МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

Литвиненко О.Є.
"_____"_____2022 р.

ДИПЛОМНИЙ ПРОЄКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Універсальний чат-бот банківських послуг на платформі Node.js

Виконавець: Коноваленко Костянтин Юрійович

Керівник: Коба Олена Вікторівна

Нормоконтролер: Тупота Євгеній Вікторович

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»

(шифр, найменування)

Освітньо-професійна програма «Системне програмування»

Форма навчання денна

ЗАТВЕРДЖУЮ

Завідувач кафедри

Литвиненко О.Є.

«_____»_____2022 р.

ЗАВДАННЯ

на виконання дипломної роботи (проєкту)

Коноваленка Костянтина Юрійовича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проєкту) Універсальний чат-бот банківських послуг на платформі Node.js

затверджена наказом ректора від "16" квітня 2022 року № 648/ст.

2.Термін виконання проєкту (роботи): з 16.05.2022 до 19.06.2022

3. Вихідні дані до роботи (проєкту): мова програмування JavaScript, платформа для взаємодії з клієнтами Chatwoot, платформа Node-RED, Telegram бот, Viber бот, сфера підтримки клієнтів

4. Зміст пояснювальної записки: 1) Аналіз проблематики та інструментарій

2) Формування вимог до чат-боту

3) розробка універсального чат-боту для банківських послуг на платформі node.js

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1)Наповнення чат-боту

2) Діаграма варіантів використання

3) Меню створення нової команди

4) Візуальний вигляд скрипту «*In*»

5) Візуальний вигляд скрипту «*Out*»

Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Отримання та узгодження технічного завдання з керівником проекту	20.05.2022	
2	Збір теоретичні відомостей	21.05.2022	
3	Аналіз існуючих рішень	23.05.2022	
4	Проєктування програмного модуля	25.05.2022	
5	Розробка програмного модуля	30.05.2022	
6	Тестування на працездатність та ефективність	01.06.2022	
7	Написання пояснювальної записки	02.06.2022	

7. Дата видачі завдання: «20» травня 2022 р.

Керівник дипломної роботи (проєкту) Коба О.В.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання Коноваленко К.Ю

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту “Універсальний чат-бот банківських послуг на платформі Node.js”,

NODE.JS, NODE-RED, MESSANGERS, BOT, CHATBOT, JAVASCRIPT, CHATWOOT, API, REDIS, NOSQL, REST.

Мета дипломного проекту – створення універсального чат-боту у сфері банківських послуг з інтеграцією платформи для взаємодії з клієнтами.

Об’єкт дослідження – автоматизація підтримки клієнтів в банківській сфері.

Предмет дослідження – універсальний чат-бот банківських послуг на платформі *Node.js*.

Практична значущість проекту – автоматизація процесу отримання типових відповідей на питання від користувачів та спілкування з операторами підтримки.

Технології, технічні та програмні засоби, задіяні в спроєктованому об’єкті – мова програмування *JavaScript*, інструмент для візуального програмування потоком даних *Node-red*, платформа для взаємодії з клієнтами *Chatwoot*, програмна платформа *Node.js*, бібліотеки *axios*, *got*, *mustache* та інші.

Основні конструктивні, технологічні та інші характеристики та показники об’єкту проєктування – скрипт (*flow*) може бути імпортований та запущений на пристроях, де встановлено *Node.js*.

Отримані результати та їх новизна – даний проєкт розроблено за допомогою *Node-red*, що означає, що його може легко підтримувати інша людина без знання мови програмування, наприклад бізнес-аналітик.

Рекомендації щодо використання результатів – рекомендується до використання в банківській або іншій фінансовій сфері, але може бути адаптовано під інші сфери, що потребують даного бота.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Node-red – пакет *npm*, інструмент візуального програмування

Node.js – платформа для виконання застосунків написаних мовою *JavaScript*, основана на *Chrome V8 engine.*

Npm (*Node Package Manager*) – пакетний менеджер за замовчуванням для Node.js.

API (*Application Programming Interface*) - являє собою набір визначень і протоколів для створення та інтеграції прикладного програмного забезпечення.

Chatwoot – це пакет із відкритим кодом для взаємодії з клієнтами, створений як альтернатива *Intercom, Zendesk і Salesforce Service Cloud*.

Flow – Потік представлений у вигляді вкладки в робочій області редактора і є основним способом організації вузлів.

Inbox – сутність Chatwoot, являє собою скриньку для вхідних повідомлень для клієнтів.

Вебхук (Webhook) – url який дозволяє програмам і службам надсилати веб-сповіщення іншим програмам, коли відбувається конкретна подія.

RedBot – назва бібліотеки *node-red-contrib-chatbot*

Subflow – це набір вузлів, які згорнуті в один вузол у робочій області.

ВСТУП

Банківські структури мають великий обсяг клієнтів, які зустрічаються з проблемами в процесі експлуатації банківських послуг. Коли клієнт стикається з такого роду проблемами він звертається до центру підтримки користувачів. Для роботи такого центру потрібно велика кількість операторів підтримки, на яких банк витрачає значну долю бюджету. Більшість запитів клієнтів є банальними та типовими питаннями на які оператори центру підтримки відповідають за шаблонними відповідями. Це і є основною причиною появи чат-ботів в даному секторі. Боти беруть на себе більшу частину навантаження на центр підтримки, тому велика кількість питань навіть не буде доходити до операторів.

Метою роботи є розробка чат-боту, який буде відповідати на типові питання клієнта, відправляти йому текст відповіді, посилання або документи, які відносяться до вирішення його питання. Тільки в разі його не спроможності допомогти з'єднувати клієнта з оператором відповідної групи підтримки.

Для цього потрібно буде:

- Дослідити можливості бібліотеки *node-red* та як правильно працювати з візуальним програмуванням в даному середовищі. Зрозуміти основні концепції цього підходу, його переваги та недоліки в порівнянні з класичним програмуванням.
- Розробити розширений функціонал для *node-red*. Якщо його стандартного функціоналу буде не достатньо для реалізації поставлених задач.
- Дослідити платформу для взаємодії з клієнтами *Chatwoot*. Зрозуміти як розвернути дану платформу. Вивчити взаємодію з нею через представлене розробниками API та розробити за його допомогою інтеграцію даної платформи з середовищем *node-red*.
- Вивчити типові вимоги від потенціальних клієнтів, тобто від банківських структур та на їх основі розробити технічне завдання, як

в текстовому вигляді так і в графічному.

- Розробити *flow* для *node-red* по розробленому раніше технічному завданню та зробити його максимально зрозумілим для подальшого супроводження.

Об’єктом дослідження є автоматизація підтримки клієнтів в банківській сфері.

Предмет дослідження це універсальний чат-бот банківських послуг на платформі *Node.js*.

В процесі роботи будуть використанні методи дослідження такі як – аналіз, порівняння, вивчення теоретичного матеріалу, тести та вивчення результатів діяльності.

Очікується, що результати даної роботи будуть використовуватись для автоматизації процесу підтримки клієнтів користувача системи, спростять процес організації центру підтримки та дозволять зменшити кількість операторів підтримки за рахунок того, що більшість відповідей буде даватися чат-ботом.

Варто зазначити, що результати даної роботи вже комерційно використовуються компанією *NovaIT*. Компанія підключає до системи, частиною якої є результати даної роботи реальних замовників. Замовниками є компанії, як банківського сектору, так і інших сфер послуг.

РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМАТИКИ ТА ІНСТУМЕНТАРІЙ

1.1 Проблематика проєкту

Позитивні враження від підтримки є ключем до створення лояльної бази клієнтів в банківській і далеко не тільки банківській сфері. Якщо люди не отримають миттєвих відповідей на свої найактуальніші запитання, банк може втратити клієнта або навіть потенційних клієнтів, яким могли порекомендувати саме ваш сервіс. Більш того не задоволений клієнт може перейти до конкурентів.

Але що станеться, якщо команда підтримки клієнтів переповнена кількістю вхідних запитань? Що робити, якщо недостатньо людей, щоб відповідати на ці запити в режимі реального часу?

У ситуаціях, коли ваш вхідний трафік вищий, ніж може обґрунтовано керувати персонал служби підтримки, чат-бот є одним із найкращих ресурсів для обробки більшого обсягу. Чат-бот це програмне забезпечення, яке використовується для ведення онлайн-чату. Чат-ботів можна запрограмувати так, щоб вони відповідали на поширені запитання в режимі реального часу. Це допомагає автоматизувати процес підтримки клієнтів і гарантувати, що клієнти отримають підтримку, необхідну для задоволення їхніх потреб.

Багато команд служби підтримки клієнтів зациклені на повторюваних завданнях, щоб знову і знову відповідати на одні й ті ж версії поширених запитань. Це може призвести до вигорання та незадоволеності роботою, що в кінцевому підсумку знизить рівень якісного догляду, за яким стоїть ваш бренд.

Оскільки чат-боти не відчувають людських почуттів, вони не реагують на повторювані чорні завдання з будь-яким ступенем незадоволення. Вони просто запрограмовані виконувати роботу та надавати необхідний покупцям рівень підтримки. Найголовніше, вони можуть автоматизувати багато з цих завдань, тим самим прискорюючи час реагування та зменшуючи незадоволеність та/або плинність працівників.

Якщо ваша команда підтримки працює стандартно по 9–5 годин, вони можуть бути недоступні для тих клієнтів, які звертаються з іншого часового поясу.

Чат-боти забезпечують підтримку в режимі 24/7 в режимі реального часу для всіх покупців, незалежно від мовного бар'єру та географічного розташування. Якщо відділи підтримки недоступні для цих міжнародних відвідувачів, чат-бот може надати винятковий рівень обслуговування без сумнівів. Таким чином, ви уникнете втратити потенційно вигідні візити міжнародних покупців.

1.2 Вибір типу чат-боту

Програмне забезпечення для живого чату. Живий чат — це ручний спосіб прямого спілкування з клієнтами, коли вони переглядають ваш веб-сайт, по суті, еквівалент *Facebook Messenger*. За даними *SuperOffice*, 79% клієнтів вважають за краще використовувати чат у реальному часі, а не стрибати під час телефонного дзвінка з агентом підтримки, оскільки він дає негайні відповіді на нагальні запитання.

За допомогою чату ви отримуєте конкретне уявлення про потреби ваших покупців і надаєте прямі відповіді на ці запити. Використовуючи ці відповіді, ви можете спрямовувати покупців до найціннішої точки конверсії, яка дасть їм потрібні товари, одночасно збільшуючи прибуток для вашого бізнесу.

Чат-боти на основі правил. Чат-боти на основі правил — це приклади типів ботів, з якими люди взаємодіяли роками на багатьох веб-сайтах. Вони

використовують попередньо визначений набір правил, в основному побудованих на операторах *if/then*, які інформують ботів про те, як автоматично відповідати на поширені запитання замість людини.

Перевага чат-ботів на основі правил полягає в тому, що їх досить легко створити та навчити взаємодіяти з клієнтами. Недоліком є те, що, оскільки вони запрограмовані за допомогою конкретних відповідей, заснованих на правилах, вони обмежені в тому, як вони можуть брати участь у корисній розмові з покупцями.

AI чат-боти. На відміну від чат-ботів, заснованих на правилах, чат-боти з штучним інтелектом можуть давати активні рекомендації покупцям, викликані поведінкою перегляду. Чат-боти використовують обробку природної мови (*NLP*) і ‘технологію машинного навчання, щоб реагувати на наміри покупців щодо поведінки покупців.

Компанії використовують чат-ботів зі штучним інтелектом, щоб не тільки автоматизувати підтримку клієнтів, але вони також допомагають оптимізувати коефіцієнт конверсії, покращити якість роботи з клієнтами та сприяти прямому прибутку. Насправді, чат-боти зі штучним інтелектом допомогли збільшити дохід електронної комерції на 60%.

Зупинимося на другому та третьому варіанті: класичний скриптовий бот та бот, який використовує *NLP*.

До недоліків *NLP* можна віднести:

- Навчання може зайняти час: якщо необхідно розробити модель з новим набором даних без використання попередньо навченої моделі, для досягнення хорошої продуктивності залежно від обсягу даних можуть знадобитися тижні.
- Це не 100% надійно: одним з недоліків машинного навчання є те, що воно ніколи не буває на 100% надійним. Є ймовірність помилок у його прогнозуванні та результатах.

Недоліки ботів на основі правил:

- Вони не можуть охопити весь спектр питань, адже мають відповіді тільки на заготовані раніше тексти. Тобто при розробці бота, потрібно підготувати текст, при вводі якого буде вибрана та чи інша категорія. Цей процес дуже полегшує можливість додати кнопки, щоб користувач натискав їх, а не користувався рукописним вводом.
- Не вміє реагувати на не заготовані сценарії. Якщо користувач введе не вірні дані та піде по не підготованому сценарію, то поведінка бота може бути непередбачуваною.

Ґрунтуючись на описаних вище недоліках зупинимось на варіанті класичного боту з прописаним раніше сценарієм, адже для нас вирішальним аргументом є точність відповіді, яка буде даватися користувачам. На конкретне запитання або категорію повинна буди дана відповідь, точність якої 100%. Досягти цього за допомогою штучного інтелекту на жаль зараз не є можливим.

1.3 Вибір платформи для взаємодії з клієнтом

З попередніх розділів ми зрозуміли, що чат-бот не є інструментом, який може вирішити всі проблеми користувачів, тому вони повинні мати можливість спілкування з оператором для вирішення більш складних та комплексних запитань.

Платформа взаємодії з клієнтами — це програмне забезпечення, яке допомагає підприємствам керувати комплексними відносинами з клієнтами. У свою чергу, реальні зв’язки з клієнтами, які є результатом, можуть покращити задоволеність клієнтів, продажі та утримання.

Платформа залучення клієнтів — це програмне забезпечення, яке допомагає компаніям керувати, аналізувати та оптимізувати шлях клієнта. Він робить це, автоматично надсилаючи персоналізовані повідомлення клієнтам на різних пристроях і платформах.

Інструменти взаємодії з клієнтами можуть дозволити бренду об’єднати всі дані своїх клієнтів в одному місці та надати клієнтам надзвичайно персоналізований досвід. Це засновано на всіх видах змінних, як-от:

- пристрій, яким вони користуються
- історія
- статус тікету підтримки
- активність у соціальних мережах

Визначимо, який саме функціонал є для нас важливим:

- Дозволяє персоналізувати досвід. Відносини мають вирішальне значення для лояльності клієнтів. Щоб забезпечити персоналізований досвід, компаніям потрібна платформа залучення клієнтів із контекстом клієнта, як-от історія підтримки клієнта, тип облікового запису та налаштування замовлень.

- Підтримує команди. Залучення клієнтів вимагає залучення співробітників. Найкраща платформа залучення клієнтів має команди підтримки з відповідними інструментами, щоб добре виконувати свою роботу.

- Надає клієнтам можливість самообслуговування. Клієнти покладаються на онлайн-ресурси компаній у нашому цифровому світі. Коли клієнти можуть самообслуговуватися, вони швидше отримують відповіді на прості запитання, а вашим агентам не доведеться витрачати час на вирішення повторюваних проблем.

- Легко налаштовується. Платформи залучення клієнтів, які легко налаштувати та прості у використанні, гарантують, що ви можете зосередитися на залученні клієнтів, замість того, щоб адмініструвати інструмент і навчати команди використовувати його.

- Забезпечує круговий огляд клієнта. Завдяки платформі взаємодії з клієнтами, яка об’єднує всі дані ваших клієнтів, де б вони не були, ваші команди можуть передбачити потреби клієнтів і

запропонувати краще персоналізований досвід. Повний огляд клієнтів гарантує, що кожна команда, від маркетингу до підтримки клієнтів, має цілісне розуміння ваших клієнтів.

Під данні пункти підходить платформа *Chatwoot*, яка до цього ж є *open-source* проектом. Пройдемося по її перевагам та можливостям:

- Можливість напряму інтегрувати месенджери або ж створити інтеграцію за допомогою *API*. Меню створення нового каналу представлено на рисунку 1.1.

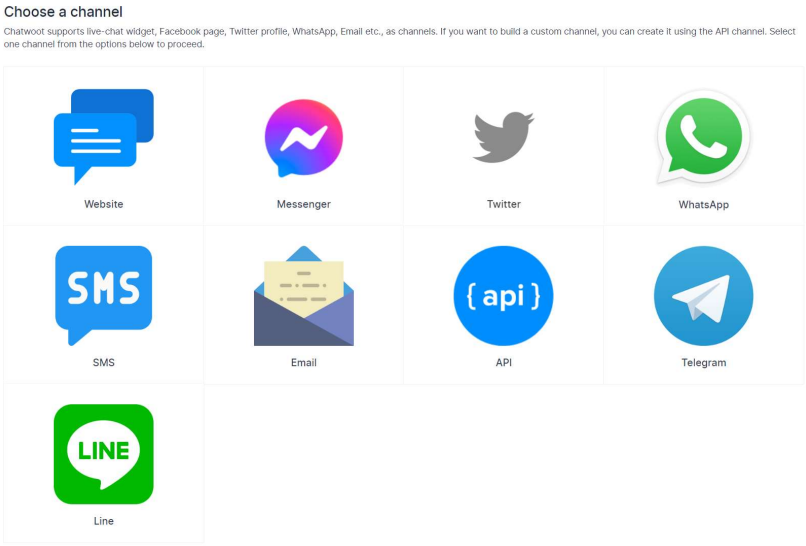


Рис. 1.1 Меню створення нового каналу

Створення уособлених команд для різних цілей. Меню створення представлено на рисунку 1.2.

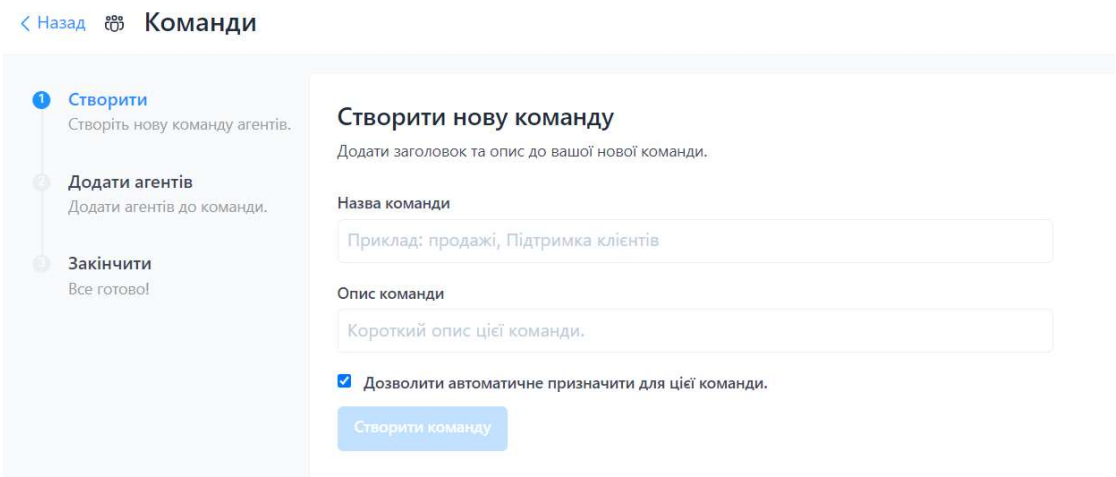


Рис. 1.2. Меню створення нової команди

Гнучке налаштування контактів користувачів, додавання до них різних атрибутів та полів для більш якісного обслуговування клієнтів. Приклад картки клієнта представлена на рисунку 1.3.

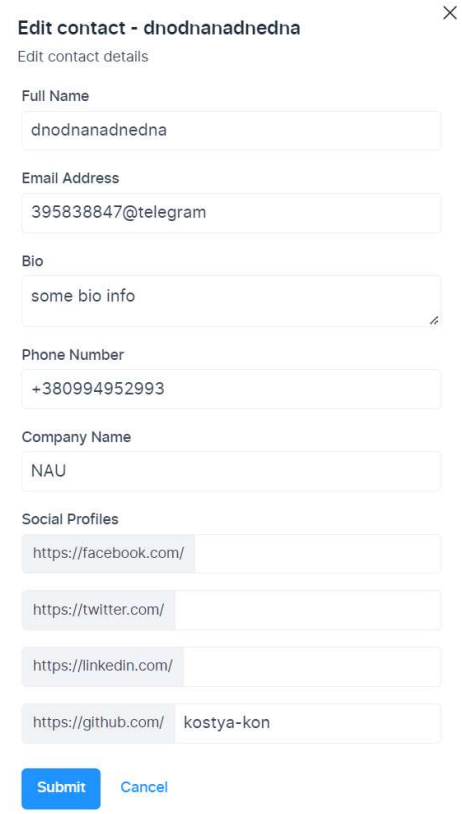


Рис. 1.3. Карточка клієнта (контакт)

Вбудована статистика по діалогам, агентам, командам та іншому. Вигляд статистики представлено на рисунку 1.4.

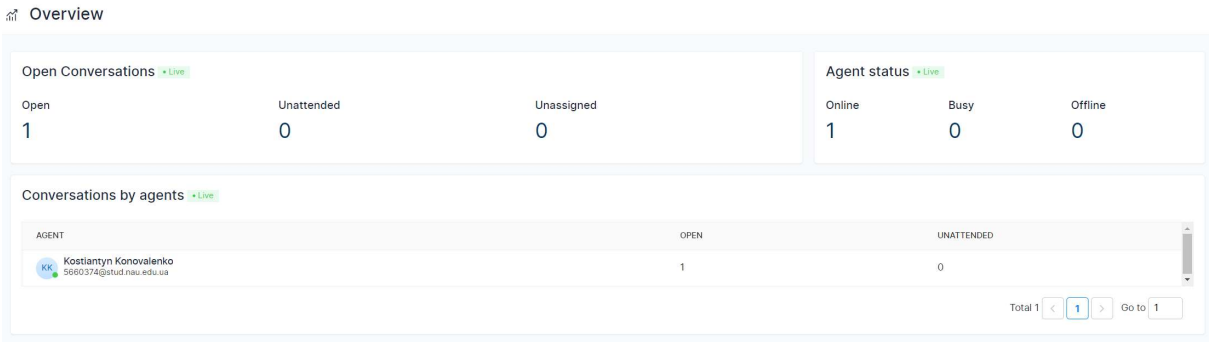


Рис. 1.4. Сторінка статистики в реальному часі

Головною перевагою є зручна інтеграція за допомогою інструменту *AgentBot* за допомогою якого і буде проводитися інтеграція чат-боту та даної платформи.

AgentBot — це веб-сервіс, підключений до *inbox chatwoot* і може виконувати роль бота, що обробляє запити клієнтів. *Chatwoot* дозволяє легко підключати вашу власну логіку бота до обробки розмови за допомогою API *AgentBot*.

Як тільки ви підключите *Bot Agent* до *inbox*, всім новим розмовам, створеним у вашому *inbox* спочатку буде присвоєно статус "бот". *Chatwoot* надсилатиме кожну подію бесіди на *URL*-адресу вашого бота як події вебхуку. На них ваш *Bot Agent* може реагувати через *API Chatwoot*.

1.4 Вибір інструментів розробки

1.4.1 Чому було обрано *Node.js*

Платформою для розробки було вибрано *Node.js*.

Що таке Node.js? Простіше кажучи, *node* - це просто сервер, такий як *Apache, IIS, TOM* тощо. Але на відміну від цих серверів *Node* не має справу з *PHP, .NET* або *JAVA*. Він виконує *JavaScript* на стороні сервера. Так, *JavaScript* без браузера; це *Node*. Сам *Node* не повністю створений у *JavaScript*, його обгортки зроблені на *C*. Він просто виконує *JavaScript*. Нижче наведено список, що не є *Node* і що таке *Node*.

Чим не є *Node.js:*

- *Node* - це не фреймворк, це сервер.
- *Node* це обгортка над *JavaScript V8 Runtime* не створена в *JavaScript*, а створена на *C*.
- Він не багатопотоковий. Він працює в одному потоці з концепцією зворотного виклику.
- Це не для початківців *JavaScript*, оскільки він дуже низького рівня.

Чим є *Node.js:*

- *Node* — це сервер, який може виконувати *JavaScript*. Щось на кшталт браузеру на стороні серверу.
- *Node* — це кросплатформена платформа з відкритим вихідним кодом для створення мережевих додатків у режимі реального часу.
- Він надає вам асинхронні *API* вводу-виводу, керовані подіями.
- Він запускає цикл на основі однопотокових подій, тому всі виконання стають неблокуючими.

Чому потрібно використовувати Node:

- **Неблокуючий код.** Це найсильніша причина вибрати *Node* як сервер. *Node* повністю керується подіями, і більшість коду виконується на основі зворотних викликів. Такий підхід допомагає додатку не зупинятися і не спати, а ставати доступним для інших запитів.

- **Швидка обробка.** *Node* використовує двигун *V8 JavaScript Runtime*, той, який використовує *Google Chrome. Node* має обгортку над цим механізмом *JavaScript*, що надає додаткові можливості для створення мережевих додатків. І оболонка *Node*, і движок *JavaScript V8* написані мовою *C*, що робить його дуже швидким. Він набагато швидше, ніж *Ruby, Python* або *Perl*.

- **Одночасна обробка запитів.** Вузол може обробляти тисячі одночасних з'єднань з дуже мінімальними накладними витратами на один процес.

- **Одне середовище.** Використання *JavaScript* на веб-сервері, а також у браузері зменшує невідповідність імпедансу між двома програмними середовищами, які можуть передавати структури даних через *JSON*, які однаково працюють з обох сторін рівняння. Дублікат коду перевірки форми може передаватися між сервером і клієнтом тощо.

- **Легко навчитися.** *Node* не є чимось новим. Це просто простий *JavaScript*. Тож розробникам *JavaScript* не потрібно докладати зайвих зусиль, щоб вивчити *Node*.

- **Популярність і спільнота.** Багато людей вже знають *JavaScript*,

навіть ті, хто не претендує на звання програмістів. Це, мабуть, найпопулярніша мова програмування. Отже, популярна мова означає популярну спільноту. Велика спільнота означає, що легко отримати допомогу чи підтримку.

1.4.2 Візуальне програмування та *node-red*

Візуальне програмування — це мова програмування або інструмент, який дозволяє людям описувати процеси за допомогою ілюстрацій. У той час як типова текстова мова програмування змушує програміста думати як комп'ютер, візуальна мова програмування дозволяє програмісту описати процес у термінах, які мають сенс для людей.

Основною причиною вибору візуального програмування для даного проєкту є те, що проєкт в подальшому повинен буде легкий для підтримки. В разі реалізації проєкту за допомого стандартних засобів програмування ця задача була би складною для користувача, який не володіє на достатньому рівні *JavaScript* та *Node.js*. Оскільки чат-бот, який буде розроблено повинен могти адаптуватися під конкретного замовника, людина, яка супроводжує проєкт повинна могти внести до нього не значні зміни, такі як наприклад: змінити текст відповіді, змінити назву кнопок, або інші незначні візуальні доробки. Саме тому, для спрощення даного процесу, було вибрано візуальне програмування, як інструмент для реалізації чат-боту. За його допомогою не потрібно буде залучати розробника для незначних змін, це зможе зробити наприклад бізнес аналітик.

Node-RED — це інструмент для програмування на основі потоків, спочатку розроблений командою *IBM Emerging Technology Services*, а тепер є частиною *OpenJS Foundation*.

Програмування на основі потоку. Винайдене Дж. Полом Моррісоном у 1970-х роках програмування на основі потоків — це спосіб опису поведінки програми як мережі чорних скриньок, або «вузлів», як їх називають у *Node-RED.* Кожен вузол має чітко визначене призначення; йому надаються деякі

дані, він щось робить з цими даними, а потім передає ці дані далі. Мережа відповідає за потік даних між вузлами.

Це модель, яка дуже добре піддається візуальному відображенню та робить її більш доступною для більш широкого кола користувачів. Якщо хтось може розбити проблему на окремі кроки, він може подивитися на потік і зрозуміти, що він робить; без необхідності розуміти окремі рядки коду в кожному вузлі.

Час виконання/редактор. *Node-RED* складається з середовища виконання на основі *Node.js*, на яке ви вказуєте веб-браузер, щоб отримати доступ до редактора потоків. У браузері ви створюєте свою програму, перетягуючи вузли зі своєї палітри в робочу область і починаєте з'єднувати їх разом. Одним клацанням миші програма повертається до середовища виконання, де вона запускається.

Палітру вузлів можна легко розширити, встановивши нові вузли, створені спільнотою, а створені вами потоки можна легко поділитися як файли *JSON*.

Вузол є основним будівельним блоком потоку. Вузли запускаються або шляхом отримання повідомлення від попереднього вузла в потоці, або шляхом очікування якоїсь зовнішньої події, наприклад, вхідного запиту *HTTP*, таймера або зміни обладнання *GPIO*. Вони обробляють це повідомлення або подію, а потім можуть надіслати повідомлення наступним вузлам потоку.

Вузол може мати щонайбільше один вхідний порт і стільки вихідних портів, скільки йому потрібно.

Основний спосіб розширення *Node-RED* - це додавати нові вузли до його палітри. Вузли можна опублікувати як модулі *npm* у загальнодоступному репозиторії *npm* і додати до бібліотеки потоків *Node-RED*, щоб зробити їх доступними для спільноти.

1.4.3 Бібліотека *RedBot*

Основною бібліотекою було вибрано *RedBot (node-red-contrib-chatbot)*. Ця бібліотека включає в собі велику кількість вузлів для *node-red*, що значно розширює його можливості. Дані вузли направлені на спрощення побудови саме чат ботів, та з коробки дозволяє підключати такі канали, як *Telegram, Viber, Facebook Messenger* та *Slack*.

Приклад найпростішого скрипту для *node-red* з допомогою даної бібліотеки представлено на рисунку 1.5.

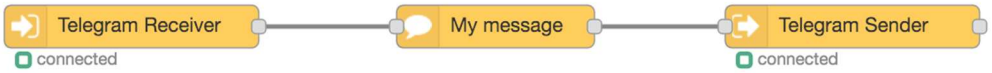


Рис. 1.5 Приклад скрипту за допомогою RedBot

На даному рисунку бачимо, як просто відправити повідомлення користувачеві в телеграм.

Основною перевагою даної бібліотеки є контекст чату. *Node-RED* має два контексту для змінних: global та flow, перший доступний скрізь у додатку, другий лише у виконаному потоці.

RedBot представляє контекст чату, де можна зберігати інформацію, пов'язану з конкретним користувачем. Вузол *Receiver* зберігає тут деяку інформацію за замовчуванням, як-от *chatId*, ім'я користувача, авторизований, ім'я, прізвище тощо, але можна зберігати будь-яку інформацію.

Пояснимо даний функціонал на прикладі. У нас є бот у якого є 4 кроки виконання. Без використання *RedBot* буде порушена логіка роботи чат боту, якщо до нього звертається декілька користувачів. Наприклад, користувач 1 пройшов два кроки та зупинився, тоді, коли напише користувач 2, він потрапить одразу на третій крок виконання програми, тому що вона

залишилась на даному кроці, після обробки першого користувача. *RedBot* вирішує дану проблему за допомогою контексту чату, завдяки ньому ми розуміємо різницю між користувачами, можемо побачити інформацію індивідуальну для них та розуміємо на якому кроці який з них знаходиться в даний момент часу.

1.4.4 Бібліотека *node-red-contrib-redis*

Дана бібліотека додає вузли для *node-red*, які спрощують взаємодію з не реляційною базою даних *Redis*. Вона містить в собі всі основні команди для взаємодії з *Redis* такі як: *get, set, del, expire* та інші. Також підтримує механізм *pub/sub* та може реагувати на *keyspace* події, які приходять з *Redis*. Приклад вузла *node-red-contrib-redis* представлено на рисунку 1.6.

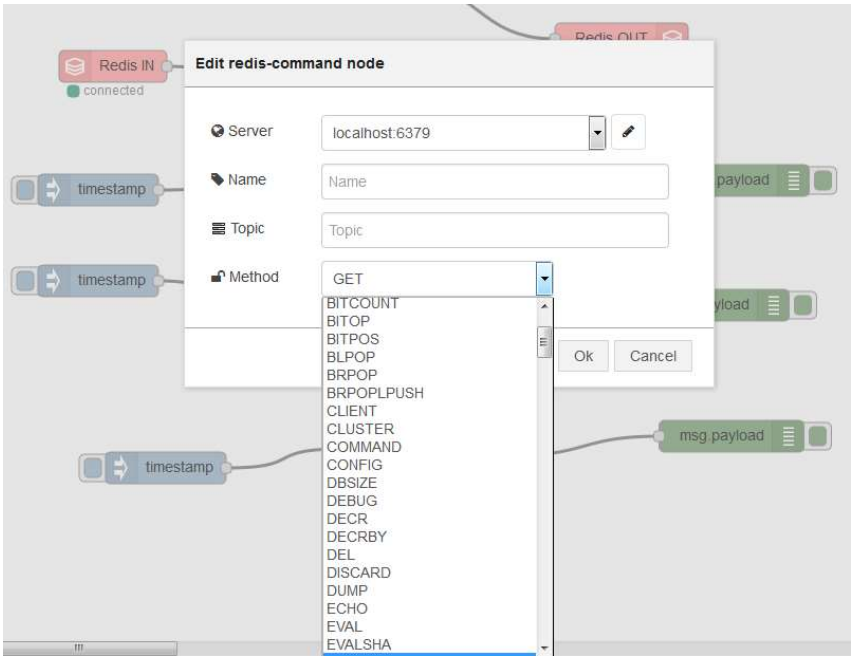


Рис. 1.6 Приклад вузла *node-red-contrib-redis*

1.5 Висновки до розділу

Було проаналізовано проблематику проєкту. Виявлено проблеми, які проєкт повинен вирішити та які переваги він надає для підтримки клієнтів. Чат-бот покращить враження від підтримки у клієнтів, а також покращить умови для самих працівників центру підтримки, адже зменшить кількість типових запитів.

Проаналізовано типи чат-ботів, їх переваги та недоліки. Було обрано бота на основі правил, так як він дає сто відсоткову точність відповіді.

Обрано інструменти для розробки проєкту. Проаналізовано переваги платформи *Node.js* та визначено в яких випадках її доцільно використовувати. Візуальне програмування виявилось підходящим інструментом для реалізації проєкту, тому було обрано бібліотеку *Node-RED* та в доповнення *Redbot*.

РОЗДІЛ 2 ФОРМУВАННЯ ВИМОГ ДО ЧАТ-БОТУ

2.1 Розроблення функціональних вимог

Діаграма варіантів використання зображення представлена на рисунку 2.1. В таблиці 1.3 наведені функціональні вимоги та їх пріоритети.

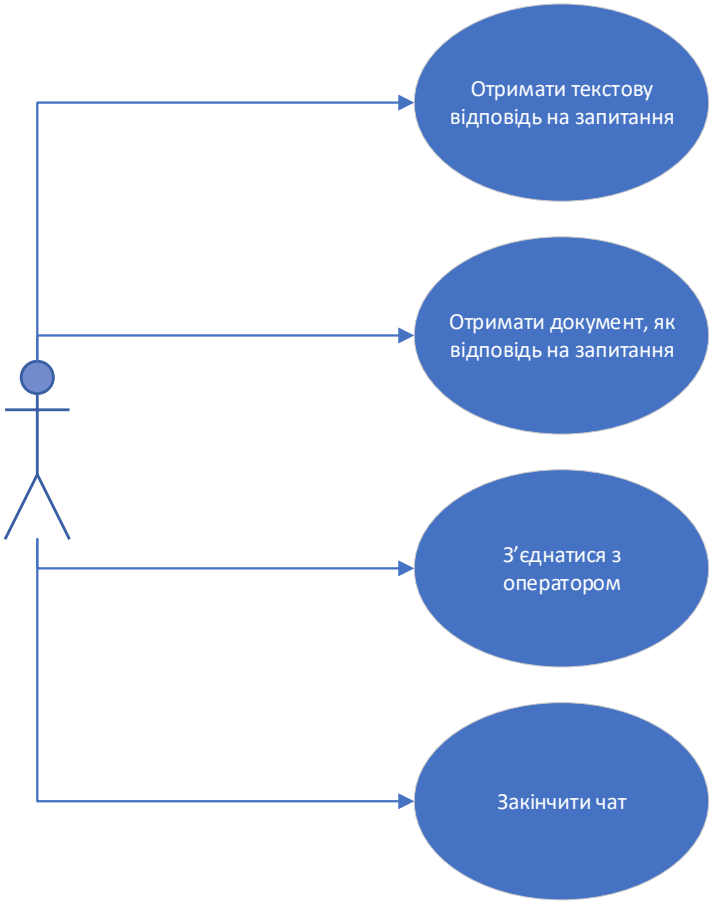


Рис. 2.1 Діаграма варіантів використання

Таблиця 2.1 – Функціональні вимоги та їх пріоритети

Варіант використання	Функціональна вимога	Пріоритет
Отримати відповідь на запитання	1. Чат-бот має давати точну відповідь по вибраній користувачем категорії. 2. Чат-бот має підтримувати посилання та коректно їх відображати в месенджері.	Високий
Отримати документ, як відповідь на запитання	1. Чат-бот повинен надавати документ в форматі *pdf* на конкретні прописані категорії. 2. Документ повинен підтримуватися та правильно відкриватися в месенджерах *Telegram* та *Viber*.	Високий
З'єднатися з оператором	1. При виборі категорії «Оператор» користувача потрібно з'єднати з оператором підтримки з команди «*Support team*» 2. При переводі діалогу на оператора логіка чат-боту повинна буди вимкнена. 3. Оператору потрібно передати данні про користувача, його ідентифікатор в месенджері та його ім'я.	Високий
Закінчити чат	1. При виборі категорії	Високий

	«Закінчити чат» потрібно вивести повідомлення про те, що чат було закінчено. 2. При закінчені чату користувач повинен перейти на перший крок взаємодії з чат-ботом. 3. Чат повинен завершатися самостійно, якщо протягом 1 години немає активності в чаті. 4. При закінчені чату статус діалогу повинен бути «Завершено».	

2.2Формування нефункціональних вимог

- Система повинна могти бути запущеною на системі Windows, Linux.
- Користувач повинен мати змогу вводити текст вручну та натискати на кнопки.
- При повторному звернені протягом 5 хвилин користувач повинен бути з’єднаним з оператором з яким він спілкувався раніше, але тільки, якщо він знаходиться в статусі «онлайн».
- Відповідна категорія повинна бути вибраною якщо користувач:
    1) Натиснув кнопку
    2) Ввів тест який повністю співпадає з текстом кнопки
    3) Ввів число, яке відповідає числу на кнопці
- Чат-бот повинен вміти призначати теги для діалогів.
- В систему повинен бути вбудований механізм тротлінгу та пропускати не більше 2 повідомлень в 1 секунду.

2.3 Формування логіки чат боту

На першому кроці чат-бот повинен привітати користувача повідомленням

«*NAU*-банк вітає Вас!»

Далі чат-бот повинен запитати

«Ви звертаєтесь як фізична чи юридична особа?».

При виборі варіанта «фізична особа» потрібно присвоїти тег «*FOP*» та відобразити повідомлення:

«Вам потрібно перейти до чату саме для фізичних осіб та ФОП.

Facebook: naubank.ua/facebook,

Telegram: naubank.ua/telegram,

Viber: naubank.ua/viber.

Дякуємо за Ваше звернення!»

та завершити діалог.

При виборі варіанту «Юридична особа» присвоюємо тег «Legal» та перейти до наступного кроку.

Крок 2 – відобразити повідомлення «Оберіть тему Вашого запиту:

1 - Питання по Клієнт-Банк *Click NAUPay*

2 - Зарплатна відомість або платіж

3 - Тарифи *NAU* Банку

4 - Відкрити рахунок у *NAU* Банку

5 - Інтернет - Еквайринг у *NAU* Банку»

та відобразити кнопки

«1

2

3

4

5

Оператор чату

Завершити діалог»

Варіанти вибору користувача:

1- перейти до кроку 3

2- перевести користувача на оператора

3- відобразити повідомлення

«Вам потрібно перейти до чату саме для фізичних осіб та ФОП.

Facebook: naubank.ua/facebook,

Telegram: naubank.ua/telegram,

Viber: naubank.ua/viber.

Дякуємо за Ваше звернення!»

та перейти на крок 4

4- відобразити повідомлення

«Вам потрібно перейти за одним з посилань: Середній та малий бізнес *https://www.naubank.com.ua/smb/tariff-packages/*

Великий бізнес *https://www.naubank.com.ua/big-corporate/tariff-packages/*

Агробізнес *https://www.naubank.com.ua/big-corporate/tariff-packages/agro/*

Вибрати тарифний пакет, та заповнити заявку на контакт з працініком банку у розділі на прикінці сторінки "Залишити повідомлення представнику банку".» та перейти на крок 4

5- відобразити повідомлення

«Перейдіть, будь ласка, за посиланням:

https://www.naubank.com.ua/smb/rko/internet-acquiring/»

та перейти на крок 4

Крок 3 – відобразити повідомлення

«Виберіть питання, що Вас цікавить:

1 - Документація по *Click NAUPay*

2 - Аварійне відновленню ключа

3 - Планова зміна ключа

4 - Перенос ключа до мобільного

5 - Помилки Click NAUPay

6 - Перший вхід до Клієнт-Банку»

Та відобразити кнопки

«1

2

3

4

5

6

Оператор чату

Головне меню

Завершити діалог»

Варіанти вибору користувача:

2, 3, 4, 6 – відобразити відповідний документ, відобразити текст «Інформацію по Вашому питанню зможете знайти у файлі у вкладенні.» та перейти на крок 4

1 - відоборазити текст

«Перейдіть, будь ласка, за посиланням:

https://naupay.com.ua/ifobsClientSMB/downloads/Click_NAUPay_User_manual_2022.zip»

та перейти на крок 4

Крок 4 – відобразити кнопки

«Оператор чату

Головне меню

Завершити діалог»

Варіанти «Оператор чату», «Головне меню» та «Завершити діалог» повинні працювати однаково на всіх кроках:

«Оператор чату» - перевести користувача на оператора

«Головне меню» - перевести користувача на крок 2

«Завершити діалог» - закрити діалог в *Chatwoot* та перевести клієнта на крок 2

Наповнення чат-боту та варіанти відповідей наведені в додатку А.

Алгоритм відповідей чат-боту наведено в додатку Д.

2.4 Висновки до розділу

В даному розділі було проаналізовано вимоги до реалізації проєкту. Було визначено варіанти використання для користувача. Визначенні функціональні вимоги для кожного варіанту використання. Також були прописані нефункціональні вимоги для коректної роботи чат-боту.

Був визначений алгоритм роботи чат-боту, як у текстовому вигляді так і у вигляді візуальної схеми, зробленої в *MS Visio*. Деталізовано кожний крок роботи чат-боту, прописані текстові відповіді, кнопки та варіанти дії в залежності від вибраної кнопки.

РОЗДІЛ 3 РОЗРОБКА УНІВЕРСАЛЬНОГО ЧАТ-БОТУ ДЛЯ БАНКІВСЬКИХ ПОСЛУГ НА ПЛАТФОРМІ NODE.JS

3.1 Розробка інтеграції чат-боту з Chatwoot

Розробимо вузол для node-red, який буде зручною обгорткою над API Chatwoot та буде розширювати його функціонал. Призначення вузла – назначати оператора або команду в діалог.

В основі буде лежати метод *API*:

POST

https://app.chatwoot.com/api/v1/accounts/{account_id}/conversations/{conversation_id}/assignments

Для розробки будемо використовувати редактор коду *VS Code*.

Створимо два файли:

chatbot-chatwoot-botagent-transfer.html – відповідає за *front-end* частину,

chatbot-chatwoot-botagent-transfer.js – відповідає за *back-end* частину.

3.1.1 Розробка *back-end* частини вузла

Файл *.js* вузла визначає поведінку вузла під час виконання.

Конструктор вузлів

Вузли визначаються функцією конструктора, яку можна використовувати для створення нових екземплярів вузла. Функція реєструється в середовищі виконання, тому її можна викликати, коли вузли відповідного типу розгорнуті в потоці.

Функції передається об'єкт, що містить властивості, встановлені в редакторі потоку.

Перше, що він повинен зробити, це викликати функцію *RED.nodes.createNode*, щоб ініціалізувати функції, спільні для всіх вузлів.

```
function ChatWootTransferNode (config) {
    RED.nodes.createNode (this, config);
    // here is the code of a specific node
}
 RED.nodes.registerType('chatbot-chatwoot-botagent-transfer',
ChatWootTransferNode);
```

Прийом повідомлень

Вузли реєструють слухача подій для отримання повідомлень від верхніх вузлів у потоці.

У *Node-RED* 1.0 було введено новий стиль прослуховування, який дозволяє вузлу повідомляти середовище виконання, коли він закінчить обробку повідомлення. Це додало два нові параметри до функції прослуховування. Потрібна певна обережність, щоб переконатися, що вузол все ще може бути встановлений у *Node-RED* 0.x, який не використовує цей новий стиль прослуховування.

```
this.on('input', async (msg, send, done) => {
  done =
   done ||
   function (error) {
     this.error(error, msg);
   };
  });
```

В даному фрагменті реалізовано альтернативну функцію *done* для підтримки більш старих версій *Node-RED*.

Якщо вузол хоче надіслати зсередини слухача вхідних подій, у відповідь на отримання повідомлення, він повинен використовувати функцію відправки, яка передається функції прослуховування. Якщо *msg* є нульовим, повідомлення не надсилається.

Якщо вузол надсилає повідомлення у відповідь на його отримання, він повинен повторно використовувати отримане повідомлення, а не створювати новий об’єкт повідомлення. Це забезпечує збереження наявних властивостей повідомлення для решти потоку.

Даний файл являє собою розширення бібліотеки *node-red-contrib-chatbot*, тому ми можемо використовувати контекст чату.

const context = msg.chat();

Для призначення діалогу на оператора нам потрібно дістати змінні *chatId, transferTarget, transferTargetId, assignmentAlgorithm.* Ідентифікатор чату ми отримуємо з об’єкту msg, інші будуть вказані на front-end частині.

Далі для призначення діалогу створимо функцію *transferConversation*:

```
async transferConversation(conversationId, targetType, targetId, assignmentAlgorithm) {
  let agentAssigneeId;
  if (!conversationId || !targetType || !targetId) {
    throw new Error('Conversation transfer error');
  }
  let reqBody;
  if (targetType === 'team') {
    reqBody = { team_id: targetId };
    switch (assignmentAlgorithm) {
      case 'RAA': {
        agentAssigneeId = await this.getAgentRAA(targetId);
        break;
      }
      default:
        break;
    }
  } else {
    reqBody = { assignee_id: targetId };
  }
  let requests = [
    this.assignConversation(reqBody, conversationId),
    this.conversationToggleStatus({ status: 'open' }, conversationId),
```

```
    ];
    if (agentAssigneeId)
      requests.push(this.assignConversation({ assignee_id: agentAssigneeId },
conversationId));
    return Promise.all(requests);
  }
```

І нарешті опишемо основну функцію *assignConversation*, яка реалізує запит до *API Chatwoot*:

```
  assignConversation(body, conversationId) {
    return got
      .post({
        url: `${this.accountUrl}conversations/${conversationId}/assignments`,
        headers: {
          api_access_token: this.token
        },
        json: body,
        timeout: this.timeout,
      })
      .then(({ statusCode, body }) => {
        if (statusCode !== 200 || !body) {
          throw new Error('Conversation transfer error');
        }
        return body;
      });
  }
```

3.1.2 Розробка *front-end* частини вузла

Файл .html вузла визначає, як вузол буде виглядати в редакторі. Він містить три окремі частини, кожна з яких містить власний тег <*script*>:

1. визначення головного вузла, зареєстрованого в редакторі. Це

визначає такі речі, як категорія палітри, властивості, які можна редагувати (за замовчуванням) і яку піктограму використовувати.

2. шаблон редагування, який визначає вміст діалогового вікна редагування для вузла. Він визначається в сценарії типу text/html з ім’ям шаблону даних, встановленим на тип вузла.
3. текст довідки, який відображається на вкладці бічної панелі Інформація. Він визначається в сценарії типу *text/html з data-help-name* встановленим на тип вузла.

Визначення вузла. Вузол необхідно зареєструвати в редакторі за допомогою функції *RED.nodes.registerType*. Ця функція приймає два аргументи; тип вузла та його визначення:

RED.nodes.registerType('chatbot-chatwoot-botagent-transfer', {

// defaults

})

«*oneditprepare*:» це функція, що викликається під час створення діалогового вікна редагування. В неї вкладемо таку логіку: При відкриті вузла, *front-end* посилає запит на серверну частину, та в свою чергу звертається до *API Chatwoot*, щоб дізнатися список агентів, та команд, далі цей список відображається в візуальній частині. Це зроблено для зручності, щоб розробник не шукав та не вводив в ручну ідентифікатор оператора або команди, а вибрав зі актуального списку за допомогою інтерфейсу. Лістинг коду в додатку Б.

Діалогове вікно редагування

Шаблон редагування для вузла описує вміст його діалогового вікна редагування. *HTML* розмітка для вікна редагування в додатку В.

Текст довідки

Коли вибрано вузол, його текст довідки відображається на вкладці інформації. Це має надати змістовний опис того, що робить вузол. Він повинен визначити, які властивості він встановлює для вихідних повідомлень і які властивості можна встановити для вхідних повідомлень.

<script type="text/html" data-help-name="chatbot-chatwoot-botagent-transfer">

* *

Use this node to transfer the ChatWoot conversation from the BotAgent to the selected team or

agent.

* *

* The transfer details can be passed through the payload by the upstream node: *

<pre><code>msg.payload = {

transferTarget: 'agent',

transferTargetId: '1'

}

return msg;

</code></pre>

* or to do a team based transfer: *

<pre><code>msg.payload = {

transferTarget: 'team',

transferTargetId: '2'

}

return msg;

</code></pre>

</script>

В результаті маємо такий зовнішній вигляд меню, представлено на рисунку 3.1 та вигляд вузла на рисунку 3.2:

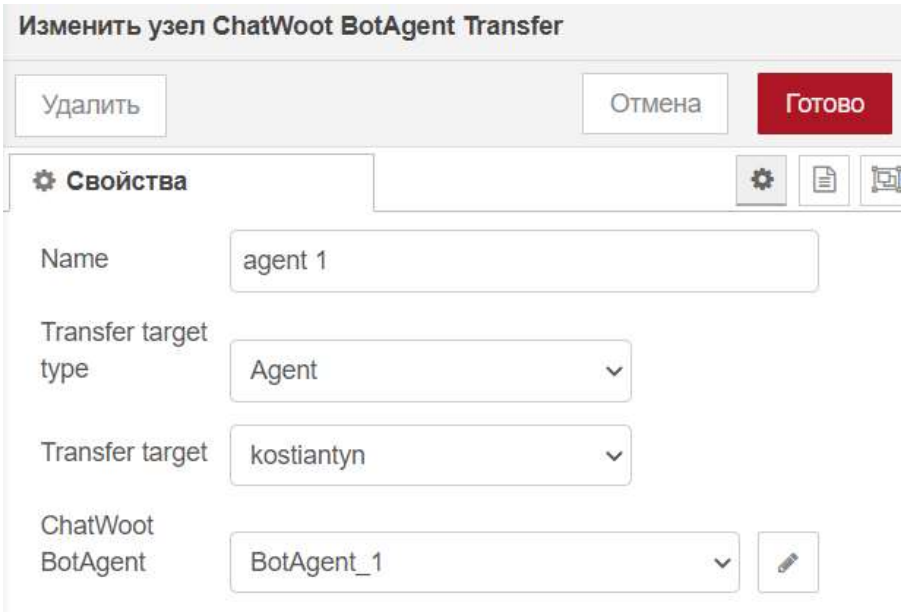


Рис 3.1 Зовнішній вигляд налаштувань вузла

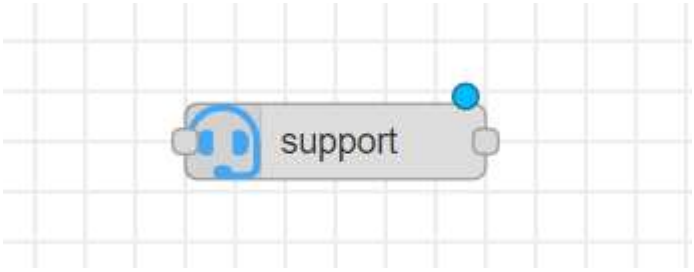


Рис 3.2 Зовнішні вигляд вузла на палітрі

3.2 Розробка скрипту для *Node-RED*

3.2.1 Розробка скрипту «*In*»

Скрипт «*In*» відповідає за підключення месенджерів до скрипту та передачу вхідних повідомлень до *Chatwoot*. Також на даному етапі скрипту використано бібліотеку *node-red-contrib-throttle* для запобіганню спаму та

навантаженню на систему. Дозволено оброблювати 2 повідомлення за 1 секунду.

Беремо з палітри вузол Telegram In та заповнюємо данні для підключення боту попередньо створивши його за допомогою *BotFather*. Налаштування конфігурації *Telegram* представлено на рисунку 3.3.

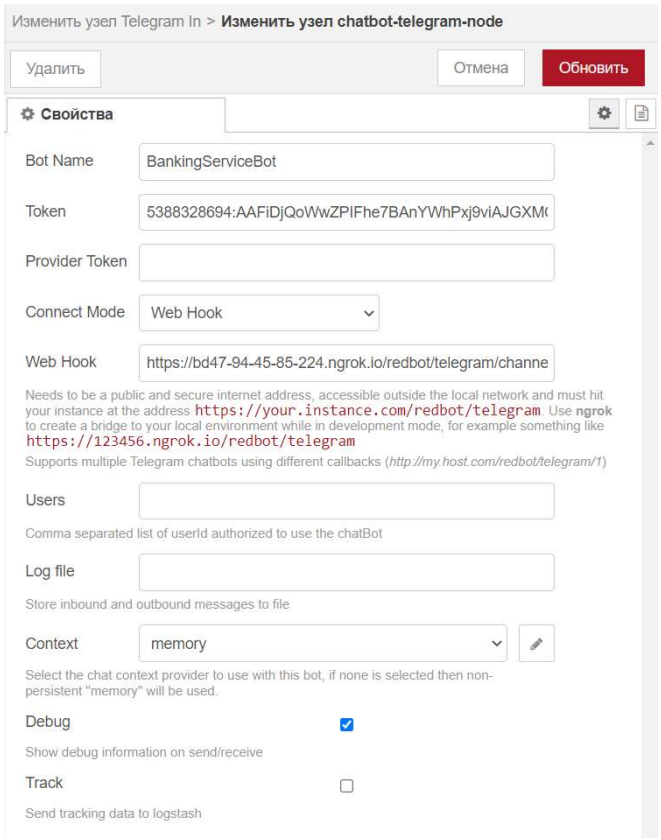


Рис 3.3 Налаштування конфігурації *Telegram*

Далі використовуючи *change* вузол нам потрібно встановити змінну *payload.username* для валідного відображення в *Chatwoot*. Налаштування *change* вузла представлене на рисунку 3.4.

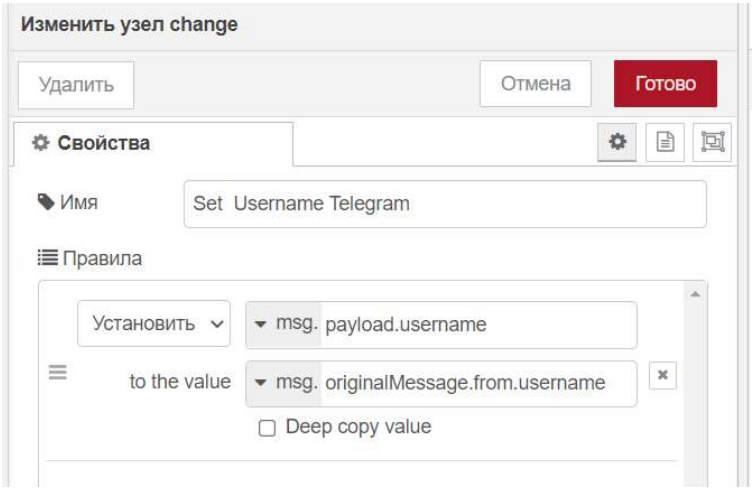


Рис. 3.4 *Change* вузол

Аналогічно повторюємо з *Viber*. Налаштування конфігурації *Viber* представлено на рисунку 3.5.

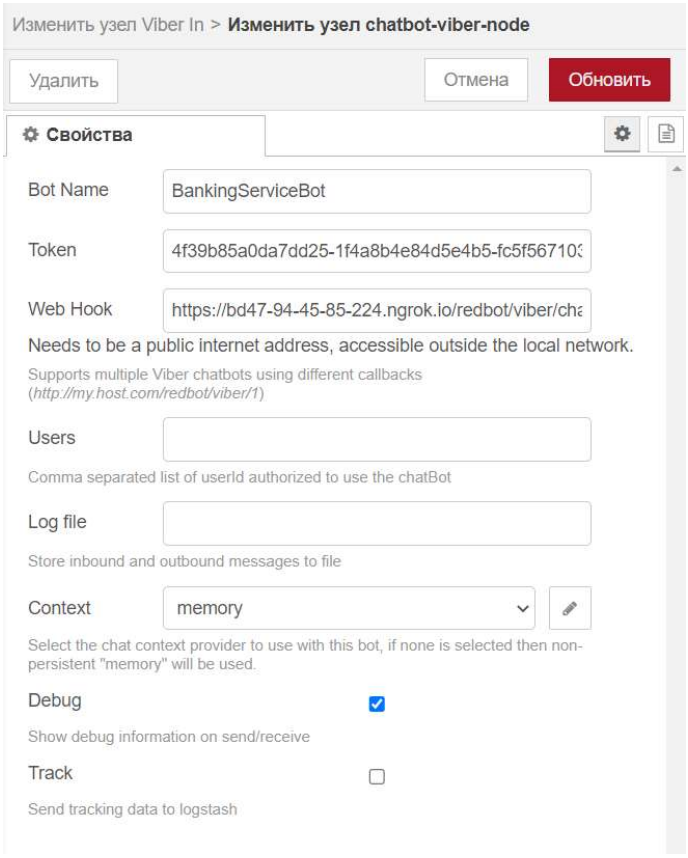


Рис. 3.5 Налаштування конфігурації *Viber*

Після чого з’єднуємо данні вузли з вузлом *throttle*. Налаштування вузла *throttle* представлено на рисунку 3.6.

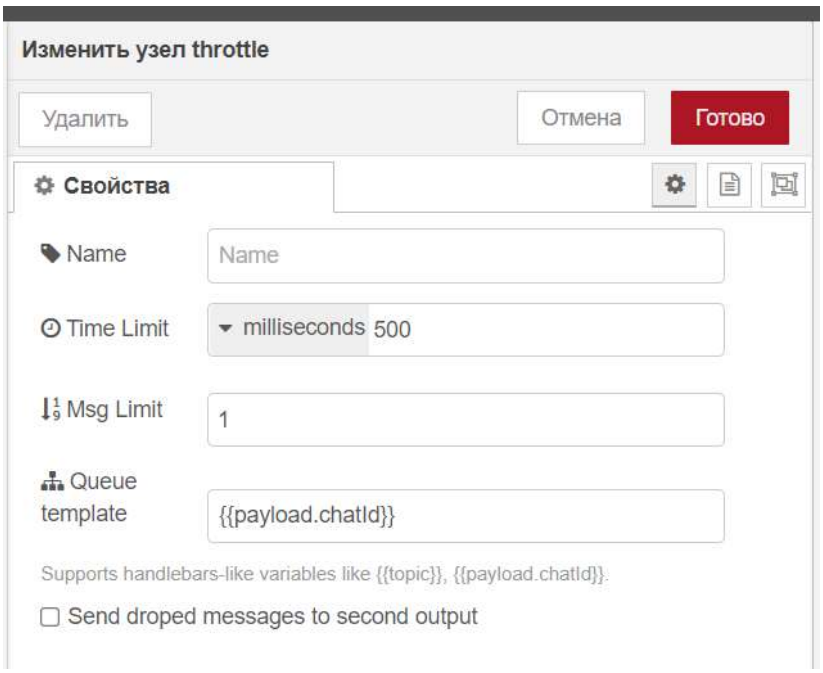


Рис. 3.6 Налаштування вузла *throttle*

Після чого додаємо вузол *Chatwoot In*. Налаштування вузла *Chatwoot In* представлено на рисунку 3.7.

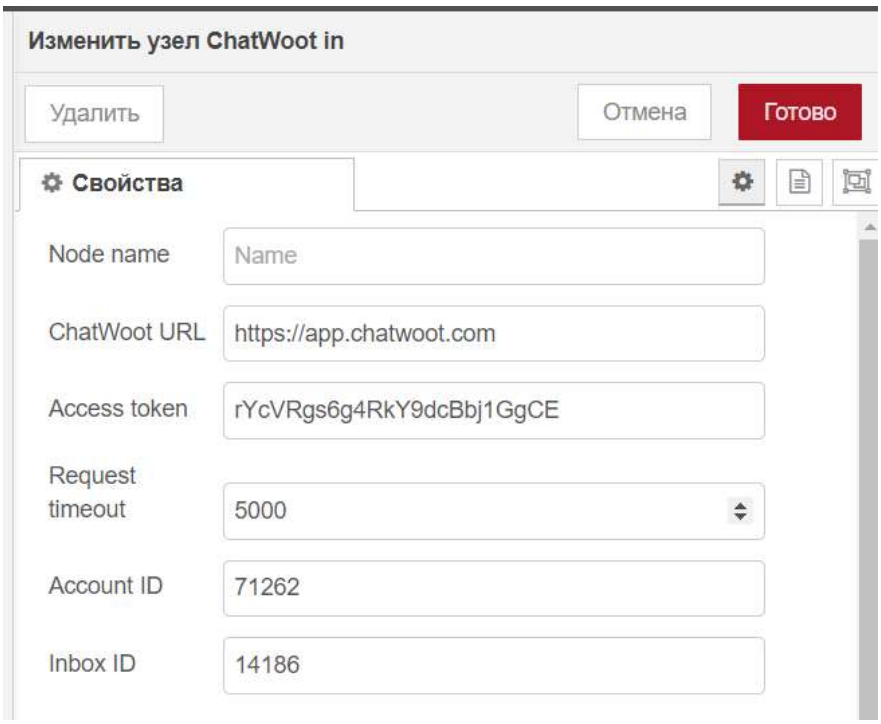


Рис. 3.7 Налаштування вузла *Chatwoot In*

З’єднуємо вузол *throttle* та *Chatwoot In* та отримуємо функціонуючий скрипт. Вигляд скрипту «*In*» представлено на рисунку 3.8.

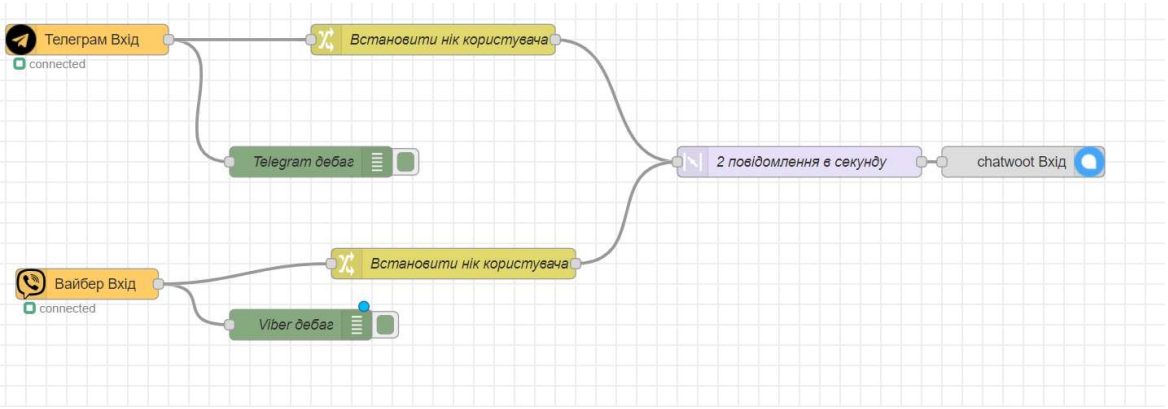


Рис. 3.8 Вигляд скрипту «*In*»

3.2.2 Розробка скрипту «*Out*»

Скрипт «*Out*» відповідає за те, щоб повідомлення які приходять на webhook зареєстрований в *Chatwoot* направлялися до вірного месенджеру. Таким чином ми обробляємо вихідні повідомлення, тобто від оператора або боту до користувача.

Створимо точку доступу */channel-chatwoot/1* – це і буде наш зареєстрований в *Chatwoot* вебхук. Її потрібно підключити до вузла *Chatwoot Out*, він буде перетворювати повідомлення с запиту, який прийде в правильний об’єкт *msg*. Налаштування точки доступу представлено на рисунку 3.9.

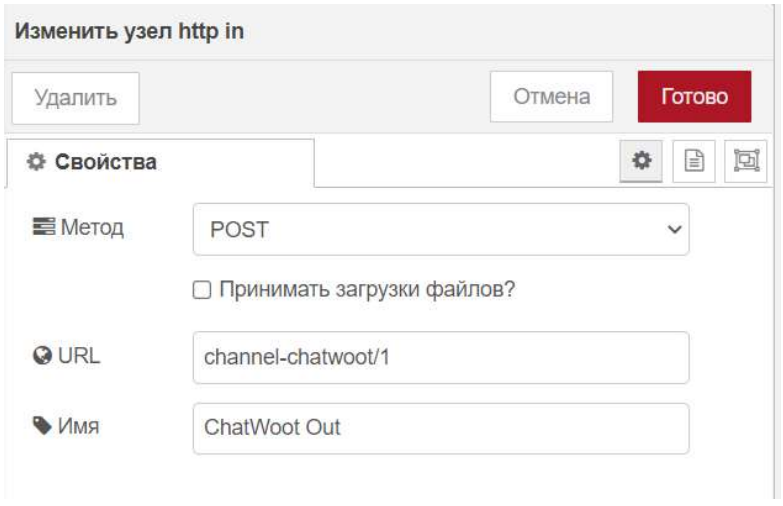


Рис. 3.9 Налаштування точки доступу

Далі створимо функцію для виправлення помилки відображення кнопок, цей функціонал працює не вірно, тому що контент з кнопками відправляється *Agent Bot* до *Chatwoot*, а потім з нього на нашу точку доступу. На стороні

Chatwoot модифікується тіло запиту, тому кнопки працюють не зовсім коректно.

```
let buttons = msg.payload.buttons;
if (!buttons) return msg;
let buttons_transform = [];
const buttonNewline = { type: "newline" };
for (let button of buttons){
    if (!button.value.match(/\d-\d/i)) {
        buttons_transform.push(buttonNewline);
    }
    buttons_transform.push(button);
}
msg.payload.buttons = buttons_transform;
return msg;
```

Далі потрібно підключити вузол *Rules*. Даний вузол буде направляти потік до потрібного вузлу: *Telegram Out, Viber Out*. Налаштування вузла *Rules* представлено на рисунку 3.10.

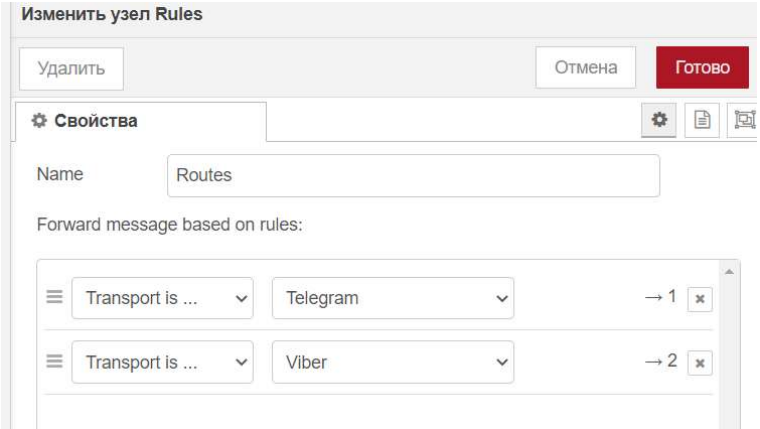


Рис. 3.10. Налаштування вузла *Rules*

Підключаємо відповідні вузли для месенджерів та маємо готовий скрипт. Вигляд скрипту «*Out*» представлено на рисунку 3.11.

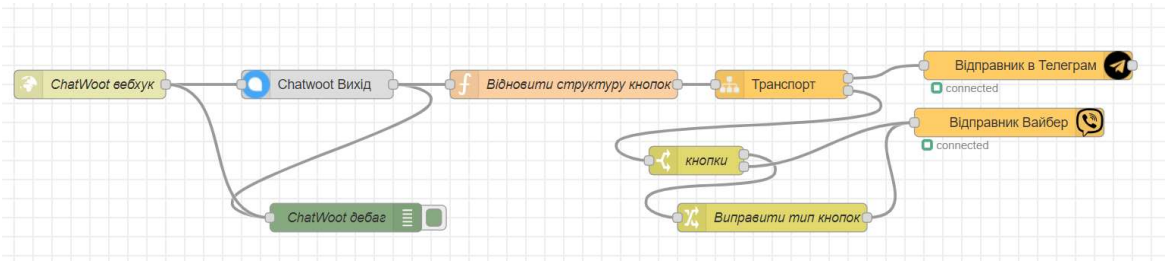


Рис. 3.11 Вигляд скрипту «*Out*»

3.2.3 Розробка скрипту «*Main*»

Основним вузлом в даному скрипті буде *AgentBot In*. Сюди будуть приходити всі повідомлення: вхідні, вихідні та події. В залежності від даних, які будуть поступати в даний вузол буде будуватися основна логіка чат-боту.

Відправимо стартове повідомлення, після чого запитом до *API* перевіряємо наявність тегів в даному діалозі. В залежності від того який тег присвоєно або не присвоєно зовсім видаємо те чи інше повідомлення. Частина скрипту «*Main*» представлена на рисунку 3.12.

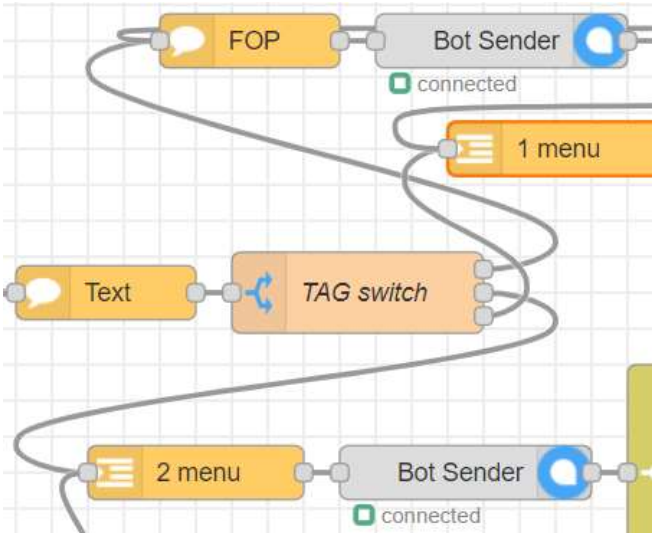


Рис. 3.12 Частина скрипту «*Main*»

Якщо логіка передбачує виставлення тегу, то робимо запит до *API* та присвоюємо відповідний тег в діалог. Призначення тегу в діалог представлено на рисунку 3.13.

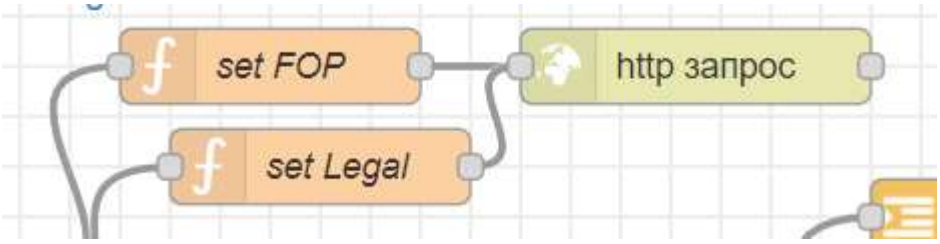


Рис. 3.13 Призначення тегу в діалог

Після кожного вузла *AgentBot* Out потрібно поставити вузол *switch* в якому ми перераховуємо можливі варіанти відповіді користувача. В залежності від відповіді користувача направляємо потік в відповідну логіку. Приклад використання вузла *switch* представлено на рисунку 3.14.

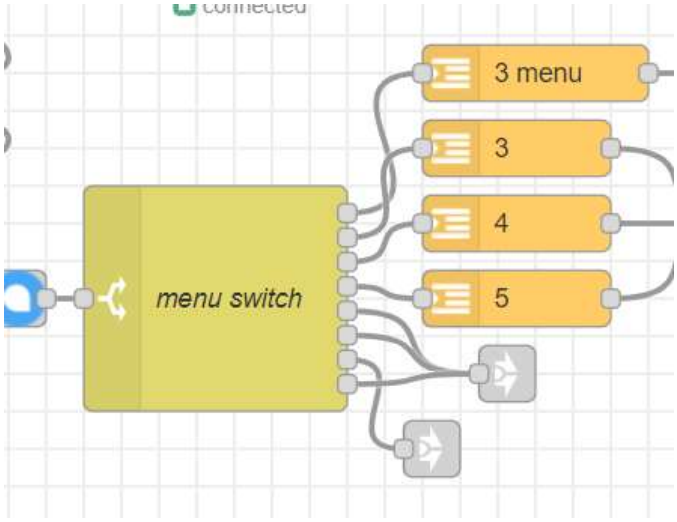


Рис. 3.14 Приклад використання вузла *switch*

Аналогічно будуємо всі інші кроки для нашого чат-боту.

Якщо клієнт натискає на кнопку «Оператор» або вводить не вірні данні, то нам потрібно призначити в діалог оператора та вимкнути логіку чат-боту, щоб не заважати спілкуванню оператора з користувачем. Перевіряємо чи протягом останніх 5 хвилин користувач вже спілкувався з оператором, якщо ні, то призначаємо його на команду підтримки з алгоритмом випадкового розподілення на оператора. Якщо так, тоді перевіряємо чи досі даний оператор має статус «онлайн», якщо так, тоді призначаємо на нього, інакше аналогічно

на команду підтримки. Далі відправляємо повідомлення про те, що користувача було з'єднано з оператором. Логіка присвоєння діалогу на оператора представлена на рисунку 3.15.

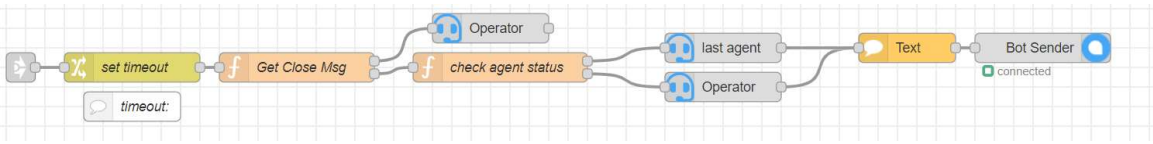


Рис. 3.15 Логіка присвоєння діалогу на оператора

Реалізуємо логіку закриття діалогу при відсутній активності користувача. Зробимо це за допомогою Redis та відповідної бібліотеки. Даний функціонал реалізуємо в вигляді *subflow* «*Resolve Inactive*».

При поступленні повідомлення від клієнту ми робимо запис до *Redis* та ставимо йому час життя рівний визначеному раніше – 10 хв. Якщо даний запис буде видалено по таймауту, тоді прийде подія в вузол де ми на неї підписалися. Тоді ми завершуємо діалог за допомогою запиту до *API*. Якщо діалог був завершений оператором, тоді ми видаляємо запис. Реалізація логіки взаємодії з *Redis* представлена на рисунку 3.16. Вигляд *subflow* в основному скрипті представлено на рисунку 3.17.

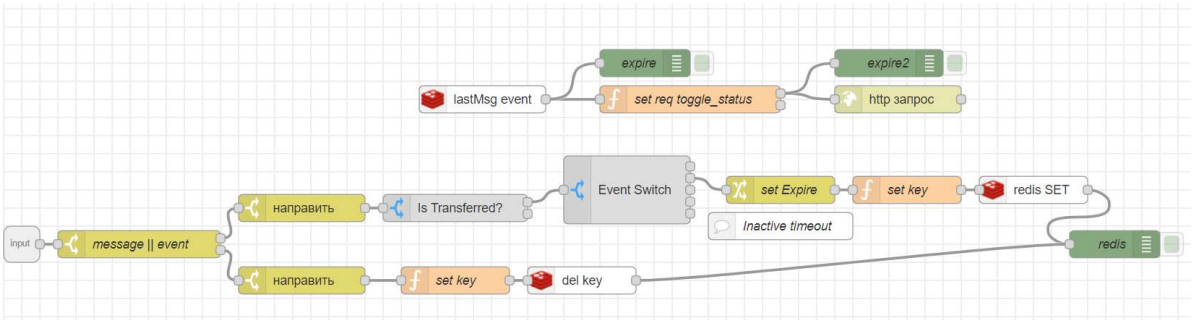


Рис. 3.16 Реалізація логіки взаємодії з *Redis*

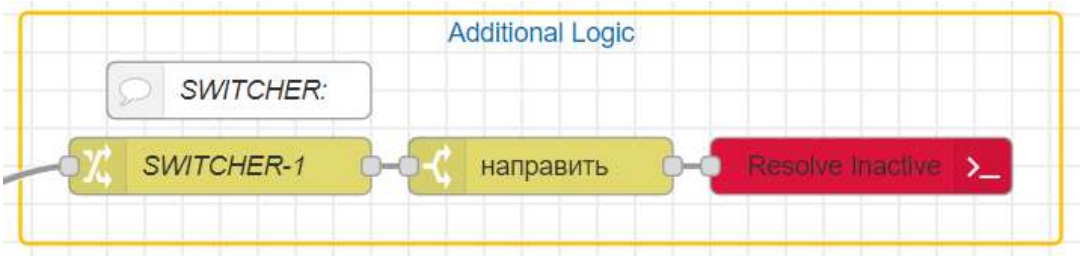


Рис. 3.17 Вигляд *subflow* в основному скрипті

При події завершення діалогу потрібно видалити оператора та команду з діалогу за допомогою запиту до *API*. Логіка при завершені діалогу представлена на рисунку 3.18.

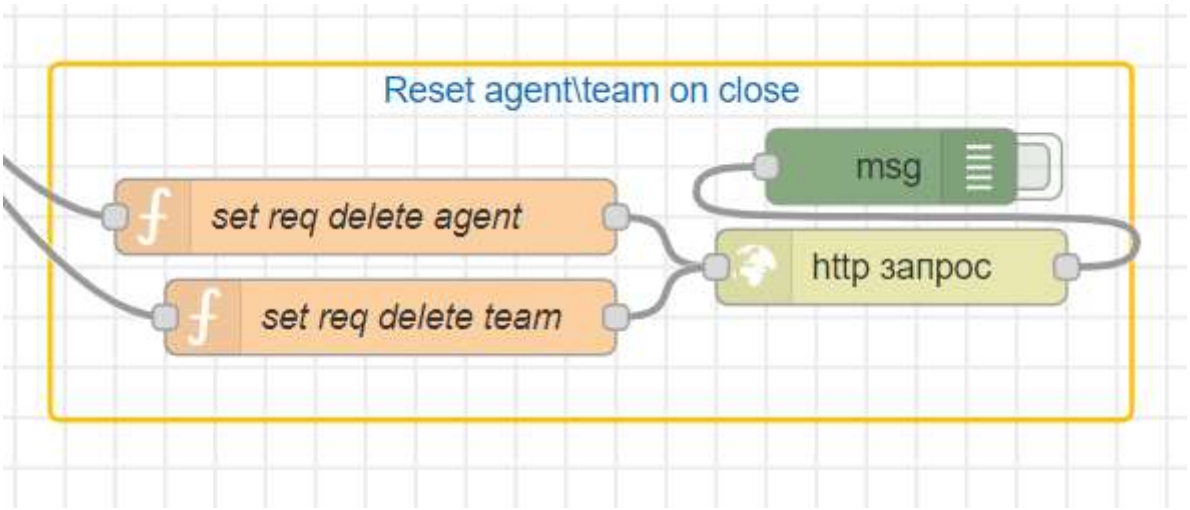


Рис. 3.18 Логіка при завершені діалогу

3.3 Висновки до розділу

Універсальний чат бот був реалізований за допомогою інструментів, які були визначені в попередньому розділі. Було реалізовано три скрипти (*flow*) для *Node-RED*.

Розроблено власний вузол, який реалізує інтеграцію з *Chatwoot API* та дозволяє додавати до діалогу оператора або команду підтримки. Була реалізована, як серверна частина так і візуальна частина даного вузла. Він дозволяє вибрати в параметрах на кого буде призначений діалог – оператора чи команду. Якщо команду дозволяється вибрати також алгоритм розподілення на оператора.

За допомогою розробленого та інших вузлів були реалізовані скрипти *«In», «Out», «Main».*

ВИСНОВКИ

Дипломний проєкт присвячений тематиці універсального чат-боту для банківських послуг на платформі *Node.js*.

Було визначено проблематику проєкту, чому і для яких цілей потрібна дана розробка. Визначено, що реалізація проєкту знизить навантаження на центри підтримки в сфері банківських послуг та покращать враження клієнтів від наданих їм послуг.

Проаналізовано які є типи чат ботів. Визначено для яких ситуацій потрібен той, чи інший тип, їх переваги та недоліки. В кінцевому рахунку вибрано для реалізації класичного скриптового боту, який надає сто відсоткову коректну відповідь.

Проведено аналіз інструментів для розробки. Було обґрунтовано переваги платформи *Node.js*. Визначено, що для реалізації проєкту найкращим вибором будуть інструменти візуального програмування та вибрано для розробки *Node-RED*. Також визначено допоміжні бібліотеки для спрощення розробки чат-боту та взаємодії з нереляційною базою даних *Redis*.

Сформовано функціональні вимоги для проєкту та також перераховано нефункціональні. Було повністю розроблено алгоритм роботи бота з наведеними текстами повідомлень та кнопок. На кожний варіант вибору було прописано відповідні реакції бота, такі як: текстова відповідь, надання документу, переведення діалогу на оператора або завершення діалогу. Сформована візуальне представлення алгоритму дії чат-боту.

Розроблено інтеграцію з сервісом *Chatwoot* за допомогою засобів *Node-RED*. Було створено власні вузли для взаємодії з *API Chatwoot*, такі як переведення діалогу на оператора. Розроблено як і серверну частину вузла, так і візуальну. Зроблена можливість вибору для переводу на команду або оператора. При виборі команди реалізовано також алгоритм розподілення на випадкового оператора.

Реалізовано три скрипти, вони ж *flow* для *Node-RED*.

«*In*» відповідає за підключення необхідних месенджерів, систему тротлінгу та направлення повідомлень до платформи для взаємодії з клієнтами *Chatwoot*.

«*Out*» відповідає за відправлення повідомлень в зворотному напрямку.

«*Main*» відповідає за основну логіку чат-бота. Направляє користувача згідно прописано сценарію. Може надавати йому відповіді або з’єднати його з оператором. Також вміє сам закривати діалоги та з’єднувати користувача з останнім оператором з яким він спілкувався.

Для розробки було використано редактор коду *VS Code.*

Для реалізації схем та діаграм було використано *MS Visio.*

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойчено С В., Іванченко О. В. *Положення про дипломні роботи (проекти) випускників національного авіаційного університету. 2017. – с.63*

2. *You don't know JS* [Електронний ресурс] / *github* – Електрон. дані – Режим доступу https://github.com/azat-io/you-dont-know-js-ru вільний – Назва з екрану – Дата перегляду 21.05.2022.

3. *Node-RED Alternatives Review: best software* [Електронний ресурс] / *compsmag* – Електрон. дані – Режим доступу https://www.compsmag.com/alternative/software/node-red/#node-red-alternative-faq вільний – Назва з екрану – Дата перегляду 21.05.2022.

4. *What is Visual Programming?* [Електронний ресурс] / *outsystems* – Електрон. дані – Режим доступу https://www.outsystems.com/glossary/what-is-visual-programming/ вільний – Назва з екрану – Дата перегляду 22.05.2022.\

5. *Node.js - reasons to use, pros and cons, best practices!* [Електронний ресурс] / *voidcanvas* – Електрон. дані – Режим доступу https://www.voidcanvas.com/describing-node-js/ вільний – Назва з екрану – Дата перегляду 22.05.2022.

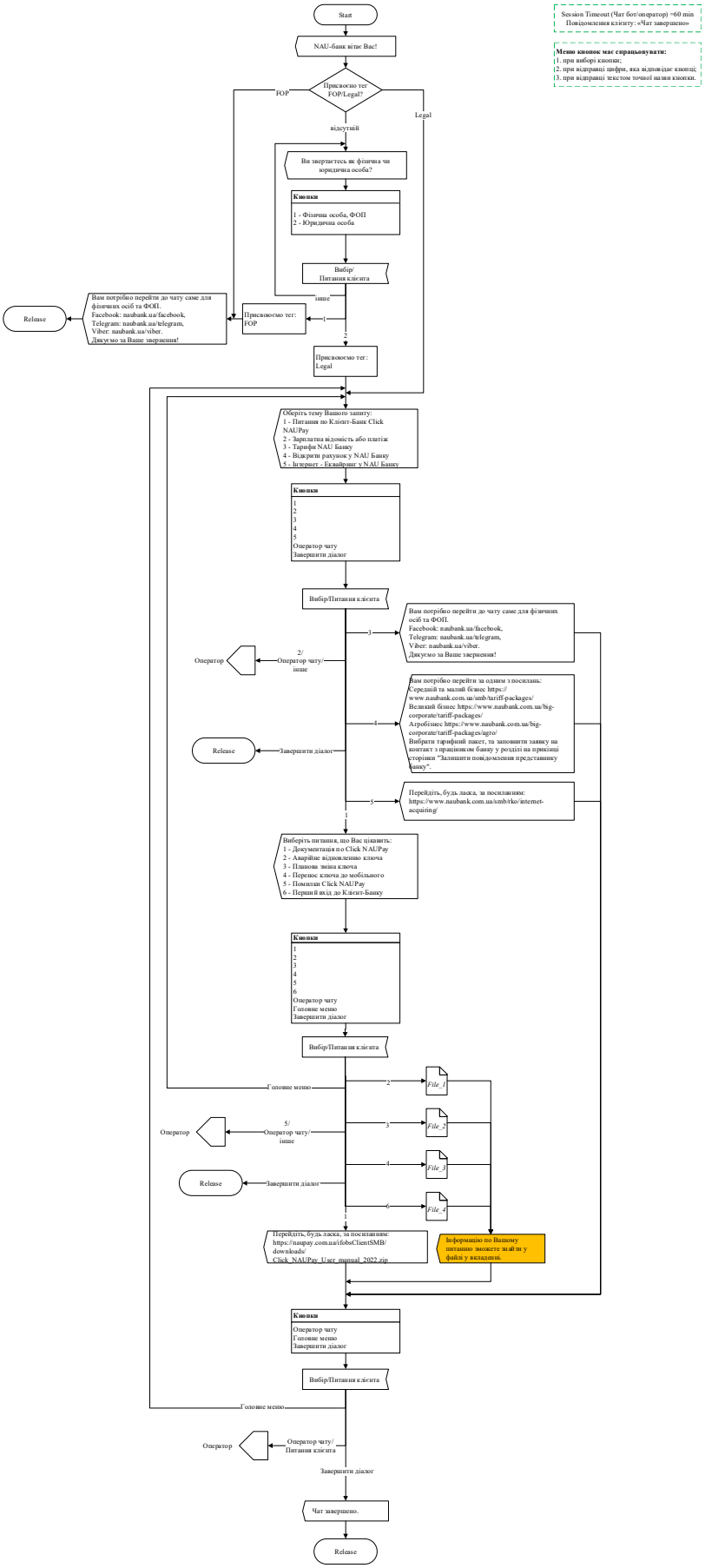
6. *Chatwoot API documentation* [Електронний ресурс] / *chatwoot* – Електрон. дані – Режим доступу https://www.chatwoot.com/developers/api/ вільний – Назва з екрану – Дата перегляду 23.05.2022.

7. *Agent Bots | Chatwoot* [Електронний ресурс] / *chatwoot* – Електрон. дані – Режим доступу https://www.chatwoot.com/developers/api/ вільний – Назва з екрану – Дата перегляду 23.05.2022.

8. ДСТУ 3008-95 Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Видавничий дім «Держстандарт України», 1995. – 88с

Додаток А

Наповнення чат-боту та можливі відповіді



Додаток Б

Лістинг функції oneditprepare

```
oneditprepare: async function () {
    let node = this;
    // let currentBotId = node.bot;
    let transferTargetId = node.transferTargetId;

    $('.form-row-assign').hide();

    function deleteOptions() {
      // delete options from select
      $('#node-input-transferTargetId option').each(function () {
        if ($(this).val() !== '') {
          $(this).remove();
        }
      });
    }

    function addOptionsFromBody(body) {
      if (typeof body === 'string') body = JSON.parse(body);
      if (body != null) {
        for (let element of body) {
          let selected = element.id == transferTargetId ? 'selected' : '';
```

```
                $('#node-input-transferTargetId').append(
                  '<option ' + selected + ' value="' + element.id + '">' + element.name
+ '</option>'
                );
              }
            }
          }

          let nodeRedUrl = $.RedBot.getNodeRedUrl();

          $('#node-input-transferTarget').change(function () {
            $('.form-row-assign').hide();
            if ($(this).val() === 'team') {
              $('.form-row-assign').show();
            }
            let currentBotId = node.bot;
            if (currentBotId) {
              deleteOptions();
              const chatwootUrl =
                $(this).val() === 'team'
                  ? nodeRedUrl + 'redbot/chatwoot-botagent/get_teams?id=' +
currentBotId
```

```
                : nodeRedUrl + 'redbot/chatwoot-botagent/get_agents?id=' +
currentBotId;
                $.get(chatwootUrl)
                  .done(addOptionsFromBody)
                  .fail((err) => {
                    addOptionsFromBody([]);
                    RED.notify('Could not reach ChatWoot API', { type: 'error', timeout:
5000 });
                  });
              }
            });
          }
```

Додаток В

HTML розмітка вікна редагування

```
<script type="text/html" data-template-name="chatbot-chatwoot-botagent-transfer">
  <div class="form-row">
    <label for="node-input-name">Name</label>
    <input type="text" id="node-input-name" placeholder="Name" />
  </div>
  <div class="form-row">
    <label for="node-input-transferTarget">Transfer target type</label>
    <select id="node-input-transferTarget" placeholder="Select target type">
      <option selected value></option>
      <option value="agent">Agent</option>
      <option value="team">Team</option>
    </select>
  </div>

  <div class="form-row">
    <label for="node-input-transferTargetId">Transfer target</label>
    <select id="node-input-transferTargetId">
      <option value></option>
    </select>
  </div>
```

```
<div class="form-row form-row-assign">
  <label for="node-input-assignmentAlgorithm">Assignment algorithm</label>
  <select id="node-input-assignmentAlgorithm">
    <option value="none" selected>None</option>
    <option value="RAA">RAA</option>
  </select>
</div>

<div class="form-row form-row-bot">
  <label for="node-input-bot">ChatWoot BotAgent</label>
  <input type="text" id="node-input-bot" placeholder="Bot" />
</div>
</script>
```

Додаток Д

Алгоритм відповідей чат-боту