backgroundColor: Colors.light.tint,

    alignSelf: "flex-start",

    marginTop: "auto",

    marginVertical: 10,

    padding: 7,

    paddingHorizontal: 15,

    borderRadius: 5,

  },

  buttonText: {

    color: "white",

    fontWeight: "600",

  },

});

export default BookItem;

Список збережених книг можна преглядати на сторінці «Мої Книги», реалізація цієї сторінки зображено в лістингу 1.6.

Лістинг 1.6

import { FlatList, StyleSheet } from "react-native";

import BookItem from "../components/BookItem";

import { View } from "../components/Themed";

import { useMyBooks } from "../context/MyBooksProvider";

export default function MyBooksScreen() {

  const { savedBooks } = useMyBooks();

  return (

    <View style={styles.container}>

      <FlatList

        data={savedBooks}

        renderItem={({ item }) => <BookItem book={item} />}

      />

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    padding: 10,

  },

  title: {

    fontSize: 20,

    fontWeight: "bold",

  },

  separator: {

    marginVertical: 30,

    height: 1,

    width: "80%",

  },

});

На рисунку 1.3-1.4 та зображено пошук в Open Library та Google Books.

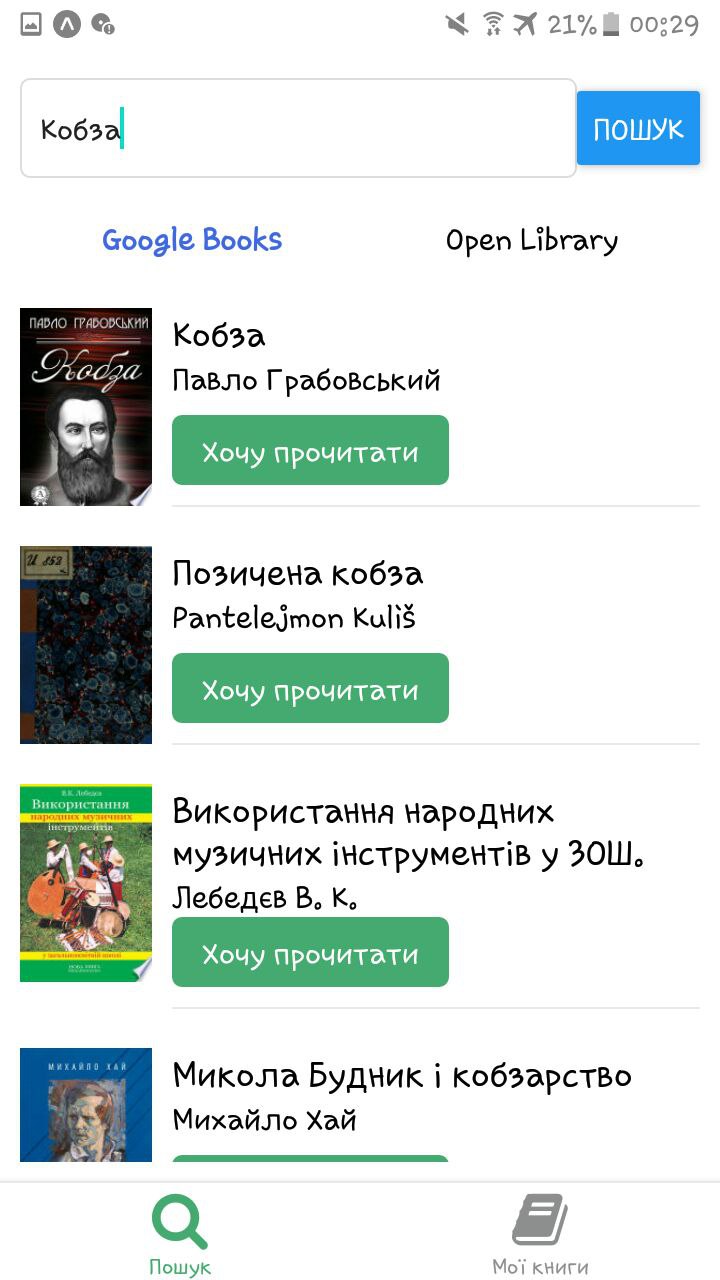


Рисунок 1.3 – Пошук книжок в Google Books

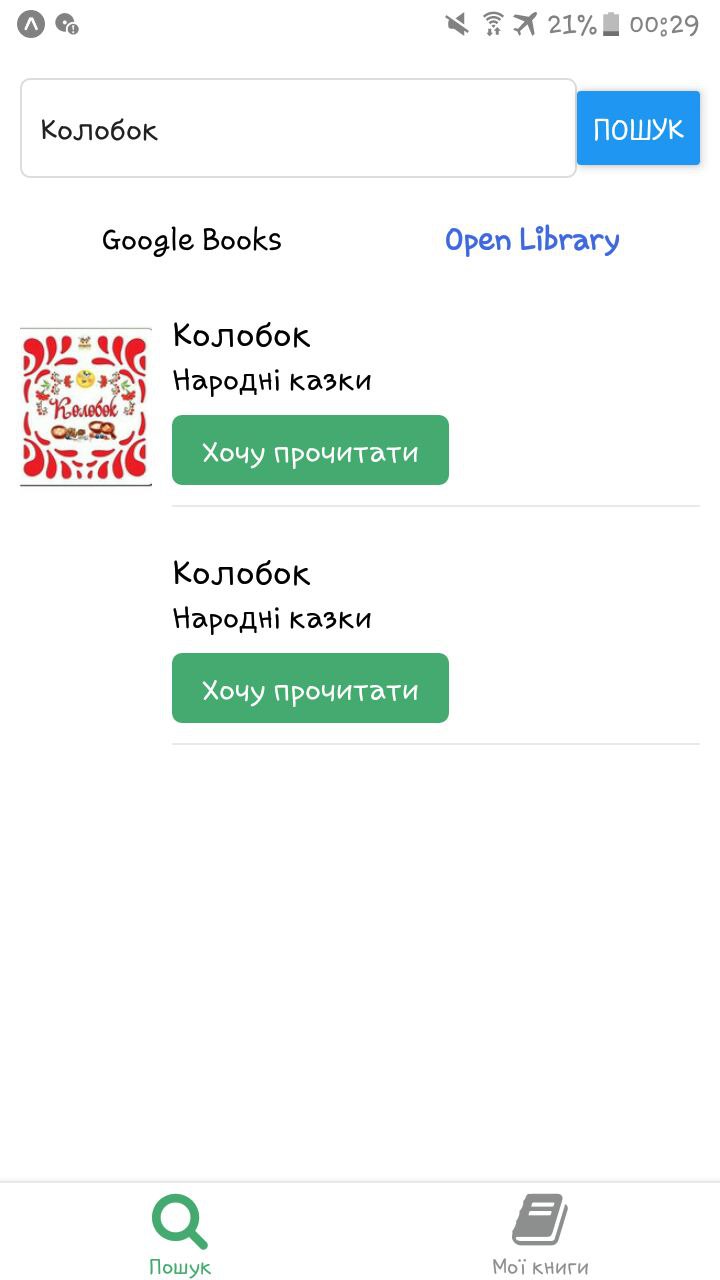


Рисунок 1.4 – Пошук книжок в Open Library

На рисунку 1.5 зображено екран з списком збережених мною книжок які я бажаю прочитати в майбутньому.

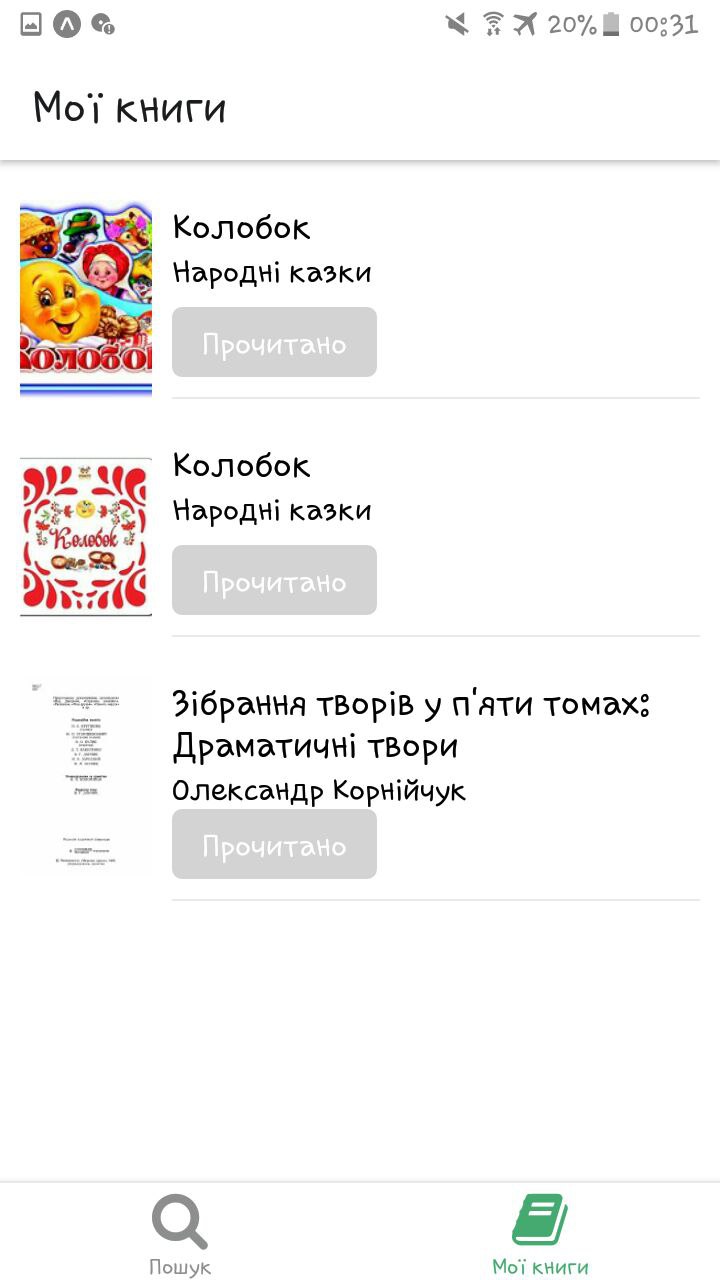


Рисунок 1.5 – Список збережених мною книг.

**Висновок**: навчився працювати з віддаленими сервісами Google Books API та Open Library API за допомогою HTTP-методів та методів аутентифікації.

<https://github.com/Ilya-Olenchenko/BooksApp.git>