Для оцінки продуктивності пошуку книг у системі було виміряно час виконання HTTP-запиту до Google Books API для різних ключових слів із параметром maxResults=.

Використовується стандартне API Google Books за URL форматом:  
https://www.googleapis.com/books/v1/volumes?q={keyword}&maxResults=

Додала спеціально функцію з набором слів та масивом значень maxResults:

 def run\_batch\_searches(self):

        search\_words = ["python", "c++", "java", "javascript", "animal",

                        "history", "nature", "machine learning", "quantum physics", "love", "zymurgy"]  # Приклад списку ключових слів

        max\_results\_list = [5, 10, 20, 40]

        for max\_results in max\_results\_list:

            print(f"\n--- Searching with maxResults={max\_results} ---")

            for query in search\_words:

                print(f"Searching for '{query}' ...")

                start\_time = time.perf\_counter()

                url = f"https://www.googleapis.com/books/v1/volumes?q={query}&maxResults={max\_results}"

                response = requests.get(url)

                if response.status\_code != 200:

                    print(f"Error fetching data for query '{query}'")

                    continue

                data = response.json()

                # Тут можна додати обробку даних, якщо потрібно

                end\_time = time.perf\_counter()

                elapsed = end\_time - start\_time

                print(f"[Timing] Query '{query}' with maxResults={max\_results} took {elapsed:.4f} seconds")

Вона підставляє слова та кількість результатів для виведення в функцію Search.

Вимірюємо за допомогою бібліотеки time час від моменту відправлення запиту до отримання повної відповіді від сервера.

Запити здійснювалися послідовно з однаковими параметрами, змінювалось лише ключове слово для пошуку.

 def search(self):

        self.save\_current\_state\_as\_memento()

        self.clear\_results()

        query = self.search\_box.text().strip()

        if not query:

            return

        start\_time = time.perf\_counter()  # Починаємо вимір

        max\_results = 20  # Можна зробити параметром

        url = f"https://www.googleapis.com/books/v1/volumes?q={query}&maxResults={max\_results}"

        response = requests.get(url)

        if response.status\_code != 200:

            print("Error fetching data")

            return

        data = response.json()

        group\_mode = self.grouping\_box.currentText()

        grouped = {}

        ungrouped = []

        for item in data.get('items', []):

            info = item.get('volumeInfo', {})

            title = info.get('title', 'N/A')

            published\_date = info.get('publishedDate', 'N/A')

            rating = info.get('averageRating', 'N/A')

            image = info.get('imageLinks', {}).get('thumbnail', '')

            authors = info.get('authors', [])

            self.notifier.notify(title)

            leaf = BookLeaf(title, image, published\_date, rating, authors)

            if group\_mode == "Group by Year":

                key = published\_date.split('-')[0] if published\_date != 'N/A' else "Unknown"

            elif group\_mode == "Group by Rating":

                key = str(rating) if rating != 'N/A' else "No Rating"

            elif group\_mode == "Group by First Letter":

                key = title[0].upper() if title and title[0].isalpha() else "#"

            elif group\_mode == "Group by Author":

                key = authors[0] if authors else "Unknown Author"

            else:

                key = None

            if key is None:

                ungrouped.append(leaf)

            else:

                if key not in grouped:

                    grouped[key] = BookComposite(key)

                grouped[key].add(leaf)

        if group\_mode == "No Grouping":

            for leaf in ungrouped:

                leaf.display(

                    self.results\_layout,

                    show\_date=self.check\_var.isChecked(),

                    show\_rating=self.check\_var2.isChecked()

                )

        else:

            for key in sorted(grouped.keys()):

                grouped[key].display(

                    self.results\_layout,

                    show\_date=self.check\_var.isChecked(),

                    show\_rating=self.check\_var2.isChecked()

                )

        end\_time = time.perf\_counter()  # Кінець виміру

        elapsed = end\_time - start\_time

        print(f"[Timing] Search for '{query}' with maxResults={max\_results} took {elapsed:.4f} seconds")

Потім просто сама функція run\_batch\_searches викликається в main:

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    app.setStyle("Fusion")

    window = BookRecommender()

    window.show()

 # Запускаємо батч-пошук після того, як вікно створено

    window.run\_batch\_searches()

    sys.exit(app.exec\_())

    sys.exit(app.exec\_())

**Результати**

| **Запит** | **maxResults=5** | **maxResults=10** | **maxResults=20** | **maxResults=40** |
| --- | --- | --- | --- | --- |
| python | 0.5104 | 0.4246 | 0.5519 | 0.8032 |
| c++ | 0.6510 | 0.6271 | 0.5702 | 0.6986 |
| java | 0.5334 | 0.5741 | 0.5726 | 1.1681 |
| javascript | 0.4566 | 0.6042 | 0.5151 | 0.6484 |
| animal | 0.5961 | 0.6965 | 0.5702 | 0.7872 |
| history | 0.5268 | 0.5080 | 0.6800 | 0.9500 |
| nature | 0.4937 | 0.6118 | 0.6837 | 0.9851 |
| machine learning | 0.4693 | 0.4965 | 0.4884 | 0.7234 |
| quantum physics | 0.4590 | 1.0894 | 0.6926 | 0.6829 |
| love | 0.6775 | 0.5758 | 0.5233 | 0.6599 |
| zymurgy | 0.5767 | 0.5808 | 0.4459 | 0.7268 |

### Аналіз

Було проведено вимірювання часу виконання пошуку для різних запитів і різної кількості максимальних результатів (maxResults). Час виконання зростає зі збільшенням maxResults, що пояснюється більшим обсягом оброблених даних. Час варіюється від ~0.4 до ~1.2 секунд залежно від запиту та параметрів. Спостерігаються незначні коливання часу через мережеві та серверні фактори. Для більш точних результатів вимірювання проводилися автоматично у коді, а не вручну, що забезпечує об’єктивність і відтворюваність.

import sys

import requests

from PyQt5.QtCore import Qt

from PyQt5.QtWidgets import (

    QApplication, QWidget, QVBoxLayout, QLabel, QLineEdit, QPushButton,

    QCheckBox, QScrollArea, QComboBox

)

from book\_components import BookComposite, BookLeaf  # переконайся, що ці класи коректні

from observer import BookNotifier, UserKeywordSubscriber

from search\_memento import SearchMemento, SearchHistory

import time

class BookRecommender(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Створюємо об’єкт BookNotifier і підписника

        self.notifier = BookNotifier()

        self.keyword\_subscriber = UserKeywordSubscriber()

        self.notifier.subscribe(self.keyword\_subscriber)

        self.history = SearchHistory()

        self.init\_ui()

    def init\_ui(self):

        self.apply\_styles()

        self.setWindowTitle("Book Recommender System")

        self.setGeometry(200, 100, 1250, 900)

        self.layout = QVBoxLayout(self)

        self.heading = QLabel("BOOK RECOMMENDATION", self)

        self.heading.setStyleSheet("font: 30px bold; color: red;")

        self.heading.setAlignment(Qt.AlignCenter)

        self.search\_box = QLineEdit(self)

        self.search\_box.setPlaceholderText("Enter book name")

        self.search\_box.setStyleSheet("font: 20px; background-color: white;")

        self.search\_button = QPushButton("Search", self)

        self.search\_button.clicked.connect(self.search)

        self.check\_var = QCheckBox("Publish Date", self)

        self.check\_var.setChecked(True)

        self.check\_var2 = QCheckBox("Rating", self)

        self.check\_var2.setChecked(True)

        self.grouping\_box = QComboBox(self)

        self.grouping\_box.addItem("No Grouping")

        self.grouping\_box.addItem("Group by Year")

        self.grouping\_box.addItem("Group by Rating")

        self.grouping\_box.addItem("Group by First Letter")

        self.grouping\_box.addItem("Group by Author")

        self.grouping\_box.currentIndexChanged.connect(self.search)

        # --- Нові елементи для підписки на ключові слова ---

        self.keyword\_input = QLineEdit(self)

        self.keyword\_input.setPlaceholderText("Enter keyword to subscribe")

        self.keyword\_input.setStyleSheet("font: 16px; background-color: white;")

        self.subscribe\_button = QPushButton("Subscribe Keyword", self)

        self.subscribe\_button.clicked.connect(self.add\_keyword\_subscription)

        self.keywords\_label = QLabel("Subscribed keywords: None", self)

        self.keywords\_label.setStyleSheet("color: blue; font: 14px;")

        # -------------------------------------------------------

        self.layout.addWidget(self.heading)

        self.layout.addWidget(self.search\_box)

        self.layout.addWidget(self.search\_button)

        self.layout.addWidget(self.check\_var)

        self.layout.addWidget(self.check\_var2)

        self.layout.addWidget(QLabel("Group by:", self))

        self.layout.addWidget(self.grouping\_box)

        # Кнопки Undo і Redo

        self.undo\_button = QPushButton("Undo", self)

        self.undo\_button.clicked.connect(self.undo\_search)

        self.redo\_button = QPushButton("Redo", self)

        self.redo\_button.clicked.connect(self.redo\_search)

        self.undo\_button.setObjectName("undoButton")

        self.redo\_button.setObjectName("redoButton")

        self.layout.addWidget(self.undo\_button)

        self.layout.addWidget(self.redo\_button)

        # Додаємо нові елементи для підписки

        self.layout.addWidget(self.keyword\_input)

        self.layout.addWidget(self.subscribe\_button)

        self.layout.addWidget(self.keywords\_label)

        self.results\_layout = QVBoxLayout()

        self.scroll\_area = QScrollArea(self)

        self.scroll\_area.setWidgetResizable(True)

        self.results\_widget = QWidget()

        self.results\_widget.setLayout(self.results\_layout)

        self.scroll\_area.setWidget(self.results\_widget)

        self.layout.addWidget(self.scroll\_area)

    def run\_batch\_searches(self):

        search\_words = ["python", "c++", "java", "javascript", "animal",

                        "history", "nature", "machine learning", "quantum physics", "love", "zymurgy"]  # Приклад списку ключових слів

        max\_results\_list = [5, 10, 20, 40]

        for max\_results in max\_results\_list:

            print(f"\n--- Searching with maxResults={max\_results} ---")

            for query in search\_words:

                print(f"Searching for '{query}' ...")

                start\_time = time.perf\_counter()

                url = f"https://www.googleapis.com/books/v1/volumes?q={query}&maxResults={max\_results}"

                response = requests.get(url)

                if response.status\_code != 200:

                    print(f"Error fetching data for query '{query}'")

                    continue

                data = response.json()

                # Тут можна додати обробку даних, якщо потрібно

                end\_time = time.perf\_counter()

                elapsed = end\_time - start\_time

                print(f"[Timing] Query '{query}' with maxResults={max\_results} took {elapsed:.4f} seconds")

    def apply\_styles(self):

        self.setStyleSheet("""

            QWidget {

                font-family: Segoe UI;

                font-size: 16px;

            }

            QLabel#heading {

                font-size: 30px;

                font-weight: bold;

                color: #B22222;

            }

            QLineEdit {

                padding: 6px;

                border: 1px solid #ccc;

                border-radius: 5px;

                background-color: #fdfdfd;

            }

            QPushButton {

                padding: 6px 12px;

                font-weight: bold;

                background-color: #007ACC;

                color: white;

                border-radius: 5px;

            }

            QPushButton:hover {

                background-color: #005F99;

            }

            QCheckBox {

                margin: 5px;

            }

            QComboBox {

                padding: 6px;

                border-radius: 5px;

            }

            QLabel#keywords {

                color: #003399;

                font-style: italic;

            }

            QScrollArea {

                background-color: #FAFAFA;

                border: none;

            }

            QPushButton#undoButton {

                background-color: #FF6F61;  /\* теплий червоний/помаранчевий \*/

                color: white;

            }

            QPushButton#undoButton:hover {

                background-color: #E85C50;

            }

            QPushButton#redoButton {

                background-color: #4CAF50;  /\* зелений \*/

                color: white;

            }

            QPushButton#redoButton:hover {

                background-color: #3E8E41;

            }

        """)

    def add\_keyword\_subscription(self):

        keyword = self.keyword\_input.text().strip().lower()

        if keyword:

            self.keyword\_subscriber.add\_keyword(keyword)

            keywords\_list = ', '.join(sorted(self.keyword\_subscriber.keywords))

            self.keywords\_label.setText(f"Subscribed keywords: {keywords\_list}")

            self.keyword\_input.clear()

    def clear\_results(self):

        while self.results\_layout.count():

            item = self.results\_layout.takeAt(0)

            widget = item.widget()

            if widget is not None:

                widget.setParent(None)

                widget.deleteLater()

    def save\_current\_state\_as\_memento(self):

        memento = SearchMemento(

            query=self.search\_box.text().strip(),

            group\_mode=self.grouping\_box.currentText(),

            show\_date=self.check\_var.isChecked(),

            show\_rating=self.check\_var2.isChecked()

        )

        self.history.save(memento)

    def restore\_search\_from\_memento(self, memento):

        self.search\_box.setText(memento.query)

        index = self.grouping\_box.findText(memento.group\_mode)

        if index != -1:

            self.grouping\_box.setCurrentIndex(index)

        self.check\_var.setChecked(memento.show\_date)

        self.check\_var2.setChecked(memento.show\_rating)

        self.perform\_search\_from\_memento(memento)

    def perform\_search\_from\_memento(self, memento):

        self.clear\_results()

        if not memento.query:

            return

        url = f"https://www.googleapis.com/books/v1/volumes?q={memento.query}&maxResults=5"

        response = requests.get(url)

        if response.status\_code != 200:

            print("Error fetching data")

            return

        data = response.json()

        group\_mode = memento.group\_mode

        grouped = {}

        ungrouped = []

        for item in data.get('items', []):

            info = item.get('volumeInfo', {})

            title = info.get('title', 'N/A')

            published\_date = info.get('publishedDate', 'N/A')

            rating = info.get('averageRating', 'N/A')

            image = info.get('imageLinks', {}).get('thumbnail', '')

            authors = info.get('authors', [])

            self.notifier.notify(title)

            leaf = BookLeaf(title, image, published\_date, rating, authors)

            if group\_mode == "Group by Year":

                key = published\_date.split('-')[0] if published\_date != 'N/A' else "Unknown"

            elif group\_mode == "Group by Rating":

                key = str(rating) if rating != 'N/A' else "No Rating"

            elif group\_mode == "Group by First Letter":

                key = title[0].upper() if title and title[0].isalpha() else "#"

            elif group\_mode == "Group by Author":

                key = authors[0] if authors else "Unknown Author"

            else:

                key = None

            if key is None:

                ungrouped.append(leaf)

            else:

                if key not in grouped:

                    grouped[key] = BookComposite(key)

                grouped[key].add(leaf)

        if group\_mode == "No Grouping":

            for leaf in ungrouped:

                leaf.display(self.results\_layout, show\_date=memento.show\_date, show\_rating=memento.show\_rating)

        else:

            for key in sorted(grouped.keys()):

                grouped[key].display(self.results\_layout, show\_date=memento.show\_date, show\_rating=memento.show\_rating)

    def undo\_search(self):

        memento = self.history.undo()

        if memento:

            self.restore\_search\_from\_memento(memento)

    def redo\_search(self):

        memento = self.history.redo()

        if memento:

            self.restore\_search\_from\_memento(memento)

    def search(self):

        self.save\_current\_state\_as\_memento()

        self.clear\_results()

        query = self.search\_box.text().strip()

        if not query:

            return

        start\_time = time.perf\_counter()  # Починаємо вимір

        max\_results = 20  # Можна зробити параметром

        url = f"https://www.googleapis.com/books/v1/volumes?q={query}&maxResults={max\_results}"

        response = requests.get(url)

        if response.status\_code != 200:

            print("Error fetching data")

            return

        data = response.json()

        group\_mode = self.grouping\_box.currentText()

        grouped = {}

        ungrouped = []

        for item in data.get('items', []):

            info = item.get('volumeInfo', {})

            title = info.get('title', 'N/A')

            published\_date = info.get('publishedDate', 'N/A')

            rating = info.get('averageRating', 'N/A')

            image = info.get('imageLinks', {}).get('thumbnail', '')

            authors = info.get('authors', [])

            self.notifier.notify(title)

            leaf = BookLeaf(title, image, published\_date, rating, authors)

            if group\_mode == "Group by Year":

                key = published\_date.split('-')[0] if published\_date != 'N/A' else "Unknown"

            elif group\_mode == "Group by Rating":

                key = str(rating) if rating != 'N/A' else "No Rating"

            elif group\_mode == "Group by First Letter":

                key = title[0].upper() if title and title[0].isalpha() else "#"

            elif group\_mode == "Group by Author":

                key = authors[0] if authors else "Unknown Author"

            else:

                key = None

            if key is None:

                ungrouped.append(leaf)

            else:

                if key not in grouped:

                    grouped[key] = BookComposite(key)

                grouped[key].add(leaf)

        if group\_mode == "No Grouping":

            for leaf in ungrouped:

                leaf.display(

                    self.results\_layout,

                    show\_date=self.check\_var.isChecked(),

                    show\_rating=self.check\_var2.isChecked()

                )

        else:

            for key in sorted(grouped.keys()):

                grouped[key].display(

                    self.results\_layout,

                    show\_date=self.check\_var.isChecked(),

                    show\_rating=self.check\_var2.isChecked()

                )

        end\_time = time.perf\_counter()  # Кінець виміру

        elapsed = end\_time - start\_time

        print(f"[Timing] Search for '{query}' with maxResults={max\_results} took {elapsed:.4f} seconds")

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    app.setStyle("Fusion")

    window = BookRecommender()

    window.show()

 # Запускаємо батч-пошук після того, як вікно створено

    window.run\_batch\_searches()

    sys.exit(app.exec\_())

    sys.exit(app.exec\_())