НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

УДК 04.089

«До захисту допущено»
Завідувач кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Дипломний проект
на здобуття ступеня бакалавра

з напряму підготовки *6.050101 «Комп’ютерні науки»*

на тему: «Комплекс задач з підтримки користувачів з використанням Сентиментального аналізу даних»

Виконав : Студент *4* курсу, групи *ІС-51*

Медведніков Даніїл Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н. Гриша Олена Василівна* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц., к.т.н. Тєлишева Тамара Олексіївна* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) *6.050101*

«Комп’ютерні науки» («Інформаційні управляючі системи та технології»)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)

“___”_____2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Медведнікову Данііlу Сергійовичу
(прізвище, ім`я, по батькові)

1. Тема проекту «Комплекс задач з підтримки користувачів з використанням сентиментального аналізу даних»
керівник проекту *Гриша Олена Василівна, доц.*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*23*”*квітня* 2019 р. №1181-с

2. Термін подання студентом проекту “*03*”*червня* 2019 року

3. Вихідні дані до проекту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів
5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна діяльності

2. Схема структурна варіантів використання

3. Схема бази даних

4. Схема структурна розгортання

5. Схема структурна послідовності

6. Рішення з математичного забезпечення

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *«23» квітня 2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	*Вивчення рекомендованої літератури*	*01.03.2019*	
2.	*Аналіз існуючих методів розв'язання задачі*	*29.03.2019*	
3.	*Постановка та формалізація задачі*	*16.04.2019*	
4.	*Розробка інформаційного забезпечення*	*23.04.2019*	
5.	*Алгоритмізація задачі*	*30.04.2019*	
6.	*Обґрунтування використовуваних технічних засобів*	*05.05.2019*	
7.	*Розробка програмного забезпечення*	*10.05.2019*	
8.	*Налагодження програми*	*20.05.2019*	
9.	*Виконання графічних документів*	*22.05.2019*	
10.	*Оформлення пояснювальної записки*	*27.05.2019*	
11.	*Подання ДП на попередній захист*	*30.05.2019*	
12.	*Подання ДП на основний захист*	*03.06.2019*	
13.	*Подання ДП рецензенту*	*05.06.2019*	

Студент _____ Д.С. Медведніков
(підпис)

Керівник проекту _____ О.В. Гриша
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
			Документи текстові		
1	А4	ДП ІС-5115.1181-с.ПЗ	Пояснювальна записка	90	
2	А4	ДП ІС-5115.1181-с.ТЗ	Технічне завдання	11	
3	А4	ДП ІС-5115.1181-с.ПМВ	Програма та методика випробувань	12	
			Матеріали графічні		
4	А3	ДП ІС-5115.1181-с.ССД	Схема структурна діяльності	1	
5	А3	ДП ІС-5115.1181-с.ССС	Схема структурна варіантів	1	
			використання		
6	А3	ДП ІС-5115.1181-с.СБД	Схема бази даних	1	
7	А3	ДП ІС-5115.1181-с. ССР	Схема структурна розгортання	1	
8	А3	ДП ІС-5115.1181-с. ССП	Схема структурна послідовності	1	
9	А3		Рішення з математичного забезпечення	1	

Зм.	Арк.	ПІП	Підп.	Дата	ДП ІС-5115.1181-с.ВДП		
Розробн.		Медведніков Д.			Відомість дипломного проекту	Літ.	Лист 1 / Листів 1
Керівн.		Гриша О.В.					
Консульт.						КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51	
Н/контр.		Тєлишева Т.О.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проекту

на тему: Комплекс задач з підтримки користувачів з використанням сентиментального аналізу даних

Київ – 2019 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з п’яти розділів, містить 25 рисунків, 14 таблиць, 1 додаток, 40 посилань.

В дипломному проекті реалізована тема «Комплекс задач з підтримки користувачів з використанням сентиментального аналізу даних» метою якої є підвищення якості та швидкості надання підтримки користувачам продукту.

Для сентиментального аналізу даних було використано метод правил та словників.

В результаті виконання роботи було створено систему, що дозволяє надавати підтримку користувачам у більш зручному форматі та з використанням сентиментального аналізу тексту.

Практичне значення роботи полягає у тому, що отриману систему можна використовувати у службах надання підтримки з метою підвищення якості.

ПІДТРИМКА, ТЕХНІЧНА ПІДТРИМКА, МЕСЕНДЖЕР, ЧАТ-БОТ, АНАЛІЗ ТОНАЛЬНОСТІ, ВЕБ-ЗАСТОСУВАННЯ.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of five sections, containing 25 figures, 14 tables, 1 application, 40 sources.

The diploma project implemented the theme «Software Solution for Users Support with the Help of Sentimental Data Analysis» aimed at improving quality and speed of customer's support process.

The rule-based method was used to implement sentimental data analysis.

The result of this work is system that allows to give users support in more comfortable way with the usage of sentimental data analysis.

Practical use of this work is that the system could be used in customer's support in order to improve quality of their work.

SUPPORT, TECHNICAL SUPPORT, MESSENGER, CHAT BOT, SENTIMENT ANALYSIS, WEB-APPLICATION.

ЗМІСТ

ВСТУП

Ми живемо в час інформаційних технологій та загальної масової комп’ютерізації. Інформатизація суспільства - це глобальний соціальний процес, основною особливістю якого є те, що домінуючим видом діяльності в сфері суспільного виробництва є збір, накопичення, передача, обробка та використання інформації на основі сучасних засобів обчислювальної техніки.

Інструментами, за допомогою яких відбувається робота з інформацією стає все більш складною. Цими інструментами можуть бути як технічні засоби, такі як комп’ютери, телефони, планшети, так і програмне забезпечення, інтернет-сервіси та інше. Але не кожен з користувачів володіє необхідним набором навичок та знань для того, щоб самостійно розібратися з тим, як користуватись певним продуктом, або вирішити проблемі, які можуть з’явитися під час використання. Для вирішення даних типів проблем існує служба підтримки користувачів.

На сьогоднішній день, робочий процес служби підтримки не завжди є оптимізованим та зручним як для користувачів, так і для працівників служби підтримки.

Мета розробки - підвищити якість надання технічної підтримки користувачам продукту шляхом використання чат-ботів популярних месенджерів та сентиментального аналізу даних.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Підтримка користувачів – сервісна структура, що вирішує проблеми клієнтів з сервісами, програмним або апаратним забезпеченням, що надає компанія.

Більшість компаній, що надають послуги, мають службу підтримки клієнтів, яка забезпечує супроводження продукту, що продається. Послуги підтримки надаються, як правило, по телефону, через Інтернет, за допомогою електронної пошти, онлайн-сервісів підтримки на веб-сайті. Також джерелом безкоштовної технічної підтримки є Інтернет, де досвідчені користувачі на різноманітних форумах та веб-сайтах можуть порадити, поділитися досвідом, допомогти у вирішенні таких самих або схожих проблем. Крім того, деякі компанії надають преміальну технічну підтримку за додаткову плату.

Великі організації мають власну службу підтримки для забезпечення відповідними сервісами свого персоналу в разі неможливості впоратися з проблемами продукту.

Найбільш розповсюдженим шляхом надання підтримки на даний момент є так звані HelpDesk’и – програмне забезпечення, завдяки якому клієнт може створити запит до служби підтримки, отримувати відповіді, відслідковувати шлях його запиту від початку і до моменту вирішення. Одними з найбільших недоліків даних систем є те, що клієнт має проходити ще одну реєстрацію на сайті HelpDesk’у, а також великий час очікування відповіді щодо вирішення своєї проблеми, внаслідок того, що звернення звичайно проходить через декілька етапів до того, як буде вирішено.

Останнім часом все більш популярним методом надання підтримки стають програми-месенджери (Telegram[1], Viber[2], Facebook Messenger[3], WhatsApp[4]). Таким чином, клієнт сам обирає месенджер для підтримки та отримує її у найбільш зручному форматі. Компанії, в свою чергу, мають

можливість зекономити кошти, які були би необхідні для розробки та підтримки платформи формату HelpDesk, так як більша частина функціоналу системи надається месенджерами. Більшість з месенджерів мають відкриті безкоштовні програмні інтерфейси[5-7] (англ. application program interface, API), що надає змогу розробникам створювати інтеграції та вводити новий функціонал у системи технічної підтримки. Організації не повинні тримати потужних серверів для зберігання та обробки великої кількості даних, бо цю частину також беруть на себе месенджер-сервіси.

Сентиментальний аналіз даних – клас методів контент-аналізу в комп'ютерній лінгвістиці, що призначений для автоматизованого виявлення в текстах емоційного забарвлення та емоційної оцінки.

Тональність – це емоційне відношення автора вислову до деякого об'єкту. Тональність усього тексту можна визначити як функцію лексичних тональностей одиниць, з яких він складається та правил їх поєднання (у найпростішому випадку – сумі).

Основною метою аналізу тональності є знаходження думки в тексті та виявлення її властивостей. Приклади тональних оцінок: позитивна, негативна, нейтральна. Під "нейтральною" розуміється, що текст не має емоційного забарвлення. Також можуть існувати і інші тональні оцінки.

Використання сентиментального аналізу текстових даних у системі підтримки клієнтів, надає змогу визначити клієнтів, у зверненнях яких є згадки про проблеми з продуктом або негативний настрій, та надати їм підтримку у найкоротший строк, в результаті чого бізнес зможе зберегти себе від збитків внаслідок втрачання клієнтів.

Також сентиментальний аналіз може бути використано для автоматичної побудови статистики за опитуванням користувачів (наприклад "чи задоволені ви продуктом?").

1.1.1 Опис процесу діяльності

Опис процесу діяльності представлено у вигляді двох процесів.

Перший процес – налаштування та створення опитування. Схема структурна діаграми діяльності наведена у графічному матеріалі до роботи.

Адміністратор за допомогою веб-інтерфейсу застосунку створює розсилку-опитування, у вигляді повідомлення, що буде доставлено користувачу (клієнту) у месенджер, котрим він користується (Telegram, Viber або Facebook Messenger). Після цього, адміністратору відображається статус повідомлення.

Другий процес – звернення клієнта у підтримку та під’єднання до нього оператора (рис. 1.2).

Клієнт, використовуючи зручний для нього месенджер, відправляє повідомлення чат-боту системи підтримки. Система проводить сентиментальний аналіз повідомлення та відображає його результати оператору у веб-застосунку. На основі результатів аналізу, оператор приймає рішення щодо під’єднання в чат та надання подальшої підтримки.

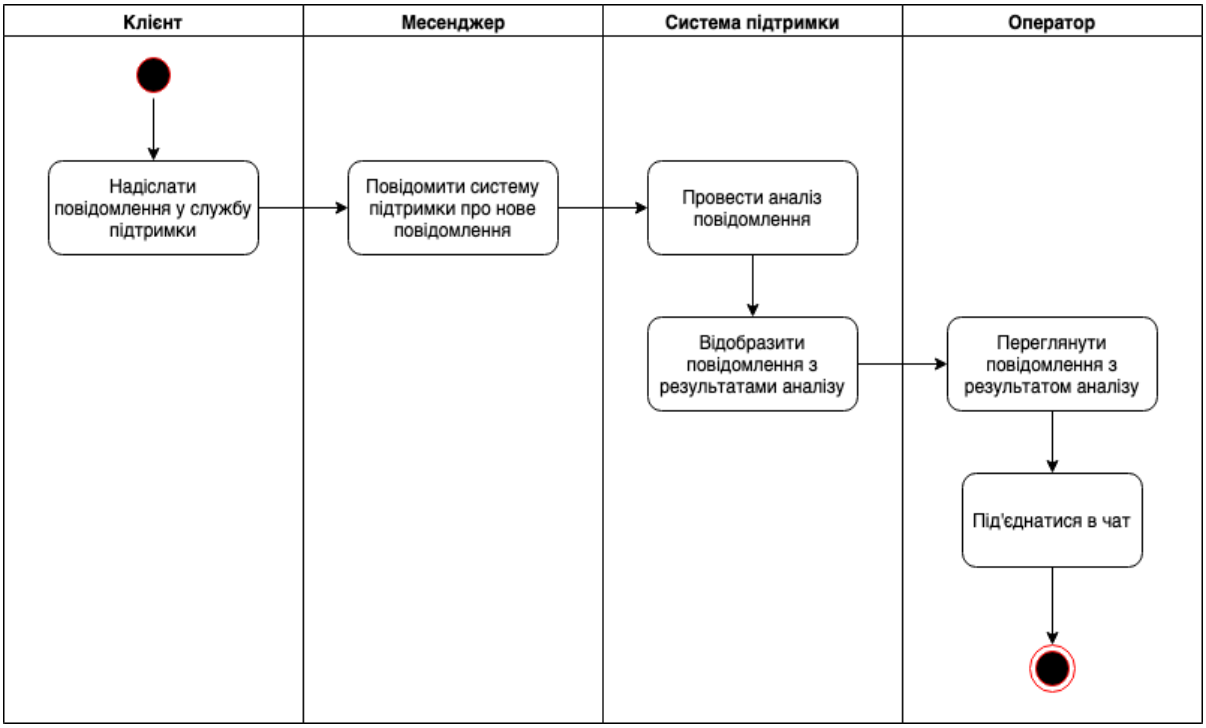

Рисунок 1.1 – Activity діаграма для звернення клієнта у підтримку та під’єднання до нього оператора

1.1.2 Опис функціональної моделі

Специфікацію функціональної поведінки системи представимо у вигляді діаграми варіантів використання. Схему структурну діаграми варіантів використання наведено у графічному матеріалі до роботи. На ній зображено всі функції системи та описано акторів, які будуть їх використовувати.

Безпосередньо із системою будуть взаємодіяти два актори:

- адміністратор – це особа, яка може налаштовувати та створювати опитування;
- оператор – працівник служби підтримки, який надає підтримку клієнтам

Відповідно до визначених варіантів використання виявлено функціональні вимоги та встановлено їх пріоритетність. Результат наведено в таблиці 1.1.

Таблиця 1.1 – Функціональні вимоги

Варіант використання	**Функціональна вимога**	**Пріоритет**
Створення опитувань	Система надає можливість адміністратору створити нове опитування.	Високий
Розсилка по базі клієнтів	Система надає можливість адміністратору створити розсилку по базі клієнтів.	Високий
Перегляд статистики	Система надає можливість адміністратору переглядати статистику попередніх опитувань.	Високий
Формування статистики	Система надає можливість сформувати статистику за обраними параметрами.	Високий
Аналіз повідомлень	Система надає можливість проводити сентиментальний аналіз відповідей клієнтів.	Високий

Продовження таблиці 1.1

Надання підтримки	Система надає можливість оператору надавати підтримку користувачам.	Високий
Надсилання повідомлень	Система надає можливість оператору надсилати повідомлення клієнтам.	Високий
Отримання повідомлень	Система надає можливість оператору отримувати повідомлення від клієнтів.	Високий

1.2 Огляд наявних аналогів

Перед початком роботи над дипломним проектом було здійснено пошук застосунків зі схожою функціональністю.

Нині існують програмні продукти (табл. 1.2), які надають можливість використовувати чат-ботів популярних месенджерів для комунікації з клієнтами. Переважно вони використовуються для автоматизації процесу продажу продукту, але жоден з них не спеціалізується на підтримці, не надає ролі користувачам системи, та не має функціоналу сентиментального аналізу повідомлень. Також у багатьох організацій існують закриті системи підтримки з використанням месенджерів (наприклад Monobank[8] в Україні)

Таблиця 1.2 – Список аналогів

Назва	**Режим доступу**
ManyChat[9]	https://manychat.com/
Leeloo[10]	https://leeloo.ai/
TextBack[11]	https://textback.ru/

Проаналізуємо функції, які виконують зазначені вище програмні продукти.

Таблиця 1.3 – Огляд функцій наявних аналогів

Функції	ManyChat	Leeloo	Textback
Підтримка Facebook Messenger	+	+	+
Підтримка Telegram	-	+	+
Підтримка Viber	–	–	+
Чат з клієнтами	+	+	+
Опитування клієнтів	+	+	+
Сентиментальний аналіз повідомлень	–	–	–

Із таблиці 1.3 видно, що ті аналоги, які представлені нині на ринку, не охоплюють увесь спектр функцій, необхідних для надання підтримки клієнтам.

Більшість з аналогів зосереджена на процесі реклами продуктів, побудові воронок продажу та інших функціях. Жоден з аналогів не надає функції сентиментального аналізу повідомлень.

1.3 Постановка задачі

1.3.1 Призначення розробки

У зв’язку з тим, що процеси надання підтримки користувачам за допомогою мережі Інтернет на даний момент є незручними та неоптимізованими, виникла мета підвищити якість надання підтримки, а саме: створити систему, орієнтовану на надання підтримки користувачам у зручному для них форматі. Клієнти зможуть користуватися одним з більш зручних для них месенджером (Telegram, Viber або Facebook Messenger), задавати свої питання службі підтримки. Працівники служби підтримки (адміністратор на оператор), в свою чергу, зможуть використовувати результати сентиментального аналізу текстових повідомлень клієнтів для надання їм підтримки.

1.3.2 Цілі та задачі розробки

Цілями розробки є:

- покращення якості надання підтримки;
- зменшення часу очікування для отримання відповіді;
- зробити процес отримання підтримки біль зручним для клієнта;

Для реалізації поставленої мети необхідно розв’язати наступні задачі:

- створення засобів для створення опитувань та комунікації з клієнтами.
- створення інтеграцій з месенджерами Telegram, Viber та Facebook Messenger.
- створення засобів сентиментального аналізу текстових даних.

Висновок до розділу

У даному розділі було розглянуто предметне середовище. Дипломний проект присвячений підтримці клієнтів з використанням чат-ботів та сентиментального аналізу текстових даних.

При описі предметного середовища було визначено та описано процеси діяльності, що представлені у вигляді двох незалежних процесів, для кожного з яких наведено структурні схеми. Також було створено функціональну модель системи, а саме: побудовано USE-case діаграму, описано акторів системи та визначено функціональні вимоги з пріоритетами виконання відповідно до варіантів використання.

Наступним етапом роботи над дипломним проектом став пошук аналогів запропонованого застосунку з подібним функціоналом. Було порівняно функції, які виконують знайдені програмні продукти. На даний момент було виявлено декілька систем зі схожим функціоналом, але вони не охоплюють увесь спектр функцій, які запропоновані в даному дипломному проекті. Жоден з аналогів продуктів не використовує сентиментальний аналіз текстових даних.

Також у одному з підрозділів було сформовано постановку задачі, визначено призначення, мету та задачі розробки.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Вхідними даними комплексу задач є:

- дані про клієнтів,
- повідомлення з месенджерів.

Так як темою роботи є саме комплекс задач з підтримки користувачів, а не створення такої системи цілком, будемо вважати, що дані про клієнтів у відповідному форматі вже зберігаються у базі даних.

Повідомлення з месенджерів надходять до системи за допомогою вебхуків, що надсилаються на сервер системи по протоколу HTTPS[12] у форматі JSON[13].

Після підключення чат-боту до системи підтримки, система підтримки відправляє запит на зовнішній API відповідного месенджеру та передає IP-адресу сервіса системи підтримки, що відповідає за прийняття та обробку повідомлень від месенджерів. Вебхук-повідомлення з різних месенджерів мають різну структуру, приклади наведено на Рисунках 2.1 - 2.3, але задля полегшення взаємодії, вони перетворюються в єдиний формат попередньою обробкою. Приклад повідомлення показано на рисунку 2.1.

```
{
    "message": {
        "chat": {
            "id": "123123"
        },
        "text": "Hello world!"
    }
}
```

Рисунок 2.1 – Структура повідомлення з месенджера Telegram

- message – об’єкт, що містить інформацію про повідомлення
- chat – об’єкт, що містить інформацію про чат
- id – унікальний ідентифікатор чата
- text – текст повідомлення

Рисунок 2.2 – Структура повідомлення з месенджера Viber

- sender – об’єкт, що містить інформацію про відправника повідомлення
- id – унікальний ідентифікатор відправника
- name – ім’я відправника
- avatar – посилання на аватар відправника, що зберігається на серверах Viber
- message - об’єкт, що містить інформацію про повідомлення
- text - текст повідомлення

Рисунок 2.3 – Структура повідомлення з месенджера Facebook Messenger

- entry – об’єкт-контейнер, що містить усі оновлення, що надсилаються вебхуком (при великій частоті надходження повідомлень можливе надсилання кількох оновлень від кількох різних користувачів одним вебхуком)
- messaging – об’єкт-контейнер, що містить усі нові повідомлення у вигляді об’єктів
- sender - об’єкт, що містить інформацію про відправника
- recipient - об’єкт, що містить інформацію про отримувача повідомлення
- id - унікальні ідентифікатори відправника та отримувача відповідно
- name - імена відправника та отримувача відповідно
- message - об’єкт, що містить інформацію про повідомлення
- text - текст повідомлення

Задля полегшення подальшої взаємодії з повідомленнями, вони проходять попередню обробку та приводяться до єдиного формату, що зображено на рисунку 2.4.

```
{
    "message_id": 123,
    "from": 338123942,
    "date": 153819302,
    "chat": "chat:329381393",
    "text": "Hello world"
}
```

Рисунок 2.4 – Структура повідомлення у форматі JSON

- message_id – id повідомлення
- from – id користувача, що надіслав повідомлення
- date – дата та час надсилання повідомлення у форматі Unix time
- chat – id чата
- text – текст повідомлення

2.2 Вихідні дані

Вихідними даними комплексу задач, є повідомлення від клієнтів у форматі, зазначеному у попередньому підрозділі, до яких додане поле результату сентиментального аналізу analysis_result. Результатом аналізу є ціле число, від’ємні числа - негативна оцінка, додатні - позитивна оцінка, нуль - нейтральна оцінка. Чим більший модуль числа, тим більш позитивною (негативною) є оцінка повідомлення.

2.3 Опис структури бази даних

Структуру бази даних наведено у вигляді ER-діаграми, яка представлена у графічному матеріалу до роботи.

На рис. 2.5 наведено схему БД, компактнішу нотацію ER-діаграми, де у прямокутниках множин сутностей містяться переліки атрибутів.

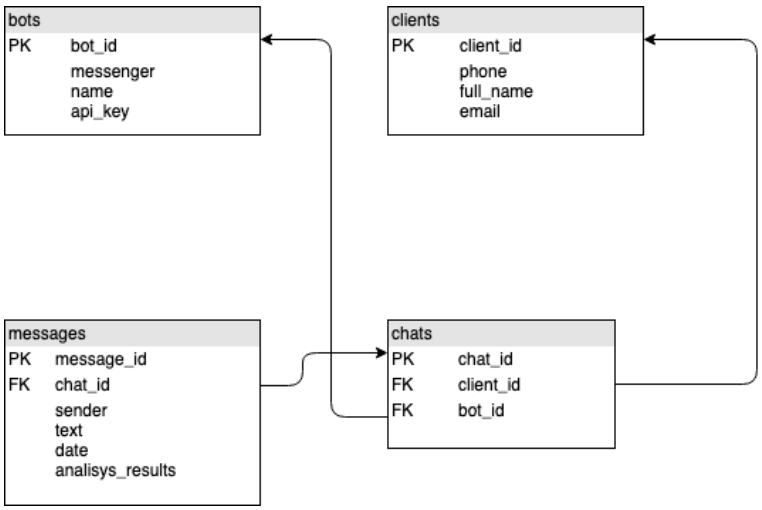

Рисунок 2.5 – Схема БД

Розпишемо детальніше вміст таблиць БД (табл. 2.1).

Таблиця 2.1 – Опис структури бази даних

Назва таблиці	**Назва стовпця**	**Тип даних**	**Призначення**
chats	chat_id	varchar	Унікальний код чату
	client_id	int	Унікальний код клієнта
	bot_id	varchar	Унікальний код бота
messages	message_id	int	Унікальний код повідомлення
	chat_id	varchar	Унікальний код чату
	sender	varchar	Ідентифікатор, від кого надійшло повідомлення (bot або client)
	text	varchar	Текст повідомлення
	date	datetime	Дата та час відправлення
	analysis_results	varchar	Результати аналізу повідомлення
bots	bot_id	varchar	Унікальний код бота
	messenger	varchar	Назва месенджеру (telegram, viber, facebook)
	name	varchar	Ім’я бота
	api_key	varchar	Ключ API
clients	client_id	int	Унікальний код клієнта
	phone	varchar	Номер телефону клієнта
	full_name	varchar	Прізвище, ім’я, по-батькові клієнта
	email	varchar	Адреса електронної пошти клієнта

Висновок до розділу

У розділі з описом інформаційного забезпечення було описано вхідні дані комплексу задач: повідомлення, що надходять у форматі JSON шляхом веб-хуків.

Для даного дипломного проекту було прийнято рішення використовувати реляційну БД PostgreSQL[14].

Також у розділі було наведено схему та структуру БД, детально розписано структуру таблиць, а також зв’язки між ними.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Сентиментальний аналіз текста (англ. Sentiment analysis, Opinion mining) - сфера обробри природніх мов, основа задача якою - визначити суб'єктивні або емоційні індикатері в тексті, тобто знайти відношення автора до зазначенного об'єкта або події[15]. Насьогодні, аналіз тональності текста широко використовується в таких галузях як соціологія (збір даних з соціальних мереж про вподобання людей), політологія (збір даних про політичні погляди окремих соціальних груп), маркетинг (створення рейтингів продуктів, компаній, людей), психологія (визначення симптомів психологічних захворювань або депресії), та, звісно, штучний інтелект, для якого розуміння людських почуттів та емоцій є дуже важливим аспектом[16-20].

Аналіз висловлювань є ефективним засобом спостереження і оцінки думок користувачів та може бути доцільно використаним у системі консультаційної та технічної підтримки клієнтів.

В наукових дослідженнях описуються наступні задачі сентиментального аналізу тексту відгуків користувачів:

- аналіз тональності текстів у висловлюваннях відносно аспектів
- виділення оціночних словосполучень слів
- класифікація текстів на рівні документів і речень

Поставлена задача: виділити відгуки, що вказують на проблеми з продуктом, та навпаки - відгуки, в яких продукту надається позитивна оцінка, використовуючи множина повідомлень користувачів. Надати кожному відгуку оцінку відповідно результатам аналізу (позитивна, негативна, нейтральна).

3.2 Математична постановка задачі

Дано множину повідомлень D:

$$D = \{d1, d2, \dots d|D|\}$$

Функція класифікації тональності лексичних одиниць, що складають множину повідомлень Ф:

$$\Phi(d_j) = \begin{cases} [-5; -1], \text{якщо оцінка негативна} \\ 0, \text{якщо оцінка нейтральна} \\ [1; 5], \text{якщо оцінка позитивна} \end{cases}$$

Необхідно визначити значення цільової функції f, яка являє собою суму оцінки лексичних тональностей одиниць $\Phi(d_j)$. Тобто:

$$f(D) = \sum_{i=1}^{D} \Phi(d_j)$$

Додатнє значення функції f являє собою позитивну оцінку, від'ємне значення - негативну. Якщо значення функції дорівнює нулю, то текст є тонально нейтральним.

3.3 Опис методів розв'язання

Існує декілька видів інструментів для сентиментального аналізу даних. Перший з них - сентиментальні карти (англ. sentiment maps). Ці карти представляють загальний настрій певної соціальної групи в певний момент часу. Карти зазвичай базуються на позитивних або негативних постах у мікроблогах.

Ще один популярний вид сентиментального аналізу це об'єкто-центричний (англ. entity-centric) метод. Цей тип аналізу представляє рейтинг певної організації, людини, продукту або події в певний момент часу. Ця інформація зазвичай збирається з соціальних мереж, мікроблогів та веб-сайтів, що мають систему відгуків.

Види класифікації

В сучасних системах автоматичного визначення емоційної оцінки тексту частіше за все використовується одновимірний емотивний простір: позитив або негатив. Основною задачею в аналізі тональності є класифікація полярності даного документа, тобто визначення, чи є думка у документі чи вислові позитивною, негативною або нейтральною. Більш розгорнуто класифікація тональності виражається, наприклад, такими емоційними станами як “злий”, “сумний”, “щасливий”.

Класифікація за бінарною шкалою

Полярність документа можна визначати за бінарною шкалою. В цьому випадку для аналізу полярності документа використовуються дві оцінки: позитивна та негативна. Одним з недоліків даного підходу є те, що не завжди емоційну складову документа можна визначити однозначно, тобто документ може містити ознаки як позитивної, так і негативної оцінки.

Класифікація за багатосмуговою шкалою

Можна класифікувати тональність документа за багатосмуговою шкалою, що було предприйнято Пангом та Снайдером. Їми було розширено основну задачу класифікації кіновідгуків від оцінка “позитивний або негативний” в бік прогнозування рейтинга по 3-х та 4-х бальній шкалі.

Системи шкалювання

Іншим методом визначення тональності є використання систем шкалювання, за допомогою чого, словам зазвичай пов’язаним з негативною, позитивною або нейтральною тональностями, ставляться у відповідність числа за шкалою від -10 до 10. Спочатку фрагмент неструктурованого тексту досліджується за допомогою інструментів та алгоритмів аналізу природної мови, а потім виділені з цього тексту вислови та терміни аналізуються з метою розуміння значення цих слів.

Методи класифікації тональності

Методи, що засновані на правилах та словниках

Цей метод базується на пошуку емотивної лексики в тексті по заздалегідь складеним тональним словником, та правилам з використанням лінгвістичного аналізу. За сукупністю знайденої емотивної лексики текст можна оцінити за шкалою. Для того щоб оцінити текст за цим методом необхідно кожному слову в тексті присвоїти значення його тональності зі словника, а потім порахувати загальну тональність, підсумувавши значення тональностей кожного окремого речення.

Основною проблемою методів, заснований на правилах та словниках, вважається трудоємність процесу підготовки словника. Для того, щоб отримати інструмент для класифікації тональності документа з високою точністю, необхідний словник, що буде мати оцінку термінів адекватну предметній області документу. Наприклад, слово “величезний” може бути позитивним щодо розміру пам’яти жорсткого диску та негативним щодо розміру мобільного телефону. Також даний метод потребує великих трудовитрат для складання великої кількості правил, необхідних для коректної роботи системи.

Машинне навчання з вчителем

Цей метод є найбільш широко використовуваним. Сутью таких методів є те, що на першому етапи навчається машинний класифікатор на заздалегідь розмічених текстах, а потім отриману модель використовують для аналіза нових документів. Приклад алгоритму:

1. Спочатку збирається колекція документів, на основі яких навчається машинний класифікатор
2. Кожен документ розкладається у вигляді вектора аспектів, за якими він буде досліджуватися
3. Вказується правильний тип тональності для кожного документа

4. Відбувається вибір алгоритму класифікації та методу для навчання класифікатора
5. Отриману модель використовуємо для визначення тональності нової колекції документів

Машинне навчання без вчителя

В основі цього методу покладено те, що терміни, які найчастіше зустрічаються у тексті і в той же час присутні у меншій кількості в інших документах цієї колекції, мають найбільшу вагу в тексті. Виділивши дані терміни, а потім визначивши їх тональність, можна зробити висновок щодо тональності тексту взагалі.

Метод, заснований на теоретико-графових моделях

В цьому методі[39] використовується припущення, що не всі слова в текстовому корпусі документа рівнозначні. Деякі слова мають більше вагу та сильніше впливають на тональність тексту. При використанні цього методу, аналіз тональності розбивається на декілька етапів:

1. Побудова графа на основі досліджуваного тексту
2. Ранжування його вершин
3. Класифікація знайдених слів
4. Підрахування результату

Для класифікації слів використовується тональний словник. Для отримання кінцевого результату необхідно визначити значення двох оцінок: позитивною складової та негативної. Для того щоб знайти позитивну складову тексту необхідно знайти суму тональностей всіх позитивних термінів в тексті. Аналогічно знаходиться негативна складова. Кінцева тональність тексту визначається як співвідношення за формулою:

$$T = \frac{P}{N}$$

де T - загальна тональність тексту, P - позитивна складова, N - негативна складова.

Таким чином текст, для якого значення T близько до 1, вважається нейтральним. Для якого T незначно перевищує 1 - позитивним, якщо значно перевищує 1 - сильно позитивним. Так само і для негативних.

Оцінка якості сентиментального аналізу

Точність та якість системи сентиментального аналізу оцінюється тим, наскільки хорошо вона узгоджується з думкою людини відносно емоційної оцінки досліджуваного тексту. Для цього можуть використовуватись такі метрики, як точність та повнота. Формула для знаходження повноти[40]:

$$R = \frac{O_c}{O_T}$$

де R - повнота аналізу, O_c - кількість коректно визначених думок, O_T - загальна кількість думок (як знайдених системою, так і не знайдених).

Точність визначається за формулою:

$$P = \frac{O_c}{O_{TS}}$$

де P - точність, O_c - кількість коректно визначених думок, O_{TS} - загальна кількість думок, знайдених системою.

Таким чином, точність показує кількість досліджуваних текстів, речень або документів, в оцінці яких думка системи сентиментального аналізу збігається з думкою експерта.

3.4 Обґрунтування методу розв’язання

Для вирішення поставленої задачі було обрано метод, що базується на словниках та правилах (англ. rule-based) [38].

Аналіз методом словників та правил передбачає визначення тональності емоційної інформації на рівні речення або словосполучення. Цей метод аналізу є одним з найскладнішим у реалізації, але він може бути найбільш точним у розумінні людських емоцій та думок. Повідомлення можуть містити як позитивні, так і негативні коментарі щодо об’єкта та обраний метод надає можливість оцінки тональності повідомлення загалом, а оцінки окремих висловів та словосполучень, що є більш точним підходом.

Реалізація сентиментального аналізу даних для української мови є дуже складною задачею. На даний момент немає словників тональності для української мови, а синтаксичний парсер все ще знаходиться у стадії розробки.

На рисунку 3.1 представлено загальну схему системи сентиментального аналізу даних. Як можна побачити, перша стадія це препроцесінг тексту. Кожне повідомлення містить дані про відправника, дату та саме текст повідомлення. Потім, виділений текст повідомлення розбивається на слова та вислови (токени, англ. tokens). Вислови визначаються за допомогою пунктуації, що є дуже строгою в українській мові. Ми робимо припущення, що автор повідомлення дотримується правил пунктуації та орфографії.

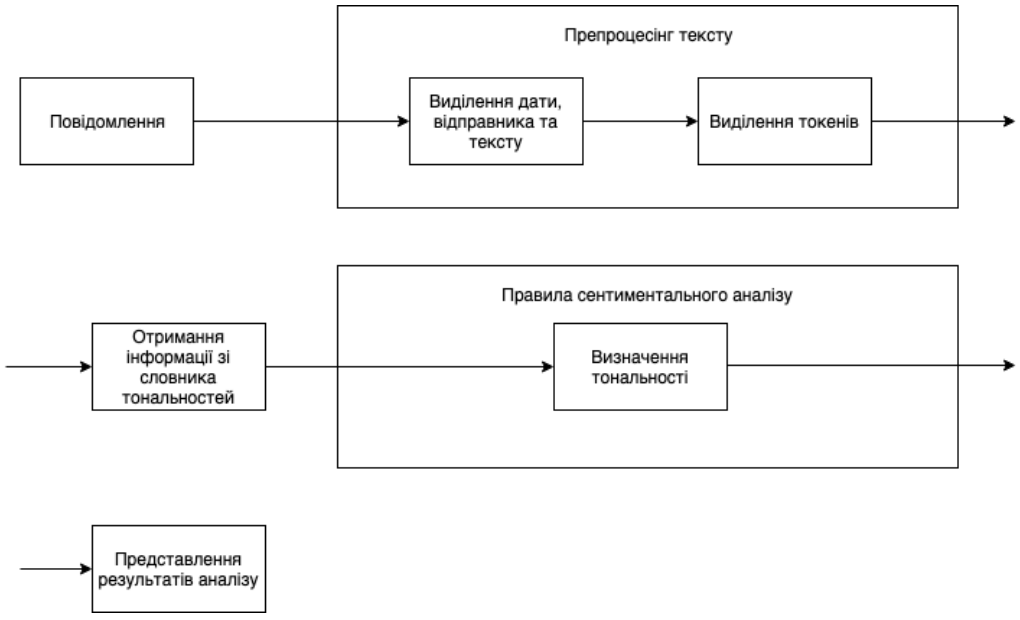

Рисунок 3.1 - Загальна схема системи сентиментального аналізу

Наступний крок алгоритму це оцінка тональності та емоцій кожного слова та виразу. Це робиться за допомогою словнику тональностей, який було сформовано шляхом перекладу словнику AFINN.

Окреме слово може відноситись до однієї з наступних категорій: позитивне (англ. positive), негативне (англ. negative), підсилювач (англ. intensifier), інвертор (англ. inverter) або нейтральне (англ. neutral). Результуюча тональність вислову, в свою чергу, може бути позитивною, негативною, дуже позитивною, дуже негативною або нейтральному.

Було сформовано наступний список правил для уніфікованого аналізу висловів незалежно від контексту:

ПІДСИЛЮВАЧ + ПОЗИТИВНЕ -> ДУЖЕ ПОЗИТИВНЕ

ПІДСИЛЮВАЧ + НЕГАТИВНЕ -> ДУЖЕ НЕГАТИВНЕ

ІНВЕРТОР + ПОЗИТИВНЕ -> НЕГАТИВНЕ

ІНВЕРТОР + НЕГАТИВНЕ -> ПОЗИТИВНЕ

ІНВЕРТОР + НЕЙТРАЛЬНЕ -> НЕЙТРАЛЬНЕ

ПОЗИТИВНЕ + НЕЙТРАЛЬНЕ -> ПОЗИТИВНЕ

НЕГАТИВНЕ + НЕЙТРАЛЬНЕ -> НЕГАТИВНЕ

ПОЗИТИВНЕ + ПОЗИТИВНЕ -> ПОЗИТИВНЕ

НЕГАТИВНЕ + НЕГАТИВНЕ -> НЕГАТИВНЕ

НЕЙТРАЛЬНЕ + ПОЗИТИВНЕ -> ПОЗИТИВНЕ

НЕЙТРАЛЬНЕ + НЕЙТРАЛЬНЕ -> НЕЙТРАЛЬНЕ

НЕЙТРАЛЬНЕ + НЕГАТИВНЕ -> НЕГАТИВНЕ

ПОЗИТИВНЕ + НЕГАТИВНЕ -> НЕГАТИВНЕ

НЕГАТИВНЕ + ПОЗИТИВНЕ -> НЕГАТИВНЕ

Таким чином, процес сентиментального аналізу вислову можно представити у вигляді дерева.

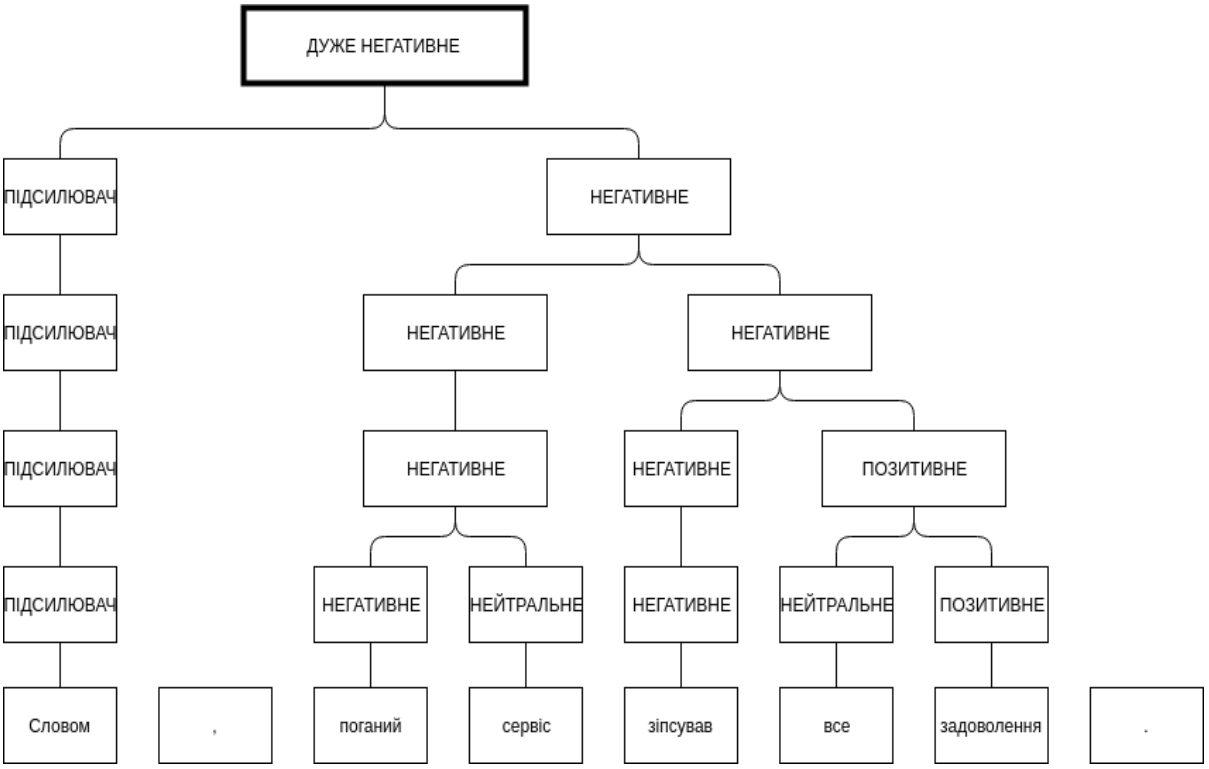

Рисунок 3.2 - Приклад дерева аналізу

На кожному рівні правила обробляються один за одним, але вони сортуються від біль розповсюджених до менш розповсюджених з метою зменшення часу, необхідного на пошук відповідного правила.

Висновок до розділу

В даному розділі було детально розібрано та порівняно різні методи до класифікації та оцінки тональності тексту. Були наведені достоїнства та недоліки кожного методу та випадки, в яких потрібно використовувати ті чи інші методи розв’язання.

Було описано та обґрунтовано запропонований підхід до вирішення даної задачі з детальними ілюстраціями та прикладами розв’язання.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

При створенні програмного продукту були використані такі програмні засоби: PhpStorm[21], PgAdmin[22], draw.io[23], Adobe Photoshop[24].

В якості мови програмування для Back-End частини було обрано JavaScript разом с програмною платформою Node.js[25].

JavaScript - це мультипарадігменна мова програмування, яка підтримує об’єктно-орієнтований, функціональний та імперативний стилі[26]. JavaScript є інтерпретованою мовою, що дозволяє заощадити значну кількість часу, що зазвичай витрачається на компіляцію програми. Основними особливостями JavaScript є динамічна типізація, слабка типізація, автоматичне керування пам’яттю та прототипне програмування. JavaScript є вузькоспеціалізованою мовою та призначений лише для виконання в веб-браузерах.

V8 - це движок з відкритим вихідним кодом, що був розроблений компанією Google та наразі використовується у багатьох продуктах, таких як Node.js, Google Chrome, Chromium, Android, Google chromeOS та інші[27]. Він написан мовою програмування C++ та реалізовує стандарт ECMA-262[28]. V8 компілює JavaScript напряму в машинний код до того як його виконати, замість того, щоб інтерпретувати в байткод або компілювати програму цілком, а потім запускати з файлової системи. Зкомпільований код додатково динамічно оптимізується під час рантайму.

Node.js - це програмна платформа, заснована на движку V8, яка дозволяє перетворити JavaScript з вузькоспеціалізованої на мову загального призначення. Вона була розроблена Райаном Далем в 2009 році, який під час своїх експериментів прийшов до висновку, що замість традиційної моделі паралелізма на основі потоків, можна використати подієво-орієнтовану систему. В склад Node.js також входить бібліотека libuv[29] (англ. library unicorn velociraptor), написана мовою програмування C++, яка відповідає за

всі операції асинхронного вводу-виводу (I/O), такі як мережеві запроси, читання та запис файлів, взаємодія з базами даних, використовуючи модель подієвого циклу (англ. Event Loop). Event Loop регулює послідовність виконання контекстів - стек[30]. Він формується під час виконання певної події, або виклику функції. Кожен раз коли виконується подія, функція, яка повинна бути виконана під час його виконання, переміщується до черги на виконання, в Event Loop, який послідовно, з кожним циклом виконує покладений до нього код.

Таким чином, в Node.js постійно працюють пов'язані між собою синхронна та асинхронна черги на виконання. Синхронна - стек, який формує чергу і пробрасує виклики функцій до асинхронної - Event Loop - які будуть виконані після завершення виконання поточного контексту. Детальна схема роботи представлена на рисунку 4.1.

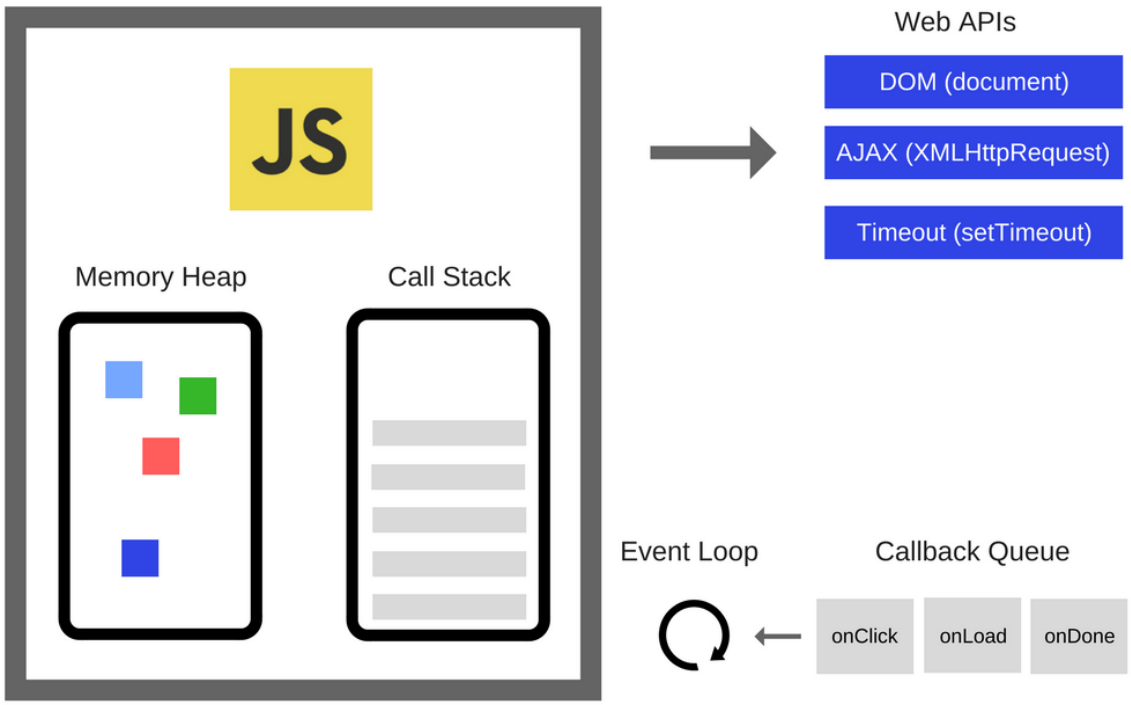

Рисунок 4.1 - Схема роботи подієвого циклу в Node.js

Завдяки такій моделі асинхронності, програми написані на Node.js можуть обробляти велику кількість запитів паралельно, що є дуже важливим для нашою системи, так як навантаження на систему може становити сотні запитів в секунду шляхом надсилання оновлень у вигляді веб-хуків.

PostgreSQL - вільна об’єктно-реляційна система управління базами даних. Вона була створена у 1996 році, як альтернатива існувавшим тоді комерційним системам. Можливості сервера PostgreSQL:

- Надійність і стійкість на великих навантаженнях;
- Кросс-платформеність;
- Необмежена кількість користувачів;
- Підтримка баз даних необмеженого розміру;
- Висока швидкість виконання команд;
- Наявність системи безпеки;

Postgres є дуже схожою на інші системи управління базами даних, такі як MySQL[31], MariaDB[32], але має деякі переваги, наприклад підтримка користувацьких об’єктів та їх поведінки, враховуючи типи даних, функції, операції, домени та індекси. Також PostgreSQL має більш широкий список типів даних, включаючи такі типи як uuid, грошовий, перераховуємий, геометричній, бінарний, мережеві адреси, бітові строки, текстовий пошук, xml, json, масиви, композитні типи та числові діапазони.

Такой Postgres реалізовує стандарт ANSI-SQL:2008 та задовольняє вимогам ACID (Atomicity, Consistency, Isolation, Durability)[33], що, наряду з відомою посилальною та транзакційною цілісністю, робить PostgreSQL однією з найнадійніших систем управління базами даних.

Для підтримання комунікації між Back-End сервісами системи було обрано RabbitMQ[34] - програмний брокер повідомлень на основі стандарта AMQP[35] з відкритим вихідним кодом. Він написаний мовою програмування Erlang, яка відома своєю швидкодією. Клієнтські інтерфейси для взаємодії з брокером існують для багатьох мов програмування, в тому

числі для Node.js, що дозволяє нам повноцінно використовувати можливості RabbitMQ.

RabbitMQ є одним з найкращих інструментів для побудови сервіс-орієнтованих систем. При обміні повідомленнями задіяні три основні сутності:

1. Producer (постачальник) - програма, що надсилає повідомлення.
2. Queue (черга) - ім'я "поштової скриньки". Вони існують всередині RabbitMQ. Всі повідомлення зберігаються всередині черг та їх може бути скільки завгодно багато. Постачальник може відправляти повідомлення як до однієї черги, так і до багатьох одночасно.
3. Consumer (споживач) - програма, що приймає повідомлення.

На малюнку 4.2 наведено простий приклад взаємодії через чергу hello постачальника P та споживача C.

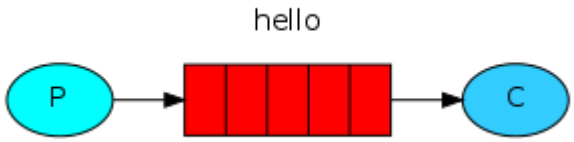

Рисунок 4.2 - Схема процессу обміну повідомленнями через RabbitMQ

Таким чином, ми можемо використати RabbitMQ для побудови взаємодії між сервісами системи, які будуть описані нижче.

Для розробки клієнтської частини системи (веб-застосування) використовуються HTML, CSS, а також фреймворк VueJS з мовою програмування TypeScript.

TypeScript[36] - мова програмування, розроблена компанією Microsoft у 2012 році, яка розширює функціонал JavaScript. Окрім статичної типізації, TypeScript додає такі опції як класи, інтерфейси, міксини, модулі, кортежі та інші. TypeScript компілюється у JavaScript, та є обернено-сумісним з ним, тобто будь-яких валідний JavaScript код є валідним TypeScript кодом. Використання TypeScript в проекті допомагає запобігти багато помилок під

час написання коду, завдяки суворій типізації, а також покращує якість кода завдяки введенню додаткових абстракцій.

VueJS[37] - JavaScript-фреймворк з відкритим вихідним кодом, призначений для розробки користувацьких інтерфейсів. Він був створений у 2013 році, колишнім співробітником Google Еваном Ю. VueJS зосереджений на декларативному рендерінгу та композиції компонентів. Фреймворк використовує шаблони, які базуються на HTML та дозволяють з'єднувати DOM-елементи з властивостями екземпляра Vue. В поєднанні з системою реактивності, Vue здатен розрахувати мінімальну кількість елементів, які потребують ре-рендеринга в результаті зміни даних у моделі. Реактивність базується на JavaScript об'єктах з оптимізованим ре-рендерингом, кожен компонент слідкує за своїми реактивними залежностями та використовує мінімальну кількість DOM-операцій для внесення змін.

Компоненти у VueJS поширюють функціонал HTML компонентів та інкапсулюють код, що можно перевикористати. Компонент, за великим рахунком, являє собою окремий екземпляр Vue з наперед визначеним набором налаштувань.

Окрім цього, разом з фреймворком VueJS зазвичай використовується бібліотека Vuex, яка призначена для керування станом застосунку, що використовує Flux архітектуру, а також Vue-Router, що призначений для навігації між "сторінками" SPA (Single Page Application).

Для створення початкового шаблону застосунку, використовується Vue-CLI 3 (Command Line Interface), яка надає можливість обрати зборщик проекту (Webpack або Parcel), мову програмування (JavaScript або TypeScript), додаткові фреймворки та бібліотеки для тестування, CSS-препроцесор, статичний аналізатор кода, компілятор та інше.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Для корректної роботи даної системи конфігурація сервера має бути натопуною:

- 64 розрядна операційна система;
- процесор з тактовою частотою не нижче 1.5 ГГц;
- достатній об’єм оперативної пам’яті (не меньше 8 Гб);
- інші складові можуть мати будь-які параметри, тому що води не значним чином впливають на роботу програми;

Додатково на сервері має бути встановлене наступне програмне забезпечення:

- операційна система Linux Debian (або базована на Debian);
- база даних PostgreSQL 10;
- Node.js 10
- RabbitMQ 3.7

Комп’ютер клієнта повинен мати підключення до інтернету та встановленний веб-браузер.

4.2.2 Опис локальної обчислювальної мережі

Використання системи вимагає забезпечення інтернет з’єднання клієнтського робочого місця з сервером зі швидкістю більше 1 мб/с. Обчислювальна мережа зі сторони сервера повинна підтримувати протокол TCP/IP.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма розгортання

Діаграма розгортання (англ. deployment diagram) - UML діаграма, що відображає обчислювальні вузли під час роботи програми, компоненти та об’єкти, що виконуються на цих вузлах.

Структурну схему діаграми розгортання наведено у графічному матеріалі до роботи. В даній діаграмі показано процес взаємодії між вузлами, які являють собою:

- комп’ютер-клієнт;
- веб-сервер, який має зворотній зв’язок з клієнтом по протоколу HTTPS та містить наступні компоненти: NodeJS API, VueJS застосунок, PostgreSQL сервер, сервіс сентиментального аналізу даних, Webhook сервіс, WebSocket сервіс, брокер повідомлень RabbitMQ;

4.3.2 Діаграма послідовності

Діаграма послідовності (англ. sequence diagram) - UML діаграма, яка відображає взаємодії об’єктів, впорядковані за часом, наприклад діаграма може відображати задіяні об’єкти та послідовність відправлених повідомлень.

Діаграма послідовності процесу звернення клієнта до служби підтримки та отримання відповіді представлена у графічному матеріалі до роботи.

Клієнт відправляє повідомлення чат-боту за допомогою месенджера, після чого вебхук-сервіс отримує пакет даних, які кажуть про те, що надійшло нове повідомлення. Дані записуються до черги на сентиментальний аналіз до RabbitMQ, після чого аналізуються сервісом аналізу, записуються до базу даних, та записуються до черги на відправлення оператору. Після цього, WebSocket сервіс зчитує дані з черги та відправляє до веб-застосунку

по веб-сокет протоколу. Веб-застосунок відображає повідомлення з результатом аналізу оператору. Оператор дає відповідь клієнту, в результаті чого веб-застосунок відправляє запрос до API, який записує нове повідомлення до бази даних і за допомогою відкритих API месенджера відправляє відповідь клієнту.

На рисунку 4.4 представлено діаграму послідовності для процесу створення розсилки. Адміністратор створює розсилку, в результаті чого веб-застосування викликає метод API для створення розсилки. API вибирає з бази даних усіх клієнтів та за допомогою методів відкритого API месенджерів робить розсилку.

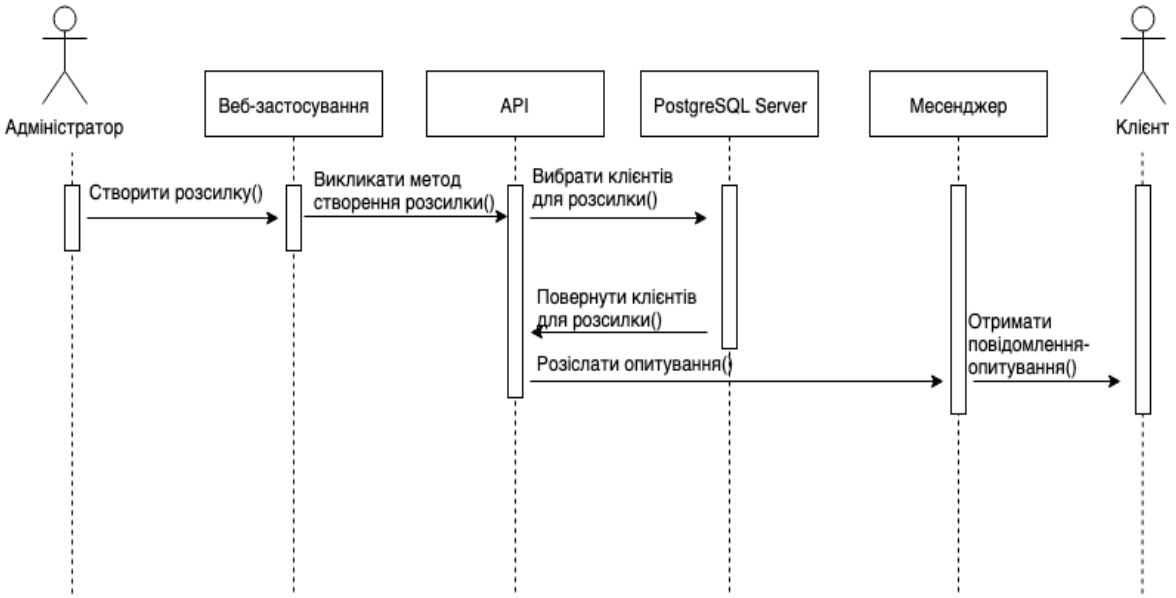

Рисунок 4.4 - Діаграма послідовності процесу створення розсилки

4.3.3 Специфікація функцій

Так як Node.js предоставляє зручні інструменти для поділу програми на модулі, система була побудована таким чином, що кожний файл являє собою інкапсульований модуль, який експортує інтерфейс для взаємодії з ним. Функції, що експортуються з модулей не мають назви (є лямбда-функціями), тому нижче у таблиці 4.1 замість назв функцій буде наведено назви файлів, які експортують їх. Представлено лише список ключових функцій.

Таблиця 4.1 – Специфікація функцій

Назва файлу	**Дія**
chat/get.js	Отримання списку чатів.
contact/delete.js	Видалення контакту.
contact/get.js	Отримання інформації про контакт.
message/get.js	Отримання списку повідомлень обраного чату.
message/send_all.js	Відправлення розсилки.
message/mark_as_read.js	Відмітити повідомлення як прочитане.
message/send_text.js	Відправити повідомлення.
bot/create.js	Додати бота в систему.
bot/remove.js	Видалити бота з системи.
bot/telegram_add.js	Додати телеграм-бота.
bot/viber_add.js	Додати вайбер-бота
bot/facebook_add.js	Додати фейсбук-бота
db/pgdb.js	Функція для роботи з базою даних.
helpers/logger.js	Функція для логування поточних даних.
facebook/send_message.js	Відправити повідомлення в фейсбук.
viber/send_message.js	Відправити повідомлення в вайбер.
telegram/send_message.js	Відправити повідомлення в телеграм.
middleware/auth.js	Функція для перевірки, чи авторизован користувач

Продовження таблиці 4.1

routes.js	Функція для реєстрації всіх ендпоінтнів.
save_message.js	Функція для збереження нового повідомлення.
facebook_update.js	Функція обробки нового повідомлення з фейсбук.
viber_update.js	Функція обробки нового повідомлення з вайбер.
telegram_update.js	Функція обробки нового повідомлення з телеграм.
sentiment.js	Функція для сентиментального аналізу тексу.
connections.js	Підтримання з’єднання з клієнтом по веб-сокету.
handle.js	Відправлення повідомлення по веб-сокету.
auth/login.js	Функція авторизації.

Висновок до розділу

В даному розділі було розглянуто програми, мови програмування, інструменти, систему управління реляційними базами даних, брокер повідомлень за допомогою яких було реалізовано систему.

Було розглянуто загальні вимоги до технічних засобів, які мають забезпечувати стабільну роботу системи.

Наведено діаграму розрортання, що детально описує архітектуру програмного забезпечення та шляхи взаємодії між сервісами системи.

Наведено діаграми послідовності для визначених процесів з детальним поясненням до них.

Було розглянуто функціонал модулей системи.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

На головній сторінці показана форма авторизації користувача (однаково для оператора та адміністратора), представлено на рисунку 5.1.

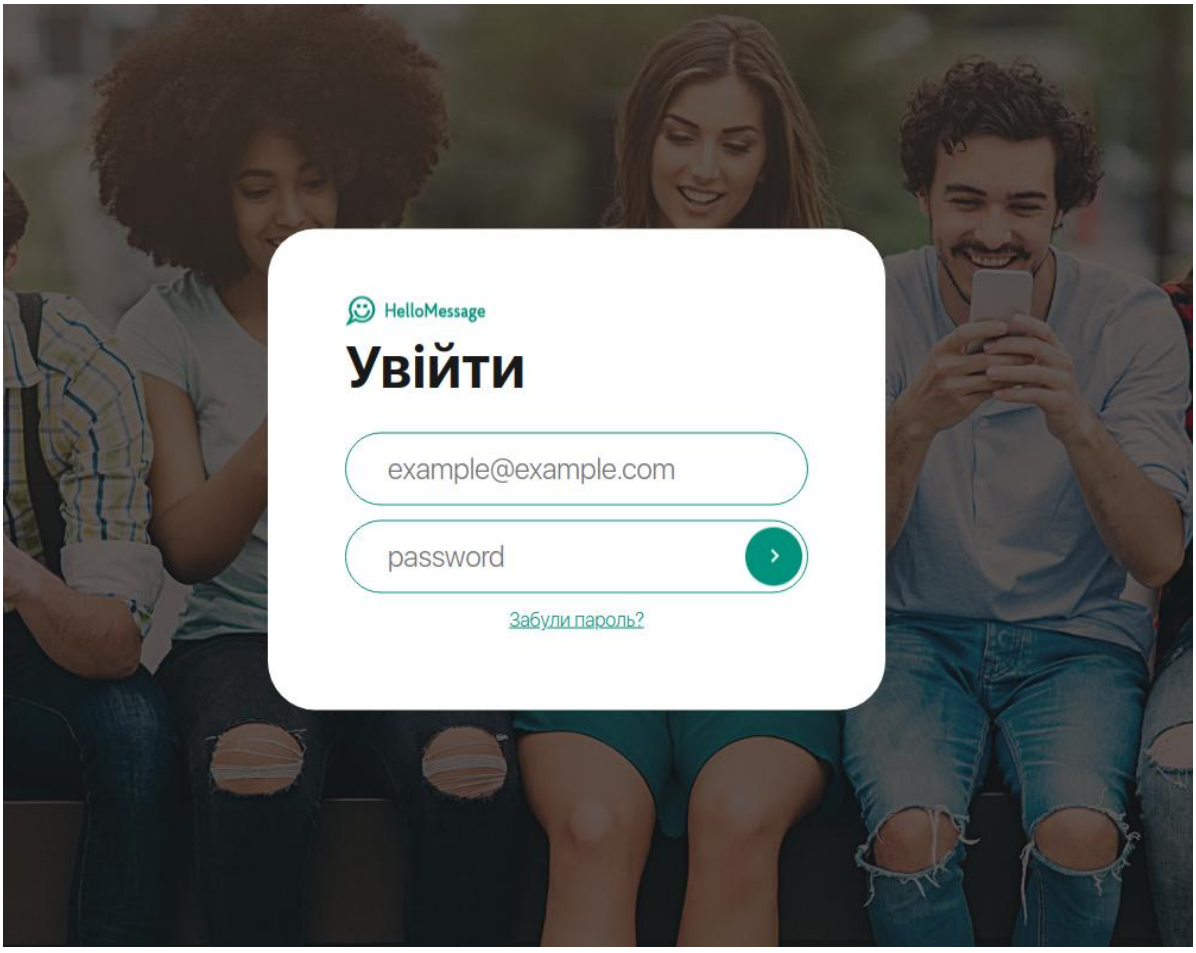

Рисунок 5.1 - Форма авторизації

Для авторизації необхідно ввести логін (адресу електронної пошти) та пароль. Якщо при введенні даних було допущено помилку, користувачу відображається повідомлення про це (рисунок 5.2).

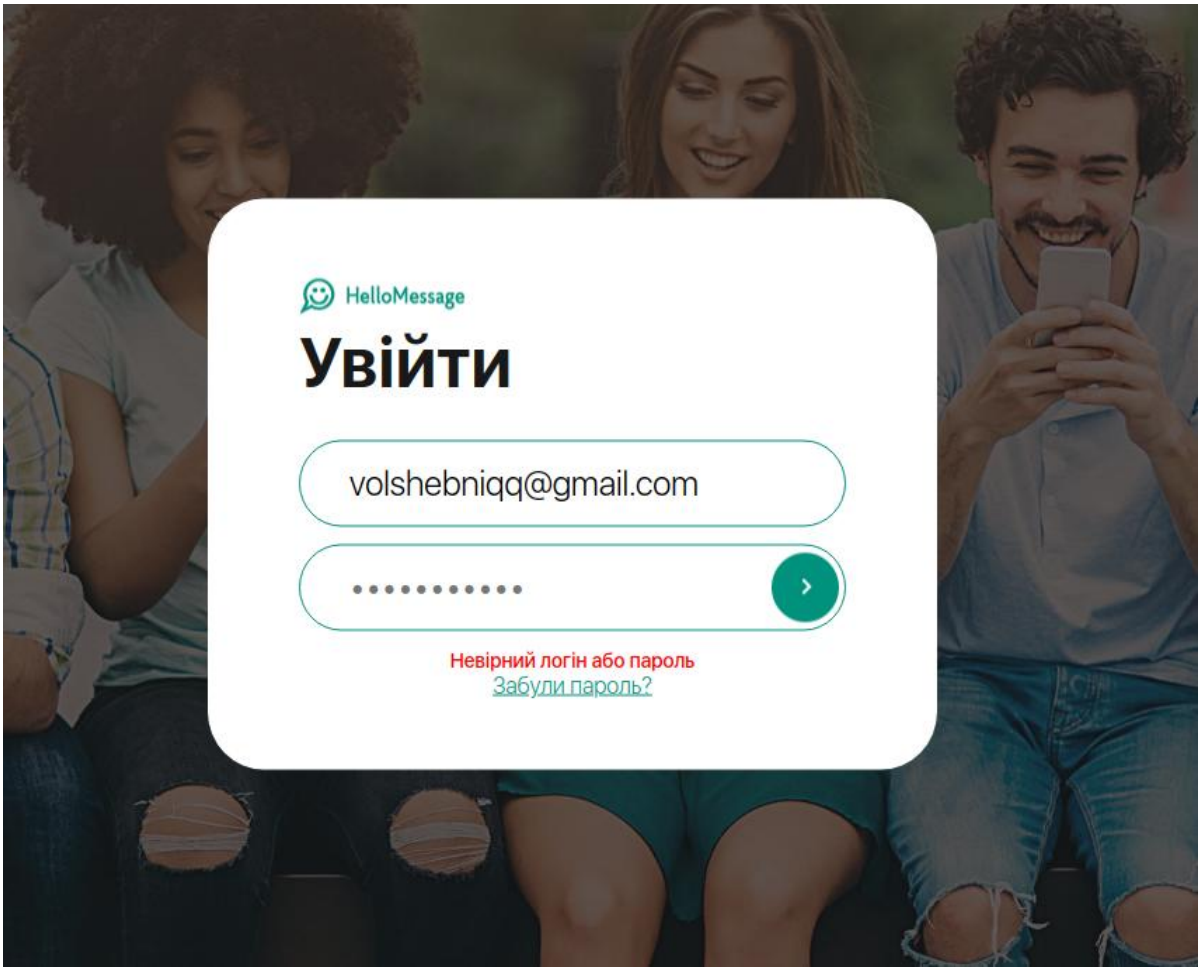

Рисунок 5.2 - Форма авторизації з помилкою

Якщо дані для авторизації було введено вірно, то користувач потрапляє на головну сторінку застосунку. Для адміністратора вона має вигляд, показаний на рисунку 5.3.

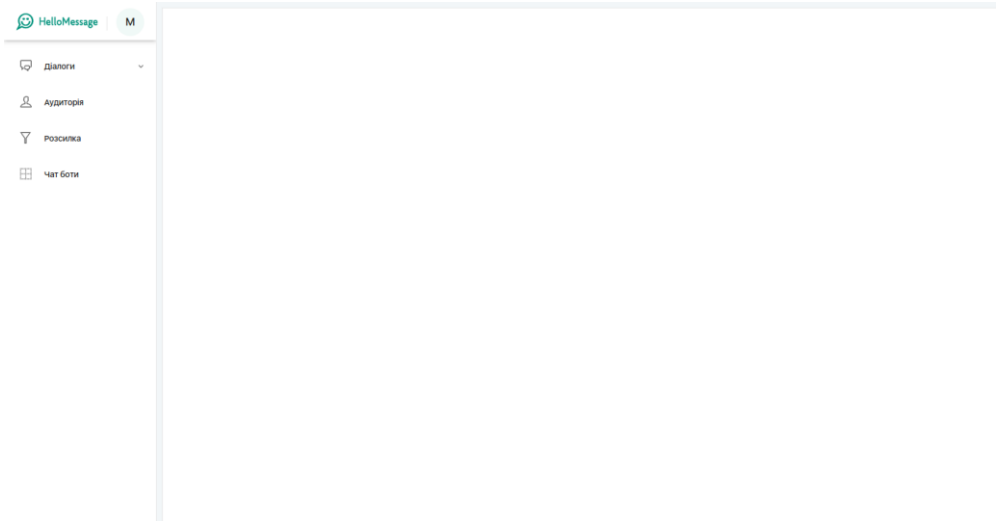

Рисунок 5.3 - Головна сторінка системи (вигляд для адміністратора)

Для оператора головна сторінка має такий самий вигляд, за винятком відсутності пунктів “Аудиторія”, “Розсилка” та “Чат боти” у головному меню, у лівій частині екрану.

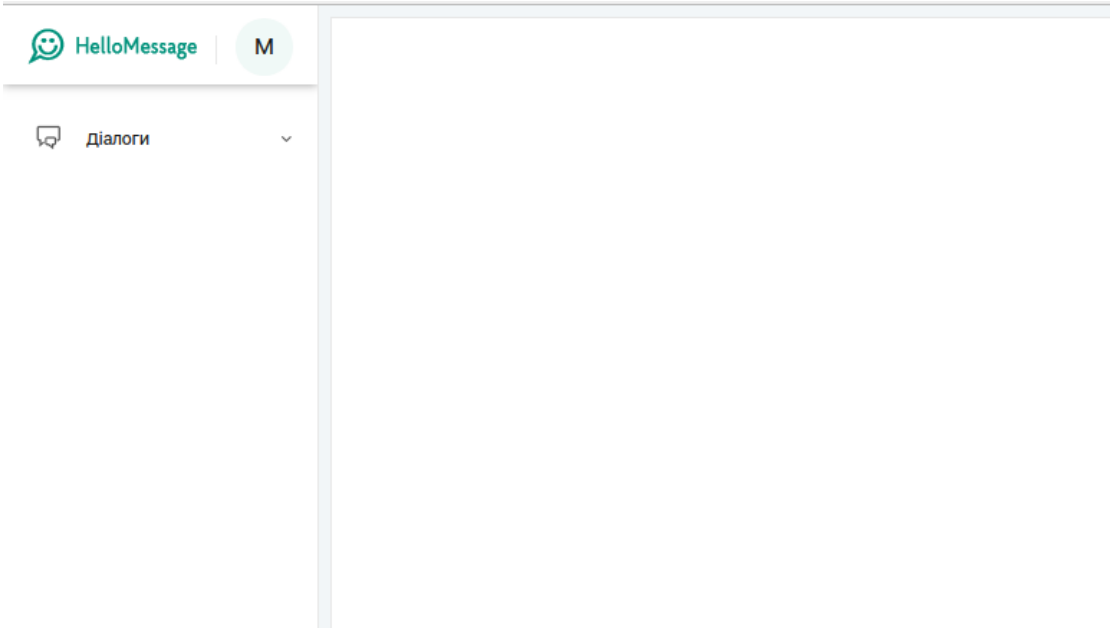

Рисунок 5.4 - Головна сторінка системи (вигляд для оператора)

Навігація між сторінками веб-застосування відбувається за допомогою меню у лівій частині екрана. Пункт меню “Діалоги” має два підпункти “Непрочитані” та “Усі”, які фільтрують чати, відображаючи тільки ті, де є непрочитані повідомлення та усі чати відповідно. Сторінка з діалогами має вигляд, зображений на рисунку 5.5. Список діалогів має кнопки для фільтрації (Усі, Telegram, Facebook Messenger Viber). Список відображає нещодавні діалоги, показуючи аватар клієнта з месенджера, ім’я, дату останнього повідомлення, перші слова останнього повідомлення та логотип месенджера, яким користується клієнт. Також у вигляді невеликого квадрата у лівому правому куту елемента, що відображає діалог, представлено результат аналізу останнього текстового повідомлення клієнта. Зелений колір означає позитивну оцінку, червоний - негативну, сірий - нейтральну.

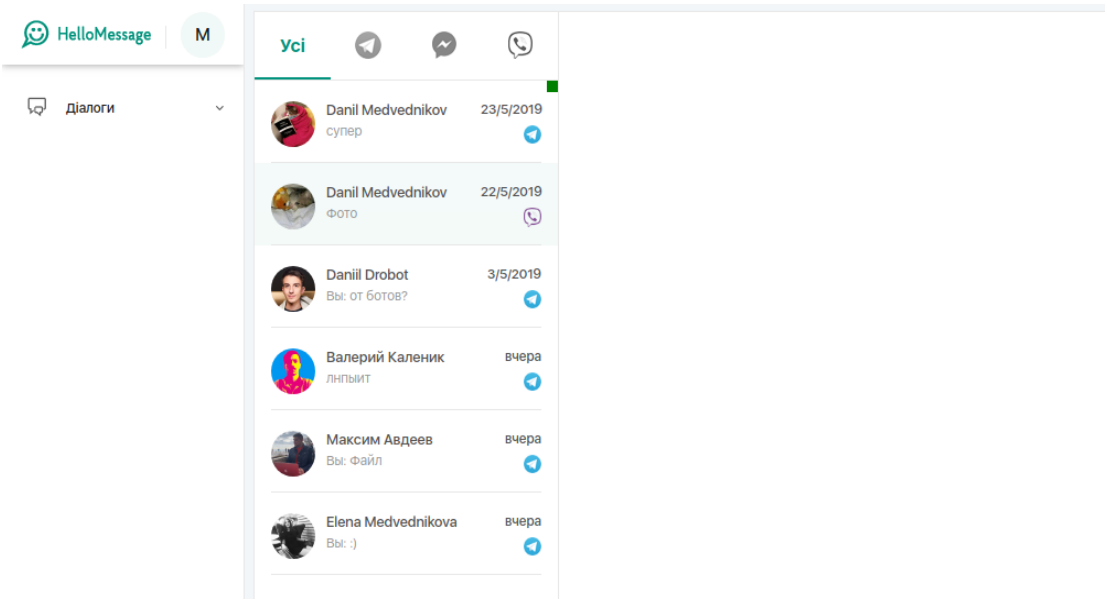

Рисунок 5.5 - Сторінка з діалогами

Після вибору діалогу в списку, у вікні справа відображається історія повідомлень з обраним клієнтом (рисунок 5.6). Крім текстових повідомлень є можливість відправляти та отримувати фотографії та документи.

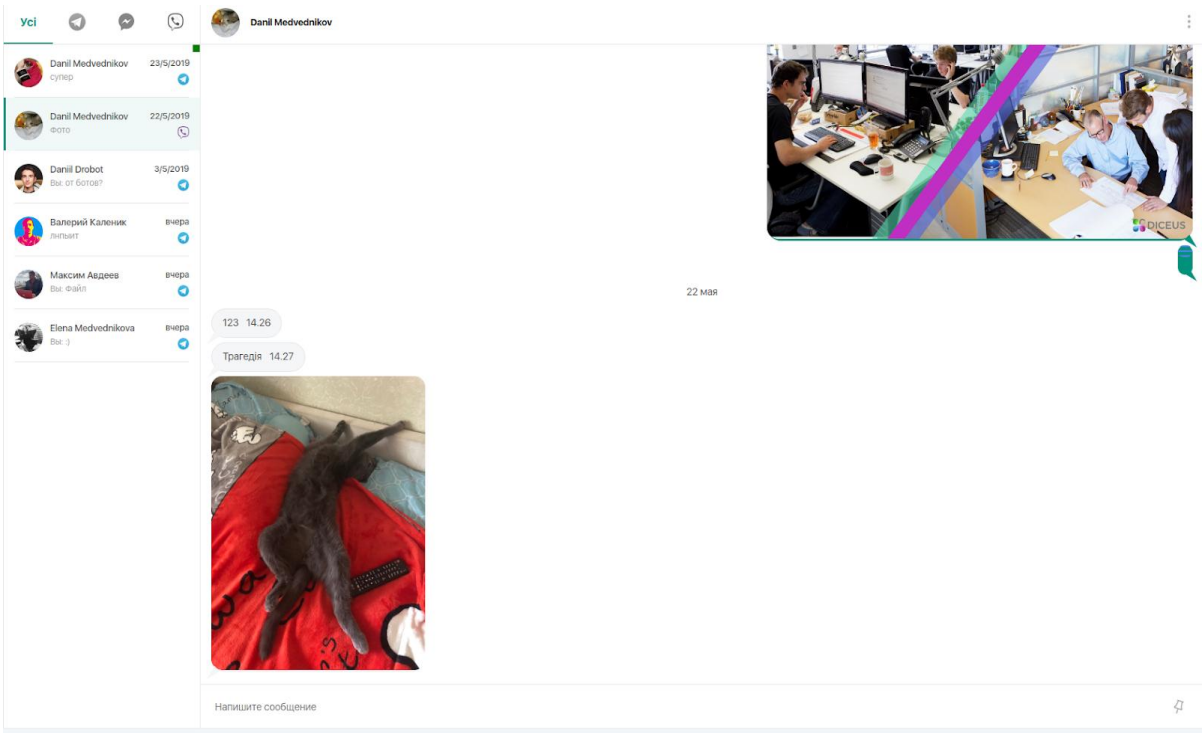

Рисунок 5.6 - Історія повідомлень з клієнтом

Для відправлення повідомлення клієнту необхідно використовувати поле вводу в нижній частині екрану історії повідомлень (рисунок 5.7). Також

можна відправити файл або фотографію, використовуючи іконку прикріплення файлу в правій частині поля вводу.

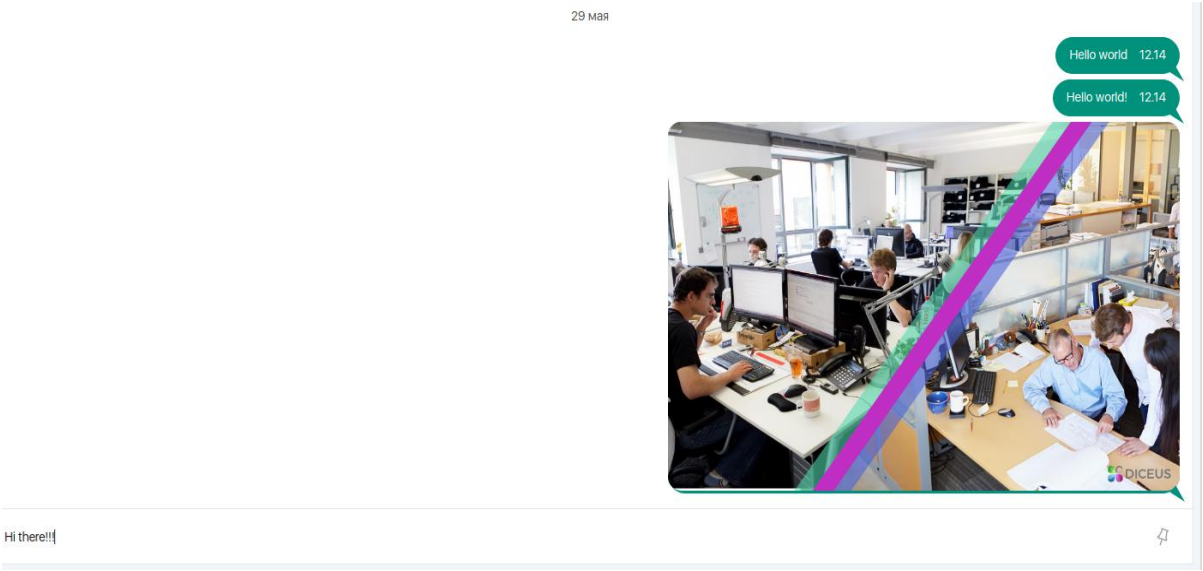

Рисунок 5.7 - Поле вводу повідомлення

Як можна побачити з інтерфейсу месенджера Telegram на рисунку 5.8, дані повідомлення було надіслано користувачеві.

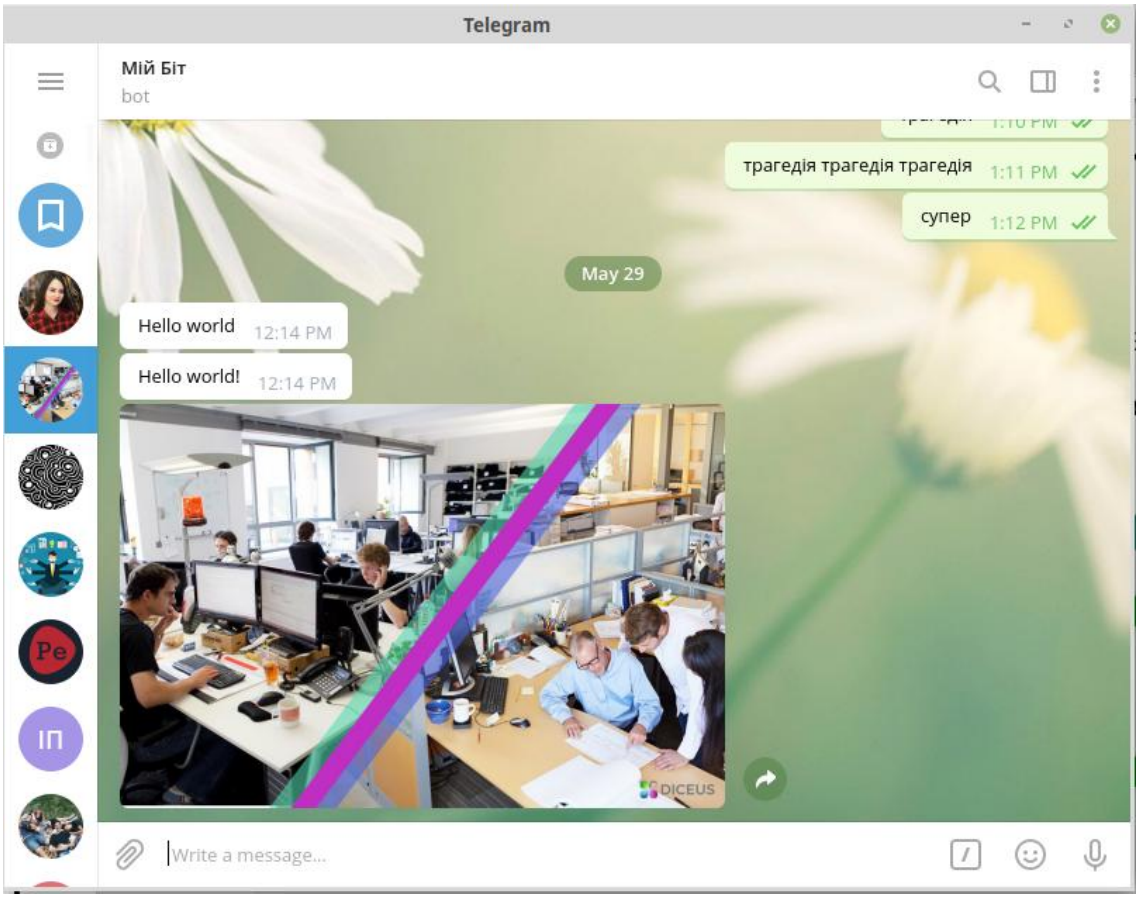

Рисунок 5.8 - Результат відправлення повідомлень у месенджер

Так саме система працює у зворотному напрямку, коли текстові повідомлення надсилаються клієнтом у месенджері, цей процес відображено на рисунках 5.9 - 5.11.

Рисунок 5.8 - Відправлення повідомлень клієнтом.

Рисунок 5.9 - Відображення нових повідомлень у меню та списку діалогів

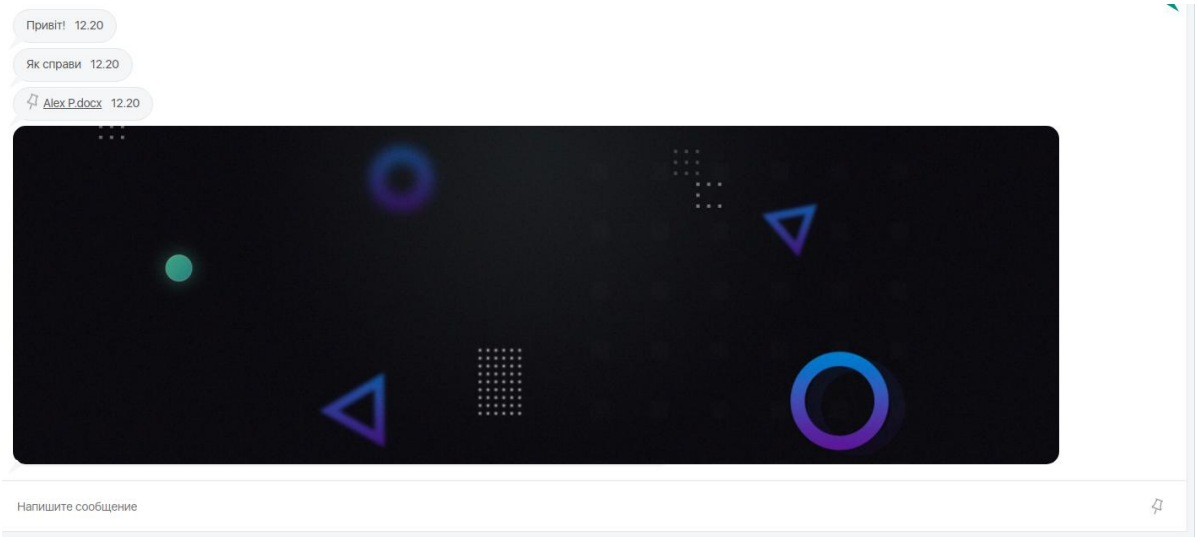

Рисунок 5.10 - Відображення повідомлень, відправлених клієнтом

Оператор або адміністратор мають змогу переглянути інформацію про клієнта, заблокувати або розблокувати клієнта, використавши випадаюче меню (рисунок 5.11) у правому верхньому кутку вікна, що відображає поточний діалог.

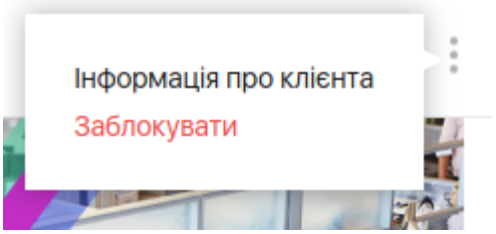

Рисунок 5.11 - Випадаюче меню для управління контактом

Інформація про клієнта відображається у спливаючому вікні (рисунок 5.12), що з’являється у правій частині сторінки. У цьому вікні буде відображено повне ім’я клієнта, аватар, месенджер який він використовує, нікнейм у месенджері та дату створення клієнту у базі даних.

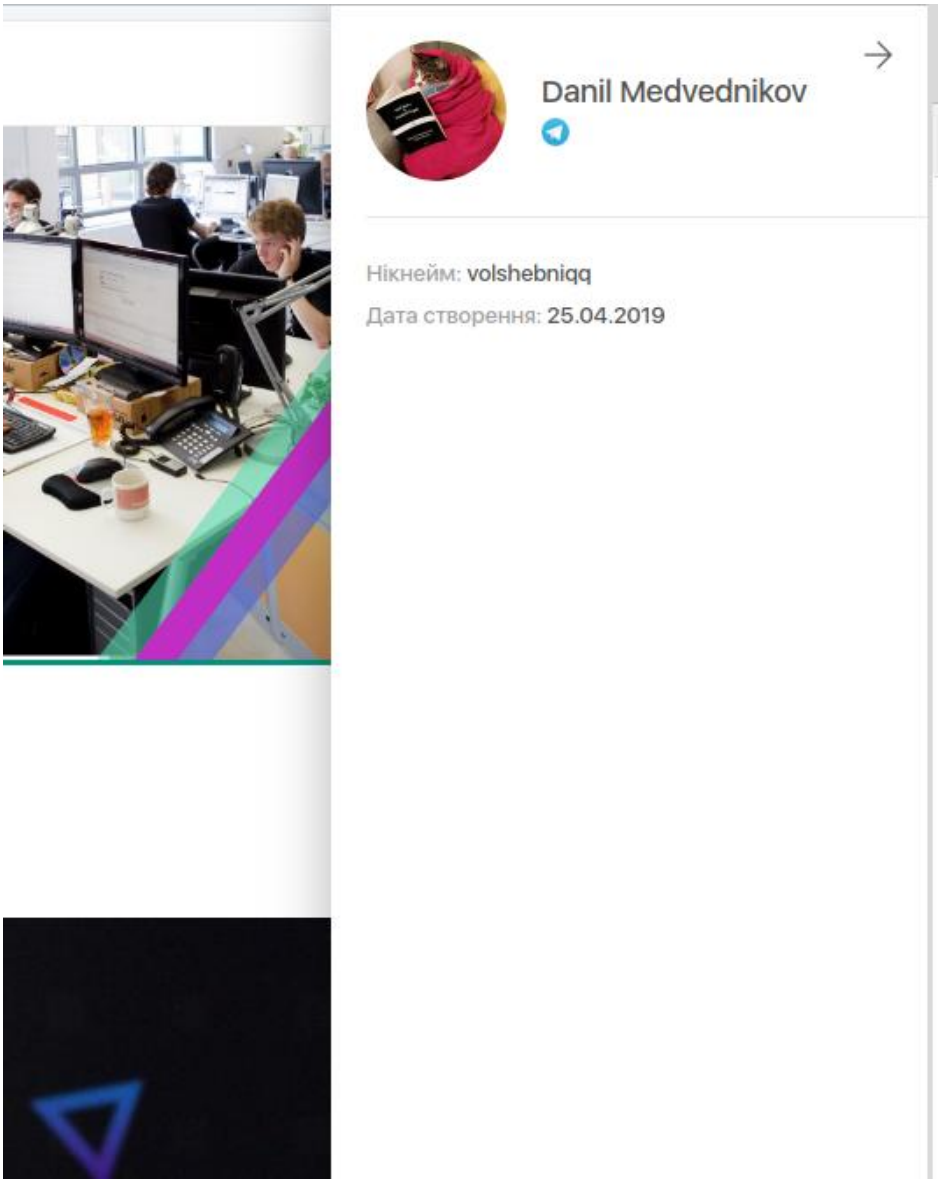

Рисунок 5.12 - Спливаюче вікно з інформацією про клієнта

Адміністратор має змогу переглянути список чат-ботів на сторінці “Чат Боти” (рисунок 5.13). По кожному чат-боту відображається наступна інформація: ім’я, нікнейм, канал та дата створення. Також, адміністратор має змогу видалити чат-бота, натиснувши на кнопку “Видалити” навпроти відповідного чат-бота.

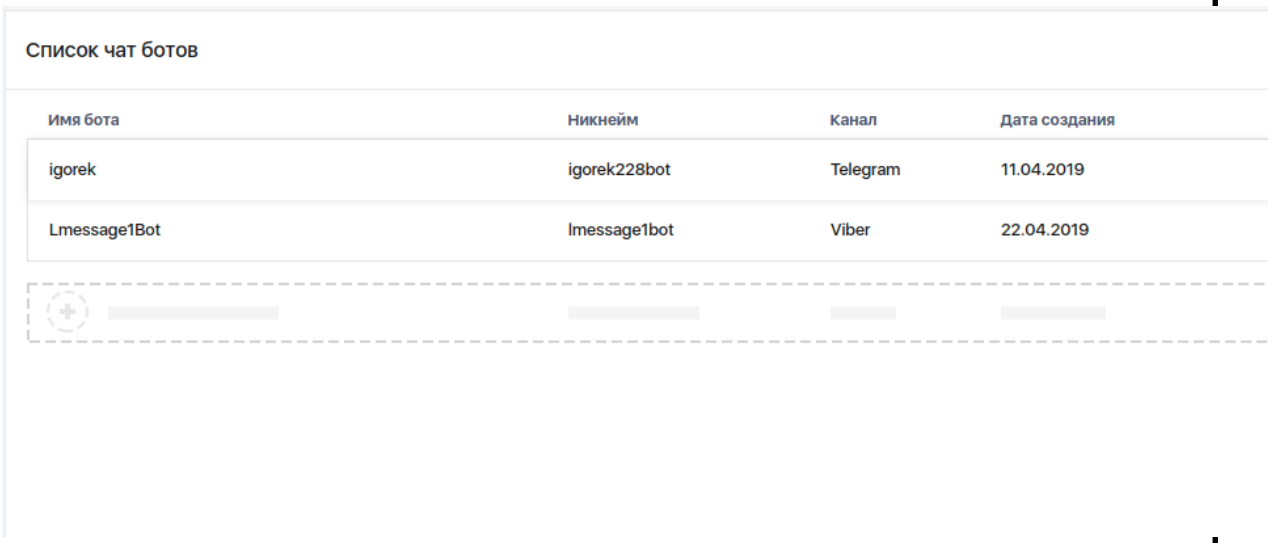

Рисунок 5.13 - Список чат-ботів в системі

Адміністратор має змогу зробити розсилку по всім клієнтам, використовуючи форму на сторінці “Розсилки”, що зображена на рисунку 5.14. Адміністратор повинен ввести повідомлення у текстове поле, після чого натиснути на кнопку “Розіслати”. Дане повідомлення буде розіслане усім користувачам.

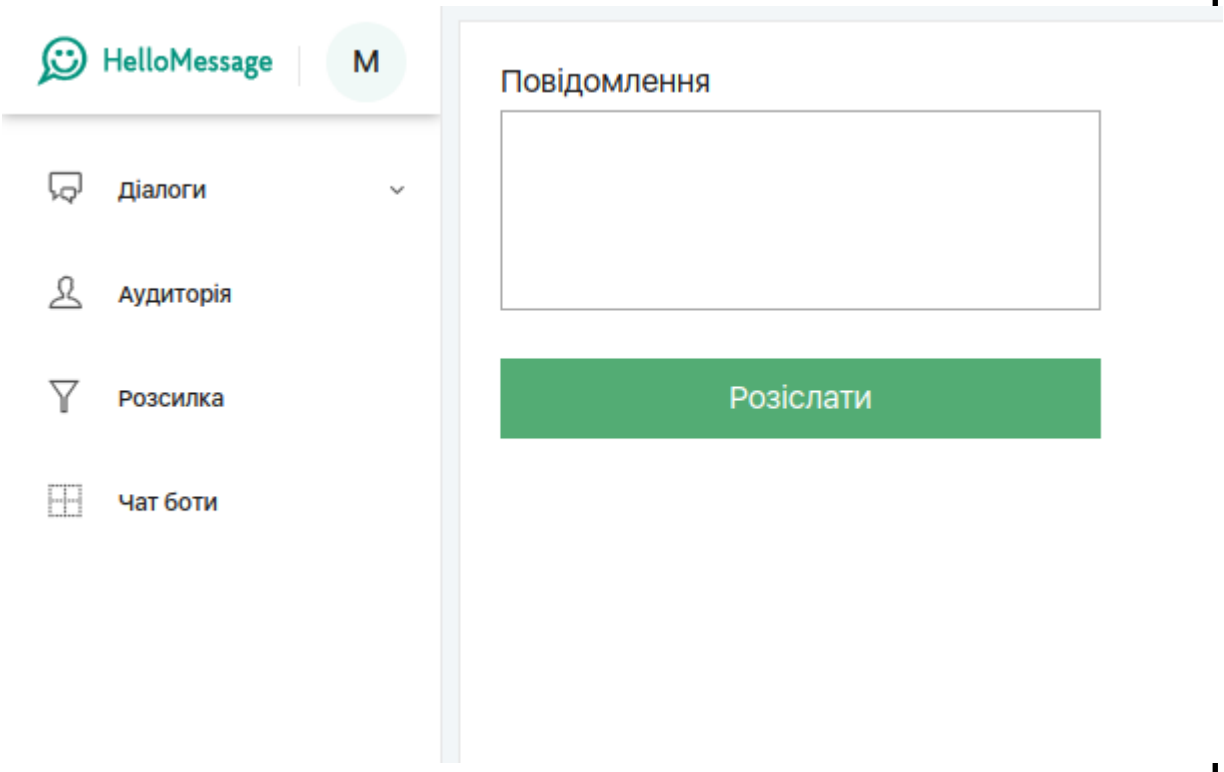

Рисунок 5.14 - Форма створення розсилки

5.2 Випробування програмного продукту

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій комплексу задач з підтримки користувачів з використанням сентиментального аналізу даних вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

- ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

В результаті тестування була перевірена уся функціональність моделі. У наступних таблицях наведено перелік випробувань основних функціональних можливостей.

Таблиця 5.1 - Вхід користувача в систему

Мета тесту	**Перевірка функції «Авторизація в системі»**
Початковий стан моделі	Відкрита сторінка аутентифікації
Вхідні дані:	Електронна пошта та пароль користувача
Схема проведення тесту:	Ввести електронну пошту та пароль користувача у відповідні текстові поля. Після цього натиснути кнопку “Увійти”
Очікуваний результат:	Перехід на головну сторінку системи

Продовження таблиці 5.1

Стан моделі після проведення випробувань:	Відкрита головна сторінка системи

Таблиця 5.2 - Відображення списку діалогів

Мета тесту	**Перевірка функції «Список діалогів»**
Початковий стан моделі	Відкрита головна сторінка системи
Вхідні дані:	
Схема проведення тесту:	Відкрити вкладку меню “Діалоги” у головному меню програми та натиснути на кнопку “Усі”
Очікуваний результат:	Перехід на сторінку діалогів
Стан моделі після проведення випробувань:	Відкритка сторінка “Діалоги”, відображено список діалогів

Таблиця 5.3 - Відображення історії повідомлень

Мета тесту	**Перевірка функції «Історія повідомлень»**
Початковий стан моделі	Відкрита сторінка діалогів
Вхідні дані:	
Схема проведення тесту:	Вибрати діалог зі списка діалогів
Очікуваний результат:	Відображено повідомлення, що відносяться до обраного діалогу
Стан моделі після проведення випробувань:	Відображено повідомлення, що відносяться до обраного діалогу

Таблиця 5.4 - Відправлення текстового повідомлення

Мета тесту	**Перевірка функції «Відправка повідомлення»**
Початковий стан моделі	Відкрито діалог з обраним користувачем
Вхідні дані:	Текст повідомлення
Схема проведення тесту:	Ввести текст повідомлення до текстового поля вводу повідомлення та натиснути клавішу Enter на клавіатурі
Очікуваний результат:	Повідомлення відправлено
Стан моделі після проведення випробувань:	Нове повідомлення відображено в історії повідомлень

Таблиця 5.5 - Відправлення фотографії

Мета тесту	**Перевірка функції «Відправлення фотографії»**
Початковий стан моделі	Відкрито діалог з обраним користувачем
Вхідні дані:	Файл фотографії для відправки
Схема проведення тесту:	Натиснути на кнопку “Прикріпити” у правій частині поля вводу та обрати потрібну фотографію зі списку файлів
Очікуваний результат:	Фотографію відправлено
Стан моделі після проведення випробувань:	Повідомлення з фотографією відображено в історії повідомлень

Таблиця 5.6 - Відправлення документів

Мета тесту	**Перевірка функції «Відправлення документів»**
Початковий стан моделі	Відкрито діалог з обраним користувачем
Вхідні дані:	Файл документу для відправки
Схема проведення тесту:	Натиснути на кнопку “Прикріпити” у правій частині поля вводу та обрати потрібний документ зі списку файлів
Очікуваний результат:	Документ відправлено
Стан моделі після проведення випробувань:	Повідомлення з документов відображено в історії повідомлень

Таблиця 5.7 - Інформація про клієнта

Мета тесту	**Перевірка функції «Інформація про клієнта»**
Початковий стан моделі	Відкрито діалог з обраним користувачем
Вхідні дані:	
Схема проведення тесту:	Натиснути на кнопку “Інформація про клієнта” з випадаючого меню на екрані діалогу
Очікуваний результат:	Відображень спливаюче вікно з інформацією про клієнта
Стан моделі після проведення випробувань:	Відображено повідомлення, що відносяться до обраного діалогу

Таблиця 5.8 - Перехід на сторінку розсилки

Мета тесту	**Перевірка функції «Сторінка створення розсилки»**
Початковий стан моделі	Відкрито головну сторінку застосунку
Вхідні дані:	
Схема проведення тесту:	Натиснути на кнопку “Розсилки” у головному меню програми
Очікуваний результат:	Перехід на сторінку “Розсилки”
Стан моделі після проведення випробувань:	Відкрито сторінку “Розсилки”

Таблиця 5.9 - Створення розсилки

Мета тесту	**Перевірка функції «Створення розсилки»**
Початковий стан моделі	Відкрито сторінку “Розсилки”
Вхідні дані:	Текст повідомлення для розсилки
Схема проведення тесту:	Ввести текст повідомлення до текстового поля та натиснути на кнопку “Відправити”
Очікуваний результат:	Відображено повідомлення про успішне надсилання розсилки
Стан моделі після проведення випробувань:	Відображено повідомлення про успішне надсилання розсилки

Висновок до розділу

В даному розділі було детального описано інструкцію користувача з використання даної системи, зважаючи на роль у системі (адміністратор або оператор). Було перелічено список функціоналу, який пристуній у системі, за допомогою скріншотів та пояснень до них.

Було проведено тесстування функціоналу веб-застосунку, результати якого детально описано у вигляді таблиць.

Результати тестування цілком реалізують мету, тобто було перевірено, що функціонал системи підтримки користувачів з викорастанням сентиментального аналізу даних повністю відповідає вимогам, висунутим у технічному завданні до даної системи.

ЗАГАЛЬНІ ВИСНОВКИ

У даній роботі була вивчена наступна тема - «Комплекс задач з підтримки користувачів з використанням сентиментального аналізу даних». Була поставлена конкретна мета - підвищення якості надання підтримки користувачам. В результаті виконання дипломного проекту було виконано комплекс задач з підтримки користувачів, що надає можливість значно підвищити якість і швидкість надання підтримки. Поставлені ціль та мета задачі були реалізовані.

Даний проект створений як науково-пізнавальний програмний продукт з обмеженим функціоналом, але за умови деяких допрацювань він може бути використаний у великих комерційних продуктах або SaaS системах.

У першому розділі було розглянуто преметне середовище, визначені на описані основні процеси діяльності, створено функціональну модель системи, розглянуті аналогічні продукти на ринку.

У розділі з інформаційного забезпечення було детально описано вхідні та вихідні дані системи, описана структура бази даних.

У розділі «Математичне забезпечення» було розглянуто основні методи для розв’язання задачі сентиментального аналізу текстових даних, обрано та обґрунтовано запропонований підхід до вирішення задачі.

У розділі з програмного забезпечення було розглянуто програми, мови програмування та інструменти, за допомогою яких було реалізовано систему, було сформульовано загальні вимоги для технічних засобів та показано опис дій у модулях системи.

У технологічному розділі було детально описано інструкцію користувача з використання системи, перелічено список функціоналу, який було реалізовано, проведене тестування веб-застосунку із детальним поясненням до нього.

ПЕРЕЛІК ПОСИЛАНЬ

1. Telegram [Електронний ресурс] : https://telegram.org/
2. Viber [Електронний ресурс] : https://www.viber.com/
3. Facebook Messenger [Електронний ресурс] : https://www.messenger.com/
4. WhatsApp [Електронний ресурс] : https://www.whatsapp.com/
5. Telegram APIs [Електронний ресурс] : https://core.telegram.org/
6. Viber REST API [Електронний ресурс] : https://developers.viber.com/docs/api/rest-bot-api/
7. Facebook Messenger Platform [Електронний ресурс] : https://developers.facebook.com/docs/messenger-platform/
8. Monobank [Електронний ресурс] : https://www.monobank.ua/
9. ManyChat [Електронний ресурс] : https://manychat.com/
10. Leeloo [Електронний ресурс] : https://leeloo.ai/
11. TextBack [Електронний ресурс] : https://textback.ru/
12. Что такое HTTPS [Електронний ресурс] : https://cityhost.ua/blog/chto-takoe-https.html
13. JSON [Електронний ресурс] : https://www.json.org/
14. PostgreSQL [Електронний ресурс] : https://www.postgresql.org/
15. Sentiment Analysis [Електронний ресурс] : https://en.wikipedia.org/wiki/Sentiment_analysis
16. MoodMap - Correlating Sentiment Data from Tweets with Deprivation Data from the Government [Електронний ресурс] : http://themoodmap.co.uk
17. Pulse of the Nation: U.S. Mood Throughout the Day inferred from Twitter [Електронний ресурс] : http://www.ccs.neu.edu/home/amislove/twittermood/
18. Opinion Crawl - sentiment analysis tool for the Web and social media [Електронний ресурс] : http://opinioncrawl.com/

19. Business Social Media Scores | Groubal Community Sentiment Index [Електронний ресурс] : http://www.groubalcsi.com/
20. Lymbix. Sentiment Analysis Reinvented, [Електронний ресурс] : http://www.lymbix.com
21. PhpStorm [Електронний ресурс] : https://www.jetbrains.com/phpstorm/
22. pgAdmin [Електронний ресурс] : https://www.pgadmin.org/
23. draw.io [Електронний ресурс] : https://www.draw.io/
24. Adobe Photoshop [Електронний ресурс] : https://www.adobe.com/ru/products/photoshop.html
25. Node.js [Електронний ресурс] : https://nodejs.org/en/
26. Современный учебник JavaScript [Електронний ресурс] : https://learn.javascript.ru/
27. What is V8? [Електронний ресурс] : https://v8.dev/
28. Standart ECMA-262 [Електронний ресурс] : https://www.ecma-international.org/publications/standards/Ecma-262.htm
29. libuv [Електронний ресурс] : https://github.com/libuv/libuv
30. Как управлять Event Loop в JavaScript [Електронний ресурс] : https://skillbox.ru/media/code/event_loop_chast_1/
31. MySQL [Електронний ресурс] : https://www.mysql.com/
32. MariaDB [Електронний ресурс] : https://mariadb.org/
33. ACID properties in DBMS [Електронний ресурс] : https://www.geeksforgeeks.org/acid-properties-in-dbms/
34. RabbitMQ [Електронний ресурс] : https://www.rabbitmq.com/
35. AMQP 1.0 Specification [Електронний ресурс] : https://www.amqp.org/resources/specifications
36. TypeScript [Електронний ресурс] : https://www.typescriptlang.org/
37. Vue.js [Електронний ресурс] : https://vuejs.org/

38. Meriana Romanyshyn. Rule-based Sentiment Analysis of Ukrainian Reviews. – Intrnational Journal of Artificial Intelligence & Applications, Vol. 4, No. 4, July 2013
39. Илья Меньшиков. Анализ тональности текста на русском языке при помощи графовых моделей. – конференція 2016.
40. Nozomi Kobayashi. Opinion Mining on the Web by Extracting Subject-Aspect-Evaluation Relations. – Nara Institute of Scence and Technology, Nara 630-0192, 2016.

ДОДАТОК А

Тексти програмного коду

Комплекс задач з підтримки користувачів з використанням сентиментального анаілзу даних

(Найменування програми (документа))

DVD-R

(Вид носія даних)

20 арк, 30 МБ

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2019 року

Лістинг програми будемо розглядати у вигляді js файлів:

1. server.js

```
const express = require('express'),
bodyParser = require('body-parser'),
fileUpload = require('express-fileupload'),
passport = require('passport'),
Strategy = require('passport-facebook').Strategy,
createFbUser = require('./lib/controllers/user/create_fb_user'),
app = express();



app.use((req, res, next) => {
       res.header("Access-Control-Allow-Origin", "*");
       res.header("Access-Control-Allow-Methods", "POST, GET, PUT, DELETE, OPTIONS");
       res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type,
Accept, x-access-token");
       next();
});


app.use('/assets', express.static(__dirname + '/assets'));


passport.use(new Strategy({
        clientID: process.env.FB_CLIENT_ID,
        clientSecret: process.env.FB_CLIENT_SECRET,
        callbackURL: process.env.FB_CALLBACK_URL,
        profileFields: ['id', 'displayName', 'emails']
    },
    function(accessToken, refreshToken, profile, cb) {
        profile.accessToken = accessToken;
        profile.refreshToken = refreshToken;
        return cb(null, profile);
    })
);


passport.serializeUser(function(user, cb) {
    cb(null, user);
});


passport.deserializeUser(function(obj, cb) {
    cb(null, obj);
});


app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(require('cookie-parser')());
app.use(require('express-session')({ secret: 'keyboard cat', resave: true,
saveUninitialized: true }));
app.use(passport.initialize());


app.use(fileUpload({
    limits: { fileSize: 50 * 1024 * 1024 },
}));


app.listen(process.env.PORT, () => console.log(`started at ${process.env.PORT}`));
```

```
    app.get('/login/facebook', passport.authenticate('facebook', {
        scope: [
            'public_profile',
            'email',
            'manage_pages',
            'pages_messaging',
            'pages_messaging_subscriptions',
            'read_page_mailboxes'
        ]
        })
    );


    app.get('/login/facebook/return',
        passport.authenticate('facebook', {
            scope: [
                'public_profile',
                'email',
                'manage_pages',
                'pages_messaging',
                'pages_messaging_subscriptions',
                'read_page_mailboxes'
            ]
        }),
        createFbUser
    );
    app.options('*', (req, res) => { return res.status(200).end(); });
    const apiRoutes = express.Router();
    app.use('/api/v1', apiRoutes);
    require('./api/routes')(apiRoutes);
```

2. routes.js

```
const registerController = require('../../lib/controllers/user/register'),
      passwordRecoveryController = require('../../lib/controllers/user/password_recovery'),
      passwordCodeController = require('../../lib/controllers/user/password_code'),
      telegramAddBotController = require('../../lib/controllers/bot/telegram_add'),
      viberAddBotController = require('../../lib/controllers/bot/viber_add'),
      facebookAddBotController = require('../../lib/controllers/bot/facebook_add'),
      getBotsController = require('../../lib/controllers/bot/get'),
      authMiddleware = require('../../lib/middleware/auth'),
      subscriptionMiddleware = require('../../lib/middleware/subscription'),
      getMessagesController = require('../../lib/controllers/message/get'),
      getAnswersController = require('../../lib/controllers/answer/get'),
      getProfileController = require('../../lib/controllers/user/profile'),
      getChatsController = require('../../lib/controllers/chat/get'),
      sendMessageController = require('../../lib/controllers/message/send_text'),
      sendPhotoController = require('../../lib/controllers/message/send_photo'),
      sendFileController = require('../../lib/controllers/message/send_file'),
      markReadController = require('../../lib/controllers/message/mark_as_read'),
      sendMessageAllController = require('../../lib/controllers/message/send_all'),
      addAnswerController = require('../../lib/controllers/answer/add'),
      deleteAnswerController = require('../../lib/controllers/answer/delete'),
      addBotSettingsController = require('../../lib/controllers/bot_settings/add'),
      getBotSettingsController = require('../../lib/controllers/bot_settings/get'),
      updateBotSettingsController = require('../../lib/controllers/bot_settings/update'),
      answerController = require('../../lib/controllers/answer/answer'),
      generateWidgetController = require('../../lib/controllers/widget/generate'),
      linkWithFbUserController = require('../../lib/controllers/user/link_with_fb_user'),
      createTagController = require('../../lib/controllers/tag/create'),
      getTagController = require('../../lib/controllers/tag/get'),
      deleteTagController = require('../../lib/controllers/tag/delete'),
      updateTagController = require('../../lib/controllers/tag/update'),
      attachTagController = require('../../lib/controllers/tag/attach'),
      detachTagController = require('../../lib/controllers/tag/detach'),
```

```
        broadcastTemplateController =
require('../../lib/controllers/broadcast/broadcast_template'),
      broadcastController = require('../../lib/controllers/broadcast/broadcast'),
      getContactsController = require('../../lib/controllers/contact/get'),
      deleteContactController = require('../../lib/controllers/contact/delete'),
      loginController = require('../../lib/controllers/user/login');

module.exports = (app) => {

    app.options('*', (req, res) => { return res.status(200).end(); });
    app.post('/register', registerController);
    app.post('/login', loginController);
    app.post('/answer', answerController);
    app.post('/password/recovery', passwordRecoveryController);
    app.post('/password/code', passwordCodeController);
    app.post('/messages/photo', sendPhotoController);
    app.post('/messages/file', sendFileController);
    app.use(authMiddleware);
    app.use(subscriptionMiddleware);

    app.post('/bot/telegram', telegramAddBotController);
    app.post('/bot/viber', viberAddBotController);
    app.post('/bot/facebook', facebookAddBotController);
    app.get('/bot', getBotsController);
    app.get('/bot/:id', getBotsController);
    app.get('/messages/:chat_id', getMessagesController);
    app.post('/messages/read', markReadController);
    app.get('/chats', getChatsController);
    app.post('/messages', sendMessageController);

    app.get('/contacts/:id', getContactsController);
    app.get('/contacts', getContactsController);
    app.delete('/contacts/:id', deleteContactController);

    app.post('/messages/all', sendMessageAllController);
    app.post('/answer/add', addAnswerController);
    app.get('/answer/:bot_id', getAnswersController);
    app.delete('/answer/:answer_id', deleteAnswerController);

    app.post('/bot_settings', addBotSettingsController);
    app.get('/bot_settings/:id', getBotSettingsController);
    app.put('/bot_settings', updateBotSettingsController);

    app.post('/widget', generateWidgetController);

    app.get('/profile', getProfileController);

    app.get('/link_fb_user', linkWithFbUserController);

    app.post('/tag', createTagController);
    app.delete('/tag/:id', deleteTagController);
    app.patch('/tag/:id', updateTagController);
    app.get('/tag/:id', getTagController);
    app.get('/tag', getTagController);
    app.post('/tag/attach', attachTagController);
    app.post('/tag/detach', detachTagController);
    app.get('/broadcast/template', broadcastTemplateController.get);
    app.get('/broadcast/template/:id', broadcastTemplateController.get);
    app.post('/broadcast/template', broadcastTemplateController.create);
    app.put('/broadcast/template/:id', broadcastTemplateController.update);
    app.delete('/broadcast/template/:id', broadcastTemplateController.remove);
    app.get('/broadcast', broadcastController.get);
    app.get('/broadcast/:id', broadcastController.get);
    app.post('/broadcast', broadcastController.create);
    app.put('/broadcast/:id', broadcastController.update);
    app.delete('/broadcast/:id', broadcastController.remove);
};
```

3. pgdb.js

```
const { Pool } = require('pg');

const pool = new Pool({
    host: process.env.DB_HOST,
    database: process.env.DB_DATABASE,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    port: process.env.DB_PORT
});

pool.connect();

module.exports = pool;
```

4. auth.js

```
const jwt = require('jsonwebtoken');

module.exports = async (req, res, next) => {

    const token = req.headers['x-access-token'];

    if (!token) return res.status(401).send({'error':'Unauthorized.'});

    jwt.verify(token, process.env.TOKEN_SECRET, (err, data) => {
        if (err) return res.status(401).send({'error':'Unauthorized'});
        next();
    });

}
```

5. rabbitmq.js

```
module.exports.sendMessageToQueue = (data) => {
    const q = 'messages';

    const open = require('amqplib').connect('amqp://localhost');

    open.then(function(conn) {
        return conn.createChannel();
    }).then(function(ch) {
        return ch.assertQueue(q).then(function(ok) {
            return ch.sendToQueue(q, Buffer.from(data));
        });
    }).catch(console.warn);
};
```

6. telegram/send_message.js

```
const pool = require('../../db/pgdb'),
      setToLive = require('../../controllers/chat/set_to_live'),
      axios = require('axios');

module.exports = async ({api_key, bot_name, chat_id, message, bot_id}) => {


    const new_message = {
        text: message
    };
```

```
    const data = await
axios.post(`https://api.telegram.org/bot${api_key}/sendMessage`, {chat_id:
chat_id, text: message}).catch(e => { console.log(e); });

    if (data.data.ok) {
        pool.query(`
        insert into messages (chat_id, sender, message, read, bot_id) values
($1, $2, $3, true, $4)
        `, [chat_id, bot_name, new_message, bot_id]).catch(e =>
console.log(e));
        setToLive(api_key, chat_id);
    }
    return true;

};
```

7. telegram/send_file.js

```
const pool = require('../../db/pgdb'),
      setToLive = require('../../controllers/chat/set_to_live'),
      axios = require('axios');

module.exports = async ({ bot_id, api_key, bot_name, chat_id, url, filename
}) => {

    const new_message = {
        attachment: {
            type: 'file',
            filename: filename,
            url: url
        }
    };

    const data = await
axios.post(`https://api.telegram.org/bot${api_key}/sendDocument`, {
        chat_id: chat_id,
        document: url
    }).catch(e => { console.log(e); });

    if (data.data.ok) {
        pool.query(`
        insert into messages (chat_id, sender, message, read, bot_id) values
($1, $2, $3, true, $4)
        `, [chat_id, bot_name, new_message, bot_id]).catch(e =>
console.log(e));
        setToLive(api_key, chat_id);
    }
    return true;

};
```

8. telegram/send_photo.js

```
const pool = require('../../db/pgdb'),
    setToLive = require('../../controllers/chat/set_to_live'),
    axios = require('axios');

module.exports = async ({ bot_id, api_key, bot_name, chat_id, url, filename
}) => {

    const new_message = {
        attachment: {
```

```
            type: 'photo',
            filename: filename,
            url: url
        }
    };

    const data = await
axios.post(`https://api.telegram.org/bot${api_key}/sendPhoto`, {
        chat_id: chat_id,
        photo: url
    }).catch(e => { console.log(e); });

    if (data.data.ok) {
        pool.query(`
        insert into messages (chat_id, sender, message, read, bot_id) values
($1, $2, $3, true, $4)
        `, [chat_id, bot_name, new_message, bot_id]).catch(e =>
console.log(e));
        setToLive(api_key, chat_id);
    }
    return true;

};
```

9. viber/send_message.js

```
const pool = require('../../db/pgdb'),
      axios = require('axios');

module.exports = async ({ bot_id, api_key, bot_name, chat_id, message }) => {

    const options = {
        'headers': {
            'Content-Type': 'application/json',
            'X-Viber-Auth-Token': api_key
        }
    };
    const body = {
        receiver: chat_id,
        type: 'text',
        sender: {
            name: bot_name
        },
        text: message
    };

    const new_message = {
        text: message
    };

    const data = await
axios.post('https://chatapi.viber.com/pa/send_message', body,
options).catch(e => { console.log(e); });

    if (data.data.status_message === 'ok') {
        pool.query(`
            insert into messages (chat_id, sender, message, read, bot_id)
            values($1, $2, $3, true, $4)
            `, [chat_id, bot_name, new_message, bot_id]).catch(e => {
console.log(e); });
    }
};
```

10. viber/send_file.js

```
const pool = require('../../db/pgdb'),
    setToLive = require('../../controllers/chat/set_to_live'),
    axios = require('axios');

module.exports = async ({ bot_id, api_key, bot_name, chat_id, url, size,
filename }) => {

    const options = {
      'headers': {
          'Content-Type': 'application/json',
          'X-Viber-Auth-Token': api_key
      }
    };


    const body = {
        receiver: chat_id,
        type: 'file',
        sender: {
            name: bot_name
        },
        media: url,
        size: size,
        file_name: filename,
    };

    const new_message = {
      media: url,
      file_name: filename,
      type: 'file',
    };

    await axios.post('https://chatapi.viber.com/pa/send_message', body,
options).catch(e => { throw new Error(e); });

    pool.query(`
      insert into messages (chat_id, sender, message, read, bot_id) values
($1, $2, $3, true, $4)
    `, [chat_id, bot_name, new_message, bot_id]).catch(e => console.log(e));
    setToLive(api_key, chat_id);

};
```

11. viber/send_photo.js

```
const pool = require('../../db/pgdb'),
    setToLive = require('../../controllers/chat/set_to_live'),
    axios = require('axios');

module.exports = async ({ bot_id, api_key, bot_name, chat_id, url, filename
}) => {

    const options = {
      'headers': {
          'Content-Type': 'application/json',
          'X-Viber-Auth-Token': api_key
      }
    };


    const body = {
```

```
        receiver: chat_id,
        type: 'picture',
        sender: {
            name: bot_name
        },
        media: url,
        thumbnail: url
    };

    const new_message = {
        attachment: {
            type: 'photo',
            filename: filename,
            url: url
        }
    };


    await axios.post('https://chatapi.viber.com/pa/send_message', body,
options).catch(e => { console.log(e); });
    pool.query(`
        insert into messages (chat_id, sender, message, read, bot_id) values
($1, $2, $3, true, $4)
    `, [chat_id, bot_name, new_message, bot_id]).catch(e => console.log(e));
    setToLive(api_key, chat_id);

};
```

12. facebook/send_message.js

```
const pool = require('../../../lib/db/pgdb'),
      parseMessage = require('./format_message'),
      axios = require('axios');

module.exports = async ({ bot_id, api_key, bot_name, chat_id, contact_id,
message, attachment }) => {

    const new_message = {};
    if (message) new_message.text = message;
    if (attachment) new_message.attachment = attachment;

    const body = parseMessage(contact_id, new_message);

    const data = await
axios.post(`https://graph.facebook.com/v2.6/me/messages?access_token=${api_ke
y}`, body)
        .catch(e => { console.log(e); });

    if (data.status === 200) {
        await pool.query(`
        insert into messages (chat_id, sender, message, read, bot_id) values
($1, $2, $3, true, $4)
        `, [chat_id, bot_name, new_message, bot_id]).catch(e =>
console.log(e));
    }

};
```

13. message/send_text.js

```
const getUser = require('../token/get_user'),
      pool = require('../../db/pgdb'),
      viberSendMessage = require('../../helpers/viber/send_message'),
      telegramSendMessage = require('../../helpers/telegram/send_message'),
      facebookSendMessage = require('../../helpers/facebook/send_message'),
      axios = require('axios');

const { sendMessageToQueue } = require('../../helpers/rabbitmq');


module.exports = async (req, res) => {
    const chat_id = req.body.chat_id,
          message = req.body.message,
          token = req.headers['x-access-token'];
    const user_id = await getUser(token);
    const bot = await pool.query(`
        select
            bots.id as bot_id,
            bots.name,
            bots.api_key,
            bots.type,
            contacts.fb_id as fb_contact_id
        from users
        join bots on users.id=bots.user_id
        join chats on bots.id=chats.bot_id
        join contacts on chats.contact_id=contacts.id
        where users.id=$1 and chats.id=$2
    `, [user_id, chat_id]).catch(e => { console.log(e); });
    if (!bot['rows'].length) return res.status(404).send({'error':'Chat not
found.'});
    const api_key = bot['rows'][0]['api_key'],
          bot_id = bot['rows'][0]['bot_id'],
          bot_type = bot['rows'][0]['type'],
          fb_contact_id = bot['rows'][0]['fb_contact_id'],
          bot_name = bot['rows'][0]['name'];

    if (bot_type === 'telegram') {
        telegramSendMessage({ bot_id, api_key, bot_name, chat_id, message });
    } else if (bot_type ==='viber') {
        viberSendMessage({ bot_id, api_key, bot_name, chat_id, message });
    } else if (bot_type === 'facebook') {
        facebookSendMessage({ bot_id, api_key, bot_name, chat_id, contact_id:
fb_contact_id, message });
    }
    sendMessageToQueue(JSON.stringify(
        {
            name: bot_name,
            bot_id: bot_id,
            sender: bot_name,
            date: new Date().toISOString(),
            chat_id: chat_id,
            user_id: user_id,
            message: {
                text: message
            }
        }
        ));
    return res.status(200).end();

};
```

14. message/send_photo.js

```
const pool = require('../../db/pgdb');
const getUser = require('../token/get_user');
const telegramSendPhoto = require('../../helpers/telegram/send_photo');
const facebookSendPhoto = require('../../helpers/facebook/send_message');
const viberSendPhoto = require('../../helpers/viber/send_photo');
const saveFile = require('../../helpers/attachments/save_file');
const { sendMessageToQueue } = require('../../helpers/rabbitmq');


module.exports = async (req, res) => {

    const token = req.headers['x-access-token'],
          chat_id = req.body.chat_id;

    const user_id = await getUser(token);
    if (!token || !chat_id || !user_id) return res.status(400).send({'error':
'Bad request'});
    if (!req.files || !req.files.photo) return res.status(400).send({'error':
'No photo was sent'});

    const { rows: [bot] } = await pool.query(`
        select
            bots.name,
            bots.id,
            bots.api_key,
            bots.type,
            chats.id as chat_id,
            contacts.fb_id as fb_contact_id
        from users
        join bots on users.id=bots.user_id
        join chats on bots.id=chats.bot_id
        join contacts on chats.contact_id=contacts.id
        where users.id=$1 and chats.id=$2
    `, [user_id, chat_id]).catch(e => { console.log(e); });


    const photo = req.files.photo;
    const filename = photo.name;
    const extenstion = photo.name.split('.').pop();

    if (extenstion !== 'png' && extenstion !== 'jpg' && extenstion !==
'jpeg') {
        return res.status(400).send({'error': 'Incorrect extension'});
    }

    let { url } = await saveFile(photo);

    if (bot.type === 'telegram') {
        telegramSendPhoto({
            bot_id: bot.id,
            api_key: bot.api_key,
            bot_name: bot.name,
            chat_id,
            url,
            filename
        }).catch(e => { console.log(e); });
    } else if (bot.type === 'facebook') {
        facebookSendPhoto({
            bot_id: bot.id,
            api_key: bot.api_key,
```

```
            bot_name: bot.name,
            contact_id: bot.fb_contact_id,
            chat_id,
            attachment: {
                url,
                filename,
                type: 'photo'
            }
        })
    } else if (bot.type === 'viber') {
        await viberSendPhoto({
            bot_id: bot.id,
            api_key: bot.api_key,
            bot_name: bot.name,
            chat_id,
            url,
            filename
        })
    }

    sendMessageToQueue(JSON.stringify(
        {
            name: bot.name,
            bot_id: bot.id,
            sender: bot.name,
            date: new Date().toISOString(),
            chat_id: bot.chat_id,
            user_id: user_id,
            message: {
                attachment: {
                    type: 'photo',
                    filename,
                    url
                }
            }
        }
    ));


    res.status(200).end();

};
```

15. message/send_file.js

```
const pool = require('../../db/pgdb');
const getUser = require('../token/get_user');
const telegramSendFile = require('../../helpers/telegram/send_file');
const viberSendFile = require('../../helpers/viber/send_file');
const facebookSendFile = require('../../helpers/facebook/send_message');
const saveFile = require('../../helpers/attachments/save_file');

const { sendMessageToQueue } = require('../../helpers/rabbitmq');

module.exports = async (req, res) => {

    const token = req.headers['x-access-token'],
        chat_id = req.body.chat_id;

    const user_id = await getUser(token);
    if (!token || !chat_id || !user_id) return res.status(400).send({'error':
'Bad request'});
```

```
    if (!req.files || !req.files.file) return res.status(400).send({'error':
'No file was sent'});

    const { rows: [bot] } = await pool.query(`
        select
            bots.id,
            bots.name,
            chats.id as chat_id,
            bots.api_key,
            bots.type,
            contacts.fb_id as fb_contact_id
        from users
        join bots on users.id=bots.user_id
        join chats on bots.id=chats.bot_id
        join contacts on chats.contact_id=contacts.id
        where users.id=$1 and chats.id=$2
    `, [user_id, chat_id]).catch(e => { console.log(e); });


    const file = req.files.file;
    const filename = file.name;
    const extenstion = file.name.split('.').pop();

    if (!['zip', 'rar', 'txt', 'doc', 'docx', 'xlsx', 'xls',
'pdf'].includes(extenstion)) {
        return res.status(400).send({'error': 'This type of file is
restricted.'});
    }

    let { url, size } = await saveFile(file);

    if (bot.type === 'telegram') {
        telegramSendFile({
            bot_id: bot.id,
            api_key: bot.api_key,
            bot_name: bot.name,
            chat_id,
            url,
            filename
        }).catch(e => { console.log(e); });
    } else if (bot.type === 'facebook') {
        facebookSendFile({
            bot_id: bot.id,
            api_key: bot.api_key,
            bot_name: bot.name,
            contact_id: bot.fb_contact_id,
            chat_id,
            attachment: {
                url,
                filename,
                type: 'file'
            }
        })
    } else if (bot.type === 'viber') {
        viberSendFile({
            bot_id: bot.id,
            api_key: bot.api_key,
            bot_name: bot.name,
            chat_id,
            url,
            size,
            filename
```

```
        })
    }

    sendMessageToQueue(JSON.stringify(
        {
            name: bot.name,
            bot_id: bot.id,
            sender: bot.name,
            date: new Date().toISOString(),
            chat_id: bot.chat_id,
            user_id: user_id,
            message: {
                attachment: {
                    type: 'file',
                    filename,
                    url
                }
            }
        }
    ));

    res.status(200).end();

};
```

16. message/send_all.js

```
const pool = require('../../db/pgdb'),
      viberSendMessage = require('../../helpers/viber/send_message'),
      telegramSendMessage = require('../../helpers/telegram/send_message'),
      getUser = require('../token/get_user');

module.exports = async (req, res) => {
    const token = req.headers['x-access-token'],
          message = req.body.message,
          bot_id = req.body.bot_id;
    if (!bot_id || !message) return res.status(400).send({'error':'Bad
request.'});
    const user_id = await getUser(token);
    const data = await pool.query(`
        select bots.api_key, bots.type, bots.name, chats.id chat_id from
users
        join bots on users.id=bots.user_id
        join chats on bots.id=chats.bot_id
        where users.id=$1 and bots.id=$2
    `, [user_id, bot_id]).catch(e => { console.log(e); });
    if (!data['rows'].length) return res.status(404).send({'error':'Bot not
found.'});
    const api_key = data['rows'][0]['api_key'],
          bot_type = data['rows'][0]['type'],
          bot_name = data['rows'][0]['name'];
    if (bot_type === 'telegram') {
        data['rows'].forEach(item => {
            telegramSendMessage(api_key, bot_name, item.chat_id, message);
        });
    }
    if (bot_type === 'viber') {
        data['rows'].forEach(item => {
            viberSendMessage(api_key, bot_name, item.chat_id, message);
        });
    }
    return res.status(200).end();
};
```

17. message/get.js

```
const pool = require('../../db/pgdb'),
      getUser = require('../token/get_user');

module.exports = async (req, res) => {

    const token = req.headers['x-access-token'],
          chat_id = req.params.chat_id,
          limit = req.query.limit || 50;

    const user_id = await getUser(token);
    const data = await pool.query(`
        select bots.name, bots.id bot_id, messages.sender, messages.message,
messages.date, chats.id chat_id from users
        join bots on users.id=bots.user_id
        join chats on bots.id=chats.bot_id
        join messages on chats.id=messages.chat_id and
chats.bot_id=messages.bot_id
        where chats.id=$1 and users.id=$2
        order by messages.date asc
    `, [chat_id, user_id]).catch(e => { console.log(e); });
    res.status(200).send({'data':data['rows']});

};
```

18. chat/get.js

```
const pool = require('../../db/pgdb'),
      getUser = require('../token/get_user');

module.exports = async (req, res) => {
    const token = req.headers['x-access-token'],
          bot_id = req.query.bot_id;

    const user_id = await getUser(token);
    let data = { };

    if (!bot_id) {
        data = await pool.query(`
            select
                bots.id,
                bots.name bot_name,
                bots.nickname,
                chats.id,
                contacts.id as contact_id,
                contacts.name,
                contacts.avatar,
                contacts.is_blocked,
                bots.type,
                mes.message,
                mes.sender,
                mes.date,
                unr.unread_count
            from chats
            left join bots on bots.id=chats.bot_id
            left join contacts on chats.contact_id=contacts.id
            left join users on bots.user_id=users.id
            left join
                (select bot_id, chat_id, message, sender, date from messages
where id in
                  (select max(id) from messages group by chat_id, bot_id)
                ) mes
```

```
            on mes.chat_id=chats.id and mes.bot_id=chats.bot_id
            left join
                 (select
                  count(chats.id) as unread_count,
                  chats.id as chat_id
                  from messages
                  join chats on messages.chat_id=chats.id and
messages.bot_id=chats.bot_id
                  join bots on chats.bot_id=bots.id
                  join users on bots.user_id=users.id
                  where messages.read=false and users.id=$1
                  group by chats.id) unr
            on chats.id=unr.chat_id
            where users.id=$1
            order by mes.date desc
        `, [user_id]).catch(e => { console.log(e); });
    } else {
        data = await pool.query(`
            select
                bots.id,
                bots.name bot_name,
                bots.nickname,
                chats.id,
                contacts.id as contact_id,
                contacts.name,
                contacts.avatar,
                contacts.is_blocked,
                bots.type,
                mes.message,
                mes.sender,
                mes.date,
                unr.unread_count
            from chats
            left join bots on bots.id=chats.bot_id
            left join contacts on chats.contact_id=contacts.id
            left join users on bots.user_id=users.id
            left join
                (select bot_id, chat_id, message, sender, date from messages
where id in
                  (select max(id) from messages group by chat_id, bot_id)
                ) mes
            on mes.chat_id=chats.id and mes.bot_id=chats.bot_id
            left join
                 (select
                  count(chats.id) as unread_count,
                  chats.id as chat_id
                  from messages
                  join chats on messages.chat_id=chats.id and
messages.bot_id=chats.bot_id
                  join bots on chats.bot_id=bots.id
                  join users on bots.user_id=users.id
                  where messages.read=false and users.id=$1
                  group by chats.id) unr
            on chats.id=unr.chat_id
            where users.id=$1 and bots.id=$2
            order by mes.date desc
        `, [user_id, bot_id]).catch(e => { console.log(e); });
    }
    if (!data['rows'].length) return res.status(200).send({'data':[]});
    res.status(200).send({'data':data['rows']});

};
```

19. bot/telegram_add.js

```
const rp = require('request-promise'),
      request = require('request'),
      pool = require('../../db/pgdb'),
      getUser = require('../token/get_user'),
      fs = require('fs');

const server_address = process.env.SERVER_ADDRESS;


function getBotProfilePhotoId(api_key, bot_id) {
  return
rp.get(`https://api.telegram.org/bot${api_key}/getUserProfilePhotos?user_id=$
{bot_id}`)
    .then(res => {
      res = JSON.parse(res);
      return res['result']['photos'][0][0]['file_id'];
    })
    .catch(e => {
      console.log('error in getting profile photo id', e.error);
    })
}

function getBotProfilePhotoPath(api_key, file_id) {
  return
rp.get(`https://api.telegram.org/bot${api_key}/getFile?file_id=${file_id}`)
    .then(res => {
      res = JSON.parse(res);
      return res['result']['file_path'];
    })
    .catch(e => {
      console.log('error in getting profile photo path', file_id, e);
    })
}

function getBotProfilePhoto(api_key, file_path) {
  return new Promise((resolve, reject) => {
    request.get(`https://api.telegram.org/file/bot${api_key}/${file_path}`,
(err, res, body) => {
      if (err) {
        console.log('error getting profile photo', file_path, err);
        reject();
        return;
      }
      let filename = 'assets/bot_avatars/' +
Math.random().toString(36).substring(2, 15) +
Math.random().toString(36).substring(2, 15) + '.jpg';
      let file = fs.createWriteStream(filename);
      file.write(res.body);
      resolve(filename);
    });
  });


}

function saveBot(api_key, bot) {
  return new Promise((resolve, reject) => {
    pool.query('insert into bots (id, name, nickname, avatar, type, api_key,
user_id) values ($1, $2, $3, $4, $5, $6, $7)', [bot.id, bot.name,
```

```
bot.nickname, bot.filename, 'telegram', api_key, bot.user_id], (err, result)
=> {
      if (err)
        console.log(err);
        reject(err);
      resolve();
    });
  });
}

function setBotWebhook(api_key, bot_id) {
  let url = server_address + `/telegram/${bot_id}/`;
  const options = {
    method: 'POST',
    url: `https://api.telegram.org/bot${api_key}/setWebhook`,
    headers: {
     'content-type': 'multipart/form-data; boundary=----
WebKitFormBoundary7MA4YWxkTrZu0gW'
    },
    formData: {
      url
    }
  };
  return new Promise((resolve, reject) => {
    request(options, function (error, response, body) {
      if (error)
        reject(error.message);
      resolve(body);
    });
  });
}

async function addBot(req, res) {
  let api_key = req.body.api_key;
  let user_id = await getUser(req.headers['x-access-token']);
  let bot = {};
  if (!api_key) return res.status(400).end();
  rp.get(`https://api.telegram.org/bot${api_key}/getMe`)
  .then(data => {
    data = JSON.parse(data);
    bot.user_id = user_id;
    bot.name = data['result']['first_name'];
    bot.nickname = data['result']['username'];
    bot.id = data['result']['id'];
  })
  .then(() => {
    return getBotProfilePhotoId(api_key, bot.id)
    .then(file_id => {
      bot.photo_file_id = file_id;
    })
  })
  .then(() => {
    return getBotProfilePhotoPath(api_key, bot.photo_file_id)
    .then(file_path => {
      bot.photo_file_path = file_path;
    })
  })
  .then(() => {
    return getBotProfilePhoto(api_key, bot.photo_file_path)
    .then(filename => {
      bot.filename = filename;
    })
```

```
  })
  .then(() => {
    return setBotWebhook(api_key, bot.id);
  })
  .then((res) => {
    return saveBot(api_key, bot);
  }).catch(e => {
    return res.status(400).send({'error':'Invalid api key'})
  });
  return res.status(200).send({'message':'OK.'});
}

module.exports = addBot;
```

20. bot/viber_add.js

```
const pool = require('../../db/pgdb'),
      axios = require('axios');
      getUser = require('../token/get_user');

module.exports = async (req, res) => {

    const token = req.headers['x-access-token'],
          api_key = req.body.api_key;

    const user_id = await getUser(token);

    const options = {
        'headers': {
            'Content-Type': 'application/json',
            'X-Viber-Auth-Token': api_key
        }
    };

    const data = await
axios.get('https://chatapi.viber.com/pa/get_account_info', options);
    if (data.data.status_message !== 'ok') return
res.status(404).send({'error':'Bot not found.'});
    const id = data.data.id,
          name = data.data.name,
          nickname = data.data.uri,
          avatar = data.data.icon;

    pool.query(`
        insert into bots (id, name, nickname, avatar, type, api_key, user_id)
        values ($1, $2, $3, $4, $5, $6, $7)
    `, [id, name, nickname, avatar, 'viber', api_key, user_id]).catch(e =>
res.status(400).send({'error':'Bot already added.'}));

    const body = {
        'url': `https://hellomeassage.space/webhook/viber/${api_key}`,
        "event_types":[
            "delivered",
            "seen",
            "failed",
            "subscribed",
            "unsubscribed",
            "conversation_started"
        ],
        "send_name": true,
        "send_photo": true
    };
    console.log(data.data);
```

```
    const d = await axios.post('https://chatapi.viber.com/pa/set_webhook',
body, options).catch(e => console.log(e));
    console.log(d.data);

    return res.status(200).end();
};
```

21. bot/facebook_add.js

```
const pool = require('../../db/pgdb'),
      getUser = require('../token/get_user');

const axios = require('axios');

module.exports = async (req, res) => {

    const token = req.headers['x-access-token'],
          user_id = await getUser(token),
          page_id = req.body.page_id;

    let { rows: [page] } = await pool.query(`
        select fp.fb_id, fp.fb_user_id, fp.access_token, fp.name
        from fb_pages fp
        join fb_users fu on fu.fb_id=fp.fb_user_id
        join users u on fu.user_id=u.id
        where u.id=$1 and fp.fb_id=$2
    `, [user_id, page_id]);

    if (!page) {
        return res.status(400).send({'error':'Page does not exist for this
user'});
    }

    const { rows: fb_bots } = await pool.query(`select * from bots where
user_id=$1 and type='facebook'`, [user_id])
        .catch(e => { console.log(e); });

    if (fb_bots.length) {
        return res.status(400).send({'error':'You already have facebook
bot'});
    }

    await
axios.post(`https://graph.facebook.com/v3.2/${page_id}/subscribed_apps?access
_token=${page.access_token}`, {
        subscribed_fields: ['messages']
    }).catch(e => {
        console.log(e);
    });

    await pool.query(`
        insert into bots (id, name, nickname, type, api_key, user_id)
        values ($1, $2, $3, 'facebook', $4, $5)
    `, [page.fb_id, page.name, page.name, page.access_token,
user_id]).catch(e => { console.log(e); });

    return res.status(200).end();

};
```

22. user/login.js

```
const pool = require('../../db/pgdb'),
      compare = require('../password/compare'),
      generateToken = require('../token/generate');

module.exports = async (req, res) => {

    let email = req.body.email,
        password = req.body.password;

    let data = await pool.query('select * from users where email=$1',
[email]).catch(e => { console.log(e); });
    if (!data['rows'].length)
        return res.status(400).send({'error':'Invalid user credentials.'});

    let password_hash = data['rows'][0]['password'],
        user_id = data['rows'][0]['id'];

    if (await compare(password, password_hash)) {
        let token = generateToken(user_id);
        return res.status(200).send({'message':'OK.', 'token':token});
    } else {
        return res.status(400).send({'error':'Invalid user credentials.'});
    }

};
```

23. user/profile.js

```
const pool = require('../../db/pgdb'),
      getUser = require('../token/get_user');
module.exports = async (req, res) => {
    const token = req.headers['x-access-token'];
    const user_id = await getUser(token);
    const user = await pool.query(`
      select * from users
      where users.id=$1
    `, [user_id]);
    const result = {
        name: user['rows'][0]['name'],
        email: user['rows'][0]['email'],
        bots: []
    };
    const bots = await pool.query(`
       select * from bots
       where user_id=$1
    `, [user_id]);

    result.bots = bots['rows'].map(item => {
        return {
            id: item.id,
            name: item.name,
            nickname: item.nickname,
            type: item.type,
            avatar: item.avatar
        }
    });

    return res.status(200).send({'data':result});

};
```

24. webhook-server.js

```
require('dotenv').config();
const express = require('express'),
          bodyParser = require('body-parser'),
      telegramNewMessage = require('./libs/telegram/telegram_update'),
      viberNewMessage = require('./libs/viber/viber_update'),
      facebookNewMessage = require('./libs/facebook/facebook_update'),
          app = express();

app.use((req, res, next) => {
        res.header("Access-Control-Allow-Origin", "*");
        res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With,
Content-Type, Accept");
        next();
});

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.listen(8443);

app.post('/webhook/telegram/:bot_id', telegramNewMessage);
app.post('/webhook/viber/:bot_id', viberNewMessage);
app.post('/webhook/facebook/', facebookNewMessage);
```

25. save_message.js

```
const pool = require('../pgdb');
const axios = require('axios');
const {sendMessageToQueue} = require('./rabbitmq.js');
module.exports = async ({ message, chat_id, sender, bot_id }) => {
    const options = {
        headers: {
            'Content-Type': 'application/json'
        }
    };
    const body = {
        chat_id: chat_id,
        bot_id: bot_id,
        sender: sender,
        message: message,
        secret: 'casino'
    };
    await pool.query(`
        insert into messages (chat_id, sender, message, bot_id) values ($1,
$2, $3, $4)
    `, [chat_id, sender, message, bot_id]);
    axios.post(process.env.SERVER_ANSWER_ROUTE, body, options).catch(e => {
console.log(e); });

    const { rows: [{ user_id, bot_name }] } = await pool.query(`
      select bots.name as bot_name, users.id as user_id from users
      join bots on users.id=bots.user_id
      where bots.id=$1
    `, [bot_id]).catch(e => { console.log(e); });
    body.user_id = user_id;
    body.date = new Date().toISOString();
    body.name = bot_name;
    sendMessageToQueue(JSON.stringify(
        body
    ));
};
```

26. viber_update.js

```
const fileHandler = require('./handlers/file_handler'),
      textMessageHandler = require('./handlers/text_message_handler');

module.exports = async (req, res) => {
    if (req.body.event === 'message') {
        const message_type = req.body.message.type,
              api_key = req.params.bot_id;
        if (message_type === 'text') {
            textMessageHandler(api_key, req.body);
        } else {
            fileHandler(api_key, req.body);
        }

    }
    console.log(req.body);
    return res.status(200).end();
};
```

27. telegram_update.js

```
const textMessageHandler = require('./handlers/text_message_handler'),
      fileHandler = require('./handlers/file_handler');

module.exports = async (req, res) => {
    console.log(req.body);
    const update = req.body,
          bot_id = req.params.bot_id;
    if (!update.message) return res.status(200).end();
    if (update.message.text || update.message.sticker) {
        textMessageHandler(bot_id, update);
    }

    if (update.message.document || update.message.photo) {
        fileHandler(bot_id, update);
    }

    return res.status(200).end();

};
```

28. facebook_update.js

```
const fileHandler = require('./handlers/file_handler'),
      textMessageHandler = require('./handlers/text_message_handler'),
      axios = require('axios');

module.exports = async (req, res) => {

    if (!req.body.entry) return res.status(200).end();

    if (req.body.entry[0].messaging[0].message.attachments) {
        fileHandler(req.body);
    } else {
        textMessageHandler(req.body);
    }

    res.status(200).end();
};
```

29. viber/text_message_handler.js

```
const pool = require('../../../pgdb'),
      saveMessage = require('../../save_message');

module.exports = async (api_key, update) => {

    const sender_name = update.sender.name || '',
          sender_id = update.sender.id,
          sender_avatar = update.sender.avatar || '',
          message_text = update.message.text;

    let bot_id = await pool.query('select * from bots where api_key=$1',
[api_key]).catch(e => { console.log(e); }) ;
    if (!bot_id['rows'].length) return res.status(200).end();
    bot_id = bot_id['rows'][0]['id'];
    const existing_chat = await pool.query('select * from chats where id=$1',
[sender_id]).catch(e => { console.log(e); });

    let chat_id = '';
    if (!existing_chat.rows.length) {
        const new_contact = await pool.query(`
          insert into contacts (name, nickname, avatar) values ($1, $2, $3)
returning *
        `, [sender_name, sender_id, sender_avatar]).catch(e => {
console.log(e); });
        const contact_id = new_contact['rows'][0]['id'];
        const chat = await pool.query(`
          insert into chats (bot_id, contact_id, id) values ($1, $2, $3)
returning *
        `, [bot_id, contact_id, sender_id]).catch(e => { console.log(e); });
        chat_id = chat['rows'][0]['id'];
    } else {
        chat_id = existing_chat['rows'][0]['id'];
    }

    const message = {
        text: message_text
    };

    saveMessage({
        message: message,
        chat_id: chat_id,
        sender: sender_name,
        bot_id: bot_id
    });

};
```

30. viber/file_handler.js

```
const pool = require('../../../pgdb'),
      saveMessage = require('../../save_message');

module.exports = async (api_key, update) => {

    const sender_name = update.sender.name || '',
          sender_id = update.sender.id,
          sender_avatar = update.sender.avatar || '',
          message_text = update.message.text;

    let bot_id = await pool.query('select * from bots where api_key=$1',
[api_key]).catch(e => { console.log(e); }) ;
```

```
    if (!bot_id['rows'].length) return res.status(200).end();
    bot_id = bot_id['rows'][0]['id'];
    const existing_chat = await pool.query('select * from chats where id=$1',
[sender_id]).catch(e => { console.log(e); });

    let chat_id = '';
    if (!existing_chat.rows.length) {
        const new_contact = await pool.query(`
          insert into contacts (name, nickname, avatar) values ($1, $2, $3)
returning *
        `, [sender_name, sender_id, sender_avatar]).catch(e => {
console.log(e); });
        const contact_id = new_contact['rows'][0]['id'];
        const chat = await pool.query(`
          insert into chats (bot_id, contact_id, id) values ($1, $2, $3)
returning *
        `, [bot_id, contact_id, sender_id]).catch(e => { console.log(e); });
        chat_id = chat['rows'][0]['id'];
    } else {
        chat_id = existing_chat['rows'][0]['id'];
    }

    const message = {
        text: message_text
    };

    saveMessage({
        message: message,
        chat_id: chat_id,
        sender: sender_name,
        bot_id: bot_id
    });

};
```

31. telegram/text_message_handler.js

```
const pool = require('../../../pgdb'),
      saveMessage = require('../../save_message');

module.exports = async (bot_id, update) => {

    const chat_id = update.message.chat.id;
    const existing_chat = await pool.query(`
        select chats.id from bots join chats on bots.id=chats.bot_id where
bots.id=$1 and chats.id=$2
    `, [bot_id, chat_id]).catch(e => { console.log(e); });

    const message = {};

    if (update.message.text) {
        message.text = update.message.text;
    } else if (update.message.sticker) {
        message.text = update.message.sticker.emoji;
    }

    const sender = update.message.from.first_name + ' ' +
update.message.from.last_name;

    if (!existing_chat['rows'].length) {
        const contact = await pool.query(`
            insert into contacts (name, nickname) values ($1, $2) returning *
```

```
        `, [update.message.chat.first_name + ' ' +
update.message.chat.last_name, update.message.chat.username])
            .catch(e => { console.log(e); });
        await pool.query('insert into chats (id, bot_id, contact_id) values
($1, $2, $3) returning *', [chat_id, bot_id,
contact['rows'][0]['id']]).catch(e => { console.log(e); });
    }

    saveMessage({
        message: message,
        chat_id: chat_id,
        sender: sender,
        bot_id: bot_id
    });

};
```

32. telegram/file_handler.js

```
const pool = require('../../../pgdb'),
    generateFilename = require('../../helpers/generate_filename'),
    downloadFile = require('../../helpers/download_file'),
    saveMessage = require('../../save_message'),
    axios = require('axios');

module.exports = async (bot_id, update) => {
    const chat_id = update.message.chat.id;
    const sender = update.message.from.first_name + ' ' +
update.message.from.last_name;


    const data = await pool.query('select api_key from bots where id=$1',
[bot_id])
        .catch(e => { console.log(e); });

    const api_key = data['rows'][0]['api_key'];
    let file_id = '';
    let directory_path = '';
    let type = '';
    let file_name = null;
    if (update.message.document) {
        file_id = update.message.document.file_id;
        file_name = update.message.document.file_name;
        directory_path = 'assets/files/';
        type = 'file';
    } else if (update.message.photo) {
        file_id = update.message.photo[update.message.photo.length -
1]['file_id'];
        directory_path = 'assets/photos/';
        type = 'photo';
    }

    console.log(file_id);

    let res = await
axios.get(`https://api.telegram.org/bot${api_key}/getFile?file_id=${file_id}`
)
        .catch(e => { console.log(e); });

    const path = res.data.result.file_path;
    const extension = path.split('.').pop();

    const link = `https://api.telegram.org/file/bot${api_key}/${path}`;
```

```
    const filename = directory_path + generateFilename(extension);

    const message = {
        attachment: {
            type: type,
            url: process.env.FILE_STORAGE_URL + filename,
            filename: process.env.HOST + file_name
        }
    };



    downloadFile(link, filename, () => {
        saveMessage({
            message: message,
            bot_id: bot_id,
            chat_id: chat_id,
            sender: sender
        });
    });

};
```

33. facebook/text_message_handler.js

```
const pool = require('../../../pgdb'),
      saveMessage = require('../../save_message');

module.exports = async (update) => {

    const msg = update.entry[0]['messaging'][0],
          sender_id = msg.sender.id,
          bot_id = msg.recipient.id,
          text = msg.message.text,
          chat_id = bot_id + '' + sender_id;

    if (!msg) return res.status(200).end();

    const { rows: data } = await pool.query(`
        select
        *
        from chats join contacts on chats.contact_id=contacts.id where
chats.id=$1
    `, [chat_id]).catch(e => { console.log(e); });

    let sender = '';
    if (!data.length) {
        let { rows: api_key } = await pool.query('select api_key from bots
where id=$1', [bot_id]).catch(e => { console.log(e); });
        api_key = api_key[0]['api_key'];
        const contact = await fetchContact(api_key, sender_id);
        const contact_name = contact.first_name + ' ' + contact.last_name;
        const { rows: new_contact } = await pool.query(`insert into contacts
(name, nickname, avatar, fb_id) values ($1, $2, $3, $4) returning *`,
            [contact_name, contact_name, contact.profile_pic,
sender_id]).catch(e => { console.log(e); });
        const { rows: new_chat } = await pool.query(`insert into chats
(bot_id, contact_id, id) values ($1, $2, $3) returning *`, [bot_id,
new_contact[0]['id'], chat_id]);
        sender = new_contact[0]['name'];
    } else {
        sender = data[0]['name'];
    }
```

```
    const message = {
        text: text
    };

    saveMessage({
        chat_id: chat_id,
        bot_id: bot_id,
        message: message,
        sender: sender
    });

};

const fetchContact = async (access_token, id) => {
    const response = await
axios.get(`https://graph.facebook.com/v3.1/${id}/?access_token=${access_token
}`)
        .catch(e => { console.log(e); });
    return response.data;
};
```

34. facebook/file_handler.js

```
const pool = require('../../../pgdb'),
      generateFilename = require('../../helpers/generate_filename'),
      saveMessage = require('../../save_message');

module.exports = async (update) => {
    const msg = update.entry[0]['messaging'][0],
          sender_id = msg.sender.id,
          bot_id = msg.recipient.id,
          chat_id = bot_id + '' + sender_id,
          attachments = update.entry[0].messaging[0].message.attachments;
    console.log(attachments);
    const { rows: data } = await pool.query(`
        select
        *
        from chats join contacts on chats.contact_id=contacts.id where
chats.id=$1
    `, [chat_id]).catch(e => { console.log(e); });
    const sender = data[0]['name'];
    attachments.forEach(async attachment => {
        const message = { attachment: { } };
        switch (attachment.type) {
            case 'image': {
                message.attachment.type = 'photo';
                break;
            }
            default: {
                message.attachment.filename = generateFilename('');
                message.attachment.type = 'file';
            }
        }
        message.attachment.url = attachment.payload.url;
        await saveMessage({
            message: message,
            bot_id: bot_id,
            sender: sender,
            chat_id: chat_id
        });
    });
};
```

35. socket-server.js

```
const io = require('socket.io')(8090);
const rabbit = require('amqplib').connect('amqp://localhost');
const connections = require('./connections');
const {getUserId} = require('./helpers/token');
const emitter = require('./emitter');

const q = 'messages';

rabbit.then((conn) => {
    return conn.createChannel();
}).then(function(ch) {
    return ch.assertQueue(q).then(ok => {
        return ch.consume(q, msg => {
            if (msg !== null) {
                const message = JSON.parse(msg.content.toString());
                emitter.emit('message', message);
                ch.ack(msg);
            }
        });
    });
}).catch(console.warn);

io.on('connection', async socket => {
    const token = socket.handshake.query.token;
    const user_id = getUserId(token);
    if (!user_id) {
        socket.disconnect();
    }
    let socket_id = socket.conn.id;
    if (connections.has(user_id)) {
        const data = connections.get(user_id);
        data.push(socket_id);
        connections.set(user_id, data);
    } else {
        connections.set(user_id, [socket_id]);
    }

    socket.on('disconnect', () => {
        const data = connections.get(user_id);
        const index = data.findIndex(item => item === socket_id);
        data.splice(index, 1);
        connections.set(user_id, data);
    });

});

emitter.on('message', message => {
    const sockets = connections.get(message.user_id);
    delete message.user_id;
    sockets.forEach(socket => {
        io.to(socket).emit('message', message);
    });
});
```

36. language-processor.js

```
var uaLanguage = require('../languages/ua/index');
var languages = {
    ua: uaLanguage
};

module.exports = {
    addLanguage: function (languageCode, language) {
        if (!language.labels) {
            throw new Error('language.labels must be defined!');
        }
        // Add emojis
        Object.assign(language.labels, emojis);
        languages[languageCode] = language;
    },
    getLanguage: function(languageCode) {
        if (!languageCode) {
            // Default to english if no language was specified
            return languages.en;
        }
        if (!languages[languageCode]) {
            // Try to load specified language
            try {
                // eslint-disable-next-line max-len
                var language = require('../languages/' + languageCode +
'/index');
                // Add language to in-memory cache
                this.addLanguage(languageCode, language);
            } catch (err) {
                throw new Error('No language found: ' + languageCode);
            }
        }
        return languages[languageCode];
    },
    applyScoringStrategy: function(languageCode, tokens, cursor, tokenScore)
{
        var language = this.getLanguage(languageCode);
        // Fallback to default strategy if none was specified
        // eslint-disable-next-line max-len
        var scoringStrategy = language.scoringStrategy ||
defaultScoringStrategy;
        return scoringStrategy.apply(tokens, cursor, tokenScore);
    }
};

var defaultScoringStrategy = {
    apply: function(tokens, cursor, tokenScore) {
        return tokenScore;
    }
};
```

37. sentiment.js

```
var tokenize = require('./tokenize');
var languageProcessor = require('./language-processor');
var Sentiment = function (options) {
    this.options = options;
};
Sentiment.prototype.registerLanguage = function (languageCode, language) {
    languageProcessor.addLanguage(languageCode, language);
```

```
};
Sentiment.prototype.analyze = function (phrase, opts, callback) {
    // Parse arguments
    if (typeof phrase === 'undefined') phrase = '';
    if (typeof opts === 'function') {
        callback = opts;
        opts = {};
    }
    opts = opts || {};
    var languageCode = opts.language || 'en';
    var labels = languageProcessor.getLabels(languageCode);
    if (typeof opts.extras === 'object') {
        labels = Object.assign(labels, opts.extras);
    }

    // Storage objects
    var tokens      = tokenize(phrase),
        score       = 0,
        words       = [],
        positive    = [],
        negative    = [];

    // Iterate over tokens
    var i = tokens.length;
    while (i--) {
        var obj = tokens[i];
        if (!labels.hasOwnProperty(obj)) continue;
        words.push(obj);

        // Apply scoring strategy
        var tokenScore = labels[obj];
        // eslint-disable-next-line max-len
        tokenScore = languageProcessor.applyScoringStrategy(languageCode,
tokens, i, tokenScore);
        if (tokenScore > 0) positive.push(obj);
        if (tokenScore < 0) negative.push(obj);
        score += tokenScore;
    }

    var result = {
        score:          score,
        comparative:    tokens.length > 0 ? score / tokens.length : 0,
        tokens:         tokens,
        words:          words,
        positive:       positive,
        negative:       negative
    };
    if (typeof callback === 'function') {
        process.nextTick(function () {
            callback(null, result);
        });
    } else {
        return result;
    }
};

module.exports = Sentiment;
```

38. tokenize.js

```
module.exports = function(input) {
    return input
        .toLowerCase()
        .replace(/\n/g, ' ')
        .replace(/[.,\/#!$%\^&\*;:{}=_`\"~()]/g, '')
        .split(' ');
};
```

39. default-scoring-strategy.js

```
var negators = require('./negators.json');

module.exports = {
    apply: function(tokens, cursor, tokenScore) {
        if (cursor > 0) {
            var prevtoken = tokens[cursor - 1];
            if (negators[prevtoken]) {
                tokenScore = -tokenScore;
            }
        }
        return tokenScore;
    }
};
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО	**ЗАТВЕРДЖУЮ**
Керівник проекту	**В.о. завідувача кафедри**
_____ *Гриша О.В.*	_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)	(підпис) (ініціали, прізвище)
“15” квітня 2019 р.	“16” квітня 2019 р.

Комплекс задач з підтримки користувачів з використанням сентиментального аналізу даних

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП ІС-5115.1181-с.ТЗ*

на 10 сторінках

Київ – 2019 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повна назва системи: *Комплекс задач з підтримки користувачів з використанням сентиментального аналізу даних*

1.2 Найменування організації-замовника та організацій-учасників робіт

Генеральним замовником проекту являється кафедра Автоматизованих систем обробки інформації та управління НТУУ "КПІ". Представником замовника є Гриша Олена Василівна.

Розробником системи є студент групи ІС-51 факультету інформатики та обчислювальної техніки НТУУ «КПІ ім. Ігоря Сікорського» Медведніков Даніїл Сергійович.

1.3 Перелік документів, на підставі яких створюється система

При розробці системи і створення проектно-експлуатаційної документації Виконавець повинен керуватися вимогами наступних нормативних документів:

- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплексність і позначення документів при створенні автоматизованих систем.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над створенням системи підтримки користувачів з використанням сентиментального аналізу даних – 15 квітня 2019 рік.

Плановий термін по закінченню роботи над створенням системи формування асортименту торгівельної організації – не пізніше 19 травня 2019 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СЕРВІСУ

2.1 Призначення розробки

Призначенням розробки є моніторинг, управління і аналіз процесу підтримки користувачів з використанням сентиментального аналізу даних.

2.2 Цілі створення застосування

Цілями розробки є збільшення спрощення та підвищення якостів процесу підтримки користувачів за рахунок:

- спрощення процесу комунікації за рахунок використання чат-ботів;
- виявлення незадовільнених клієнтів за допомогою сентиментального аналізу текстових даних;

Для досягнення поставлених цілей необхідно вирішити наступні задачі:

- розробити засіб для комунікації клієнтами;
- розробити засіб для створення опитувань;
- розробити засоби для інтеграцій з месенджерами Telegram, Viber, Facebook Messenger;
- розробити засіб для сентиментального аналізу даних;

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Працювати з застосуванням можуть адміністратор та оператор служби підтримки.

Об’єктом автоматизації є процес надання підтримки клієнтам.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Застосування має створювати умови для підтримки комунікації і повинне задовольняти потреби користувачів. Застосування має виконувати наступні функції:

1. Система повинна надати адміністратору та операторуу можливість входу за завчасно заданим паролем та логіном.
2. Система повинна надати адміністратору можливість керувати застосуванням та базою даних.
3. Система надає можливість адміністратору створити розсилку-опитування по базі клієнтів.
4. Система надає можливість адміністратору переглянути статистику попередній розсилок.
5. Система надає можливість проводити сентиментальний аналіз відповідей клієнтів.
6. Система надає можливість оператору надсилати повідомлення клієнтам.
7. Система надає можливість оператору отримувати повідомлення від клієнтів.

4.2 Вимоги до надійності

Програма повинна зберігати працездатність і забезпечувати відновлення своїх функцій при виникненні наступних позаштатних ситуацій:

- при помилках в роботі апаратних засобів (крім носіїв даних і програм).

Програмний продукт повинен поєднувати надійність та функціональність. У разі виникнення аварійних ситуацій необхідно сповіщати користувача та надавати інструкцію для подальших дій. Будь-які аварійні ситуації ма-

ють бути задокументовані у звіті, який при необхідності надсилається розробнику для визначення причини збою в роботі та усуненні помилок, які могли привести до нестабільної роботи програмного продукту.

4.3 Вимоги до складу і параметрів технічних засобів

Склад, структура і способи організації даних в системі повинні бути визначені на етапі технічного проектування.

Структура технічних засобів визначається виходячи із можливості їх забезпечити виконання встановлених операцій процесу технічного обслуговування.

Для правильної роботи розробленої системи до складу технічних засобів повинен входити комп’ютер, що має конфігурацію наведену нижче:

а) комп’ютер з такою конфігурацією:
   1) процесор з тактовою частотою не нижче 1 ГГц;
   2) об’єм оперативної пам’яті не менше 1 ГБ;
   3) інші складові можуть мати будь-які параметри, тому що вони не значним чином впливають на роботу застосування;
   4) підключення до мережі інтернет;

б) комп’ютерна периферія, до складу якої входить:
   1) монітор;
   2) мишка;
   3) клавіатура;

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Основні етапи виконання робіт з розробки системи ведення наукової роботи.

№ п/п	Назва етапу роботи	Термін виконання етапу	Результат виконання
1.	Підготовка технічного завдання на розробку програмного продукту	07.02.2019	
2.	Розробка сценарію роботи	12.02.2019	
3.	Технічне проектування – функціональність, модулі, задачі, цілі тощо	20.02.2019	
4.	Узгодження з керівником інтерфейсу користувача	02.03.2019	
5.	Розробка інформаційного забезпечення	17.03.2019	
6.	Розробка програмного забезпечення	29.03.2019	
7.	Налагодження програми	13.04.2019	
8.	Тестування програми	27.04.2019	
9.	Здача готового програмного продукту замовнику	12.05.2019	

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

6.1 Види випробувань

Для контролю правильності роботи програмного забезпечення буде проведено модульне тестування (Unit Test). В ході тестування буде проведено випробування основних елементів системи, які представлені бізнес-логікою та компонентами інтерфейсу користувача (UI-компонентами).

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО

Керівник проекту

_____ *Гриша О. В.*
(підпис) (ініціали, прізвище)

“16” травня 2019 р.

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)

“17” травня 2019 р.

Комплекс задач з підтримки користувачів з використанням сентиментального аналізу даних

ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ

Шифр *ДП ІС-5115.1181-с.ПМВ*

на 12 сторінках

Київ – 2019 року

ЗМІСТ

1 ОБ'ЄКТ ВИПРОБУВАННЯ

1.1 Найменування програми

Темою дипломного проекту є «Комплекс задач з підтримки користувачів з використанням сентиментального аналізу даних».

1.2 Область застосування

Областю застосування програми є системи надання підтримки користувачам та клієнтам.

1.3 Умовне позначення програми

Умовне позначення програми – веб-ресурс HelloMessage.

.

2 МЕТА ВИПРОБУВАНЬ

Мета проведення випробувань – перевірка відповідних характеристик розробленої програми (програмного вироби) функціональним і окремим іншим видам вимог, викладених в документі технічного завдання.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функціональних характеристик

Програмне застосування повинно:

- бути функціонально достатнім (повним);
- бути надійним (застосування повинно корректно завершити роботу без втрати даних);
- бути придатним до модернізації та масштабування;
- мати інтуїтивно зрозумілий для користувача інтерфейс;
- бути стійким до хибних дій користувача.

3.1.1 Вимоги до складу виконуваних функцій

Застосування має виконувати наступні функції:

- вдало авторизувати користувачів;
- відображати оператору список діалогів;
- вести комунікацію між оператором та клієнтом у вигляді онлайн чату;
- відображати інформацію про клієнтів;
- створювати розсилки клієнтам;
- проводити сентиментальний аналіз повідомлень клієнтів;

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмний продукти розробляється на основі Технічного Завдання.

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5 СКЛАД І ПОРЯДОК ВИПРОБУВАНЬ

Умови проведення випробувань: програма з готовими модулями та зв’язками між ними.

Умови початку та завершення окремих етапів тестування: тестування кожних елементів програми має відбуватися з урахуванням всіх можливих виключних ситуацій в залежності від функціональних можливостей програмного продукту.

Обмеження щодо умов проведення тестування: тестування має проводитися в рамках функціонального апарату програмного забезпечення.

Вимоги до технічного обслуговування системи: система має бути пристроєм на операційній системі Windows, мати браузер Google Chrome (або інші на його основі) та мати доступ до мережі Internet.

Міри, забезпечуючі безпеку та безаварійність проведення тестування: тестування системи не може визвати аварійних ситуацій.

Порядок взаємодій організацій, які беруть участь у тестуванні: тестування проводить один студент КПІ групи ІС-51 Медведніков Даніїл Сергійович

6 МЕТОДИ ВИПРОБУВАНЬ

У наступних таблицях наведено перелік випробувань основних функціональних можливостей.

Таблиця 6.1 - Вхід користувача в систему

Мета тесту	**Перевірка функції «Авторизація в системі»**
Початковий стан моделі	Відкрита сторінка аутентифікації
Вхідні дані:	Електронна пошта та пароль користувача
Схема проведення тесту:	Ввести електронну пошту та пароль користувача у відповідні текстові поля. Після цього натиснути кнопку “Увійти”
Очікуваний результат:	Перехід на головну сторінку системи
Стан моделі після проведення випробувань:	Відкрита головна сторінка системи

Таблиця 6.2 - Відображення списку діалогів

Мета тесту	**Перевірка функції «Список діалогів»**
Початковий стан моделі	Відкрита головна сторінка системи
Вхідні дані:	
Схема проведення тесту:	Відкрити вкладку меню “Діалоги” у головному меню програми та натиснути на кнопку “Усі”

Продовження таблиці 6.2

Очікуваний результат:	Перехід на сторінку діалогів
Стан моделі після проведення випробувань:	Відкритка сторінка “Діалоги”, відображено список діалогів

Таблиця 6.3 - Відображення історії повідомлень

Мета тесту	**Перевірка функції «Історія повідомлень»**
Початковий стан моделі	Відкрита сторінка діалогів
Вхідні дані:	
Схема проведення тесту:	Вибрати діалог зі списка діалогів
Очікуваний результат:	Відображено повідомлення, що відносяться до обраного діалогу
Стан моделі після проведення випробувань:	Відображено повідомлення, що відносяться до обраного діалогу

Таблиця 6.4 - Відправлення текстового повідомлення

Мета тесту	**Перевірка функції «Відправка повідомлення»**
Початковий стан моделі	Відкрито діалог з обраним користувачем
Вхідні дані:	Текст повідомлення

Продовження таблиці 6.4

Схема проведення тесту:	Ввести текст повідомлення до текстового поля вводу повідомлення та натиснути клавішу Enter на клавіатурі
Очікуваний результат:	Повідомлення відправлено
Стан моделі після проведення випробувань:	Нове повідомлення відображено в історії повідомлень

Таблиця 6.5 - Відправлення фотографії

Мета тесту	**Перевірка функції «Відправлення фотографії»**
Початковий стан моделі	Відкрито діалог з обраним користувачем
Вхідні дані:	Файл фотографії для відправки
Схема проведення тесту:	Натиснути на кнопку “Прикріпити” у правій частині поля вводу та обрати потрібну фотографію зі списку файлів
Очікуваний результат:	Фотографію відправлено
Стан моделі після проведення випробувань:	Повідомлення з фотографією відображено в історії повідомлень

Таблиця 6.6 - Відправлення документів

Мета тесту	**Перевірка функції «Відправлення документів»**
Початковий стан моделі	Відкрито діалог з обраним користувачем
Вхідні дані:	Файл документу для відправки
Схема проведення тесту:	Натиснути на кнопку “Прикріпити” у правій частині поля вводу та обрати потрібний документ зі списку файлів
Очікуваний результат:	Документ відправлено
Стан моделі після проведення випробувань:	Повідомлення з документов відображено в історії повідомлень

Таблиця 5.7 - Інформація про клієнта

Мета тесту	**Перевірка функції «Інформація про клієнта»**
Початковий стан моделі	Відкрито діалог з обраним користувачем
Вхідні дані:	
Схема проведення тесту:	Натиснути на кнопку “Інформація про клієнта” з випадаючого меню на екрані діалогу
Очікуваний результат:	Відображень спливаюче вікно з інформацією про клієнта
Стан моделі після проведення випробувань:	Відображено повідомлення, що відносяться до обраного діалогу

Таблиця 6.8 - Перехід на сторінку розсилки

Мета тесту	**Перевірка функції «Сторінка створення розсилки»**
Початковий стан моделі	Відкрито головну сторінку застосунку
Вхідні дані:	
Схема проведення тесту:	Натиснути на кнопку “Розсилки” у головному меню програми
Очікуваний результат:	Перехід на сторінку “Розсилки”
Стан моделі після проведення випробувань:	Відкрито сторінку “Розсилки”

Таблиця 6.9 - Створення розсилки

Мета тесту	**Перевірка функції «Створення розсилки»**
Початковий стан моделі	Відкрито сторінку “Розсилки”
Вхідні дані:	Текст повідомлення для розсилки
Схема проведення тесту:	Ввести текст повідомлення до текстового поля та натиснути на кнопку “Відправити”
Очікуваний результат:	Відображено повідомлення про успішне надсилання розсилки
Стан моделі після проведення випробувань:	Відображено повідомлення про успішне надсилання розсилки

Графічний матеріал
до дипломного проекту

на тему: «Комплекс задач з підтримки користувачів з використанням сентиментального аналізу даних»

Київ – 2019 року