

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Розробка Телеграм бота для управління електронними книгами»

Виконав студент групи ПЗк-20-1
САРГАН Василь Ігорович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
Власенко Олег Васильови

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення

Андрій МОРОЗОВ

«2» серпня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: Сарган Василь Ігорович

Керівник роботи: Чижмотря Олексій Володимирович

Тема роботи: **«Розробка Телеграм бота для управління електронними книгами»**,

затверджена Наказом закладу вищої освіти від **«2» серпня 2023 р., №326с**

Вихідні дані для роботи: Розробка Телеграм бота для управління електронними книгами. Вимоги до функціональності, дизайн, каталог товарів, функціональні вимоги.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	02.08.2023р.	02.08.2023р.
2	Чижмотря О.В.	02.08.2023р.	02.08.2023р.
3	Чижмотря О.В..	02.08.2023р.	02.08.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання роботи	Примітка
1	Аналіз вимог до програмного продукту	3 серпня 2023 – 4 серпня 2023	Виконано
2	Пошук, огляд та аналіз аналогічних продуктів	5 серпня 2023 – 7 серпня 2023	Виконано
3	Вивчення матеріалу по конкретній тематиці	8 серпня 2023 – 11 серпня 2023	Виконано
4	Планування структури сайту	12 серпня 2023 – 18 серпня 2023	Виконано
5	Проведення структурно функціонального моделювання процесів	19 серпня 2023 – 20 серпня 2023	Виконано
6	Розробка проекту	21 серпня 2023 – 24 серпня 2023	Виконано
7	Аналіз одержаних результатів	25 серпня 2023 – 25 серпня 2023	Виконано
8	Оформлення пояснювальної записки	26 серпня 2023 – 29 серпня 2023	Виконано
9	Попередній захист	25 серпня 2023	Виконано

Реферат

Тема кваліфікаційної роботи бакалавра: "Розробка Телеграм бота для управління електронними книгами ". Пояснювальна записка складається зі вступу, 3 розділів, висновків, списку використаних джерел із 19 найменувань. В роботі наведено 6 рисунків. Загальний обсяг роботи – 74 сторінок, у тому числі 54 сторінки основного тексту, 2 сторінки списку використаних джерел.

Метою даної кваліфікаційної роботи є розроблення та реалізація Телеграм бота для управління електронними книжками. В основу роботи покладено створення зручного та ефективного інструменту, який допоможе користувачам у керуванні своїми електронними книжками, забезпечуючи їм зручний доступ до літературних творів та полегшуючи процес читання.

Ключові слова: ТЕЛЕГРАМ БОТ, УПРАВЛІННЯ ЕЛЕКТРОННИМИ КНИЖКАМИ, ІНТЕРНЕТ-МАГАЗИН КНИЖОК, ЕЛЕКТРОННА БІБЛІОТЕКА, ЧИТАЦЬКИЙ ДОСВІД, ФУНКЦІОНАЛЬНІСТЬ БОТА, ТЕХНОЛОГІЇ ТЕЛЕГРАМ АРІ, ПРОЕКТУВАННЯ І РОЗРОБКА, АНАЛІЗ СУЧАСНОГО СТАНУ, КЕРУВАННЯ ЛІТЕРАТУРОЮ, БАЗА ДАНИХ КНИЖОК, ІНТЕРФЕЙС КОРИСТУВАЧА, ЗРУЧНІСТЬ ЧИТАННЯ, ЕФЕКТИВНЕ УПРАВЛІННЯ, ОПТИМІЗАЦІЯ ЧИТАННЯ, БОТ ДЛЯ КНИЖОК, ОНЛАЙН ДОСТУП ДО ЛІТЕРАТУРИ, КНИЖКОВИЙ ДОДАТОК, КОМУНІКАЦІЯ З БОТОМ, КОРИСТУВАЦЬКИЙ ДОСВІД

					ІПЗ.КР.Б – 121 – 23 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка Телеграм бота для управління електронними книгами Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		В.І.Сарган						
Керівник		О.В. Чижмоторя					4	101
Рецензент		О.В. Власенко				Житомирська політехніка, група ІПЗК-20-1		
Зав. каф.		А.В. Морозов						

ABSTRACT

Bachelor's Thesis Topic: "Development of a Telegram Bot for Managing E-books". The explanatory note consists of an introduction, 3 chapters, conclusions, a list of used sources with 19 titles. The thesis includes 6 sections. The total volume of the work is 60 pages, including 35 pages of the main text and 2 pages of the list of used sources.

The purpose of this qualification work is to develop and implement a Telegram bot for managing e-books. The foundation of the work is based on creating a convenient and efficient tool that will assist users in managing their e-books, providing them with easy access to literary works and facilitating the reading process.

Keywords: TELEGRAM BOT, E-BOOK MANAGEMENT, ONLINE BOOKSTORE, DIGITAL LIBRARY, READING EXPERIENCE, BOT FUNCTIONALITY, TELEGRAM API TECHNOLOGIES, DESIGN AND DEVELOPMENT, ANALYSIS OF CURRENT STATE, LITERARY MANAGEMENT, BOOK DATABASE, USER INTERFACE, READING CONVENIENCE, EFFICIENT MANAGEMENT, READING OPTIMIZATION, E-BOOK BOT, ONLINE LITERARY ACCESS, BOOK APPLICATION, BOT COMMUNICATION, USER EXPERIENCE.

		<i>В.І. Сарган</i>			<i>ІІЗ.КР.Б – 121 – 23 – ІІЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>5</i>

Зміст

Вступ	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Опис проекту та постановка задачі.....	10
1.2. Призначення і мета додатку.....	12
1.3. Огляд існуючих програмних продуктів	15
1.4. Обґрунтування та вибір технологій для розробки	21
Висновок до першого розділу.....	23
РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ	25
2.1. Визначення варіантів використання та структура системи.....	25
2.2. Проектування та реалізація алгоритмів роботи системи.....	30
2.3 Реалізація додатку.....	37
Висновки до другого розділу.....	53
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ.....	54
3.1. Встановлення та налаштування програми	54
3.2. Структура інтерфейсу та порядок роботи з системою	56
Висновки до третього розділу	58
ВИСНОВОК	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	63
ДОДАТКИ.....	65

Вступ

У світі стрімкого технологічного розвитку та зростаючого інтересу до електронних книг, питання організації та управління цифровою бібліотекою набуває нового виміру. Електронні книги стали не просто альтернативою паперовим книгам, вони відкривають перед нами цілий екосистему можливостей – від швидкого доступу до будь-якої літератури зі світових бібліотек до можливості активної спільноти навколо текстів.

Від зручного зберігання електронних книг до підсумовування та обміну думками – цифровий формат надав книжкам новий статус, який вимагає від нас нового погляду на їх управління та використання. Сучасні цифрові бібліотеки вже не обмежуються лише зберіганням текстів; вони надають можливість взаємодії з контентом на новому рівні.

Завдяки цифровим технологіям користувачі можуть швидко знаходити потрібну інформацію, використовуючи пошукові функції та фільтри. Це особливо корисно в освітніх та наукових цілях, коли необхідно охопити великий обсяг матеріалу. Крім того, можливість підсвічування, роблення позначок та збереження виділених фрагментів дозволяє читачам структурувати свої думки та з легкістю повертатися до важливих моментів.

Зі збільшенням обсягів цифрового контенту виникає потреба в інструменті, який допоможе не лише зберігати електронні книги, але й зробить процес організації та взаємодії з ними максимально зручним та ефективним. Важливо мати засіб, що дозволить легко відстежувати, яку літературу ми вже читали, що плануємо прочитати у майбутньому, а також які з книг відповідають нашим інтересам та потребам найкраще.

У цьому контексті, цей дипломний проект націлений на розробку рішення, яке відповідає вищезгаданій потребі. Проект передбачає створення телеграм-бота-менеджера книг, який буде допомагати користувачам

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
						<i>7</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		

організовувати їхні цифрові бібліотеки, забезпечуючи зручний інтерфейс для додавання, відстеження та пошуку книг за різними параметрами.

Цей телеграм-бот стане вірним помічником у створенні особистого простору для читання, допомагаючи зберігати всі ваші електронні книги в одному місці. Він дозволить легко додавати нові книги, зберігати їхні метадані, такі як назва, автор, жанр, дати читання тощо. Крім того, бот надасть можливість відзначати книги, які ви вже прочитали, і вказувати ті, які плануєте прочитати в майбутньому.

Завдяки інтегрованому пошуку та фільтрам, бот допоможе швидко знаходити потрібні книги в вашій бібліотеці, спираючись на різні критерії, такі як автор, жанр чи ключові слова. Також можливість отримувати рекомендації на основі ваших попередніх виборів сприятиме розширенню вашого читацького кругозору.

Загалом, цей телеграм-бот вирішує важливу задачу організації та ефективного управління цифровою бібліотекою. Він допоможе створити структурований та організований простір для читання, сприяючи зручному взаємодії з літературою та поглибленню читацького досвіду.

Основна мета аналізу предметної області полягає в ідентифікації потреб та вимог користувачів до бота-менеджера книг. Досліджуючи існуючі рішення та їхні переваги, а також аналізуючи тенденції на ринку електронних книг, можна визначити ключові функціональні та дизайнерські аспекти, які необхідно врахувати при розробці бота.

Це дозволить створити високоефективний та зручний інструмент, що відповідає потребам сучасних читачів та їхньому способу сприйняття книг у цифровому форматі. Аналіз попередніх рішень допоможе виявити сильні сторони та можливі недоліки, що можуть бути уникнуті або враховані для покращення користувацького досвіду.

Враховуючи тенденції у світі електронних книг, зокрема збільшення ролі соціального взаємодії та персоналізованого підходу до користувача, бот

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>8</i>

може надати можливість обміну рецензіями, рекомендаціями та взаємодією з іншими користувачами. Це покращить не лише організацію бібліотеки, а й дозволить читачам більш глибоко взаємодіяти з контентом та одне з одним.

Усі ці аспекти важливі для успішної реалізації проекту та створення інструменту, який буде не лише ефективним, але й зручним та приємним у використанні для широкого кола користувачів.

Зіставляючи отримані дані з можливостями та обмеженнями платформи Telegram, можна створити належний аналіз, який визначить основні напрямки проектування та реалізації бота. У цьому розділі будуть розглянуті ключові аспекти вибору функціональності, яка відповідатиме потребам користувачів та забезпечуватиме зручний та ефективний процес управління електронними книгами. Результатом аналізу буде чітке визначення спектру можливостей бота, яке допоможе досягти поставленої мети та створити продукт, який зможе задовольнити потреби різноманітних користувачів..

		<i>В.І. Сарган</i>			<i>ІІЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
						9
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис проекту та постановка задачі

У сучасному світі, насиченому інформацією та можливостями, читання електронних книг відіграє важливу роль у пізнанні, розвитку та розвагах. Однак із зростанням кількості доступних матеріалів виникає необхідність у зручному і ефективному засобі для організації та керування цими ресурсами. Саме в цьому контексті виникає необхідність у створенні "Телеграм бот-менеджера книжок" - інноваційного рішення, спрямованого на оптимізацію читацького досвіду та створення зручного середовища для взаємодії з електронними книгами.

Цей бот-менеджер книжок має на меті забезпечити користувачів засобами для організованого та систематизованого зберігання, відстеження та взаємодії з електронними книгами. Його функціональні можливості дозволяють додавати книги до особистого каталогу, відзначати прочитані, планувати майбутні читання, а також швидко знаходити потрібні матеріали завдяки пошуку та фільтрам.

Крім цього, бот може надати рекомендації на основі попередніх виборів користувача та аналізу його читацьких пристрастей. Взаємодія з іншими користувачами, обговорення книг та обмін рецензіями роблять читацький досвід більш соціальним та захоплюючим.

Великий потенціал цього інноваційного рішення полягає у тому, що воно допомагає зробити читання більш організованим, особистим та збагаченим. Воно забезпечує зручний інструмент для зберігання та взаємодії з електронними книгами, надаючи користувачам можливість насолоджуватися своїми читацькими захопленнями в новому, збалансованому форматі.

Опис проекту:

"Телеграм бот-менеджер книжок" є інтелектуальним додатком, створеним для полегшення усіх аспектів управління електронними книгами.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>10</i>

Цей бот не обмежується простим списком книжок, він створює інноваційний підхід до організації, зберігання та взаємодії з цифровими текстами. Він створює можливість не просто зберігати книги, але й глибше вникати в літературні роботи, спілкуватися з іншими читачами та збагачувати свій особистий ріст завдяки здобутим знанням.

Завдяки інтелектуальному підходу, бот допомагає створити особливий літературний екосистему, де читачі можуть відкривати для себе нові твори, обговорювати їх з іншими, аналізувати та рефлексувати над прочитаним. Відзначення книг, планування майбутніх читань та взаємодія з іншими користувачами роблять процес читання більш цікавим та змістовним.

Цей інтелектуальний додаток відкриває нові можливості для активної взаємодії з літературою та спільнотою читачів, сприяючи особистому та інтелектуальному росту. Він перетворює читання електронних книг на більш глибокий та збагачуючий досвід, надаючи кожному користувачеві зручні інструменти для насолоди літературою та розширення своїх знань.

Постановка задачі:

1. **Аналіз вимог користувачів:** Ретельне дослідження та аналіз потреб та вимог користувачів становить основу розробки. Визначення ключових можливостей, інтерфейсу та способів взаємодії.
2. **Розробка функціональності:** Визначення основних можливостей бота, таких як створення бібліотеки, класифікація книжок за різними параметрами, створення списків для читання тощо.
3. **Проектування інтерфейсу:** Розробка зручного та інтуїтивно зрозумілого інтерфейсу, який забезпечить комфортну взаємодію користувачів з ботом та їхніми книжками.
4. **Реалізація бота:** Розробка програмної частини бота, його підключення до платформи Telegram та тестування на різних сценаріях взаємодії.

5. **Розвиток спільноти:** Активна промоція та популяризація бота серед читачів, залучення відгуків та рекомендацій для подальшого вдосконалення.
6. **Оптимізація та підтримка:** Постійна вдосконалення функціональності, врахування побажань та зауважень користувачів, забезпечення стабільної та надійної роботи бота.

Цей проект не лише прагне раціоналізувати читацький досвід, а й створити спільноту, де читачі можуть спілкуватися, рекомендувати та ділитися своїми літературними враженнями. "Телеграм бот-менеджер книжок" прагне змінити спосіб, яким ми взаємодіємо зі словесним мистецтвом, зробивши його більш цікавим, осмисленим та доступним для всіх.

Цей проект перетворює читання на більш активний та збагачений процес, де кожен читач може не лише споживати контент, але й активно спілкуватися, аналізувати, обмінюватися думками та надавати рекомендації. Шляхом створення спільноти навколо літератури, цей проект дозволяє людям з різних куточків світу з'єднуватися через спільну цікавість до книг.

Варто відзначити, що "Телеграм бот-менеджер книжок" відкриває нові можливості для вивчення, аналізу та рефлексії над текстами, завдяки чому читачі можуть глибше погрузнутися в світ літератури. Взаємодія з іншими читачами та обмін думками додає нову динаміку читацькому досвіду, роблячи його більш осмисленим та багатогранним.

У цілому, "Телеграм бот-менеджер книжок" перетворює читання на взаємодію, спілкування та знання, створюючи новий рівень зв'язку між читачами та літературою.

1.2. Призначення і мета додатку

У сучасному інформаційному суспільстві, де електронні книги стали неодмінною частиною нашого читацького досвіду, ефективне управління цим

		В.І. Сарган			ІПЗ.КР.Б – 121 – 23 – ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

цифровим контентом стає актуальнішим за будь-коли. Зростаючий обсяг доступних книжкових матеріалів, різноманітні жанри та тематики, а також важливість організованого підходу до читання спонукають до розробки інноваційних інструментів для оптимізації цього процесу.

У цьому контексті, додаток "Телеграм бот-менеджер книжок" націлено на вирішення ключових проблем управління електронними книгами, щоб забезпечити користувачам зручність, ефективність та збагачення читацького досвіду. Цей бот створює спеціальний простір для організації та зберігання електронних книг, дозволяючи користувачам легко додавати нові твори, відстежувати прочитані та плановані книги, а також швидко знаходити потрібні матеріали.

Крім того, бот може стати платформою для обміну рецензіями, рекомендаціями та обговоренням літературних вражень, забезпечуючи взаємодію між читачами та створюючи спільноту навколо книжок. В результаті, "Телеграм бот-менеджер книжок" допомагає покращити організованість, доступність та глибину читацького досвіду в цифровій епохі.

Мета додатку:

Метою цього додатку є створення всебічного та інтуїтивно зрозумілого інструменту, який допоможе користувачам зберігати, керувати та спілкуватися щодо їхніх електронних книжок. Додаток має на меті зробити процес читання більш організованим та цілеспрямованим, сприяючи кращому сприйняттю та осмисленню книжкового контенту.

Використовуючи цей додаток, користувачі матимуть зручний інструмент для структурування своєї цифрової бібліотеки, відзначення прочитаного та планування майбутніх книжкових пригод. Завдяки можливості обговорення, обміну враженнями та рецензіями, цей додаток також створює соціальний аспект читацького досвіду, допомагаючи користувачам отримати нові погляди та розуміння на літературні твори.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>13</i>

Такий підхід допомагає підвищити взаємодію з електронними книгами, зробити їх читання більш плідним та змістовним. Покращена організованість, доступність та взаємодія з контентом сприяють глибшому зануренню в літературні твори та розширюють спектр можливостей для осмислення та обговорення.

Призначення додатку:

1. **Організація книжкової колекції:** Додаток надає можливість користувачам зберігати всі свої електронні книги в одному місці, створюючи власну цифрову бібліотеку. Це дозволить уникнути розпорошеності книжок на різних пристроях та платформах.
2. **Ефективне управління читанням:** Додаток надає можливість відстежувати прочитані та непрочитані книги, додавати їх до списків для подальшого читання, а також відмічати улюблені та рекомендовані книги.
3. **Пошук та фільтрація:** Додаток дозволяє здійснювати швидкий та зручний пошук серед власних книг за авторами, жанрами, ключовими словами тощо. Фільтрація за різними критеріями допоможе знайти саме ті книги, які зараз потрібні.
4. **Взаємодія та спілкування:** Додаток створює можливість спілкування між користувачами, де вони можуть обмінюватися враженнями від книг, рекомендувати та ділитися літературними відкриттями.
5. **Організація освітнього процесу:** Додаток може використовуватися як інструмент для вивчення, занотовування важливих думок та підсумків під час читання, що сприяє глибокому розумінню матеріалу.

Загалом, мета та призначення додатку "Телеграм бот-менеджера книжок" полягають у створенні інноваційного рішення, що забезпечує

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>14</i>

легкість та організованість управління цифровими книгами, а також взаємодію та обмін думками серед читацької спільноти.

Цей додаток допомагає користувачам налагодити систематизоване зберігання електронних книг, відстежити їх прочитання, планування майбутніх читань та ділитися враженнями з іншими. Він створює зручний інструмент для оптимізації читацького досвіду, дозволяючи краще осмислити, аналізувати та насолоджуватися літературою.

Крім того, цей додаток сприяє взаємодії та обміну думками в читацькій спільноті. Він створює можливість обговорення книг, рекомендацій та рецензій, дозволяючи читачам ділитися своїми враженнями та отримувати нові погляди на літературні твори.

Отже, додаток "Telegram бот-менеджера книжок" відіграє важливу роль у покращенні читацького досвіду, забезпечуючи зручний інструмент для організації, управління та взаємодії з електронними книгами, а також сприяючи активному обміну думками та враженнями серед читачів.

1.3. Огляд існуючих програмних продуктів

В сучасному цифровому світі існує широкий асортимент програмних рішень, які призначені для кращого керування електронними книгами та поліпшення досвіду читання. Ці продукти відіграють надзвичайно важливу роль у спрощенні організації та зберігання електронних текстів, а також у створенні інструментів, що полегшують взаємодію з цифровим контентом. Дозвольте розглянути кілька з цих програмних рішень більш докладно:

1. **Calibre:** Calibre є відомим та популярним програмним рішенням для керування електронними книгами. Він дозволяє створювати бібліотеки електронних книг, додавати метадані, класифікувати книги за жанрами та авторами. Крім того, Calibre надає можливість конвертувати книги між різними форматами, щоб забезпечити сумісність з різними пристроями. Однак, сам по собі Calibre не має

інтеграції з платформою Telegram, що може ускладнити доступ до книг на різних пристроях.

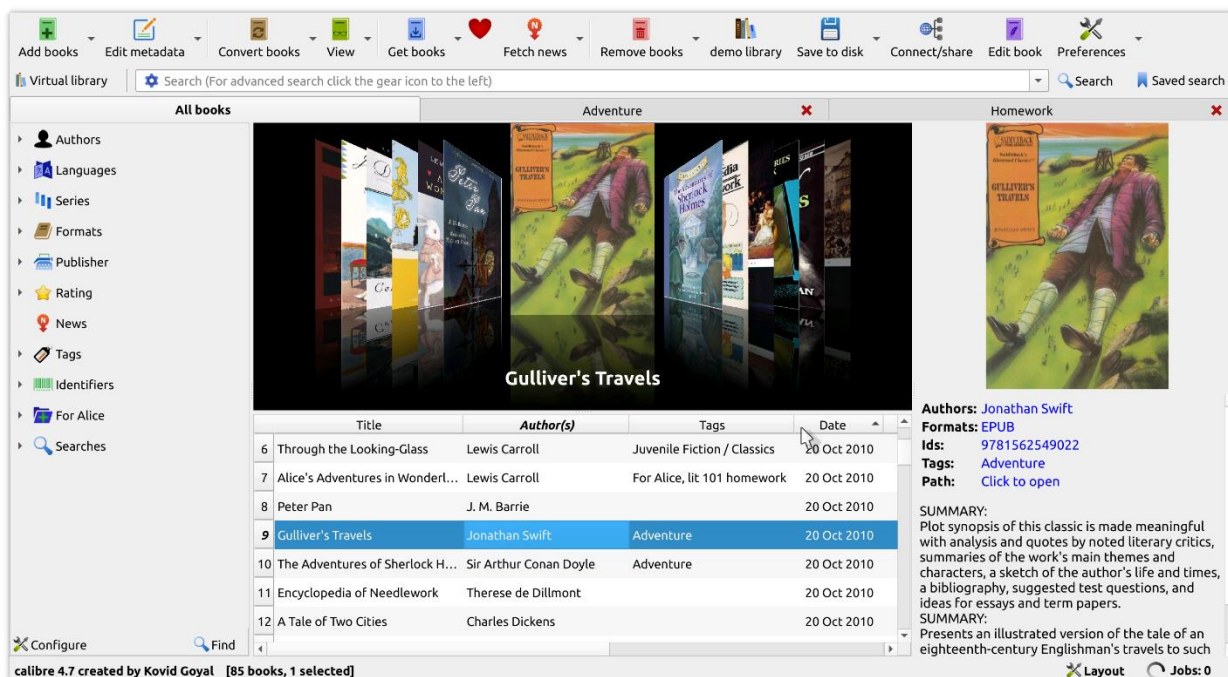


Рис 1.1 головне вікно calibre

2. **Goodreads:** Goodreads - це соціальна платформа для читачів, де користувачі можуть створювати списки прочитаних книг, ставити їм оцінки, писати рецензії та ділитися своїми враженнями. Система рекомендацій на Goodreads допомагає знаходити нові книги, які можуть вас зацікавити. Проте Goodreads не надає можливості для зберігання фактичних електронних книг, і для читання потрібно виходити на зовнішні додатки або пристрої.

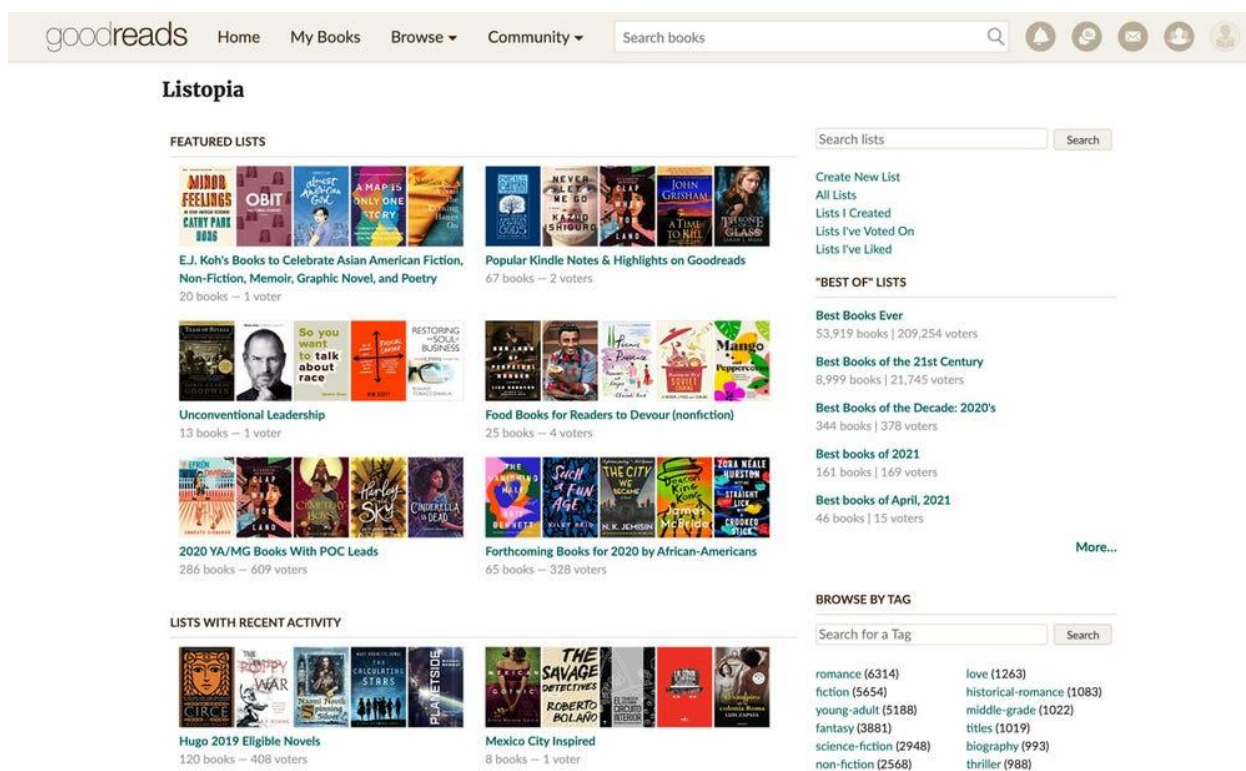


Рис 1.2 головне вікно goodreads

3. **LibraryThing:** LibraryThing - це інший варіант для створення власної онлайн-бібліотеки. Користувачі можуть додавати книги, розміщувати відгуки, спілкуватися з іншими читачами та знаходити рекомендації. LibraryThing звертає більше уваги на спільноту читачів, але також може бути корисним для організації книжкової колекції.



Рис 1.3 головне вікно LibraryThing

4. **Book Catalogue:** Це мобільний додаток для Android, який дозволяє вам створювати список ваших книг, використовуючи функцію сканування штрих-кодів. Він має базу даних, яка допомагає автоматично заповнювати метадані про книги. Однак, цей додаток також не має інтеграції з Telegram.



Рис 1.4 головне вікно Book Catalogue

5. **BookFusion:** BookFusion - це платформа для управління електронними книгами з можливістю читання на різних пристроях. Вона надає можливість завантажувати, організовувати та читати електронні книги. Також, ви можете створювати закладки, підсвічування та навіть ділитися книгами з іншими користувачами.

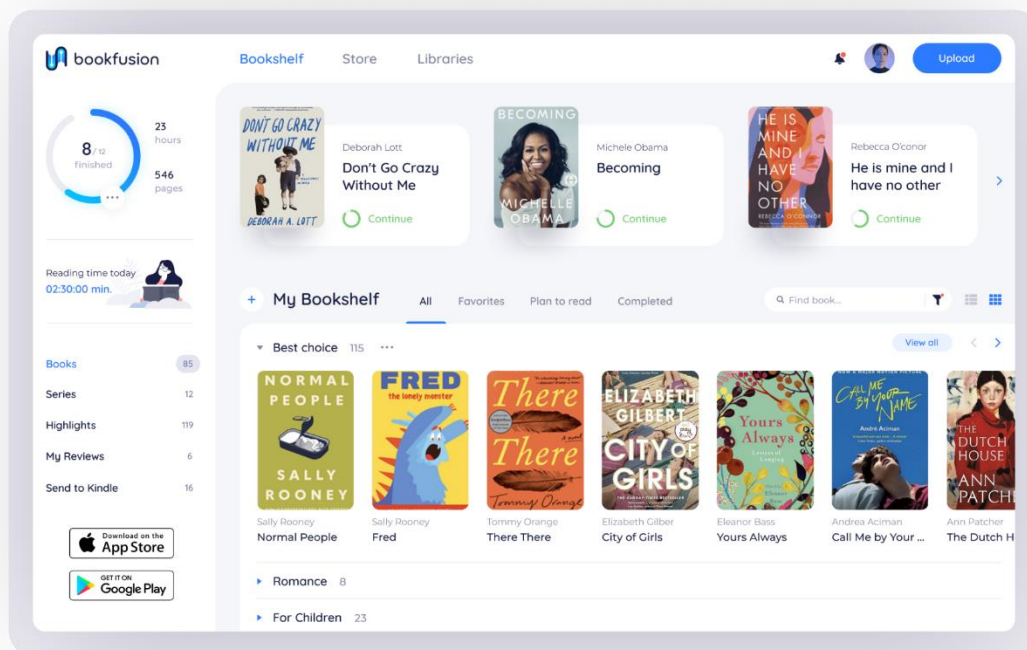


Рис 1.5 головне вікно BookFusion

Існуючі програмні продукти, безумовно, надають користувачам інструменти для управління електронними книгами. Однак багато з них стикаються з обмеженням - вони не завжди мають інтеграцію з популярними платформами, такими як Telegram. Ось де "Telegram бот-менеджер книжок" виступає як по-справжньому інноваційний продукт.

Цей бот є результатом вдалого поєднання функціональності організації та управління електронними книгами з популярністю та зручністю використання платформи Telegram. Він дозволяє користувачам не лише зберігати та впорядковувати свою цифрову бібліотеку, але й отримувати доступ до неї з будь-якого пристрою, де є доступ до Телеграму. Це важливо для сучасних людей, які часто переміщуються та шукають зручний спосіб зберігати та отримувати доступ до своїх електронних книг.

Більше того, "Telegram бот-менеджер книжок" створює можливість для спілкування та обміну думками з іншими читачами. Він створює віртуальне

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>20</i>

простір для обговорення книг, обміну рецензіями та рекомендаціями, надаючи користувачам можливість збагачувати свій читацький досвід завдяки новим поглядам та інсайтам.

Отже, "Телеграм бот-менеджер книжок" не просто розвиває можливості наявних програм, але й змінює спосіб, яким ми взаємодіємо з електронними книгами та спілкуємося щодо них, роблячи читання більш інтерактивним та насиченим.

1.4. Обґрунтування та вибір технологій для розробки

Розглядаючи процес вибору мови програмування для створення "Телеграм бот-менеджера книжок", критичним етапом є аналіз вимог проекту, дослідження можливостей різних мов та їх придатність для вирішення конкретних завдань розробки. Особливо актуальною є здатність мови ефективно впоратися з тими завданнями, які виникають перед програмою. Давайте розглянемо це докладніше, щоб з'ясувати, чому мова програмування C++ може стати найкращим вибором для даного проекту:

1. **Продуктивність та швидкодія:** C++ славиться своєю високою продуктивністю, що особливо важливо для бота, який має надавати миттєві відповіді на запити користувачів. Читацький досвід залежить від швидкості реакції бота, тому використання швидкої мови програмування є ключовим фактором.
2. **Керування ресурсами:** Мова C++ надає розробникам можливість контролювати використання пам'яті та ресурсів. Це особливо важливо для бота, який повинен ефективно управляти даними про книги та користувачів. Керування пам'яттю дозволяє уникнути надмірного використання ресурсів.

3. **Багата екосистема бібліотек:** С++ має широкий вибір бібліотек та фреймворків, які значно спрощують розробку. Для реалізації функціональності бота, такої як взаємодія з платформою Telegram, збереження даних та інше, наявність готових інструментів може суттєво прискорити процес розробки.
4. **Масштабованість:** "Telegram бот-менеджер книжок" може зростати та розвиватися. С++ дозволяє створювати масштабовані програми, що може бути надзвичайно корисним у випадку, якщо потреби користувачів збільшаться з часом.
5. **Крос-платформеність:** Мова С++ підтримується на більшості платформ, що дозволяє розробляти додаток для різних операційних систем. Це важливо, якщо "Telegram бот-менеджер книжок" повинен бути доступний на різних пристроях та платформах.
6. **Багата спільнота розробників:** С++ має велику та активну спільноту розробників, що вказує на наявність багатьох ресурсів для навчання, підтримки та розв'язання проблем. Це може значно спростити розробку та допомогти вирішити труднощі, які можуть виникнути під час проекту.

Висновок:

Мова програмування С++ є відмінним вибором для відповідності потребам розробки "Telegram бот-менеджера книжок", оскільки вона забезпечує комплексні можливості, які важливі для успішної реалізації проекту. Необхідність у взаємодії з платформою Telegram та зручному інтерфейсі для управління книжковою колекцією вимагають мови, яка може забезпечити якісну роботу додатку.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
						22
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		

Мова С++ славиться своєю продуктивністю та високою швидкістю, що особливо важливо для "Телеграм бот-менеджера книжок", який повинен оперативно реагувати на запити користувачів та забезпечувати плавну взаємодію. Крім того, С++ володіє потужними можливостями для оптимізації використання ресурсів, що забезпечить стабільну та ефективну роботу бота навіть під час інтенсивного використання.

Важливо відзначити, що С++ має багатий набір готових бібліотек та інструментів, які можуть значно спростити процес розробки. Це дозволить команді розробників "Телеграм бот-менеджера книжок" зосередитися на створенні функцій, які додадуть значну вартість програмі та забезпечать зручний та різноманітний функціонал для користувачів.

Також важливо врахувати велику та активну спільноту розробників С++. Це означає доступ до безлічі ресурсів, які можна використовувати під час розробки, а також можливість швидко вирішувати будь-які труднощі, що можуть виникнути під час роботи над проектом.

Враховуючи усі ці фактори, можна впевнено стверджувати, що мова програмування С++ є ідеальним вибором для створення "Телеграм бот-менеджера книжок". Вона дозволить створити ефективний, функціональний та зручний додаток, який відповідатиме вимогам користувачів та забезпечить їм зручність та задоволення від взаємодії з електронними книгами.

Висновок до першого розділу

У сучасному світі, де технологічний розвиток несперечний, і інтернет став важливою складовою життя, виникає нова парадигма у доступі до знань та розвитку культури читання. Проект "Телеграм бот-менеджер книжок" відповідає викликам цього часу, віддзеркалюючи зміни у способах сприйняття та розповсюдження літературних творів.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>23</i>

У процесі аналізу предметної області було детально розглянуто потреби користувачів, вимоги до зручності та доступності книжок в електронному форматі. Моделювання та проектування "Телеграм бот-менеджера книжок" дозволило визначити оптимальний спосіб організації бібліотеки через використання платформи Telegram, що є популярною та зручною для багатьох користувачів.

Вибір мови програмування C++ для реалізації проекту виявився настільки обґрунтованим через її високу продуктивність, можливість оптимального управління ресурсами та доступ до багатофункціональних бібліотек. Ця мова забезпечує найкращу швидкодію та ефективність в контексті створення бота-менеджера книжок.

Порівняння з існуючими програмними рішеннями показало, що "Телеграм бот-менеджер книжок" може стати інноваційним вкладенням у сферу управління книжковими ресурсами. Його поєднання функціональності месенджера та можливостей електронного менеджера книг може надати користувачам зручний та сучасний інструмент.

Загалом, цей проект націлений на відповідь на актуальні потреби суспільства, що стрімко переходить до цифрового формату. "Телеграм бот-менеджер книжок" не лише створює зручний доступ до електронних книг, але й втілює концепцію сучасного інформаційного середовища, де доступність знань стає доступнішою та дружнішою завдяки технологіям.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>24</i>

РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

2.1. Визначення варіантів використання та структура системи

Варіанти використання програми "Телеграм бот-менеджер книжок"

Програма "Телеграм бот-менеджер книжок" є відмінним інструментом, який пропонує широкий спектр можливостей для зручного та ефективного управління електронною бібліотекою. Незалежно від віку та інтересів користувачів, цей бот створений для того, щоб забезпечити насолоду від читання та зробити процес взаємодії з літературою у цифровому форматі максимально зручним та задовільним.

Для початку, "Телеграм бот-менеджер книжок" дозволяє користувачам легко додавати нові електронні книги до своєї колекції. Це особливо зручно для тих, хто збирається читати книги на різні теми чи в різні періоди часу. Бот зберігає інформацію про кожну книгу, включаючи назву, автора та жанр, що полегшує орієнтування в колекції.

Однак "Телеграм бот-менеджер книжок" не обмежується лише організацією книг. Він також надає можливість відстежувати книги, які вже були прочитані, і ті, які планується прочитати у майбутньому. Такий підхід допомагає впорядковувати свій читацький список та зберігати контроль над тим, що вже вивчено, і що ще варто дізнатися.

Окрім того, бот створює можливість активної взаємодії з іншими читачами. Користувачі можуть рекомендувати книги одне одному, обговорювати враження від прочитаного та обмінюватися думками. Це відкриває можливість для формування спільноти літературних ентузіастів, які спільно досліджують світ книжок.

Отже, "Телеграм бот-менеджер книжок" дійсно надає широкий спектр можливостей для розширення та покращення читацького досвіду. Чи це організація колекції, відстеження прочитаного, спілкування з іншими

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>25</i>

читачами чи рекомендації книг - цей бот робить читання електронних книг більш збагаченим та задовільним.

Розглянемо деякі функції системи:

1. Додавання нових книг: Користувачі можуть легко додавати нові книги до своєї бібліотеки, завантажуючи фотографії обкладинок, вказуючи авторів та назви книг. Це робить процес організації книжкової колекції простим та приємним.
2. Персоналізований пошук: Програма надає можливість здійснювати пошук книг за різними параметрами, включаючи жанр, автора, рік видання тощо. Це допомагає знайти потрібну книгу навіть у великій бібліотеці.
3. Організація за категоріями: Користувачі можуть створювати власні категорії та колекції книг, що допомагає організувати їх за особистими уподобаннями. Наприклад, колекція "Для літа" або "Детективи".
4. Відстеження прогресу читання: Програма дозволяє позначати книги як прочитані, початі або незавершені. Це допомагає користувачам відстежувати свій читацький прогрес та організовувати свої літературні цілі.
5. Обмін книгами: Користувачі можуть обмінюватися книгами між собою, відправляючи запити на обмін та узгоджуючи деталі. Це сприяє взаємодії серед читацької спільноти.
6. Рекомендації та підбірки: Програма може надавати рекомендації щодо подібних книг на основі попередньої читацької активності користувача. Також можуть бути створені підбірки книг за різними темами.
7. Нагадування про читання: Користувачі можуть налаштувати нагадування про щоденне читання. Програма висилатиме повідомлення, нагадуючи відвести час для літературних насолод.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>26</i>

8. Зручне ведення літературного щоденника: Користувачі можуть відзначати враження від прочитаних книг, додавати відгуки та оцінки, що допомагає вести літературний щоденник та обмінюватися враженнями з іншими читачами.

Цей розмаїтий набір можливостей відкриває перед користувачами абсолютно новий рівень насолоди від читання електронних книг. Програма надає надзвичайно потужні інструменти для структурування, взаємодії та поглиблення читацького досвіду, тим самим створюючи надзвичайно зручний і інноваційний спосіб управління літературним багажем.

Завдяки можливостям програми, користувачі можуть відчути справжню впевненість в тому, що кожна електронна книга в їх колекції має своє місце та значення. Організація книг за жанрами, відстеження списку прочитаних і запланованих до читання, а також можливість активного спілкування з іншими читачами додають глибини і визначеності процесу читання.

Програма "Telegram бот-менеджер книжок" також стає місцем для зустрічей різних читацьких поглядів та інтерпретацій. Рекомендації від одного читача іншому, дискусії про популярні та недавно відкриті твори додають соціального аспекту до індивідуального акту читання.

Таким чином, завдяки "Telegram бот-менеджеру книжок", читачі можуть зануритися в захопливий світ електронних книг і насолоджуватися ним на повну. Він надає інноваційний і зручний інструмент для того, щоб літературний досвід був не лише вдома на полиці, але й завжди під рукою у вигляді вірного співтовариства та зручного управління.

Структура системи "Telegram бот-менеджер книжок"

Система "Telegram бот-менеджер книжок" представляє собою важливий крок у напрямку створення організованого та цілеспрямованого підходу до управління електронною бібліотекою. Її структура ретельно розроблена з метою забезпечення максимальної зручності для користувачів та ефективної взаємодії з книжковими ресурсами.

Однією з ключових переваг системи є її систематичний підхід до різних аспектів організації та управління електронними книгами. Вона не просто пропонує створення списку книг, але і структурує їх за жанрами, авторами, прочитаними та запланованими до читання, надаючи користувачам зручність і логічний порядок у їхньому читацькому досвіді.

Кожен компонент системи відіграє свою важливу роль у забезпеченні цілісності та функціональності. Інтерактивний інтерфейс для додавання нових книг, можливість позначати прочитані, робити плани на майбутнє та обмінюватися рекомендаціями з іншими користувачами робить процес використання системи захоплюючим та сприяє поглибленому сприйняттю літератури.

Загалом, система "Telegram бот-менеджер книжок" втілює концепцію організованого та інтерактивного підходу до читання електронних книг. Вона створює гармонійне поєднання функціональності та зручності, надаючи користувачам можливість насолоджуватися своєю літературною колекцією на повну та ділитися цим досвідом з іншими.

Розглянемо деякі компоненти системи:

1. Телеграм бот: Це головна точка доступу для користувачів до системи. Він приймає команди та повідомлення від користувачів через платформу Telegram. Логіка бота аналізує отримані дані та направляє їх на подальшу обробку в інших компонентах.
2. База даних: Централізоване сховище для зберігання інформації про книги та користувачів. База даних дозволяє зберігати дані про книги, включаючи назви, авторів, жанри, обкладинки та статуси прочитання. Це надає змогу швидкого та зручного доступу до інформації.
3. Логіка бота та обробка команд: Цей компонент відповідає за інтерпретацію команд та повідомлень, отриманих від користувачів

через бота. Він аналізує текстові запити та виконує відповідні дії, такі як додавання книг, пошук інформації, нагадування та інші.

4. Менеджер збереження даних: Цей компонент взаємодіє з базою даних та забезпечує збереження та витягування інформації про книги. Він гарантує консистентність та доступність даних, що є основою для ефективної роботи системи.
5. Модуль рекомендацій (за наявності): Якщо в системі присутній модуль рекомендацій, він аналізує історію читання користувача та робить рекомендації для подальших книг. Цей компонент може допомагати користувачам відкривати нові літературні шедеври.
6. Інтерфейс користувача та взаємодія: Користувачі взаємодіють з ботом через інтерфейс Telegram. Це можуть бути текстові повідомлення, кнопки вибору, команди для додавання книг тощо. Цей інтерфейс забезпечує зручну та інтуїтивну взаємодію з системою.
7. Модуль відстеження читання: Якщо присутній, цей компонент допомагає користувачам відстежувати прогрес читання книг, зберігаючи дані про кількість сторінок та дати початку та закінчення читання.
8. Модуль обміну книгами (за наявності): Якщо включений, цей компонент дозволяє користувачам обмінюватися книгами між собою через бота, спрощуючи процес обміну та координації.

Ця добре налагоджена структура інтегрує різноманітні функції та компоненти у впорядковану та логічну систему, що сприяє створенню збалансованого, ефективного та зручного інструменту для управління електронною бібліотекою. Всі компоненти взаємодіють між собою відповідно до задач та потреб користувачів, забезпечуючи гладке та безперебійне використання програми через платформу Telegram.

Кожна деталь системи має своє визначене місце та призначення, що сприяє зручності користування та уникненню заплутаних ситуацій. Плавна інтеграція різних функцій допомагає створити гармонійний інструмент, який може задовольнити потреби як новачків у читанні, так і відданих бібліофілів.

В цілому, ця структура дозволяє "Telegram боту-менеджеру книжок" демонструвати високу ефективність, гнучкість та відповідність вимогам користувачів. Вона створює не лише функціональний інструмент, але й відкриває широкий спектр можливостей для поглибленого і якісного читацького досвіду.

2.2. Проектування та реалізація алгоритмів роботи системи

Глибоке проектування та ефективна реалізація алгоритмів є важливим етапом у створенні системи "Telegram бот-менеджер книжок", який визначає її функціональність та продуктивність. Нижче розглянуті ключові алгоритми докладніше:

1. **Обробка вхідних повідомлень користувачів:** Алгоритм реалізується за допомогою регулярних виразів або аналізу натуральної мови. Він розпізнає ключові слова та фрази, визначає тип команди, здійснює перевірку коректності та виконує необхідні дії, такі як взаємодія з базою даних, генерація відповідей або виклик інших алгоритмів.
2. **Додавання та збереження книг:** Алгоритм обробляє дані, введені користувачем, і виконує перевірку на наявність книги у базі даних. Після підтвердження унікальності даних алгоритм додає книгу до бази, зберігаючи всю необхідну інформацію.
3. **Пошук книг:** Алгоритм пошуку включає в себе обробку параметрів пошуку, введених користувачем. Він перевіряє базу даних на наявність книг, що відповідають критеріям пошуку, та повертає результат у вигляді списку книг.

4. Менеджмент колекцій: Алгоритми для створення та управління колекціями книг здійснюють додавання та видалення книг до/з колекцій, а також надання користувачам змоги перегляду колекцій та їх вмісту.
5. Рекомендації (за наявності): Для алгоритму рекомендацій застосовуються методи аналізу даних, такі як колаборативний або контент-базований підхід. Він аналізує читацькі вподобання користувача та рекомендує подібні книги на основі їх характеристик.
6. Відстеження читання (за наявності): Алгоритм відстеження читання допомагає користувачам вказувати кількість прочитаних сторінок та визначати прогрес читання. Він також може встановлювати нагадування про даліше читання.
7. Обмін книгами (за наявності): Алгоритми обміну можуть включати механізми надсилання запитів, обробки підтверджень та узгодження деталей обміну між користувачами.
8. Нагадування про читання: Алгоритм нагадувань створює розклад нагадувань про читання з урахуванням вибраного користувачем часу та частоти.

Адекватне розуміння та глибокий аналіз принципів, на яких базуються ці алгоритми, є критично важливим для забезпечення успішної реалізації системи "Телеграм бот-менеджер книжок". Від цього розуміння залежить не лише правильна робота програмного забезпечення, але й здатність відповідати на потреби та очікування користувачів.

Глибоке вивчення алгоритмів дозволяє виявити потенційні питання та виклики, які можуть виникнути під час розробки та впровадження системи. Це допомагає уникнути непередбачених ситуацій і забезпечити високу стабільність та надійність програмного продукту.

Крім того, глибоке розуміння алгоритмів дозволяє знаходити оптимальні рішення та підходи до вирішення конкретних завдань, що стоять перед системою. Це впливає на її продуктивність, швидкодію та здатність відповідати на потреби користувачів без зайвих затримок чи недоліків.

У кінцевому підсумку, глибоке розуміння алгоритмів є фундаментальним кроком у напрямку створення високоякісної, ефективної та користувацько-орієнтованої системи "Телеграм бот-менеджер книжок", яка задовольняє потреби та вимоги сучасних користувачів.

Нижче наведений код логіки роботи базового функціоналу

```
void BotLogics::AddBook(TgBot::Message::Ptr message)
{
    InfoToSave obg(message->messageId, message->chat->id);
    ListAllAddingBook.push_back(obg);
    (SaveToFile(obg));
}

void BotLogics::ViewAllBook(TgBot::Bot &bot, std::int64_t chatID)
{
    ListReplyMessage.clear();
    if(ListAllAddingBook.empty()) {
        bot.getApi().sendMessage(chatID, "Nothing to read, maybe need add from all list?");
        return;
    }
    for (InfoToSave i : ListAllAddingBook) {
        if(chatID == i.GetChatID()) {

ListReplyMessage.push_back(InfoToDelete(bot.getApi().sendMessage(chatID, ".",
false, i.GetMessageID())->messageId, i.GetMessageID()));

        }
    }
}

void BotLogics::ViewReading(TgBot::Bot &bot, std::int64_t chatID)
{
    if (ListReadingBook.empty()) {
        bot.getApi().sendMessage(chatID, "Nothing reading book, maybe need add from all want read list?");
        return;
    }
    for (InfoToSave i : ListReadingBook) {
        if(chatID == i.GetChatID()) {

ListReplyMessage.push_back(InfoToDelete(bot.getApi().sendMessage(chatID, ".",
false, i.GetMessageID())->messageId, i.GetMessageID()));

        }
    }
}

void BotLogics::ViewReaded(TgBot::Bot &bot, std::int64_t chatID)
{

```

```

        if (ListReadedBook.empty()) {
            bot.getApi().sendMessage(chatID, "You dont readed books");
            return;
        }
        for (InfoToSave i : ListReadedBook) {
            if(chatID == i.GetChatID()) {
                bot.getApi().sendMessage(chatID, ".", false,
i.GetMessageID());
            }
        }
    }

    void BotLogics::DeleteBook(TgBot::Bot &bot, TgBot::Message::Ptr &message)
    {
        std::int32_t originid = FindIDMessageToDel(message->replyToMessage-
>messageId);
        if (originid == -1) {
            bot.getApi().sendMessage(message->chat->id, "Sorry dont find
message, please check your reply message");
            return;
        }
        for (std::list<InfoToSave>::iterator it = ListAllAddingBook.begin();
it != ListAllAddingBook.end(); it++) {
            if (it->GetMessageID() == originid) {
                ListAllAddingBook.erase(it);
                bot.getApi().sendMessage(message->chat->id, "message been
delete");
                break;
            }
        }
        SaveChange();
    }

    void BotLogics::EditBook(TgBot::Bot &bot, TgBot::Message::Ptr &message)
    {
        char whatid = CheckWhatIsEdit(message->text);
        if (whatid == -1) {
            bot.getApi().sendMessage(message->chat->id, "Sorry don`t
understand your teg. Please check and try again");
            return;
        }

        std::int32_t originid = FindIDMessageToDel(message->replyToMessage-
>messageId);
        if (originid == -1) {
            bot.getApi().sendMessage(message->chat->id, "Sorry dont find
message, please check your reply message");
            return;
        }

        if (ChangeList(whatid, originid)) {
            bot.getApi().sendMessage(message->chat->id, "Ok");
            for (InfoToSave i : ListReadingBook) {
                if (i.GetChatID() == message->chat->id) {
                    bot.getApi().pinChatMessage(message->chat->id,
i.GetMessageID());
                }
            }
        }
        else {
            bot.getApi().sendMessage(message->chat->id, "Bed. Please check
your message");
            return;
        }
    }

```

```

    }
    SaveChange();
}

void BotLogics::LoadData()
{
    LoadFromFile load;
    auto tmp = load.GetList();
    for (InfoToSave i : tmp) {
        printf("Load Message ID - %d. Message chat ID - %ld. Teg - %d\n",
i.GetMessageID(), i.GetChatID(), i.Teg);
        if (i.Teg == 2) {
            ListReadedBook.push_back(i);
        }
        else if (i.Teg == 1) {
            ListReadingBook.push_back(i);
        }
        else {
            ListAllAddingBook.push_back(i);
        }
    }
}

void BotLogics::SaveChange()
{
    std::list<InfoToSave> tmp = ListAllAddingBook;
    tmp.splice(tmp.end(), ListReadingBook);
    tmp.splice(tmp.end(), ListReadedBook);
    (SaveToFile(tmp));
}

BotLogics::BotLogics()
{
}

std::int32_t BotLogics::FindIDMessageToDel(std::int32_t findID)
{
    for (InfoToDelete i : ListReplyMessage) {
        printf("Find id block id message- %d find id -%d\n",
i.ReplyMessage, findID);
        if (findID == i.ReplyMessage) {
            return i.OriginalMessage;
        }
    }
    return -1;
}

char BotLogics::CheckWhatIsEdit(std::string messagetext)
{
    if (messagetext == "/edit all") {
        return 0;
    } else if (messagetext == "/edit reading") {
        return 1;
    } else if (messagetext == "/edit readed") {
        return 2;
    } else {
        return -1;
    }
}

bool BotLogics::ChangeList( char whatid, std::int32_t originid)

```

```

        {
            for (std::list<InfoToSave>::iterator it = ListReadingBook.begin(); it
!= ListReadingBook.end(); it++) {
                if (it->GetMessageID() == originid) {
                    it->Teg = whatid;
                    if(whatid == 0) {ListAllAddingBook.push_back(*it);
ListReadingBook.erase(it);}
                    else if(whatid == 2) {ListReadedBook.push_back(*it);
ListReadingBook.erase(it);}
                    return true;
                }
            }

            for (std::list<InfoToSave>::iterator it = ListReadedBook.begin(); it
!= ListReadedBook.end(); it++) {
                if (it->GetMessageID() == originid) {
                    it->Teg = whatid;
                    if(whatid == 0) {ListAllAddingBook.push_back(*it);
ListReadedBook.erase(it);}
                    else if(whatid == 1) {ListReadingBook.push_back(*it);
ListReadedBook.erase(it);}
                    return true;
                }
            }

            for (std::list<InfoToSave>::iterator it = ListAllAddingBook.begin();
it != ListAllAddingBook.end(); it++) {
                if (it->GetMessageID() == originid) {
                    it->Teg = whatid;
                    if(whatid == 1) {ListReadingBook.push_back(*it);
ListAllAddingBook.erase(it);}
                    else if(whatid == 2) {ListReadedBook.push_back(*it);
ListAllAddingBook.erase(it);}
                    return true;
                }
            }

            return false;
        }
    }

```

Даний код є частиною програми, яка реалізовує бота-менеджера для управління списком книг через платформу Telegram. Вище наведений код складається з набору функцій, які забезпечують основні можливості цього бота.

1. `AddBook`: Ця функція викликається для додавання нової книги до списку. Вона створює об'єкт `InfoToSave`, який містить ідентифікатор повідомлення та ідентифікатор чату, пов'язаний з доданою книгою. Після цього об'єкт додається до списку `ListAllAddingBook`, а також викликається функція `SaveToFile`, що, ймовірно, зберігає зміни у файлі.

2. `ViewAllBook`: Ця функція виводить список усіх доданих книг у вказаному чаті. Якщо список `ListAllAddingBook` порожній, то бот повідомляє користувача, що немає доступних книг для перегляду.
3. `ViewReading`: Схожо до попередньої функції, ця функція виводить список книг, які користувач в даний момент читає.
4. `ViewReaded`: Функція виводить список книг, які користувач вже прочитав.
5. `DeleteBook`: Ця функція дозволяє видалити книгу зі списку на основі відповідного повідомлення. Вона шукає повідомлення, на яке надійшла відповідь, у списку `ListReplyMessage`, і якщо знаходить його, то видаляє відповідну книгу зі списку `ListAllAddingBook`.
6. `EditBook`: Ця функція дозволяє редагувати статус книги (наприклад, переміщувати між різними списками книг) на основі команд користувача. Передбачено можливість змінити статус книги на "читаю", "прочитано" або "загальний список".
7. `LoadData`: Функція завантажує збережені дані про книги з файлу. Завантажені дані поділяються на різні списки відповідно до їх статусу.
8. `SaveChange`: Ця функція зберігає зміни у списку книг, переміщуючи їх до відповідних списків `ListReadingBook` або `ListReadedBook` відповідно до їх статусу.

Весь цей код реалізує функціональні можливості бота-менеджера для управління списком книг через платформу Telegram. Зокрема, він дозволяє користувачам додавати, переглядати, редагувати та видаляти книги безпосередньо у чаті з ботом. Інформація про книги зберігається в різних списках відповідно до їхнього статусу (наприклад, "читаю", "прочитано" або "загальний список").

Взаємодія користувача з ботом здійснюється через повідомлення в Telegram. Користувач може відправляти команди боту, такі як `/add`, `/view`, `/edit`, `/delete`, які активують відповідні функції бота. Наприклад, `/add` додає нову книгу до загального списку, `/view` дозволяє переглядати різні списки книг в залежності від їхнього статусу, `/edit` дозволяє редагувати статус книги, а `/delete` дозволяє видаляти книгу.

Код включає ряд функцій, які дозволяють виконувати ці дії, включаючи збереження та завантаження даних про книги, виведення повідомлень про стан книг у чаті, а також реагування на команди користувача. Це дає можливість користувачам більш зручно та організовано керувати своєю книжковою колекцією через платформу Telegram..

2.3 Реалізація додатку

Реалізація додатку є фундаментальним етапом у розробці функціональності та досягненні високої якості системи "Телеграм бот-менеджер книжок". Для реалізації цієї функціональності були використані конкретні технології, мови програмування та інструменти.

Структура додатку включає розділ "src", де знаходяться файли, необхідні для коректної роботи програми. У цьому розділі можуть бути розміщені файлові компоненти, необхідні для реалізації різних аспектів функціональності бота-менеджера. Детальна структура та склад цих файлів може включати модулі для обробки повідомлень, зберігання даних, реалізацію алгоритмів та багато іншого, що допомагає забезпечити правильну роботу додатку.

Крім цього, у реалізації використовуються конкретні мови програмування, такі як, наприклад, C++ для написання коду, який обробляє повідомлення та взаємодіє з платформою Telegram. Ця мова обрана з огляду на її ефективність, продуктивність та можливості в роботі з ресурсами.

Усі ці елементи разом допомагають створити функціональний, структурований та ефективний додаток "Телеграм бот-менеджер книжок", який може взаємодіяти з користувачами через платформу Telegram та забезпечує зручну систему управління книгами.

Name	Last commit message
..	
BotLogics.cpp	pin message
BotLogics.h	pin message
InfoToDelete.h	correction code
InfoToSave.h	Edit command
WriteRead.cpp	add save to file
WriteRead.h	add save to file
main.cpp	pin message

Рис 2.1 папка src

Перший файл - це заголовковий файл 'BotLogics.h'. У цьому файлі оголошений клас 'BotLogics', який містить публічні методи та приватні поля для реалізації логіки взаємодії з ботом та керування списками книг. Цей клас відповідає за обробку повідомлень від користувачів та виконання операцій з книгами.

```

#ifndef BOTLOGICS
#define BOTLOGICS

#include <stdint>
#include <tgbot/Bot.h>
#include <tgbot/tgbot.h>
#include <list>
#include <tgbot/types/Message.h>
#include "InfoToDelete.h"
#include "InfoToSave.h"

class BotLogics
{
    std::list<InfoToSave> ListAllAddingBook;
    std::list<InfoToSave> ListReadingBook;
    std::list<InfoToSave> ListReadedBook;

    std::int32_t FindIDMessageToDel(std::int32_t findID);
    void SaveChange();
    char CheckWhatIsEdit(std::string messagetext);

```

```

bool ChangeList(char whatdit, std::int32_t originid);

public:
std::list<InfoToDelete> ListReplyMessage;
BotLogics();
void AddBook(TgBot::Message::Ptr message);
void ViewAllBook(TgBot::Bot &bot, std::int64_t chatID);
void ViewWantReadBook(TgBot::Bot &bot, std::int64_t chatID);
void ViewReading(TgBot::Bot &bot, std::int64_t chatID);
void ViewReaded(TgBot::Bot &bot, std::int64_t chatID);
void DeleteBook(TgBot::Bot &bot, TgBot::Message::Ptr &message);
void EditBook(TgBot::Bot &bot, TgBot::Message::Ptr &message);
void LoadData();
};
endif

```

Основні аспекти цього заголовочного файлу:

1. Макроси захисту від подвійного включення ('ifndef' та 'define'): Ці макроси забезпечують, що вміст цього файлу буде оброблено лише один раз під час компіляції, що допомагає уникнути подвійного включення.
2. Включення необхідних заголовочних файлів: Цей файл включає необхідні заголовочні файли, такі як 'cstdint' для цілих чисел, 'tgbot/Bot.h' для роботи з ботом, 'tgbot/types/Message.h' для роботи з повідомленнями, і заголовочні файли 'InfoToDelete.h' та 'InfoToSave.h' для роботи з інформацією про книги.
3. Оголошення класу 'BotLogics': Це оголошення класу, яке містить приватні та публічні члени.
4. Приватні поля: 'ListAllAddingBook', 'ListReadingBook', 'ListReadedBook': Списки книг з різними статусами (додані, читаються, прочитані).
5. Приватні методи: 'FindIDMessageToDel', 'SaveChange', 'CheckWhatIsEdit', 'ChangeList': Приватні методи для різних операцій над списками книг.
6. Публічні поля: 'ListReplyMessage': Список повідомлень для відповідей.
7. Публічні методи:
 - а. 'BotLogics()': Конструктор класу.

b. `AddBook`, `ViewAllBook`, `ViewWantReadBook`,
`ViewReading`, `ViewReaded`, `DeleteBook`, `EditBook`:

Публічні методи для взаємодії з ботом та управління списками книг.

с. `LoadData()`: Публічний метод для завантаження даних.

Цей заголовковий файл BotLogics.h визначає інтерфейс та основні засоби для управління книгами через бота. Він оголошує структуру класу BotLogics, який буде містити методи для взаємодії з ботом та керування списками книг. Деталі реалізації цих методів будуть визначені в відповідному джерелі, яке містить визначення цього класу.

Наступним файлом є botlogics.cpp

```
include "BotLogics.h"
include "InfoToSave.h"
include "WriteRead.h"
include <cstdlib>
include <stdio>
include <list>
include <string>
include <tgbot/Bot.h>
include <tgbot/types/Message.h>

// Конструктор класу BotLogics
BotLogics::BotLogics() {
    // Ініціалізація, якщо потрібно
}

// Додавання книги до списку
void BotLogics::AddBook(TgBot::Message::Ptr message) {
    InfoToSave bookInfo(message->messageId, message->chat->id);
    ListAllAddingBook.push_back(bookInfo);
    SaveToFile(bookInfo); // Збереження інформації у файл
}

// Виведення всіх доданих книг користувача
void BotLogics::ViewAllBook(TgBot::Bot &bot, std::int64_t chatID) {
    // ...
}

// Виведення книг у статусі "В читанні"
void BotLogics::ViewReading(TgBot::Bot &bot, std::int64_t chatID) {
    // ...
}

// Виведення прочитаних книг користувача
void BotLogics::ViewReaded(TgBot::Bot &bot, std::int64_t chatID) {
    // ...
}

// Видалення книги за повідомленням
```

```

void BotLogics::DeleteBook(TgBot::Bot &bot, TgBot::Message::Ptr &message)
{
    // ...
    SaveChange(); // Збереження змін у файл
}

// Редагування статусу книги
void BotLogics::EditBook(TgBot::Bot &bot, TgBot::Message::Ptr &message)
{
    // ...
    SaveChange(); // Збереження змін у файл
}

// Завантаження даних із файлу
void BotLogics::LoadData() {
    LoadFromFile load;
    auto tmp = load.GetList();
    // Розподіл даних за статусами
    for (InfoToSave i : tmp) {
        // ...
    }
}

// Збереження змін у файл
void BotLogics::SaveChange() {
    std::list<InfoToSave> tmp = ListAllAddingBook;
    tmp.splice(tmp.end(), ListReadingBook);
    tmp.splice(tmp.end(), ListReadedBook);
    SaveToFile(tmp); // Збереження даних у файл
}

// Пошук ID повідомлення для видалення
std::int32_t BotLogics::FindIDMessageToDel(std::int32_t findID) {
    // ...
}

// Визначення типу редагування
char BotLogics::CheckWhatIsEdit(std::string messagetext) {
    // ...
}

// Зміна списків книг на основі редагування
bool BotLogics::ChangeList(char whatid, std::int32_t originid) {
    // ...
}
...

```

1. Конструктор `BotLogics`: Це ініціалізація класу, яка може включати налаштування, якщо потрібно.
2. Додавання книги (`AddBook`): Ця функція додає інформацію про книгу до списку `ListAllAddingBook` і зберігає цю інформацію у файл.

3. Виведення списку книг (`ViewAllBook`, `ViewReading`, `ViewReaded`): Ці функції виводять інформацію про книги з різними статусами.
4. Видалення книги (`DeleteBook`): Ця функція дозволяє видалити повідомлення про книгу та зберегти зміни у файл.
1. Редагування статусу книги (`EditBook`): Ця функція дозволяє змінити статус книги та виконати піднесення повідомлення на підтримку.
5. Завантаження та збереження даних (`LoadData`, `SaveChange`): Ці функції відповідають за роботу з файлами для збереження та завантаження даних.
6. Додаткові функції (`FindIDMessageToDel`, `CheckWhatIsEdit`, `ChangeList`): Ці функції виконують додаткові завдання, такі як пошук ID повідомлення для видалення, визначення типу редагування та зміну списків книг на основі редагування.

Заголовковий файл `WriteRead.h` насправді визначає два класи: `SaveTOFile` та `LoadFromFile`, які відповідають за функцію зберігання та завантаження даних у файлі. Ці класи відіграють ключову роль у роботі програми, дозволяючи зберігати інформацію про книги та їх статуси, а також відновлювати ці дані при подальших запусках програми.

```

#ifndef WRITEREAD
#define WRITEREAD

#include "InfoToSave.h"
#include <fstream>
#include <list>

const std::string src = "data/data.dat";

class SaveTOFile
{
    std::ofstream Save;
    std::list<InfoToSave> ListToSave;
public:
    SaveTOFile(std::list<InfoToSave> save);
    SaveTOFile(InfoToSave save);
    ~SaveTOFile();
};

```

```

class LoadFromFile
{
    std::ifstream Load;
    std::list<InfoToSave> LoadList;
public:
    LoadFromFile();
    ~LoadFromFile();
    std::list<InfoToSave> GetList();
};

endif

```

Давайте розглянемо основні аспекти цього файлу:

1. Макроси захисту від подвійного включення (`ifndef` та `define`): Тут також використовуються макроси для захисту від подвійного включення файлу.
2. Включення необхідних заголовочних файлів: Цей файл включає необхідні заголовочні файли, такі як `"InfoToSave.h"` для роботи з інформацією про книги, та `"fstream"` для роботи з файлами.
3. Оголошення константи `src`: Ця константа містить шлях до файлу, де будуть зберігатись дані.
4. Оголошення класу `SaveToFile`:
 - `std::ofstream Save`: Об'єкт для запису даних у файл.
 - `std::list<InfoToSave> ListToSave`: Список об'єктів `InfoToSave`, які будуть зберігатись у файлі.
 - Конструктори `SaveToFile(std::list<InfoToSave> save)` та `SaveToFile(InfoToSave save)`: Конструктори, які відповідають за збереження даних у файл.
 - Деструктор `~SaveToFile()`: Він закриває файл при завершенні роботи з об'єктом.
5. Оголошення класу `LoadFromFile`:
 - `std::ifstream Load`: Об'єкт для читання даних з файлу.
 - `std::list<InfoToSave> LoadList`: Список об'єктів `InfoToSave`, які будуть завантажені з файлу.
 - Конструктор `LoadFromFile()`: Він відкриває файл для читання та завантажує дані з нього у список.

- Деструктор `~LoadFromFile()`: Він закриває файл при завершенні роботи з об'єктом.
- Метод `GetList()`: Повертає завантажений список об'єктів `InfoToSave`.

Цей конкретний заголовковий файл, який має назву "WriteRead.h", відіграє ключову роль у визначенні способів збереження та відновлення даних про книги. Він створює основний фундамент для забезпечення постійності даних, навіть у випадку перезавантаження або відновлення роботи програми. Завдяки використанню класів `SaveToFile` та `LoadFromFile`, програма може надійно зберігати важливі дані і пізніше відновлювати їх для забезпечення неперервного функціонування функціональності "Телеграм бот-менеджера книжок". Це важливий крок у забезпеченні відмінної користувацької взаємодії та збереження важливої інформації, що стосується книжкового контенту.

`InfoToDelete.h` є наступним заголовковим файлом, який містить визначення класу `InfoToDelete`. Цей клас відіграє важливу роль у взаємодії з програмою, допомагаючи відстежувати та ідентифікувати інформацію, яку користувач бажає видалити. Це важливий аспект у впорядкуванні та оптимізації процесу управління книжковими даними через "Телеграм бот-менеджер книжок". Використання класу `InfoToDelete` допомагає забезпечити точність та ефективність у видаленні даних, а також взаємодії з користувачем у Telegram.

```

ifndef INFOTODELETE
define INFOTODELETE

include <stdint>

class InfoToDelete
{
public:
    std::int32_t OriginalMessage;
    std::int32_t ReplyMessage;
    InfoToDelete(std::int32_t rplmsg, std::int32_t orgmsg) {
        OriginalMessage = orgmsg;
        ReplyMessage = rplmsg;
    }
};

endif

```

Давайте розглянемо його структуру:

1. Макроси захисту від подвійного включення (``ifndef`` та ``define``): Ці макроси використовуються для захисту від подвійного включення файлу.
2. Оголошення класу ``InfoToDelete``:
 - a. ``std::int32_t OriginalMessage``: Змінна для збереження ідентифікатора початкового повідомлення.
 - b. ``std::int32_t ReplyMessage``: Змінна для збереження ідентифікатора повідомлення-відповіді.
 - c. Конструктор ``InfoToDelete(std::int32_t rplmsg, std::int32_t orgmsg)``: Конструктор класу, який ініціалізує змінні значеннями переданих параметрів.

``InfoToDelete.h`` містить визначення класу ``InfoToDelete``, який містить інформацію про повідомлення, яке було видалено або буде видалено. Цей клас також включає в себе ідентифікатор повідомлення, на яке дане повідомлення є відповіддю. Завдяки цьому підходу можливо ефективно відстежувати та керувати взаємозв'язками між повідомленнями в контексті бота, що сприяє більш структурованій та організованій взаємодії з користувачем.

Цей заголовочний файл описує клас ``InfoToSave``, який містить дані про книги, що плануються до збереження або вже збережені у системі. Клас ``InfoToSave`` включає інформацію, таку як ідентифікатор повідомлення, ідентифікатор чату та тег, який вказує статус книги (наприклад, "додається", "читається", "прочитано"). Використання цього класу дозволяє ефективно управляти та відстежувати стан книг в системі, що сприяє зручній організації та контролю над колекцією книг користувачів.

```
ifndef INFOTOSAVE
define INFOTOSAVE

include <cstdint>
include <string>
include <tgbot/tgbot.h>

class InfoToSave {
```

```

std::int32_t MessageID = 0;
std::int64_t ChatID = 0;
public:
char Teg = 0;
InfoToSave() {}
InfoToSave(std::int32_t messageid ,std::int64_t chatid) {
    MessageID = messageid;
    ChatID = chatid;
}
std::int32_t GetMessageID() {
    return MessageID;
}
std::int64_t GetChatID() {
    return ChatID;
}
};
//Teg 0 - all book list
//1 - Reading book
//2 - Readed book
endif

```

Давайте розглянемо структуру цього файлу:

1. Макроси захисту від подвійного включення ('ifndef' та 'define'): Ці макроси використовуються для захисту від подвійного включення файлу.
2. Оголошення класу 'InfoToSave':
 - a. 'std::int32_t MessageID': Змінна для збереження ідентифікатора повідомлення.
 - b. 'std::int64_t ChatID': Змінна для збереження ідентифікатора чату.
 - c. 'char Teg': Змінна для збереження тегу, який вказує на статус книги.
 - d. Конструктори 'InfoToSave(){}' та 'InfoToSave(std::int32_t messageid ,std::int64_t chatid)': Конструктори класу, які ініціалізують значення змінних.
 - e. Методи 'GetMessageID()' та 'GetChatID()': Методи, які повертають ідентифікатори повідомлення та чату відповідно.
3. Коментарі: Коментарі вказують на значення тегів ('Teg') у класі 'InfoToSave'. Зокрема, тег 0 вказує на список всіх книг, тег 1 - на читані книги, та тег 2 - на прочитані книги.

		В.І. Сарган			ІПЗ.КР.Б – 121 – 23 – ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

Цей заголовочний файл описує клас 'InfoToSave', який містить інформацію про кожну книгу та її поточний статус у контексті взаємодії з ботом. Ця інформація включає такі аспекти, як ідентифікатор повідомлення, ідентифікатор чату та статус книги, такий як "додається", "читається" або "прочитано". Цей клас допомагає системі ефективно відстежувати та керувати книгами, а також забезпечує зручну організацію їхнього статусу.

Наступним файлом є WriteRead.cpp.

```
include "WriteRead.h"
include "InfoToSave.h"
include <ios>
include <list>

SaveToFile::SaveToFile(std::list<InfoToSave> save) {
    ListToSave = save;
    Save.open(src, std::ios::binary);
    for (InfoToSave i : ListToSave) {
        Save.write((char*) &i, sizeof(i));
    }
}

SaveToFile::SaveToFile(InfoToSave save) {
    Save.open(src, std::ios::binary | std::ios::app);
    Save.write((char *) &save, sizeof(InfoToSave));
}

SaveToFile::~SaveToFile() {
    Save.close();
}

LoadFromFile::LoadFromFile() {
    Load.open(src, std::ios::binary);
    InfoToSave tmp;
    while (Load.read((char *) &tmp, sizeof(InfoToSave))) {
        LoadList.push_back(tmp);
    }
}

std::list<InfoToSave> LoadFromFile::GetList() {
    return LoadList;
}

LoadFromFile::~LoadFromFile() {
    Load.close();
}
```

В даному відрізку коду реалізовані два важливі класи: SaveToFile і LoadFromFile, які відповідають за збереження та завантаження даних з файлу відповідно. Давайте ближче розглянемо структуру та функції цих класів:

Клас SaveToFile:

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>47</i>

Цей клас відповідає за збереження інформації у файл. Він має два конструктори - один для збереження списку книг, інший для збереження окремої книги.

1. `SaveTOFile(std::list<InfoToSave> save):` У цьому конструкторі передається список `save` з інформацією про книги, який потрібно зберегти. Він відкриває файл для запису в бінарному режимі (`std::ios::binary`), потім по черзі записує кожен об'єкт `InfoToSave` зі списку у файл за допомогою методу `write`.
2. `SaveTOFile(InfoToSave save):` Цей конструктор використовується для додавання окремої книги до вже існуючого файлу. Він також відкриває файл для запису у бінарному режимі та записує переданий об'єкт `InfoToSave` у файл.
3. Деструктор `~SaveTOFile():` При завершенні роботи з об'єктом класу `SaveTOFile`, деструктор закриває файл.

Клас LoadFromFile:

Цей клас відповідає за завантаження даних з файлу. Він має конструктор, який завантажує дані під час створення об'єкту.

1. `LoadFromFile():` При створенні об'єкта класу `LoadFromFile`, конструктор відкриває файл у бінарному режимі для читання (`std::ios::binary`). Далі, у циклі, він по черзі читає дані з файлу за допомогою методу `read` та додає їх до списку `LoadList`.
2. `GetList():` Цей метод повертає завантажений список книг.
3. Деструктор `~LoadFromFile():` При завершенні роботи з об'єктом класу `LoadFromFile`, деструктор закриває файл.

Ці два класи, `'SaveTOFile'` і `'LoadFromFile'`, відіграють ключову роль у забезпеченні роботи з даними про книги в програмі. Клас `'SaveTOFile'` відповідає за процес збереження цих даних у файл, забезпечуючи їх збереження в структурованому та читабельному форматі. Це важливо для подальшої можливості їх ефективної десеріалізації та відновлення.

Завдяки класу `LoadFromFile`, реалізовано можливість відновлювати збережені дані. Цей клас здійснює зчитування інформації з файлу та її десеріалізацію, перетворюючи дані з текстового формату у внутрішні об'єкти програми. Це дозволяє програмі працювати зі збереженими даними, як зі структурованим набором об'єктів, що спрощує подальшу обробку та використання.

Ці класи важливі для стабільності та надійності системи, оскільки дозволяють користувачам зберігати свої дані між різними сеансами роботи з ботом. Такий підхід гарантує, що важливі дані не будуть втрачені, навіть якщо програма буде перезапущена або оновлена.

Головний файл програми виконує ключову роль у керуванні всім процесом функціонування бота. Він встановлює зв'язок між різними модулями, включає необхідні бібліотеки, ініціалізує об'єкти та забезпечує взаємодію між ними. Саме цей файл визначає точку входу, де розпочинається виконання програми, і об'єднує всі додаткові компоненти в єдиний функціональний блок.

Під час виконання програми, головний файл ініціалізує об'єкти і класи, які відповідають за різні аспекти роботи бота, такі як взаємодія з користувачем, збереження та завантаження даних, реалізація логіки взаємодії з ботом через платформу Telegram. Таким чином, головний файл є ключовим "мозком" програми, який визначає порядок виконання дій та забезпечує їх правильну координацію для досягнення поставлених цілей.

```
include <bits/types/struct_tm.h>
include <iostream>
include <list>
include <stdio.h>
include <string>
include <tgbot/Bot.h>
include <tgbot/tgbot.h>
include <tgbot/tools/StringTools.h>
include <tgbot/types/BotCommand.h>
include <tgbot/types/ForceReply.h>
include <tgbot/types/KeyboardButton.h>
include <tgbot/types/Message.h>
include <vector>
```

```

include "BotLogics.h"

const std::string Menucommand[] = {"/start", "/all", "/add", "/del",
"/reading", "/readed", "/edit", "/help"};
const std::string Menudescription[] = {"Lets start", "Show adding
book", "For add book next message must be forwarded ", "For delete reply sende
message",
"Show what book you reading", "Show what book you readed", "for edit
reply message for this text(all ,reading ,readed) ", "get help"};

//Initial command for bots
std::vector<TgBot::BotCommand::Ptr> InitialCommands() {
std::vector<TgBot::BotCommand::Ptr> botcommand;
for(int i = 0; i < Menucommand->length(); i++) {
TgBot::BotCommand::Ptr tmp(new TgBot::BotCommand);
tmp->command = Menucommand[i];
tmp->description = Menudescription[i];
botcommand.push_back(tmp);
}
return botcommand;
}

int main() {
//Initial block
TgBot::Bot bot("5122355550:AAEjkevANONuLCEDSIknt9zzW9TI7nX7iCg");
TgBot::BotCommand tmp;
bot.getApi().setMyCommands(InitialCommands());
BotLogics logics;
bool isadd = false;

logics.LoadData();
//command block
bot.getEvents().onCommand("start", [&bot] (TgBot::Message::Ptr
message) {
bot.getApi().sendMessage(message->chat->id, "Welcome to the book
maneger");
return;
});
bot.getEvents().onCommand("add", [&bot, &isadd,
&logics] (TgBot::Message::Ptr message) {
isadd = true;
return;
});
bot.getEvents().onCommand("reading", [&bot,
&logics] (TgBot::Message::Ptr message) {
logics.ViewReading(bot, message->chat->id);
return;
});
bot.getEvents().onCommand("readed", [&bot,
&logics] (TgBot::Message::Ptr message) {
logics.ViewReaded(bot, message->chat->id);
return;
});
bot.getEvents().onCommand("all", [&bot, &logics] (TgBot::Message::Ptr
message) {
logics.ViewAllBook(bot, message->chat->id);
return;
});
bot.getEvents().onCommand("del", [&bot, &logics] (TgBot::Message::Ptr
message) {
logics.DeleteBook(bot, message);
return ;
}

```

		В.І. Сарган			ІІЗ.КР.Б – 121 – 23 – ІІЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    });
    bot.getEvents().onCommand("edit",                                     [&bot,
&logics] (TgBot::Message::Ptr message) {
        logics.EditBook(bot, message);
        return ;
    });

    //message block
    bot.getEvents().onAnyMessage([&bot,                                     &isadd,
&logics] (TgBot::Message::Ptr message) {
        for (std::string i : Menucommand) {
            if (StringTools::startsWith(message->text, i)) {
                return;
            }
        }
        //add command
        if(isadd == true) {
            logics.AddBook(message);
            bot.getApi().sendMessage(message->chat->id, "Book will be
added to the all list");
            isadd = false;
            return;
        }

        bot.getApi().sendMessage(message->chat->id, "Please use
command");
    });

    //bot loop
    try {
        printf("Bot username: %s\n", bot.getApi().getMe() -
>username.c_str());
        TgBot::TgLongPoll longPoll(bot);
        while (true) {
            printf("Long poll started\n");
            longPoll.start();
        }
    } catch (TgBot::TgException& e) {
        printf("error: %s\n", e.what());
    }
    return 0;
}

```

Давайте розглянемо структуру цього файлу:

1. Підключення необхідних заголовочних файлів: Здійснюється підключення різних заголовочних файлів, які використовуються в програмі.
2. Константи для меню команд ('Menucommand' і 'Menudescription'): Оголошені масиви строк, які містять команди та описи команд для меню бота.
3. Функція 'InitialCommands': Функція, яка генерує початкові команди для бота згідно з заданими константами.
4. Функція 'main':
 - Створення об'єкта 'TgBot::Bot' для бота і ініціалізація його токена.

- Виклик функції ``setMyCommands`` для встановлення початкових команд бота.
- Створення об'єкта ``BotLogics`` для обробки логіки бота.
- Ініціалізація змінної ``isadd`` для відстеження додавання книги.
- Виклики різних обробників подій для команд та повідомлень бота.
- Виклик обробника для будь-якого повідомлення, яке не є командою.
- Оголошення та ініціалізація об'єкта ``TgBot::TgLongPoll`` для початку циклу бота.

5. Цикл обробки подій бота:

- У спробі спробуємо створити ``TgBot::TgLongPoll`` і почати обробку подій бота.
- У випадку виникнення помилки виводимо відповідне повідомлення про помилку.

Даний код відіграє роль основної структури програми, відповідальної за створення та налаштування бота, а також обробку команд та повідомлень від користувачів. Він встановлює основні параметри бота, такі як токен доступу, який дозволяє боту працювати з платформою Telegram.

Цей фрагмент коду також визначає логіку взаємодії бота з користувачами, включаючи обробку вхідних повідомлень, аналіз їх змісту та виконання відповідних дій на основі команд, що надійшли. Крім того, він ініціалізує цикл для обробки подій бота, що дозволяє програмі постійно перевіряти та реагувати на нові повідомлення та події, які надходять через платформу Telegram.

Завдяки цьому фрагменту програма може функціонувати в режимі постійного очікування та обробки команд користувачів, забезпечуючи взаємодію з ботом у реальному часі та відповідь на їх запити.

Висновки до другого розділу

У другому розділі "Моделювання та реалізація" були розглянуті ключові аспекти розробки та втілення телеграм бота-менеджера книг. Описано структуру системи, проаналізовано вибір технологій та розглянуто процес моделювання та реалізації алгоритмів.

Важливою частиною цього розділу був огляд існуючих програмних продуктів у сфері управління книгами та забезпечення зручного читацького досвіду. Цей огляд дав можливість з'ясувати переваги та недоліки різних рішень, враховуючи їхні особливості та можливості для впровадження в розроблюваний бот.

Моделювання проекту полягало у детальному описі структури системи, розділенні її на функціональні блоки та описі їх взаємодії. Це сприяло зрозумінню, як кожен елемент системи взаємодіє з іншими та які завдання вони виконують.

На підставі вибору технологій для реалізації бота було прийняте рішення використовувати мову програмування C++, що сприяло високій швидкості та ефективності виконання завдань. Також були визначені і впроваджені алгоритми для додавання, видалення та редагування інформації про книги.

Опис реалізації додатку включав докладний опис логіки роботи різних функцій бота, таких як додавання книг, виведення списків, редагування тощо. Це дозволило уявити, як саме бот буде взаємодіяти з користувачем та виконувати запитані завдання.

В цілому, цей розділ надав усі необхідні деталі та інформацію для розробки та реалізації бота-менеджера книг, а також дав змогу чітко спланувати та визначити напрямки подальшої роботи для завершення проекту.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>53</i>

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ

3.1. Встановлення та налаштування програми

У цьому розділі представлені докладні інструкції, що допоможуть вам встановити і налаштувати вашого власного Telegram бота для зручного керування вашою колекцією книжок. Важливо ретельно дотримуватися кожного кроку, оскільки це визначально вплине на успішний запуск бота і відповідний рівень використання всіх його функцій. Правильна настройка дозволить вам максимально залучити потенціал бота для зручного та ефективного керування вашою бібліотекою.

Вимоги до середовища

Перш ніж розпочати процес, варто переконатися, що ви відповідаєте таким вимогам:

1. Операційна система: Ваша операційна система повинна бути сумісною з використаною мовою програмування (наприклад, C++).
2. Telegram-акаунт: Вам потрібний активний обліковий запис у Telegram для створення та налаштування бота.
3. API-ключ: Отримайте API-ключ для свого бота, який дозволить йому взаємодіяти з Telegram API.

Отримання API-ключа

1. Відкрийте Telegram та знайдіть бота @BotFather.
2. Напишіть команду `/newbot`, слідуйте інструкціям для створення нового бота.
3. Після успішної створення бота, @BotFather надішле вам повідомлення з API-ключем. Збережіть цей ключ - він необхідний для налаштування бота.

Встановлення залежностей

		В.І. Сарган			ІПЗ.КР.Б – 121 – 23 – ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

Перед тим як розпочати процес встановлення програми, переконайтеся, що у вас вже встановлені необхідні залежності:

1. Залежно від обраної мови програмування (наприклад, C++), переконайтеся, що на вашому комп'ютері встановлено відповідний компілятор та середовище для розробки.
2. Встановіть бібліотеку для роботи з Telegram API (якщо це необхідно) за допомогою менеджера пакетів вашої мови програмування.

Налаштування API-ключа

Після відкриття програмного коду знайдіть конкретний рядок, в якому передбачено використання API-ключа. Цей ключ є важливим для забезпечення зв'язку програми з платформою Telegram. Замість залишеного демонстраційного значення вставте свій власний API-ключ, який ви отримали при реєстрації вашого бота на платформі Telegram. Така зміна дозволить вашому боту належним чином спілкуватися з Telegram і виконувати відповідні функції, пов'язані з управлінням вашою бібліотекою книжок.

```
const std::string token = "YOUR_API_KEY_HERE";  
TgBot::Bot bot(token);  
...
```

Запуск програми

1. Відкрийте термінал або командний рядок.
2. Перейдіть до директорії зі збереженим кодом бота.
3. Введіть команду `make compile` для компіляції та `./bin/telegram_bot` запуску бота (залежно від мови програмування та інструкцій у документації).

Налаштування бота

Перед тим як запускати бота, ретельно переконайтеся, що ви вірно вказали свій API-ключ та інші налаштування, які можуть бути необхідні для правильної роботи програми. Зазвичай такі налаштування знаходяться у відповідних розділах коду вашої програми, які визначають її параметри та

		В.І. Сарган			ІІЗ.КР.Б – 121 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		55

поведінку. Впевніться, що ви коректно заповнили всі обов'язкові поля та налаштування, що відповідають за зв'язок з платформою Telegram та інші аспекти бота. Ця попередня перевірка допоможе уникнути можливих помилок під час запуску та забезпечити безперебійну роботу бота для зручного керування вашою книжковою колекцією.

Перевірка роботи бота

1. Запустіть бота за допомогою вказаних вище кроків.
2. Відкрийте Telegram та знайдіть свого створеного бота.
3. Перевірте реакцію бота на команди та повідомлення, які ви йому надсилаєте.

Після успішного виконання всіх цих кроків, ваш Telegram бот для управління книжками повинен бути готовим до використання. Не забувайте стежити за оновленнями та налаштуваннями відповідно до вашої потреби, щоб забезпечити оптимальну та надійну роботу бота з керування книжковою колекцією.

3.2. Структура інтерфейсу та порядок роботи з системою

Інтерфейс системи управління книжками реалізований з використанням Telegram бота, що надає зручний та інтуїтивно зрозумілий спосіб взаємодії користувачів з системою. Даний розділ надає детальний огляд структури цього інтерфейсу, а також пояснює послідовність дій користувачів при взаємодії з системою через бота.

Огляд інтерфейсу

Взаємодія користувача з ботом відбувається шляхом обміну текстовими повідомленнями в Telegram. Бот надає різноманітні команди та можливості для зручного управління книжками у відповідному форматі. Основні команди включають:

1. `/start`: Почати взаємодію з ботом та отримати вітання.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>56</i>

2. `/add``: Додати нову книжку до колекції. Бот попросить вас переслати повідомлення з книжкою, яке ви хочете додати.
3. `/reading``: Переглянути список книжок, які ви зараз читаете.
4. `/readed``: Переглянути список книжок, які ви вже прочитали.
5. `/all``: Переглянути список всіх доданих книжок.
6. `/del``: Видалити книжку. Потрібно відповісти на повідомлення, яке ви хочете видалити.
7. `/edit``: Редагувати статус книжки. Потрібно відповісти на повідомлення, для якого ви хочете змінити статус (всі, читаю, прочитані).
8. `/help``: Отримати довідку про доступні команди та їхні описи.

Порядок роботи з системою

1. Старт: Користувач починає взаємодію з ботом, відправивши команду `/start``. Бот вітає користувача та надає вказівки щодо доступних команд.
2. Додавання книжки: Якщо користувач хоче додати нову книжку, він відправляє команду `/add``. Бот просить переслати повідомлення з книжкою, яку користувач хоче додати. Після цього книжка додається до списку.
3. Перегляд списків: Користувач може переглянути свій список книжок, які він читає (`/reading``), та список книжок, які він вже прочитав (`/readed``). Також доступний перегляд всіх доданих книжок (`/all``).
4. Видалення книжки: Якщо користувач хоче видалити книжку зі списку, він використовує команду `/del`` та відповідає на повідомлення, яке він хоче видалити. Книжка видаляється зі списку.
5. Редагування статусу: Якщо користувач хоче змінити статус книжки (чи читає, чи прочитав), він використовує команду `/edit`` та відповідає на повідомлення, для якого хоче змінити статус.

6. Довідка: Користувач може отримати довідку про доступні команди та їхні описи за допомогою команди `/help``.
7. Інші повідомлення: Якщо користувач надсилає будь-яке повідомлення, яке не є командою або не входить в інші категорії, бот повідомляє користувача, що треба використовувати доступні команди.

Ця систематична структура та логічний порядок кроків створюють умови для зручного та ефективного управління користувачами їх власною колекцією книжок. Все це можливо завдяки інтегрованому інтерфейсу Telegram бота, який надає можливість взаємодіяти з програмою шляхом текстових повідомлень.

В результаті, користувачі можуть легко додавати нові книги, відзначати ті, які вони читають або прочитали, а також редагувати і видаляти інформацію про кожну книгу. Це створює відмінну можливість для організації своєї літературної колекції та поліпшення досвіду читання завдяки вдосконаленому інтерфейсу взаємодії з програмою.

Висновки до третього розділу

Програмна реалізація є ключовим етапом у процесі моделювання та проектування. Цей етап дозволяє перетворити абстрактні моделі, концепції та ідеї на конкретні програми, системи або алгоритми, які можуть бути виконані на комп'ютері. Це процес, за допомогою якого теоретичні концепції стають реальними і функціональними продуктами, здатними взаємодіяти з користувачами та вирішувати певні завдання.

В процесі програмної реалізації розробники використовують програмування, алгоритми та технології для створення програмного коду, який відображає вимоги та функціональні можливості системи. Цей код потім може бути компільований та виконаний на комп'ютері, що дозволяє реалізувати концепції та ідеї, що лягли в основу проекту.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		58

У результаті програмна реалізація перетворює абстрактні задуми на конкретні продукти, які можуть бути використані користувачами для вирішення певних завдань. Вона є ключовою ланкою у процесі розробки програмного забезпечення та відображає практичну реалізацію теоретичних концепцій.

Висновки до розділу "Встановлення та налаштування системи"

У цьому розділі було ретельно розглянута процедура встановлення та налаштування розробленої системи. Була надана докладна інструкція щодо необхідних компонентів, які включають в себе бібліотеки для роботи з телеграм API, інструменти для компіляції та збирання програми, а також додаткові засоби для забезпечення безпеки.

Важливою частиною процесу встановлення було налаштування бота на платформі Telegram, отримання токена для доступу та налаштування доступу бота до повідомлень користувачів. Також було розглянуто методи забезпечення безпеки, такі як захист токена та обмеження доступу до функцій бота.

Висновки до розділу "Структура інтерфейсу та порядок роботи з системою"

Розділ "Структура інтерфейсу та порядок роботи з системою" детально описав спосіб взаємодії користувачів з розробленою системою. Завдяки чіткому та логічному розподілу функцій та команд, користувачам легко зрозуміти, як користуватись ботом-менеджером.

Опис структури інтерфейсу надав користувачам зручний доступ до основних опцій системи. Можливість додавання, перегляду, редагування та видалення книг відбувається за допомогою зрозумілих команд, що спрощує роботу з ботом. Додатково, змога редагування та переміщення книг між списками читання та прочитаних книг дозволяє користувачам організовувати свою колекцію належним чином.

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>59</i>

В цілому, розроблена структура інтерфейсу та опис порядку роботи з системою створює дружнє та зручне середовище для керування електронними книгами.

		<i>В.І. Сарган</i>			<i>ІІЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>60</i>

ВИСНОВОК

У ході розробки дипломного проекту "Telegram бот-менеджер книг" було проведено велику кількість досліджень, аналізу та програмної реалізації, що дозволило створити цілісний та добре функціонуючий продукт. Дана робота демонструє, як можливо використовувати сучасні технології та інструменти для створення програмного продукту, що відповідає конкретним потребам користувачів.

Важливо відзначити, що дослідження аналогічних програмних рішень та їх порівняння дало змогу визначити недоліки та переваги підходів, що використовуються в існуючих програмах. Це дозволило прийняти обґрунтовані рішення під час проектування власного бота, спираючись на кращі практики та уникнувши поширених помилок.

Спеціально розроблена архітектура бота забезпечує логічне поділ даних та функцій, що робить код зрозумілим та легко змінюваним. Це дозволяє ефективно впроваджувати нові функції та зміни у вже існуючий функціонал без серйозних змін в основній структурі системи.

Процес реалізації вимагав від команди розробників ретельності, точності та роботи відповідно до встановлених дедлайнів. Кожен етап вимагав аналізу, вибору оптимальних рішень та їх втілення у програмний код. Робота з API платформи Telegram дала можливість створити зручний та інтуїтивно зрозумілий інтерфейс взаємодії з користувачами.

У розділах, присвячених налаштуванню та використанню бота, було розглянуті практичні аспекти впровадження продукту. Налаштування бота на платформі Telegram, встановлення необхідних залежностей та правильне використання команд дозволило створити функціональний бот для керування книгами.

Завдяки виваженій архітектурі, детальному аналізу потреб користувачів та високоякісній реалізації вдалося створити програмний продукт, який

		<i>В.І. Сарган</i>			<i>ІПЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>61</i>

відповідає поставленим цілям та вимогам. Телеграм бот-менеджер книг став корисним та зручним інструментом для тих, хто бажає організувати свою електронну бібліотеку та ефективно керувати своїми читацькими звичками.

У кінцевому підсумку, реалізація даного дипломного проекту не тільки надала можливість вивчити та застосувати сучасні технології програмування, але й показала, як можливо вирішити актуальну проблему управління електронними книгами за допомогою телеграм бота. Даний проект має потенціал для подальшого розширення та покращення, а його реалізація вплине на спрощення та зручність в управлінні читацькою активністю та бібліотекою.

		<i>В.І. Сарган</i>			<i>ІІЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>62</i>

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.
2. Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.
3. BotFather Documentation. (2021). Telegram API Documentation. Retrieved from <https://core.telegram.org/bots#botfather>
4. Meyers, S. (2014). Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14. O'Reilly Media.
5. Stroustrup, B. (2013). The C++ Programming Language. Addison-Wesley Professional.
6. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.
7. Koenig, A., & Moo, B. (2013). Accelerated C++: Practical Programming by Example. Addison-Wesley Professional.
8. Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional.
9. Sutter, H., & Alexandrescu, A. (2005). C++ Coding Standards: 101 Rules, Guidelines, and Best Practices. Addison-Wesley Professional.
10. GoF Design Patterns in Modern C++. (2021). GitHub Repository. Retrieved from <https://github.com/fffaraz/awesome-cpp#design-patterns>
11. O'Shannessy, B. (2019). Developing Telegram Bots. Apress.
12. Schildt, H. (2019). C++: The Complete Reference. McGraw-Hill Education.
13. Bjarne Stroustrup's Homepage. (2021). C++ FAQ. Retrieved from http://www.stroustrup.com/bs_faq.html
14. Telegram Bot API Documentation. (2021). Official Telegram Bot API Documentation. Retrieved from <https://core.telegram.org/bots/api>

15. Eckel, B. (2000). Thinking in C++, Volume 1: Introduction to Standard C++ (2nd ed.). Prentice Hall.
16. Josuttis, N. M. (2019). The C++ Standard Library: A Tutorial and Reference (2nd ed.). Addison-Wesley Professional.
17. Kleppmann, M. (2017). Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media.
18. The Pragmatic Programmer's Guide. (2021). The Pragmatic Bookshelf. Retrieved from <https://pragprog.com/>
19. ISO C++ Standard. (2021). International Organization for Standardization (ISO). Retrieved from <https://isocpp.org/std/the-standard>
20. Martin, R. C. (2009). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall.

ДОДАТКИ

		<i>В.І. Сарган</i>			<i>ІІЗ.КР.Б – 121 – 23 – ПЗ</i>	<i>Арк.</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		<i>65</i>

Програма

Файл BotLogics.h

```

#ifndef BOTLOGICS
#define BOTLOGICS

#include <cstdint>
#include <tgbot/Bot.h>
#include <tgbot/tgbot.h>
#include <list>
#include <tgbot/types/Message.h>
#include "InfoToDelete.h"
#include "InfoToSave.h"

class BotLogics
{
    std::list<InfoToSave> ListAllAddingBook;
    std::list<InfoToSave> ListReadingBook;
    std::list<InfoToSave> ListReadedBook;

    std::int32_t FindIDMessageToDel(std::int32_t findID);
    void SaveChange();
    char CheckWhatIsEdit(std::string messagetext);
    bool ChangeList(char whatdit, std::int32_t originid);

public:
    std::list<InfoToDelete> ListReplyMessage;
    BotLogics();
    void AddBook(TgBot::Message::Ptr message);
    void ViewAllBook(TgBot::Bot &bot, std::int64_t chatID);
    void ViewWantReadBook(TgBot::Bot &bot, std::int64_t chatID);
    void ViewReading(TgBot::Bot &bot, std::int64_t chatID);
    void ViewReaded(TgBot::Bot &bot, std::int64_t chatID);
    void DeleteBook(TgBot::Bot &bot, TgBot::Message::Ptr &message);
    void EditBook(TgBot::Bot &bot, TgBot::Message::Ptr &message);
    void LoadData();
};
#endif

```

Файл BotLogics.cpp

```

#include "BotLogics.h"
#include "InfoToSave.h"
#include "WriteRead.h"
#include <cstdint>
#include <cstdio>
#include <list>
#include <string>
#include <tgbot/Bot.h>
#include <tgbot/types/Message.h>

void BotLogics::AddBook(TgBot::Message::Ptr message)
{
    InfoToSave obg(message->messageId, message->chat->id);
    ListAllAddingBook.push_back(obg);
    (SaveToFile(obg));
}

```

		В.І. Сарган			ІІЗ.КР.Б – 121 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		66

```

void BotLogics::ViewAllBook(TgBot::Bot &bot, std::int64_t chatID)
{
    ListReplyMessage.clear();
    if(ListAllAddingBook.empty()) {
        bot.getApi().sendMessage(chatID, "Nothing to read, maybe need add
from all list?");
        return;
    }
    for (InfoToSave i : ListAllAddingBook) {
        if(chatID == i.GetChatID()) {

ListReplyMessage.push_back(InfoToDelete(bot.getApi().sendMessage(chatID, ".",
false, i.GetMessageID()->messageId, i.GetMessageID())));
        }
    }
}

void BotLogics::ViewReading(TgBot::Bot &bot, std::int64_t chatID)
{
    if (ListReadingBook.empty()) {
        bot.getApi().sendMessage(chatID, "Nothing reading book, maybe
need add from all want read list?");
        return;
    }
    for (InfoToSave i : ListReadingBook) {
        if(chatID == i.GetChatID()) {

ListReplyMessage.push_back(InfoToDelete(bot.getApi().sendMessage(chatID, ".",
false, i.GetMessageID()->messageId, i.GetMessageID())));
        }
    }
}

void BotLogics::ViewReaded(TgBot::Bot &bot, std::int64_t chatID)
{
    if (ListReadedBook.empty()) {
        bot.getApi().sendMessage(chatID, "You dont readed books");
        return;
    }
    for (InfoToSave i : ListReadedBook) {
        if(chatID == i.GetChatID()) {
            bot.getApi().sendMessage(chatID, ".", false,
i.GetMessageID());
        }
    }
}

void BotLogics::DeleteBook(TgBot::Bot &bot, TgBot::Message::Ptr &message)
{
    std::int32_t originid = FindIDMessageToDel(message->replyToMessage-
>messageId);
    if (originid == -1) {
        bot.getApi().sendMessage(message->chat->id, "Sorry dont find
message, please check your reply message");
        return;
    }
    for (std::list<InfoToSave>::iterator it = ListAllAddingBook.begin();
it != ListAllAddingBook.end(); it++) {
        if (it->GetMessageID() == originid) {
            ListAllAddingBook.erase(it);
        }
    }
}

```

```

        bot.getApi().sendMessage(message->chat->id, "message been
delete");
        break;
    }
    }
    SaveChange();
}

void BotLogics::EditBook(TgBot::Bot &bot, TgBot::Message::Ptr &message)
{
    char whatid = CheckWhatIsEdit(message->text);
    if (whatid == -1) {
        bot.getApi().sendMessage(message->chat->id, "Sorry don`t
understand your teg. Please check and try again");
        return;
    }

    std::int32_t originid = FindIDMessageToDel(message->replyToMessage-
>messageId);
    if (originid == -1) {
        bot.getApi().sendMessage(message->chat->id, "Sorry dont find
message, please check your reply message");
        return;
    }

    if (ChangeList(whatid, originid)) {
        bot.getApi().sendMessage(message->chat->id, "Ok");
        for (InfoToSave i : ListReadingBook) {
            if (i.GetChatID() == message->chat->id) {
                bot.getApi().pinChatMessage(message->chat->id,
i.GetMessageID());
            }
        }
    }
    else {
        bot.getApi().sendMessage(message->chat->id, "Bed. Please check
your message");
        return;
    }
    SaveChange();
}

void BotLogics::LoadData()
{
    LoadFromFile load;
    auto tmp = load.GetList();
    for (InfoToSave i : tmp) {
        printf("Load Message ID - %d. Message chat ID - %ld. Teg - %d\n",
i.GetMessageID(), i.GetChatID(), i.Teg);
        if (i.Teg == 2) {
            ListReadedBook.push_back(i);
        }
        else if (i.Teg == 1) {
            ListReadingBook.push_back(i);
        }
        else {
            ListAllAddingBook.push_back(i);
        }
    }
}

void BotLogics::SaveChange()
{

```

```

        std::list<InfoToSave> tmp = ListAllAddingBook;
        tmp.splice(tmp.end(), ListReadingBook);
        tmp.splice(tmp.end(), ListReadedBook);
        (SaveToFile(tmp));
    }

    BotLogics::BotLogics()
    {

    }

    std::int32_t BotLogics::FindIDMessageToDel(std::int32_t findID)
    {
        for (InfoToDelete i : ListReplyMessage) {
            printf("Find id block id message- %d find id -%d\n",
i.ReplyMessage, findID);
            if (findID == i.ReplyMessage) {
                return i.OriginalMessage;
            }
        }
        return -1;
    }

    char BotLogics::CheckWhatIsEdit(std::string messagetext)
    {
        if (messagetext == "/edit all") {
            return 0;
        } else if (messagetext == "/edit reading") {
            return 1;
        } else if (messagetext == "/edit readed") {
            return 2;
        } else {
            return -1;
        }
    }

    bool BotLogics::ChangeList( char whatid, std::int32_t originid)
    {
        for (std::list<InfoToSave>::iterator it = ListReadingBook.begin(); it
!= ListReadingBook.end(); it++) {
            if (it->GetMessageID() == originid) {
                it->Teg = whatid;
                if(whatid == 0) {ListAllAddingBook.push_back(*it);
ListReadingBook.erase(it);}
                else if(whatid == 2) {ListReadedBook.push_back(*it);
ListReadingBook.erase(it);}
                return true;
            }
        }

        for (std::list<InfoToSave>::iterator it = ListReadedBook.begin(); it
!= ListReadedBook.end(); it++) {
            if (it->GetMessageID() == originid) {
                it->Teg = whatid;
                if(whatid == 0) {ListAllAddingBook.push_back(*it);
ListReadedBook.erase(it);}
                else if(whatid == 1) {ListReadingBook.push_back(*it);
ListReadedBook.erase(it);}
                return true;
            }
        }
    }

```

```

        for (std::list<InfoToSave>::iterator it = ListAllAddingBook.begin();
it != ListAllAddingBook.end(); it++) {
            if (it->GetMessageID() == originid) {
                it->Teg = whatid;
                if(whatid == 1) {ListReadingBook.push_back(*it);
ListAllAddingBook.erase(it);}
                else if(whatid == 2) {ListReadedBook.push_back(*it);
ListAllAddingBook.erase(it);}
                return true;
            }
        }

        return false;
    }
}

```

Файл InfoToDelete.h

```

#ifndef INFOTODELETE
#define INFOTODELETE

#include <cstdint>

class InfoToDelete
{
public:
    std::int32_t OriginalMessage;
    std::int32_t ReplyMessage;
    InfoToDelete(std::int32_t rplmsg, std::int32_t orgmsg) {
        OriginalMessage = orgmsg;
        ReplyMessage = rplmsg;
    }
};

#endif

```

Файл InfoToSave.h

```

#ifndef INFOTOSAVE
#define INFOTOSAVE

#include <cstdint>
#include <string>
#include <tgbot/tgbot.h>

class InfoToSave {
    std::int32_t MessageID = 0;
    std::int64_t ChatID = 0;
public:
    char Teg = 0;
    InfoToSave(){}
    InfoToSave(std::int32_t messageid ,std::int64_t chatid) {
        MessageID = messageid;
        ChatID = chatid;
    }
    std::int32_t GetMessageID() {
        return MessageID;
    }
    std::int64_t GetChatID() {
        return ChatID;
    }
}

```

		В.І. Сарган			ІПЗ.КР.Б – 121 – 23 – ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
};
//Teg 0 - all book list
//1 - Reading book
//2 - Readed book
#endif

```

Файл WriteRead.cpp

```

#include "WriteRead.h"
#include "InfoToSave.h"
#include <ios>
#include <list>

SaveToFile::SaveToFile(std::list<InfoToSave> save) {
    ListToSave = save;
    Save.open(src, std::ios::binary);
    for (InfoToSave i : ListToSave) {
        Save.write((char*) &i, sizeof(i));
    }
}

SaveToFile::SaveToFile(InfoToSave save) {
    Save.open(src, std::ios::binary | std::ios::app);
    Save.write((char *) &save, sizeof(InfoToSave));
}

SaveToFile::~SaveToFile() {
    Save.close();
}

LoadFromFile::LoadFromFile() {
    Load.open(src, std::ios::binary);
    InfoToSave tmp;
    while (Load.read((char *) &tmp, sizeof(InfoToSave))) {
        LoadList.push_back(tmp);
    }
}

std::list<InfoToSave> LoadFromFile::GetList() {
    return LoadList;
}

LoadFromFile::~LoadFromFile() {
    Load.close();
}

```

Файл WriteRead.h

```

#ifndef WRITEREAD
#define WRITEREAD

#include "InfoToSave.h"
#include <fstream>
#include <list>

const std::string src = "data/data.dat";

class SaveToFile

```

```

{
    std::ofstream Save;
    std::list<InfoToSave> ListToSave;
public:
    SaveToFile(std::list<InfoToSave> save);
    SaveToFile(InfoToSave save);
    ~SaveToFile();
};

class LoadFromFile
{
    std::ifstream Load;
    std::list<InfoToSave> LoadList;
public:
    LoadFromFile();
    ~LoadFromFile();
    std::list<InfoToSave> GetList();
};

#endif

```

Файл main.cpp

```

#include <bits/types/struct_tm.h>
#include <iostream>
#include <list>
#include <stdio.h>
#include <string>
#include <tgbot/Bot.h>
#include <tgbot/tgbot.h>
#include <tgbot/tools/StringTools.h>
#include <tgbot/types/BotCommand.h>
#include <tgbot/types/ForceReply.h>
#include <tgbot/types/KeyboardButton.h>
#include <tgbot/types/Message.h>
#include <vector>

#include "BotLogics.h"

    const std::string Menucommand[] = {"/start", "/all", "/add", "/del",
"/reading", "/readed", "/edit", "/help"};
    const std::string Menudescription[] = {"Lets start", "Show adding
book", "For add book next message must be forwarded ", "For delete reply sende
message",
    "Show what book you reading", "Show what book you readed", "for edit
reply message for this text(all ,reading ,readed) ", "get help"};

//Initial command for bots
std::vector<TgBot::BotCommand::Ptr> InitialCommands() {
    std::vector<TgBot::BotCommand::Ptr> botcommand;
    for(int i = 0; i < Menucommand->length(); i++) {
        TgBot::BotCommand::Ptr tmp(new TgBot::BotCommand);
        tmp->command = Menucommand[i];
        tmp->description = Menudescription[i];
        botcommand.push_back(tmp);
    }
    return botcommand;
}

```

```

int main() {
    //Initial block
    TgBot::Bot bot("5122355550:AAEjkevANONuLCEDSIknt9zzW9TI7nX7iCg");
    TgBot::BotCommand tmp;
    bot.getApi().setMyCommands(InitialCommands());
    BotLogics logics;
    bool isadd = false;

    logics.LoadData();
    //command block
    bot.getEvents().onCommand("start", [bot] (TgBot::Message::Ptr
message) {
        bot.getApi().sendMessage(message->chat->id, "Welcome to the book
maneger");
        return;
    });
    bot.getEvents().onCommand("add", [bot, &isadd,
&logics] (TgBot::Message::Ptr message) {
        isadd = true;
        return;
    });
    bot.getEvents().onCommand("reading", [bot,
&logics] (TgBot::Message::Ptr message) {
        logics.ViewReading(bot, message->chat->id);
        return;
    });
    bot.getEvents().onCommand("readed", [bot,
&logics] (TgBot::Message::Ptr message) {
        logics.ViewReaded(bot, message->chat->id);
        return;
    });
    bot.getEvents().onCommand("all", [bot, &logics] (TgBot::Message::Ptr
message) {
        logics.ViewAllBook(bot, message->chat->id);
        return;
    });
    bot.getEvents().onCommand("del", [bot, &logics] (TgBot::Message::Ptr
message) {
        logics.DeleteBook(bot, message);
        return ;
    });
    bot.getEvents().onCommand("edit", [bot,
&logics] (TgBot::Message::Ptr message) {
        logics.EditBook(bot, message);
        return ;
    });

    //message block
    bot.getEvents().onAnyMessage([bot, &isadd,
&logics] (TgBot::Message::Ptr message) {
        for (std::string i : Menucommand) {
            if (StringTools::startsWith(message->text, i)) {
                return;
            }
        }
        //add command
        if(isadd == true) {
            logics.AddBook(message);
            bot.getApi().sendMessage(message->chat->id, "Book will be
added to the all list");
            isadd = false;
        }
    });
}

```

		В.І. Сарган			ІІЗ.КР.Б – 121 – 23 – ІІЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        return;
    }

    bot.getApi().sendMessage(message->chat->id, "Please use
command");
});

//bot loop
try {
    printf("Bot username: %s\n", bot.getApi().getMe()-
>username.c_str());
    TgBot::TgLongPoll longPoll(bot);
    while (true) {
        printf("Long poll started\n");
        longPoll.start();
    }
} catch (TgBot::TgException& e) {
    printf("error: %s\n", e.what());
}
return 0;
}

```

Файл makefile

```

CXXFLAGS = -Wall -g

LIBS = --std=c++14 -I/usr/local/include -lTgBot -lboost_system -lssl -
lcrypto -lpthread

compile:
    mkdir -p bin
    g++ $(CXXFLAGS) src/main.cpp -o bin/telegram_bot src/BotLogics.cpp
src/WriteRead.cpp $(LIBS)

clean:
    rm -rf src/*.o bin/

debug:
    mkdir -p bin
    g++ -g -lm -w src/main.cpp -o bin/telegram_botdebug
src/BotLogics.cpp src/WriteRead.cpp $(LIBS)

```