МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ

«ПОЛТАВСЬКИЙ ПОЛІТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Циклова комісія дисциплін програмної інженерії

**ЗВІТ**

**з технологічної практики**

«Розробка і супроводження програмного продукту»

Система для пошуку навчальних відео з обраної теми

на тему \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Виконав: здобувач освіти 4 курсу, групи \_\_\_\_\_\_\_  
 спеціальності 121   
 Інженерія програмного забезпечення

Кравченко А. О.

44

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (прізвище та ініціали)

Бабич О.В

Керівник \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 (підпис) (прізвище та ініціали)

Полтава – 2024

**ЗМІСТ**

[ВСТУП 3](#_Toc150854328)

[1. ПОСТАНОВКА ЗАВДАННЯ 4](#_Toc150854329)

[1.1. Основні вимоги до продукту 4](#_Toc150854330)

[1.2. Вимоги до інтерфейсу 5](#_Toc150854331)

[2. ПЛАНУВАННЯ СИСТЕМИ 7](#_Toc150854332)

[2.1. Архітектура системи 7](#_Toc150854333)

[2.2. Тестування 8](#_Toc150854334)

[2.3. Інструкція з використання системи 8](#_Toc150854335)

[ВИСНОВКИ 10](#_Toc150854336)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 11](#_Toc150854337)

[ДОДАТОК А. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ПРОГРАМНИХ ПРОДУКТІВ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ 12](#_Toc150854338)

[ДОДАТОК Б. UML ДІАГРАМА ПРЕЦЕДЕНТІВ 14](#_Toc150854339)

[ДОДАТОК В. ПРОТОТИП ІНТЕРФЕЙСУ 15](#_Toc150854340)

[ДОДАТОК Г. UML ДІАГРАМА КЛАСІВ (АБО ІНШІ) 16](#_Toc150854341)

[ДОДАТОК Д. СТРУКТУРА САЙТУ ( Цей додаток буде лише тоді коли ви розробляєте сайт) 17](#_Toc150854342)

[ДОДАТОК Е. ER ДІАГРАМА ( Цей додаток буде лише тоді коли ваш продукт використовує базу даних) 18](#_Toc150854343)

[ДОДАТОК Д. ВИХІДНІ КОДИ 19](#_Toc150854344)

[ДОДАТОК Ж. UML ДІАГРАМА РОЗГОРТАННЯ 20](#_Toc150854345)

[ДОДАТОК И. РЕЗУЛЬТАТИ ТЕСТУВАННЯ 21](#_Toc150854346)

[ДОДАТОК Е. ЗНІМКИ ЕКРАНУ 22](#_Toc150854347)

# ВСТУП

Місцем моєї технологічної практики була компанія ТОВ "Юнова Консалтінг", яка є офіційним партнером SAP в Україні. Компанія працює на ринку послуг з впровадження програмного забезпечення SAP з 2004 року та має значний досвід у цій сфері. Основними напрямками діяльності компанії є адаптація ПЗ SAP для українських користувачів, створення власних рішень на основі продуктів SAP та реалізація масштабних проєктів у галузі енергетики, зокрема впровадження систем SAP ERP/IS-U. Крім того, компанія спеціалізується на впровадженні галузевих рішень для енергетичних підприємств, зокрема таких як SAP IS-U, і бере участь у реформах енергоринку.

Компанія має багаторічний досвід роботи з замовниками в Україні та за її межами. За останні два роки фахівці компанії реалізували масштабні проєкти в АТ «Сумиобленерго» та АТ «Вінницяобленерго», які стали визначними досягненнями у впровадженні рішень SAP в Україні. ТОВ "Юнова Консалтінг" також розробило інноваційне рішення «Управління автопарком на базі SAP», яке отримало призове місце у конкурсі партнерів SAP у СНД.

У ході практики я ознайомилася з роботою відділу розробки програмного забезпечення, що займається створенням рішень на базі SAP для автоматизації бізнес-процесів. Переді мною було поставлено завдання розробити систему для пошуку навчальних відео з обраної теми, реалізовану у вигляді Telegram-бота. Основна мета проєкту - створити інструмент, який дозволяє автоматизувати пошук відеоматеріалів, надаючи користувачам можливість швидко знаходити корисний контент за ключовими словами.

Для виконання поставленого завдання мені стали в нагоді знання, вміння й навички, здобуті під час вивчення таких дисциплін:

* Об’єктно-орієнтоване програмування — для створення функціонального коду Telegram-бота мовою Python.
* Алгоритми та структури даних — для реалізації ефективного пошуку і обробки даних.
* Бази даних — для інтеграції PostgreSQL та управління збереженням інформації.
* Архітектура програмного забезпечення — для проектування системи, її модульності та масштабованості.

Під час практики я працювала з сучасним програмним забезпеченням і технологіями, такими як:

* Python — основна мова програмування для реалізації Telegram-бота.
* Flask — серверний фреймворк для створення RESTful API.
* YouTube API — для отримання відео за заданими ключовими словами.
* Telegram API — для інтеграції з платформою Telegram.

Окрім технічних навичок, я вдосконалила свої здібності до роботи в команді, управління часом та документування коду. Практика в компанії ТОВ "Юнова Консалтінг" дозволила мені не лише розширити свої знання у сфері розробки програмного забезпечення, але й отримати досвід роботи з інноваційними технологіями в реальних проєктах.

У результаті практики я створила Telegram-бот, який відповідає сучасним вимогам користувачів, надає вичерпну функціональність і може бути інтегрований у різні бізнес-процеси.

# 1. ПОСТАНОВКА ЗАВДАННЯ

Мною було отримано завдання розробити програмний продукт – систему для пошуку навчальних відео з обраної теми. Система має полегшити пошук корисного навчального контенту на основі ключових тем, слів, фільтрації за категоріями, тривалістю та популярністю [Додаток А]. Основною метою є створення інтуїтивно зрозумілої та функціональної програми для користувачів, які прагнуть підвищити рівень знань у певній галузі [Додаток Б (1)]. Для розробки системи я обрала середовище розробки Visual Studio Code та мову програмування Python [2].

Замовник висунув наступні вимоги:

* Зручний пошук: можливість шукати відео за ключовими словами, фільтрами та обраною темою.
* Система категоризації: поділ навчальних відео на теми та підкатегорії.
* Інтерфейс: простий та адаптивний дизайн для користувачів різного рівня досвіду.
* Модуль збереження: можливість додавання відео у "Вибране".
* Гнучкість: підтримка інтеграції нових функцій у майбутньому.

Терміни розробки:

* Планування та аналіз: 1 тиждень
* Проєктування інтерфейсу та бази даних: 2 тижні
* Розробка функціональності: 2 тижні
* Тестування: 4 дні
* Впровадження та документування: 1 тиждень
* Загальна тривалість розробки: 4 тижнів.

Функціональність програми:

* Пошук навчальних відео за ключовими словами та фільтрами.
* Система рекомендацій на основі переглянутих відео.
* Авторизація користувачів та персоналізований інтерфейс.
* Збереження відео у списки "Вибране".
* Аналітика популярності відео.
* Налаштування інтерфейсу для різних пристроїв.

Замовник також вимагає залишити можливість для подальшого супроводу програми:

* Супровід: оновлення бази даних відео та підтримка користувачів.
* Розширюваність: додавання нових функцій (наприклад, інтеграція з YouTube API або іншими сервісами).
* Технічна підтримка: реагування на помилки та оновлення програмного забезпечення.

## 1.1. Основні вимоги до продукту

ФУНКЦІОНАЛЬНІ ВИМОГИ [Додаток Б (2)]:

* Пошук відео за ключовими словами: користувач вводить запит, і бот повертає список релевантних навчальних відео.
* Фільтрація результатів: можливість відсортувати відео за тривалістю, рейтингом, популярністю тощо.
* Рекомендаційна система: бот пропонує відео на основі попередніх запитів та вподобань користувача.
* Збереження у "Вибране": можливість зберігати цікаві відео для подальшого перегляду.
* Інтеграція з YouTube API [3]: автоматичне отримання навчального контенту з YouTube.
* Авторизація через Telegram: персоналізація функціональності для кожного користувача.
* Розсилка повідомлень: регулярні сповіщення про новий релевантний контент.
* Командний інтерфейс: підтримка основних команд для взаємодії з ботом, наприклад, /search, /favorite, /recommend.

НЕФУНКЦІОНАЛЬНІ ВИМОГИ:

* Шифрування даних користувача.
* Захист від несанкціонованого доступу.
* Інтуїтивно зрозумілий інтерфейс команд.
* Швидке реагування на запити (затримка не більше 2 секунд).
* Підтримка одночасної взаємодії з великою кількістю користувачів.
* Гарантія стабільної роботи навіть при збоях у сервері (автоматичний перезапуск).
* Легке додавання нових функцій без необхідності переробляти існуючу архітектуру.
* Підтримка української та англійської мов.

Telegram-бот для пошуку навчальних відео базується на клієнт-серверній архітектурі, яка забезпечує ефективну взаємодію між клієнтською частиною та сервером. Продукт складається з кількох підсистем: клієнтська частина (Telegram-бот), серверна частина, база даних та модуль аналітики. Telegram-бот працює через Telegram API [1], що забезпечує взаємодію з користувачами, а серверна частина обробляє запити, здійснює пошук у YouTube API, та зберігає дані в базу PostgreSQL.

Продукт побудовано з використанням Python у середовищі розробки Visual Studio Code. Для створення серверної частини використовується Flask, а основними бібліотеками є python-telegram-bot (для роботи з Telegram API), google-api-python-client (для інтеграції з YouTube API) та SQLAlchemy (для управління базою даних). Усі дані користувачів та збережені відео обробляються та зберігаються у базі даних PostgreSQL [4].

Програма залежить від кількох компонентів: Python 3.9 або новішої версії, Telegram API версії 5.3 або новішої, PostgreSQL 14 або новішої. Для роботи з Telegram-ботом користувачам потрібно лише Telegram-додаток з доступом до Інтернету.

Функціональність програми включає: пошук навчальних відео за ключовими словами, фільтрацію результатів за критеріями (тривалість, рейтинг), генерацію персональних рекомендацій на основі запитів, збереження відео у "Вибране" та розсилку повідомлень про новий релевантний контент.

ПЕРЕЛІК РЕАЛІЗОВАНИХ ЗА ЧАС ПРАКТИКИ ФУНКЦІОНАЛЬНИХ ВИМОГ [Додаток Г, Ж, Є]:

* Реалізовано пошук відео за ключовими словами через Telegram-бот: користувач може ввести запит, і бот повертає список релевантних відео.
* Інтеграція з YouTube API: отримання даних про навчальні відео, включаючи назву, опис та тривалість.
* Функція "Додавання у Вибране": користувач може зберігати відео у персональний список для подальшого перегляду.
* Фільтрація результатів пошуку: доступна фільтрація за тривалістю та популярністю.
* Авторизація через Telegram: індивідуальний доступ кожного користувача до свого облікового запису в боті.

ПЕРЕЛІК НЕРЕАЛІЗОВАНОЇ ФУНКЦІОНАЛЬНОСТІ:

* Рекомендаційна система. Причина: недостатньо часу для налаштування алгоритму, який аналізує запити та пропонує відповідні відео.
* Статистика популярності відео. Причина: обмежені ресурси для інтеграції аналітичного модуля, який відображає дані про перегляди та вподобання.
* Розсилка повідомлень про нові релевантні відео. Причина: фокус на реалізації основної функціональності, що не дозволило реалізувати систему автоматичних сповіщень.
* База даних. Причина: недостатньо часу та функціональних можливостей для розробки повноцінної бази даних, що зберігає та організовує всі необхідні відео та користувацькі дані.

## 1.2. Вимоги до інтерфейсу

Вимоги користувача до зовнішнього вигляду програми [Додаток В]:

* Простота та зрозумілість інтерфейсу: інтерфейс має бути інтуїтивно зрозумілим і не потребувати додаткового навчання. Усі команди мають бути чіткими та легко сприйматися користувачем.
* Доступність: програма повинна працювати через Telegram і підтримувати всі платформи (мобільні пристрої, ПК, планшети).
* Оперативність: результати пошуку мають швидко відображатися у вигляді повідомлень з посиланнями та описами відео.
* Локалізація: інтерфейс бота повинен бути українською мовою для забезпечення зручності для цільової аудиторії.
* Модульність меню: усі основні функції (пошук, рекомендації, "Вибране") повинні бути доступні через кнопки або меню команд у Telegram.

Інтерфейс Telegram-бота створений за допомогою Telegram Bot API, з використанням бібліотеки python-telegram-bot. Для створення інтерактивних елементів, таких як кнопки, використано InlineKeyboardMarkup. У цьому випадку Telegram-клієнт виконує функцію GUI-фреймворку, оскільки всі елементи відображаються безпосередньо в додатку Telegram.

Якщо розробляти графічний додаток для настільних систем, можна було б використовувати Tkinter, але в рамках Telegram-бота необхідності в додатковому GUI-фреймворку немає.

Прийняті рішення для покращення зручності використання:

* Кнопки для основних функцій: при виконанні пошуку бот надає кнопки "Додати у Вибране", "Переглянути Рекомендації", що дозволяє виконувати дії без необхідності вводити додаткові команди вручну.
* Зручне меню команд: всі доступні функції, включаючи /start, /search, /favorite, /recommend, додані у вбудоване меню Telegram, що дозволяє швидко викликати потрібну дію.
* Відображення результатів у структурованому вигляді: результати пошуку містять заголовок відео, короткий опис, посилання для перегляду та кнопки для додавання у "Вибране".
* Локалізація: інтерфейс повністю адаптований до української мови, що робить його більш зручним для цільової аудиторії.

Опис зовнішнього вигляду програми [Додаток Е]:

* Головне меню: після введення команди /start [1] користувач отримує привітальне повідомлення та список доступних функцій: /search для пошуку відео; /favorite для перегляду збережених відео; /recommend для отримання рекомендацій; /help для ознайомлення з можливостями бота.
* Вікно пошуку відео: користувач вводить текстовий запит, і бот надсилає результати у вигляді списку відео, включаючи кнопки для дій, таких як "Додати у Вибране".
* Вікно "Вибраного": користувач отримує список збережених відео зі швидким доступом до перегляду або видалення.

# 2. ПЛАНУВАННЯ СИСТЕМИ

Життєвий цикл проєкту складався з декількох етапів, що дозволили поступово створити функціональний продукт. Основні етапи:

* Ініціація: визначення мети проєкту, аналіз потреб користувачів і замовника, постановка завдань.
* Планування: розробка чіткого плану роботи, визначення етапів, інструментів та ресурсів.
* Проєктування: створення архітектури системи, побудова діаграм класів, компонентів та розгортання.
* Розробка: реалізація коду, інтеграція з Telegram API [1] та YouTube API [3], налаштування бази даних.
* Тестування: перевірка працездатності бота, функцій, взаємодії з базою даних та зовнішніми API.
* Впровадження: розгортання продукту на сервері, забезпечення доступу для користувачів.
* Супровід: внесення виправлень, оптимізація роботи, планування додаткових функцій.

Для реалізації проєкту обрана ітеративна модель життєвого циклу (Iterative Model). Вона дозволяє створювати продукт поступово, додаючи нові функції в кожній ітерації.

Основні причини вибору:

* Можливість швидко отримати працюючу версію продукту.
* Гнучкість у внесенні змін за результатами тестування.
* Зменшення ризиків, пов’язаних з великими обсягами одночасної розробки.

Розробка велась за методологією Agile (Scrum). Робота поділялася на короткі спринти тривалістю один тиждень. Кожен спринт включав такі етапи:

* Планування завдань.
* Розробка функцій.
* Тестування і вдосконалення.
* Презентація результатів.

Етапи життєвого циклу:

* Ініціація: на цьому етапі визначили основну мету продукту – створення Telegram-бота для пошуку навчальних відео. Описано функціональні та нефункціональні вимоги.
* Планування: розроблено план реалізації, включаючи часові рамки для кожного етапу. Використовувались інструменти для керування проєктом (Trello, Google Sheets).
* Проєктування: створено архітектуру системи, яка включає клієнт (Telegram), сервер (Flask), базу даних (PostgreSQL) та зовнішній API (YouTube API). Розроблено діаграми класів, компонентів та розгортання.
* Розробка: реалізовано основні функції: пошук, фільтрація, збереження відео у "Вибране". Інтегровано Telegram API для обробки запитів користувачів. Підключено YouTube API для отримання навчального контенту. Налаштовано базу даних PostgreSQL для зберігання інформації.
* Тестування: проведено тестування всіх функцій бота, включаючи перевірку коректності пошуку, роботи бази даних, взаємодії з API.
* Впровадження: продукт розгорнуто на сервері, забезпечено доступ до Telegram-бота для тестових користувачів.
* Супровід: забезпечено моніторинг роботи бота, внесено виправлення для оптимізації швидкості та стабільності.

Програмні засоби для планування та керування проєктом:

* GitHub: система контролю версій. Спільна робота над кодом. Використання Pull Requests для перевірки змін.
* Draw.io: створення діаграм компонентів, класів і розгортання.
* Google Sheets: побудова Gantt-діаграми для відстеження прогресу.
* Postman: перевірка роботи API-запитів до Telegram API та YouTube API.

Життєвий цикл проєкту Telegram-бота базувався на ітеративній моделі, що дозволило поступово створювати та вдосконалювати продукт. Agile (Scrum) забезпечив ефективну організацію роботи, а використання сучасних інструментів (Trello, GitHub, Postman) дозволило зберігати високу якість виконання завдань на всіх етапах розробки.

## 2.1. Архітектура системи

Telegram-бот для пошуку навчальних відео побудований на основі клієнт-серверної архітектури. Бот взаємодіє з користувачем через клієнтський додаток Telegram, обробляє запити на серверній частині, отримує відео з YouTube API, зберігає інформацію в базу даних і повертає результати користувачу [Додаток Г, З, К].

Використані патерни проектування:

* Шаблон MVC (Model-View-Controller). Model: обробляє дані користувачів, списки відео, інформацію з бази даних. View: забезпечується Telegram API, який відповідає за візуалізацію. Controller: серверна логіка на Flask, яка приймає запити, обробляє їх та виконує інтеграцію з YouTube API.
* Шаблон "Фасад" (Facade). Сприяє спрощенню доступу до функціональності системи, забезпечуючи єдиний інтерфейс для роботи з Telegram API та YouTube API.
* Шаблон "Фабричний метод" (Factory Method). Використовується для створення об’єктів запитів до API залежно від типу запиту користувача.

Система складається з наступних основних компонентів:

* Telegram-клієнт: інтерфейс, через який користувачі взаємодіють з ботом. Надсилає команди та отримує результати у вигляді повідомлень.
* Серверна частина. Виконана за допомогою Flask: приймає запити від Telegram-клієнта, інтегрується з YouTube API для отримання інформації про відео, обробляє дані користувачів і зберігає їх у базі.
* YouTube API: зовнішній компонент для отримання інформації про відео на основі запитів користувачів.
* База даних (PostgreSQL): зберігає інформацію про користувачів, їхні налаштування, списки "Вибране" та кеш результатів пошуку.

Підсистеми системи:

* Підсистема обробки запитів: приймає запити від Telegram API, перевіряє їх валідність, передає у відповідний модуль для обробки.
* Підсистема інтеграції з YouTube API: виконує пошук відео, фільтрацію та сортування результатів.
* Підсистема управління базою даних: реалізована з використанням SQLAlchemy. Відповідає за збереження та отримання даних.
* Підсистема взаємодії з користувачем: формує відповіді у вигляді повідомлень у Telegram, створює інтерактивні кнопки для полегшення взаємодії.

Telegram-бот для пошуку навчальних відео реалізований мовою Python з використанням бібліотеки python-telegram-bot. Серверна частина побудована з залученням фреймворку Flask [5] для обробки запитів, а інтеграція з YouTube API здійснюється через бібліотеку google-api-python-client. Усі дані зберігаються у базі даних PostgreSQL, яка керується через ORM-бібліотеку SQLAlchemy.

Код розбитий на кілька основних модулів [Додаток Д]:

* main.py: головний файл запуску Telegram-бота, містить точки входу та основну логіку взаємодії.
* handlers.py: модуль обробки команд користувачів (/start, /search, /favorite, /recommend).
* database.py: реалізує підключення до PostgreSQL [4] та маніпуляції з базою даних.
* youtube\_integration.py: використовується для запитів до YouTube API та отримання інформації про відео.
* config.py: зберігає конфігураційні дані, такі як токен бота та API-ключі.

Особливості реалізації:

* Модульність: код розділений на окремі файли та модулі для зручності підтримки та подальшого масштабування.
* Використання шаблонів проектування: шаблон MVC використовується для чіткого розподілу логіки, обробки даних та взаємодії з користувачем. Шаблон "Фасад" для спрощення взаємодії з API.
* Безпека: конфіденційні дані, такі як токен бота та API-ключі, зберігаються в окремому файлі (.env). Дані користувачів шифруються перед зберіганням у базі даних.
* Логування: уся активність записується у лог-файл для подальшого моніторингу та відстеження помилок.

## 2.2. Тестування

Види використаного тестування [Додаток И]:

* Юніт-тестування (Unit Testing). Тестували окремі функції програми, наприклад: коректність запитів до YouTube API, збереження відео у базі даних, генерацію рекомендацій. Для юніт-тестування використовувався фреймворк Pytest, який дозволив автоматизувати тести та отримати детальні звіти.
* Інтеграційне тестування. Перевіряли взаємодію між Telegram API, серверною частиною та YouTube API. Перевіряли цілісність передачі даних між компонентами (наприклад, від користувача до бази даних і назад).
* Тестування навантаження. Моделювали ситуації одночасного використання бота кількома користувачами (до 50 запитів/хвилину). Використовували бібліотеку Locust для симуляції навантаження.
* Тестування сумісності. Перевіряли роботу бота на різних платформах (мобільні пристрої, десктопи, Telegram Web).
* Стрес-тестування. Імітували перевантаження системи шляхом надсилання великої кількості запитів за короткий час. Перевіряли стійкість до збоїв у роботі API (наприклад, у випадку недоступності YouTube API).
* Тестування доступності. Перевіряли зручність використання для користувачів із різними технічними знаннями.

Використані засоби тестування:

* Pytest: автоматизація юніт-тестів для перевірки функцій бота.
* Postman: тестування API-запитів до YouTube API та Telegram API.
* Locust: тестування навантаження системи.
* Selenium: автоматизація тестування Telegram Web для перевірки роботи інтерфейсу.

Процес тестування:

* Підготовка: Розроблено набір юніт-тестів для перевірки основних функцій (запит до API, збереження даних, генерація рекомендацій). Створено сценарії для інтеграційного та навантажувального тестування.
* Проведення тестування: Виконували юніт-тести для кожного модуля окремо. Інтеграційне тестування включало тестування повного циклу: від введення команди користувачем до відображення результатів. Стрес-тестування виконувалося шляхом збільшення кількості запитів до 100/хвилину.
* Знайдені помилки:
* Помилка: некоректна обробка запитів із довгими ключовими словами. - Рішення: введено обмеження на довжину запиту (до 50 символів).
* Помилка: відсутність обробки ситуації, коли YouTube API повертає порожній результат. - Рішення: реалізовано повідомлення користувачу про відсутність результатів.
* Помилка: помилки при одночасному використанні великої кількості запитів до бази даних. - Рішення: оптимізовано запити до бази даних із використанням пулу з’єднань.
* Внесені зміни: Додано обробку крайових випадків (наприклад, порожній запит, відсутність результатів). Оптимізовано роботу з базою даних для уникнення перевантаження. Додано таймаут для запитів до API для уникнення довгих очікувань.

Зауваження від замовника:

* Пропозиція: Додати детальніший опис помилок у разі їх виникнення. - Рішення: Реалізовано текстові повідомлення з описом причин помилки.
* Пропозиція: Додати англійську мову інтерфейсу. - Рішення: Локалізація планується у майбутньому.

## 2.3. Інструкція з використання системи

Для взаємодії з ботом користувач потребує встановленого Telegram.

Сценарії використання [Додаток Х]:

* Запуск бота (прецедент "Запуск програми"): Відкрийте Telegram і знайдіть бота за посиланням. Натисніть кнопку "Start" [Додаток Х (1)].
* Пошук відео (прецедент "Пошук відео"): Введіть команду /search [тема]. Бот поверне список релевантних відео з короткими описами та кнопками для дій (перегляд або додавання у "Вибране") [Додаток Х (2)].
* Додавання у "Вибране" (прецедент "Додавання у 'Вибране'"): Натисніть кнопку "Додати у Вибране" під обраним відео. Бот підтвердить успішне збереження [Додаток Х (3)].
* Перегляд "Вибраного" (прецедент "Перегляд 'Вибраного'"): Введіть команду /favorite. Бот поверне список збережених відео з кнопками для видалення чи перегляду [Додаток Х (4)].
* Отримання рекомендацій (прецедент "Перегляд рекомендацій"): Введіть команду /recommend. Бот згенерує список відео на основі ваших попередніх запитів [Додаток Х (5)].
* Отримання довідки (прецедент "Отримання допомоги"): Введіть команду /help. Бот надасть опис доступних команд і їхнього призначення [Додаток Х (6)].

# ВИСНОВКИ

Під час проходження практики у компанії ТОВ "Юнова Консалтінг" [7], яка є офіційним партнером SAP в Україні, переді мною було поставлено завдання створити систему для пошуку навчальних відео з обраної теми у вигляді Telegram-бота. Основна мета полягала в автоматизації процесу пошуку навчальних матеріалів, що надає користувачам зручний доступ до відеоконтенту за ключовими словами.

У результаті виконано такі роботи:

* Розроблено Telegram-бот з базовим функціональністю: пошук відео за ключовими словами через YouTube API, фільтрація та збереження результатів у базу даних, надання рекомендацій на основі попередніх запитів, інтерактивний інтерфейс для зручності користувачів.
* Проведено тестування системи, що підтвердило її працездатність і стабільність під час навантажень.
* Реалізовано інтеграцію з базою даних PostgreSQL для збереження користувацьких налаштувань.

Однак деякі функції залишилися нереалізованими через обмежені часові рамки, зокрема:

* Розширена система рекомендацій.
* Підтримка мультимовності.

Результати виконаної роботи були високо оцінені керівництвом компанії. Бот використовується для демонстрації можливостей автоматизації навчального процесу та може бути інтегрований у внутрішні навчальні проєкти компанії.

Практика дозволила мені вдосконалити знання, здобуті під час вивчення таких дисциплін:

* Об’єктно-орієнтоване програмування – реалізація основної функціонаьності бота.
* Алгоритми та структури даних – оптимізація пошукових запитів.
* Бази даних – розробка схеми бази даних та керування інформацією в PostgreSQL.
* Системний аналіз – проектування архітектури системи.

Я навчилася працювати з API-сервісами (Telegram API, YouTube API), опанувала використання Python для розробки автоматизованих рішень, а також вдосконалила навички командної роботи, документування коду та керування проєктом за допомогою Trello і GitHub.

Створений Telegram-бот має потенціал для подальшого розвитку. Я планую продовжити роботу над проєктом, удосконалити систему рекомендацій, додати мультимовність і нові функції, такі як аналітика пошукових запитів. Крім того, цей продукт може стати частиною мого дипломного проєкту.

Практика в ТОВ "Юнова Консалтінг" надала мені безцінний досвід у реальній розробці програмного забезпечення, який я планую використовувати у своїй майбутній професійній діяльності.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

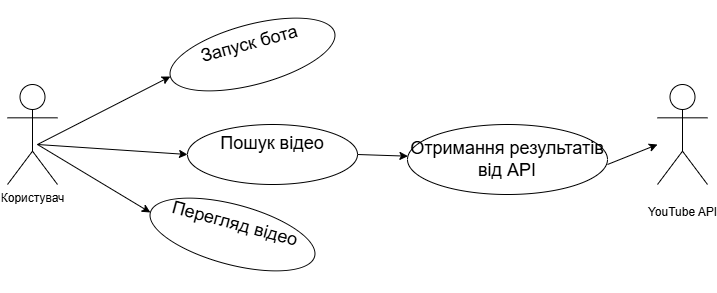
1. Telegram Bot API. Офіційна документація. [Електронний ресурс]. URL: <https://core.telegram.org/bots/api> (дата звернення: 17.12.2024).
2. Python Telegram Bot. Документація бібліотеки python-telegram-bot. [Електронний ресурс]. URL: [https://python-telegram-bot.readthedocs.io](https://python-telegram-bot.readthedocs.io/) (дата звернення: 17.12.2024).
3. Google API Client for Python. Офіційна документація. [Електронний ресурс]. URL: <https://developers.google.com/api-client-library/python> (дата звернення: 17.12.2024).
4. PostgreSQL. Офіційна документація. [Електронний ресурс]. URL: <https://www.postgresql.org/docs/> (дата звернення: 17.12.2024).
5. Flask Framework. Документація фреймворку Flask. [Електронний ресурс]. URL: [https://flask.palletsprojects.com](https://flask.palletsprojects.com/) (дата звернення: 17.12.2024).
6. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні вимоги та правила складання. [Текст]. Київ: ДП «УкрНДНЦ», 2015. – 17 с.
7. Sap Unova. ТОВ "Юнова Консалтінг". [Електронний ресурс]. URL: [https://unova.com.ua](https://unova.com.ua/) (дата звернення: 17.12.2024).

# ДОДАТОК А. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ПРОГРАМНИХ ПРОДУКТІВ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vimeo | Dailymotion | Створений програмний продукт |
| Адаптивність | + | + | + |
| Багатокористувацький режим | + | + | + |
| Інтуїтивно-зрозумілий інтерфейс користувача | + | + | + |
| Наявність української мови | - | - | + |
| Можливість імпорту/експорту даних | - | - | - |
| Крос-платформність (наявність версій для Linux/MacOS) | + | + | + |
| Автентифікація з використанням соціальних облікових записів (Google/Facebook/Twitter тощо) | + | + | + |
| Наявність довідкових матеріалів та документації | + | + | + |
| Можливості з налагодження та розширення функціональності (Rest API/SDK) | + | + | + |
| Вартість ліцензії | Умовно безкоштовно | Умовно безкоштовно | Безкоштовно |

# ДОДАТОК Б. UML ДІАГРАМА ПРЕЦЕДЕНТІВ

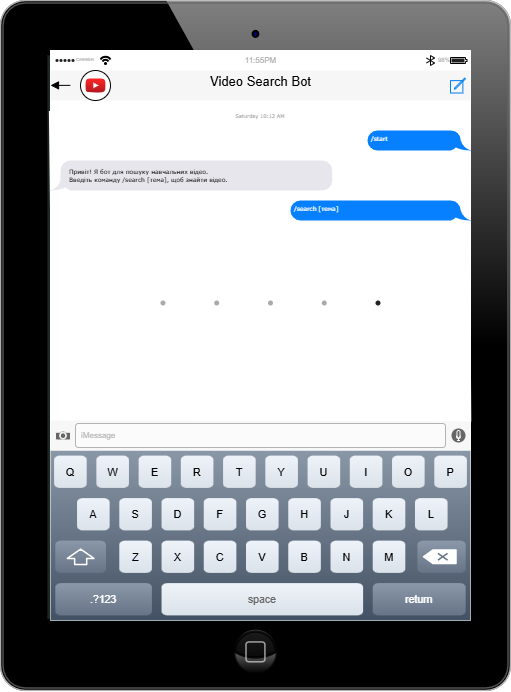
1:



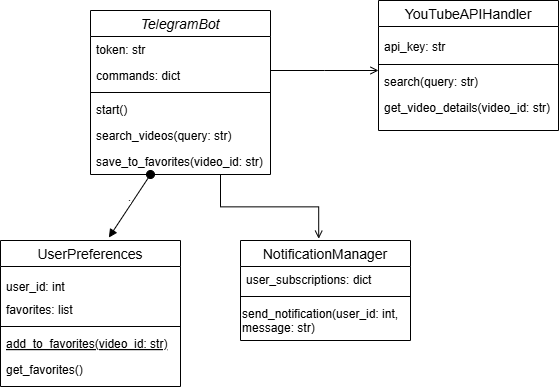
2:



# ДОДАТОК В. ПРОТОТИП ІНТЕРФЕЙСУ



# ДОДАТОК Г. UML ДІАГРАМА КЛАСІВ



# 

# ДОДАТОК Д. ВИХІДНІ КОДИ

**import** sqlite3

**import** telebot

**import** logging

**from** telebot **import** types

**import** requests

# Параметри бота

DB\_NAME = 'bot\_database.db' # Назва файлу бази даних

TOKEN = '7841518347:AAENiHfUyxNWFKS\_qJLH8p0eOaq6kPJ5WQU' # Заміни на свій токен

bot = telebot.TeleBot(TOKEN)

logging.basicConfig(

filename='bot.log',

format='%(asctime)s - %(levelname)s - %(message)s',

level=logging.INFO

)

# Підключення до бази даних

**def** connect\_to\_db():

**return** sqlite3.connect(DB\_NAME)

# Ініціалізація бази даних

**def** initialize\_database():

connection = sqlite3.connect(DB\_NAME)

cursor = connection.cursor()

# Створення таблиці користувачів

cursor.execute('''

CREATE TABLE IF NOT EXISTS Users (

Id INTEGER PRIMARY KEY,

Counter INTEGER DEFAULT 0

)

''')

# Створення таблиці для результатів пошуку

cursor.execute('''

CREATE TABLE IF NOT EXISTS SearchResult (

Id INTEGER PRIMARY KEY AUTOINCREMENT,

UserId INTEGER,

Link TEXT,

FOREIGN KEY(UserId) REFERENCES Users(Id)

)

''')

# Створення таблиці для вибраних відео

cursor.execute('''

CREATE TABLE IF NOT EXISTS Favorites (

Id INTEGER PRIMARY KEY AUTOINCREMENT,

UserId INTEGER,

Link TEXT,

FOREIGN KEY(UserId) REFERENCES Users(Id)

)

''')

connection.commit()

connection.close()

# Додавання користувача до бази даних

**def** add\_user(chat\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute('INSERT OR IGNORE INTO Users (Id) VALUES (?)', (chat\_id,))

connection.commit()

connection.close()

# Збереження результатів пошуку

**def** save\_search\_results(chat\_id, links):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute('DELETE FROM SearchResult WHERE UserId = ?', (chat\_id,))

connection.commit()

**for** link **in** links:

cursor.execute('INSERT INTO SearchResult (UserId, Link) VALUES (?, ?)', (chat\_id, link))

connection.commit()

connection.close()

# Додавання відео в обране

**def** add\_to\_favorites(chat\_id, link):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute('INSERT INTO Favorites (UserId, Link) VALUES (?, ?)', (chat\_id, link))

connection.commit()

connection.close()

# Видалення відео з обраного

**def** remove\_from\_favorites(chat\_id, link):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute('DELETE FROM Favorites WHERE UserId = ? AND Link = ?', (chat\_id, link))

connection.commit()

connection.close()

# Отримання списку обраних відео

**def** get\_favorites(chat\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute('SELECT Link FROM Favorites WHERE UserId = ?', (chat\_id,))

favorites = cursor.fetchall()

connection.close()

**return** [item[0] **for** item **in** favorites]

# Пошук відео на YouTube

**def** youtube\_search(query, sort\_by="relevance"):

API\_KEY = 'AIzaSyBKuvZ7SGUXkvw8vzaRCkHjxVWVnhx32YE' # Заміни на свій API ключ

url = 'https://www.googleapis.com/youtube/v3/search'

params = {

'part': 'snippet',

'q': query,

'type': 'video',

'maxResults': 5,

'order': sort\_by,

'key': API\_KEY

}

response = requests.get(url, params=params)

data = response.json()

links = []

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

video\_id = item['id']['videoId']

links.append(f'https://www.youtube.com/watch?v={video\_id}')

**return** links

# Основне меню для користувача

**def** main\_menu():

markup = types.ReplyKeyboardMarkup(resize\_keyboard=True)

markup.add(types.KeyboardButton('/search'), types.KeyboardButton('/favorite'))

markup.add(types.KeyboardButton('/recommend'), types.KeyboardButton('/help'))

**return** markup

# Обробка команди /start

@bot.message\_handler(commands=['start'])

**def** handle\_start(message):

chat\_id = message.chat.id

add\_user(chat\_id)

bot.send\_message(chat\_id, 'Привіт! Обери одну з команд нижче:', reply\_markup=main\_menu()) # Привітання

# Обробка команди /help

@bot.message\_handler(commands=['help'])

**def** handle\_help(message):

chat\_id = message.chat.id

help\_text = (

"Доступні команди:\n"

"/search - Пошук відео на YouTube \n"

"/favorite - Перегляд обраних відео ❤️\n"

"/recommend - Отримати рекомендації (поки в розробці) \n"

"/help - Отримати допомогу \n"

" Про програму:\n"

"Цей бот допомагає шукати відео на YouTube, зберігати їх у список обраного та переглядати відео на основі ваших запитів. \n"

"Ви можете легко додавати відео до обраного та видаляти їх за допомогою зручних кнопок.\n"

"‍ Автор: Кравченко Анна\n"

" Дата створення: 25 грудня 2024 року\n\n"

"Залишайтеся на зв'язку та користуйтеся всіма можливостями бота! "

)

bot.send\_message(chat\_id, help\_text, reply\_markup=main\_menu()) # Допомога

# Обробка команди /search

@bot.message\_handler(commands=['search'])

**def** handle\_search(message):

chat\_id = message.chat.id

bot.send\_message(chat\_id, 'Введіть запит для пошуку: ', reply\_markup=types.ReplyKeyboardRemove()) # Запит на пошук

@bot.message\_handler(func=**lambda** msg: True)

**def** handle\_query(msg):

query = msg.text

links = youtube\_search(query)

save\_search\_results(chat\_id, links)

**for** link **in** links:

markup = types.InlineKeyboardMarkup()

markup.add(types.InlineKeyboardButton("Додати в обране ❤️", callback\_data=f'add\_favorite\_{link}'))

bot.send\_message(chat\_id, link, reply\_markup=markup)

bot.send\_message(chat\_id, 'Ви можете додати відео в обране, натиснувши кнопку.', reply\_markup=main\_menu()) # Пояснення

# Обробка команди /favorite

@bot.message\_handler(commands=['favorite'])

**def** handle\_favorite(message):

chat\_id = message.chat.id

favorites = get\_favorites(chat\_id)

**if** favorites:

bot.send\_message(chat\_id, 'Ваші обрані відео: ❤️')

**for** link **in** favorites:

markup = types.InlineKeyboardMarkup()

markup.add(types.InlineKeyboardButton("Видалити з обраного ❌", callback\_data=f'remove\_favorite\_{link}'))

bot.send\_message(chat\_id, link, reply\_markup=markup)

**else**:

bot.send\_message(chat\_id, 'Ваш список обраних порожній. ')

bot.send\_message(chat\_id, 'Виберіть одну з команд нижче:', reply\_markup=main\_menu()) # Запит на вибір

# Обробка додавання в обране через inline кнопку

@bot.callback\_query\_handler(func=**lambda** call: call.data.startswith('add\_favorite\_'))

**def** handle\_add\_favorite(callback\_query):

chat\_id = callback\_query.message.chat.id

link = callback\_query.data.replace('add\_favorite\_', '')

add\_to\_favorites(chat\_id, link)

bot.answer\_callback\_query(callback\_query.id, text=f'Відео додано в обране: {link} ')

bot.send\_message(chat\_id, f'Відео додано в обране: {link}')

bot.send\_message(chat\_id, 'Виберіть одну з команд нижче:', reply\_markup=main\_menu())

# Обробка видалення з обраного через inline кнопку

@bot.callback\_query\_handler(func=**lambda** call: call.data.startswith('remove\_favorite\_'))

**def** handle\_remove\_favorite(callback\_query):

chat\_id = callback\_query.message.chat.id

link = callback\_query.data.replace('remove\_favorite\_', '')

remove\_from\_favorites(chat\_id, link)

bot.answer\_callback\_query(callback\_query.id, text=f'Відео видалено з обраного: {link} ')

bot.send\_message(chat\_id, f'Відео видалено з обраного: {link}')

bot.send\_message(chat\_id, 'Виберіть одну з команд нижче:', reply\_markup=main\_menu())

# Обробка команди /recommend

@bot.message\_handler(commands=['recommend'])

**def** handle\_recommend(message):

chat\_id = message.chat.id

bot.send\_message(chat\_id, 'Рекомендації на основі ваших запитів поки в розробці! ')

bot.send\_message(chat\_id, 'Виберіть одну з команд нижче:', reply\_markup=main\_menu()) # Розробка

# Запуск бота

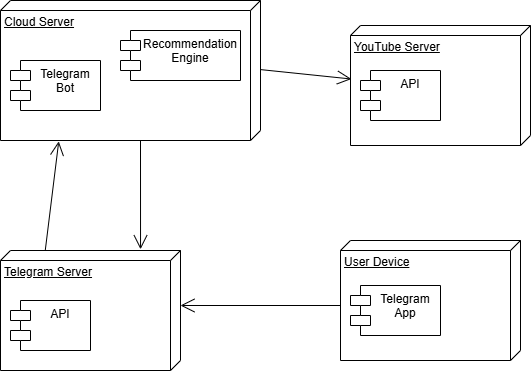
**if** \_\_name\_\_ == '\_\_main\_\_':

initialize\_database()

bot.polling(none\_stop=True)

\*Повні вихідні коди застосунку наведені в публічному репозиторії, доступному за посиланням https://..." і додати знизу або правруч чи ліворуч:) QR-код

# ДОДАТОК Ж. UML ДІАГРАМА РОЗГОРТАННЯ

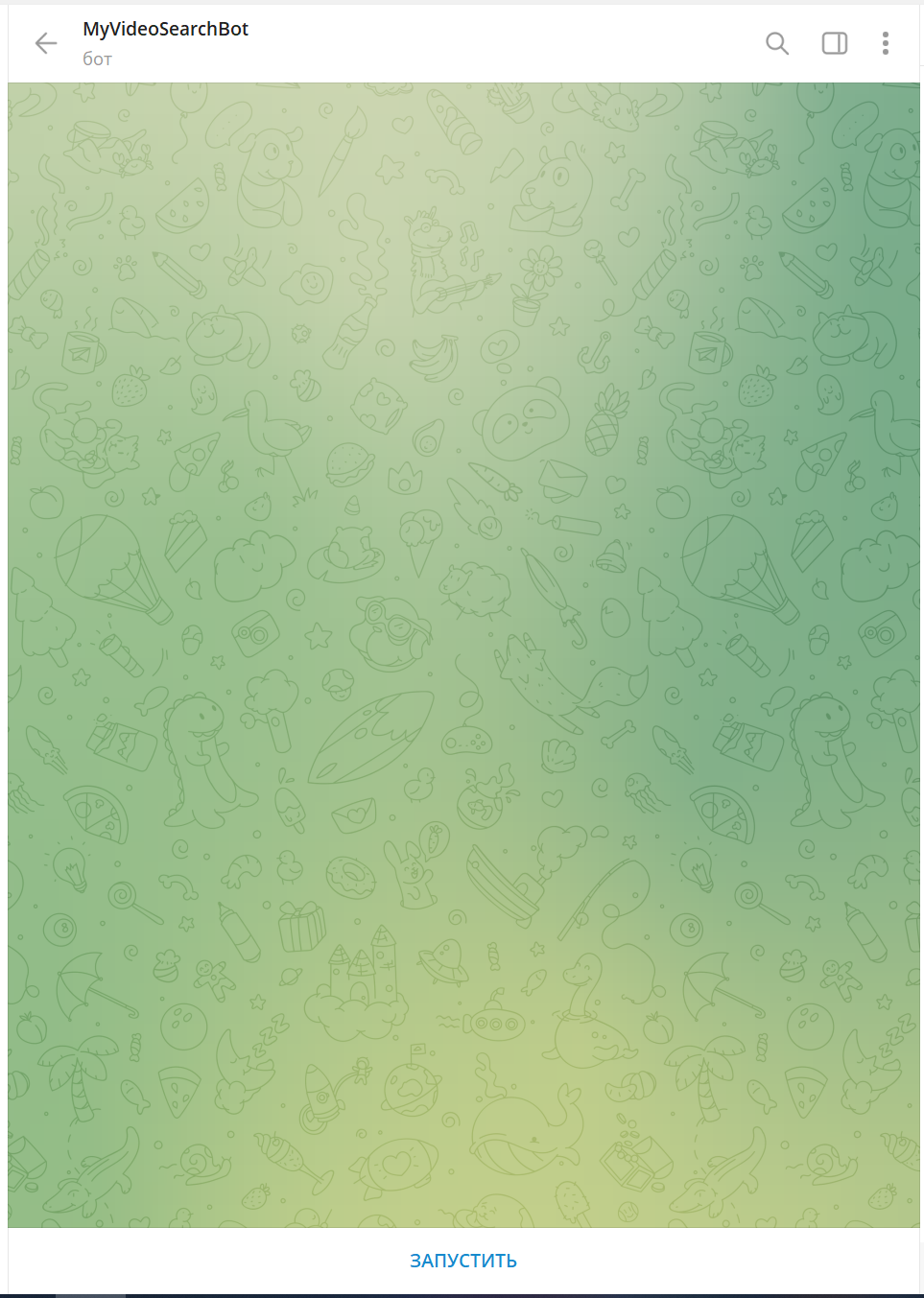


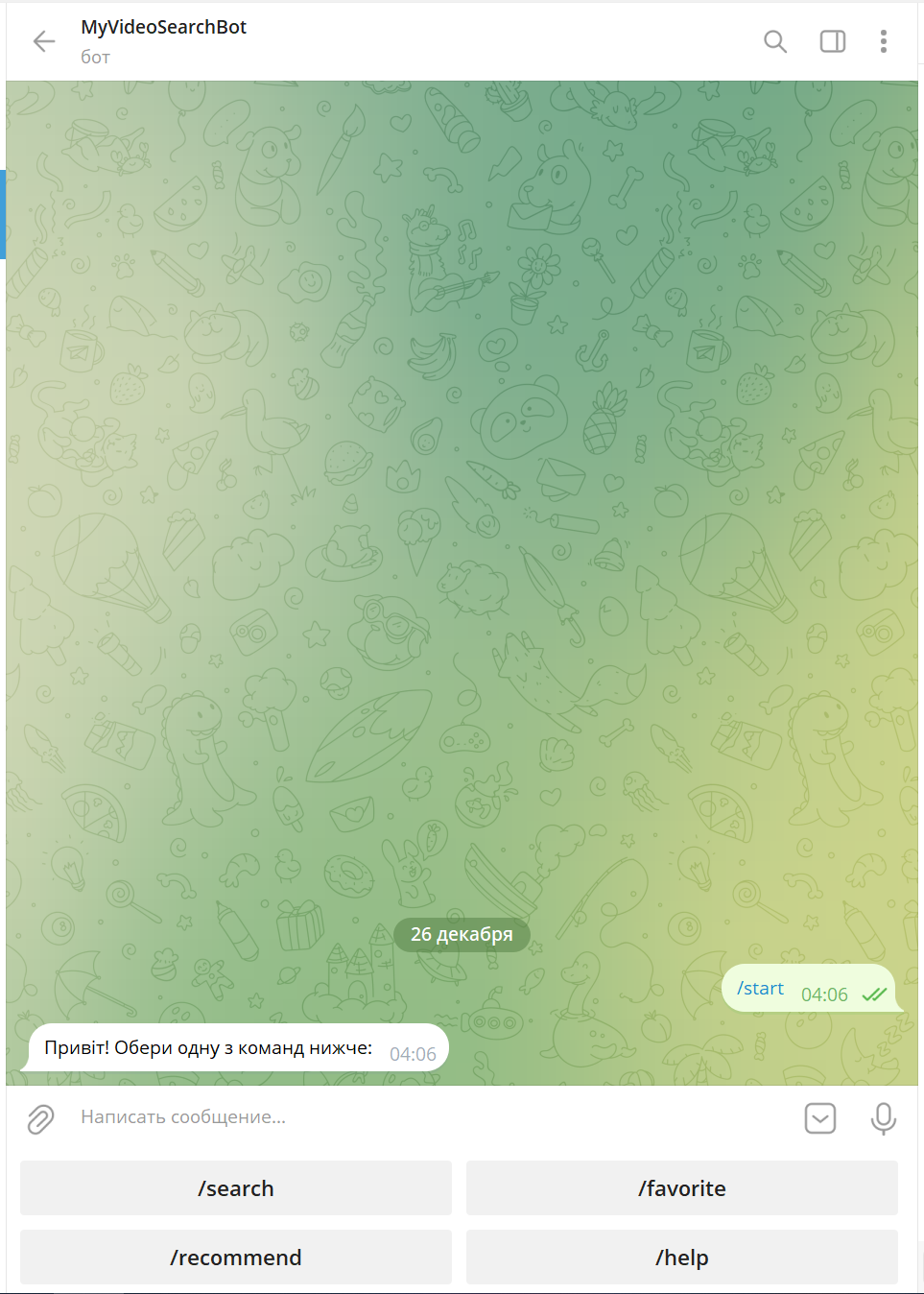
# ДОДАТОК И. РЕЗУЛЬТАТИ ТЕСТУВАННЯ

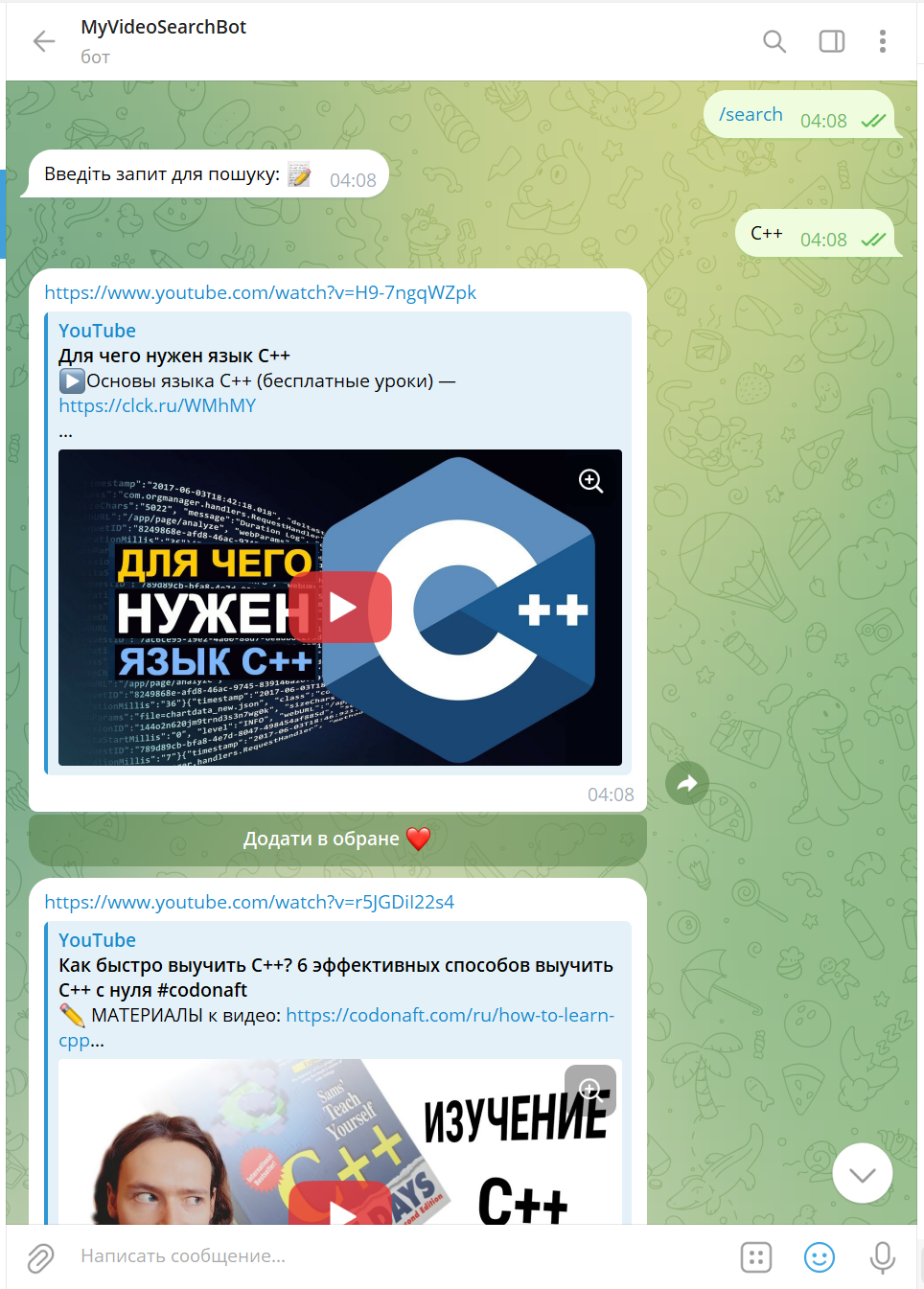
Цей додаток є підсумком розділу 2.2.!

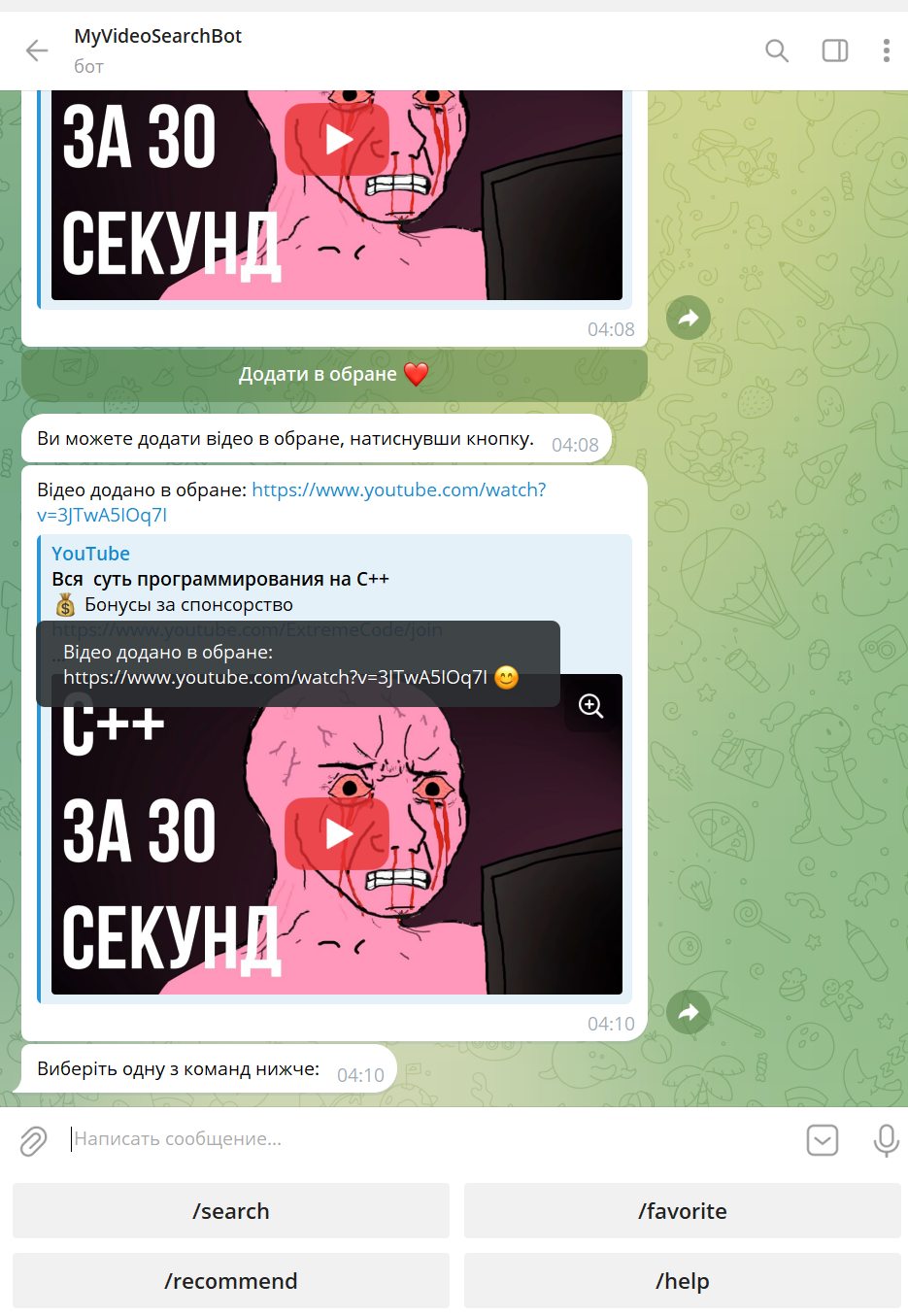
Використовуйте відповідні сервіси та фреймворки для проведення тестів (дивіться вище – опис розділу, присвяченому тестуванню)

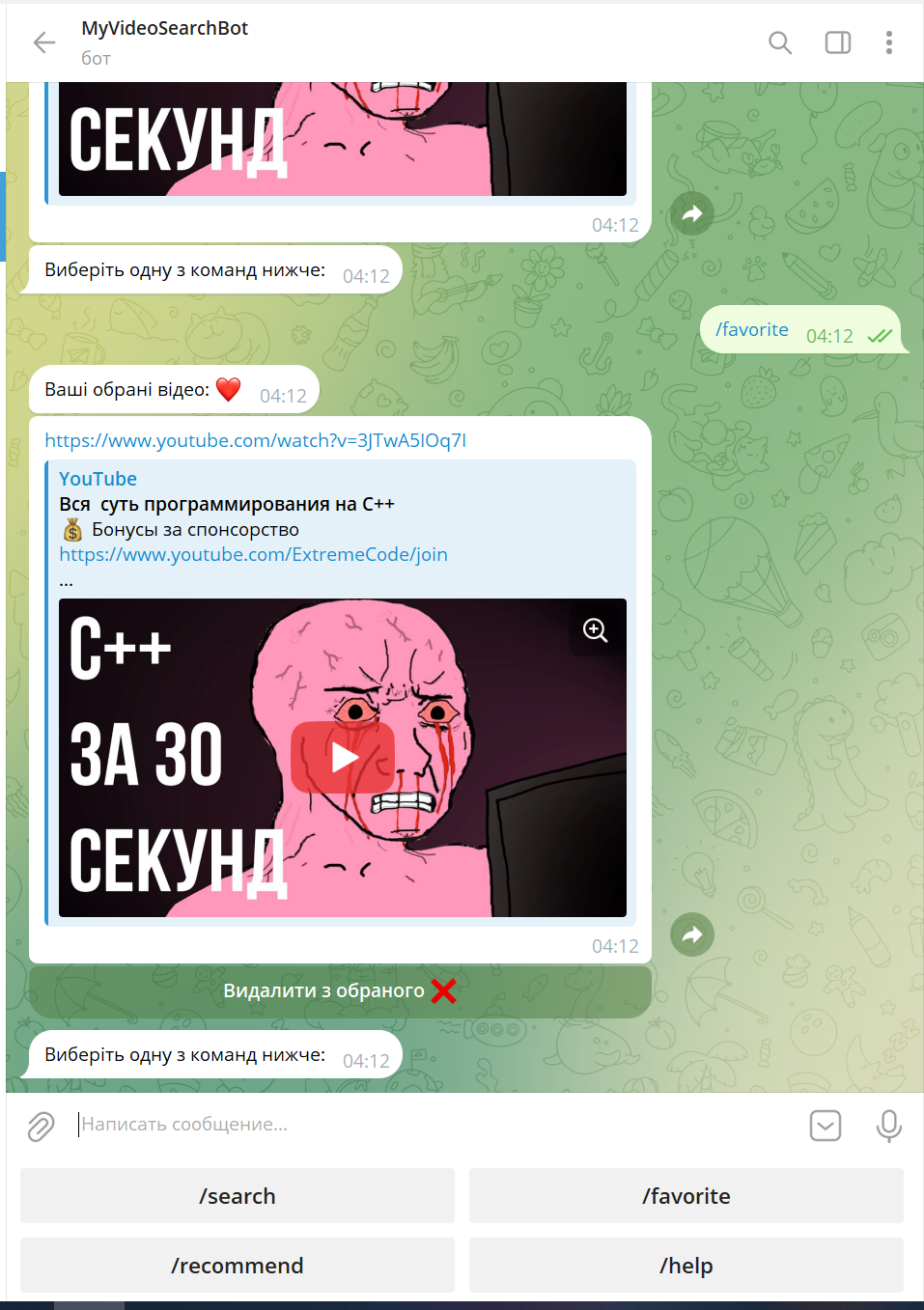
# ДОДАТОК Е. ЗНІМКИ ЕКРАНУ

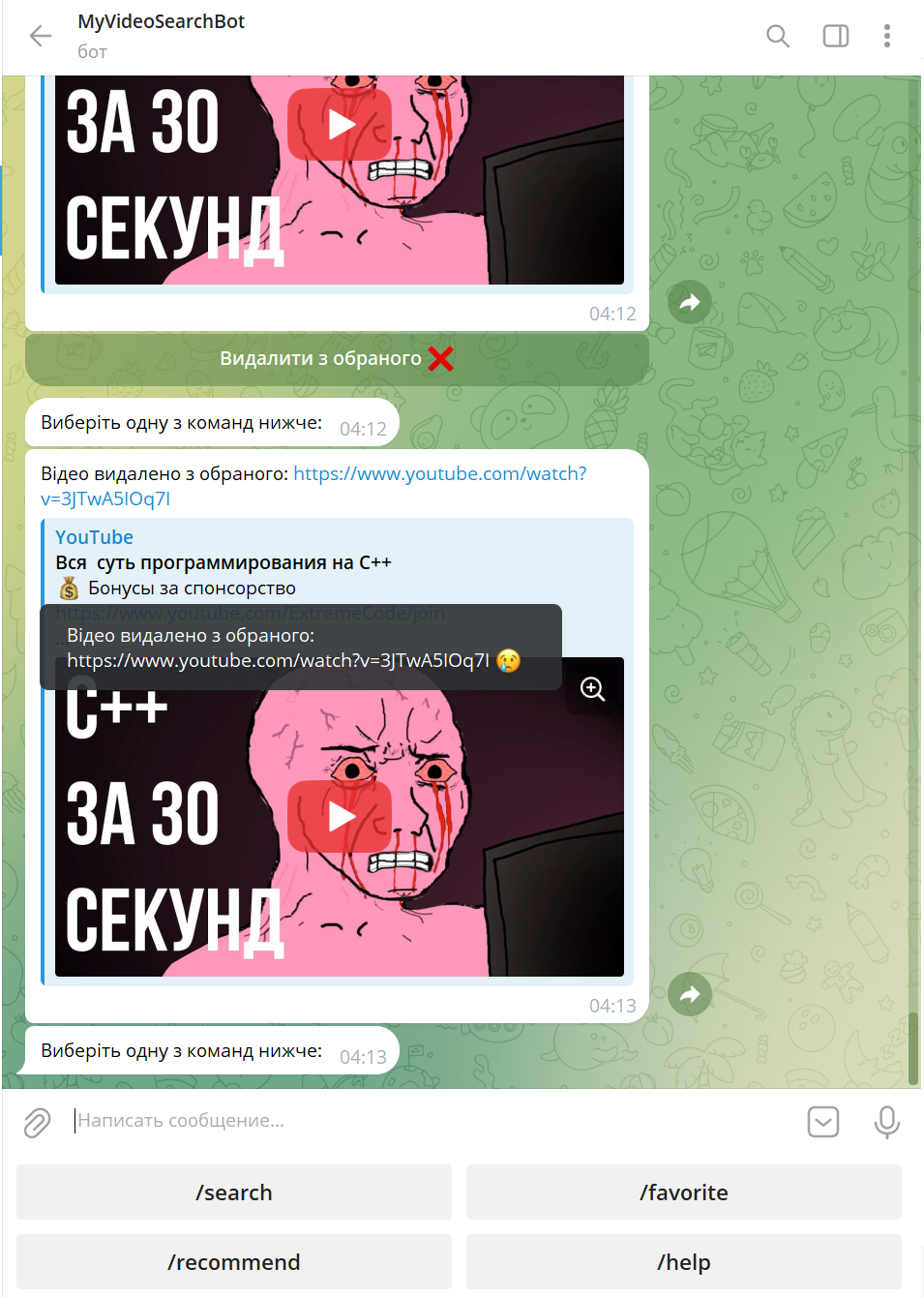


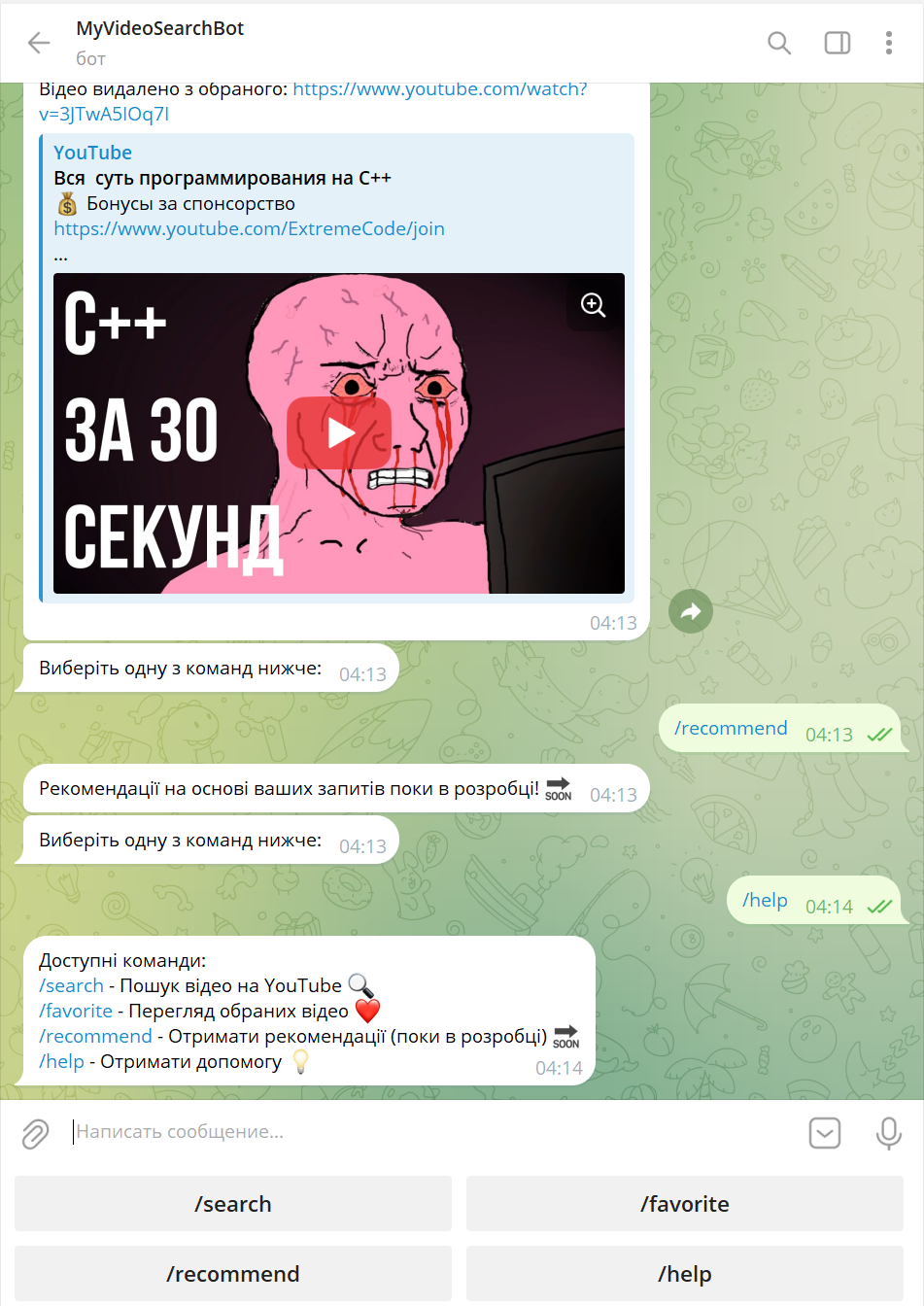




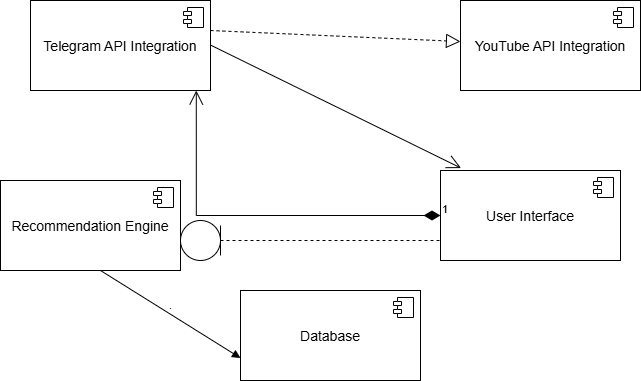




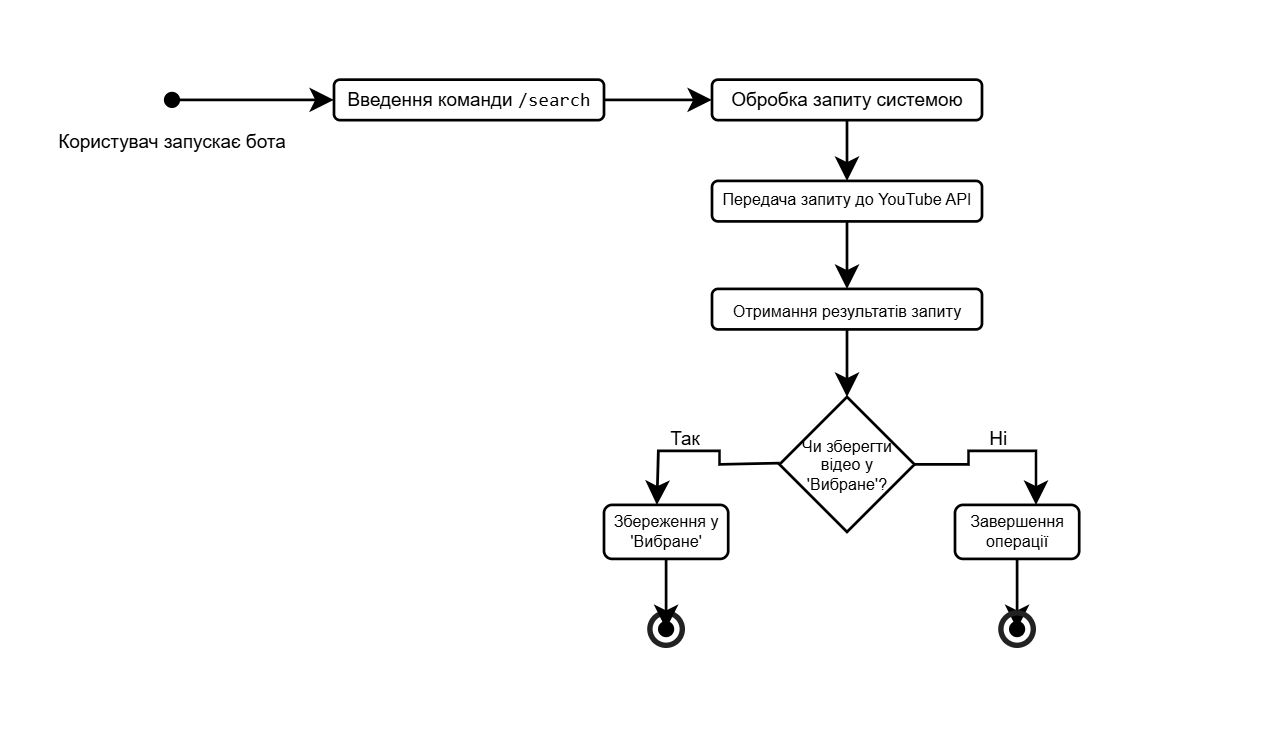




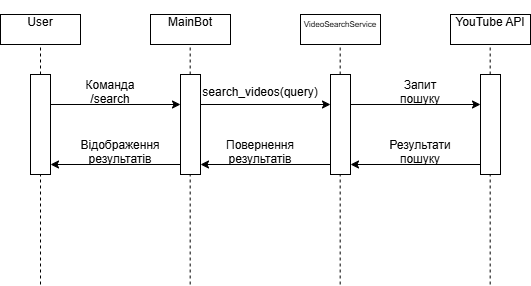
**ДОДАТОК Є. ДІАГРАМА КОМПОНЕНТІВ**

****

**ДОДАТОК З. ДІАГРАМА ДІЯЛЬНОСТЕЙ**

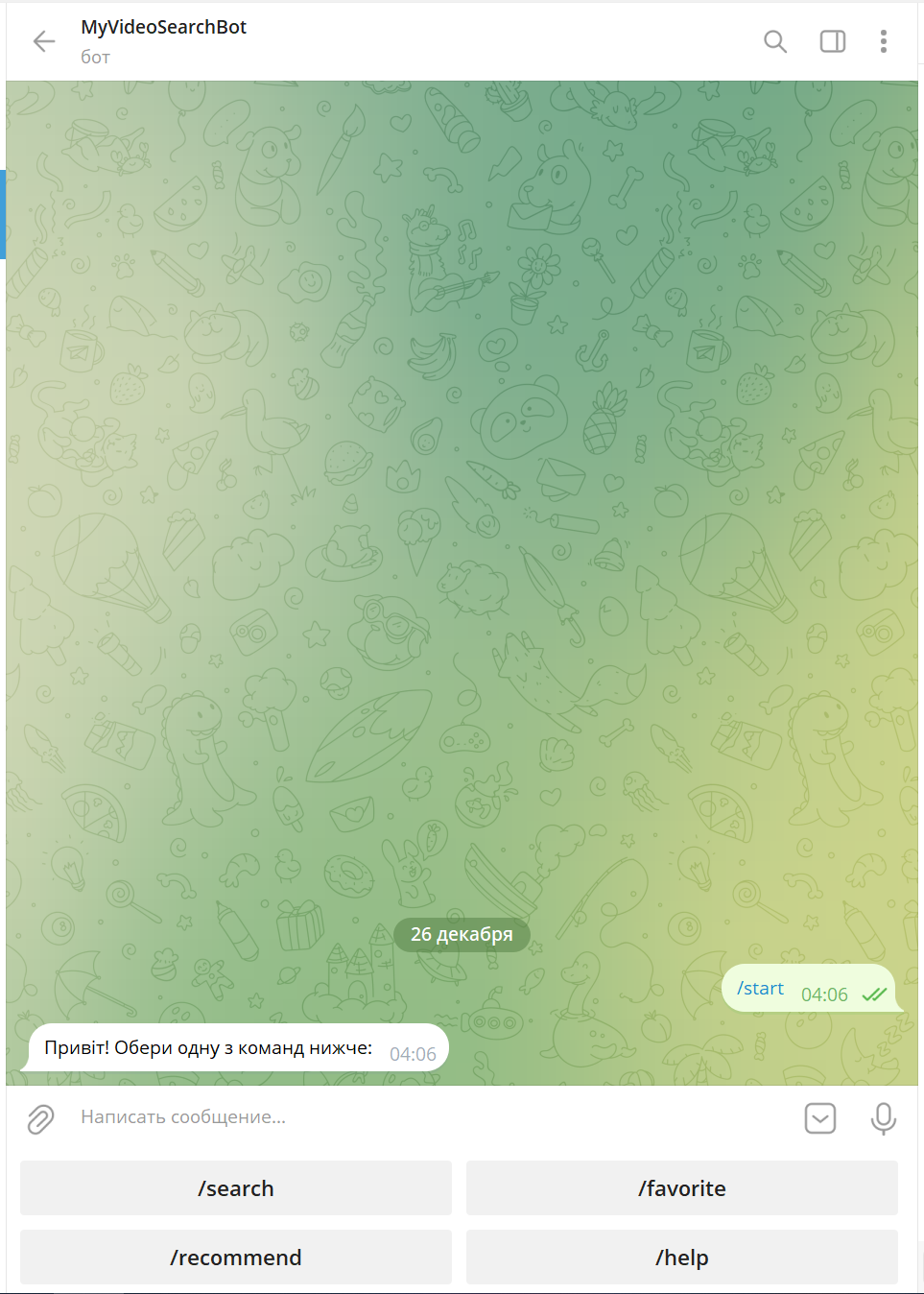
****

**ДОДАТОК К. ДІАГРАМА ПОСЛІДОВНОСТЕЙ**

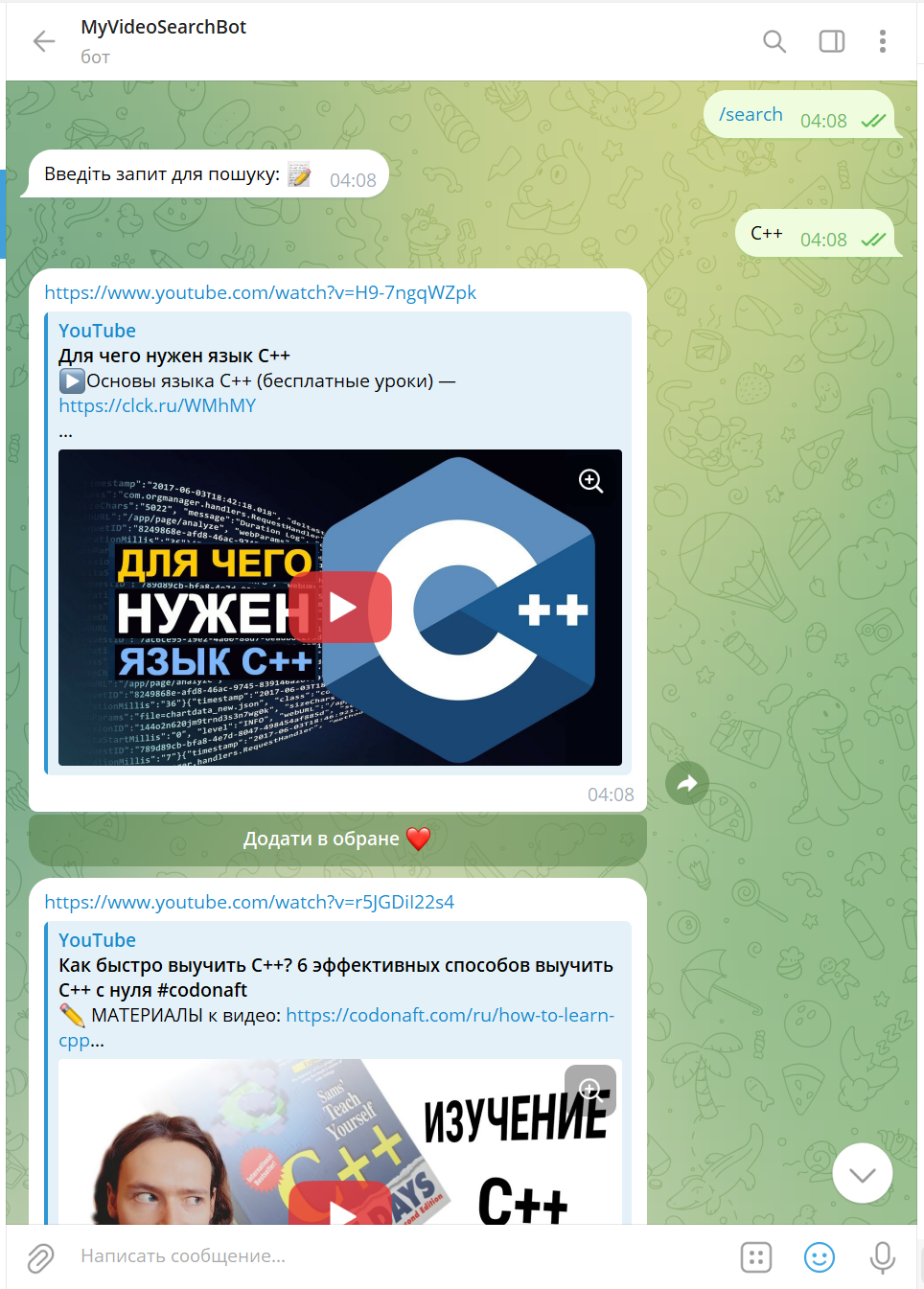
****

**ДОДАТОК Х. ІНСТРУКЦІЯ З ВИКОРИСТАННЯ СИСТЕМИ**

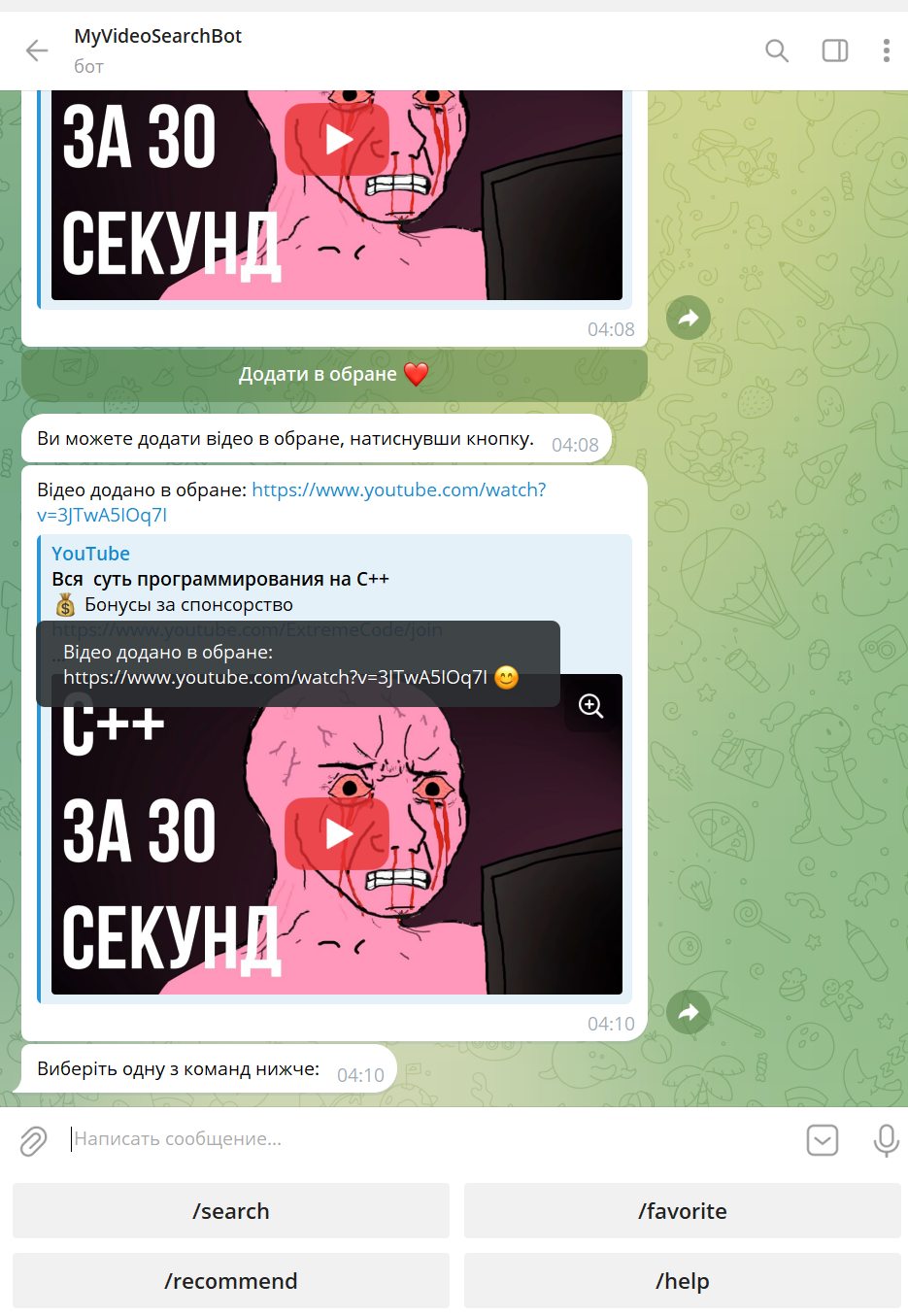
1:



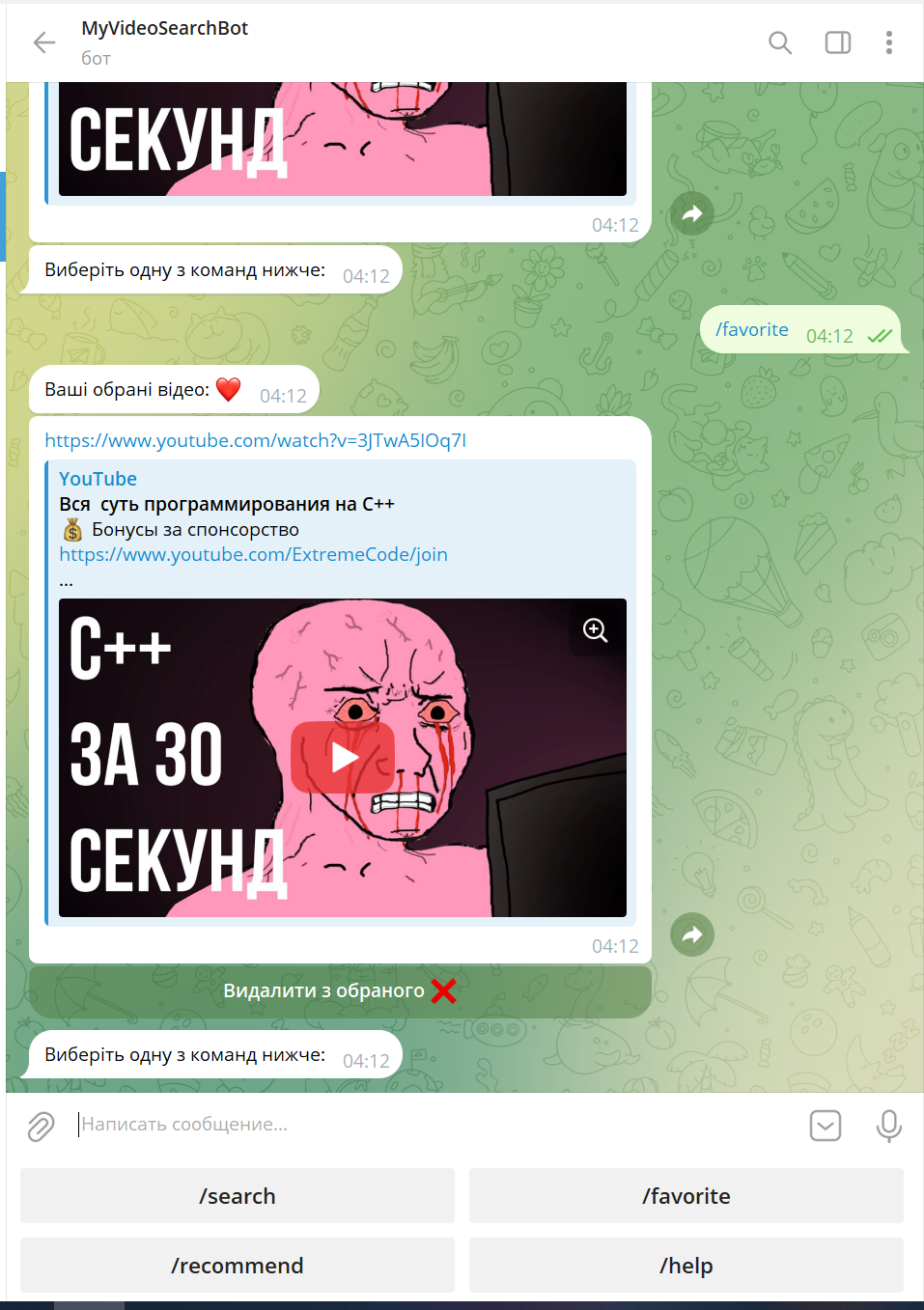
2:



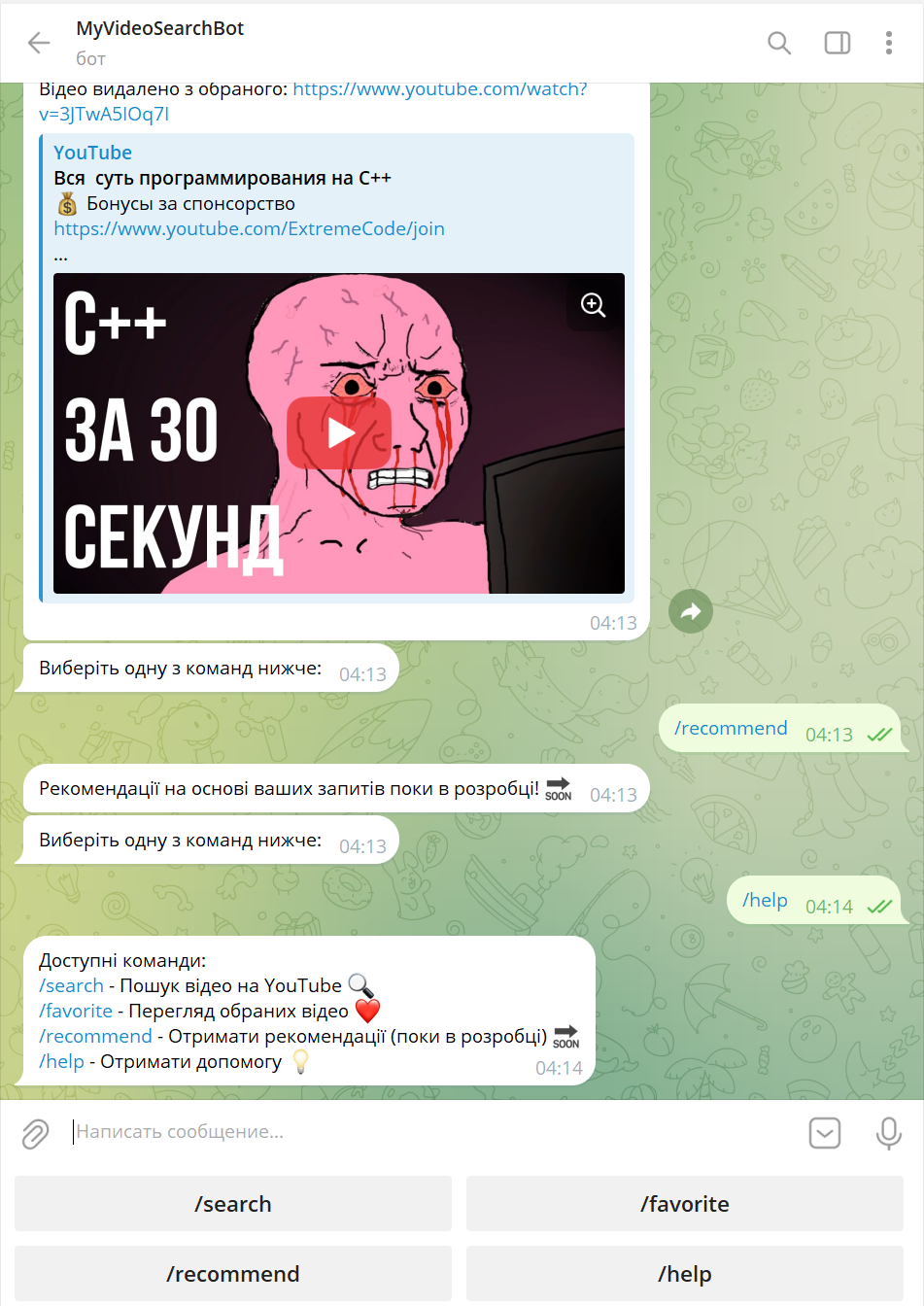
3:



4:



5:



6:

