Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти __ другий (магістерський)_____
(рівень вищої освіти)

Спеціалізовані програмні засоби для управління закладками в__ інтернет-ресурсах_____
(тема)

Виконав:
здобувач ____2_____ року навчання,
групи _____СКСм-23-1_____

_____Мякшин А.С._____
(прізвище, ініціали)

Спеціальність _123 – Комп'ютерна_____ інженерія_____
(код і повна назва спеціальності)

Тип програми ___освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_Спеціалізовані_____ комп'ютерні системи_____
(повна назва освітньої програми)

Керівник ___проф., д.т.н. Кривуля Г.Ф.___
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ __Чумаченко С.В.__
(підпис) (прізвище, ініціали)

20_25 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія
(шифр і назва)

Тип програми Освітньо професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Мякшину Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Спеціалізовані програмні засоби для управління закладками в інтернет-ресурсах

затверджена наказом по університету від 08 листопада 2024 р. № 1189
2. Термін подання студентом роботи до екзаменаційної комісії 13 січня 2025 р.
3. Вихідні дані до роботи
1. Документація мови програмування Python
2. Документація фреймворку Flask
3. Документація бібліотеки Tkinter
4. Редактор коду: PyCharm

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області
2. Засоби реалізації програмного застосунку
3. Опис програмної реалізації
4. Системні вимоги та інструкція користувача
5. Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) __Комп'ютерна презентація. Слайдів – 25_____
_____
_____
_____
_____
_____
_____
_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

7. Дата видачі завдання ___01.09.2024_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів	Примітка
1	Отримання завдання	01.09	
2	Аналіз наукових та технічних ресурсів	02.09-03.09	
5	Вибір технологій та інструментів розробки	04.09-06.09	
6	Вибір архітектурного шаблону та структури	07.09-08.09	
7	Створення дизайну та інтерфейсу програми	09.09-11.09	
8	Розробка програмного забезпечення	12.09-20.10	
9	Тестування та дослідження проєкту	21.10-25.10	
10	Написання кваліфікаційної роботи	26.10-09.11	

Здобувач _____
(підпис)

Керівник роботи _____ __проф. Кривуля Г.Ф._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 152 с., 23 рис., 0 табл., 2 дод., 32 джерела.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ЗАКЛАДКА, ПРОГРАМНИЙ ЗАСІБ, УПРАВЛІННЯ ЗАКЛАДКАМИ, PYTHON, ШТУЧНИЙ ІНТЕЛЕКТ, РЕКОМЕНДАЦІЇ КОНТЕНТУ, ПЕРЕВІРКА ПОСИЛАННЯ

Об'єкт дослідження – управління процесом організації та взаємодії з цифровими закладками за допомогою спеціалізованих програмних інструментів.

Програмний засіб повинен забезпечувати як стандартні (додання закладки до списку, опис закладки, зміна статусу (пріоритету), видалення закладки), так і нові можливості (перевірка на активність посилань з відповідним повідомленням, можливість працювати офлайн, рекомендація схожого контенту за допомогою штучного інтелекту, перевірка на коректність та інші).

Мета роботи – розробка спеціалізованого програмного засобу, що передбачає універсальний підхід для взаємодії з закладками, а також реалізація інтерактивного гнучкого функціоналу на базі ШІ для рекомендації актуального контенту.

Метод дослідження – порівняльний аналіз існуючих програмних рішень, ознайомлення з відповідною науковою літературою та технічною документацією, складання, налагодження та тестування програми.

Програмний засіб можна використовувати як альтернативу для існуючих програм. Наявність певних додаткових функцій та своєчасних актуальних рекомендацій від ШІ робить ПЗ універсальним. Програма складена на мові програмування Python у середовищі розробки PyCharm.

ABSTRACT

Master's thesis 152 pages, 23 figures, 0 tables, 2 appendices, 32 sources.

SOFTWARE, BOOKMARK, SOFTWARE TOOL, BOOKMARK MANAGEMENT, PYTHON, ARTIFICIAL INTELLIGENCE, CONTENT RECOMMENDATIONS, LINK CHECK

The major goal of this thesis is to manage the organization and interaction process with digital bookmarks using specialized software tools.

The software tool should provide both standard (adding a bookmark to the list, description of the bookmark, changing the status (priority), deleting the bookmark) and new capabilities (checking the activity of links with the appropriate message, the ability to work offline, recommending similar content using artificial intelligence, checking for correctness and others).

The goal of the work is the development of a specialized software tool that provides a universal approach for interacting with bookmarks, as well as the implementation of an AI-based virtual assistant for recommending relevant content.

The research method is a comparative analysis of existing software solutions, familiarization with relevant scientific literature and technical documentation, compilation, debugging, and testing of the program.

The software can be used as an alternative to existing programs. Certain additional functions and timely, relevant recommendations from AI make it universal. The program is written in the Python programming language in the PyCharm development environment.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – прикладний програмний інтерфейс (англ., Application Programming Interface)

CMS – система керування контентом (англ., Content Management System)

HTML – мова розмітки гіпертексту (англ., HyperText Markup Language)

JSON – запис об'єктів JavaScript (англ., JavaScript Object Notation)

MVVM – Модель, Вид, Модель вигляду (англ., Model-View-ViewModel)

UI – інтерфейс користувача (англ., User Interface)

URL – єдиний вказівник на ресурс (англ., Uniform Resource Locator)

RSS – простий доступ до інформації (англ., Really Simple Syndication)

БД – бази даних

ОС – операційна (операційні) системи

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

СКБД – система керування базами даних

ШІ – штучний інтелект

ВСТУП

Завдяки сучасним технологіям у XXI столітті людина має швидкий доступ до великого обсягу інформації та широкий вибір ресурсів для її отримання. Це призводить до взаємодії з великим потоком даних, що надходить із різних джерел: спілкування, радіо, телевізор, газети, але саме інтернет грає особливу роль, так як саме завдяки ньому з'явилися передумови стрімкого розвитку інформаційних технологій. Особливість ознайомлення з ресурсами через мережу «Інтернет» полягає в можливості пошуку необхідних даних, не змінюючи свого місцезнаходження, та в отриманні інформації в будь-який момент часу.

У процесі перегляду за допомогою всесвітнього павутиння певного контенту іноді постає потреба у його збереженні з різних причин: зміст є корисним, цікавим або важливим для користувача [30]. Сучасні браузери, які надають можливість переглядати вебсторінки, мають широкий функціонал, в тому числі мають функцію збереження певного контенту (у спеціальній папці), а сам уніфікований ідентифікатор ресурсу (Uniform Resource Locator, URL) зараз має назву «закладка» [1].

Слід зазначити, що закладки були вперше реалізовані ще у 1992 році у такому браузері як ViolaWWW, а також були присутні у браузері Mosaic у 1993 році. У 1996 році з'явилися сервіси соціальних закладок. Соціальна закладка є централізованим онлайн-сервісом. Він дозволяє користувачам зберігати, змінювати, ділитися посиланнями на вебсторінки.

Відмінність від звичайних закладок полягає в розташуванні: звичайні закладки зберігаються на комп'ютері користувача, у той час, коли соціальні – в інтернеті, що дозволяє мати доступ до обраних сторінок навіть у випадках, коли є певні проблеми з комп'ютером користувача. Такого роду сервіси не одразу мали популярність та здобули її тільки після 2003 року. Закладки, які були створені за допомогою спеціального вебзастосунку, зберігаються на

віддаленому вебсервері, доступ до якого можна отримати з будь-якого місця.

У 2004 році з’явилися «живі» закладки (від Mozilla Firefox). Їх відмінність від звичайних полягала у тому, що «живі» закладки зберігали список посилань на останні статті, які надавалися блогом чи новинним сайтом. Це надавало користувачам можливість динамічно відстежувати зміни в певних ресурсах. Зміст вебсайту, доступний у форматі RSS (канал RSS), вважається за закладку, а не за сторінку HTML (як вважає більшість агрегаторів новин). Закладки, у такому випадку оновлюються в реальному часі. У 2018 році ця функція була видалена.

Підхід до зберігання закладок відрізняється в залежності від наступних факторів: браузер, версія цього браузеру, ОС [3]. Наприклад, деякі браузери зберігають закладки у єдиному HTML-кодованому файлі. Цей підхід працює на різних платформах. Важливо відзначити, що деякі браузери зберігають закладки в форматі транзакційно безпечної бази даних – SQLite. Також використовується підхід зі збереженням закладки у вигляді окремого файлу, який має розширення .URL.

У кожному сучасному браузері функція закладки представляє собою спеціальне меню. Є можливість додати сторінку до закладки, додати усі вкладки до закладки. Можна налаштувати показ панелі закладок – місця у верхній частині вікна браузера, де розташовані збережені посилання. Є можливість імпортувати посилання та налаштування. Присутній диспетчер закладок, де можна шукати та переглядати збережені посилання, додавати папки чи закладки, сортувати зміст, імпортувати та експортувати посилання. Вбудований інструмент для керування закладками має схожі функції в різних браузерах. Відмінність полягає у розташуванні цього інструменту та наявності деяких функцій.

У наш час існує багато програм та застосунків, які мають більш широкий функціонал зі збереження закладок у порівнянні з браузерами [31]. Так як інтернет – це динамічний віртуальний простір, посилання на сайти та вебзастосунки можуть змінюватись або просто припиняти своє існування.

Через це постає потреба у реалізації перевірки посилання на його активність. На жаль, не всі популярні рішення мають таку можливость.

З’єднання – це найважливіший фактор, що враховується під час використання Всесвітнього павутиння . Під час роботи з закладками важливо мати до них постійний доступ, не дивлячись на технічні перешкоди. Сучасні рішення не надають повноцінного доступу до збережених посилань під час відсутності інтернету, що є підставою для реалізації подібного функціоналу.

Слід сказати, що програми для збереження закладок можуть бути використані в якості своєрідного альтернативного джерела інформації [32]. Сучасні інструменти зі збереження посилань рекомендують певний контент (статі, поради тощо), але він не пов’язаний з інтересами користувача. Зі стрімким розвитком ШІ відкриваються можливості для реалізації саме тих рекомендацій, які відповідають тим темам, які є актуальними та індивідуальними для кожного користувача.

Бувають випадки, коли посилання було не зовсім точно скопійоване до буферу обміну. Сучасні рішення зі збереження закладок дозволяють зберігати будь-яке посилання: правильне та неправильне (що нікуди не перенаправляє), не дивлячись на його роботу. Через це постає потреба в перевірці адреси з відповідним відображенням про її стан.

Актуальність роботи полягає у розвитку ШІ та його можливостей, що дає можливість значно розширити взаємодію з закладками.

Підставою для проведення роботи є реалізація нових функцій для керування закладками, які значно підвищать ефективність та швидкість роботу користувача в мережі «Інтернет». Мета роботи – вдосконалення використання закладок, розробка нових можливостей та інтеграція ШІ в процес організації інтернет-ресурсів.

Галуззю застосування програми може бути її використання в особистих цілях (корисна інформація), під час навчання (ресурси) та роботи (важливі дані) у ролі універсального багатофункціонального інструмента для взаємодії зі збереженими ресурсами з метою їх аналізу, організації та систематизації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття спеціалізованих програмних засобів для управління закладками та опис задачі.

Спеціалізовані програмні засоби для управління закладками – це програми або вебзастосунки, що надають широкий функціонал для управління закладками, їх організації та взаємодії з ними.

Після проведення огляду відповідної наукової літератури та технічної документації, був сформований напрям дослідження, який полягає у порівняльному аналізі існуючий рішень, виявленні певних питань та проблем, що потребують розв’язання шляхом реалізації універсального функціоналу з використанням штучного інтелекту, що значно підвищить ефективність організації цифрових закладок.

Задачею цієї роботи є розгляд напрямку управління збереженими посиланнями та розробка спеціалізованої програми для управління закладками в інтернет-ресурсах.

1.2 Аналіз існуючих рішень

На даний момент функціонал сучасних браузерів щодо керування закладками певною мірою обмежений [23]. Через це на ринку існує багато вебзастосунків та програм, які мають більш широкі можливості та надають різні варіанти щодо взаємодії зі збереженими посиланнями. Такого роду програмні рішення мають спільні базові функції, до яких можна віднести:

- створення, редагування та видалення закладки;
- додавання тегу;
- сортування;
- встановлення способу відображення закладок;

- додання іконки до закладки;
- написання нотатку;
- створення папки;
- можливість поділитися закладкою;
- відкриття закладки в новій вкладці;
- додавання закладки до «Обраного»;
- наявність акаунту;
- вибір мови;
- зміна теми.

Інтерфейс менеджера закладок у браузері представлений на рисунку 1.1.

Рисунок 1.1 – Інструмент «Закладки» в Google Chrome

Інші функції відрізняються в залежності від програми, так як кожен програмний засіб реалізує власний підхід до організації збережених посилань.

1.2.1 Raindrop.io

Raindrop.io – це програмне забезпечення з керування закладками. Програма була розроблена однією людиною – програмістом Рустемом Мусабековим та має широкий функціонал з організації посилань.

Окрім базового функціоналу, програма має інші можливості. Пункт «Усі закладки» для збереження всіх доданих посилань. Це дозволяє одразу ознайомитись з усім контентом. Розділ «Несортовані» потрібен для відображення закладок, які не розподілені по папкам, що дає можливість одразу побачити, які саме посилання не мають прив’язки до папок. Пункт «Кошик» містить посилання, які були видалені. Є можливість створювати іконки для папок. Це допомагає візуально розділити усі папки за категоріями та швидко знайти необхідну інформацію. Пункт «Фільтри» сортує контент за певними критеріями, а саме: «Посилання» (для збереження посилань), «Статті» (для збереження статей), «Документи» (містить документи) та «Без тегів» (містить посилання, що не мають тегу). Присутня можливість експорту закладок у різних форматах: HTML, CSV, Text. Інтерфейс вебзастосунку представлений на рисунку 1.2.

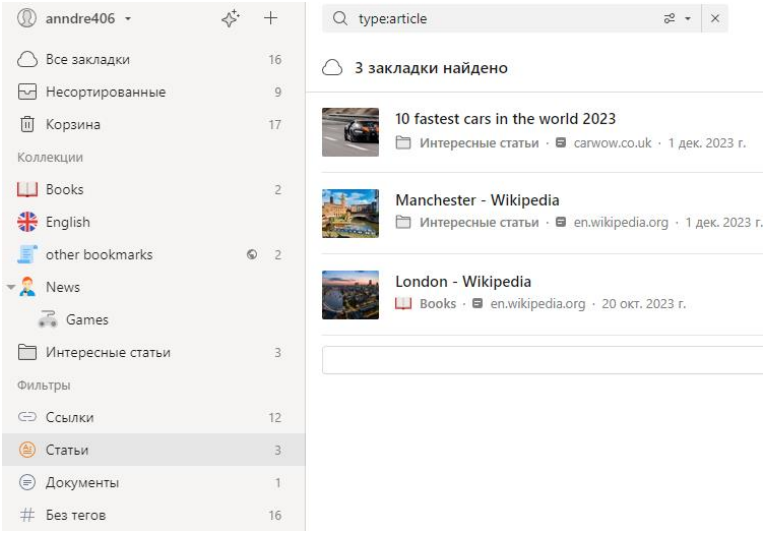

Рисунок 1.2 – Raindrop.io

До переваг Raindrop.io можна віднести широкий безкоштовний функціонал, який дозволяє детально організувати процес керування закладками та, у той самий час, спрости взаємодію з посиланнями, використовуючи різноманітні функціональні можливості.

Існує спеціальне розширення для браузера від Raindrop, що дозволяє використовувати можливості інструмента в більш гнучкому форматі. Усі дані користувача знаходяться на серверах Amazon AWS. Також існує мобільна версія програми. Вона доступна як для Android, так і для iOS. Raindrop має також і десктопну версію програми для Windows, macOS та Linux. Існує як безкоштовна, так і платна версія програми. Платна версія надає більше корисних функцій, у тому числі робота з ШІ. Серед недоліків програми можна виділити відсутність таких можливостей, як робота офлайн у десктопній версії, рекомендації контенту та перевірка посилань на корекність. Слід зазначити, що програма має перевірку на активність посилань: для цього доступні різні режими.

1.2.2 Pocket

Pocket – це інструмент для керування закладками, що дозволяє зберігати статті, зображення, відео з мережі «Інтернет». Pocket з’явився у 2007 році у вигляді доповнення браузера Mozilla Firefox. Тоді інструмент мав назву «Read It Later». Після ребрендингу, застосунок отримав назву «Pocket». Інструмент використовують 17 мільйонів користувачів. Окрім базових функцій, Pocket дозволяє додати збережене посилання до списку. Це може бути корисно, якщо потрібно зберігати закладки за певною тематикою тощо. Можна додати закладки до архіву [19]. Така функція дозволяє зберігати посилання, які не є важливими, у спеціальному місці.

Розділ «Highlights» використовується для збереження певних рядків тексту. Це зручно, коли потрібно зберегти не сам текст, а тільки уривок для подальшого використання. Застосунок має розділ «Нове», що містить рекомендований для ознайомлення матеріал. Ця функція дозволяє користувачеві читати деякі рекомендовані статті безпосередньо в Pocket та одразу зберігати їх. Також є розділ «Колекції», де можна ознайомитись з різними статтями та зберегти їх. Збережені матеріали додаються до списку

облікового запису. Все це є синхронізованим для усіх підключений пристроїв. Застосунок Pocket є доступним у вигляді вебверсії, а також для таких операційних систем, як Windows, macOS. Інтерфейс вебзастосунку зображений на рисунку. 1.3.Існує версія для мобільних пристроїв: Android та iOS. Також інструмент можна встановити для Windows Phone, BlackBerry, Kobo eReaders. Існує безкоштовна версія та платна. Платна версія має розширений функціонал.

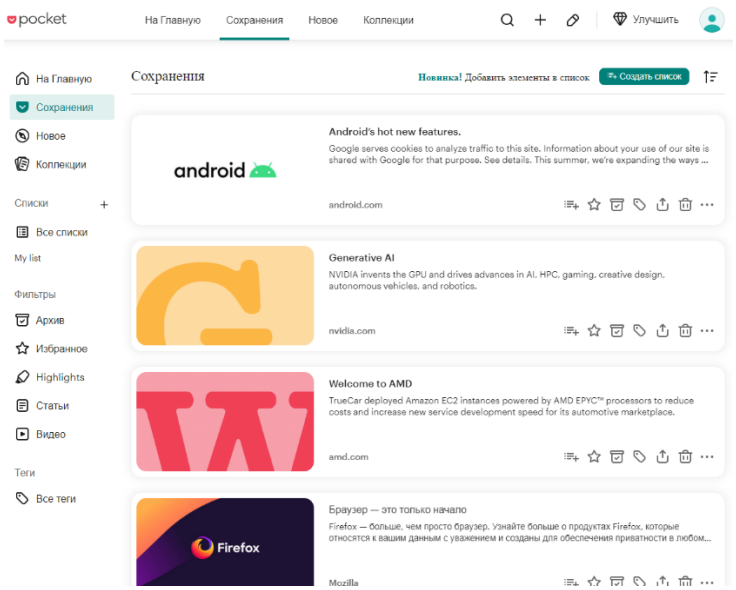

Рисунок 1.3 – Pocket

До переваг Pocket можна віднести досить інтуїтивно зрозумілий інтерфейс, який дозволяє швидко та ефективно керувати усім збереженим контентом. До недоліків застосунку можна віднести відсутність можливості перевірки посилань на активність, роботи офлайн (нема десктопної версії), рекомендації контенту не відповідають інтересам користувача, дозволяється додавати некоректні посилання.

1.2.3 Bookmark OS

Bookmark OS – це програмне забезпечення з керування закладками.

Програма була розроблена програмістом Девідом Линамом та надає різноманітний функціонал щодо керування збереженими посиланнями.

Окрім базового функціоналу присутні інші можливості. До них можна віднести додання не тільки закладки, а й нотатки. Є можливість вказати її назву, тег, після чого написати текст нотатки. У Bookmark OS присутній широкий вибір інструментів для написання нотатки: вибір шрифту та його розміру, створення списку, створення таблиці, є блок для запису програмного коду та інше. Є функція, яка дозволяє додавати задачі. Інтерфейс вебзастосунку представлений на рисунку 1.4.

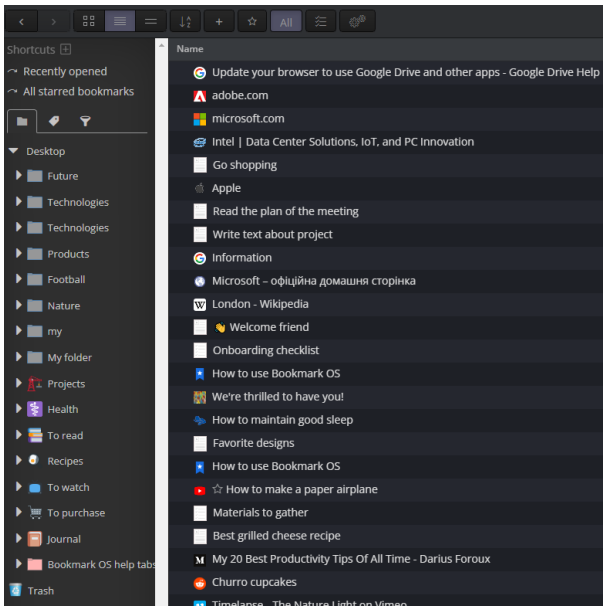

Рисунок 1.4 – Bookmark OS

Можна вказати ім’я та тег задачі, а також дату, сортувати задачі, дивитися виконані задачі тощо. Є функція, яка дозволяє додавати файл. Достатньо перетягнути потрібний файл і він з’явиться у списку. Всі ці можливості дозволяють використовувати Bookmark OS не тільки як програму для керування закладками, а й як блокнот та менеджер задач, а також сховище для файлів, що робить це програмне забезпечення універсальним.

Існує спеціальне розширення Bookmark OS, за допомогою якого можна

швидко створювати закладки безпосередньо під час перегляду вебсторінок. Слід зазначити, що система синхронізує закладки між різними пристроями через хмару. Програма не має мобільної версії, але доступна у вигляді вебзастосунку. Існує як безкоштовна версія програми, так і платна. Платна версія має більш розгорнутий функціонал. До переваг Bookmark OS можна віднести широкий продуманий функціонал у безкоштовній версії та універсальний підхід для організації закладок: зрозумілий інтерфейс та широкі можливості щодо створення власного робочого простору. До недоліків програми можна віднести відсутність таких можливостей, як робота офлайн у десктопній версії (версії для ПК не існує), рекомендації контенту та перевірка посилань на коректність (все ще можна додавати некоректне посилання).

1.3 Об’єкт та предмет дослідження

Об’єкт дослідження – управління процесом організації та взаємодії з цифровими закладками за допомогою спеціалізованих програмних інструментів.

Предмет дослідження – теоретичні, методичні та практичні аспекти реалізації універсального програмного засобу з керування закладками з використанням штучного інтелекту.

1.4. Постановка задачі

Задача дослідження полягає в аналізі підходів до реалізації програмних рішень з керування закладками та в вирішенні проблеми підвищення ефективності організації та управління збереженими посиланнями за допомогою нових функціональних рішень та використання штучного інтелекту.

Для вирішення цього завдання були сформовані наступні загальні

кроки:

- аналіз існуючих продуктів на ринку (повноцінне тестування їх функціоналу, теоретично ознайомитись з платними версіями, виявити конкретні функціональні чи методологічні аспекти, які можуть бути вдосконалені та потребують реалізації в силу своєї актуальності та ефективності;

- аналіз технологій, які будуть використані в процесі реалізації програмної частини та вибір відповідної мови програмування, оптимального фреймворку (бібліотеки) та бази даних, які є доцільними та найбільш актуальними;

- вибір (грунтуючись на обраних технологіях) спеціалізованого середовище розробки (або редактору коду), яке буде цілком відповідати всім вимогам, необхідним для реалізації завдання;

- вибір підходящого архітектурного шаблону та відповідної структури майбутнього проєкта з огляду на його функціонал та запланований масштаб;

- розробка дизайну інтерфейса користувача (можливо, різних варіантів) майбутньої програми;

- реалізація функціоналу програмного забезпечення відповідно до сформованої структури проєкту;

- тестування програмного забезпечення (на різних пристроях з метою переконатися в тому, що функціонал працює згідно запланованим вимогам та без помилок);

- виправлення виявлених помилок (якщо вони були виявлення на етапі тестування програмного засобу);

- повторне тестування програми для перевірки функціоналу;

- створення відповідних системних вимог та інструкції користувача (на основі проведеного повноцінного тестування);

- формування висновків, щодо проведеного дослідження.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАСОБУ

2.1 Мова програмування Python

Основною мовою програмування для розробки програмного засобу з керування закладками був обраний Python. Ця мова програмування є інтерпретованою та об'єктно-орієнтованою. Python був розроблений у 1990 році Гвідо ван Россумом.

Мова підтримує кілька парадигм програмування:

- процедурна;
- об'єктно-орієнтована;
- аспектно-орієнтована;
- функціональна.

Python має низку переваг, серед яких: переносимість програм, чистий синтаксис, можливість використання Python в діалоговому режимі, стандартний дистрибутив має багато корисних модулів, підходить для розв'язання математичних проблем, має просте та потужне середовище розробки (IDLE), написане мовою Python, відкритий код [15].

Стандартна бібліотека Python є різноманітною. З її допомогою можна працювати з різними форматами інтернету та мережевими протоколами. Є можливість розробляти кросплатформні застосунки [8]. Для цього присутні спеціальні модулі. Можна працювати з текстовим кодуванням, регулярними виразами, криптографічними протоколами, серіалізацією даних, юніт-тестуванням, архівами, мультимедійними форматами та з іншими темами. Також доступна велика кількість інших бібліотек [13].

Мова має продуману систему для об'єктно-орієнтованого програмування, а також підходить для написання сценаріїв (скриптів) [2]. Сфери застосування мови Python зображені на рисунку 1.5.

Мова програмування може бути використана у різних проєктах та грати різні ролі [4]. Наприклад, може бути основною, а також може бути корисною для реалізації інтеграції застосунків та розширень. У створенні прототипів програм Python теж грає велику роль.

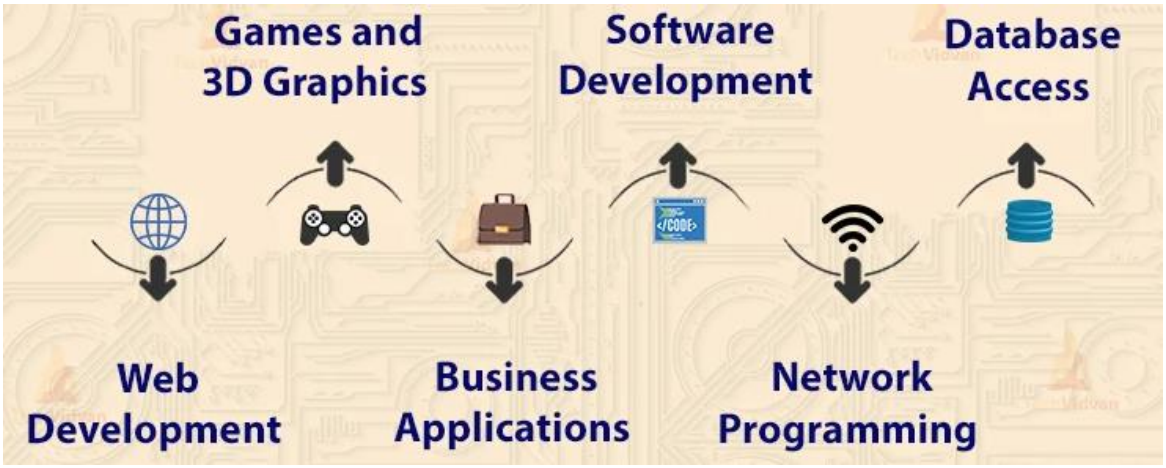

Рисунок 2.1 – Можливості Python

Слід зазначити, універсальність дає змогу використовувати мову для вирішення завдань на багатьох платформах, включаючи iOS, Android, Windows та серверні ОС [11]. Важливою властивістю мови є те, що Python має англомовний синтаксис, що суттєво спрощує читання та розуміння коду. Сфера застосування цієї мови є доволі широкою: розробка онлайн та мобільних застосунків, створення ігор, автоматизація математичних розрахунків та машинне навчання. Його можна використовувати для вирішення будь-яких завдань. Легке читання коду є важливою характеристикою Python.

Під час реалізації високонавантажених проектів, над якими працюють відразу кілька розробників, саме читабельність коду відіграє вирішальне значення, тому що кожен програміст повинен знати, що роблять члени його команди. Мова виставляє суворі вимоги до оформлення коду та влаштована таким чином, щоб мінімізувати кількість рядків. Тому читати її код завжди зручно та легко.

Мова Python має великий фан-клуб розробників та підтримується великими компаніями IT-індустрії, серед яких Google, Spotify та Facebook.

Важливо сказати, що мова має велику базу бібліотек [25]. Працюючи з Python, програміст не залежить від зовнішніх бібліотек. Мова оснащена потужною стандартною базою функцій, які можна залучати під час вирішення комерційних завдань, а це, в свою чергу, значно спрощує роботу. База бібліотек мови зображена на рисунку 1.6.

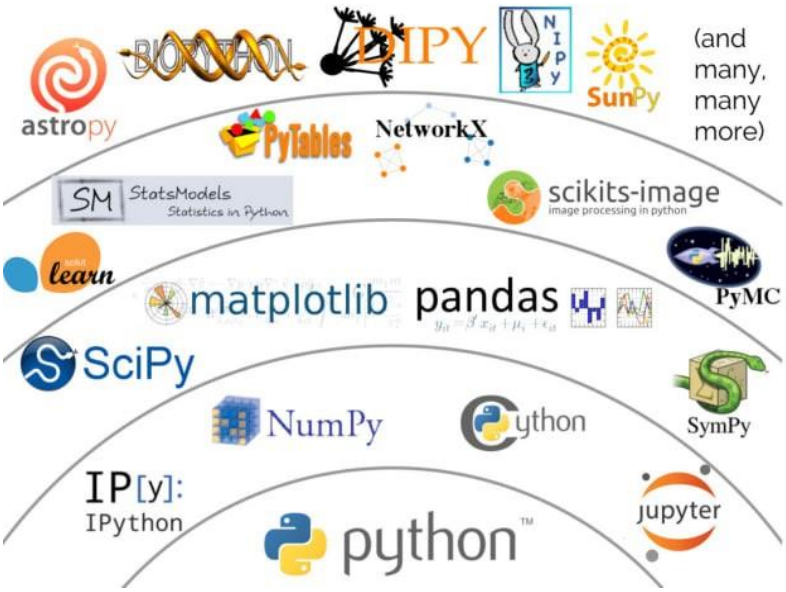

Рисунок 2.2 – Бібліотеки Python

Портативність грає важливу роль під час використання Python. У багатьох мовах програмування для запуску програми на різних платформах потрібно вносити зміни в код, але Python працює іншим чином. Написавши код один раз, можна запускати його на будь-яких пристроях без додаткових змін.

Є велика кількість сфер, де використовується ця мова. Серед них – веброзробка. Існує велика кількість фреймворків для розробки: Pylons, TurboGears, Pyramid, CherryPy, Flask. Найпопулярнішим можна вважати

Django [18]. Python може бути використаний для розробки сайтів за допомогою готових CMS, серед яких: Abilian SBE, Django-CMS, Wagtail, Ella, Saleor.

Також мова використовується під час розробки парсерів. Це дає можливість автоматизувати збір даних в Інтернеті [17].

Python використовується в популярних сайтах та компаніях, таких як YouTube, Google, Facebook, Intel та NASA.

Мова застосовується для розробки десктопних програм [5]. Можна виділити наступні програми: Blender (створення тривимірної графіки), Calibre (перегляд та конвертація електронних книг різних форматі), GIMP (графічний редактор для Linux).

Python застосовується в мобільній розробці. Зазвичай він використовується для створення серверної частини. Як приклад, клієнт популярної соцмережі Instagram написаний на Objective-C, а серверна частина – на Python.

Зараз штучний інтелект грає важливу роль в IT сфері. Для роботи з ним, нейромережами та машинним навчанням найчастіше використовується Python. Він, в свою чергу, може застосовуватися як основна мова або для реалізації окремих модулів.

Для розробки ігор також використовується Python [21]. Популярні комп'ютерні ігри використовують мову для реалізації скриптів, які відповідають за обробку різних подій, запуск сцен та взаємодію між персонажами.

Python часто використовується для створення вбудованих систем – це програмне забезпечення, інтегроване у різні фізичні пристрої. Наприклад, програмне забезпечення на Python використовується в Raspberry Pi та в банкоматах [22].

Існує багато галузей, де використовується ця мова. Це й наукові дослідження. Python простий в засвоєнні, володіє бібліотеками, призначеними спеціально для досліджень, наприклад NumPy, SciPy та

Matplotlib. Мова застосовується для написання алгоритмів, що використовуються програмами, пов'язаними з машинним навчанням. Крім цих моментів, мова застосовується різними сервісами для хмарного зберігання, обробки даних та парсингу. Зокрема, Google використовує Python для індексації вебресурсів.

Мова має й мінуси. Python використовує динамічну типізацію, тобто перевірка типів змінних виконується під час виконання, а не в момент оголошення. Це спрощує роботу розробника та робить мову програмування більш гнучкою, але це збільшує споживання пам'яті та уповільнює роботу програми. Через те, що Python – інтерпретована мова, через це код виконується без попередньої компіляції машинною мовою. Тому швидкість такого виконання суттєво поступається іншим мовам, що компілюються (Objective-C, C++,). Також Python – високорівнева мова, вона схожа на людський, ніж на машинний, тому процес обробки виконується довше.

Мова Python підходить для розробки засобу з керування закладками, так як має багато бібліотек та фреймворків. Мова дозволяє розробникам створювати програми, які працюватимуть на різних операційних системах, таких як Windows, macOS і Linux. Це допомагає розширити аудиторію для застосунку. Python підтримує численні бібліотеки для роботи з базами даних, такі як MySQL, SQLite, PostgreSQL, тощо. Все це спияє ефективному та безпечному зберіганню закладок [16].

2.2 Фреймворк Flask

Для поточного проєкту використовується мікрофреймворк Flask. Він не вимагає спеціальних бібліотек та засобів [6]. Перевірки форм, рівень абстракції для роботи з базою даних, інші компоненти відсутні. Фреймворк був розроблений у 2010 році Арміном Ронакером. Flask забезпечує: вбудовану підтримку юніт-тестів, сервер для розробки, а також відлагоджувач, управління запитами RESTful, підтримку безпечних

куків (сесії на стороні клієнта), використання шаблонів jinja2, докладну документацію, підтримка Unicode, розширення для бажаної поведінки, сумісність з Google App Engine [7]. Flask є гарним рішенням у випадку, коли програма, над якою ведеться розробка, спочатку буде не дуже великою, але у перспективі буде швидко зростати.

Гнучкість – одна з основних переваг фреймворку. Це корисно, тому що проєкт можна буде розвивати в іншому напрямку, а також тому що при певних змінах структура не буде порушена [24]. Основні переваги фрейворку предсталвені на рисунку 2.3.

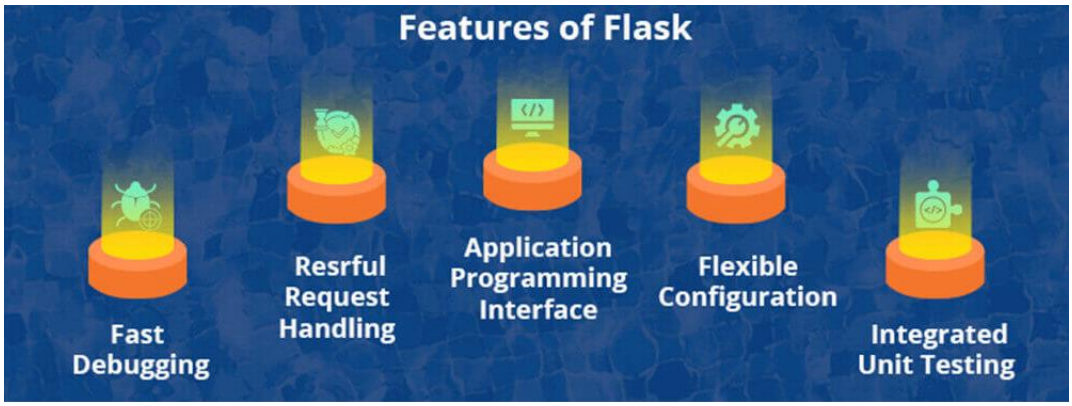

Рисунок 2.3 – Особливості Flask

Слід зазначити, що Flask підтримує модульне програмування, тобто функціональність може бути розділена на кілька модулів. Вони, в свою чергу, – взаємозамінні. Можна сказати, що кожен модуль – це будівельний блок, що реалізує певну частину можливостей. Всі складові структури є гнучкими [26].

Приставка «мікро» не означає, що вебзастосунок, побудований на Flask, повинен поміщатися в один файл з кодом на Python. «Мікро» означає, що Flask дотримується простого, але розширюваного ядра. Це означає, що мікрофреймворк не вирішує за програміста, що та як йому робити. При проектуванні вебзастосунків на Flask програміст всі рішення приймає

самостійно: наприклад, які патерни використовувати, чи використовувати ORM, або працювати на чистому SQL, яку базу даних підключити [27]. Flask не містить великих шарів абстракції та саме тому є швидким.

Фреймворк Flask підтримує багато розширень додавання різної функціональності, які забезпечують з валідацію форм, інтеграцію базами даних, різні технології аутентифікації. Фреймворк має безліч параметрів конфігурації з розумними значеннями за замовчуванням та мало попередніх угод.

До переваг Flask можна віднести:

- мінімалістичний дизайн і простий синтаксис, що спрощують початок роботи та розробку;
- підходить для проектів, де не потрібен повноцінний стек функцій Django;
- гнучкий, легкий фреймворк, що дає змогу розробникам обирати тільки ті інструменти, які є необхідними;
- вбудований дебаггер;
- власний сервер;
- безліч шаблонів і готових рішень.

Flask дозволяє розробляти безліч застосунків, при цьому швидкість розробки більша за рахунок коробкових рішень. Популярність Flask зумовлена кількома факторами. Це й мінімалістичний дизайн та легкість полегшують вивчення та використання, особливо для початківців. Крім того, легка природа Flask дозволяє швидко розробляти та розгортати. Широка документація та велика спільнота розробників сприяють його популярності, надаючи підтримку, плагіни та навчальні посібники.

Flask – це вебфреймворк. Його основне призначення – створення вебзастосунків. Використання Flask для розробки спеціалізованого програмного засобу для управління закладками має кілька переваг. По-перше, це – легкість та простота розробки, так як Flask дозволяє швидко

створювати RESTful API для взаємодії з даними, що спрощує розробку бекенд-частини програми. За допомогою розширень, таких як Flask-SQLAlchemy, можна легко інтегрувати роботу з базами даних. Flask легко масштабується та дозволяє додавати нові функції в міру необхідності.

Слід вказати, що, як приклад, одним з варіантів використання фреймворку може бути проєкт з наступної структурою. Перш за все, це – створення Flask-застосунка з основними маршрутами для додавання, видалення та перегляду закладок. Далі слід виконати налаштування бази даних для зберігання інформації. Процес розробки API буде складатися з певної кількості етпапів. Слід визначити RESTful API для взаємодії з фронтендом. Далі реалізувати маршрути для операцій. Потім потрібно створити інтерфейс користувача. Наприклад, можна використати AJAX для асинхронної взаємодії з бекенд-частиною на Flask. Це тільки один із варіантів використання фреймворку в проєкті.

2.3 Бібліотека Tkinter

Tkinter – це подієво-орієнтована графічна бібліотека. Назва походить від Tk interface. Tk (Toolkit) – бібліотека базових елементів графічного інтерфейсу. Tk застосунки – це програми, що керовані подіями. Бібліотека Tkinter була створена Стіном Лумхольтом та Гвідо ван Россумом. Слід зазначити, що Tkinter входить у стандартну бібліотеку Python [10].

Бібліотека забезпечує конвертування звернень Python у звернення Tcl, тобто мови, що тісно інтегрована з Tk. Tkinter не реалізує власний інтерфейс до бібліотеки Tk. Можна сказати, що Tkinter – обгортка для Tcl/Tk. Бібліотека забезпечує організацію діалогів у програмі, використовуючи віконний графічний інтерфейс.

Склад бібліотеки містить загальні графічні компоненти, а саме:

- frame (рамка): забезпечує групування віджетів та містить інші візуальні компоненти;

- toplevel/Tk: вікно верхнього рівня;
- label (етикетка): відображає графічне зображення чи певний текст;
- text: форматоване поле введення тексту, яке дозволяє показувати, форматувати, редагувати текст за допомогою різних стилів, впроваджувати в текст вікна, малюнки;
- entry: однорядкове поле для введення тексту;
- button (кнопка): кнопка для виконання команди, інших дій;
- canvas (полотно): використовується для виведення графічних примітивів: лінії, еліпси, прямокутники, текст, вікна, зображення;
- checkbutton (прапорець): має можливість множинного вибору, надаючи окрему змінну на кожен екземпляр віджету;
- radiobutton (перемикач): одне з альтернативних значень деякої змінної. Коли користувач вибирає будь-яку опцію, з обраного в цій же групі елемента вибір знімається;
- scale (шкала з повзунком): допомагає задати числове значення шляхом переміщення двигуна;
- scrollbar (смуга прокручування): може використовуватися разом з деякими іншими компонентами для їх прокручування;
- menu (меню): використовується для організації спливаючих та спадаючих меню;
- listbox (список): список, з якого користувач може виділити один або кілька елементів;
- message (повідомлення): дозволяє загортати довгі рядки, легко змінює свій розмір;
- menubutton (кнопка-меню): використовується для організації pulldown-меню та інші.

Tkinter – це популярна бібліотека для створення графічних інтерфейсів користувача в мові програмування Python [28]. Вона є стандартною бібліотекою Python, тобто входить у стандартну поставку Python та не

потребує додаткової установки. Це робить Tkinter зручним для використання, так як немає потреби в додаткових залежностях [12]. Інтерфейс користувача Tkinter зображений на рисунку 2.4.

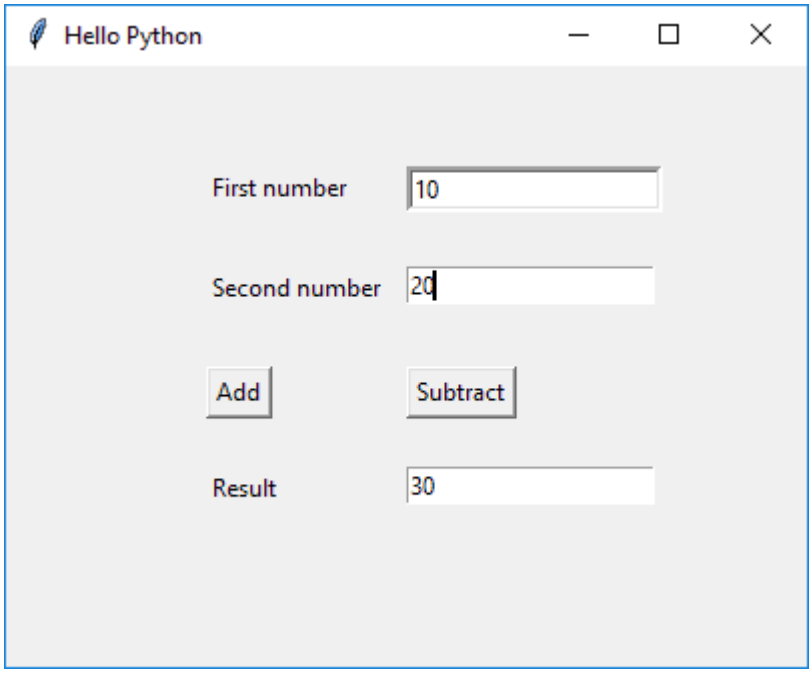

Рисунок 2.4 – Приклад інтерфейсу

Завдяки своїм перевагам Tkinter підходить для реалізації інтерфейсу користувача програмного засобу з управління закладками. Графічні компоненти відповідають потребам проєкта, які пов’язані зі взаємодією з користувачем. Інші бібліотеки такого плану, такі як wxPython та PyQT не є частиною стандартної бібліотеки Python. Tkinter не потребує додаткових налаштувань для адаптації під різні ОС на відміну від wxPython. Завдяки тому, що Tkinter вбудований у Python, ця бібліотека є простою в розгортанні.

2.4 Система керування реляційними базами даних MySQL

MySQL – це вільна система керування реляційними базами даних. Ця система була розроблена як альтернатива комерційним системам.

Система керування базами даних (СКБД) розроблена компанією «ТсХ» для підвищення швидкодії обробки великих баз даних. Перший запуск відбувся у 1995 році. Ця СКБД має підтримку з боку різних мов програмування та використовується для створення динамічних вебсторінок [9].

Слід зазначити, що MySQL – це багатопотоковий сервер баз даних, який має простоту використання, а також високу швидкість. Для проєктів, які є не дуже великими чи середніми підходить саме MySQL. Якщо говорити про UNIX-системи, то там спостерігається підвищення продуктивності, так як саме у такому випадку розкриваються усі можливості сервера. Все це відбувається за рахунок багатопоточності.

До можливостей сервера MySQL можна віднести:

- наявність ефективної та простої системи безпеки;
- кількість рядків у таблицях може досягати 50 млн;
- підтримується необмежена кількість користувачів;
- простота у встановленні, а також використанні;
- висока швидкість виконання команд.

Цю СКБД використовують багато великих компаній. Серед них Google, Joomla!, NASA, Nokia, Apple, Amazon.com, WordPress, MediaWiki та інші. MySQL входить до складу наступних серверів:

- LAMP;
- WAMP;
- AppServ.

У портативні зборки серверів:

- XAMPP;
- Денвер.

Підтримка великої кількості типів таблиць робить СКБД MySQL

гнучкою. MySQL також можна запускати на платформах хмарних обчислень, таких як Amazon Elastic Compute Cloud, Microsoft Azure, Oracle Cloud Infrastructure [14]. Слід зазначити, що є можливість графічно керувати базами даних, а також візуально проєктувати структури баз даних за допомогою MySQL Workbench. Інтерфейс MySQL Workbench представлений на рисунку 2.5.

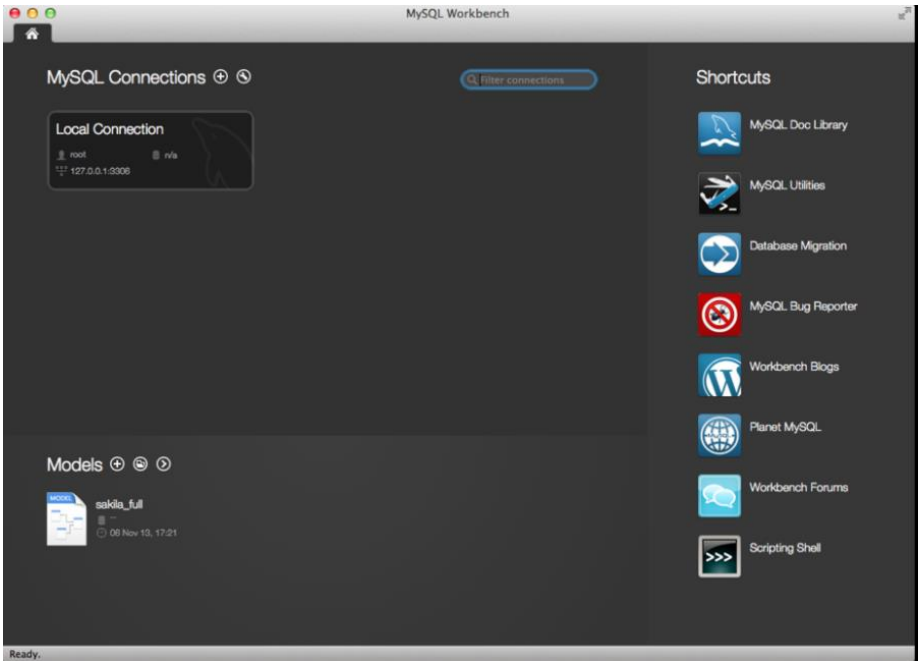

Рисунок 2.5 – MySQL Workbench для macOS

Також існує інтерфейс командного рядка. У такому випадку користувач віддає команди програмі, вводить послідовні рядки тексту. Слід зазначити, що існує оболонка MySQL. Це інструмент, який служить для інтерактивного використання, а також адміністрування БД. Інструмент підтримує режими JavaScript, Python чи SQL і може використовуватися для доступу, адміністрування.

Цю систему керування реляційними базами даних можна створити та встановити вручну з вихідного коду. Частіше встановлюється з двійкового пакета. Linux система керування пакетами може завантажити та встановити

MySQL. Для цього будуть потрібні лише невеликі зусиллями (у більшості дистрибутивів).

MySQL забезпечує надійність у роботі та високу швидкість обробки даних, що важливо для програм, які зберігають і обробляють велику кількість даних. Саме тому MySQL підходить для розробки менеджера закладок. Ця СКБД дозволяє швидко виконувати запити та гарантує збереження даних. MySQL підтримує масштабованість як на вертикальному, так і на горизонтальному рівнях: застосунок може обробляти великі обсяги даних та збільшувати свої потужності в разі зростання кількості даних чи користувачів. СКБД підтримується багатьма мовами програмування та фреймворками.

Бібліотеки, такі як SQLAlchemy чи mysql-connector-python, дозволяють легко інтегрувати MySQL в десктопні застосунки. СКБД пропонує розширені функції безпеки: шифрування даних, контроль доступу на основі ролей, та захист від SQL-ін'єкцій. Це грає важливу роль для забезпечення цілісності даних закладок та конфіденційності.

Слід зазначити, що MySQL – СКБД з відкритим вихідним кодом та доступний безкоштовно. Це економічно вигідне рішенням для розробки програми. Важливо відмітити, що MySQL дозволяє налаштовувати різні параметри відповідно до вимог програми, що забезпечує оптимальну продуктивність.

2.5 Інтеграція штучного інтелекту

У програмному забезпеченні з керування закладками штучний інтелект грає важливу роль для генерації контенту, який безпосередньо пов’язаний з інтересами користувача. До такого роду матеріалу відносяться як статті, так і відео. Для цього був використаний OpenAI API.

OpenAI API – це інтерфейс програмування застосунків. Він надає доступ до розумових моделей штучного інтелекту, розроблених OpenAI. API

дозволяє розробникам, а також компаніям інтегрувати потужні функціональності мовного моделювання, штучного інтелекту в свої програми, сервіси, продукти.

OpenAI API використовується у наступних випадках:

- генерація контенту (OpenAI API може бути використаний для створення текстового, візуального та аудіо контенту);
- мовна генерація (OpenAI володіє мовними моделями. Вони здатні створювати граматично вірний та семантично багатозначний текст);
- рекомендації та схожий матеріал (можна використовувати OpenAI API для отримання рекомендацій та знаходження схожих сторінок за запитами користувачів).

OpenAI API має низку переваг. Серед них потужність моделей, так як OpenAI використовує великі, передові мовні моделі, які навчені на величезних обсягах даних. Це дозволяє створювати високоякісний та різноманітний контент. Також до переваг можна віднести гнучкість та адаптивність [29].

API надає гнучкий інтерфейс, який можна адаптувати до різноманітних вимог та сценаріїв використання. OpenAI API надає широкі можливості. Цей інструмент може бути використаний в різних сферах, таких як робота з мовою, створення контенту, розробка продуктів, та інше.

Слід зазначити, що штучний інтелект дозволяє API надавати більш інтелектуальні та нестандартні рішення.

OpenAI API грає важливу роль у розробленому спеціалізованому програмного засобі для управління закладками в інтернет-ресурсах, так як дозволяє генерувати унікальний контент для кожного користувача. Матеріал, який надається, базується тільки на індивідуальних перевагах, що дозволяє отримувати актуальні статті, сайти, відео тощо.

API можна налаштовувати для специфічних завдань щодо рекомендації контенту. Як приклад, можна використовувати попереднє навчання на основі

набору даних з закладками для підвищення точності рекомендацій [20]. Слід зазначити, що моделі OpenAI постійно оновлюються, що забезпечує високу актуальність та точність рекомендацій. Використання останніх моделей дозволяє користувачам отримувати найсвіжіші рекомендації.

2.6 Інтегроване середовище розробки PyCharm

PyCharm – це інтегроване середовище розробки для Python. PyCharm було розроблене компанією JetBrains. Перший випуск був у 2010 році.

Середовище дозволяє аналізувати код, розробляти на Django, використовувати інструмент для запуску юніт-тестів, надає графічний відлагоджувач. Інтерфейс середовища зображений на рисунку 2.6.

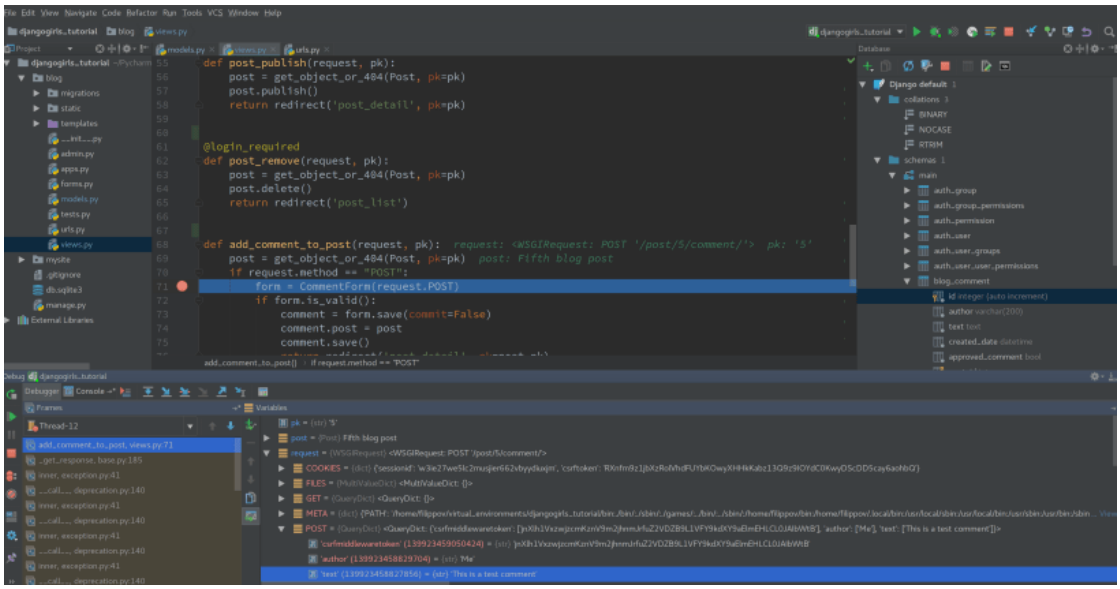

Рисунок 2.6 – PyCharm

Важливо вказати, що PyCharm може використовуватись на різних операційних системах: Windows, macOS, Linux. Також середовище забезпечує навігацію серед проєктів, рефакторинг (редагування коду), розробку з використанням Google App Engine (сервіс хостингу сайтів, а також web-аплікацій).

Слід зазначити, що використовуючи плагіни, які користувачі можуть писати самостійно, є можливість розширити певні функції цього середовища.

PyCharm має вбудовану підтримку роботи з базами даних: MySQL, PostgreSQL, SQLite. Це дозволяє легко підключатися до БД програмного застосунку з управління закладками, керувати даними безпосередньо з середовища розробки, виконувати SQL-запити. PyCharm підтримує інтеграцію з Git, SVN, Mercurial, та іншими системами контролю версій. Це дає змогу відстежувати зміни у коді та зберігати історії проєкту. Середовище має вбудовану підтримку для написання та виконання тестів, що дозволяє забезпечити якість коду.

Є можливість використовувати бібліотеки для тестування, щоб перевірити функціонування програми. PyCharm надає потужні інструменти для рефакторингу коду, що дозволяє змінювати структуру коду без втрати його функціональності. Це дає змогу підтримувати організованість коду та чистоту в великому проєкті. Для програмного застосунку з управління закладками, який має вебінтерфейс, PyCharm пропонує розширену підтримку веброзробки, а саме фреймворки, такі як Flask та Django.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Структура програмного застосунку

Створений проєкт побудований за архітектурним підходом MVVM (Model-View-ViewModel). Цей шаблон проєктування полегшує відокремлення розробки графічного інтерфейсу від розробки бізнес логіки, відомої як модель. Можна розділити підхід на 3 окремі частини:

- Модель (Model), певні дані, які необхідні для роботи програми;
- Вигляд (View), інтефейс програми, кнопки та інші елементи;
- Модель вигляду (ViewModel, можна розшифрувати, як «Model of View»), містить частину Модель, що є перетвореною до Вигляду, та певні команди для Вигляду для керуванням Моделі. Ця частина відслідковує зміни в даних, які робе користувач, здійснює зв'язок між вікном та моделлю та забезпечує логіку роботи View.

Загальна схема Model-View-ViewModel архітектури представлена на рисунку 3.1.

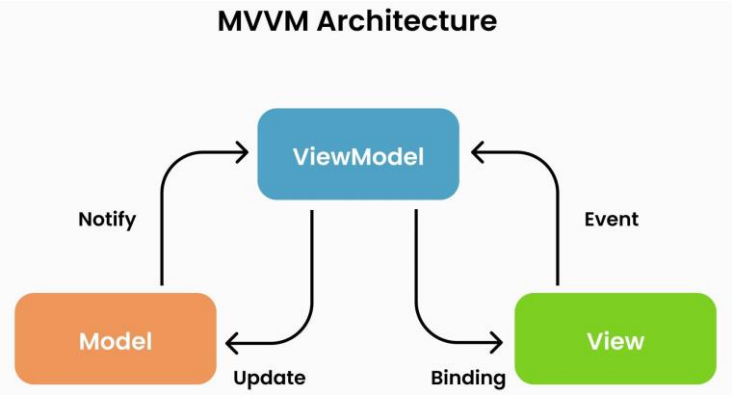

Рисунок 3.1 – MVVM

Структура розробленого програмного застосунка відповідає підходу Model-View-ViewModel. Його частини можна представити у виглядів шарів.

Компоненти проэкта представлені на рисунку 3.2.

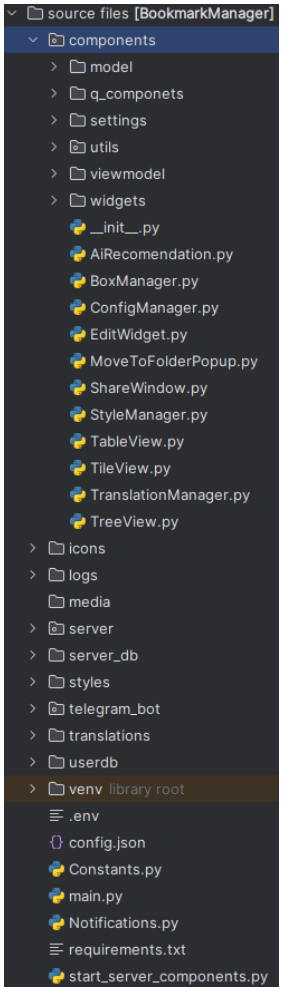

Рисунок 3.2 – Структура проєкту

Шар Моделі (Model) містить такі компоненти, як server_db, userdb, utils. Ця частина відповідає за бізнес-логіку та керування даними. Шар відповідає за взаємодію з базою даних та забезпечення необхідною інформацією ViewModel.

Шар Моделі вигляду (ViewModel) містить наступні компоненти: viewmodel, BoxManager.py, ConfigManager.py. Ця частина є посередником між View та Model. Шар обробляє дані з Моделі та поітм надсилає команди до View для оновлення інтерфейсу користувача.

Шар Подання (View) включає в себе такі компоненти, як widgets, icons,

media та styles. Ця частина містить усі UI-компоненти: кнопки, вікна та стилі, що виконують взаємодію з користувачем. Будь-які зміни в інтерфейсі надсилаються назад до ViewModel для здійснення обробки.

3.2 Структурна частина Model

Частина Model відповідає за збереження даних, що необхідні для роботи програми. Це дані, які стосуються користувача та сервера. Присутні функції для перевірки синтаксису URL, доступності інтернет-з’єдання та URL, а також клас для керування рівнями важливості.

Імпортуються бібліотека re, яка є модулем для роботи з регулярними виразами, який потрібен саме для перевірки посилання. Також присутній модуль threading, завдяки якому відбувається робота з потоками. Для відправки HTTP-запитів використовується бібліотека requests. Обробка помилок відбувається завдяки модулю urllib3, який відповідає за роботу з HTTP-запитами.

Для керування рівнями важливості реалізується клас ImportanceLevelsClass. Конструктор ініціалізує два атрибута: self.levels (список рядкових значень, які представляють рівні важливості) та self.levels_ints (словник, який зіставляє кожному рівню числове значення). Метод `convert()` приймає рівень важливості (рядок) та потім повертає його числове уявлення, використовуючи словник levels_ints. Якщо не нічого знайдено, повертає -1.

Для підвищення рівня важливості на один крок віще у списку (levels_ints) використовується метод `increase()`. Якщо рівень не знайдено (виключення ValueError), повертається green. Для пониження важливості на один крок використовується метод `decrease()`. Якщо рівень вже мінімальний (green), повертає сам рівень. При некоректному сначені рівня повертає green. Створюється екземпляр класу ImportanceLevelsClass, який буде відповідати за керування рівнями важливості. Реалізація класу

ImportanceLevelsClass представлена на лістингу 3.1.

Лістинг 3.1 – Клас ImportanceLevelsClass (файл __init__.py)

```
Class ImportanceLevelsClass:
    def __init__(self):
        self.levels = ['green', 'yellow', 'orange', 'red']
        self.levels_ints = {
            'gray': 0,
            'green': 1,
            'yellow': 2,
            'orange': 3,
            'red': 4
        }
    def convert(self, level):
        return self.levels_ints.get(level, -1)
    def increase(self, level):
        try:
            index = self.levels.index(level)
            if index < len(self.levels) - 1:
                return self.levels[index + 1]
            else:
                return level  # If 'level' is already the
highest, return itself
        except ValueError:
            return 'green'  # Return None if 'level' is not in
the list
    def decrease(self, level):
        try:
            index = self.levels.index(level)
            if index > 0:
                return self.levels[index - 1]
            else:
                return level  # If 'level' is already the
lowest, return itself
        except ValueError:
            return 'green'  # Return None if 'level' is not in
the list
```

Перевірка синтаксису URL відбувається завдяки функції `check_url_syntax()`. Вона використовує регулярний вираз для перевірки чи є наданий URL валідним. Схема може бути наступною: http, https чи ftp. Далі йде домен чи IP-адреса, порт (необов’язковий), шлях, строка запиту та фрагмент. Повертається True, якщо URL відповідає шаблону, інакше повертається False.

Перевірка інтернет-з’єднання реалізується через функцію `check_internet_connection()`, яка перевіряє підключення до мережі Інтернет завдяки відправленню запиту на публічний DNS-сервер (1.1.1.1). При умові, що відповідь успішна, повертається True. Якщо є виключення (випадок, коли з’єднання не встановлено), тоді повертається False.

Доступність URL відбувається за допомогою функції `check_url_availability()`. Відправляє HTTP-запит на надісланий URL зі вказаними заголовками. Якщо код відповіді 404, тоді повертається «not_found». Якщо відповідь 200, тоді повертається «ok».

Реалізована перевірка різних виключень, які залежать від типу помилки. ConnectionError відповідає за перевірку інтернет-з’єднання та повертає помилки no_internet та connection_error. Якщо URL не містить схему (наприклад, http://), тоді повертається «no_schema» (MissingSchema). LocationParseError, InvalidURL, Timeout відповідають помилкам «connection_error» чи «timeout_error».

Реалізується потік для перевірки URL-адрес з використанням бібліотеки PyQt6. Вона створює потік, який взаємодіє з об’єктом manager для перевірки адрес та відправки результату. Відбувається імпорт модуля time, що надає певні функції для роботи з часом (як приклад, sleep для паузи в виконанні програми). Імпортуються класи QThread и pyqtSignal з модулю QtCore бібліотеки PyQt6. Слід зазначити, що QThread використовується для створення потоків, а pyqtSignal – для визначення користувацьких сигналів, які можуть буті відправлені з потоку. Визначається певний клас class UrlCheckThread(QThread). Він успадковується від QThread та його робота буде проходити в окремому потоці. Створюється сигнал resultReady, що відправляє об’єкт в інший компонент програми, коли результат готовий. Слід сказати, що сигнал буде передавати об’єкт типу object (будь-який) тип даних. `def __init__(self, manager)` визначає конструктор класу UrlCheckThread, що приймає об’єкт manager. Відбувається збереження надісланий manager в атрибуті екземпляра класу для використання в інших

методах. Відбувається виклик конструктора базового класу QThread, щоб ініціалізувати потік. Метод `def run(self)` є основним методом потоку. Він виконується, коли потік запускається. Все, що відбувається в методі `run()` виконується в окремому потоці та блокування основного потоку програми не відбувається.

Вхід у цикл (`while self.manager.WAIT`), який чекає, поки флаг WAIT у об'єкті manager стане False чи 0. Щоб не перенавантажувати процесор використовується пауза на 0.1 секунди (`time.sleep(0.1)`). Як тільки потік виходить з циклу очікування, встановлюється флаг WAIT в True, щоб запобігти запуск інших потоків до закінчення поточної задачі.

Викликається метод `check_address()` об'єкта ITEM_EDIT для перевірки адреси та збереження результату в змінну result. За допомогою `self.resultReady.emit(result)` відправляється сигнал resultReady (передається результат result). Це дає змогу іншим компонентам програми (віджети та інше) отримати повідомлення про те, що відбулось завершення перевірки адреси, та що можна починати роботу з результатом. Використовуючи `self.manager.WAIT = False`, скидається флаг WAIT у False та іншим потокам дозволяється продовжувати свою роботу. Реалізація класу class UrlCheckThread(QThread) представлена на лістингу 3.2.

Лістинг 3.2 – Клас class UrlCheckThread(QThread) (файл UrlCheckThread.py)

```
import time
from PyQt6.QtCore import QThread, pyqtSignal
class UrlCheckThread(QThread):
    resultReady = pyqtSignal(object)
    def __init__(self, manager):
        self.manager = manager
        super().__init__()
    def run(self):
        while self.manager.WAIT:
            time.sleep(0.1)
        self.manager.WAIT = True
        result = self.manager.ITEM_EDIT.check_address()
        self.resultReady.emit(result)
        self.manager.WAIT = False
```

Імпортуються бібліотеки для роботи з файлами та даними (формат JSON), для графічного інтерфейсу та компонент для взаємодії з конфігураційними файлами. Описується функція `create_folder(folder_path)`, яка виконує перевірку на те, чи існує папка за вказаним шляхом. Якщо папка не існує, вона створюється за допомогою `os.makedirs`. Це гарантує, що база даних користувачів матиме своє місце для зберігання файлів.

Для роботи з базою даних користувача відповідає клас Database. Він використовує файл для збереження, а також завантаження даних з бази у вигляді JSON-файлів. Конструктор `__init__()` створює новий екземпляр класу Folder. Завантажуються дані користувача з відповідного JSON-файлу, шлях до якого залежить від логіна користувача. Присутні методи для збереження, завантаження та взаємодії з деревом даних.

Клас Folder моделює папку з елементами (папки чи файли) та зберігає інформацію про назву, дату створення та певні властивості. Присутній метод для серіалізації об'єкта в JSON, а також метод для зчитування даних з JSON.

Клас Item відповідає за моделювання окремого елементу (файлу), який має назву, тег, адресу, опис, властивості. Клас представляє один елемент у базі даних. Item має кілька атрибутів, таких як ім'я, тег, опис та адреса.

Конструктор `__init__()` ініціалізує новий об'єкт Item з name (назва елемента; за замовчуванням порожній рядок), tag (тег, який описує елемент; за замовчуванням «normal»), address (адреса елемента, наприклад, URL; за замовчуванням «example.com»), description (опис елемента; за замовчуванням порожній рядок), parent (батьківський елемент, зазвичай це папка, що містить елемент; за замовчуванням None). Встановлюється значення атрибуту name, tag, address, parent, для екземпляра Item. Ініціалізується порожній список категорій та відбувається встановлення атрибуту icon для відображення елемента. Встановлюється значення атрибуту importance, що вказує на рівень важливості елемента. Ініціалізується атрибут photo, який зберігає зображення, асоційоване з елементом. Відбувається встановлення дати. За

замовчуванням стоїть 0 (не встановлена). Призначається опис для елемента.

Встановлюється тип даних для елемента як 'item'. Це використовується для ідентифікації типу об'єкта під час серіалізації. Відбувається ініціалізується порожнього словнику для збереження проміжних даних. Метод `get_cache`() повертає значення з кешу за вказаним ключем. Якщо ключ відсутній, у такому випадку повертається None. Метод set_name(): встановлює нове значення назви елемента. Відбувається процес оновлення тексту в кешованому об'єкті інтерфейсу.

Викликається збереження бази даних через `db.save()` після зміни. За допомогою методу `set_tag()` встановлюється нове значення для тега та відбувається збереження зміни в базі даних. Метод `set_address()` встановлює нову адресу для елемента, а тьакож зберігає зміни в базі даних. Новий опис для елемента та збереження зміни в базі даних відбуваються завдяки методу метод set_desc(). Метод `to_json()` серіалізує елемент у словник для перетворення у формат JSON. Включає всі основні атрибути елемента.

Метод `from_json()` завантажує дані з JSON-об'єкта та призначає відповідні атрибути Item. Кожен ключ у JSON відповідає атрибуту класу. Слід зазначити, що для кожного атрибуту задається значення за замовчуванням, якщо відповідного ключа немає в JSON. Метод `copy()` створює копію поточного об'єкта Item.

Спочатку серіалізує поточний об'єкт у JSON, після чого створює новий об'єкт Item, завантажуючи в нього дані з JSON. Реалізація класу Item представлена на лістингу 3.3.

Лістинг 3.3 – Клас Item (файл Database.py)

```
class Item:
    def __init__(self, name='', tag='normal',
address='example.com', description="", parent=None):
        self.name = name
        self.tag = tag
```

```
        self.address = address
        self.parent: Folder = parent
        self.categories = []
        self.icon = 'globe.png'
        self.importance = 'green'
        self.photo = None
        self.date = 0
        self.description = description
        self.data_type = 'item'
        self.cache = {}
    def get_cache(self, key):
        return self.cache.get(key)
    def set_name(self, value):
        self.name = value
        self.cache["q_item"].setText(value)
        db.save()
    def set_tag(self, value):
        self.tag = value
        db.save()
    def set_address(self, value):
        self.address = value
        db.save()
    def set_desc(self, value):
        self.description = value
        db.save()
    def to_json(self):
        return {'name': self.name, 'tag': self.tag, 'address':
self.address, 'description': self.description,
                'data_type': self.data_type, 'categories':
self.categories, 'icon': self.icon,
                'importance': self.importance, 'photo':
self.photo, 'date': self.date}
    def from_json(self, json_data):
        self.name = json_data.get('name', '')
        self.tag = json_data.get('tag', '')
        self.address = json_data.get('address', '')
        self.description = json_data.get('description', '')
        self.categories = json_data.get('categories', [])
        self.icon = json_data.get('icon', self.icon)
        self.importance = json_data.get('importance',
self.importance)
        self.photo = json_data.get('photo', self.photo)
        self.date = json_data.get('date', 0)
        return self
    def copy(self):
        copy = Item()
        copy.from_json(self.to_json())
        return copy
```

Для реалізації рекомендацій використовується діалогове вікно завантаження, використовуючи PyQt6 та API OpenAI. За допомоги функції

`normalize_url()` відбувається створення стандартного формату, додаючи певну схему (http чи https), якщо така відсутня. Перевірка доступності URL відбувається завдяки функції `check_url_result()`, яка викликає іншу функцію з utils, щоб перевірити доступність URL-адреса (через HTTP-запит або інші методи).

Клас AiLoadingPopup – це основне вікно для користувача. Воно містить текстове поле, кнопку для запуску пошуку рекомендованого контенту, а також панель прогресу. Після натискання кнопки старту пошуку створюється певний потік для отримання рекомендацій, а панель прогресу починає відображати процес.

Потік GetRecommendationThread відповідає за те, щоб не блокувати основний інтерфейс при отриманні контенту від API. Викликається функція `get_by_url()`, що надсилає запит до OpenAI на отримання схожих ресурсів. За отримання рекомендацій відповідає функція `get_by_url()`, що робить запит до OpenAI для доступу до списоку рекомендацій у текстовому форматі.

Результат потім передається в основне вікно для відображення. Обробка результату відубувається після завершення запиту до API та інформація відображається у вікні. Регулярний вираз використовується для пошуку URL у тексті відповіді. Слід зазначити, що для кожного URL створюються текстові поля та кнопки. Вони дозволяють скопіювати посилання до буфера обміну. Передбачається обробка можливих помилок під час роботи з API та при спробах аналізувати отримані відповіді.

Один з реалізованих класів – GetRecommendationThread. Він використовується для асинхронного отримання рекомендацій на основі URL. Клас GetRecommendationThread успадковується від класу QThread. Це дозволяє виконувати код у фоновому потоці (не відбувається блокування основного інтерфейсу користувача). Визначається сигнал resultReady. Це особливий сигнал PyQt. Він буде випромінюватись після завершення потоку. Передається об'єкт як аргумент, де об'єкт – результат операції, наприклад, це може бути текстовий рядок з рекомендаціями. Ініціалізатор класу

(конструктор) приймає URL як аргумент. Присвоює значення змінній self.url. Воно буде використовуватися для отримання рекомендацій (це URL, для якого будуть шукатися подібні ресурси). Викликається конструктор базового класу QThread для правильної ініціалізації об'єкта потоку. Оголошується методу `run()`. Це основний метод, який виконується при умові, коли потік запускається. Слід сказати, що все, що знаходиться в цьому методі, виконується у фоновому режимі.

Викликається функція `get_by_url(self.url)`, яка надсилає запит до API на основі URL, а потім отримує результат (саме список рекомендацій). Після отримання результату, сигнал resultReady випромінюється разом із результатом (передає результат у основний інтерфейс). Метод `result()` відповідає за повернення результату виконання функції `get_by_url(self.url)`. Він може бути використаний, якщо результат потрібно отримати без запуску потоку чи в іншому випадку. Реалізація класу GetRecommendationThread представлена на лістингу 3.4.

Лістинг 3.4 – Клас GetRecommendationThread (AiRecomendation.py)

```
class GetRecommendationThread(QThread):
    resultReady = pyqtSignal(object)
    def __init__(self, url):
        self.url = url
        super().__init__()
    def run(self):
                self.resultReady.emit(get_by_url(self.url))
    def result(self):
        return get_by_url(self.url)
```

Діалогові вікна різних типів в PyQt6 створюються через клас BoxManager. Функція `make_box()` створює діалогове вікно типу QMessageBox з текстом, іконкою та заголовком (опціонально). Це універсальна функція для створення будь-якого типу вікна повідомлення.

Функція `getUserInput()` відкриває вікно для введення тексту користувачем. Вона повертає введений текст та певне булеве значення, що

вказує, чи було введення підтверджено користувачем.

Відображення інформаційного діалогового вікно (наприклад, для загальної інформації) реалізоване за допомогою функції `informationBox()`. Функція **`ErrorBox`**`()` відображає діалогове вікно з повідомленням про помилку, що показує критичну інформацію чи повідомляє про проблеми.

Одна з функцій потрібна вирішує питання відображення діалогового вікна попередження для сповіщення користувача про можливі ризики або проблеми реалізована функція `warningBox()`. Визначається метод `warningBox()`, який є частиною класу. Метод приймає параметр text (обов'язковий параметр, текст повідомлення, який буде відображено в вікні)

Також є параметр title (необов'язковий параметр, значення за замовчуванням None) – заголовок вікна. Викликається метод `make_box()`, який є частиною того ж класу. Він створює діалогове вікно з параметрами

text (текст повідомлення, що відобразиться у вікні), title (заголовком вікна, якщо він наданий) та QMessageBox.Icon.Warning (іконка діалогового вікна встановлюється на тип «попередження» – вказівка користувачеві на наявність потенційної проблеми.

Для ввідображення діалогового вікна на екрані викликається метод `exec()` та очікує взаємодії від користувача. Забезпечується те, що вікно буде модальним, тобто програма призупинить виконання цього потоку, поки користувач не закриє вікно. Реалізація методу `warningBox()` представлена на лістингу 3.5.

Лістинг 3.5 – Метод `warningBox()` (файл BoxManager.py)

```
def warningBox(self, text, title=None):
    self.make_box(text, title, QMessageBox.Icon.Warning).exec()
```

Для управління конфігураційним файлом у форматі JSON відповідає клас ConfigManager. У конструкторі приймається шлях до конфігураційного файлу, що зберігається як атрибут. Завдяки методу `update()` оновлюється

конфігураційний файл новими даними: спочатку відбувається зчитування існуючого вмісту, додаються нові дані та відбувається запис оновленого файлу назад. Метод `read_config()` відкриває та читатє конфігураційний файл. Якщо файл не знайдений, він створюється як порожній. У разі помилки в декодуванні JSON, відбувається вивід повідомлення про поточну проблему.

Один з присутніх методів – це метод `write_config()`. Він приймає два аргументи: self (для доступу до атрибутів та методів класу), а також config_data (дані, які потрібно записати в конфігураційний файл). У блоку try виконується спроба процесу запису даних у файл. Якщо виникає помилка, у такому випадку програма перейде до блоку except.

Відбувається відкриття файлу, шлях до якого зберігається в атрибуті self.config_file, у режимі запису.

Завдяки використанню with можна гарантувати, що файл буде автоматично закритий після завершення всіх дій.

Метод `json.dump()` використовується для запису об'єкта config_data у спеціальний файл у форматі JSON. Параметр indent=4 визначає, що кожен рівень вкладеності буде відформатований з відступом у саме в 4 пробіли (файл буде зручним для читання).

Можуть бути ситуації, коли при записі файлу виникає помилка вводу / виводу (файл неможливо відкрити для запису), виконання перейде до наступного блоку. При виникненні помилки, виводиться відповідне повідомлення про те, що не вдалося записати дані в конфігураційний файл, із вказаним шляхом до файлу. Реалізація методу `write_config()` представлена на лістингу 3.6.

Лістинг 3.6 – Метод `write_config()` (файл ConfigManager.py)

```
def write_config(self, config_data):
    try:
        with open(self.config_file, 'w') as file:
            json.dump(config_data, file, indent=4)
    except IOError:
        print(f"Couldn't write to the config file
```

```
'{self.config_file}'.")
```

Для реалізації перекладів у програмі була використана бібліотека gettext. Були налаштовані шляхи до файлів перекладів за допомогою `gettext.bindtextdomain` та `gettext.textdomain`. Це дозволяє програмі провести завантажувати файли перекладів з конкретного місця в системі. Клас Translation створює об'єкт для роботи з перекладами з налаштуванням мови інтерфейсу.

Відбувається зчитування поточної мови з конфігураційного файлу через клас ConfigManager та збереження її у властивості selected. Для створення попереджень користувачу (наприклад, про необхідність перезапустити програму після зміни мови) використовує клас BoxManager.

Зміна мови в програмі відбувається завдяки методу `change_language()`. Він приймає параметр language, що представляє нову мову для переключення.

Описуються перевірка на те, чи нова мова співпадає з уже вибраною (збереженою у властивості self.selected). У випадку, коли мова вже вибрана, метод припиняє виконання, щоб уникнути зайвої обробки. Відбувається зчитування поточного конфігураційного файлу через метод `read_config()` класу ConfigManager.

Дані з конфігурації зберігаються у змінну c (словник). Оновлюється конфігурація: додається чи змінюється значення ключа «language» на нову мову, що була передана в параметрі language.

Завдяки методу `write_config()` оновлені дані конфігурації записуються назад у конфігураційний файл.

Відбувається оновлення вибраної мови у внутрішньому об'єкті перекладів програми, зберігаючи нову мову в атрибуті `t.selected`.

Викликається метод `warningBox()` з попереджувальним повідомленням для користувача. Використовується функція l для отримання перекладу повідомлення «Зміни ввійдуть в силу після перезапуску» на

поточну мову. Реалізація методу `change_language()` представлена на лістингу 3.7.

Лістинг 3.7 – Метод change_language() (файл TranslationManager.py)

```
def change_language(self, language):
    if language == self.selected:
        return
    c = self.config.read_config()
    c.update({'language': language})
    self.config.write_config(c)
    t.selected = language
    self.box_manager.warningBox(l("Зміни ввійдуть в силу після
перезапуску."))
```

Описується клас QDatabaseItem, який успадковує від QStandardItem з бібліотеки PyQt6. Основне завдання класу полягає в тому, щоб представляти елементи БД, такі як файли чи папки, в графічному інтерфейсі користувача.

Метод `__init__()` (конструктор класу), який викликається при створенні нового екземпляра, приймає параметр data_item, який може бути або об'єктом типу Item, або Folder (за замовчуванням дорівнює None). Присутні також додаткові аргументи *args та **kwargs, які можуть бути передані. Він викликає конструктор батьківського класу, передаючи йому всі отримані аргументи, а також задає текст для елемента, використовуючи ім'я елемента з data_item.

Зберігається передане значення в екземплярі класу для подальшого використання. Якщо data_item є папкою, у такому випадку до тексту додається спеціальний символ для візуального позначення цього типу. Використовуючи метод `setText()`, задається текст для елемента. Він складається з назви data_item (яка є ім'ям елемента) та додаткового символу «---↓», якщо тип даних елемента є «folder».

У ситуації, коли поточний тип не «folder», у такому випадку нічого не додається. У класі визначено два атрибути: removable, що позначає, чи можна видаляти певний елемент, та q_type, що ідентифікує тип елемента як

«normal» (використовується для ідентифікації типу елемента у контексті програми). Реалізація методу `__init__()` представлена на лістингу 3.8.

Лістинг 3.8 – Метод `__init__()` (файл QDatabaseItem.py)

```
def __init__(self, data_item=None, *args, **kwargs):
    super().__init__(*args, **kwargs)
    self.data_item: Union[Item, Folder] = data_item
    self.setText(
        data_item.name +
        ("  ---↓" if data_item.data_type == 'folder' else '')
    )
    self.removable = True
    self.q_type = 'normal'
```

Метод `remove()` відповідає за видалення елементу зі списку, викликаючи відповідний метод моделі, пов'язаної з елементом, щоб видалити певний рядок.

Для збереження конфігурацій та змінних середовища імпортується модуль os, який надає функції для взаємодії з операційною системою, зокрема для роботи з певними файлами та директоріями. Імпортується функція `dotenv_values()` з модуля dotenv. Слід сказати, що функція відповідає за дозвіл зчитування змінних поточного середовища. Відбувається виклик функції `os.getcwd()` з метою отримання шляху до поточної робочої директорії, а також зберігається шлях в спеціальній змінній current_directory. Використовується функцію dotenv_values, щоб зчитати змінні середовища. Змінні з файлу зберігаються у словнику env_vars.

3.3 Структурна частина ViewModel

Частина ViewModel виконує роль посередника між View та Model. Вона обробляє дані з частини Model та потім надсилає команди до View для оновлення інтерфейсу користувача.

Імпортуються стандартні бібліотеки, серед яких time, traceback, logging,

а також json для обробки часу, помилок, журналів та JSON-файлів. Відбувається імпорт компонентів PyQt6 (QTimer, QIcon, QMainWindow тощо) для створення інтерфейсу користувача. Імпортуються спеціальні модулі, що відповідають за роботу з базою даних (Database), менеджмент стилів (StyleManager), конфігурацій (ConfigManager), обробку сповіщень та перевірку URL-адрес. Функція `create_folder('media')` створює папку media для збереження медіа, якщо така папка не існує.

Слід сказати, що BookmarkManager – це основний клас програми, що успадковує можливості QMainWindow для задачі створення головного вікна програмного застосунку. Ініціалізація базових компонентів відбувається наступним чином: викликається конструктор базового класу QMainWindow та потім виконується ініціалізація основних змінних. Для роботи з БД ініціалізується об'єкт db, що відповідає за збереження та обробку даних закладок. Створюється об'єкт BookmarkViewModel, який пов'язує базу даних та графічний інтерфейс програми. Ініціалізуються менеджери конфігурацій (ConfigManager) та стилів (StyleManager), які відповідають за зчитування налаштувань з файлу та застосування стилів до інтерфейсу.

Створюється та додається головний центральний віджет CentralWidget. Він містить основні компоненти інтерфейсу (таблиці, поля для редагування закладок та інше).

Описуються спеціальні властивості (properties), такі як TABLE_VIEW, ITEM_EDIT, TREE_VIEW. Вони звертаються до різних компонентів центрального віджета. Їх мета полягає в управлінні таблицями, деревами та редагуванні елементів інтерфейсу. Метод `show_import_dialog(self)` відповідає за відкриття діалогового вікна для імпорту даних із файлу.

Після вибору файлу поточні дані завантажуються в базу, та користувачу відображається інформаційне повідомлення про успішний імпорт інформації. Метод `show_export_dialog(self)` показує діалогове вікно для експорту закладок в певний файл. Після успішного експорту дані зберігаються у вибраний файл, та виводиться повідомлення про успішне

збереження. Метод `quit(self)` відповідає за збереження всі даних з бази перед тим, як завершити роботу програмного застосунку, і закриває його. Метод `onQuit(self)` зберігає поточний стан всіх елементів (розширені та згорнуті теки) перед виходом та зберігає всі дані в БД. Метод `repopulate(self)` відповідає за оновлення відображення елементів у таблиці та дереві інтерфейсу у випадку, коли з базою даних відбулися певні зміни.

Оголошується метод `create_new_bookmark()`, мета якого створити нову закладку. Слід сказати, що цей метод належить до класу, використовує атрибути, а також оркемі методи цього клас. Відбувається отримання вибраного елемента (теку або іншу закладку) з таблиці TABLE_VIEW, яка відображає список елементів.

Далі відбувається процес відкриття діалогового вікна для введення користувачем назви нової закладки. Слід вказати, що title відповідає за заголовок діалогового вікна, content – за повідомлення з проханням ввести назву нової закладки.

Повертається два значення: name (введена назва закладки), ok (булеве значення, яке вказує, чи натиснув користувач «ОК » або «Скасувати». Слід сказати, що у випадку, коли користувач натиснув «Скасувати » або не підтвердив введення (ok == False), процес створення закладки припиняється, та поточна функція повертає None.

Створюється новий елемент закладки з назвою name, яку ввів користувач, та додається в обрану теку folder. Відбувається виклик методу `add_named_item()` у дереві TREE_VIEW, яке представляє структуру папок і закладок. До створеної закладки додається мітка часу, щоб зафіксувати час створення.

Функція `time.time()` повертає поточний час в секундах (з 1970 року, час в UNIX форматі). Оновлюється вибір у таблиці TABLE_VIEW, щоб після створення нової закладки знову була вибрана тека, в яку ця закладка була додана.

Перезаповнюється таблиця TABLE_VIEW для відображення всіх

актуальних змін (нову закладку). Реалізація методу `create_new_bookmark()` представлена на лістингу 3.9.

Лістинг 3.9 – Метод `create_new_bookmark()` (файл main.py)

```
def create_new_bookmark(self):
    folder = self.TABLE_VIEW.selected_item
    if not folder or folder.data_type != 'folder' or "q_item"
not in folder.cache:
        folder = self.database.data
    name, ok =
self.box_manager.getUserInput(title=l("Редагування"),
content=l("Введіть назву нової закладки:"))
    if not ok:
        return
    item = self.TREE_VIEW.add_named_item(name, folder)
    item.date = time.time()
    self.TABLE_VIEW.selected_item = folder
    self.TABLE_VIEW.repopulate()
    self.ITEM_EDIT.item_edit(item)
```

Створюється вебсервер на базі Flask, який реалізує кілька REST API для обробки реєстрації, входу, збереження даних користувачів, змін паролів, нікнеймів. Він працює з локальними JSON-файлами для збереження даних користувачів. Імпортуються необхідні бібліотеки, такі як Flask для створення вебсервера, модуль asyncio для асинхронного запуску сервера, модуль os для роботи з файловою системою, користувацький клас ConfigManager, для роботи з JSON-конфігураціями. Слід сказати, що функція `create_folder()` створює папку за вказаним шляхом, якщо така не існує. Це потрібно для зберігання даних користувачів у спеціальних файлах. Ініціалізуються три об'єкти ConfigManager для збереження інформації про користувачів, відповідні нікнейми та дані у файлах JSON.

За допомогою декоратора Flask визначається маршрут /register, який реагує на HTTP-запити методом GET. Слід сказати, що поточний користувач має можливість надсилати дані на цей маршрут через певні URL-параметри. Оголошення функції `register()`, яка обробляє запити на реєстрацію нового користувача. Використовуючи `request.args.get()`, функція отримує

значення параметрів username, а також password з GET-запиту (повинні бути передані через URL).

Відбувається перевірка: чи були передані значення username та password. Якщо будь-яке з них відсутнє, тоді функція повертає JSON-відповідь з помилкою та статус-кодом 400 (тобто помилка клієнта). Далі перевіряється, чи вже існує користувач з таким логіном у системі, або чи не намагається поточний користувач зареєструвати логін «guest» (такий логін зарезервований). Якщо логін зайнятий, у такому випадку повертається відповідь з помилкою та статус-кодом 400. У випадку, коли логін не зайнятий, система зберігає новий обліковий запис, додаючи у файл даних користувачів пару username: password. Після успішного збереження нового користувача відбувається повернення JSON-відповіді з повідомленням про успішну реєстрацію та статус-кодом 200 (успішна операція). Реалізація функції `register()` представлена на лістингу 3.10.

Лістинг 3.10 – Функція `register()` (файл __main__.py)

```
def register():
    username = request.args.get('username')
    password = request.args.get('password')
    if not username or not password:
        return jsonify({'error': 'Введіть логін та пароль.'}),
400
    if username in users.read_config() or username == 'guest':
        return jsonify({'error': 'Логін вже зайнято.'}), 400
    users.update({username: password})
    return jsonify({'message': 'Реєстрація успішна!'}), 200
```

Маршрут /login (вхід) приймає запит на вхід користувача. Відбувєаться перевірка введених логіна, пароля та у разі входу, який завершився успіхом, повертає повідомлення, що містить нікнейм користувача. Маршрут /send_json приймає JSON-дані від користувача (після авторизації) і зберігає їх. Описується маршрут/set_nickname, який відповідає за прийом нікнейму від користувача та зберкжкння його після перевірки логіна та пароля. Маршрут /set_new_password дозволяє змінювати пароль користувача після

успішної аутентифікації. Описується маршрут /retrieve_json, що повертає збережені раніше JSON-дані користувача після процесу авторизації.

Асинхронний запуск сервера використовується для запуску Flask-сервера через спеціальну асинхронну функцію. Запуск сервера Flask може відбуватися у режимі налагодження чи без нього, залежить від конфігурації.

За управління закладками в базі даних та їх відображення в графічному інтерфейсі, реалізованому за допомогою PyQt6, відповідає клас BookmarkViewModel. Конструктор класу приймає об'єкт бази даних та менеджер, які зберігаються в атрибутах з метою подальшого використання. Метод `move_source_to_destination()` дозволяє переміщати закладку з однієї папки в іншу за умови, що цільова папка не є спеціальною (наприклад, «обрані» чи «важливі»), та закладка, що переміщується, не є «недоступною». Після процесу переміщення оновлюється структура закладок у базі даних.

Один з методів `move_alongside()` відповідає за переміщення закладки поруч з іншою закладкою в створеній структурі бази даних. Метод `move_alongside()` приймає два аргументи: source_item, це – закладка, що переміщується, та destination_item, тобто місце, куди закладка буде переміщена. Якщо будь-який з аргументів source_item або destination_item не існує (є None), тоді метод завершить виконання без додаткових дій.

У випадку, коли destination_item є елементом, позначеним як «обране», метод завершить виконання. Переміщення в «обране» не дозволено. Схожа ситуація з перевіркою на важливість та редагування. Якщо source_item позначено як «недоступне для редагування», метод також завершить виконання.

Відбувається видалення з батька, тобто закладка source_item видаляється зі списку елементів свого батьківського елемента. Вилучається відповідний рядок source_item з кешу в батьківському елементі та оновлюється відображення в інтерфейсі.

У випадку, кол у destination_item немає батьківського елемента (він є кореневим), виконуються подальші дії. Закладка source_item додається до

списку елементів destination_item, якщо destination_item не має свого батька. Відбувається встановлення нового батьківського елементу для source_item, який тепер є destination_item. Якщо destination_item має батька, виконуються інші дії. Закладка source_item додається до вже до списку елементів батьківського елемента destination_item.

Встановлюється новий батьківський елемент для source_item та тепер він ж батьком destination_item. Зміни в базі даних обов'язково зберігаються, щоб усі зміни стали постійними.

Метод для повторного заповнення відображення дерева закладок викликається з метою відображення нової структури після переміщення елемента. Реалізація методу `move_alongside()` представлена на лістингу 3.11.

Лістинг 3.11 – Метод `move_alongside()` (файл BookmarkViewModel.py)

```
def move_alongside(self, source_item: QDatabaseItem,
destination_item: QDatabaseItem):
    if not destination_item or not source_item:
        return
    if destination_item.data_item.cache.get('favorites'):
        return
    if destination_item.data_item.cache.get('importants'):
        return
    if source_item.data_item.cache.get('uneditable'):
        return
source_item.data_item.parent.items.remove(source_item.data_item)
source_item.data_item.parent.cache['q_item'].removeRow(source_it
em.row())
    if not destination_item.data_item.parent:
destination_item.data_item.items.append(source_item.data_item)
        source_item.data_item.parent =
destination_item.data_item
    else:
destination_item.data_item.parent.items.append(source_item.data_
item)
        source_item.data_item.parent =
destination_item.data_item.parent
    self.database.save()
    self.manager.TREE_VIEW.repopulate()
```

Імпортується клас QSystemTrayIcon з бібліотеки PyQt6.QtWidgets, який

використовується для створення символу системного трея, що дозволяє взаємодіяти з користувачем через спливаючі повідомлення. Імпортується клас Database з модуля components.model.Database та імпортується функцію `check_url_availability()`. Оголошується функція `check_and_notify()`, яка приймає один параметр tray_icon. Він є екземпляром QSystemTrayIcon. Слід сказати, що ця функція перевіряє доступність URL-адрес та, у певному випадку, якщо потрібно, відправляє поточні повідомлення. Створюється новий екземпляр класу Database. Отримується список елементів (посилань) з бази даних, що зберігається в змінній items. Ініціалізується пустий список links, у який будуть додаватися URL-адреси, які не є доступними.

Запускається цикл, який перебирає всі item в списку items. Викликається функцію `check_url_availability()` для кожної URL-адреси item.address. Якщо адреса не доступна (тобто не повертає «ok»), виконуються інші дії. URL-адресу додається до списку links, якщо вона не є доступною.

Відбувається перевірка, чи є в списку links недоступні URL-адреси. Відбувається об'єднання всіх недоступних URL-адреси в один рядок, розділений новими рядками, та збереження в змінній notification_text. Викликається метод `showMessage()` на об'єкті tray_icon, щоб показати спливаюче повідомлення, текстом, що містить недоступні URL-адреси, іконкою інформаційного повідомлення, яке відображається протягом певного часу. Реалізація функції `check_and_notify()` представлена на лістингу 3.12.

Лістинг 3.12 – Метод `check_and_notify()` (файл Notifications.py).

```
def check_and_notify(tray_icon):
    database = Database()
    items = database.data.items
    links = []
    for item in items:
        if check_url_availability(item.address) != 'ok':
            links.append(item.address)
    if links:
        notification_text = "\n".join(links)
        tray_icon.showMessage(
```

```
            "Новые уведомления",
            notification_text,
            QSystemTrayIcon.MessageIcon.Information,
            15000
        )
```

Імпортується модуль asyncio, який дозволяє працювати з асинхронним програмуванням у Python, забезпечуючи можливість запуску асинхронних функцій. Імпортується функцію start_flask_server з модуля __main__ у пакеті server. Відбувається оголошення асинхронної функції main, яка буде використовуватися з метою запуску певних асинхронних завдань.

Створюється асинхронне завдання (flask_task) для запуску функції `start_flask_server()`, яка запускає Flask-сервер. Відбувається перевірка, чи є цей файл головним модулем програми, що виконується. У випадку, якщо так, то будуть виконуватися подальші дії. Описується асинхронна функція `main()` за допомогою `asyncio.run()`, що запускає асинхронну подію та управляє її виконанням. Це ініціює весь процес запуску Flask-сервера. Реалізація функції `main()` представлена на лістингу 3.13.

Лістинг 3.13 – Функція `main()` (start_server_components.py)

```
async def main():
    flask_task = asyncio.create_task(start_flask_server())
    await asyncio.gather(flask_task)
if __name__ == "__main__":
    asyncio.run(main())
```

3.4 Структурна частина View

Частина View містить усі UI-компоненти, до яких можна віднести кнопки, вікна та стилі, які взаємодіють з користувачем. Всі зміни в інтерфейсі надсилаються назад до ViewModel для обробки.

Клас CentralWidget – центральний віджет у графічному інтерфейсі програмного застосунку на основі бібліотеки PyQt6. Мета класу полягає в створенні та організації різних елементів інтерфейсу для управління

закладками та даними. При створенні об'єкта класу CentralWidget ініціалізуються певні компоненти, які будуть відображатися у вікні для користувача. До них можна віднести дерева закладок, таблиці, плитки та панель редагування. Використовується QSplitter для розділення вмісту на різні частини, що дозволяє користувачеві змінювати розміри вікон. Праворуч від дерева закладок розміщується віджет, в якому можна перемикатися між таблицею та плиткою.

Один з методів `right_menu()` реалізований для розділення віджета на дві частини використовується метод. Метод не приймає жодних параметрів, окрім self (вказує на екземпляр класу). Створюється новий об'єкт QSplitter. QSplitter є віджетом у PyQt, що дозволяє ділити простір на декілька частин та змінювати розмір, перетягуючи межі. Встановлюється орієнтація на вертикальну (додані віджети будуть розташовані один над одним). Додається віджет self.right_stacked. self.right_stacked зазвичай є віджетом (QStackedWidget), який може містити декілька віджетів, а також показувати лише один з них в певний момент.

Встановлюєтья коефіцієнт розтягування для першого віджета (індекс 0, self.right_stacked) на 3. Віджет займатиме більше простору в порівнянні з іншим віджетом. Встановлює коефіцієнт розтягування для іншого віджета (індекс 1, тобто self.ITEM_EDIT) на 1 Віджет займатиме менше місця в порівнянні з першим віджетом. Формуються два віджети: self.right_stacked і self. ITEM_EDIT, з налаштованими коефіцієнтами розтягування. Реалізація методу `right_menu()` представлена на лістингу 3.14.

Лістинг 3.14 – Метод `right_menu()` (файл CentralWidget.py)

```
def right_menu(self):
    splitter = QSplitter()
    splitter.setOrientation(Qt.Orientation.Vertical)
    splitter.addWidget(self.right_stacked)
    splitter.addWidget(self.ITEM_EDIT)
    splitter.setStretchFactor(0, 3)
    splitter.setStretchFactor(1, 1)
```

```
return splitter
```

Метод `menu_bar()` створює меню для програми з пункамии для управління закладками, експорту та імпорту даних. Додаються дії щодо налаштуваннями, виходу з програми, створення нових закладок та редагування. Для створення верхньої панелі, що відповідає за відображення привітання користувачу (з врахуванням його імені) та елементи управління для вибору виду відображення (список чи плитка), поле для пошуку закладок був використаний метод `top_layout()`. Завдяки компонентам TreeView, TableView, TileView та EditWidget, реалізується механізми для перегляду, редагування та управління закладками в зручному для користувача вигляді.

Теми програми реалізовані за допомогою файлу стилів для Qt6. Визначаються певні змінні $BACKGROUND для загального фону, $ACCENT_COLOR для основного кольору, та $DARKER_ACCENT_COLOR для темнішого кольору. Змінні відповідають за текст, фонові кольори, межі та поведінку елементів під час наведення чи при взаємодії. За допомогою QMainWindow задається фоновий колір для головного вікна програми. QWidget задає фон для всіх віджетів та колір тексту. QLabel відповідає за текст (відображається у світло-сірому кольорі).

QPushButton реалізує зелений фон, білий текст з обведенням та зміну кольору при наведенні. QScrollBar задає певний стиль та розміри вертикальної смуги прокрутки. За допомогою QComboBox, QLineEdit, QPlainTextEdit задаються фонові кольори, а також всі межі для випадаючих списків, полів для введення тексту.

Задаються стилі для індикаторів вибору (галочки та радіо-кнопки) з кольорами для вибраних та не вибраних станів за допомогою QCheckBox, QRadioButton. Записуються селектори для індикаторів чекбоксів та радіо-кнопок. У Qt такого роду індикатори –графічні частини віджетів, що відображають стан вибору (галочку або радіо-кнопку). За допомогою background-color: $ACCENT_COLOR;: визначається фоновий колір

індикатора. Для цього використовується змінна $ACCENT_COLOR. border: 1px solid $ACCENT_COLOR; встановлюється рамка навколо індикатора. Реалізація стилів для індикаторів вибору представлена на лістингу 3.15.

Лістинг 3.15 – Стилі для індикаторів вибору (файл Light.qss)

```
QCheckBox::indicator, QRadioButton::indicator {
    background-color: $ACCENT_COLOR; /* Material BLUE */
    border: 1px solid $ACCENT_COLOR; /* Material Blue border */
}
```

За допомогою QSplitter задається стиль, який дозволяє користувачу змінювати розмір розділених вікон.

Для для редагування елементів та папок, що зберігаються в базі даних програми використовується клас EditWidget. Для інтерфейсу користувача використовується QGridLayout для організації віджетів. Клас надає можливості щодо редагувати назви, тегів, URL та описів елементів.

Для відображення статусу URL, зміну його зовнішнього вигляду реалізований метод `paint_available()`. Він приймає параметр available, що за замовчуванням дорівнює True, тобто посилання є доступним. Якщо URL доступний, тоді текст у полі для введення URL (self.url_edit) змінюється на стандартний колір основного тексту, що визначається у стилях програми.

Одночасно, поле для повідомлення (self.url_tag) очищується від будь-якого тексту, через те, що немає необхідності повідомляти про проблеми з посиланням. У випадку, коли URL недоступний, тоді текст у полі для введення URL змінюється на червоний, попереджаючи про проблему. У віджеті для повідомлень (self.url_tag) з'являється відповідний текст («Посилання не існує!») того ж червоного кольору, символізуючи помилку. Слід зазначити, що результат відображається користувачеві.

Для редагування даних реалізована можливість для користувача щодо зміни назви, тегу, опису елемента або переміщення елементу до іншої папки. Описані методи для обробки редагування як папок, так і окремих елементів.

Слід сказати, якщо елемент має фото, воно відображається, при цьому користувач може переглядати URL, копіювати його в буфер обміну чи відкривати посилання в браузері.

Описана реалізація можливості ініціювати пошук схожих елементів через штучний інтелект (використовується компонент AiLoadingPopup). Описується масове управління потоками: для асинхронної обробки запитів (наприклад, перевірки доступності URL) використовується потоки UrlCheckThread, так як дозволяють запускати паралельні операції без блокування інтерфейсу.

Для створення діалогового вікна MoveToPopup використовується бібліотека PyQt6. Вікно дозволяє користувачу вибирати місце розташування з випадаючого списку та підтверджувати або відмінити вибір.

Клас MoveToPopup наслідує QDialog. Це дозволяє створити модальне діалогове вікно. Слід зазначити, що конструктор класу приймає об'єкт manager, який містить дані для заповнення випадаючого списку з доступними місцями розташування (директоріями або папками).

За іціалізацію інтерфейсу користувача для діалогового вікна відповідає метод `init_ui()`. Створюється мітка, що показує текст «Місцезнаходження». Створюється випадаючий список, який дає користувачеві вибрати один із доступних варіантів.

До випадаючого списку додається назва поточної бази даних, яка міститься в об'єкті `self.manager.database.data.name`. Другим параметром передається об'єкт бази даних для подальшого використання.

Відбувається ітерація по всіх елементах бази даних, отриманих за допомогою методу `get_everything()`. Метод повертає всі об'єкти, які знаходяться в базі даних.

При умові, що поточний елемент має тип «folder», виконується наступний крок. Якщо елемент є папкою, її ім'я додається у випадаючий список як варіант для подальшого вибору. Другим параметром передається сам об'єкт папки.

Описується кнопка підтвердження з написом «OK». Вона використовується для підтвердження вибору. Створюється відповідна кнопка скасування з написом «Cancel» для дозволу користувачу відмовитися від вибору. Створюється спеціальний вертикальний макет, що організовує розташування елементів інтерфейсу один під одним.

До макета додається мітка з текстом «Місцезнаходження». До макета додається випадаючий список, який дає можливість вибору папки, та кнопка підтвердження і відміни.

Створений макет застосовується до поточного діалогового вікна, що встановлює розташування елементів інтерфейсу. Під час натискання кнопки підтвердження викликається метод `accept()`, який закриває діалогове вікно та підтверджує вибір.

Слід зазначити, що при натисканні кнопки «Cancel» викликається метод `reject()`, який закриває діалогове вікно без підтвердження певного вибору. Реалізація методу `init_ui()` представлена на лістингу 3.16.

Лістинг 3.16 – Метод `init_ui()` (файл MoveToFolderPopup.py)

```
def init_ui(self):
    # Create widgets
    label = QLabel(l("Місцезнаходження")+":")
    self.combo_box = QComboBox()
    self.combo_box.addItem(self.manager.database.data.name,
self.manager.database.data)
    for folder in self.manager.database.get_everything().items:
        if folder.data_type == 'folder':
            self.combo_box.addItem(folder.name, folder)
    ok_button = QPushButton("OK")
    cancel_button = QPushButton("Cancel")
    # Create layout
    layout = QVBoxLayout()
    layout.addWidget(label)
    layout.addWidget(self.combo_box)
    layout.addWidget(ok_button)
    layout.addWidget(cancel_button)
    # Set layout for the dialog
    self.setLayout(layout)
    # Connect button signals to slots
    ok_button.clicked.connect(self.accept)
    cancel_button.clicked.connect(self.reject)
```

Випадаючий список заповнюється папками з бази даних менеджера. Кнопка підтвердження приймає зміни та закриває діалогове вікно. Кнопка відміни скасовує зміни та закриває поточне вікно.

Для керування стилями та кольорами для графічного інтерфейсу користувача визначається клас StyleManager. Він імплементує механізм для налаштування зовнішнього вигляду елементів управління через стильові таблиці (QSS) та виконує збереження налаштувань у конфігураційному файлі config.json. У конструкторі класу __init__ ініціалізуються параметри, такі як додаток (app), менеджер (manager), стилі та кольори. За замовчуванням встановлюється стиль «Fusion» та колір «QLight». Відбувається читання конфігурації. Слід сказати, що використовуються певні властивості, такі як all_styles та all_colors, які повертають списки доступних стилів та кольорів для відображення. Метод `change_style()` змінює стиль програми, зберігаючи його в конфігурації. Слід зазначити, що метод `change_color()` аналогічно змінює колір, завантажуючи стильові таблиці з файлу QSS.

За читання файлу стилів в форматі QSS (Qt Style Sheet) та обробку його вмісту відповідає метод `read_qss_file()`. Визначає метод read_qss_file, який приймає один аргумент file_name. Метод використовується для читання файлу стилів.

Контекстний менеджер with відповідає за запезпечення відкриття файлу з ім'ям file_name в режимі читання («r »). Це гарантує, що файл буде автоматично закритий після завершення роботи. Читається весь вміст файлу та зберігається в змінній content. Ця змінна міститиме текстовий вміст QSS-файлу.

Викликає метод interpret_color_map, передаючи вміст файлу content як аргумент. Результати повертаються у двох змінних: content (оновлений вміст) та color_map (словник з мапою кольорів). Повертається кортеж з двох значень: оновленого вмісту стилів content та мапи кольорів color_map. Дані використовуються для застосування стилів у GUI розробленого додатку.

Методи `inject_arrows()` та `set_background()` надають специфічний

стиль для елементів (заголовки таблиць, додаючи стрілки та фонові зображення). Реалізована обробка виключень, щоб логувати помилки при зміні кольорів, забезпечуючи надійність при налаштуванні оформлення.

Для реалізації графічного інтерфейсу для відображення даних у вигляді таблиці за допомогою PyQt6 використовується клас TableView.

Клас TableView успадковується від QTableView та використовується для створення таблиці, що відображає елементи даних. У конструкторі ініціалізуються властивості, такі як стиль відображення, обробка подій кліків на заголовках стовпців, можливість сортування, а також відбувається налаштування зовнішнього вигляду таблиці. Метод `headerClick()` змінює порядок сортування таблиці під час натискання на заголовок стовпця. Метод `selectItem()` обробляє вибір елемента (відкривається редактор для елементів або папок, залежно від типу даних).

Метод `data()` повертає дані для вибраної папки, а `repopulate()` оновлює модель таблиці з новими даними. Метод `sort_item()` відповідає за сортування даних за різними стовпцями, зважаючи на певний тип.

Для відображення кіл, що символізують значення елементу (папка чи ступень важливості закладки) реалізований клас EllipseDelegate, який наслідується від QStyledItemDelegate. Це дозволяє використовувати спеціальний стиль для відображення даних у таблиці.

Метод `paint` забезпечує відображення (тобто процес малювання) вмісту комірок в таблиці. Він приймає параметр painter (об'єкт, що використовується для малювання на екрані), option (інформація про стиль та стан елемента) та index (індекс елемента в моделі даних).

Викликається базовий метод `paint()` батьківського класу, щоб виконати стандартне малювання елементів таблиці перед тим, як реалізувати додатковий стиль. Для отримання даних для поточного елемента за індексом метод `data()` повертає значення, що відображається в таблиці для відповідного індексу (Qt.ItemDataRole.DisplayRole).

Відбувається перевірка типу даних елемента. Якщо це папка (folder), то

встановлюється сірий колір для кола. Якщо це інший тип елемента, у такому випадку обирається колір, що відповідає важливості елемента (importance). Створюється прямокутник (QRectF), всередині якого буде намальоване коло. Прямокутник має розміри 15 на 15 пікселів та зміщений на 10 пікселів праворуч і 5 пікселів вниз від верхнього лівого кута комірки.

Встановлюється пензель (QBrush) з відповідним кольором (QColor) для малювання кола. Колір залежить від типу елемента (сірий чи колір ступеня важливості). Відбувається малювання кола в межах заданого прямокутника, використовуючи налаштований пензель.

Реалізація класу EllipseDelegate представлена на лістингу 3.17.

Лістинг 3.17 – Клас EllipseDelegate (TableView.py)

```
class EllipseDelegate(QStyledItemDelegate):
    def paint(self, painter, option, index):
        super().paint(painter, option, index)
        item = index.data(Qt.ItemDataRole.DisplayRole)
        if item.data_type == 'folder':
            color = "gray"
        else:
            color = item.importance
        ellipse_rect = QRectF(option.rect.left() + 10,
option.rect.top() + 5, 15, 15)
        painter.setBrush(QBrush(QColor(color)))
        painter.drawEllipse(ellipse_rect)
```

Клас CustomModel реалізує QAbstractTableModel для управління даними таблиці. Визначаються рядки та певні стовпців, а також те, які саме дані мають відображатися у кожному з них. Метод `headerData()` надає заголовки для стовпців. Метод `data()` повертає конкретні дані в залежності від стовпця та рядка.

Для відображення та управління елементами у вигляді плиток в інтерфейсі користувача використовується клас TileView. Він інтегрується з системою закладок (bookmarks) та забезпечує кілька функцій для взаємодії з елементами в певному віджеті. При створенні об'єкта класу TileView ініціалізуються базові налаштування QListView, та відбувається прив'яка до

різних обробок подій (контекстне меню, клік та подвійний клік миші). Відбувається збереження посилання на менеджер закладок та в модель для відображення елементів (QStandardItemModel).

Метод `init_settings()` відповідає за основні налаштування для віджета: керує дозволом щодо перетягування елементів, налаштовання розмірів іконок, режиму відображення, а також фільтрії події для обробки натискань клавіш. Метод `selectItem()` викликається, коли користувач робе клік на елементі. Обирається відповідний елемент, який передається в інший віджет для виконання процесу редагування. Метод `doubleClick()` реагує на подвійне клацання на елементі. Якщо це папка, вміст змінюється на вміст цієї папки. Якщо це інший тип елемента, він передається для проесу редагування.

Метод `repopulate()` перерисовує елементи у віджеті, відображаючи дочірні елементи певної папки. Він також додає можливість навігації «вгору» (показуючи спеціальний символ «.. » для переходу до батьківської папки). Підтримується перетягування елементів за допомогою миші. Метод `dropEvent()` визначає місце, куди було перетягнуто елемент, та ініціалізує його переміщення до потрібної папки.

Для показу контекстного меню (меню, що з'являється при кліку правою кнопкою миші) у вказаній точці point на віджеті tree_view використовується метод `showContextMenu()`. Індекс елемента (index) з’являється у tree_view, на якє зробтв клік користувач. Метод `indexAt()` визначає індекс елемента на за орієнтацією по координатам точки point. На основі отриманого індексу береться відповідний елемент моделі типу QStandardItem з `self.model()`.

Створюється об'єкт контекстного меню (menu) типу QMenu. Цей об'єкт представляє меню, яке буде показано. Створюється дія (action_remove), з якою можна ознайомитись в меню. Текс дії «Видалити» береться з спеціального локалізованого джерела тексту через функцію `l()`. До дії action_remove прив'язується подія (тобто обробник) через triggered.connect, що буде викликана при виборі цієї дії. У цьому випадку анонімна функція lambda виконує виклик методу `removeItem()`, який видаляє елемент за

вказаним індексом.

Перевіряється, чи отриманий індекс є дійсним (тобто чи насправді на цій позиції знаходиться певний елемент). У випадку, коли індекс недійсний, меню не буде реалізовано для поточного індексу. Якщо індекс дійсний, до меню додається дія action_remove (дія буде відображена в меню). Меню відображається в тому місці, де клацнув користувач. За допомогою `tree_view.mapToGlobal(point)` координати point перетворюються відносно вікна на глобальні координати на екрані, щоб меню з'явилося в правильному місці. Реалізація методу `showContextMenu()` показана на лістингу 3.18.

Лістинг 3.18 – Метод `showContextMenu()` (TileView.py)

```
def showContextMenu(self, point, tree_view):
    index = tree_view.indexAt(point)
    q_item: QStandardItem = self.model().itemFromIndex(index)
    menu = QMenu(self)
    action_remove = QAction(l('Видалити'), self)
    action_remove.triggered.connect(lambda:
self.removeItem(index))
    if index.isValid():
        menu.addAction(action_remove)
    menu.exec(tree_view.mapToGlobal(point))
```

Реалізується обробка подій клавіатури: дозволяється, наприклад, видаляти елементи при натисканні клавіші «Delete». Клас працює з об'єктами закладок та може отримувати та змінювати певий вміст закладок через спеціальну властивість data.

Імпортуються стандартні модулі, такі як time для роботи з часовими мітками, uuid для генерації унікальних ідентифікаторів та webbrowser для відкриття вебсторінок у браузері. Імпортуються модулі з бібліотеки PyQt6 такі як QtCore, QPoint для управління основними властивостями, такими як положення та події, QtGui для роботи з графічними елементами, QtWidgets для роботи з віджетами. Відбувається імпортує локальних модулів та класів: BoxManager (менеджер для управління діалогами), Item, Folder (класи для роботи з даними про закладки), MoveToPopup (спливаюче вікно для

переміщення елементів у папки), l (функція для локалізації тексту, QDatabaseItem (клас для представлення елементів БД у вигляді виджетів), ImportanceLevels (рівні важливості збережених закладок), BookmarkViewModel (модель, які відповідає за керування закладками).

Оголошується клас TreeView, що наслідується від QTreeView. TreeView відповідає за інтерфейс для відображення, а також взаємодії з деревом закладок. Він забезпечує відображення дерева, обробку перетягування елементів, відображення контекстних меню та взаємодію з користувачем. Під час створення екземпляра TreeView ініціалізуються основні компоненти, до яких відноситься модель даних і методи для взаємодії з обраними та важливими закладками. Відбуваються налаштування дій щодо події кліку миші, подвійного кліку та контекстного меню.

За налаштування функціональності для процесу перетягування та переміщення певних елементів у дереві (тобто drag-and-drop) відповідає метод `init_drag_settings()`. Методи `populate_favorite()` та `populate_important()` створюють спеціальні вузли для обраних та важливих закладок, а також додають їх до дерева, зберігаючи закладки відповідних категорій.

Метод `repopulate(self)` оновлює (перезаповнює) дерево з елементами в користувацькому інтерфейсі. Очищується модель даних для дерева, видаляючи всі елементи. Забезпечується редагування кореневого елемента дерева, роблячи його доступним для змін користувачем. Оновлюється кеш даних, позначаючи, що деякі елементи не можна редагувати. У кеші встановлюється спеціальний прапорець, який вказує, що всі закладки повинні бути видимими.

Задається іконка для дерева закладок, яка буде відображатися. Викликається метод `populate()`, щоб заповнити дерево елементів, отримуючи їх з бази даних за допомогою `get_tree_view()`, та прив'язуючи їх до кореня дерева. Дерево заповнюється закладками, які відмічені як «обрані» (favorites), використовуючи метод `populate_favorite()`.

Додаються «обрані» закладки до кореневого елемента дерева як новий рядок. Відбувається заповнення дерева закладками, що відмічені як «важливі» (important). Це забезпечується завдяки методу `populate_important()`. «Важливі» закладки додаються до кореневого елемента дерева як новий рядок.

Всі елементи дерева розкриваються, щоб показати весь його вміст. Цикл проходить по всіх елементах БД, які повертає метод `get_everything()`. При умові, що елемент є папкою та ще не розкритий (expanded), виконуються інші дії, а саме елемент складається у дереві, звертаючись до індексу в кеші (q_item). Реалізація методу `repopulate(self)` представлена на лістингу 3.19.

Лістинг 3.19 – Метод `repopulate(self)` (файл TreeView.py)

```
def repopulate(self):
    self.model().clear()
    self.root.setEditable(True)
    self.manager.database.data.cache['uneditable'] = True
    self.manager.database.data.cache['all_bookmarks'] = True
    self.manager.database.data.icon = 'all.png'
    self.populate(self.manager.database.get_tree_view(),
self.root)
    self.favorites = self.populate_favorite()
    self.root.appendRow(self.favorites)
    self.importants = self.populate_important()
    self.root.appendRow(self.importants)
    self.expandAll()
    for element in self.manager.database.get_everything().items:
        if element.data_type == 'folder' and not
element.expanded:
            self.collapse(element.cache["q_item"].index())
```

Слід зазначити, що потік даних відбувається наступним чином: дані та бізнес логіка рухаються від Model до ViewModel та потім до View, де вони відображаються користувачеві.

4 СИСТЕМНІ ВИМОГИ ТА ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Системні вимоги до програмного застосунка

Для використання розробленого програмного застосунка з управління закладками в інтернет ресурсах потрібна ОС Windows (для Linux та macOS необхідні додаткові процеси, такі як використання Python-інтерпретатора чи емулятор), 64-бітний процесор, 2 ГБ оперативної пам’яті, 1.34 ГБ простору на диску, вбудована відеокарта, мережа «Інтернет» (за вибором користувача).

4.2 Інструкція користувача

Для запуску програмного застосунку потрібно запустити файл StartGUI.exe. Відкривається вікно програми, а саме – гостьовий режим. Інтерфейс програми представлений на рисунку 4.1.

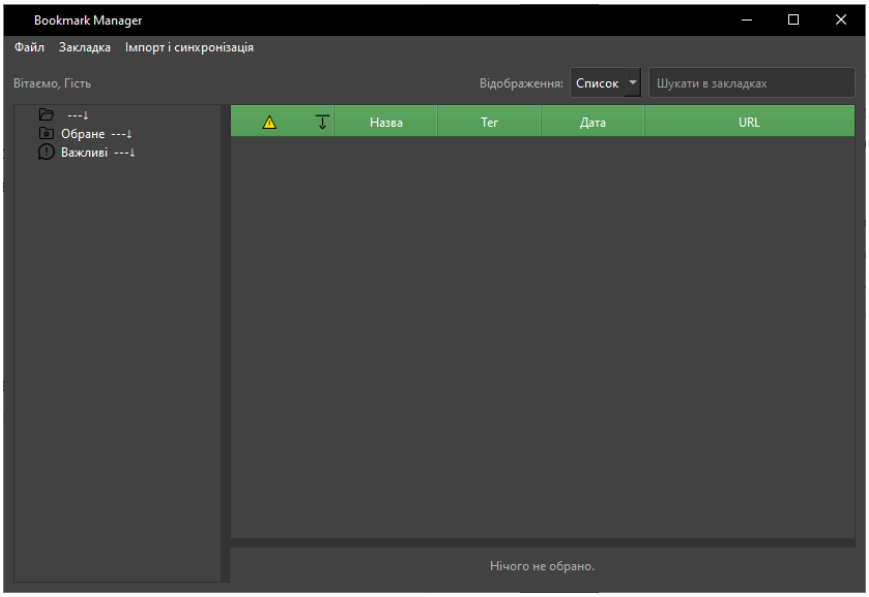

Рисунок 4.1 – Головне меню

В гостьовому режимі користувач може швидко створити закладки та не

проходити процедуру створення та входу в акаунт. Це корисно тоді, коли час обмежений та треба одразу додати посилання до списку.

Головне меню програми можна розділити на 4 частини:

- керування закладками (ліва частина);
- додаткові дії, пов’язані з обілковим записом і закладками (лівий верхній кут);
- список закладок (центральна частина);
- робота з закладкою (з’являється при кліку на назву ресурсу).

Для створення облікового записку потрібно натиснути на напис «Файл» та перейти в «Налаштування». У розділі «Обліковий запис» потрібно ввести логін, пароль та натиснути кнопку «Реєстрація». Розділ «Обліковий запис» представлений на рисунку 4.2.

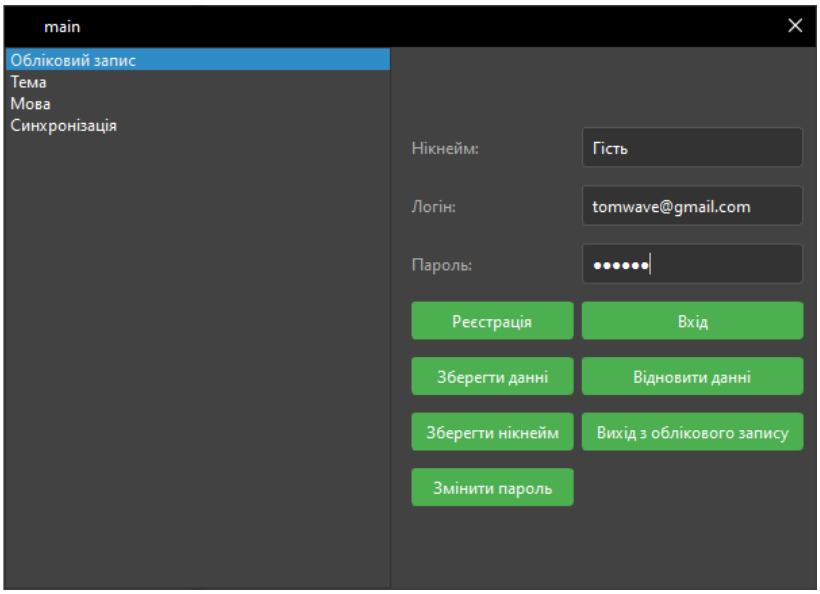

Рисунок 4.2 – Реєстрація користувача

Слід зазначити, що користувач може встановити свій нікнейм, який буде використовуватись у якості звернення в головному меню для більш персолізованого використання. Для цього потрібно вписати необхідне ім’я в поле «Нікнейм» та натиснути на кнопку «Зберегти нікнейм». Це важливо,

тоді, коли, наприклад, програмою одночасно користується декілька людей (члени родини), друзі: унікальне ім'я відокремлює один контент від іншого для зручного використання застосунка. Є функція зміни пароля. Для того, щоб змінити поточний пароль, потрібно натиснути на кнопку «Змінити пароль», ввести старий, після чого треба створити новий та підтвердити дії.

Користувач має можливість змінити тему застосунку. Можна змінити тему оформлення та колір. Тема впливає на оформлення елементів програми, та залежно від неї змінюється загальне представлення застосунку. Це має значення у випадку, коли користувач звик до певного інтерфейсу та знайомий з його елементами.

Одна з тем (тема: «Windows», колір: «Green») представлена на рисунку 4.3.

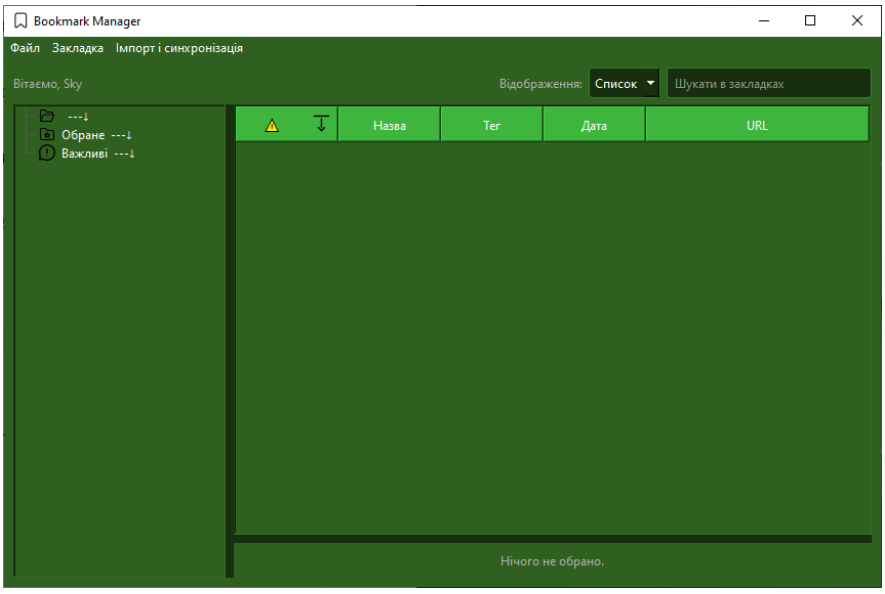

Рисунок 4.3 – Зелена тема

Є можливість змінити фон програмного забезпечення. Користувач може обрати власне зображення зі свого ПК. Фон буде розповсюджуватись на список закладок та папок у центральній частині програми.

Це може бути корисним для більш персоналізованого використання програмою разом з темою та кольором.

Встановлене фонове зображення представлено на рисунку 4.4.

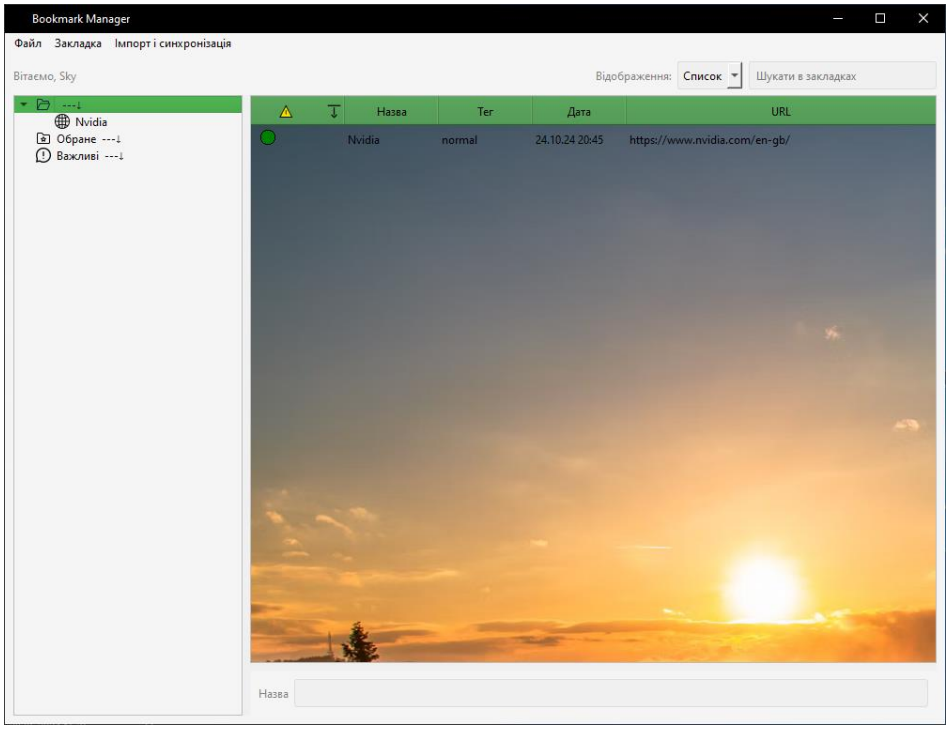

Рисунок 4.4 – Фон від користувача

Програма підтримує багатомовність та синхронізацію (експорт та імпорт JSON файлу).

Є такі можливості як створення нової закладки з буферу обміну, копіювання URL, відкриття URL, імпорт та синхронізація, доступ до яких відбувається через верхню частину вікна.

Користувач може створити закладку правим кліком миші на вільну зону в лівій частині застосунку. Є можливість створити закладку або одразу теку (папку) для зберігання закладок, вказавши відповідну назву, після чого елемент з'явиться у списку.

Присутні 3 розділи:

- всі папки (зберігають весь список закладок);
- «Обране» (окремі закладки);
- «Важливі» (закладки, що потребують уваги).

Частина з закладками та папками представлена на рисунку 4.5.

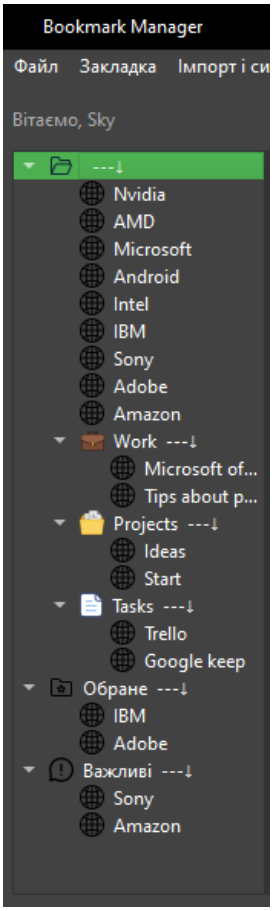

Рисунок 4.5 – Закладки та папки

Є можливість відрити закладку, копіювати її URL, додати до певної теки, додати до обраного, помітити, як важливе, видалити, змінити іконку, редагувати, підвищити чи понизити важливість та додати фото.

Користувач має можливість виставляти ступінь важливості для кожної закладки. Є можливість як підвищувати ступінь, так і знижувати його. Зміни можна відслідковувати в загальному списку закладок та папок.

Присутні 4 вида ступіня:

- звичайна закладка (зелене коло);
- закладка середньої важливості (жовте коло);
- важлива (помаранчеве коло);
- дуже важлива (червоне коло).

Папкам відповідає сіре коло, що виділяє їх з-поміж закладок (посилань). Використання ступенів важливості представлене на рисунку 4.6.

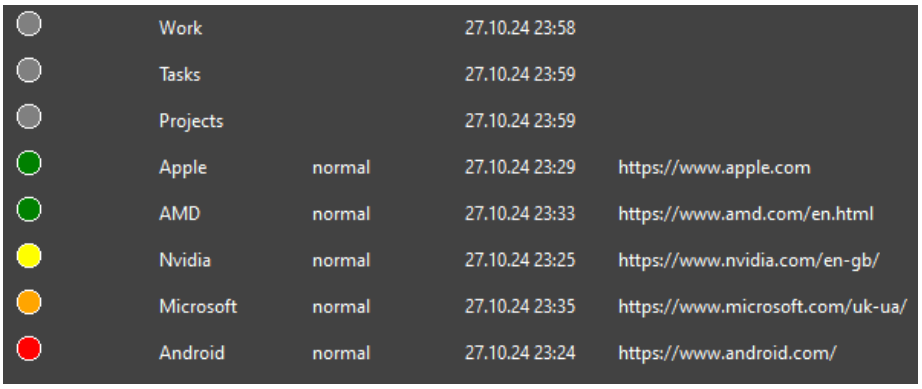

Рисунок 4.6 – Пріоритетність ресурсів

Є фунція додання фото до закладки. Користувач може додаватати свої фото до розділу закладки для того, щоб відокремити її від інших чи зробити відображення змісту. Зображення для закладки представлене на рисунку 4.7.

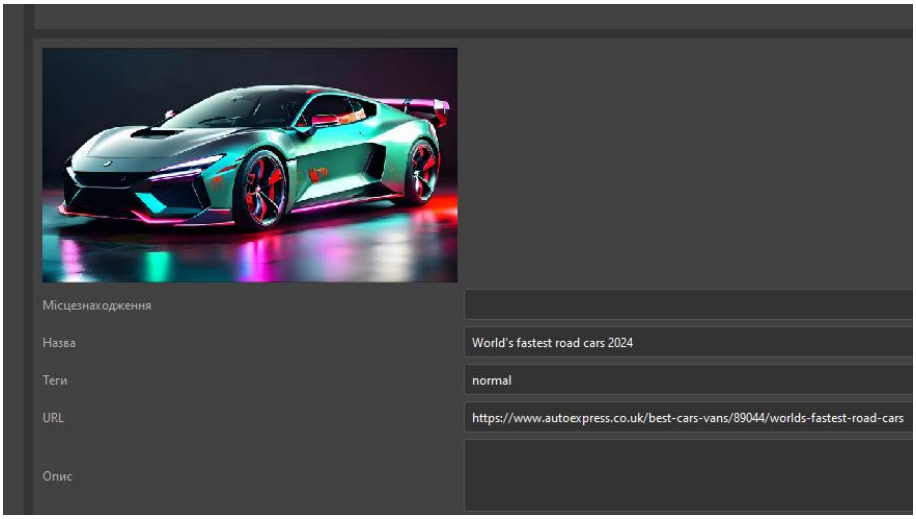

Рисунок 4.7 – Встановлене зображення

Реалізована функція drag-and-drop (перетягування) для переміщення закладок та тек.

Користувач має різні можливості щодо взаємодії з закладками. Є такі

функції, як зміна місцезнаходження, редагування назви (закладки, папки), встановлення тегу, зміна URL, перевірка на активність посилання та опис закладки.

Встановлення тегу дає можливість відділити елементи один від іншого за обраними власноруч користувачем критеріями: тематика, пріоритетність, зацікавленність тощо.

При введені неактивного посилання буде відображене відповідне повідомлення. Попередження представлене на рисунку 4.8.

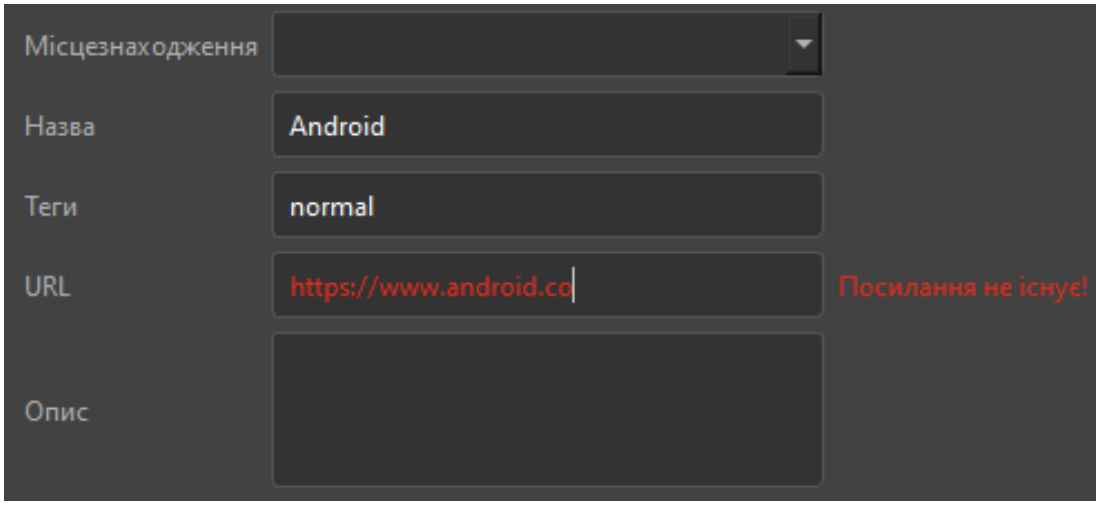

Рисунок 4.8 – Повідомлення про посилання

Є можливість в цьому розділі копіювати URL, відкрити його, перевірити посилання та шукати рекомендований контент.

Для перевірки посилання закладки на активність потрібно натиснути на відповідну кнопку («Перевірити URL»). У випадку, якщо писилання активне, тоді з'явиться повідомлення «Посилання працює нормально!». У ситуації, коли посилання не є активним, з'явиться відповідне повідомлення.

Користувачу не обов'язково власноруч перевіряти адреси, так як при виявленні неактивного посилання, про це буде отримано повідомлення в Windows. Це допомагає користувачу не витрачати зайвий час з метою відслідковування стану кожного посилання.

Повідомлення дають актуальну інформацію про закладки без зайвих дій та додаткових перевірок, дозволяючи сфокусуватися на взаємодії з реальними ресурсами. Повідомлення про стан посилання представлене на рисунку 4.9.

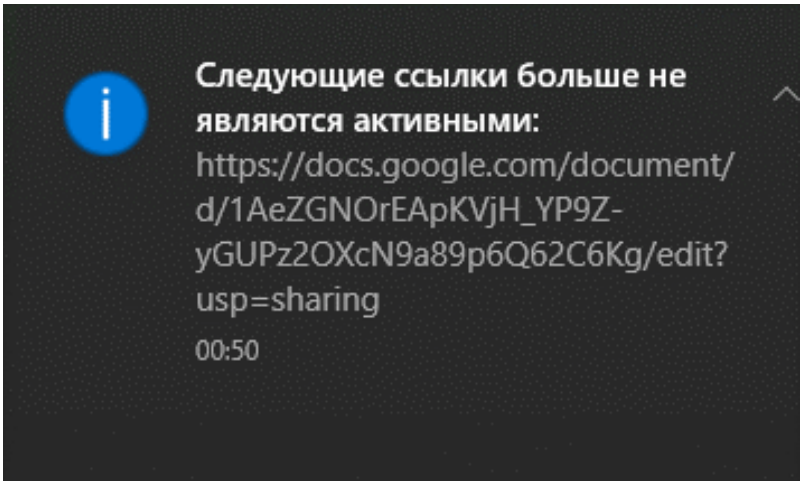

Рисунок 4.9 – Інформація про посилання

Для пошуку рекомендованого контенту треба натиснути на кнопку «Шукати схоже» та «Почати пошук». Користувач може шукати контент, який безпосередньо пов’язаний зі збереженими посиланнями.

Це можуть наступні ресурси:

- сайти (новини, технології, спеціалізовані сайти);
- статті (статті на різні теми, спеціалізовані);
- вебзастосунки (спеціалізовані ресурси, соціальні мережі та інше).

Є можливість одразу скопіювати посилання через відповідну кнопку, після чого можна створити нову закладку чи знайти інформацію щодо ресурсу в інтернеті.

Пушук контенту дає можливість дізнаватись про нові інформаційні джерела, нових авторів та різні сфери. Слід зазначити, що контент буде згенерований на основі відповідного посилання. Ресурси від ШІ представлені

на рисунку 4.10.

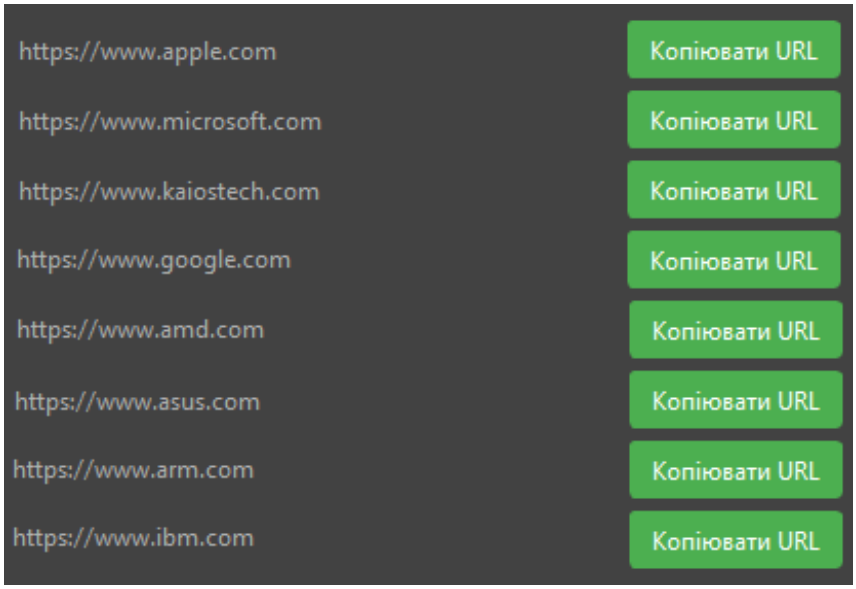

Рисунок 4.10 – Рекомендований контент

З’явиться список посилань, які безпосередньо пов’язані з тематикою збереженого ресурсу.

У центрі вікна користувач може ознайомитись з загальним списком закладок та папок. Є можливість обрати тип відображення: «список» чи «плитки». Існує спеціальне поле для пошуку окремої закладки серед інших. Присутні наступні розділи:

- ступінь важливості;
- назва;
- тег;
- дата;
- URL.

Сортування передбачає наступні можливості: розташування закладок від найважливіших чи від найменш важливих, сортування за назвою, за тегом, датою та URL. Користувач має широкий вибір розділення закладок за різними категоріями для більш швидкого та якісного пошуку інформації.

Комбінування цих параметрів чітко відокремлює одні ресурси від інших та дозволяє систематизувати збережені дані за різними критеріями для візуального огляду. Список закладок та папок представлений на рисунку 4.11.

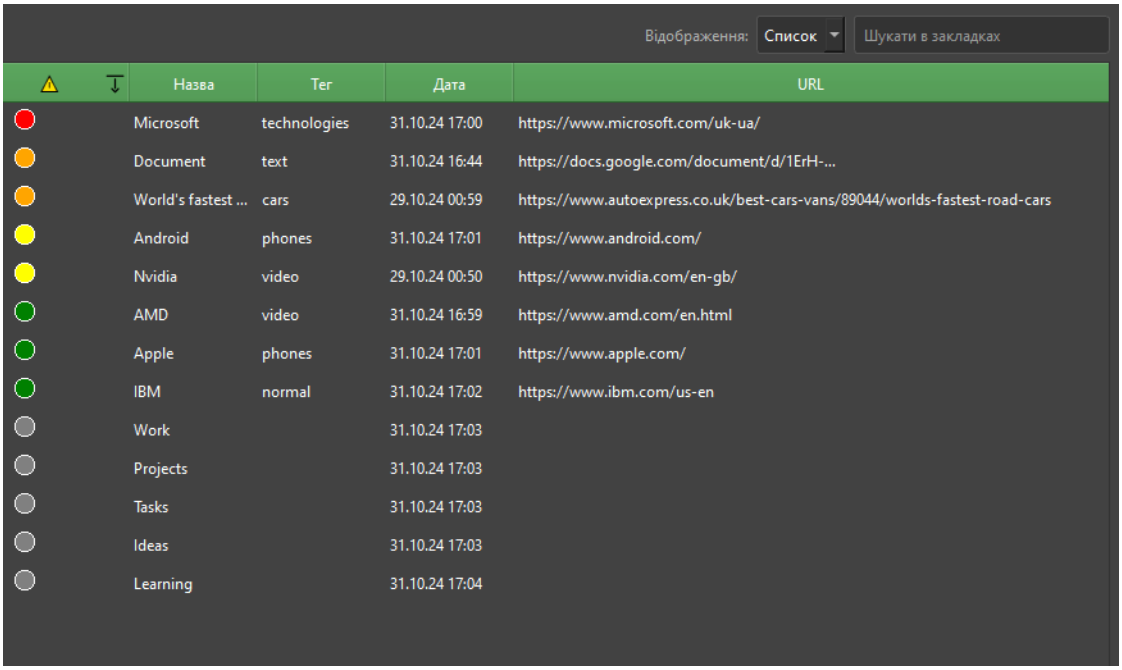

Рисунок 4.11 – Збережені ресурси

Для виходу з програми достатньо натиснути на хрестик у правому верхньому куті вікна чи перейти до «Файл» та натиснути на кнопку «Вийти».

РЕЗУЛЬТАТИ ТЕОРЕТИЧНИХ ТА ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

Огляд існуючих програмних рішень сформував базове уявлення стану проблеми щодо актуальності програмного забезпечення з керування закладками. Була виявлена відсутність суттєвих функціональних можливостей, які значно підвищують процес керування закладками.

Проведений аналіз наукової літератури та інтернет-джерел дозволив розробити підхід до вирішення обраного питання та оцінити перспективи потенційної розробки програми. Був оцінений розмір проєкта, його потенційні можливості та перспективи розвитку. На цій основі було побудовано ахітектурний підхід до майбутнього поєкту, який цілоком би відповідав вимогам програми.

Огляд документації мови програмування (Python), фреймворків та додаткових технологій дозволив сформувати план програмної реалізації засобу та взаємодію між різними його частинами.

Розроблена програма реалізує всі заплановані функціональні можливості. Було проведене тестування засобу, яке пройшло успішно. Інтеграція штучного інтелекту дозволила значно розширити можливості засобу та створити підґрунтя для подальшого розширення проєкта як з точки зору функціоналу, так і з точки зору взаємодії з віртуальним помічником у вигляді ШІ.

Проведене дослідження виявило актуальність розробки програмного засобу для управління закладками, та полягало в реалізації інноваційних функцій, а також їх перевірки на реальних ситуаціях. Створені програмні можливості є універсальними, а використання штучного інтелекту в цій галузі дозволяє відійти від стандартних реалізацій подібного роду засобів.

Дослідження підтвердило можливість розробки комплексної програми з управління закладками та важливу роль ШІ, як невід’ємної її складової.

ВИСНОВКИ

Результатом проведеного дослідження є розроблений спеціалізований програмний засіб з керування закладками. Програма повністю відповідає всім передбачуваним вимогам. Був реалізований функціонал, який значно покращує взаємодію між користувачем та засобом, підвищує ефективність організації закладок і розширює можливості програми завдяки штучному інтелекту. Засіб виконує всі функції та має перспективу розширення можливостей.

Теоретична частина дослідження, яка представляла собою процес оцінки актуальності теми та виявлення певних проблем і питань, пов’язаних з функціоналом існуючих рішень на ринку. Було встановлено, що широкі можливості не розкривали весь потенціал процесу керування закладками, через те, що були відсутні важливі елементи взаємодії з користувачем та контентом. Практична частина дослідження полягала в розробці програмного засобу для вирішення виявлених проблем.

Проведений огляд наукових джерел сприяв формуванню підходу до практичного вирішення питання, шляхом аналізу технологій та потенційного розміру проєкта. Обрана мова програмування, фреймворк та інші засоби дозволили знайти оптимальне рішення питання щодо програмного забезпечення як з точки зору безпосередньої реалізації, так і з точки зору його потенційного розвитку: додання функціоналу до вже розробленого проєкту.

На основі проведених методів наукового дослідження, таких як накопичення фактів (збір інформації щодо існуючих рішень), описування фактів (суть отриманої інформації), аналіз фактів (виявлення питань та проблем), обґрунтування наукових висновків (результати дослідження), вибір наукових рекомендацій (шляхи вирішення проблем), експериментальна перевірка рекомендацій (тестування програми) була реалізована практична

частина дослідження – програмний засіб.

Вітчизняні програмні рішення орієнтуються на інші функціональні можливості та не фокусуються на управлінні закладками. У таких програмах можливості зі збереження посилань є неосновними. Світові аналоги, такі як Raindrop, Pocket та Bookmark OS мають широкий функціонал, але вони не дають можливості використовувати ШІ повною мірою та не мають деяких важливих реалізацій.

Розроблений програмний засіб має перевірку на активність посилань з відповідним повідомленням, що реалізована не у всіх аналогах. Робота офлайн є важливою функцією, але вона майже відсутня в сучасних програмах такого типу, але присутня в розробленому рішенні.

Рекомендований контент, що присутній в застосунку Pocket, не має чіткого зв’язку з користувачем, але реалізована взаємодія з ШІ у розробленому засобі, орієнтується тільки на збережені посилання. Майже всі аналоги дозволяють зберігати неіснуючі (неправильні) посилання, які є неактуальною інформацією. Перевірка посилання в розробленій програмі не дозволяє зберегти неактивне посилання та одразу відбувається сповіщення про це користувача.

Розроблене програмне забезпечення можна розглядати як альтернативу вже існуючим рішенням кафедр ХНУРЕ. Окремі аспекти дослідження були висвітлені в відповідних наукових конференціях. Подальший аналіз у цій галузі може базуватися на основі реалізованої програми в напрямку нового функціоналу, розширення можливостей штучного інтелекту, його ролі щодо взаємодії з користувачем та збереженим контентом.

Засіб може бути використаний в навчальному процесі університету. Це може бути здійснено шляхом обрання відповідного подальшого розвитку програми. У такому випадку, виникає необхідність формування основної задачі засобу, його контенту та ролі ШІ. Наприклад, це може бути програмним забезпеченням для зберігання наукової літератури у вигляді посилань, де штучний інтелект буде грати роль помічника у подборі ресурсів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. A bookmarking service for organizing and sharing URLs / R. Kellerand and others; Computer Networks and ISDN Systems. – Amsterdam, 1997. – 1103–1114 p.

2. Barry P. Head First Python: A Learner's Guide to the Fundamentals of Python Programming, A Brain-Friendly Guide: O'Reilly Media. Sebastopol, 2023. 663 p.

3. CropSight: a scalable and open-source information management system for distributed plant phenotyping and IoT-based crop management / D. Reynold and others; GigaScience. – Oxford, 2019. – 14–15 p.

4. Development of a digital twin operational platform using Python Flask / M. Bonney and others; Data-centric Engineering. – Cambridge, 2022. – 2–3 p.

5. Doskas C. The Python Programming: Processing NVD Data: ISSA Journal. Silver Spring, 2020. – 37–40 p.

6. Gatto R. FastAPI vs Flask: The Ultimate Showdown of Python Web Frameworks! // Medium. 2023. 30 травня. URL: https://medium.com/@romulo.gatto/fastapi-vs-flask-the-ultimate-showdown-of-python-web-frameworks-1f2700ac8852 (дата звернення: 02.11.2024).

7. Flask vs. Django: Which Framework to Choose? // Intellectsoft. 2023. 5 жовтня. URL: https://www.intellectsoft.net/blog/flask-vs-django/ (дата звернення: 01.11.2024).

8. Gaddis T. Starting Out with Python: Pearson. London, 2017. 744 p.

9. Getting Started with MySQL // dev.mysql.com. 2024 URL: https://dev.mysql.com/doc/mysql-getting-started/en/ (дата звернення: 01.11.2024).

10. Graphical User Interfaces with Tk // python.org. 2024. 10 листопада. URL: https://docs.python.org/3/library/tk.html (дата звернення: 01.11.2024).

11. Hetland M. L. Beginning Python: From Novice to Professional: Apress.

New York, 2017. 559 p.

12. Lacey N. Tkinter GUI : Part I: Learning Python. New York, 2019. – 110–123 p.

13. Python 3.13.0 documentation // python.org. 2024 URL: https://www.python.org/doc/ (дата звернення: 02.11.2024).

14. Baweja C. Python and MySQL Database: A Practical Introduction // Real Phyton. URL: https://realpython.com/python-mysql/ (дата звернення: 01.11.2024)

15. Mahnken M. Python: Everything a Beginner Needs to Know // CourseReport. 2024. 12 липня. URL: https://www.coursereport.com/blog/what-is-python-programming – (дата звернення: 02.11.2024).

16. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming: O'Reilly Media. Sebastopol, 2022. 1012 p.

17. Regalado M. Reference Service in a Digital Age: Papers Based on the Library of Congress Institute: Reference & User Services Quarterly. Washington, 1998. 120 p.

18. Pertile F.G. Review of Flask, From a Django Developer // Giuliano Pertile. 2022. 9 липня. URL: https://www.giulianopertile.com/blog/review-of-flask-from-a-django-developer/ (дата звернення: 02.11.2024).

19. Review of the Pocket App for Chrome & Best Alternatives in 2024 // Workona. 2024. URL: https://workona.com/reviews/pocket-app-alternatives-for-chrome/ (дата звернення: 02.11.2024).

20. Rodriguez C. Generative AI Foundations in Python: Discover key techniques and navigate modern challenges in LLMs: Packt Publishing. Birmingham, 2024. 190 p.

21. SBML2HYB: a Python interface for SBML compatible hybrid modeling / J. Pinto and others; Bioinformatics. – Oxford, 2023. – 21–24 p.

22. Serverance C. Python for Everybody: Exploring Data in Python 3: CreateSpace Independent Publishing Platform. Scotts Valley, 2016. 248 p.

23. Sherman C. A bookmark manager roundup: Online. Medfold, 1999. –

56– 62 p.

24. Singh V., B. Prakash, U. Kumari. ChatBot using Python Flask: IEEE. New York, 2023. – 8–9 p.

25. Theis T. Getting Started with Python: Step-by-Step Guide for Beginners to Learn Core Concepts and Build Real-World Python Applications: Rheinwerk Computing. Bonn, 2024. 475 p.

26. User's Guide: Flask. 2010. URL: https://flask.palletsprojects.com/en/stable/#user-s-guide (дата звернення: 02.11.2024).

27. Deery M. What Is Flask and How Do Developers Use It? A Quick Guide // CF BLOG. 2023. 30 серпня. URL: https://careerfoundry.com/en/blog/web-development/what-is-flask/ (дата звернення: 02.11.2024).

28. Lakatos D. What is Tkinter in Python? // Medium. 2024. 11 червня. URL: https://medium.com/@lktsdvd/what-is-tkinter-in-python-7c39c8b48a40 (дата звернення: 01.11.2024).

29. Yves D. Using Machine Learning To Predict Student Performance: Guilford College Thesis Collection. Greensboro, 2017. – 28–30 p.

30. Мякшин, А.С. Програмне забезпечення зі збереження, організації та керування закладками з використанням штучного інтелекту / А.С. Мякшин // Матеріали V Міжнародної студентської наукової конференції Чернівці, 22 грудня, 2023. – Чернівці: Молодіжна наукова ліга, 2023. – С. 262-264.

31. Мякшин А.С. Програмне забезпечення зі збереження, організації та керування закладками з використанням штучного інтелекту / А.С. Мякшин // Матеріали XXVIII Міжнародний молодіжного форуму Харків, 16-18 квітня, 2024. – Харків: ХНУРЕ, 2024. – С.38-40.

32. Мякшин А.С. Програмне забезпечення зі збереження, організації та керування закладками з використанням штучного інтелекту / А.С. Мякшин // Матеріли Всеукраїнської Інтернет-конференції молодих учених і студентів Івано-Франківськ, 10 жовтня, 2024. – С. 46-47.