МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПОЛІСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій, обліку та фінансів
Кафедра комп'ютерних технологій
і моделювання систем

Кваліфікаційна робота
на правах рукопису

Котляр Вадим Петрович

УДК 004.413.2: 004.415.2.031.43:004.421.2

КВАЛІФІКАЦІЙНА РОБОТА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АВТОМАТИЗАЦІЇ ПОДІЙ НА ПЛАТФОРМІ DISCORD

122 «Комп'ютерні науки»

Подається на здобуття освітнього ступеня бакалавр

кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____
(підпис, ініціали та прізвище здобувача вищої освіти)

Керівник роботи
Молодецька Катерина Валеріївна
доктор технічних наук, професор

Житомир – 2024

Висновок кафедри _____
за результатами попереднього захисту: _____

Протокол засідання кафедри _____
№ ____ від «_____» _____ 20____ р.

Завідувач кафедри _____

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)
«_____» _____ 20____ р.

Результати захисту кваліфікаційної роботи

Здобувач вищої освіти _____ захистив (ла)
(прізвище ,ім'я, по батькові)
кваліфікаційну роботу з оцінкою:

сума балів за 100-бальною шкалою _____
за шкалою ECTS _____
за національною шкалою _____

Секретар ЕК

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)

АНОТАЦІЯ

Котляр В.П. Інформаційна технологія автоматизації подій на платформі Discord. – Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавра за спеціальністю 122 – комп’ютерні науки. – Поліський національний університет, Житомир, 2024.

Кваліфікаційна робота присвячена інформаційній системі автоматизації подій на платформі Discord. Для досягнення мети роботи у теоретичній частині проведено дослідження предметної області, особливості задач та потреб користувачів платформи у автоматизації, підходи до спрощення та пришвидшення процесу автоматизації. Описано архітектуру моделі системи автоматизації подій та її окремих модулів, WEB-застосунок для кастомізації налаштувань та Discord Application, в ролі виконавчого модулю. У практичній частині проведено тестовий запуск системи для автоматизації одного Discord серверу. Отримано та проаналізовано результати роботи системи. Наведено рекомендації, щодо впровадження системи та її використання та щодо її подальшого розвитку.

Ключові слова: система автоматизації подій, платформа Discord, Discord Bot, Discord Application, WEB-застосунок, Discord.js.

SUMMARY

Kotlyar V.P. Event automation information technology on the Discord platform. – Qualification work as a manuscript.

Qualification work for obtaining a bachelor's degree in specialty 122 – computer science. – Polissia National University, Zhytomyr, 2024.

The qualification work is devoted to the event automation information system on the Discord platform. To achieve the goal of the work, in the theoretical part, a study of the subject area, specific tasks and needs of platform users in automation, approaches to simplifying and speeding up the automation process was conducted. The architecture of the event automation system model and its individual modules, the WEB-application for customizing settings and the Discord Application as an executive module are described. In the practical part, a test launch of the system for automating one Discord server was carried out. The results of the system were obtained and analyzed. Recommendations are given on the implementation of the system and its use and on its further development.

Keywords: event automation system, Discord platform, Discord Bot, Discord Application, WEB-application, Discord.js.

ПОЛІСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій, обліку та фінансів

Кафедра комп'ютерних технологій і моделювання систем

Спеціальність 122 «Комп’ютерні науки»

“ЗАТВЕРДЖУЮ”
Завідувач кафедри комп’ютерних
технологій і моделювання систем
_____О. М. Николюк
“___” _____ 2024

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Котляр Вадим Петрович

1. Тема кваліфікаційної роботи: «Інформаційна технологія автоматизації подій на платформі Discord».

затверджена наказом №1457/ст від 13.10.2023

2. Термін подання роботи: 17.05.2025р.

3. Предмет дослідження: методи, способи та підходи до автоматизації визначених подій у межах замкнених спільнот на онлайн-платформі Discord.

4. Об’єкт дослідження: процес застосування методів, способів та підходів щодо автоматизації визначених подій у межах замкнених спільнот на онлайн-платформі Discord.

5. Методи дослідження: методи моделювання, методи аналізу, методи порівняння, методи комп'ютерного проектування та моделювання.

6. Інформаційна база дослідження: відкриті електронні інформаційні ресурси, порівняльний аналіз існуючих рішень та їх можливостей, власних досвід користування подібними системами.

7. Зміст роботи: аналіз інформаційних потреб користувачів платформи Discord, розробка моделі інформаційної технології автоматизації подій на платформі Discord, створення та тестування інформаційної технології автоматизації подій.

8. Перелік графічного матеріалу: 2 табл., 25 рис., 30 джерел.

9. Дата видачі завдання: 13.10.2023.

Керівник роботи

науковий ступінь, вчене звання _____ д.т.н., проф. К. В. Молодецька

Завдання прийняв

до виконання _____ В. П. Котляр

КАЛЕНДАРНИЙ ПЛАН РОБОТИ

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1.	Вибір теми кваліфікаційної роботи	07.09.2023	виконано
2.	Розробка завдання та плану кваліфікаційної роботи. Погодження з науковим керівником	13.10.2023	виконано
3.	Написання розділу 1	21.11.2023 - 28.03.2024	виконано
4.	Написання розділу 2	01.03.2024 - 10.03.2024	виконано
5.	Написання розділу 3	15.03.2024 - 25.03.2024	виконано
6.	Підготовка тез для участі в конференції	26.03.2024 - 01.05.2024	виконано
7.	Підготовка матеріалів до друку	10.05.2024	виконано
8.	Підготовка доповіді та презентації до захисту кваліфікаційної роботи	10.05.2024	виконано

Здобувач вищої освіти _____ В. В. Колесник

Керівник роботи

науковий ступінь, вчене звання _____ д.т.н., проф. К. В. Молодецька

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

DB – Data Base / БД – База Даних;

App – Application;

TS – TypeScript;

JS – JavaScript;

SSR – Server-Side Rendering;

SDK – Software Development Kit;

API – Application Programming Interface.

ВСТУП

У сучасних умовах розвитку інформаційних технологій спостерігається стрімке зростання попиту на автоматизацію процесів, що сприяє підвищенню ефективності роботи різних онлайн-спільнот. Однією з популярних платформ для реалізації таких завдань є Discord [1] – [3], що надає можливості для інтеграції ботів (нині перейменованих в App) [4] – [6] та автоматизації подій. Автоматизація подій на платформі Discord дозволяє значно спростити управління різноманітними процесами всередині замкнених спільнот, що є актуальним для забезпечення комфортної та ефективної комунікації та взаємодій між користувачами.

Метою роботи є створення інформаційної технології для виконання та створення модулів автоматизації подій на платформі Discord. Ця система повинна включати впровадження графічного інтерфейсу для конструювання модулів та відображення результатів їх роботи, що полегшить та пришвидшить процес створення модулів автоматизації для широкої аудиторії, включаючи тих користувачів, які не знайомі з методиками та мовами програмування.

Об’єктом роботи є процес автоматизації подій на платформі Discord, що досягається шляхом надання користувачам можливості створювати власні модулі автоматизації, які можуть бути налаштовані відповідно до індивідуальних потреб кожного користувача чи спільноти. Такий підхід дозволяє значно розширити функціональність платформи, забезпечуючи високий рівень адаптивності та персоналізації.

Предметом дослідження є конструктор для створення власних алгоритмів та модулів автоматизації подій у межах платформи Discord. Дослідження також охоплює документацію базового користування системою, яка допомагає користувачам освоїти базові навички створення та налагодження модулів автоматизації.

Актуальність теми обумовлена зростаючими потребами користувачів та спільнот Discord, які постійно розширюються та ускладнюються. Із зростанням кількості користувачів збільшується і кількість окремих модулів, які задовольняють специфічні потреби вузького кола користувачів [7]. Сучасні умови

вимагають об'єднання всіх можливих сценаріїв у один універсальний інструмент, так званий all-in-one підхід. Це дозволяє швидко та легко налаштовувати і конфігурувати необхідний функціонал без нагромадження окремих, незалежних модулів. Також, спільний підхід до вирішення різних задач сприяє формування консистентного інтерфейсу та функціонального балансу.

Запропонована система дозволяє користувачам швидко і легко реалізувати необхідний функціонал самостійно, навіть якщо готове рішення ще не знайдено. Такий підхід спрощує використання різноспрямованих функцій та оптимізує процес налаштування системи, що є важливим для зростаючих спільнот, де швидкість і зручність користування мають вирішальне значення.

Розробка інформаційної технології автоматизації подій на платформі Discord є важливим кроком у забезпеченні ефективного управління та функціонування онлайн-спільнот. Система, що об'єднує всі можливі сценарії автоматизації в одному інтерфейсі, забезпечує простоту і зручність використання для широкого кола користувачів, сприяючи їх залученню в процес.

Також, за темою кваліфікаційної роботи опубліковано наукові публікації в контексті наукових конференцій, а саме:

– Котляр В.П. Перспективи застосування інформаційної технології автоматизації подій на платформі Discord: збірник праць учасників між факультетської науково-практичної інтернет-конференції «Безпека, технології, інновації: нові горизонти», 2023, с. 16-17

– Котляр В.П. Основні методики створення інформаційної технології автоматизації подій на платформі Discord: збірник праць учасників Всеукраїнської науково-практичної конференції здобувачів вищої освіти і молодих вчених «інформаційні технології та моделювання систем», 2024, с. 53-54

РОЗДІЛ 1. АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТРЕБ КОРИСТУВАЧІВ ПЛАТФОРМИ DISCORD

1.1 Особливості автоматизації подій платформи

Автоматизація подій на платформі Discord відкриває перед користувачами широкі можливості для створення інтерактивного та організованого середовища. Discord, як платформа для обміну повідомленнями та спілкування в голосових чатах, стала невід'ємною частиною багатьох онлайн-спільнот, від ігрових гільдій до професійних колективів та навчальних груп. Однак, щоб ефективно використовувати всі можливості платформи, необхідно глибше зрозуміти поточні можливості та обмеження ботів, сучасні методики їх реалізації, а також самі можливості додатку Discord у контексті автоматизації.

У сучасному цифровому світі автоматизація стає ключовим елементом ефективного управління різноманітними процесами. Платформа Discord, спочатку створена для геймерів, набула значної популярності серед різних аудиторій завдяки своїй гнучкості та багатофункціональності [8] – [9]. Автоматизація подій на платформі Discord допомагає користувачам оптимізувати рутинні завдання, зменшити навантаження на адміністраторів серверів, та забезпечити плавний та інтерактивний досвід для всіх учасників спільнот.

Автоматизація подій дозволяє значно покращити взаємодію між учасниками спільнот та забезпечує високий рівень організованості. Наприклад, автоматичні відповіді на часто задавані питання, управління ролями та правами доступу, проведення опитувань та активностей, а також інтеграція з зовнішніми сервісами – все це робить платформу Discord більш зручною та функціональною.

Автоматизація також дозволяє розробникам створювати складні системи, які можуть виконувати специфічні завдання на основі заданих параметрів. Це включає в себе не лише обробку команд користувачів, але й реакцію на певні події, такі як приєднання нових учасників, зміна налаштувань сервера або оновлення контенту. Боти в Discord стали важливим елементом управління серверами та взаємодії між користувачами. Вони виконують різноманітні завдання, від автоматичного

привітання нових учасників до управління ролями та виконання команд. Поточні можливості ботів можна умовно розділити на кілька категорій:

Модерація та управління сервером:

– Автоматичне привітання та ролі: Боти можуть вітати нових учасників сервера автоматичними повідомленнями, надаючи їм інформацію про правила та особливості спільноти. Вони також можуть автоматично призначати ролі новим користувачам, спрощуючи процес інтеграції;

– Модерація контенту: Відстеження та видалення небажаних повідомлень, блокування спам-ботів та автоматичне попередження користувачів, які порушують правила. Наприклад, боти можуть автоматично видаляти повідомлення з ненормативною лексикою або зображення, які не відповідають політиці сервера;

– Управління ролями: Автоматизація надання та зняття ролей на основі дій користувачів або визначених правил. Це може включати надання спеціальних ролей активним учасникам або учасникам, які пройшли певні перевірки.

Взаємодія з користувачами:

– Відповіді на команди: Боти можуть реагувати на команди, що вводяться користувачами, виконуючи різні дії, такі як надання інформації, запуск ігор або інтерактивних опитувань. Це робить взаємодію на сервері більш динамічною та інтерактивною;

– Опитування та голосування: Проведення опитувань та голосувань серед учасників сервера, що дозволяє швидко збирати думки спільноти та приймати колективні рішення;

– Ігри та розваги: Боти можуть пропонувати різноманітні міні-ігри, вікторини та інші розваги, що сприяє активному залученню користувачів та створенню дружньої атмосфери.

Інтеграція з іншими сервісами:

– Сповіщення та оновлення: Інтеграція з сервісами, такими як YouTube, Twitch, GitHub, що дозволяє отримувати повідомлення про нові відео, трансляції

або зміни в репозиторіях безпосередньо в каналах Discord. Це зручно для спільнот, які хочуть бути в курсі останніх новин та оновлень;

– Автоматичне поширення контенту: Боти можуть автоматично публікувати повідомлення з новинами або оновленнями з різних платформ, що знижує необхідність вручну передавати інформацію учасникам сервера.

Автоматизація подій та процесів:

– Планування подій: Боти можуть допомагати в плануванні та організації подій, таких як зустрічі, ігрові сесії або вебінари, відправляючи нагадування та сповіщення учасникам. Вони також можуть відстежувати присутність і надавати звіти про участь;

– Нагадування та розклад: Автоматичне відправлення нагадувань про важливі події, дедлайни або завдання. Це може бути особливо корисним для навчальних або робочих груп, де важливо дотримуватися встановлених графіків;

– Обробка запитів: Боти можуть автоматично обробляти запити користувачів, такі як заявки на приєднання до певних груп, запити на інформацію або підтримку.

Ці можливості роблять ботів незамінними помічниками для адміністраторів серверів та активних учасників спільнот, забезпечуючи більш ефективне управління та взаємодію. Вони дозволяють автоматизувати рутинні завдання, покращити організацію роботи та створити більш інтерактивне середовище для всіх учасників платформи.

Однак, автоматизація подій на платформі Discord також має свої виклики. Це включає в себе необхідність врахування обмежень API [10] – [11], управління правами доступу [12], забезпечення безпеки та конфіденційності даних [13]. Крім того, реалізація складних сценаріїв автоматизації може вимагати значних технічних знань та досвіду у програмуванні та роботі алгоритмів. Таким чином, важливо розробити рішення, які будуть не лише функціональними, але й зручними у використанні для кінцевих користувачів. Але не зважаючи на широкі можливості, боти в Discord мають певні обмеження, які варто враховувати під час розробки та впровадження автоматизованих рішень [14] – [15]:

Обмеження API:

– Ліміти запитів: Discord API має обмеження на кількість запитів, які бот може виконувати за певний проміжок часу. Це означає, що бот не може безперервно надсилати або отримувати запити без ризику бути тимчасово заблокованим. Це обмеження запобігає перевантаженню серверів Discord та забезпечує стабільну роботу платформи;

– Частковий доступ до функціоналу: Не всі можливості Discord доступні через API. Наприклад, деякі дії, пов'язані з керуванням серверами або обробкою мультимедійного контенту, можуть бути обмеженими або взагалі недоступними для ботів.

Права доступу:

– Обмеження за ролями: Боти можуть виконувати дії тільки в межах дозволених їм прав. Це означає, що для виконання певних завдань адміністратори серверів повинні заздалегідь налаштувати відповідні права доступу для ботів. Неправильне налаштування прав може призвести до того, що бот не зможе виконувати свої функції;

– Безпека доступу: Налаштування прав доступу ботів повинно бути ретельним, щоб уникнути випадкового надання їм надмірних привілеїв, що може створити ризики безпеки.

Складність налаштування:

– Технічні знання: Хоча створення базових ботів відносно просте, реалізація складних сценаріїв автоматизації може вимагати глибоких знань програмування та розуміння архітектури Discord. Для багатьох користувачів, які не мають технічної підготовки, це може стати суттєвою перепоною;

– Налаштування та підтримка: Підтримка та оновлення ботів може бути складним процесом, особливо якщо ботів багато і вони виконують різні завдання. Це потребує постійного моніторингу та вдосконалення коду для підтримки його актуальності та ефективності.

Безпека та конфіденційність:

– Зберігання даних: Боти можуть обробляти та зберігати персональні дані користувачів, що створює ризики витоку інформації. Важливо забезпечити належний рівень захисту даних та дотримуватися політик конфіденційності;

– Захист від атак: Боти можуть стати ціллю для зловмисників, які можуть спробувати отримати контроль над ними або використовувати їх для проведення DDoS-атак. Це вимагає впровадження заходів захисту, таких як перевірка прав доступу, шифрування даних та регулярне оновлення безпекових патчів.

Взаємодія з користувачами:

– Розуміння команд: Боти повинні точно розпізнавати команди користувачів та відповідати на них належним чином. Однак, складність природної мови може спричиняти помилки в розумінні команд, що може знижувати ефективність взаємодії;

– Зручність використання: Інтерфейс взаємодії з ботами повинен бути інтуїтивно зрозумілим, щоб користувачі могли легко користуватися їхніми функціями. Це особливо важливо для користувачів, які не мають технічних знань.

Інтеграція з іншими сервісами:

– Сумісність: Боти можуть мати проблеми з інтеграцією з деякими зовнішніми сервісами через обмеження API або зміну політик доступу цих сервісів. Це може впливати на стабільність та функціональність ботів;

– Залежність від зовнішніх платформ: Платформи, з якими інтегруються боти, можуть змінювати свої API або припиняти підтримку певних функцій, що може призводити до необхідності переробки ботів та їхнього функціоналу.

Урахування цих обмежень є важливим при плануванні та розробці ботів для Discord. Незважаючи на технічні та організаційні виклики, які можуть виникати, правильне проектування та налаштування ботів дозволяє максимально використовувати їх потенціал для автоматизації завдань та покращення взаємодії в рамках платформи Discord. Для розробки ефективних ботів необхідно враховувати поточні можливості платформи Discord, включаючи підтримку інтеграцій, можливість використання Webhooks, а також гнучкість у налаштуванні ролей та прав доступу, а також і недоліки самого підходу до створення таких систем,

обмеження Discord та зовнішніх залежностей. Це дозволяє створювати системи, які можуть автоматично реагувати на різні події та виконувати необхідні дії без втручання людини.

1.2. Підходи та методи автоматизації подій платформи

Автоматизація подій на платформі Discord включає різні методи та інструменти, які дозволяють користувачам налаштовувати та керувати своїми серверами більш ефективно. Сучасні підходи до автоматизації можна класифікувати наступним чином:

– Широкий вибір: Створено безліч боті, призваних спростити або повністю взяти на себе виконання якихось функцій на серверах Discord. Найпопулярнішими із таких системи є: MEE6, Dyno, Carl-bot, та інші [16] – [17]. Вони дозволяють автоматизовувати основний набір базових та більш складних задач, таких як: модерація, привітання нових учасників, проведення опитувань, та інше;

– Налаштування через веб-інтерфейс: Більшість із існуючих систем не пропонують зручні веб-інтерфейси для налаштування, що не дозволяє адміністраторам легко інтегрувати та налаштовувати функціонал без необхідності детального вивчення документації та можливостей системи. Впровадження такого рішення є одним із ключових параметрів при виборі системи автоматизації;

– Гнучкість: Створення власного бота надає максимальну гнучкість, дозволяючи реалізувати специфічні сценарії, які не підтримуються готовими ботами. Це може включати індивідуальні команди, інтеграцію з власними сервісами, або особливі механізми модерації;

– Використання SDK та API: Для створення власних систем зазвичай використовуються бібліотеки та SDK, такі як discord.js для JavaScript/Node.js, discord.py для Python, та інші. Ці інструменти дозволяють розробникам взаємодіяти з Discord API та створювати ботів з необхідними функціями;

– Webhooks: Webhooks дозволяють отримувати повідомлення про події в реальному часі та використовуються для інтеграції Discord з зовнішніми сервісами. Це може включати автоматичне відправлення повідомлень про нові відео на YouTube, оновлення в репозиторіях GitHub, або сповіщення про стріми на Twitch;

– Інтеграції через API: Використання API зовнішніх сервісів дозволяє розширювати функціональність ботів та додавати нові можливості. Наприклад, інтеграція з сервісами аналітики або базами даних для зберігання та обробки даних учасників.

Незважаючи на різноманітність та багатофункціональність сучасних ботів, існують певні недоліки, які обмежують їх ефективність та зручність використання. Ці недоліки можуть проявлятися у вигляді складності налаштування, необхідності технічних знань для повноцінного використання, обмеженої функціональності окремих ботів, а також у потенційних конфліктах між різними ботами, встановленими на одному сервері. Крім того, зміни в API або умовах використання Discord можуть призвести до нестабільної роботи ботів, що вимагає постійного моніторингу та оновлення. Усі ці фактори можуть суттєво ускладнити процес автоматизації та знижувати загальну ефективність використання ботів для управління серверами.

Основними можна виділити перенасичення платформи різнонаправленими та вузьконаправленими ботами, які не здатні чітко та якісно задовольнити потреби користувачів. Користувачам часто доводиться використовувати кілька різних ботів для виконання різних завдань. Це ускладнює управління та може призводити до конфліктів між ботами. Багато ботів пропонують обмежений набір функцій, що змушує користувачів шукати додаткові рішення для задоволення всіх потреб. Причини та наслідки такої проблеми можна визначити в декількох категоріях:

Складність налаштування та інтеграції:

– Технічні навички: Налаштування та інтеграція ботів часто вимагають технічних знань, що може бути проблемою для користувачів без відповідного досвіду;

– Часовитратність: Процес налаштування та підтримки кількох ботів може займати багато часу та ресурсів.

Проблеми сумісності та стабільності:

– Конфлікти між ботами: Використання кількох ботів на одному сервері може призводити до конфліктів та проблем із сумісністю, що впливає на стабільність роботи сервера;

– Зміни API: Зміни в API Discord або зовнішніх сервісів можуть призводити до збоїв у роботі ботів, що потребує постійного моніторингу та оновлення коду.

Окрім ботів, платформа Discord має вбудовані можливості, які можуть бути використані для автоматизації без необхідності додаткових інструментів. Ці можливості надають адміністраторам та користувачам зручні інструменти для організації та управління сервером, дозволяючи налаштовувати різні аспекти функціонування спільноти. Використання вбудованих функцій Discord може значно спростити процес автоматизації, зменшивши залежність від сторонніх ботів та інструментів, а також підвищивши стабільність та безпеку роботи сервера. Основні вбудовані можливості автоматизації, присутні на платформі можна поділити на наступні категорії:

Серверні налаштування:

– Ролі та права доступу: Discord дозволяє налаштовувати різні ролі та права доступу для учасників сервера, що забезпечує контроль за діями користувачів та автоматизацію управління правами.

– Інтеграції з іншими платформами: Discord пропонує інтеграції з платформами, такими як Twitch та YouTube, що дозволяє користувачам мати в профілі відмітку при прив’язану інтеграцію.

Канали та категорії:

– Структурування контенту: Використання каналів та категорій дозволяє організувати контент на сервері та спростити навігацію для користувачів.

– Автоматичні сповіщення: Можливість налаштовувати сповіщення для різних каналів забезпечує інформування учасників про важливі події та оновлення.

Простий інтерфейс для управління:

– Управління учасниками: Вбудовані інструменти для управління учасниками, такі як бан, кік або призначення ролей, спрощують адміністративні завдання.

– Модерація: Інструменти для модерації, включаючи фільтри мови та управління контентом, забезпечують безпечне та організоване середовище на сервері.

Сучасні підходи до автоматизації подій на платформі Discord включають використання стандартних ботів, створення власних ботів та інтеграцію з зовнішніми сервісами. Хоча ці методи мають свої переваги, вони також мають недоліки, зокрема велика кількість різних ботів, складність налаштування, проблеми сумісності та стабільності. Вбудовані можливості Discord дозволяють автоматизувати деякі завдання без необхідності додаткових інструментів, що може спростити управління серверами та покращити користувацький досвід.

Висновки до першого розділу

Поглиблений аналіз та розуміння усіх сучасних підходів та методів, які використовуються розробниками для реалізації систем автоматизації подій в межах платформи Discord показав основні переваги та недоліки самої концепції, методів розробки та уже готових рішень.

З одного боку, автоматизація подій сприяє оптимізації управління серверами та взаємодії між користувачами, забезпечуючи підвищену продуктивність та зручність для учасників. З іншого боку, існують вагомі обмеження та недоліки у сучасних підходах, що вимагає пошуку оптимальних рішень та постійного вдосконалення технічних засобів. Прозорий аналіз проблем та переваг допомагає визначити стратегії подальшого розвитку інформаційної технології в контексті Discord, спрямовані на покращення якості та функціональності системи.

Такий підхід підсилює необхідність розвитку нових технологій та інструментів для автоматизації, що дозволить досягти максимального рівня зручності та ефективності використання для всіх учасників спільноти. Для вирішення цих завдань та задоволення зростаючих потреб є необхідність створення інноваційних рішень для оптимальної автоматизації подій на платформі Discord із можливістю створення власних модулів, що забезпечить прозорість використання та вільні можливості для користувачів у створенні власної логіки роботи системи у необхідним їм ситуаціям.

РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ ІНФОРМАЦІЙНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ ПОДІЙ НА ПЛАТФОРМІ DISCORD

2.1. Архітектура інформаційної системи автоматизації подій на платформі Discord

Система автоматизації подій на платформі Discord реалізована у вигляді чотирьох окремих модулів, які взаємодіють між собою, створюючи ефективну і зручну екосистему для управління серверами та інтеграції автоматизованих процесів. Систематизований опис всіх модулів можна провести за такими характеристиками: призначення та функції, технології, функціональні можливості. Самі модулі та опис їх характеристик наведено нижче:

WEB-інтерфейс для конфігурації системи та взаємодії з «Розробниками»

– Призначення та функції: WEB-інтерфейс слугує точкою входу для користувачів з роллю «Розробник». Він надає зручні інструменти для створення, налаштування та управління модулями автоматизації. Користувачі можуть за допомогою цього інтерфейсу задавати події, встановлювати фільтри та обмеження, а також писати обробники подій на JavaScript. Також, в тестовому режимі доступна можливість повністю графічного створення модулів автоматизації, із деякими обмеженнями, порівняно із стандартним JS обробником;

– Технології: Інтерфейс реалізовано на Next.js у зв’язці з TypeScript, що забезпечує динамічність і інтерактивність користувацького досвіду. Для взаємодії з бекендом використовується RTK Query, що дозволяє обробляти запити за допомогою оптимізованих алгоритмів із кешуванням. Back-end також реалізований на Next.js і пересилає запити на виконавчий модуль;

– Функціональні можливості: Користувачі можуть створювати власні модулі автоматизації, налаштовувати параметри тригерів, визначати умови активації та додавати виконавчі блоки, що будуть виконуватись при спрацюванні певної події.

Виконавчий модуль

– Призначення та функції: Виконавчий модуль є основним механізмом, що відповідає за виконання створених користувачами сценаріїв автоматизації. Цей модуль отримує події від Discord API, аналізує їх відповідно до заданих тригерів і фільтрів, та виконує необхідні дії;

– Технології: Виконавчий модуль реалізовано за допомогою NodeJS, express, Discord.js та деяких пакетів оптимізації, що дозволяє ефективно взаємодіяти з Discord API, базами даних та виконувати різноманітні автоматизовані завдання;

– Функціональні можливості: Модуль обробляє події, такі як команди користувачів, зміни ролей, вхід нових учасників тощо, та виконує дії, прописані в модулях автоматизації, створених через WEB-інтерфейс.

База даних

– Призначення та функції: База даних зберігає всю інформацію про створені користувачами модулі автоматизації. Вона складається з двох частин: власної бази даних для зберігання модулів автоматизації та даних, створених користувачами, а також бази даних Discord, яка зберігає інформацію про користувачів, сервери та інше;

– Для зберігання даних використовується MongoDB, що забезпечує гнучкість, масштабованість та високу продуктивність. MongoDB дозволяє ефективно працювати з великими обсягами даних та забезпечує швидкий доступ до інформації про модулі автоматизації. Варто також зауважити, що з точки зору розробки, MongoDB не є найкращим вибором, через надлишкове навантаження на базу при збільшенні кількості запитів;

– Функціональні можливості: Зберігання модулів автоматизації: MongoDB забезпечує надійне збереження та швидкий доступ до інформації про створені користувачами модулі автоматизації. Структура бази даних дозволяє зберігати дані у форматі JSON-подібних документів, що спрощує процес зберігання та обробки складних даних. База даних інтегрується з Discord API для отримання та зберігання необхідної інформації про користувачів, сервери, ролі та інші дані, що використовуються у процесі автоматизації. Це дозволяє забезпечити

актуальність даних та швидкий доступ до них у будь-який момент. MongoDB забезпечує високу масштабованість, що дозволяє ефективно працювати зі збільшенням обсягів даних та кількістю користувачів системи. Це особливо важливо для великих спільнот та серверів, де кількість подій та даних може швидко зростати. База даних забезпечує надійний захист даних та можливість резервного копіювання, що дозволяє уникнути втрат важливої інформації та забезпечити її відновлення у разі необхідності.

Discord

– Призначення та функції: Discord є ключовим компонентом системи, що надає платформу для взаємодії користувачів з автоматизованими процесами. Користувачі взаємодіють із платформою як уже до цього звикли. Майже кожна дія, яка змінює стан системи буде доступна для обробки у виконавчому модулі, який виконує створені сценарії автоматизації.

– Технології: Discord надає API, через яке виконавчий модуль взаємодіє з платформою, отримуючи події та виконуючи команди. Користувачі ж взаємодіють із системою напряму через варіант застосунку: WEB, Mobile або Desktop;

– Функціональні можливості: Discord забезпечує середовище для взаємодії із користувачами та виконання автоматизованих завдань, таких як відповіді на команди користувачів, автоматичне управління ролями, сповіщення про події та багато іншого.

Розглянувши окремо основні складові системи та їх характеристики і можливості, наступним правильним кроком буде дослідження того, як вони взаємодіють між собою. Для більш точного представлення повного комплексу процесів взаємодії їх краще зобразити на часовій прямій у вигляді UML-діаграми послідовності. Зазначену діаграму приведено на рисунку 2.1 із використанням раніше описаних складових програми.

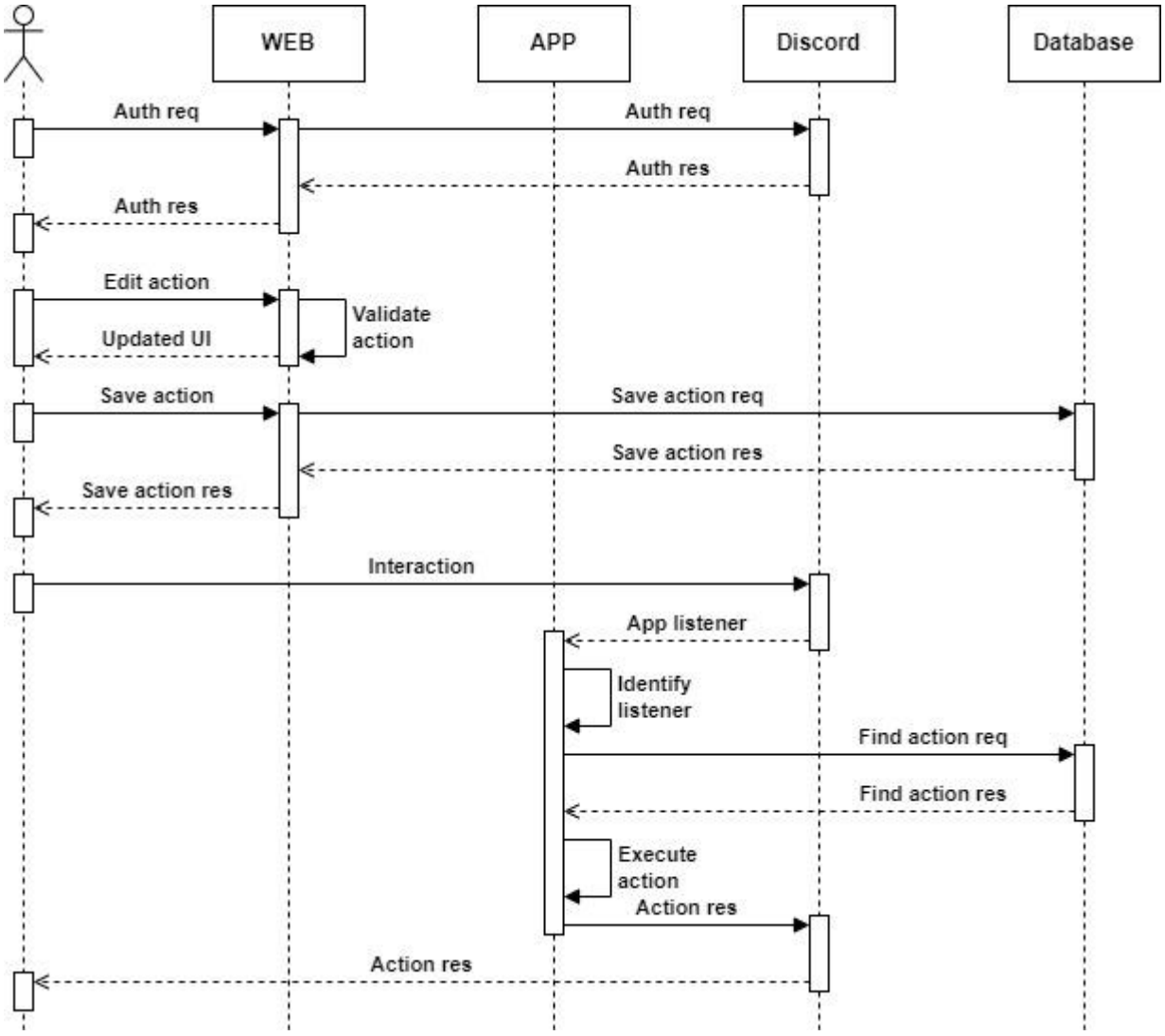

Рисунок 2.1. – UML-діаграма послідовності.

В даному випадку існує п’ять окремих акторів, які приймають участь у роботі системи, а саме: користувач, інтерфейс (WEB), виконавчий модуль, система Discord та база даних, що є комбінацією власної БД та API самого Discord.

В контексті інформаційної технології користувач здатен взаємодіяти напряму лише із WEB-інтерфейсом програми та додатком Discord:

- Реєстрація/авторизація;
- Створення та редагування власного модулю автоматизації;
- Збереження власного модулю автоматизації;
- Використання модулю автоматизації (може бути доступно всім, хто задовольняє встановлені, на етапі створення обробника, умови).

На кожен із таких випадків інтерфейс, передавши власні команди далі буде відповідати необхідними повідомленнями. У випадку взаємодії із додатком Discord, шляхом виконання тригеру, користувач отримує результат виконання відповідного обробника, який був прив’язаний до цього тригера.

При отриманні команд від користувача WEB-інтерфейс по ланцюжку передає їх на конструктор, за винятком авторизації – тоді команда направляється напряму в базу даних. При редагуванні представленого в інтерфейсі обробника, відправляється команда на синхронізацію та первинну перевірку алгоритму до модуля конструктору. Залежно від отриманої відповіді виводиться відповідний результат. При виборі тригеру необхідно перевірити можливість його прив’язки, такий запит також відправляється до конструктора. При надходженні команди на збереження відправляється відповідний запит на конструктор і залежно від результату знову виводиться відповідне повідомлення.

Модуль конструктора виконує функцію перевірки, синхронізації та налагодження створюваних користувачем обробників, при цьому напряму із користувачем він не взаємодіє. При отриманні команд від WEB-інтерфейсу він проводить внутрішні процеси над даними і формує відповідну відповідь. При надходженні запиту на збереження необхідно перевірити валідність отриманого алгоритму, для цього конструктор передає команду до модуля Discord, імітуючи виконання користувачем тригеру і звірює отримані результати із бажаними. Далі приймається рішення про збереження обробника і відправляється відповідний запит до бази даних. При отриманні відповіді вона передається назад до WEB-інтерфейсу.

Модуль додатку Discord, окрім уже описаних процесів, буде взаємодіяти із користувачем. Раніше він отримував імітацію виконання тригеру та виконував алгоритм обробника для конструктора, а цього разу він отримуватиме реальну інформацію про виконання користувачем тригеру та виконуватиме прив’язаний до нього обробник, відображаючи відповідний результат у додатку Discord.

2.2. Проектування та моделювання інформаційної системи

Для досягнення найкращого результату та максимального спрощення і скорочення процесу розробки, потрібно правильно та детально спроектувати систему, враховуючи наявність всіх модулів, шляхи взаємодії між ними та інші важливі аспекти. Використання сучасних методик моделювання, таких як IDEF0, IDEF3, та UML, дозволяє створити чітку структуру та зрозумілу архітектуру системи. Також для подальшого розвитку знадобиться базове представлення інтерфейсу основних частин WEB-застосунку та розуміння необхідності збереження відповідної інформації у базу даних.

Важливість правильного проектування:

– Чітка візуалізація процесів: Використання методик моделювання дозволяє розробникам та зацікавленим сторонам отримати чітке уявлення про процеси, що відбуваються в системі. Це спрощує комунікацію між членами команди та забезпечує зрозумілість архітектури;

– Визначення ролей та взаємодій: Методи моделювання, такі як UML (Unified Modeling Language), дозволяють визначити ролі та взаємодії між різними компонентами системи. Це допомагає забезпечити узгодженість та злагоджену роботу всіх модулів;

– Оптимізація процесів: Методики, такі як IDEF0 та IDEF3, допомагають ідентифікувати та оптимізувати бізнес-процеси. Це сприяє підвищенню ефективності системи та зменшенню витрат часу та ресурсів на виконання завдань;

– Забезпечення гнучкості та масштабованості: Правильне моделювання дозволяє створити гнучку та масштабовану архітектуру, що може бути легко адаптована до змінних вимог та збільшення обсягів даних. Це особливо важливо для великих систем, які постійно розвиваються.

Для розуміння процесу використання інформаційної технології автоматизації подій на платформі Discord розглянемо основний етап її роботи – створення модулю/блоку автоматизації. Необхідні вхідні дані, методи управління, механізми виконання та результат зображено у вигляді діаграми типу IDEF0, на рисунку 2.2.

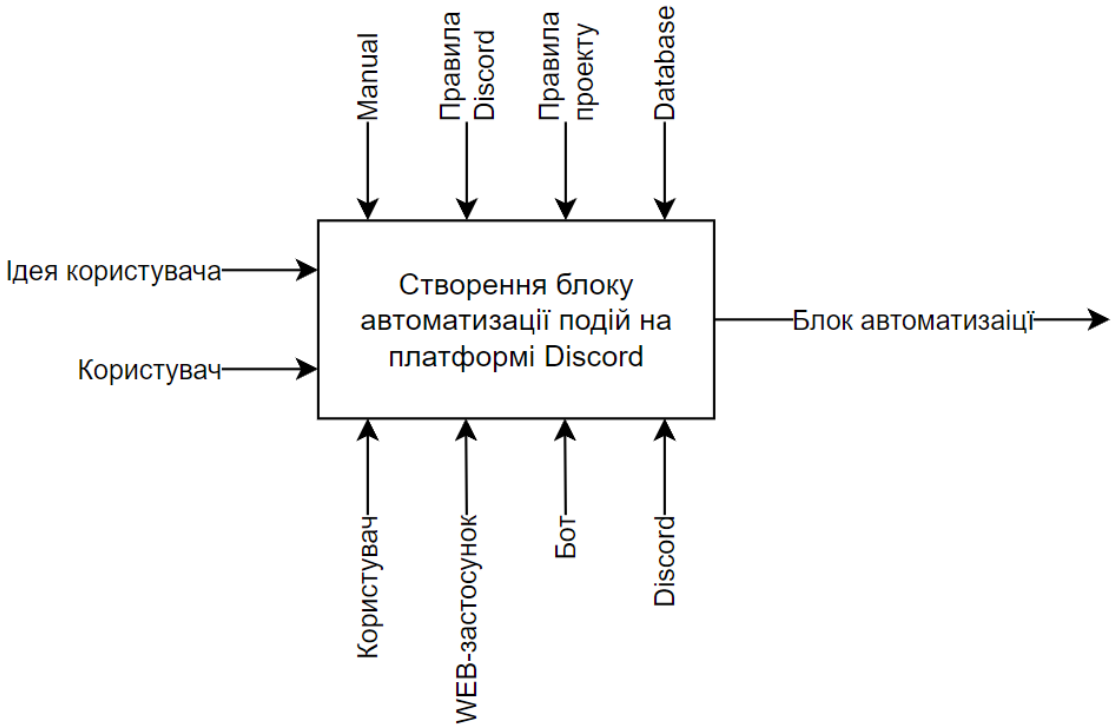

Рисунок 2.2. – IDEF0 діаграма процесу створення модулю/блоку автоматизації подій для платформи Discord.

Конструкційний модуль інформаційної технології (далі «конструктор») приймає на вхід ідею користувача або його задумку про бажаний результат роботи цільового модуля та самого користувача. Процес створення виконується самим користувачем із залученням окремих елементів програми, таких як сам конструктор; WEB-застосунок, в якому цей конструктор відкрито; додаток або WEB-додаток Discord; та сам виконавчий модуль системи – її бот. Керується процес правилами спільноти Discord; правилами проекту; базою даних (яка утримує інформацію про користувачів та створені ними модулі); та інструкцією користувача. Декомпозицію моделі IDEF0 приведено на рисунку 2.3.

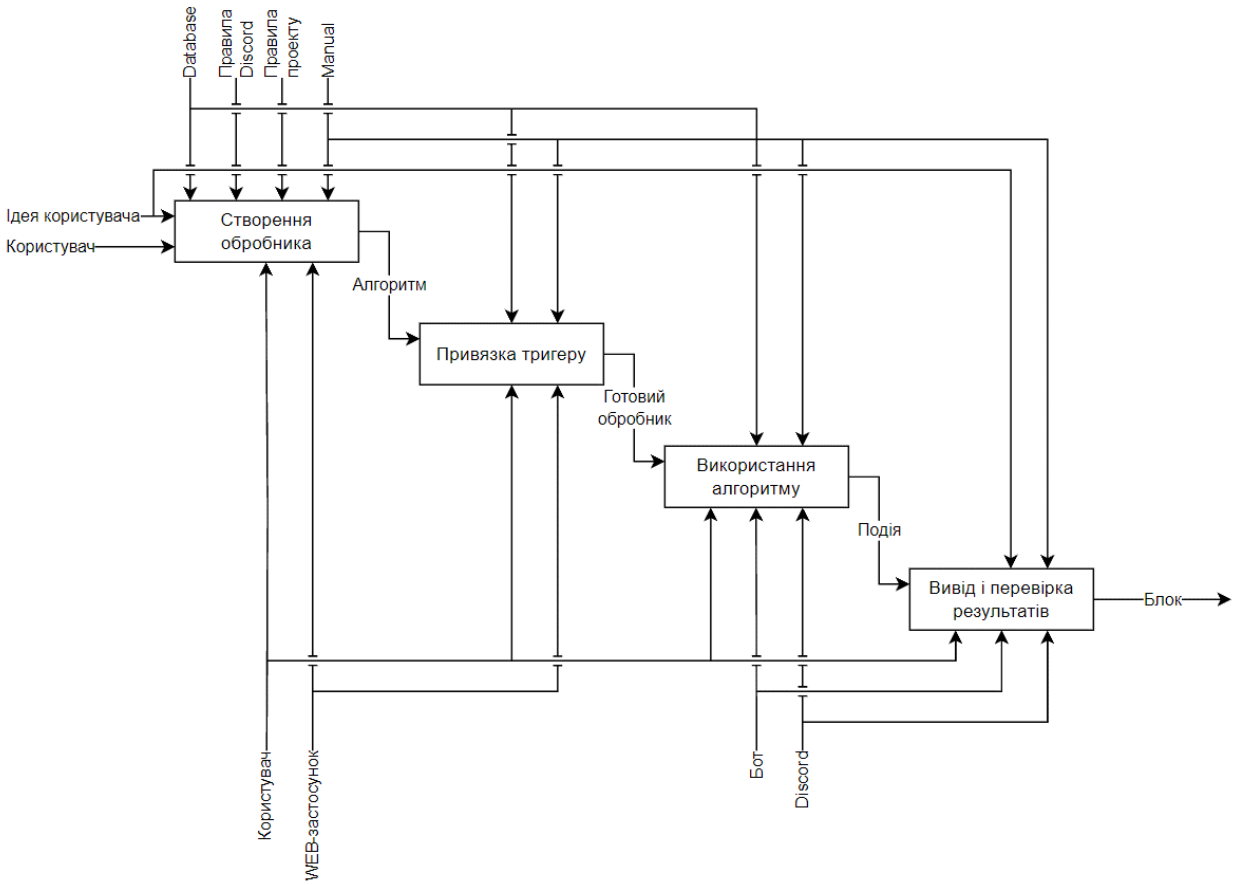

Рисунок 2.3. – Декомпозиція IDEF0 моделі «створення модулю/блоку автоматизації подій для платформи Discord».

На схемі декомпозиції першого рівня більш детально зображено процес створення готового модулю автоматизації. На першому кроці, користувач, керуючись власною ідеєю та правилами проекту, створює алгоритм/обробник, здатний задовольнити його потреби. Процес виконується у графічному інтерфейсі WEB-застосунку. На другому кроці, із випадаючого меню користувач повинен обрати тригер – умову, за якої буде спрацьовувати його алгоритм. На наступному кроці користувач повинен задіяти свій алгоритм, даючи боту, в додатку Discord, команду на його виконання. Останнім кроком буде – звірити отриманий результат із бажаним, тим який користувач запланував у своїй ідеї. Для більш детального розгляду найгроміздкішої функції на рисунку 2.4. наведено декомпозицію другого рівня для «Створення обробника».

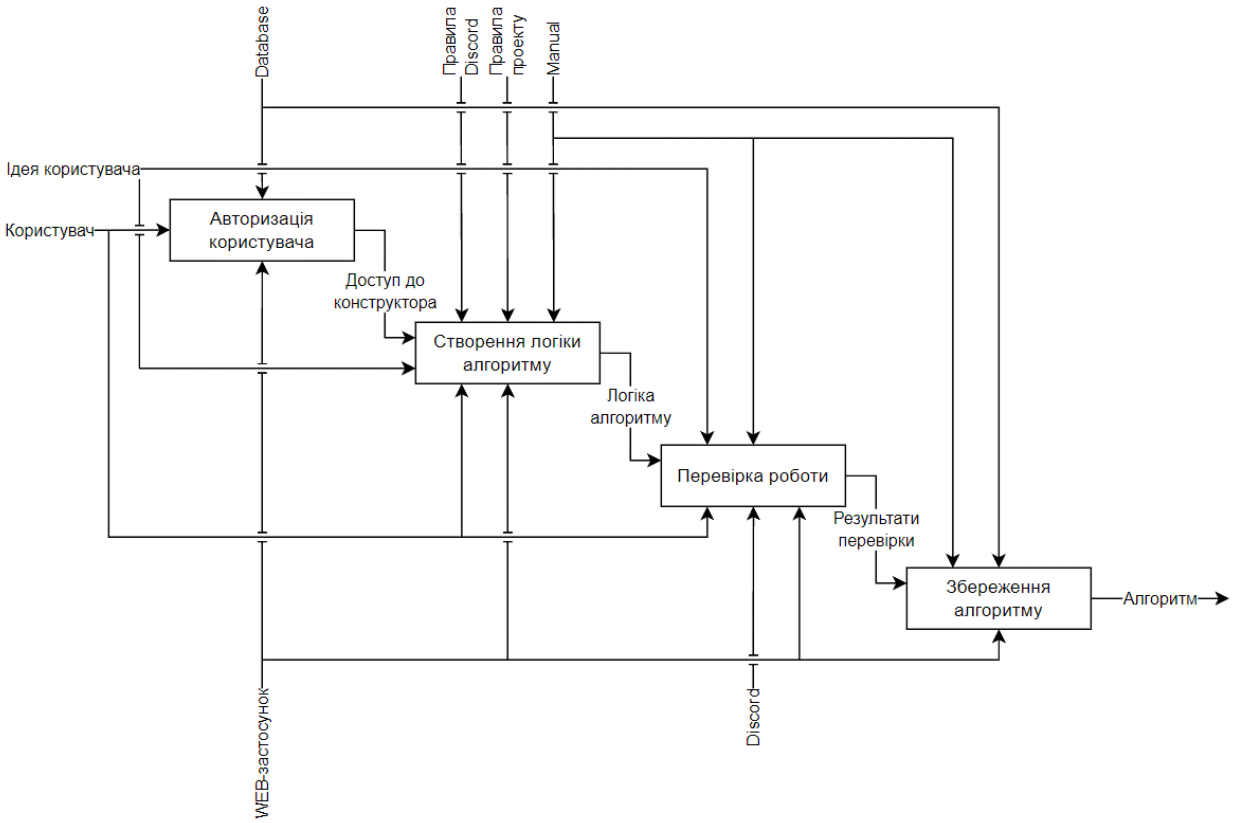

Рисунок 2.4. Декомпозиція IDEF0 моделі «Створення обробника».

Більш детальний процес створення обробника, зображений на схемі, описує що: на першому кроці користувачу необхідно авторизуватися в системі, тільки після цього йому буде надано доступ до конструктора; на другому кроці користувач у вигляді блок-схеми або програмного коду описує бажаний алгоритм; після чого, на третьому кроці, проводить ізольований тест працездатності свого алгоритму; і лише після проведення тестування, яке показало працездатність алгоритму користувач може зберегти його в базу даних та рухатися далі.

Моделі IDEF-0 описують послідовності та взамодії користувача із системою, але для більш детального розуміння всіх неочевидних аспектів роботи програми краще розглянути сформовані, для інформаційної технології, моделі типу IDEF3, розподілені для різних, окремих елементів програми.

Взаємодія із такими системами завжди починається із авторизації. Послідовність дій для авторизації користувача уже давно всім відома і виглядає як: користувач входить на сайт, у форму авторизації/реєстрації вводить необхідні дані, далі система перевіряє цю інформацію і або надає користувачеві доступ, або

повідомляє про невдачу, далі користувач сам вирішує – спробувати іще раз або покинути сайт. Схему для процесу авторизації наведено нижче, на рисунку 2.5.

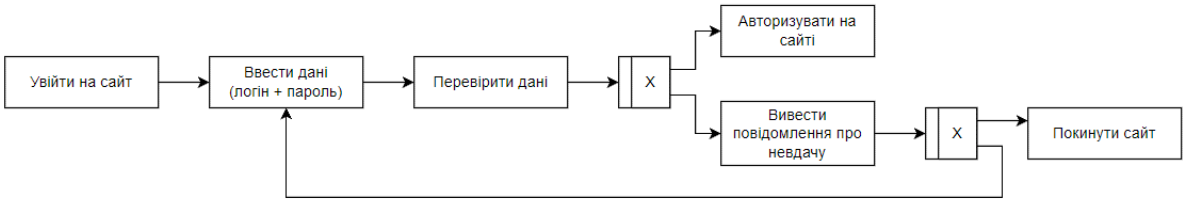

Рисунок 2.5. – IDEF3 модель авторизації користувача в системі.

Логічним продовженням використання інформаційної технології стане створення необхідного модулю автоматизації з допомогою конструктора. Складна, за першого погляду, модель описує процес створення обробника із використання конструктора алгоритмів на сайті і зображена на рисунку 2.6.

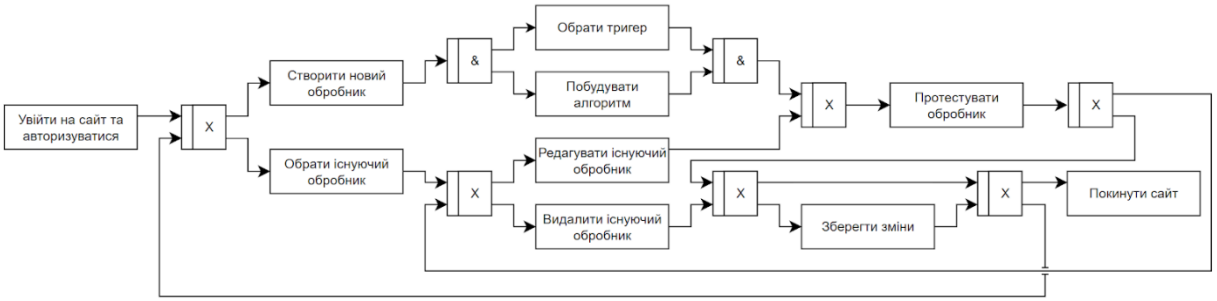

Рисунок 2.6. – IDEF3 модель процесу створення готового обробника для автоматизації події.

Як і в будь-якому WEB-застосунку взаємодія починається із авторизації користувача. Далі у користувача є вибір або обрати уже створений обробник, або створити новий. Створюючи новий обробник користувач зобов’язаний побудувати алгоритм його роботи та прив’язати до нього тригер. Обравши готовий обробник користувач може або редагувати, або видалити його. Видаляючи обробник користувачеві залишиться або повернутися на «головну», або завершити роботу. При редагуванні готового обробника, користувач «поєднує» потік із «створенням обробника» і далі виконує спільні роботи, а саме – тестування своєї розробки і

подальші внесення правок або збереження змін і завершення роботи із обраним обробником.

Після цього останнім етапом взаємодії користувача із створеним обробником буде – його використання. За умови, що усі попередні дії користувач виконав вірно, для завершення роботи із системою створення модуля автоматизації йому залишається лише використовувати цей модуль за призначенням. Для цього йому необхідно увійти в застосунок Discord, будь то настільна, мобільна чи WEB версія додатку, авторизуватися в ньому та виконати, встановлений на етапі побудови обробника, тригер, для отримання відповідного результату виконання, створеного раніше, алгоритму. Цей, досить простий, процес зображений на рисунку 2.7.

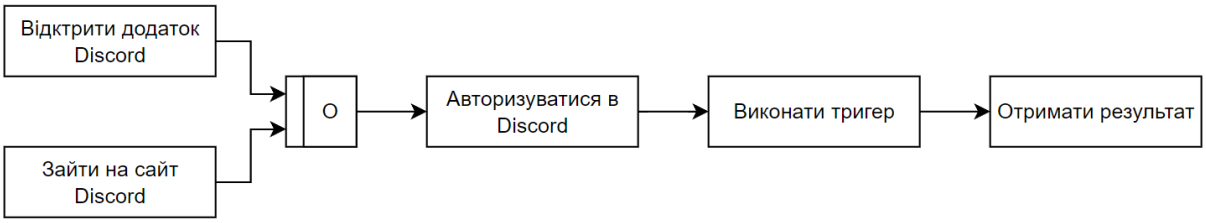

Рисунок 2.7. – IDEF3 модель використання користувачем створеного модуля автоматизації подій.

Маючи можливість виконувати створені модулі, користувачу варто також знати і ті етапи, які проходить уже готовий обробник в процесі його використання. Від моменту створення до видалення обробник проходить лише декілька етапів. Вхідною точкою для нього є момент створення, коли він переходить до стану існування. Наступним етапом є очікування тригері, після спрацювання якого виконується визначений обробником алгоритм і або циклічне повторення процесі, або вихідна точка діаграми – знищення чи видалення обробника. Рисунок 2.8 демонструє послідовність станів обробника.

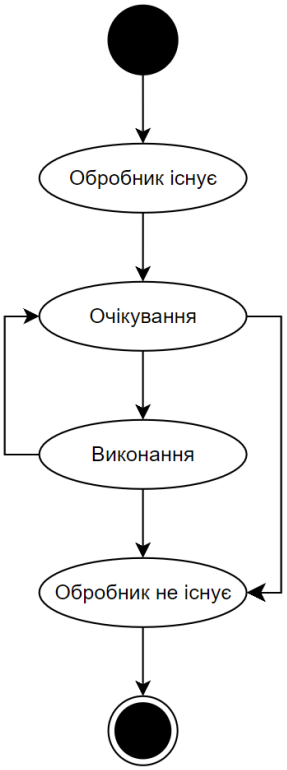

Рисунок 2.8. – UML-діаграма станів створеного модуля автоматизації.

Висновки до другого розділу

Під час моделювання інформаційного технологічного модуля було проведено більш детальний аналіз, який дозволив визначити основні функціональні вимоги до системи та способи реалізації окремих модулів. Створено структуровану та зручну для розуміння модель системи, яка відображає всі аспекти її функціонування та взаємозв'язки між її складовими частинами. Низка UML діаграм дозволяють візуально побачити та оцінити технічну складову системи, її взаємозв’язки та функції. Ці діаграми включають діаграми класів, діаграми випадків використання, діаграми активності та діаграми послідовностей, що разом дають комплексне уявлення про внутрішню архітектуру та поведінку системи.

Такий підхід до моделювання дозволив не лише чітко визначити необхідні функції системи, способи їх виконання, ієрархію побудови модулів, але й покращити базове представлення всієї системи в цілому. Це сприяє більш ефективній координації серед команди розробників, полегшує виявлення та

виправлення потенційних проблем на ранніх етапах розробки, а також забезпечує гнучкість у майбутньому розширенні та модернізації системи.

Маючи чітку та систематизовану модель системи, процес її створення та практичного використання значно спрощується, як для розробників, так і для кінцевих користувачів. Розробники отримують інструменти для швидкої інтеграції нових функцій та виправлення помилок, тоді як користувачі зможуть легко освоїти нову систему завдяки її логічній та інтуїтивно зрозумілій структурі.

РОЗДІЛ 3. СТВОРЕННЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ ПОДІЙ

3.1. Створення основних модулів системи та тестування результатів

Для забезпечення стабільності, простоти розробки та безпечності додатку варто правильно обрати стек технологій перед початком створення, адже саме сучасний підхід допомагає більшості інформаційних технологій із постійним зростанням потреб користувачів залишатися зручними, масштабованими та «реактивними». Використання найсучаснішого стеку доступного для створення саме запропонованої системи, а саме: React, Next.js, TypeScript, Discord.js та MongoDB обумовлене потужністю окремо визначених компонентів та їх комбінації в цілому, гнучкістю, та можливістю швидкої інтеграції, що дозволяє створювати унікальні та високоефективні рішення для користувачів.

– TypeScript [18] – [19] забезпечить надійність, безпеку та збоєстійкість коду, допомагаючи уникнути помилок та покращити структуру програми. Він також полегшить та прискорить процес розробки і, в перспективі майбутнього розвитку, надасть можливість швидкого розуміння та управління кодом для всього розробницького колективу;

– React [20], будучи найпопулярнішим Front-end фреймворком дозволить створити зручний та швидкий користувацький інтерфейс для панелі керування ботом, конструктора обробників та інших важливих частин системи. Це дозволить користувачам легко та інтуїтивно взаємодіяти з усіма функціями;

– Next.js [21] допоможе оптимізувати продуктивність веб-додатка за рахунок серверного рендерингу [22], що є важливим для реактивності та швидкості роботи панелі керування, конструктора обробників та інших функцій;

– Discord.js [23], являючись фактично єдиною JS бібліотекою для взаємодії з Discord надає зручний інтерфейс для взаємодії з API Discord без прямих запитів, спрощуючи процес розробки самої системи та пришвидшуючи подальшу взаємодію між ботом та Discord;

– MongoDB [24], являється однією із найпопулярніших NoSQL баз даних та дозволить зручно зберігати та управляти даними системи, такими як конфігураційні налаштування, дані користувачів, логи подій, налаштування безпеки подій та інші конфігураційні дані. Сформовану схему єдиного власного комплексного модулю бази даних можна побачити на рисунку А1.

Розробка будь-якої системи починається із створення репозиторію та організації файлового простору. Відповідна структура файлів забезпечує логічну сегментацію коду, що спрощує навігацію, читання та подальшу підтримку проекту. Чітка організація дозволяє розробникам легко знаходити необхідні файли, розуміти залежності між компонентами та модифікувати окремі частини без ризику порушення цілісності всього проекту. У контексті створення проекту на основі таких технологій, як React, Next.js, TypeScript та Node.js, важливо структурувати файли за функціональними модулями, що включає окремі директорії для компонентів, утиліт, сторінок, API, конфігураційних файлів та тестів.

Після створення базового робочого файлового простору необхідно налаштувати основні модулі системи, а саме – API, Store та Design провайдерів. Результати налаштування API за методиками Next.js для переадресація на паралельний сервер виконавчого модулю наведено на рисунку 3.1, та невеликий helper для приведення запиту в потрібний формат на рисунку 3.2. Сам виконавчий API роут наведено на рисунках А2 та А3

```
1  import { nextToAxiosRequest } from '@utils/helpers'
2  import { NextApiRequest, NextApiResponse } from 'next'
3
4  export default async function handler(req: NextApiRequest, res: NextApiResponse) {
5    try {
6      return res.status(200).json(await nextToAxiosRequest(req))
7    } catch (error: any) {
8      return res.status(error.response.status).json(error.data)
9    }
10 }
11
```

Рисунок 3.1 – Приклад API роуту, для переадресації запитів із RTK Query до виконавчого модулю.

```ts
import axios from 'axios'
import { NextApiRequest } from 'next'

export const nextToAxiosRequest = async (req: NextApiRequest) => {
  const { method, query, body, headers, url } = req

  const axiosUrl = `http://${headers.host?.replace(/:\d+$/, '')}:${
    process.env.CLIENT_PORT ?? 3001
  }${url}`

  const axiosConfig = {
    method,
    url: axiosUrl,
    query,
    data: body,
  }

  return (await axios(axiosConfig)).data
}

```

Рисунок 3.2 – Код хелпера nextToAxiosRequest

Оскільки для взаємодії між Frontend та Backend частинами застосунку було обрано пакет redux-toolkit, а саме його Query, то необхідно налаштувати точку входу в застосунок для використання цього модулю. Також, в це й же момент необхідно налаштувати модуль авторизації next-auth, який буде використовуватися у зв’язці із Discord Auth Provider. Конфігурацію застосунку для використання описаних модулів відображено на рисунку 3.3.

```ts
import '@styles/global.css'
import '@styles/reset.css'
import '@styles/variables.css'

import { store } from '@store/index'
import { ConfigProvider } from 'antd'
import { SessionProvider } from 'next-auth/react'
import { Provider as StoreProvider } from 'react-redux'

const App: React.FC<any> = ({ Component, pageProps: { session, ...pageProps } }) => {
  return (
    <StoreProvider store={store}>
      <SessionProvider session={session}>
        <ConfigProvider>
          <Component {...pageProps} />
        </ConfigProvider>
      </SessionProvider>
    </StoreProvider>
  )
}

export default App

```

Рисунок 3.3 – Точка входу в WEB-застосунок

Варто також звернути увагу на таку особливість системи, при якій модулі системи є незалежними і запускаються в окремих процесах одного домену, але на різних портах. Таким чином програму розділено на дві частини. Точку входу в першу частину – WEB-застосунок конфігурації вже наведено на рисунку 3.3. А точка входу в виконавчий модуль реалізована у вигляді класу SkynetClient, який і запускає необхідні модулі виконавчого модулю. Створення екземпляру такого класу наведено на рисунку 3.4.

```
require('tsconfig-paths').register()
require('dotenv').config({ path: process.env.NODE_ENV === 'production' ? '.env' : '.env.local' })

import { GatewayIntentBits } from 'discord.js'
import { SkynetClient } from './client'

export const client = new SkynetClient({
  intents: [
    GatewayIntentBits.Guilds,
    GatewayIntentBits.GuildVoiceStates,
    GatewayIntentBits.GuildMessages,
    GatewayIntentBits.MessageContent,
  ],
})

```

Рисунок 3.4 – Точка входу у виконавчий модуль

3.2. Приклади практичного використання системи

Основну взаємодію користувач проводитиме із комплексною та динамічною формою створення модулю автоматизації, яка реалізована на окремій сторінці у вигляді динамічної послідовності окремих блоків, призначених для визначення трьох різних типів взаємодії, а саме:

- When: Модуль який визначатиме за якої події потрібно виконати даний обробник. Являється точною входження в процес виконання алгоритму обробника.
- If: Модуль, який визначатиме умову, за якої отриману в блоці When подію можна класифікувати та визначити серед інших. Існує для уникнення перехресного використання неправильно створених обробників одного типу події. Наприкад, при створенні обробників для двох різних кнопок, необхідно

вказати в блок If деяку кастомну назву кнопки, за якої вона і буде ідентифікована системою.

- Then: Найпростіший для розуміння модуль, який просто виконує описану в ньому дію, будь це відповідь на взаємодію, створення повідомлення, оновлення стану користувача, його ролей та іншого.

Візуальне представлення елементів такої форми наведено на рисунках 3.2.1 – 3.2.2, на прикладі створення обробника команди «/hello», який би відповідав привітанням при використанні команди. Цілісне представлення форми в двох режимах, створення та редагування можна побачити на рисунках В3 та В4 відповідно.

Рисунок 3.2.1 – Алгоритм команди «/hello» (тригер)

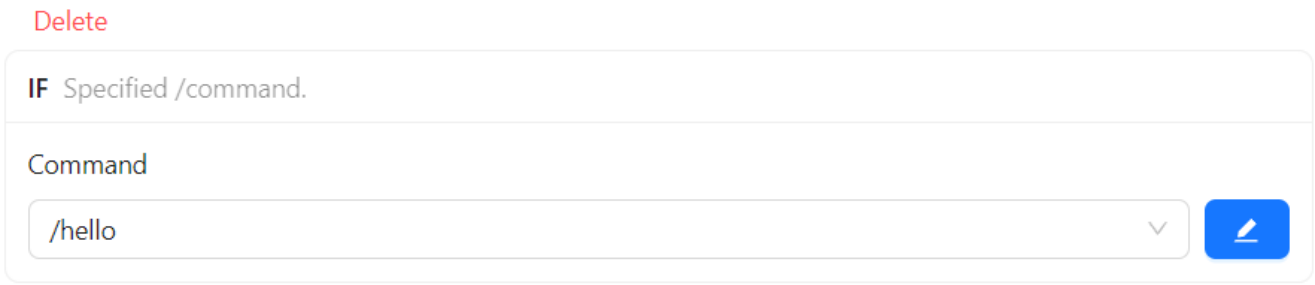

Рисунок 3.2.2 – Алгоритм команди «/hello» (умова)

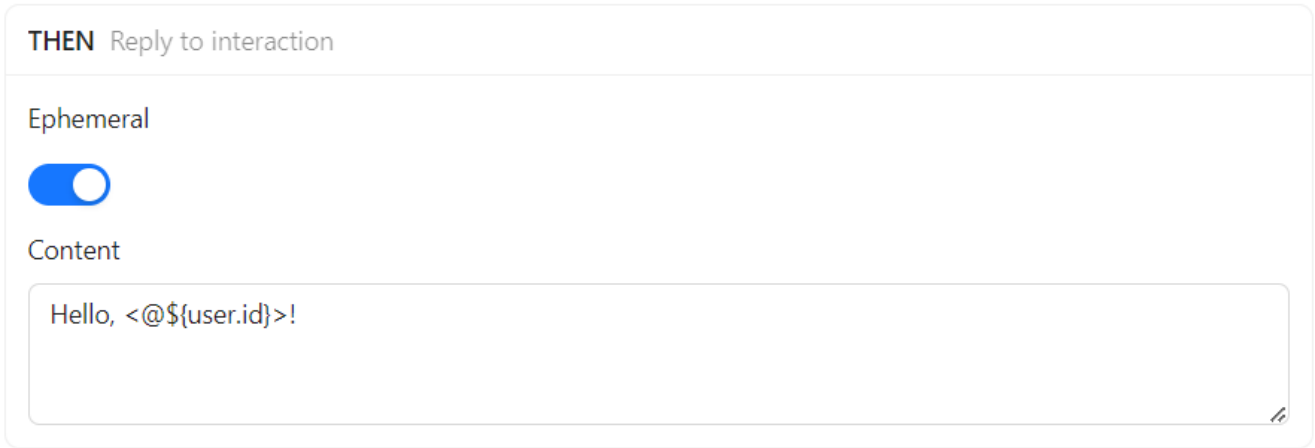

Рисунок 3.2.3 – Алгоритм команди «/hello» (дія)

Такий короткий обробник буде опрацьовано наступним чином: при використанні користувачем команди Discord передасть цю інформацію у виконавчий модуль, який дістане із бази даних усі обробники із типом «команда», які існують у отриманому контексті, після чого буде перевіряти кожен із них на виконання усіх блоків If і той обробник, умови якого будуть задоволені виконається. В даному випадку при використанні користувачем команди «/hello» система відповість повідомленням «Hello, <user_mention>!», де user_mention – це візуалізоване на стороні Discord згадування користувача, із посиланням на його профіль.

Виглядатиме використання такого алгоритму наступним чином: користувач в додатку Discord виконує команду, як зображено на рисунку 3.3 та отримує відповідь, як зображено на рисунку 3.4.

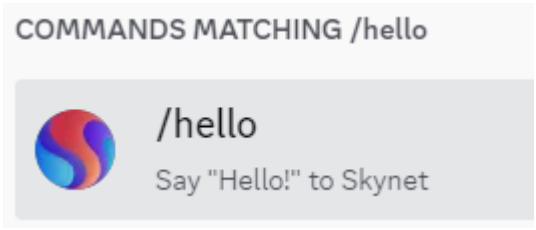

Рисунок 3.3 – Використання команди «/hello»

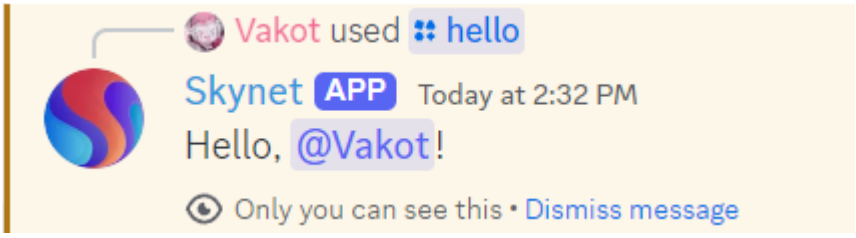

Рисунок 3.4 – Результат виконання команди «/hello»

Іншою складовою системи, із якою часто взаємодіятиме користувач буде загальна навігація по сторінках системи та допоміжні модулі, такі як «Перегляд слухачів подій», «Зареєстровані команди», «Сховище», «Історія» і подібне. Навігація реалізовується відповідно до серверу користувача, а саме за принципом того, що кожен сервер матиме свої модулі, своє сховище та історію. Навігація реалізована у вигляді Sidebar та сторінки навігації із відповідними картками. Ці елементи можна побачити на рисунках В1 та В2.

Для використання будь-якого застосунку потрібно володіти системою здатною виконати цей самий застосунок. Можливість системи запустити та справно виконувати увесь спектр можливостей застосунку називається системними вимогами, в яких вказані мінімально необхідні можливості або складові системи.

Системні вимоги до використання запропонованої інформаційної технології автоматизації подій на платформі Discord буде залежати від двох факторів, оскільки поділена на два модулі. Модуль конструктора – відділений від виконавчого модулю, WEB-інтерфейс побудови логіки виконавчого алгоритму, при створенні обробника подій. Його системні вимоги будуть визначені стеком розробки та вимогами для виконання створеного WEB-застосунку. На етапі планування, стек розробки виглядає наступним чином: React, Redux, Redux-Toolkit, SCSS. Оскільки більшість сучасних браузерів підтримують вищезазначені технології розробки WEB-додатків, тому можна орієнтуватися на системні вимоги браузерів, наприклад Google Chrome. Системні вимоги Google Chrome для різних операційних системи наведено в таблиці 3.1.

Таблиця 3.1. – Системні вимоги до Google Chrome [25].

	Windows	MacOS	Linux
Операційна система	Windows 10+.	MacOS 10.13+.	Ubuntu 14.04+, Debian 8+, OpenSUSE 13.3+, Fedora Linux 24+.
Процесор	Процесор Intel Pentium 4 або більш пізні версії процесорів із підтримкою SSE3.		
Вільна місце на диску	350 Мб		
Об’єм оперативної пам’яті	512 Мб		

Другим важливим фактором для формування системних вимог являється виконавчий модуль, який працюватиме окремо від WEB-інтерфейсу. Він

отримуватиме команди від додатку Discord та відправлятиме відповіді також до додатку Discord. Роблячи висновок того, що для користування модулем виконання користувачу необхідно запустити додаток Discord в будь-якому вигляді: WEB-додаток, мобільну чи настільну версію. Системні вимоги різних версій додатку наведено в таблиці 3.2.

Таблиця 3.2. – Системні вимоги до додатку Discord.

	Desktop	Mobile	Browser
Операційна системи	Windows 7+, MacOS 10.11+, Linux Ubuntu 18.04+, Linux Debian 10+, Linux openSUSE 15.2+, Linux Fedora 32+.	Android 6+, iOS 12+.	Google Chrome 80+, Firefox 80+, Microsoft Edge 17+, Safari 11+.
Процесор	Будь-який процесор, здатний працювати на частоті більшій ніж 1.2 ГГц та із підтримкою необхідної операційної системи.		
Вільне місце на диску	167 Мб		
Об’єм оперативної пам’яті	256 Мб		

Висновки до третього розділу

Процес створення системи автоматизації подій на платформі Discord продемонстрував ефективність системного підходу, що включає шаблонне створення та перевикористання коду. Такий підхід дозволив не лише зменшити час розробки, але й покращити якість та надійність коду. Використання сучасних технологій та фреймворків, таких як React, Next.js, TypeScript, Discord.js та MongoDB, забезпечило створення гнучкої та масштабованої системи, яка легко адаптується до потреб користувачів.

У розділі також наведено приклади використання системи, зокрема процес створення обробника події команди та його подальше використання. Це демонструє практичну цінність розробленої системи, яка дозволяє користувачам швидко та ефективно налаштовувати процеси автоматизації, не потребуючи глибоких знань у програмуванні. Такий підхід сприяє більш широкому впровадженню автоматизації у спільнотах Discord, підвищуючи їх ефективність та комфорт користувачів цих спільнот.

Завдяки структурованому та методичному підходу, що включає шаблони та перевикористання коду, було створено потужну та зручну у використанні систему, яка значно полегшує процес розробки та впровадження модулів автоматизації. Це забезпечує як розробникам, так і користувачам впевненість у стабільності та ефективності системи.

ВИСНОВКИ

У результаті проведеного в межах кваліфікаційної роботи дослідження було виявлено значну вагомість теми автоматизації подій на платформі Discord у сучасних умовах. За рахунок швидкого розвитку інформаційних технологій та зростаючої складності онлайн-середовищ, автоматизація виявляється ключовим елементом для забезпечення продуктивної та зручної взаємодії у спільнотах, особливо в межах такої стрімко розвиваючоїся платформи, як Discord. Платформа Discord, що надає можливості для розробки інтегрованих систем та автоматизації подій, є важливим інструментом для створення та ефективного управління онлайн-спільнотами та забезпечення комфортного взаємодії її учасників. Безліч компаній користуються послугами платформи, починаючи від шкільних гуртків, закінчуючи великим бізнесом.

Навіть така ідеальна, на перший погляд, платформі, як Discord має свої переваги, та недоліки. Сучасний підхід до автоматизації, а також акцент на актуальності розвитку інноваційних рішень для оптимального управління спільнотами у цифровому просторі не здатен забезпечити достатнього рівня продуктивності та комфорту користування. Існуючі наразі системи мають кардинально різні підходи до реалізації модулів автоматизації в межах платформи, що, хоч і виконує свою функцію, але значно ускладнює процес конфігурації та первинного налаштування для користувачів.

Для створення власного, продуманого та систематизованого рішення, здатного задовольнити потреби більшості користувачів разом була обрана концепція all-in-one (все в одному) і спроектовано відповідну систему. Основною перевагою такого рішення стала можливість користувачам самостійно створювати алгоритми автоматизації визначених ними ж подій. В такому випадку, маючи дружній та систематизований інтерфейс і підхід до створення модулів автоматизації користувачі зможуть швидко адаптувати систему для досягнення максимальної продуктивності і важливій їм сфері шляхом її автоматизації.

Створивши та продумавши модель системи її було реалізовано на прикладі найбільш cross-platform рішення, а саме WEB-застосунку. В такому випадку

проблеми сумісносту будуть практично відсутні. Для розробки було обрано сучасний та ефективний стек технологій, компоненти якого гармонійно доповнюють один-одного. Підхід до розробки також був обраний системний та шаблонний, за рахунок чого вдалося досягти максимальної швидкості та простоти розробки із можливостями перевикористання готових модулів та коду. Також, було проведено практичне тестування системи і доведено її працездатність в умовах реальної ситуації на прикладі простих модулів автоматизації подій в межах платформи Discord.

У цілому, протягом виконання роботи було детально розкрито необхідність та вагомість розвитку інформаційних технологій для забезпечення ефективного управління та функціонування онлайн-спільнот. Інноваційні підходи до автоматизації, системний підхід до розробки, а також практична віддача розробленої системи свідчать про потужний потенціал у галузі. Продовження досліджень та вдосконалення інформаційних технологій в цьому напрямку є ключовим для подальшого покращення якості та ефективності управління онлайн-спільнотами, особливо на прикладі автоматизації подій в межах платформи Discord.

Із системою, створеною в результаті розробки можна ознайомитися на GitHub за посиланням: https://github.com/vakot/Skynet

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт застосунку Discord, URL: https://cutt.ly/IepIRSUa
2. Що таке Discord, URL: https://cutt.ly/lepIR2fT
3. Що таке Discord, URL: https://cutt.ly/UepITq2F
4. Що таке Discord Application, URL: https://cutt.ly/PepOpenN
5. Створити Discord Application, URL: https://cutt.ly/depITi9t
6. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2021. 5 с., URL: https://cutt.ly/hepICERL
7. Vance J. The Rise of Discord, University of Virginia Charlottesville, Virginia, Department of Computer Science, 2021. 12 с., URL: https://cutt.ly/yepIOaqI
8. Anderson M. Discord and the harbormen gaming community, comm 416, 2019. 16 с., URL: https://cutt.ly/1epICXsf
9. Статистичні дані Discord, URL: https://cutt.ly/PepITjzc
10. Документація по Discord, URL: https://cutt.ly/3epITbbG
11. Документація по API Discord, URL: https://cutt.ly/HepITEzr
12. Discord «Terms of Service», URL: https://cutt.ly/vepITOGh
13. Discord «Guidelines», URL: https://cutt.ly/SepIKnDx
14. Переваги та недоліки Discord, URL: https://cutt.ly/oepOoK25
15. Переваги та недоліки Discord Application, URL: https://cutt.ly/2epOo0fS
16. Підбір Discord Application, URL: https://cutt.ly/ZepITZ5j
17. Підбір Discord Application, URL: https://cutt.ly/vepIT0Ne
18. Про JavaScript, URL: https://developer.mozilla.org/ru/docs/Web/JavaScript
19. Про TypeScript, URL: https://www.typescriptlang.org
20. Про React.js, URL: https://react.dev
21. Про Next.js, URL: https://next.js.org
22. Переваги SSR перед Static Generation, URL: https://cutt.ly/tepOoIPq

Переваги і недоліки

23. Про Discord.js, URL: https://discord.js.org
24. Про MongoDB, URL: https://www.mongodb.com
25. Системні вимоги Google Chrome, URL: https://cutt.ly/wepOhsT2

26. Трегуб В.Г. Проектування систем автоматизації: Ліра-К, 2019. 344 с., URL: https://cutt.ly/aepIQtPF
27. Томашевський О.М., Цегелик Г.Г., Вітер М.Б., Дубук В.І. Інформаційні технології та моделювання бізнес процесів: Центр навчальної літератури, 2012. 296 с., URL: https://cutt.ly/ZepIWfaU
28. Пількевич І.А., Молодецька К.В., Сугоняк І.І., Лобанчикова Н.М. Основи побудови автоматизованих систем управління: навч. посібник. Житомир: ЖДУ ім. І. Франка, 2014. 226 с., URL: https://cutt.ly/zepIbK3I
29. Державний стандарт України. ДСТУ 3008:2015 «Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання». URL: https://cutt.ly/wzOjLTy
30. Державний стандарт України. ДСТУ 8302:2015 «Інформація та документація. Бібліографічне посилання. Загальні вимоги та правила складання» URL: https://cutt.ly/hzOjM1h

ДОДАТКИ

Додаток А – Код виконавчої програми

```
import { Guild, User } from 'discord.js'
import mongoose, { Schema } from 'mongoose'

import { IDocument } from '@bot/models/document'
import { SkynetEvents } from '@bot/models/event'

export interface ICondition {
  event: SkynetEvents
  type: string
  property: string
  // callback: (...args: any) => boolean
  value: any
}

export interface IAction {
  event: SkynetEvents
  type: string
  property: string
  // callback: (...args: any) => void
  value: any
}

export interface IAutomation extends IDocument {
  author?: User
  name?: string
  description?: string
  guild?: Guild['id']
  event: SkynetEvents
  conditions: ICondition[]
  actions: IAction[]
}

export const AutomationSchema: Schema = new Schema<IAutomation>({
  author: { type: String, required: false },
  name: { type: String, required: false },
  description: { type: String, required: false },
  guild: { type: String, required: false },
  event: { type: String, required: true },
  conditions: [{ type: Schema.Types.Mixed, required: false }],
  actions: [{ type: Schema.Types.Mixed, required: false }],
})

export const Automation =
  (mongoose.models.Automation as mongoose.Model<IAutomation>) ||
  mongoose.model<IAutomation>('automation', AutomationSchema)
```

Рисунок А1 – Модель бази даних Automation

```ts
import { Automation } from '@bot/models/automation'
import express from 'express'
import { FilterQuery } from 'mongoose'

const router = express.Router()

router.get('/automation', async (req, res) => {
  try {
    if (req.method !== 'GET') {
      return res.status(405).send('incompatible method')
    }

    const { ids: automationIds, guild: guildId, action: actionId, event } = req.query

    const filter: FilterQuery<typeof Automation> = {}

    if (automationIds) {
      filter._id = typeof automationIds === 'string' ? automationIds : { $in: automationIds }
    }

    if (guildId) {
      filter.$or = [
        { guild: typeof guildId === 'string' ? guildId : { $in: guildId } },
        { guild: { $exists: false } },
        { guild: null },
      ]
    }

    if (event) {
      filter.event = event
    }

    const automations = await Automation.find(filter)

    return res.status(200).json(automations)
  } catch (error) {
    return res.status(500).send(error)
  }
})

router.post('/automation', async (req, res) => {
  try {
    if (req.method !== 'POST') {
      return res.status(405).send('incompatible method')
    }

    if (!req.body) {
      return res.status(400).send('unresolved automation')
    }

    if (!req.body.guild) {
      return res.status(400).json('unresolved guild id')
    }

    const automation = await Automation.create(req.body)

    if (!automation) {
      return res.status(422).send('unable to create automation')
    }

    return res.status(200).json(automation)
  } catch (error) {
    return res.status(500).send(error)
  }
})

export default router

```

Рисунок А2 – API роут для обробки get та post запиту для automation

```
import { Automation } from '@bot/models/automation'
import express from 'express'

const router = express.Router()

router.get('/automation/:id', async (req, res) => {
  try {
    if (req.method !== 'GET') {
      return res.status(405).send('incompatible method')
    }

    const automation = await Automation.findById(req.para
ms.id)

    if (!automation) {
      return res.status(404).send('unknown automation')
    }

    return res.status(200).json(automation)
  } catch (error) {
    return res.status(500).send(error)
  }
})

router.patch('/automation/:id', async (req, res) => {
  try {
    if (req.method !== 'PATCH') {
      return res.status(405).send('incompatible method')
    }

    if (!req.body) {
      return res.status(400).json('unresolved automatio
n')
    }

    if (!req.body.guild) {
      return res.status(400).json('unresolved guild id')
    }

    const automation = await Automation.findByIdAndUpdate
(req.params.id, req.body)

    if (!automation) {
      return res.status(404).send('unknown automation')
    }

    return res.status(200).json(automation)
  } catch (error) {
    return res.status(500).send(error)
  }
})

export default router

```

Рисунок А3 – API роут для обробки get та patch запиту для automation/:id

Додаток Б – Код WEB-застосунку

```tsx
export interface EditDataFormProps extends EditFormProps {
  data: T['_id']
  onFinish?: (value?: T) => void
}

export const EditDataForm: React.FC<EditDataFormProps> = ({
  form: _form,
  data: dataId,
  onFinish,
  onFinishFailed,
  onAbort,
  showControls,
  ...props
}) => {
  const [form] = Form.useForm(_form)

  const { data, isLoading: isDataLoading } = useGetDataQuery(dataId, {
    skip: !dataId,
  })
  const [addData, { isLoading: isAddLoading }] = useAddDataMutation()
  const [editData, { isLoading: isEditLoading }] = useEditDataMutation()

  const isLoading = isDataLoading || isEditLoading || isAddLoading

  const handleFinish = async (fields: any) => {
    try {
      const response = data ? await editData({ id: data._id, ...fields }) : await addData(fields)

      if ('data' in response) {
        onFinish?.(response.data)
      } else if ('error' in response) {
        throw response.error
      }
    } catch (error: any) {
      onFinishFailed?.(error)
    }
  }

  const handleAbort = () => {
    form.resetFields()
    onAbort?.()
  }

  useEffect(() => {
    if (data && dataId) {
      form.setFieldsValue(data)
    } else {
      form.resetFields()
    }
  }, [form, dataId, data])

  return (
    <Form onFinish={handleFinish} form={form} layout="vertical" {...props}>
      {showControls && (
        <Flex justify="end" gap={8}>
          <Button type="default" onClick={handleAbort} disabled={isLoading}>
            Cancel
          </Button>
          <Button type="primary" onClick={form.submit} disabled={isLoading}>
            Save
          </Button>
        </Flex>
      )}
    </Form>
  )
}
```

Рисунок Б1 – Шаблон форми створення/редагування модуля програми

Додаток В – Інтерфейс WEB-застосунку

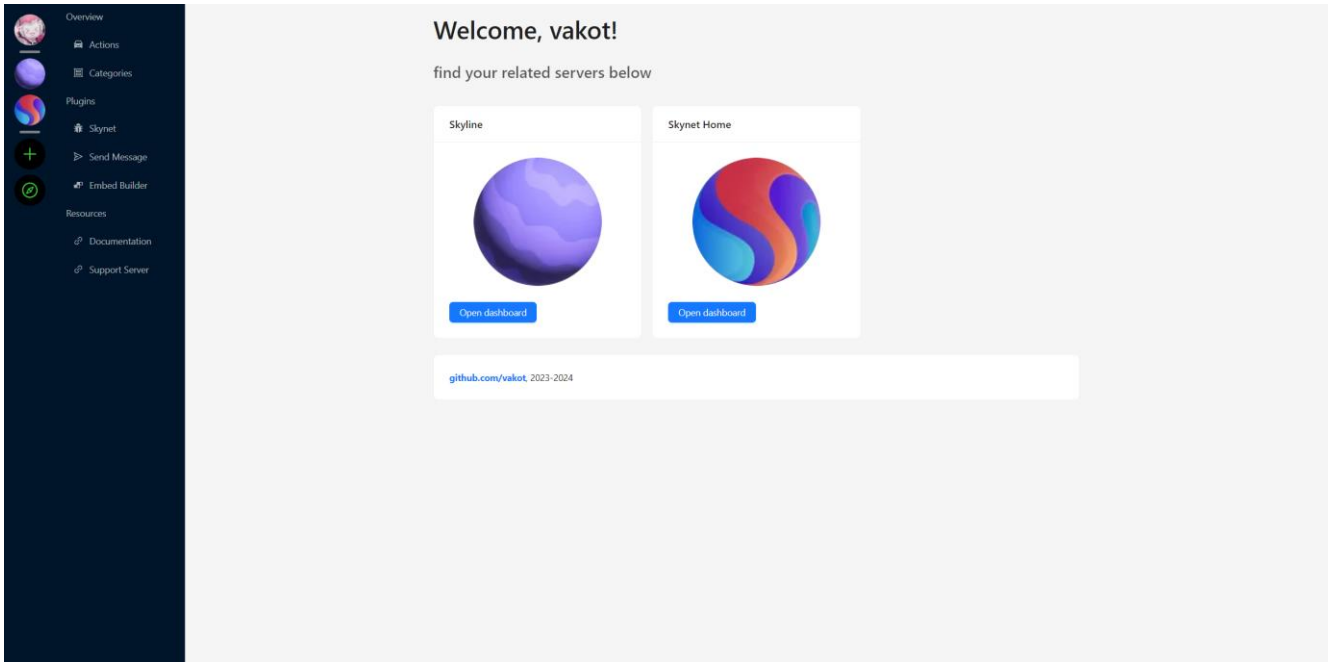

Рисунок В1 – Головна сторінка вибору сервера для конфігурації

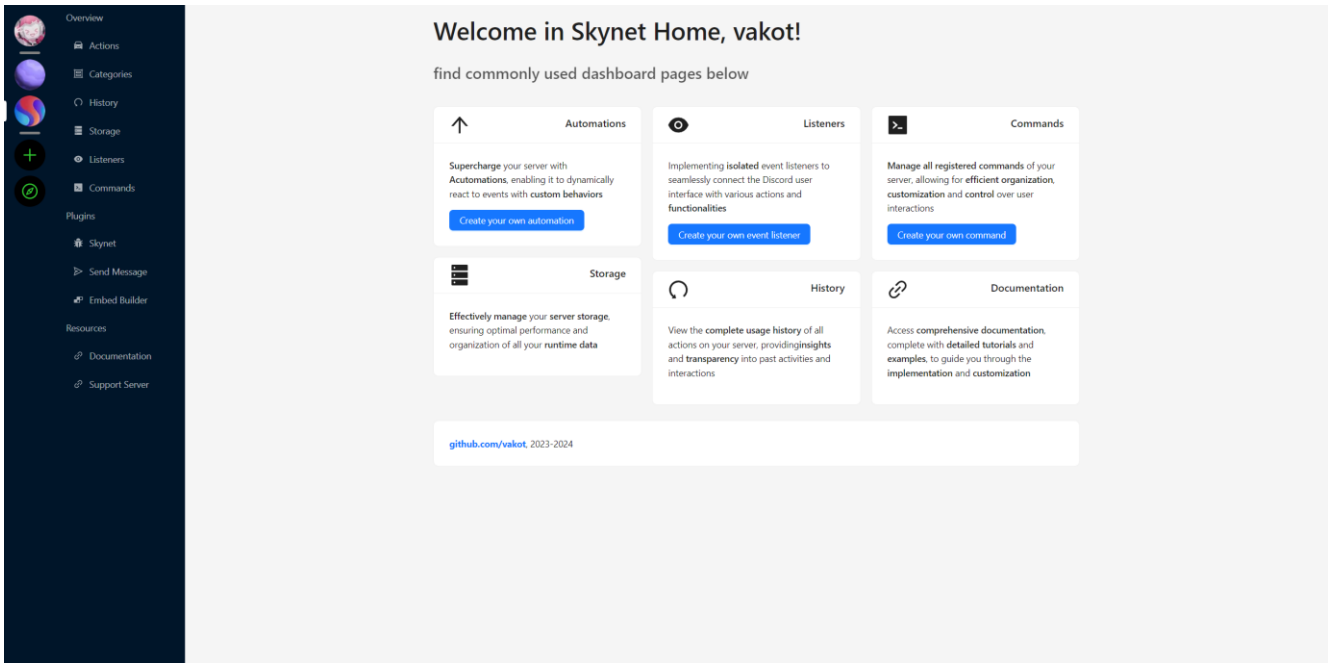

Рисунок В2 – Сторінка вибору серверного модуля для конфігурації

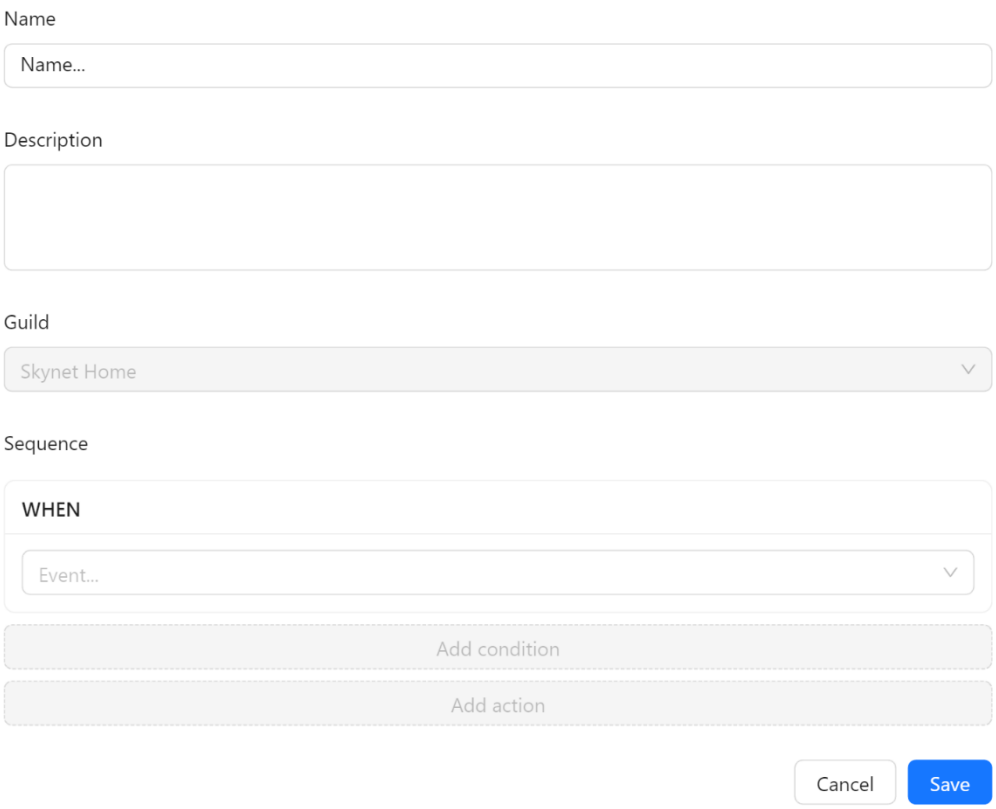

Рисунок В3 – Форма створення модуля автоматизації

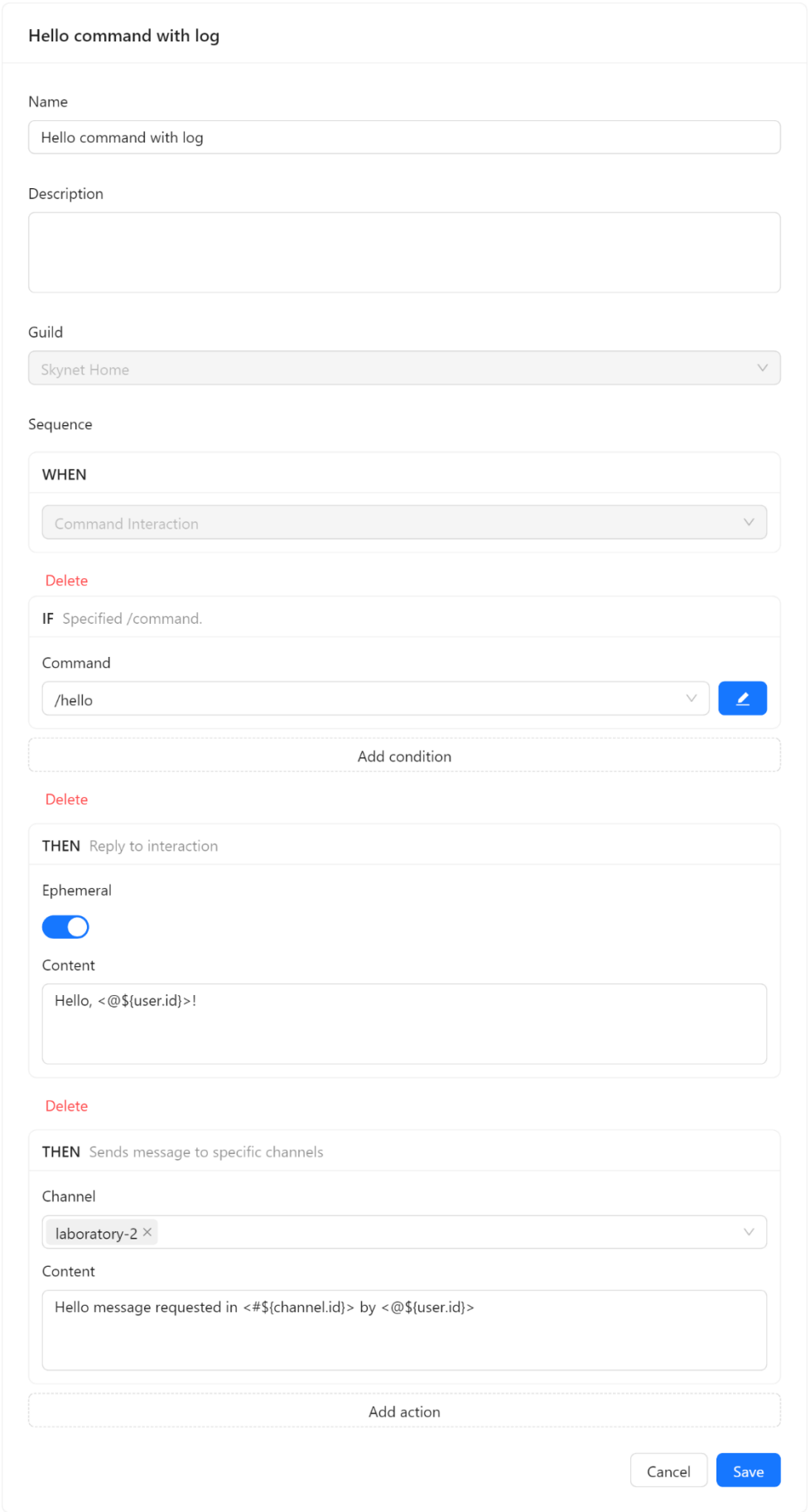

Рисунок В3 – Форма редагування модуля автоматизації