МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ
ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

_____ І.А. Жуков
(підпис)

« ____ » _____ 2020 р.

ДИПЛОМНА РОБОТА

(**ПОЯСНЮВАЛЬНА ЗАПИСКА**)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

ЗА СПЕЦІАЛЬНІСТЮ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема: « Система керування чат-ботом на базі технології *React* »

Виконавець: студент, КС-231(М), Грушак Сергій Сергійович
(студент, група, прізвище, ім'я, по батькові)

Керівник: к.т.н., доцент, Гузій Микола Миколайович
(наукова ступінь, вчене звання, прізвище, ім'я, по батькові)

Нормоконтролер: _____ Малярчук В.О.
(підпис) (ПІБ)

Засвідчую, що у дипломній роботі немає
запозичень праць інших авторів без відповідних посилань
Студент _____ Грушак С.С.
(підпис) (ПІБ)

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних систем та мереж
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ І.А. Жуков
(підпис)

« ____ » _____ 2020 р.

ЗАВДАННЯ
на виконання дипломної роботи

Грушака Сергія Сергійовича
(прізвище, ім'я, по батькові)

1. Тема роботи: Система керування чат-ботом на базі технології *React*

затверджена наказом ректора від « 25 » вересня 2020 року № 1793/ст.
2.Термін виконання роботи: з 05.10.2020 р. по 30.12.2020 р.
3.Вихідні дані до роботи: вимоги до системи керування чат-ботом

4.Зміст пояснювальної записки (перелік питань, що підлягають розробці):
Вступ, порівняльний аналіз платформ чат-ботів, аналіз та вибір технологій для розробки системи керування чат-ботом, розробка серверної частини системи керування чат-ботом, розробка клієнтської частини системи керування чат-ботом, висновки, додатки

5.Перелік обов'язкового графічного матеріалу:
Презентація *PowerPoint*

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Отримати перелік питань та завдання для дослідження на виконання дипломної роботи	05.10.2020	
2	Скласти план дипломної роботи	06.10.2020	
3	Ознайомитись з літературними джерелами	07.10.2020	
4	Провести дослідження технологій чат-ботів	14.10.2020	
5	Написати перший розділ	19.10.2020	
6	Ознайомитись з літературними джерелами	26.10.2020	
7	Написати другий розділ	02.11.2020	
8	Ознайомитись з літературними джерелами	09.11.2020	
9	Спроектувати архітектуру системи	16.11.2020	
10	Написати третій розділ	19.11.2020	
11	Спроектувати *UI* системи	26.11.2020	
12	Написати четвертий розділ	30.11.2020	
13	Зробити висновки по роботі	07.11.2020	

7. Дата видачі завдання: «05» жовтня 2020 р.

Керівник дипломної роботи: _____ Гузій М.М.
(підпис) (П.І.Б)

Завдання прийняв до виконання: _____ Грушак С.С.
(підпис випускника) (П.І.Б)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Система керування чат-ботом на базі технології *React*”: 84 ст., 42 рисунків, 14 таблиць, 30 використаних джерел, 3 додатки.

ЧАТ-БОТ, МЕСЕНДЖЕР, *API* ПЛАТФОРМИ ЧАТ-БОТА, *TELEGRAM*, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, *NODE.JS*, *JAVASCRIPT*, *SQLITE*, *REACT.JS*, МІКРОСЕРВІСНА АРХІТЕКТУРА, ПРОЕКТУВАННЯ *UI*.

Мета дипломної роботи – провести дослідження сучасних платформ для чат-ботів, проаналізувати та обґрунтувати вибір технологій для розробки веб-додатків та на їх основі створити систему керування чат-ботом.

Об’єкт дослідження: технології чат-ботів.

Предмет дослідження: система керування чат-ботом.

Методи дослідження: системний аналіз, евристичний аналіз, методи програмної інженерії, компонентне проектування, клієнт-серверна архітектура, архітектура *MVC*, *UML*-проектування.

Отримані результати та наукова новизна: відображена значущість чат-ботів у інформаційному просторі, проведений розширений аналіз технологій для створення веб-додатків, запропоновані нові механізми обміну даними у веб-додатках, набули подальшого розвитку концепції розгортання додатків з мікросервісною архітектурою, створена система керування чат-ботом.

Рекомендації щодо використання результатів: реалізація запропонованих механізмів обміну даними при розробці веб-додатків, впровадження концепцій розгортання додатків з мікросервісною архітектурою, використання системи керування чат-ботом за призначенням.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ACID – *Atomicity, Consistency, Isolation, Durability*
AJAX – *Asynchronous Javascript and XML*
CORS – *Cross-Origin Resource Sharing* (перехресний обмін ресурсами)
DNS – *Domain Name System*
gRPC – *Google Remote Procedure Calls*
IDE – *Integrated Development Environment*
JSON – *JavaScript Object Notation*
MFD – *Mobile First Development*
MVC – *Model-View-Controller* (модель-вид-контролер)
RPI – *Remote Procedure Invocation* (віддалений виклик процедур)
RWD – *Responsive Web Design*
SSL – *Secure Sockets Layer*
TCP – *Transmission Control Protocol*
TLS – *Transport Layer Security*

ВСТУП

Чат-бот – це комп’ютерна програма, основним призначенням якої є обмін інформацією з користувачем (людиною) за попередньо визначеним алгоритмом та/або за допомогою елементів штучного інтелекту.

Стрімкий розвиток та поширення чат-ботів зумовлюється втратою популярності звичайних додатків при умові існування чат-ботів з аналогічною функціональністю, а також з масовим переходом на різноманітні служби повідомлень (месенджери). Таким чином, замість встановлення чергового додатку на свій пристрій у користувача з’являється аналог – сервіс всередині звичайного йому месенджеру.

На сьогодні існує ціла низка світових компаній, які так чи інакше вбудували функціонал чат-ботів у свої продукти. Для прикладу, *Facebook* (*Facebook Messenger*), *Microsoft* (*Skype*), *Viber Media, Inc* (*Viber*), *Telegram FZ-LLC* (*Telegram*). Надалі, у контексті цієї роботи, продукти вищезазначених компаній є *платформами для чат-ботів* або просто – *платформи*.

Функціональні можливості чат-ботів та стрімкий розвиток даної галузі зробило її привабливою з точки зору як звичайних користувачів, так і компаній. Одна з перших комерціалізацій чат-ботів відбулася шляхом їх вбудовування у веб-сайти для допомоги користувачам у знаходженні відповідей на їх питання та оповіщення про події. З подальшим розвитком чат-боти ставали розумнішими, розвиток галузі штучних нейронних мереж та механізмів обробки природньої мови цьому активно сприяли. Вміло запрограмовані чат-боти стали частково розуміти контекст розмови та почали надавати свої пропозиції у відповідь на неточні питання користувачів. З’явилося поняття “сервіс всередині чат-бота”, створилися нові платформи, чат-боти стали невід’ємною складовою багатьох державних та приватних бізнесів, наприклад *RailwayBot* Укрзалізниці, *NovaPoshtaBot* Нової Пошти тощо.

Розробка додатків для зв’язку з чат-ботом та його керуванням вимагає чіткого розуміння принципів роботи протоколів передачі даних, навичок та досвіду

програмування серверних додатків, вивчення *API* конкретної платформи, а також при необхідності: знань з програмування веб-додатків (*HTML*, *CSS*, *JS*), протоколів *HTTP* та *HTTPS*, інтерфейсів веб *APIs*, а також вузькоспеціалізованих знань в області *web-security*. До того ж, в рамках виконання дипломної роботи, необхідними є знання фреймворку *React* для розробки клієнтського веб-додатку.

Отже, обрана тема дипломної роботи є актуальною з точки зору сучасних тенденцій, а також потребує значних інженерних знань по проектуванню та створенню програмного забезпечення.

Проблематика та напрямки роботи чат-ботів:

1. Оптимізація повторюваних завдань або процесів.
2. Цілодобовий зворотній зв'язок з користувачем.
3. Продажі, реклама та комерція.
4. Інтеграція з різноманітними інформаційними системами.
5. Використання алгоритмів штучного інтелекту для комунікації з користувачем.

Чат-боти можуть використовуватись в будь-якій сфері, де вирішення певної проблеми описується чітким алгоритмом.

Наприклад, чат-боти використовуються в різних сферах бізнесу (банки, оператори зв’язку, комерція, служби технічної підтримки тощо) в тих ситуаціях, коли у клієнта компанії є типове питання. Клієнт має змогу звернутись до чат-бота та отримати оперативну відповідь на своє запитання з переліку заздалегідь підготовлених варіантів відповідей. Якщо в сценарії бота немає потрібної відповіді, то клієнт перемикається на чергового оператора.

Завдяки тому, що при використанні чат-ботів створюється ефект “індивідуалізації”, їх розповсюджено використовують для процесів навчання.

Результатами використання ботів для корпоративного навчання є: створення аналогу тренера (швидкі відповіді на запитання); планування навчання; процес навчання та доступу до навчальних матеріалів відбувається аналогічно до систем дистанційного навчання; виникнення почуття персоналізованого та ефективного навчання.

Ще одна активна сфера використання чат-ботів – комерція. За останні роки активно збільшується попит на розробку чат-ботів для вирішення завдань з підбору, опису та продажу товарів. При цьому для залучення та роботи з клієнтами активно використовуються такі можливості чат-ботів як: масові розсилки інформаційних листів, проведення анонімних та публічних опитувань, інтеграція з платіжними системами, інтеграція з інформаційними системами тощо.

Також до основної проблематики дипломної роботи можна віднести необхідність проектування архітектури системи керування чат-ботом, вибір технологій та підходів для її розробки, дослідженні обмежень та можливостей обраної платформи чат-ботів, створенні серверної та клієнтської частини з відповідною їх взаємодією у режимі реального часу, оптимізації швидкодії системи.

Метою дипломної роботи є проведення дослідження сучасних платформ для чат-ботів, аналіз та обґрунтування вибору технологій для розробки веб-додатків та на основі отриманих даних створення системи для керування чат-ботом з її подальшою оптимізацією та тестуванням.

Завдання дипломної роботи полягає у дослідженні технологій роботи з чат-ботами та створенні єдиної інформаційної системи, за допомогою якої користувач зможе отримувати, передавати, зберігати, проглядати та керувати даними, що стосуються безпосередньої роботи чат-бота на базі обраної платформи.

Об’єктом дослідження є технології чат-ботів.

Предметом дослідження є система керування чат-ботом: компонентний склад, механізми обміну даними, взаємозв’язки між компонентами, захист даних, відображення та керування даними системи.

У ході виконання дипломної роботи були використані наступні **методи**: системний аналіз, евристичний аналіз, методи програмної інженерії, мікросервісна архітектура, клієнт-серверна архітектура, архітектура *MVC*, *UML*-проектування.

Використання вищезазначених методів обумовлюється необхідністю у проведенні дослідження по технологіям чат-ботів, аналізу технологій передачі даних у інформаційних системах, проектуванню та створенню системи керування чат-ботом.

Наукова новизна отриманих результатів:

1. Узагальнене та розширене поняття чат-бот: проведене поєднання функціональних можливостей чат-ботів у єдиній системі на прикладі месенджеру *Telegram*.

2. Проведений аналіз популярних месенджерів з платформами для чат-ботів: функціональних можливостей, *API*, обмежень.

3. Запропоноване використання механізму для синхронізації даних між клієнтом (браузером) та веб-сервером у сучасних веб-додатках на основі технологій: *WebSockets* та *WebWorkers*.

4. На основі запропонованого механізму розширено функціонал веб-додатків, який передбачає ітеративне зменшення використовуваного трафіку, збільшення швидкості роботи, можливість використання веб-додатків у офлайн режимі.

5. Створена система керування чат-ботом, яка розширює функціональні можливості платформи чат-ботів месенджеру *Telegram*, об’єднує передові практики по розробці ПЗ та створює унікальний веб-інтерфейс користувача на основі технології *React*.

6. Уточнені переваги та недоліки механізмів розгортання розподілених мікросервісів: порівняні підходи з використанням єдиного домену та субдоменів.

Виходячи з мети дипломної роботи необхідно вирішити наступні **задачі**:

1. Розкрити проблематику створення та використання чат-ботів у контексті потреб сучасної Інтернет-спільноти.

2. Провести аналіз сучасних сервісів для створення чат-ботів та обрати один для подальшого опрацювання.

3. Сформувати основні функціональні вимоги до створюваної системи.

4. Провести аналіз сучасних технологій та підходів до розробки веб-серверів та веб-додатків. Детально розглянути технологію *React*.

5. Описати процеси передачі інформації в створеній системі.

6. Спроектувати базу даних та обрати архітектурні підходи до розроблюваного ПЗ.

7. Розробити серверну частину системи.

8. Розробити клієнтську частину системи на базі технології *React*.

9. Надати рекомендації щодо процесу розгортання системи.

Практичне значення отриманих результатів полягає у проведеному дослідженні технологій використання чат-ботів та створеній системі для керування чат-ботом на базі платформи *Telegram*.

РОЗДІЛ 1
ПОРІВНЯЛЬНИЙ АНАЛІЗ ПЛАТФОРМ ЧАТ-БОТІВ

1.1. Класифікація чат-ботів

Підходи до створення і застосування чат-ботів в різних сферах діяльності людини присвячено багато робіт вітчизняних і зарубіжних вчених [1–10].

В роботі авторів [7] зазначається, що месенджери використовуються для вирішення різноманітних завдань, які виходять за рамки простого обміну текстовими повідомленнями: для клієнтської взаємодії з компаніями, пошуку потрібних товарів, споживання контенту та іншого. У роботах [3; 4; 6] подано загальний огляд найбільш популярних інструментальних засобів для розробки чат-ботів.

В роботах [1–2; 5; 8–10] описуються проблеми створення чат-ботів та їх використання.

Чат-боти зазвичай створюються для персонального або ділового користування.

Персональні чат-боти використовуються як особисті помічники користувача і виконують завдання відправки текстів, управління календарем, прийому викликів, пошуку і відтворення аудіо- та відео-файлів тощо.

Ділові чат-боти розробляються для бізнесу і призначені для залучення клієнтів до діалогу, супроводження бізнес-процесів пов’язаних з маркетингом, продажами та іншими допоміжними завданнями (табл. 1.1).

Напрямок розробки та використання чат-ботів постійно розвивається і потребує все більш детального аналізу та обґрунтування використаних підходів, фреймворків та платформ.

Проведемо узагальнений аналіз чат-ботів. Першочергово необхідно з’ясувати, які взагалі існують класифікації та види чат-ботів. Класифікацію здійснимо

відповідно до можливих ознак: користувач, інтерфейс, призначення, доступ принцип роботи. Узагальнена класифікація чат-ботів представлена у табл. 1.1.

Таблиця 1.1

Узагальнена класифікація чат-ботів

Ознака	**Вид**
Користувач	Персональний Бізнес
Інтерфейс користувача	Текстовий Кнопковий Голосовий
Призначення	Комунікаційний Функціональний
Доступ	Доданий до групи За підпискою Вбудований в діалог
Принцип роботи	Шаблонний Який навчається

Чат-бот може взаємодіяти з користувачем за допомогою кнопок, тексту або голосових команд. Кнопковий чат-бот має інтерфейс у вигляді кнопок і команд. Діалог організований таким чином, що користувачеві пропонуються на вибір категорії, питання або пропозиції, які можуть його зацікавити та відбувається шляхом натискання відповідних кнопок. Кнопки у даному випадку є своєрідними командами. Команди найчастіше визначаються при його створенні, або ж, у більш рідких випадках, можуть змінюватися по ходу роботи.

Спілкування з текстовим чат-ботом наближене до реального людського, але має деякі функціональні особливості. Можливості ботів такого типу теоретично ширше кнопкових, проте вони обмежені знаннями та навченістю самого бота по реагуванню на той чи інший текст.

Найбільш складним керуванням чат-ботом вважається керування голосом. Підтримка такого типу керування залежить від конкретної платформи. Користувач надсилає аудіо-повідомлення боту, мова з якого розпізнається та переводиться в текст, який в свою чергу аналізується, а вже після чого формується відповідь.

Інтерфейси чат-бота можуть комбінуватися та доповнювати одне одного.

Існують три основні форми доступу: групова, за підпискою та вбудована. Групова форма доступу передбачає обслуговування одразу декількох користувачів. Чат-боти за підпискою призначені виключно для персоналізованого використання. Вбудовані (*inline*) боти можуть бути викликані у будь-якому діалозі за допомогою спеціальної команди. Такі чат-боти виконують певний запит, результат якого одразу пересилається співрозмовникові.

За принципом роботи можна чат-боти розділяються на два види: які працюють за заздалегідь заданим шаблоном та які навчаються в процесі спілкування. Шаблонні чат-боти працюють виключно у рамках заздалегідь визначеного автомату з жорсткою логікою – дерева рішень. Діалоги в них зазвичай лінійні і структуровані. Чат-боти, які навчаються, розробляються на основі рішень штучного інтелекту. Принцип їх роботи заснований на постійному аналізі діалогу для подальшого удосконалення своїх комунікативних навичок.

Розглянемо просту модель взаємодії чат-ботів з платформою месенджера (рис. 1.1). Зростання популярності чат-ботів привело до появи безлічі інструментальних засобів для їх розроблення [2; 4; 6]. Логіка роботи чат-боту може бути створена з нуля на різних мовах програмування. В основному для цих цілей використовуються мови програмування, що найкраще себе зарекомендували у сфері розробки серверного програмного забезпечення: *Node.JS*, *C++*, *Python*, *Ruby*.

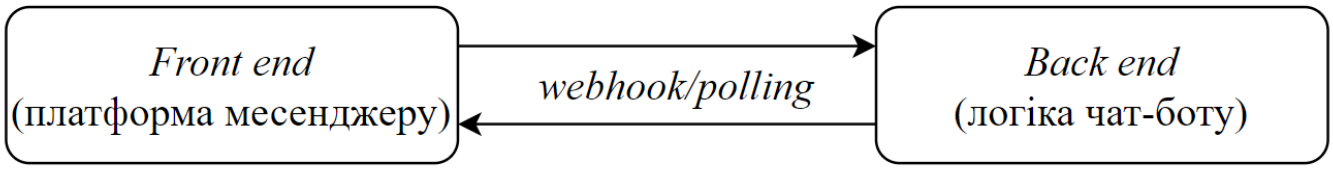

Рис.1.1. Модель взаємодії чат-бота з платформою месенджеру

Back end – серверний додаток, що призначений для обробки подій (дій та команд користувачів), що отримуються від *front end* сторони.

Front end – клієнтська платформа чат-боту, якою може бути будь-який популярний месенджер, наприклад, *Facebook Messenger*, *Telegram*, *Skype*, *Viber* тощо.

Webhook – механізм зв’язку між *back end* та *front end* сторонами, при якому *front end* при настанні будь-якої події надсилає *HTTP POST* запит на *back end* (рис. 1.2). Інакше кажучи, платформа месенджеру стає відповідальною за своєчасне сповіщення логіки чат-боту (серверного додатку) про настання подій. Форматом даних найчастіше є *JSON*. Варто зазначити, що при такому механізмі зв’язку потрібна додаткова автентифікація. Автентифікація може бути виконана одним або багатьма з наступних методів:

1. Зазвичай платформи чат-ботів надають перелік *IP*-адрес з яких надходять запити. Необхідно додати ці *IP*-адреси до білого списку мережевого фільтру, а запити з усіх інших *IP*-адрес – заборонити.

2. Зазвичай платформи чат-ботів присвоюють унікальний секретний токен кожному боту. Цей токен передається в окремій структурі при кожному запиті.

3. Встановлення з’єднання може відбуватися за допомогою протоколу *TLS*, що забезпечує процес передачі даних по шифрованому каналу.

4. Взаємна автентифікація *front end* та *back end* сторін відбувається за допомогою *SSL*-сертифікатів.

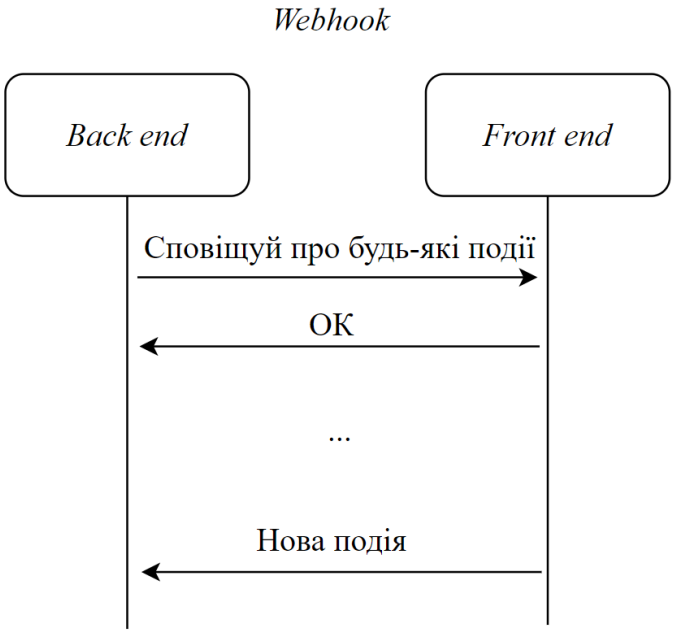

Рис.1.2. Візуалізація механізму *Webhook*

Polling – механізм зв’язку між *back end* та *front end* сторонами, при якому *back end* сторона надсилає *HTTP POST* запит на *front end* та очікує на відповідь (рис. 1.3).

Встановлене з’єднання при цьому залишається допоки не відбудеться яка-небудь подія. При настанні події, *front end* надсилає відповідь на запит. Процедура повторюється. Форматом даних є *JSON* або *URL query string*. Автентифікація при такому механізму зв’язку найчастіше відбувається за допомогою унікального токену чат-бота, що передається при кожному запиті у окремій структурі.

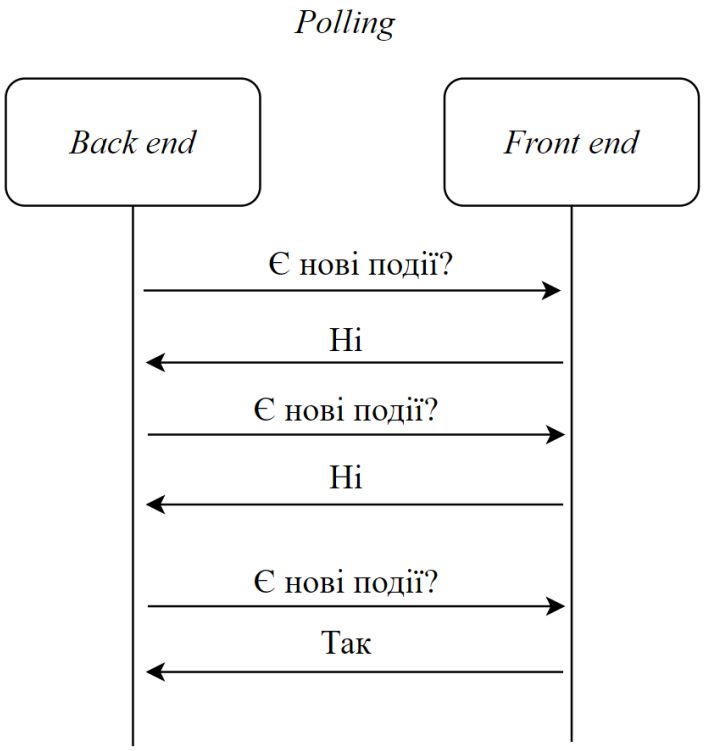

Рис.1.3. Візуалізація механізму *Polling*

1.2. Порівняльний аналіз месенджерів з платформою чат-ботів

Месенджер – це система, яка призначена для обміну миттєвими повідомленнями (текст та медіа), може включати в себе додаткові сервіси (голосові та відео виклики, чат-боти, магазин покупок, ігри), та одночасно використовується великою кількістю користувачів.

У контексті дипломної роботи необхідно проаналізувати та обрати один із сучасних месенджерів з підтримкою платформи чат-ботів. Для аналізу оберемо наступні месенджери: *Facebook*, *Telegram*, *Skype* та *Viber*.

Параметри для аналізу наступні: форма використання, доступи, інтерфейси, механізми зв’язку, протоколи передачі даних, формати передачі даних, основні функціональні можливості (листування, опитування, передача файлів, магазин покупок, ігри).

В табл. 1.2 представлена порівняльна інформація популярних месенджерів з підтримкою платформи чат-ботів.

Таблиця 1.2

Порівняльна характеристика платформ для чат-ботів популярних месенджерів

	Facebook	***Telegram***	***Skype***	***Viber***
Форма використання	Умовно безоплатна	Безоплатна	Умовно безоплатна	Умовно безоплатна
Доступи	Доданий до групи, за підпискою, вбудований в діалог	Доданий до групи, за підпискою, вбудований в діалог	Доданий до групи, за підпискою	За підпискою, вбудований в діалог
Інтерфейси користувача	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий
Механізми зв’язку	*webhook*	*webhook*, *polling*	*webhook*	*webhook*
Протоколи передачі даних	*https*			
Формати передачі даних	*JSON*, *multipart/form-data*	*JSON*, *URL query*, *multipart/form-data*	*JSON*, *multipart/form-data*	*JSON*, *multipart/form-data*
Функціональні можливості	Листування, опитування, передача файлів, магазин покупок, ігри	Листування, опитування, передача файлів, магазин покупок, ігри	Листування, передача файлів, магазин покупок, ігри	Листування, передача файлів, магазин покупок, ігри

Відповідно до проведеного аналізу робимо висновок, що для досягнення цілей дипломного проекту доцільно використати месенджер *Telegram*, оскільки, в порівнянні з іншими месенджерами, він має наступні переваги:

1. Має повністю безоплатну форму використання.

2. Підтримує механізм зв’язку *Polling*, який значно спрощує розробку та налагодження системи на початкових етапах.

3. Підтримує повний список функціональних можливостей, що дозволить реалізувати усі вимоги системи, а за подальшої потреби удосконалювати та розширювати їх.

1.3. Чат-бот у контексті *Telegram*

Чат-бот у контексті *Telegram* – це сторонній додаток, який працює в екосистемі *Telegram*. Користувачі взаємодіють з ботами надсилаючи їм повідомлення, команди та *inline*-запити. Для керування *Telegram*-ботом існує спеціальне *API*, яке вимагає використання *HTTPS*-протоколу.

За допомогою *Telegram*-боту можливо:

1. Отримання кастомізованих сповіщень та новин.

2. Інтеграція зі сторонніми сервісами: *Gmail*, *Wiki*, *YouTube*, *Github* тощо.

3. Отримання платежів від користувачів.

4. Створення корисних утиліт: перекладачі з різних мов, для форматування тексту, для нагадування про події тощо.

5. Розробка ігор, що базуються на технології *HTML*5.

Telegram-бот з точки зору месенджеру – це спеціальний аккаунт, з яким користувачі можуть взаємодіяти двома способами:

1. Відкрити чат з ботом, або ж додати його до групи, після чого надсилати текст або команди через поле вводу повідомлень.

2. Відкрити будь-який чат та вписати “@ім’я бота” в поле вводу повідомлень. Таким чином працюють з *inline*-ботами.

Варто зазначити, що:

1. Ініціювати спілкування з ботом може тільки користувач, але не навпаки. Це попереджає можливість спаму від недоброчесних ботів.

2. Боти мають обмежене хмарне сховище для повідомлень. Потрібно заздалегідь створити локальне сховище для зберігання застарілих повідомлень, якщо у цьому є необхідність.

3. Ім’я ботів завжди мають закінчення “*bot*”.

Розглянемо *Telegram Bot API*. Кожен бот потребує авторизації з боку серверів *Telegram*. При створенні бота генерується спеціальний унікальний токен (далі – <*token*>) авторизації (приклад – “*123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11*”), який передається частиною *URL* при кожному запиті. Усі запити до *Telegram Bot API* повинні відбуватися з використанням *HTTPS* протоколу на наступну адресу: “*https://api.telegram.org/bot<token>/METHOD_NAME*”. Підтримуються *GET* та *POST HTTP* запити. Існують наступні методи передачі параметрів у запиті:

1. *URL query string*.

2. *Application/x-www-form-urlencoded*.

3. *Application/json*.

4. *Multipart/form-data*.

URL query string – частина *URL*, яка містить пари “ключ=значення”, де ключ – певний параметр. Структура *query string*: *URL*?*param*1=1&*param*2=2&*param*3=3, де *URL* – унікальний ідентифікатор ресурсу, ? – символ, що відділяє *URL* та безпосередньо *query string*, & –символ, що розділяє декілька параметрів.

Application/x-www-form-urlencoded – тип тіла *HTTP* запиту, при якому дані запиту передаються у форматі “ключ=значення”. Тип зазначається у заголовка *Content-Type* запиту.

Зазначені вище методи можуть використовувати спеціальне кодування символів – *URL encoding* або ж інакше *percent encoding*. Це кодування використовуються для того, щоб вирішити проблему використання заборонених символів (приклад для *URL* – пробіл). Механізм кодування – заміна 8-ми бітових заборонених символів іншими, дозволеними символами. Кодування символів представлене на рис. 1.4.

':'	'/'	'?'	'#'	'['	']'	'@'	'!'	'$'	'&'	"'"	'('	')'	'*'	'+'	','	';'	'='	'%'	' '
%3A	%2F	%3F	%23	%5B	%5D	%40	%21	%24	%26	%27	%28	%29	%2A	%2B	%2C	%3B	%3D	%25	%20 or +

Рис.1.4. Кодування символів по механізму *URL encoding*

Символ пробілу кодується як “%20” при використанні в *URL*, та як “+” при використанні *Application/x-www-form-urlencoded*.

Application/json – тип тіла *HTTP* запиту, при якому дані запиту передаються у форматі *JSON*. Тип зазначається у заголовку *Content-Type* запиту.

JSON – текстовий формат обміну даними між комп’ютерами, призначений для опису об’єктів та структур даних. *JSON* нативно підтримується мовою *JavaScript* для серіалізації та десеріалізації об’єктів. Підтримується два варіанти структур:

1. Пари “ключ-значення”.

2. Впорядкований масив значень (масив, вектор, послідовність тощо).

Multipart/form-data – тип тіла *HTTP* запиту, який призначений для передачі файлів у бінарному вигляді. Тип зазначається у заголовку *Content-Type* запиту. Детально передача файлів з використанням протоколу *HTTP* описана у документі стандарту *RFC*7578 [11].

При запиті до *Telegram Bot API* у відповідь *Telegram* надсилає об’єкт *JSON*, який завжди містить булеве поле “*ok*” та строку “*description*”. Якщо результат запиту успішний, то поле “*ok*” рівне “*true*”, інакше – “*false*”. При неуспішності запиту поле “*description*” містить опис помилки, а також присилається додаткове поле “*error_code*” з кодом помилки.

Варто зазначити, що усі методи *Telegram Bot API* не чутливі до регістру, а кодування даних має відбуватись у форматі *UTF*-8.

Розглянемо функціональні можливості та обмеження *Telegram*-ботів. При розгляді зазначатимемо короткий опис функціональної можливості, роботу з нею з точки зору *Telegram Bot API* (використовувані методи *API*), обмеження за їх наявності.

Першочергово потрібно зазначити, що робота з *Telegram Bot API* поділена на дві частини: отримання оновлень та виклик методів *API*. У першому випадку *Telergam* надсилає повідомлення про конкретні дії користувача з ботом, у другому випадку – бот надсилає дані користувачу.

Отримання оновлення відбувається з отримання об’єкту *Update*. Під оновленнями розуміються події, що користувачі генерують під час взаємодії з чат-ботом. Основні поля об’єкту *Update* показані в табл. 1.3.

Таблиця 1.3

Основні поля об’єкту *Update*

Поле	**Тип**	**Опис**
update_id	Цілочисельний	Унікальний ідентифікатор оновлення.
message	*Message*	Необов’язкове. Нове повідомлення – текст, фото, відео або голосове повідомлення тощо.
edited_message	*Message*	Необов’язкове. Відредаговане повідомлення.
callback_query	*CallbackQuery*	Необов’язкове. Зворотній запит (результат натискання на кнопку клавіатури).

Листування. Основною функціональною можливістю будь-якого месенджеру є можливість листування між його користувачами. Листування відбувається за допомогою текстових, фото, відео або голосових повідомлень тощо. В *API* для цілей листування існує основний об’єкт – *Message*, основні поля якого показані в табл. 1.4.

Таблиця 1.4

Основні поля об’єкту *Message*

Поле	**Тип**	**Опис**
1	2	3
message_id	Цілочисельний	Унікальний ідентифікатор поточного чату.
date	Цілочисельний	Час в форматі *Unix*.
chat	*Chat*	Чат, якому належить цей *Message*.
edit_date	Цілочисельний	Необов’язкове. Останній час редагування в форматі *Unix*.
entities	Масив *MessageEntity*	Масив нікнеймів, *URL* та команд бота, що є у тексті повідомлення.
text	Рядок	Необов’язкове. Текст

		повідомлення в форматі *UTF*-8, 0-4096 символів.
caption	Рядок	Необов’язкове. Текст прикріплений до файлового повідомлення в форматі *UTF*-8, 0-1024 символів.

Продовження табл. 1.4

1	2	3
photo	Масив *PhotoSize*	Необов’язкове. Фото, масив розмірів фото.
document	*Document*	Необов’язкове. Звичайний файл.
reply_markup	*InlineKeyboardMarkup*	Необов’язкове. *Inline*-клавіатура.
Chat		
id	Цілочисельний	Унікальний ідентифікатор чату.
type	Рядок	Тип чату: приватний, група, супер група, канал.
username	Рядок	Необов’язкове. Нікнейм користувача.
first_name	Рядок	Необов’язкове. Ім’я користувача.
last_name	Рядок	Необов’язкове. Прізвище користувача.

Клавіатури. Додатково до будь-якого повідомлення *Telegram* дає можливість прикріпити клавіатуру з довільними кнопками. Це значно розширює користувацький досвід та надає можливість створювати у боті меню, відкривати посилання на сторонні веб-ресурси, проводити опитування тощо. Хорошою практикою вважається оновлення повідомлень при використанні клавіатур замість надсилання нових.

Бот має змогу прикріпити клавіатуру до об’єкту *Message* зазначивши поле *reply_markup*. Основні поля *InlineKeyboardMarkup* показані в табл. 1.5.

Таблиця 1.5

Основні поля об’єкту *InlineKeyboardMarkup*

Поле	**Тип**	**Опис**

1	2	3
inline_keyboard	Масив з масиву *InlineKeyboardButton*	Масив рядів кнопок.
InlineKeyboardButton		
text	Рядок	Текст, що відображається на кнопці.
url	Рядок	Необов’язкове. Посилання.

Продовження табл. 1.5

1	2	3
callback_data	Рядок	Дані, що будуть надіслані разом з зворотнім запитом (*CallbackQuery*).
CallbackQuery		
id	Рядок	Унікальний ідентифікатор запиту.
from	*User*	Чат, з якого прийшов цей запит.
message	*Message*	Оригінальний *Message*, що пов'язаний з цим запитом.
data	Рядок	Дані, що були асоційовані з кнопкою.

Клавіатура надсилається ботом користувачеві, при цьому в кожному об’єкті *InlineKeyboardButton* зазначається поле *callback_data*. При натисканні на кнопку клавіатури користувачем, *Telegram* надсилає об’єкт *CallbackQuery* на *back end* бота. Виходячи з вмісту поля *data* бот може розпізнати, яку дію запросив користувач. Відповідний набір команд (або ж свого роду *API* команд бота) формує розробник. Підтримувані типи клавіатур: звичайна (рис. 1.5) та *inline*-клавіатура (рис. 1.6).

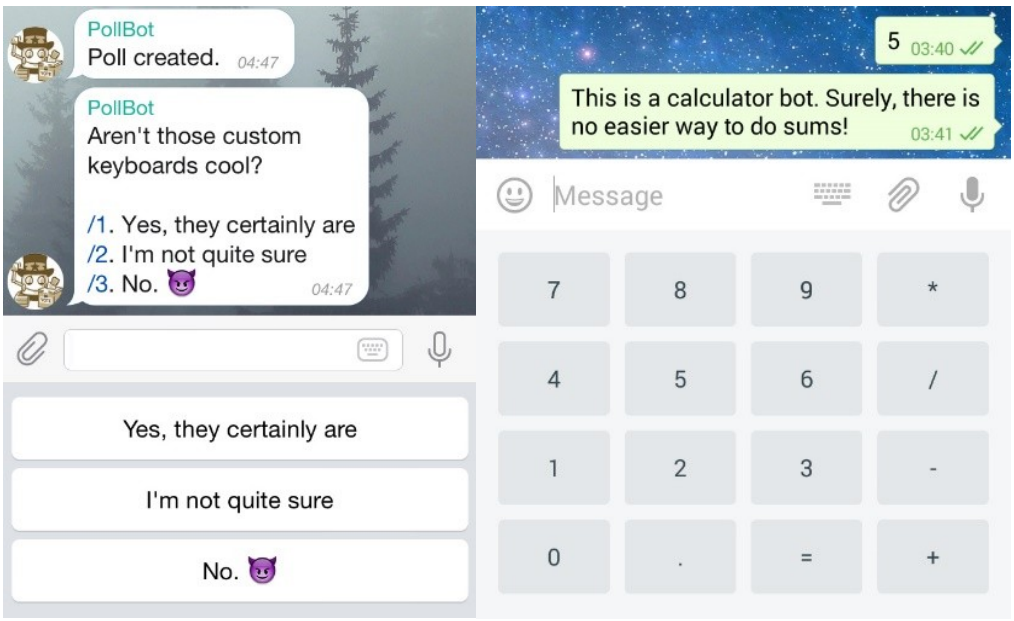

Рис.1.5. Звичайна клавіатура *Telegram*-бота

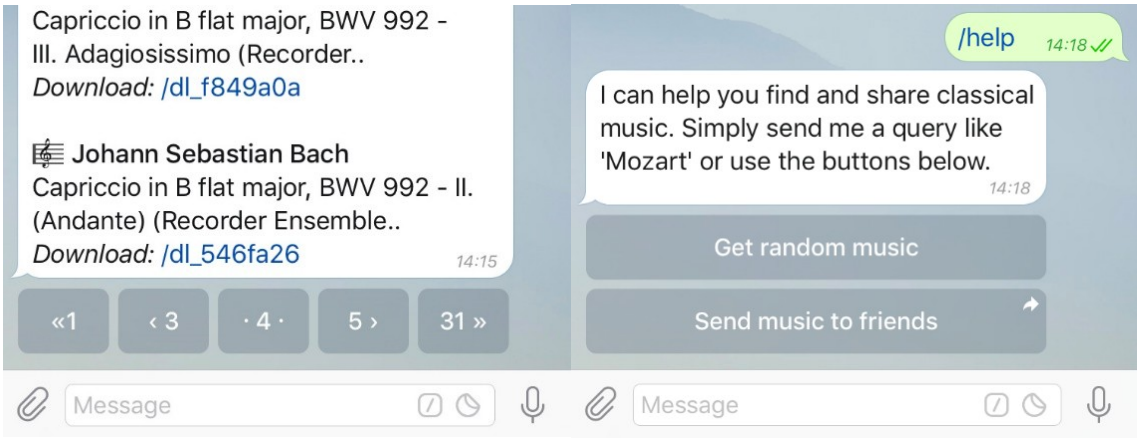

Рис.1.6. *Inline*-клавіатура *Telegram*-бота

Команди. Команди – це більш гнучкий спосіб комунікації з *Telegram*-ботом. Синтаксис команд наступний: “/*command*”. Команди завжди мають починатися з символу “/” та мати не більш ніж 32 символи після. Команди присутні в полі об’єкта *Message* – *MessageEntity*. Основні поля об’єкту *MessageEntity* представлені в табл. 1.6.

Таблиця 1.6

Основні поля об’єкту *MessageEntity*

Поле	**Тип**	**Опис**
type	Рядок	Тип: “*mention*”, “*hashtag*”, “*cashtag*”, “*bot_command*”, “*url*”, “*email*”, “*phone_number*” тощо.

offset	Цілочисельний	Здвиг до початку *MessageEntity*.
length	Цілочисельний	Довжина *MessageEntity*.

При введені символу “/” користувачу буде запропонований перелік доступних команд боту. Будь-яка надіслана команда підсвічується на кшталт посилання та може бути повторно надіслана за допомогою натискання на неї. Стандартні команди, що має підтримувати будь-який бот: “/*start*”, “/*help*”, “/*settings*”. При цьому команда “/*start*” є обов’язковою, оскільки вона використовується при ініціюванні діалогу користувачем з ботом. Команди та відповідні дії на них (*API*) формує розробник. Приклади використання команд представлені на рис. 1.7.

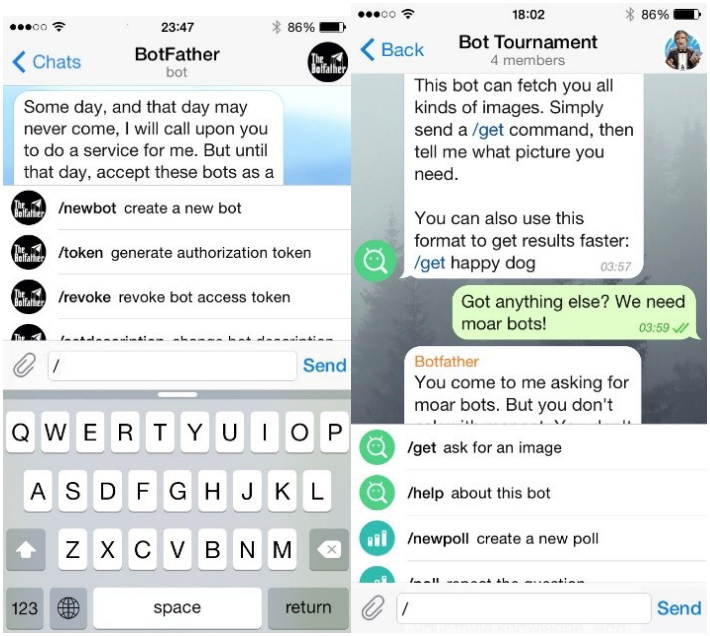

Рис.1.7. Приклади використання команд в *Telegram*-боті

Форматування повідомлень. *Telegram*-бот *API* надає можливість форматувати текст повідомлень. Тип форматування вказується в полі *parse_mode* при використанні методів надсилання повідомлень. Основні методи надсилання повідомлень представлені в табл. 1.7.

Таблиця 1.7

Основні методи надсилання повідомлень за допомогою *Telegram Bot API*

sendMessage		
chat_id	Цілочисельний	Унікальний ідентифікатор чату.
text	Рядок	Текст повідомлення. 0-4096 символів.
parse_mode	Рядок	Необов’язкове. Тип форматування.
reply_markup	Масив *InlineKeyboardMarkup*	Необов’язкове. *Inline*-клавіатура.
sendPhoto, sendDocument		
1	2	3
chat_id	Цілочисельний	Унікальний ідентифікатор чату.

Продовження табл. 1.7

1	2	3
photo, document	*InputFile*	Фото або файл.
caption	Рядок	Необов’язкове. Підпис фото. 0-1024 символів.
parse_mode	Рядок	Необов’язкове. Тип форматування.
reply_markup	Масив *InlineKeyboardMarkup*	Необов’язкове. *Inline*-клавіатура.

У табл. 1.7 зазначений об’єкт *InputFile*. Існує три способи надіслати файл:

1. Якщо файл вже присутній на серверах *Telegram* достатньо зазначити його унікальний ідентифікатор – *file_id*.

2. Якщо файл присутній у глобальній мережі Інтернет достатньо зазначити *HTTP URL* для його завантаження. Таким чином можливо завантаження фото розміром не більше 5 МБ, а інших файлів – до 20 МБ.

3. Якщо файл необхідно завантажити з локального диску, тоді використовується метод *multipart/form-data*. Таким чином можливо завантаження фото розміром не більше 10 МБ, а інших файлів – до 50 МБ.

Клієнти *Telegram* підтримують типи форматування (*parse_mode*) *HTML* (рис. 1.8) та *Markdown* (рис. 1.9). Підтримуються тільки зазначені теги.

```
<b>bold</b>, <strong>bold</strong>
<i>italic</i>, <em>italic</em>
<u>underline</u>, <ins>underline</ins>
<s>strikethrough</s>, <strike>strikethrough</strike>, <del>strikethrough</del>
<b>bold <i>italic bold <s>italic bold strikethrough</s> <u>underline italic bold</u></i> bold</b>
<a href="http://www.example.com/">inline URL</a>
<a href="tg://user?id=123456789">inline mention of a user</a>
<code>inline fixed-width code</code>
<pre>pre-formatted fixed-width code block</pre>
```

Рис.1.8. Підтримувані *HTML*-теги в *Telegram*

```
*bold \*text*
_italic \*text_
__underline__
~strikethrough~
*bold _italic bold ~italic bold strikethrough~ __underline italic bold___ bold*
[inline URL](http://www.example.com/)
[inline mention of a user](tg://user?id=123456789)
`inline fixed-width code`
```
pre-formatted fixed-width code block
```
```python
pre-formatted fixed-width code block written in the Python programming language
```
```

Рис.1.9. Підтримуваний *Markdown*-синтаксис в *Telegram*

Описаний функціонал та *Telegram Bot API* буде використано у подальшому для розробки системи керування чат-ботом *Telegram*. Детальніше про *Telegram Bot API* див. посилання [12].

Висновки за розділом

У розділі розглянуто поняття “чат-бот” та їх основна класифікація, детально описані механізми обміну даними між платформами чат-ботів та клієнтами, приведені обґрунтування використання описаних механізмів, здійснено

порівняльний аналіз сучасних месенджерів з підтримкою платформ для чат-ботів, розглянутий месенджер *Telegram* та його *Telegram Bot API*.

У результаті встановлено, що існують чат-боти зі штучним інтелектом та програмовані: сфери їх використання та можливості. Окремої уваги заслуговують основні ознаки чат-ботів, оскільки від них залежать безпосередні можливості: типи підтримуваного інтерфейсу, доступу, призначення. Також були досліджені механізми встановлення з'єднання з *API* чат-ботів: *polling* та *webhook*. Проведене їх порівняння, уточнені основні переваги та недоліки, надані рекомендації щодо використання.

Важливою частиною роботи стало дослідження різноманітних месенджерів та їх платформ для чат-ботів. Встановлено, що усі найпопулярніші месенджери (*Facebook*, *Telegram*, *Skype*, *Viber*) підтримують платформу чат-ботів. Уперше здійснено порівняльне дослідження їх технічних можливостей, переваг та недоліків. Проведення дослідження було здійснено за допомогою аналізу документацій кожного месенджеру та особистого тестування функціональних можливостей.

Виходячи з результатів дослідження, для виконання завдання дипломної роботи по розробці системи керування чат-ботом, був здійснений та обґрунтований вибір месенджеру *Telegram*. Вибір месенджеру зумовлений наявністю документованого *API*, широким набором функціональних можливостей, підтримкою механізму *Polling*, безоплатністю та невеликою кількістю технічних обмежень.

У подальшому, з метою імплементації системи керування чат-ботом, був здійснений опис *Telegram Bot API*: методи передачі та кодування даних, автентифікація на серверах *Telegram*, методи *API* для отримання даних та передачі команд. Окрема увага присвячена механізмам обміну файлами. Усі розглянуті теми безпосередньо стосуються та використовуються у подальшій розробці системи керування чат-ботом.

РОЗДІЛ 2
АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ КЕРУВАННЯ ЧАТ-БОТОМ

2.1. Формування функціональних вимог

Інформаційна система – організаційно-технічна система, в якій реалізується технологія обробки інформації з використанням технічних і програмних засобів. (Згідно закону України “Про захист інформації в інформаційно-телекомунікаційних системах”).

В будь-якій інформаційній системі вирішуються наступні задачі:

1. Передача даних.
2. Аналіз та обробка даних, прийняття рішень.
3. Зберігання даних.

Відповідно до поставлених задач будь-яка інформаційна система повинна мати конкретні реалізації методів та способів для їх вирішення. Іншими словами, основа будь-якої інформаційної системи складається з:

1. Засобів передачі даних та повідомлень між складовими компонентами інформаційної системи.
2. Засобів аналізу, обробки і представлення інформації.
3. Засобів фіксації та збору інформації.
4. Засобів збереження інформації.

Додатково до вищеописаного, з точки зору проектування програмного забезпечення (ПЗ) необхідно:

1. Сформувати специфікацію ПЗ.
2. Провести проектування ПЗ.
3. Здійснити програмування та тестування.
4. Провести розгортання ПЗ.

Процес формування специфікації ПЗ – це процес, при якому відбувається аналіз вимог та документується опис поведінки системи, з урахуванням усіх технічних можливостей та обмежень. Відповідно до завдання дипломної роботи необхідно сформувати список функціональних вимог до системи керування чат-ботом:

1. Перегляд користувачів чат-ботом (загальна інформація про користувача, дата останнього візиту, стан).

2. Чат з можливістю листування між ботом та користувачами у форматі текстових та медіа повідомлень, а також зі збереженням історії листувань.

3. Файловий сервіс для збереження будь-яких файлів, що були задіяні у роботі бота.

4. Сервіс розсилок з можливістю надсилання текстових та медіа повідомлень групам користувачів: можливість планувати розсилки на певну дату, перегляд поточного стану розсилок, архів розсилок.

5. Перегляд статистичних даних (кількість нових користувачів за період часу, повідомлень за період часу, запланованих та активних розсилок, об’єм використаного сховища даних).

На основі переліку функціональних вимог є необхідністю сформувати конкретні вимоги: інтерфейсу користувача, програмного інтерфейсу, комунікаційних протоколів, бази даних.

Інтерфейс користувача системи керування чат-ботом будемо розробляти у вигляді крос-браузерного веб-інтерфейсу з доступом за допомогою системи логіна/паролю. Це спростить розробку та підтримку, оскільки браузерні додатки підтримуються на будь-яких платформах та операційних системах (ОС), і залежать лише від використовуваного браузеру, а розробка додатків для персональних комп’ютерів (ПК) та мобільних платформ вимагають більшого об’єму роботи з точки зору кросплатформеності кодової бази. Технологія для написання веб-інтерфейсу – *React*, згідно з темою дипломної роботи.

2.2. Аналіз та вибір архітектури

Проведемо проектування ПЗ. Основним і найважливішим етапом проектування ПЗ є вибір архітектури.

Архітектура ПЗ – це процес структурування ПЗ на слабкозв’язані та незалежні частини, формування зв’язків та опис процесів передачі даних між ними. Архітектура ПЗ будується з метою найкращої відповідності вимогам проекту.

Згідно з сформованими функціональними вимогами вирішено розглянути наступні архітектурні шаблони та стилі:

1. Клієнт-серверна архітектура.
2. *Front end* та *back end*.
3. Монолітна/мікросервісна архітектура.
4. *MVC*.

Клієнт-серверна архітектура – домінуюча концепція створення розподілених мережевих застосунків, яка встановлює правила взаємодії та обміну даними між ними.

Передбачаються наступні компоненти:

1. Набір серверів, які надають інформацію на запит.
2. Набір клієнтів, які використовують сервери та отриману від них інформацію.
3. Мережа, що забезпечує обмін інформацією між клієнтами та серверами.

В системі керуванням чат-ботом існують три сторони:

1. *Telegram Bot API*.
2. Сервер з основною логікою роботи чат-бота.
3. Клієнт.

Серверною стороною виступають сервери *Telegram* по відношенню до серверу логіки чат-бота, сервер логіки чат-бота по відношенню до клієнтського веб-інтерфейсу, а клієнтами – сервери логіки чат-бота по відношенню до серверів *Telegram* та клієнтський веб-інтерфейс по відношенню до серверів логіки чат-бота (рис. 2.1).

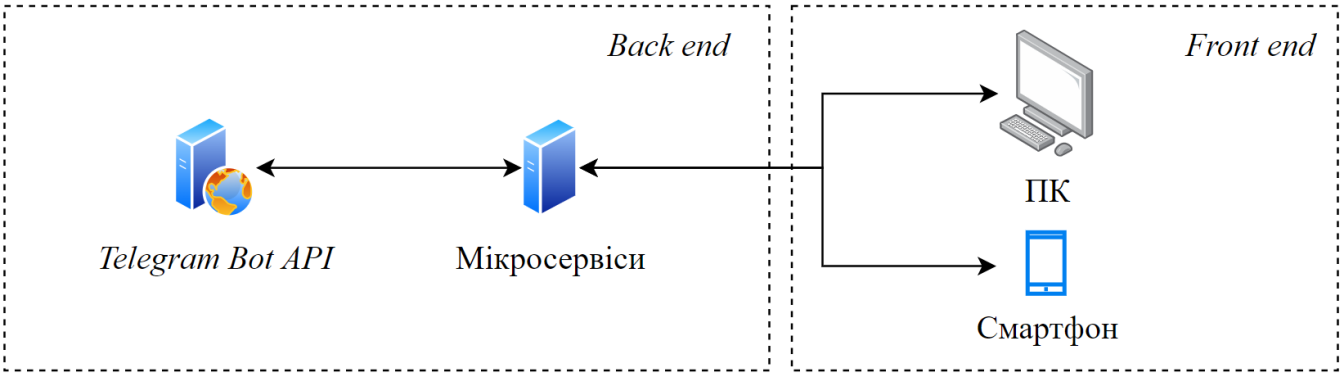

Рис.2.1. Клієнт-серверний погляд на архітектуру системи

Клієнтський веб-інтерфейс відповідає за рівень представлення даних (відображення даних та ввід команд), *Microservices* – за рівень управління даними та прикладний рівень (збереження даних та їх логічна обробка).

Монолітна/мікросервісна архітектура. При розробці великих систем, або ж систем з великою кількістю слабозв'язаних компонентів, критичною необхідністю є вибір основної архітектури. За роки розвитку ПЗ сформувалися два основні підходи.

Монолітна архітектура – це архітектура ПЗ, у якій усі компоненти додатку знаходяться в межах одного процесу ОС та обмінюються даними за допомогою внутрішніх інтерфейсів.

Використання монолітної архітектури має наступні переваги:

1. Спрощений процес розробки. Сучасні інтегровані середовища розробки (*IDE*) першочергово підтримують розробку монолітних додатків.

2. Спрощений процес розгортання.

3. Спрощений процес масштабування. Додаток можливо масштабувати за допомогою запуску декількох копій за балансувальником навантаження.

Проте зі зростанням складності та розмірів ПЗ монолітна архітектура набуває недоліків:

1. Складність у підтримці, розумінні та модифікаціях при великій кодовій базі додатку.

2. Сповільнення швидкості роботи *IDE*, процесу завантаження додатку.

3. Унеможливлення частих оновлень додатку, оновлення одного компоненту втручається в роботу інших через необхідність перезавантаження усього додатку.

4. Помилка в одному компоненті веде до відмови усієї системи.

5. Масштабування додатку можливе лише в одній площині.

6. Складність зміни технологій та їх конкретних версій.

Мікросервісна архітектура – це архітектура ПЗ, в якій усі його компоненти логічно поділяються для виконання слабкозв’язаних функцій та працюють в різних процесах ОС. Мікросервіси одного додатку можуть працювати на різних фізичних машинах, а обмін даними між ними базується на стандартизованих протоколах *RPI* або обміну повідомленнями (*Messaging*).

Використання мікросервісної архітектури має наступні переваги:

1. Створення невеликих, слабкозв’язаних компонентів веде до спрощення їх розуміння, розробки, тестування, підтримки, оновлення та розгортання.

2. Пришвидшення розробки за рахунок зростання швидкодії *IDE* та зменшення часу завантаження мікросервісів.

3. Покращення надійності системи. Відмова або помилка в одному компоненті ізольована від втручання в роботу інших.

4. Можливість заміни або оновлення технологій та фреймворків без критичного впливу на терміни або роботу проекту.

Недоліки мікросервісної архітектури:

1. З’являється необхідність створення механізмів обміну даними між мікросервісами.

2. Ускладняється синхронізація даних.

3. Ускладняється процедура розгортання мікросервісного додатку.

Згідно з завданням дипломної роботи, поставленим вимогам та аналізу недоліків монолітної архітектури зроблено висновок, що використання мікросервісної архітектури забезпечить наступні переваги:

1. Забезпечить незалежну розробку кожного компонента системи.

2. Призведе до можливості простішого масштабування по так званим осям *X*, *Y* та *Z* у порівнянні з монолітною архітектурою (рис. 2.2) [13].

3. Забезпечить використання сучасних технологій та фреймворків, з можливістю їх легкої заміни за необхідністю.

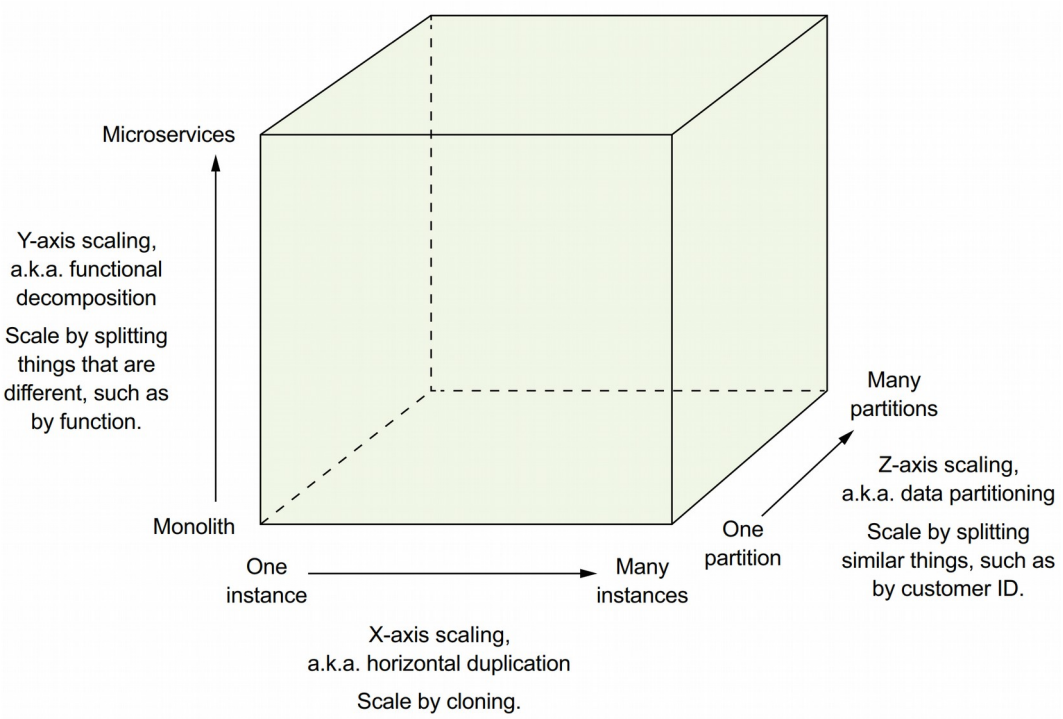

Рис.2.2. Масштабування мікросервісних додатків

Model-View-Controller(MVC) – це шаблон розробки ПЗ, який ділить класичне відображення, зберігання та обробку даних на окремі взаємопов’язані частини (рис. 2.3).

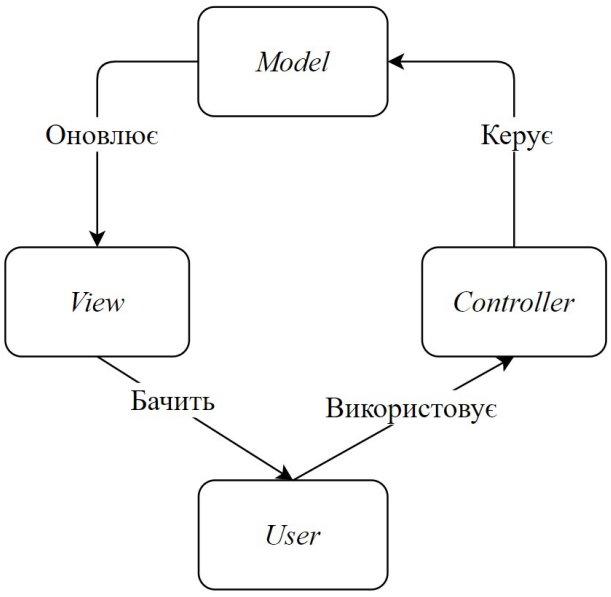

Рис.2.3. Діаграма дій *MVC*

Модель – це частина, яка відповідає за зберігання даних, та яка оновлюється в залежності від дій користувача (елементи управління – контролер).

Вид – це частина, яка відповідає за відображення даних (стану моделі) користувачу. Вид оновлюється тільки при оновленні моделі.

Контролер – це частина, яка відповідає за обробку дій користувача і найчастіше виступає стандартними елементами управління користувацького інтерфейсу (кнопками, полем вводу, перемикачами тощо).

2.3. Вибір мови програмування

Вибір мови програмування для будь-якого проекту є важливим процесом, у подальшому від якого безпосередньо буде залежати швидкість та якість його виконання.

Факторами розвитку та використання мов програмування є:

1. Наявність та підтримка *IDE*.
2. Зручність супроводження та відлагодження додатків.
3. Вартість розробки ПЗ.
4. Чіткість та зрозумілість синтаксису мови.
5. Можливість використання шаблонів, підходів та найкращих практик розробки ПЗ.
6. Можливості стандартної бібліотеки.
7. Мова є компільованою/інтерпретованою.

Згідно з вищезазначеними факторами виокремлюються мови програмування, які найкраще застосовуються в тій чи іншій галузі:

1. Наукові обчислення та обробка інформації: *C++*, *Python*, *Java*.
2. Системне програмування: *C*, *C++*.
3. Опис документів та стилів: *HTML*, *XML*, *CSS*.
4. Штучний інтелект: *LISP*, *Python*.
5. Мережеві додатки та веб-сервери: *C#*, *Java*, *JavaScript*.
6. *Front end* додатки: *JavaScript*.

2.3.1. Платформа N*ode.js*

Відповідно до поставлених вимог завдання дипломного проекту, необхідності активного мережевого обміну даними між створеною системою та серверами *Telegram*, мікросервісами, мікросервісами та *front end* додатком, а також за необхідністю швидкої розробки та простого відлагодження, було зроблено рішення використати мову програмування *JavaScript* для створення системи керування чат-ботом, а конкретно – платформу *Node.js*.

Node.js – це платформа для побудови масштабованих мережевих додатків за допомогою мови *JavaScript*. Основні властивості:

1. Асинхронна одно-нитева модель виконання запитів.

2. Неблокуючий ввід-вивід.

3. Рушій *JavaScript Google V*8.

В основу *Node.js* закладений цикл подій (*Event loop*). Цикл подій послідовно виконує фази:

1. *Timers* – відбувається виклик функцій, які були заплановані за допомогою методів *setTimeout*() та *setInterval*().

2. *Pending callbacks* – відбувається обробка системних викликів операцій вводу/виводу.

3. *Idle, prepare*.

4. *Poll* – отримання нових подій вводу/виводу та виконання функцій, які з ними пов’язані.

5. *Check* – виконуються функції, що були заплановані за допомогою методу *setImmediate*().

6. *Close calbacks* – виконуються методи для закриття з’єднань.

Варто зазначити, що стандартна бібліотека *Node.js* наразі підтримує суттєвий функціонал, який є необхідним для виконання завдання дипломного проекту, а саме модулі: *HTTP*, *Events*, *Crypto*, *Process*, *File system*.

2.3.2. Фреймворки *Express.js* та *Socket.io*

Створення будь-якого веб-додатку супроводжується необхідністю в активному обміні даними між веб-сервером та *front end* додатком. При цьому сам процес обміну даними строго підпорядковується правилам протоколу *HTTP*. Протокол *HTTP* встановлює базові методи для обміну даними та іншою інформацією (*GET*, *POST*, *PUT*, *DELETE*, *OPTIONS* тощо). Текстові дані зазвичай представляються у форматі *JSON* з типом *application/json*, а бінарні дані – типом *multipart/form-data*.

Node.js має стандартний модуль *HTTP*, який дозволяє створити та запустити веб-сервер на довільному *IP* та порті. Модуль має повноцінну підтримку протоколу *HTTP*, і саме тому є доволі низкорівневим компонентом. Тому для спрощення створення веб-додатків є доцільним використати надбудову над стандартним модулем *HTTP Node.js*.

Express.js – це веб-фреймворк для *Node.js*, який додає до стандартного модуля *HTTP* набір високорівневих методів та систему проміжних обробників (*middleware*). *Middleware* забезпечує потужну та просту систему маршрутизації запитів, при яких кожному *URI* назначається одна або декілька функцій. Для одного *URI* може бути назначено декілька різних *HTTP* методів. *URI* при цьому зазначається у вигляді рядка або регулярного виразу, що забезпечує гнучкий підхід до визначення кінцевих точок (*endpoints*) веб-додатку.

Express.js буде використаний при розробці системи керування чат-ботом як надбудова над *HTTP* модулем *Node.js*.

Відповідно до поставлених вимог необхідно створити чат для обміну повідомленнями з користувачами чат-боту. Однієї із функціональних вимог такого чату є пересилання повідомлень у режимі реального часу для обох сторін. Стандартний підхід до реалізації обміну даними між веб-сервером та клієнтом є варіант “запит-відповідь”, що не задовольняє функціональні вимоги чату у повній мірі. Необхідно використати підхід, що заснований на використанні технології *WebSocket*.

WebSocket – це протокол, що призначений для обміну інформацією між браузером та веб-сервером в режимі реального часу. Обмін інформацією відбувається через двонаправлений повнодуплексний канал передачі даних через один *TCP* сокет. Хоча *API WebSockets* стандартизований та прописаний в документі *RFC*6455 [14] сама технологія має різну реалізацію та підтримку в окремих браузерах. Для вирішення цієї проблеми є доцільним використання готового фреймворку.

Socket.io – фреймворк для веб-застосунків і обміну даними в реальному часі. Складається з двох частин: клієнтської, яка запускається в браузері і серверної для *Node.js*. Обидва компоненти мають схожий *API*.

Socket.IO головним чином використовує протокол *WebSocket*, але якщо потрібно, використовує інші методи, наприклад *AJAX* запити, надаючи той же самий інтерфейс.

Протокол *WebSocket* буде використаний при розробці системи керування чат-ботом для цілей синхронізації даних між клієнтом та сервером та реалізації функцій чату.

2.4. Вибір бази даних. *SQLite*

База даних (БД) – це набір даних та структур, що мають певні ознаки та зв’язки. Об’єднання даних у єдину базу даних забезпечує формування варіацій групування інформації.

При розробці веб-додатків зазвичай використовуються реляційні БД, що зберігають усю необхідну інформацію у вигляді таблиць та зв’язків між ними. З метою доступу та керування інформації використовується мова *SQL*, що дозволяє будувати запити до БД для запису, оновлення, видалення, отримання та структурування даних.

Популярними системами управління базами даних (СУБД) є: *Oracle Database*, *Postgre SQL*, *Microsoft SQL*, *SQLite*.

СУБД вирішують наступні задачі:

1. Управління операціями з даними.

2. Підтримка цілісності БД (коректність та несуперечливість).

3. Бекап та відновлення стану БД.

4. Організація одночасного доступу до даних.

5. Захист даних.

Аналізуючи вимоги до системи керування чат-ботом та зважаючи на обрану мікросервісну архітектуру вірним рішенням буде використання шаблону *Database per service*. Таким чином буде забезпечено найвищий рівень слабкозв'язності мікросервісів.

Відповідно до вищезазначеного, найкращою СУБД для проекту стане локальна СУБД.

SQLite – це реляційна СУБД написана на мові *C*, яка вбудовується безпосередньо в додаток.

Особливості *SQLite*:

1. Повністю підтримує концепцію *ACID*.

2. Реалізує значну частину стандарту *SQL*-92.

3. Має прив'язки до багатьох мов програмування, включно з *Node.js*.

4. Зберігає всю БД одним крос-платформовим файлом, що доступний локально, максимальний розмір файлу – 281 ТБ.

5. Не потребує первинної конфігурації.

6. Не має зовнішніх залежностей, увесь код розміром до 350 Кб.

7. Має простий та зрозумілий *API*.

2.5. Технологія *React.js*

React.js – це декларативна, швидка та гнучка технологія побудови інтерфейсів користувача за допомогою мови *JavaScript*. *React.js* має декілька особливостей, які значно спрощують процес розробки веб-інтерфейсів. Розглянемо деякі з них.

Компоненти. Функціональною одиницею технології є *Components* (компоненти). Говорячи про функціональні одиниці маємо на увазі візуальні

елементи користувацького інтерфейсу. Ці елементи створюються за допомогою синтаксису *JSX*. Використання цього синтаксису дозволяє вбудовувати *HTML* безпосередньо до коду *JavaScript*, об'єднуючи *View* та *Controller* архітектури *MVC* воєдино.

Віртуальний *Document Object Model* (*DOM*). Особливість *React.js*, яка робить високопродуктивною роботу з відображенням компонентів у *DOM* (рис. 2.4).

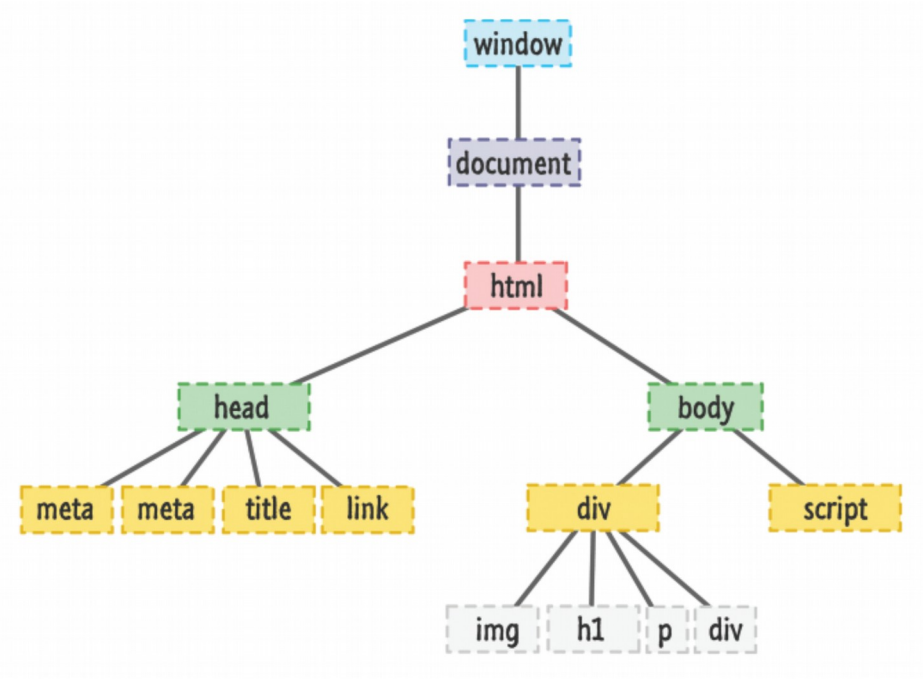

Рис.2.4. Приклад складових частин *DOM*

Підхід заснований на створенні віртуальної моделі *DOM* в оперативній пам'яті та, при будь-яких змінах, її першочерговому оновленні. Після оновлення віртуального *DOM* відбувається процедура порівняння даних в реальному та віртуальному *DOM*. У результаті оновлюється лише та частина реального *DOM* веб-сторінки, в якій були знайдені розбіжності. Цей процес називається *reconciliation*.

Життєвий цикл компонентів та відображення даних. Керування відображенням даних в компонентах відбувається за допомогою двох підходів: *state* та *props*. *State* зберігає візуальний стан компонента (дані для відображення) у конкретний момент часу, при цьому його зміна можлива лише при умові зміни *State* за допомогою методу *setState*(). *Props* – це параметри компонентів, які надходять до

компоненту ззовні, найчастіше від батьківських компонентів. На основі зміни *props* компонент може змінювати своє представлення.

Життєвий цикл компоненту – це період, за час якого компонент створюється, змінюється та знищується. Відслідковування життєвого циклу компоненту засноване на програмуванні його методів: *componentWillMount*, *componentDidMount*, *componentWillReceiveProps*, *componentWillUpdate*, *componentDidUpdate*, *componentWillUnmount*. Життєвий цикл компоненту показано на рис. 2.5.

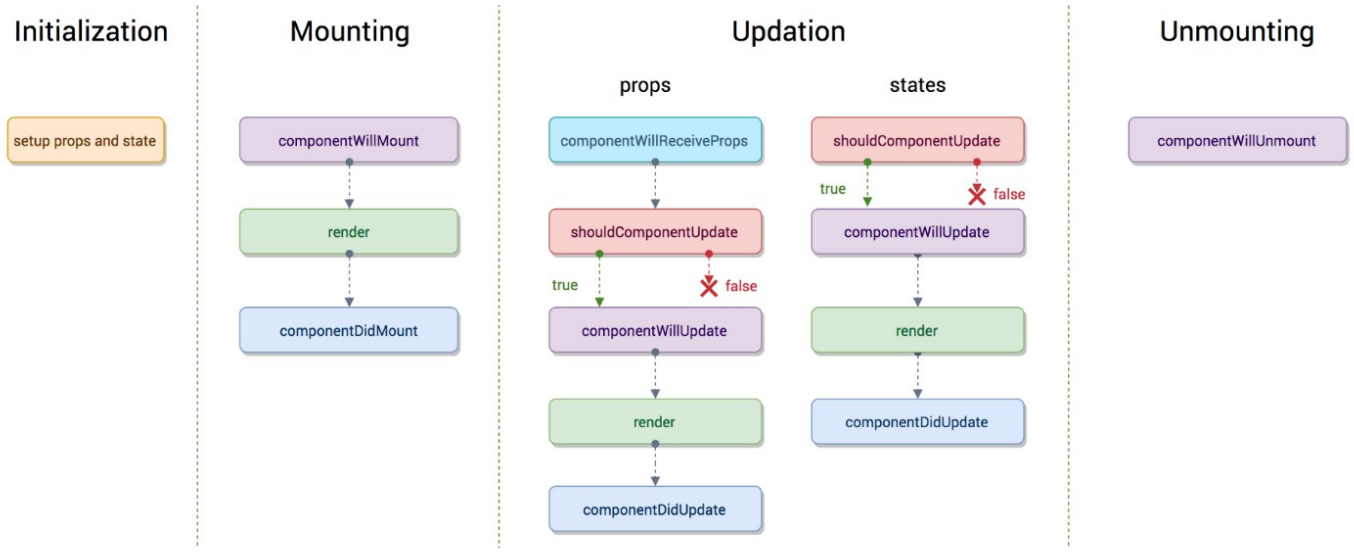

Рис.2.5. Життєвий цикл компоненту *React.js*

В методах життєвого циклу зазвичай відбувається отримання початкових даних для представлення компоненту та їх оновлення. Для реагування на дії користувача використовується система подій (*events*), яка майже повністю повторює концепцію *Event – EventListener* в класичному *DOM*.

Висновки за розділом

У цьому розділі розглянуті питання аналізу та вибору технологій для розробки системи керування чат-ботом: проаналізовано поняття “система” та його значення у контексті розробки ПЗ; визначений план процесу розробки ПЗ по стандартному шаблону; сформовані основні вимоги до розроблюваної системи керування чат-ботом; проведений аналіз та вибір архітектури ПЗ; обґрунтований вибір мови

програмування та фреймворків; обрана БД для зберігання даних; детально описана технологія *React.js*.

У результаті встановлено, що для розробки системи керування чат-ботом, як і будь-якого іншого комплексного ПЗ, необхідно: сформувати вимоги, здійснити формальне проектування, здійснити програмування та тестування, описати процеси розгортання. Тому, відповідно до послідовності описаного процесу, були сформовані основні функціональні вимоги до системи керування чат-ботом: листування, розсилки, обмін файлами та інші.

Окрема увага у розділі присвячена питанню вибору програмної архітектури системи. Детально описані та порівнянні два підходи: монолітний та мікросервісний. У результаті порівняння був зроблений висновок, що мікросервісна архітектура найкраще підходить для реалізації системи через можливості масштабування окремих сервісів та декомпозиції по функціоналу. Наголошено на негативних наслідках такого вибору.

Важливою частиною розділу є обґрунтування вибору технологій для розробки. З метою пришвидшення та спрощення розробки системи керування чат-ботом були обрані технології *Node.js*, *Express.js*, *Socket.io* для програмування логіки передачі та обробки даних. Варто зазначити, що їх використання відповідає обраній архітектурі. БД *SQLite* обрана через повну підтримку стандарту *SQL* та локальний характер зберігання даних.

Особлива увага приділена технології *React.js*, яка дозволяє проектувати та реалізовувати клієнтські веб-інтерфейси у компонентному стилі, має розвинуту систему керування та оновлення *DOM*, керується передбачуваною моделлю поведінки компонентів та частково заснована на шаблоному патерні *MVC*.

РОЗДІЛ 3
РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ КЕРУВАННЯ ЧАТ-БОТОМ

3.1. Декомпозиція системи на мікросервіси

Одним із основних етапів створення системи керування чат-ботом є розробка її серверної частини. Серверна частина буде відповідальна за наступне:

1. Обмін даними з серверами *Telegram*, логіка чат-бота, надсилання/отримання запитів до/від *Telegram Bot API*.

2. Функціональні компоненти системи у вигляді мікросервісів: розподіл зон відповідальності обробки даних, їх обмін та збереження.

3. Веб-*API*.

4. Захист даних.

Мікросервіси. Відповідно до обраної архітектури системи необхідно здійснити декомпозицію системи на мікросервіси.

Існує декілька підходів для здійснення процесу декомпозиції, наприклад: розбиття на основі бізнес-можливостей та розбиття на основі субдоменної моделі. Незалежно від обраного підходу необхідно притримуватись наступних принципів:

1. *Single responsibility principle* (принцип єдиної відповідальності).

2. *Common closure principle* (принцип відкритості/закритості).

Здійснимо декомпозицію системи керування чат-ботом. Для цього доцільним буде визначити окремі субдоменні моделі. Субдоменними моделями будуть:

1. Логіка чат-бота (автомат з жорсткою логікою дій бота, дані чат-бота).

2. Взаємодія з *Telegram Bot API*.

3. Авторизація користувачів.

4. Керування файлами.

5. Робота з повідомленнями (чат).

6. Здійснення розсилок.

Відповідно до визначених субдоменних моделей отримуємо список сервісів (рис. 3.1).

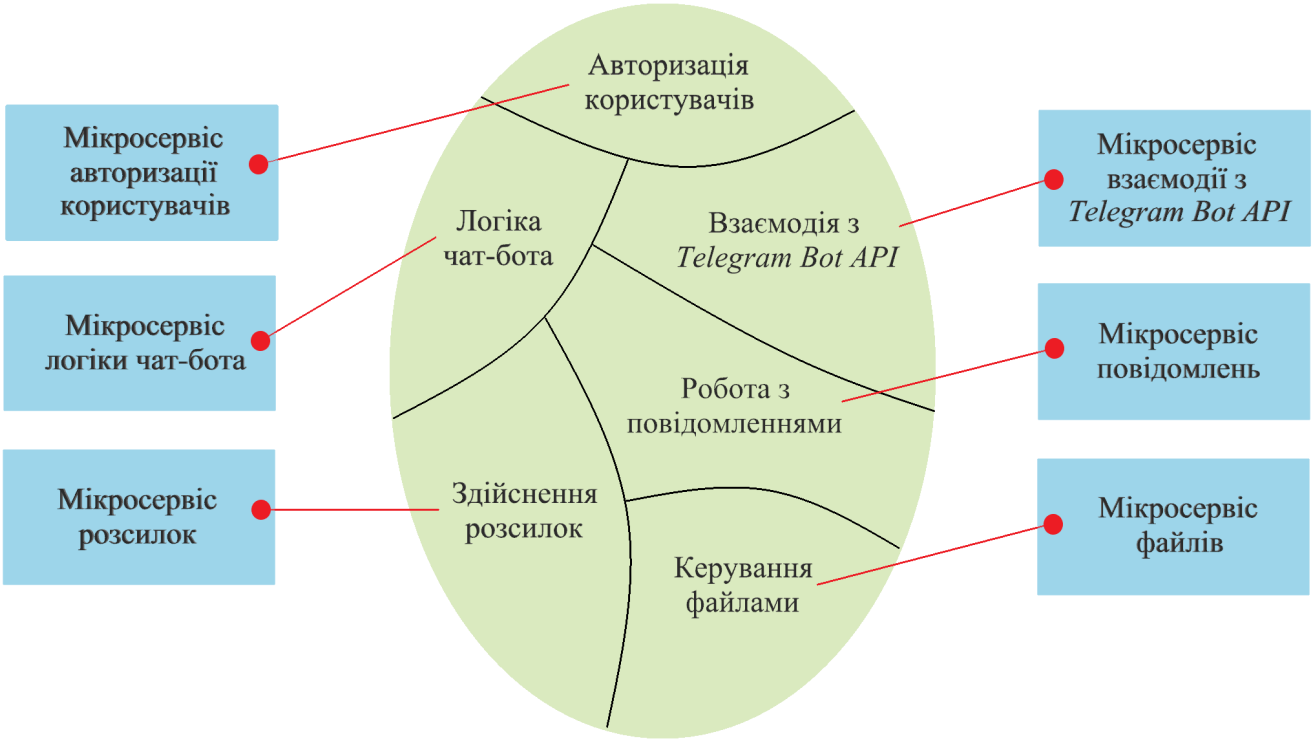

Рис.3.1. Мікросервіси системи керування чат-ботом

3.2. Опис *API* та БД мікросервісів

API мікросервісів. Наступним кроком у розробці системи є визначення *API* кожного сервісу: операцій (команд) та подій. Команди сервісу існують з двох причин: деякі команди призначені для виконання операцій, що викликаються зовнішніми клієнтами або іншими сервісами, інші ж команди призначені виключно для підтримки взаємодії між сервісами. Події зазвичай використовуються для взаємодії між сервісами: для підтримки синхронізації та узгодженості даних або для сповіщення зовнішніх клієнтів (наприклад *WebSockets* надсилають події до браузеру клієнта).

Початкова точка створення *API* сервісів полягає у визначенні операцій для кожного. Також встановимо взаємозв’язок між сервісами та їх операціями (рис. 3.2).

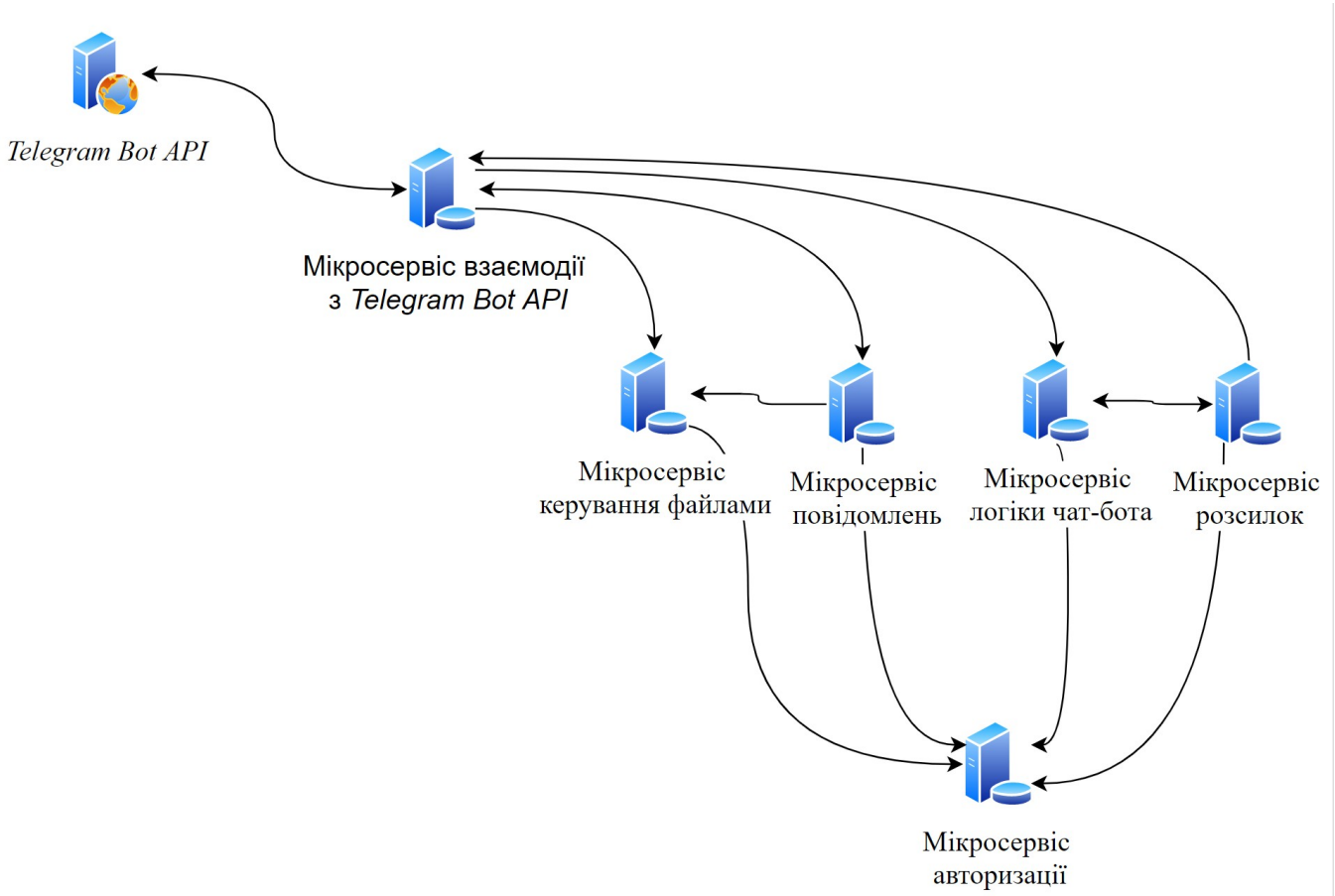

Рис.3.2. Взаємозв’язок між сервісами

Визначимо *API* мікросервісу взаємодії з *Telegram Bot API* (табл. 3.1).

Таблиця 3.1

API мікросервісу взаємодії з *Telegram Bot API*

Операції	**Взаємодія з сервісами**	**Опис**
connectToTelegramAPI(params) *createBotGateServiceListener(params)* *createBotLogicServiceConnection(params)* *createFileServiceConnection(params)*		Ініціалізація сервісу.
onTelegramAnyAction(*action*)	Сервіс логіки чат-бота	Обробка подій отриманих від *Telegram Bot API*.
onTelegramTextMessage(*msg*) *onTelegramPhotoMessage*(*msg*) *onTelegramDocumentMessage*(*msg*)	Сервіс повідомлень	Обробка текстових або медіа повідомлення.
sendTelegramTextMessage(*msg*) *sendTelegramPhotoMessage*(*msg*) *sendTelegramDocumentMessage*(*msg*)	Сервіс файлів	Відправка повідомлень.

Даний мікросервіс працюватиме як обгортка над методами обміну даними з *Telegram Bot API*. Сервіс зберігатиме усі необхідні для його роботи дані у оперативній пам’яті, без необхідності у БД.

Оскільки сервіс відповідальний за створення з’єднання між системою та серверами *Telegram*: *connectToTelegramAPI(params)*, варто зазначити, що саме на цьому етапі визначається механізм з’єднання: *Polling* або *Webhook*.

Також важливою частиною є обробка помилок, які можуть виникнути при запитах до *Telegram Bot API*. Можливі помилки:

1. Користувач заблокував чат-бота.

2. Перевищений ліміт надсилання запитів.

3. Помилкові параметри запиту.

4. Інші мережеві помилки.

Ліміт повідомлень встановлений в *Telegram* – не більше 30 повідомлень в секунду. Визначимо *API* мікросервісу логіки чат-бота (табл. 3.2).

Таблиця 3.2

API мікросервісу логіки чат-бота

Операції	**Взаємодія з сервісами**	**Опис**
1	2	3
createWebServerListener(*params*) *createWebSocketsListener*(*params*) *createBotLogicServiceListener*(*params*) *createAuthServiceConnection*(*params*)		Ініціалізація сервісу.
createOrUpdateTelegramUser(*tgUser*) *getTelegramUsers*() *getTelegramUserById*(*tgUserId*)		Основні міжсервісні операції з даними користувачів чат-ботом.
getTelegramUsers() *getTelegramUsersCount()* *getTelegramUsersByParams*(*params*) *createReferral*(*referral*) *getReferrals*() *getReferralById*(*referralId*) *updateReferral*(*referral*) *syncTelegramUsersPages*() *syncTelegramUsers*(*params*)	Сервіс авторизації.	*HTTP* та *WebSockets API*.

Продовження таблиці 3.2

1	2	3
telegramUserCreated(*tgUser*) *telegramUserUpdated*(*tgUser*)		Події *WebSockets*.

Даний сервіс здійснює обробку даних користувачів чат-ботом та системи рефералів.

Система рефералів призначена для створення спеціальних іменованих посилань на чат-бот. Створені реферали використовуються для підрахунку нових користувачів, з врахуванням того, через яке саме посилання користувач перейшов до боту.

Мікросервіс буде оновлювати дані у результаті керівних команд від сервісу взаємодії з *Telegram Bot API*. Даний мікросервіс є одним із найважливіших, оскільки його дані містять інформацію про основний ресурс чат-ботів – користувачів.

Доступ до даних клієнтам надається за допомогою протоколів *HTTP* та *WebSockets*.

Усі дані зберігаються у БД *SQLite*, схема таблиць представлена на рис. 3.3.

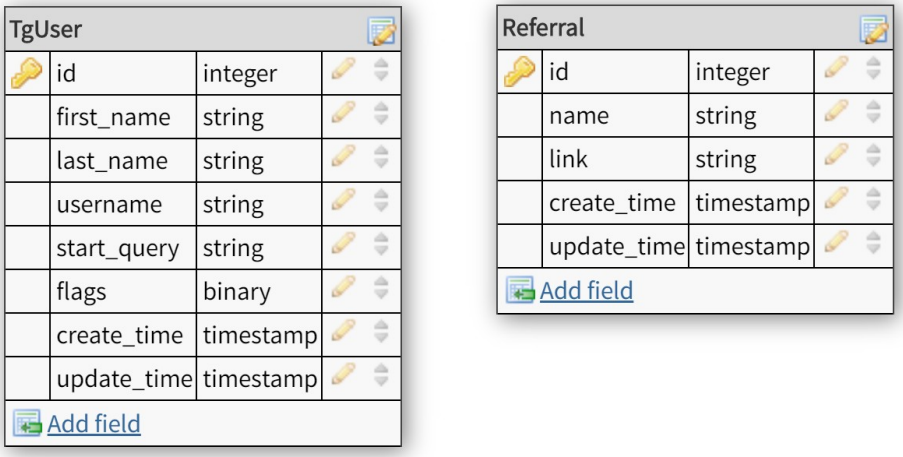

Рис.3.3. Схема БД сервісу логіки чат-бота

Визначимо *API* сервісу авторизації користувачів веб-додатком (табл. 3.3). Даний сервіс буде використаний усіма сервісами, які підтримують передачу даних через протоколи *HTTP* та *WebSockets*.

Таблиця 3.3

API мікросервісу авторизації

Операції	**Взаємодія з сервісами**	**Опис**
createAuthServiceListener(*params*) *createWebServerListener*(*params*)		Ініціалізація сервісу.
login(*login*, *password*) *validateUserSession*(*userSessionId*) *validateToken*(*token*)		Керування сесіями користувачів.
service()		Перевірка протермінування сесій.

Сервіс буде забезпечувати створення сесій користувачів та токенів доступу при їх успішній автентифікації за допомогою логіна/паролю. Подальша ідентифікація користувача відбуватиметься за допомогою механізму *HTTP-only cookies* (рис. 3.4).

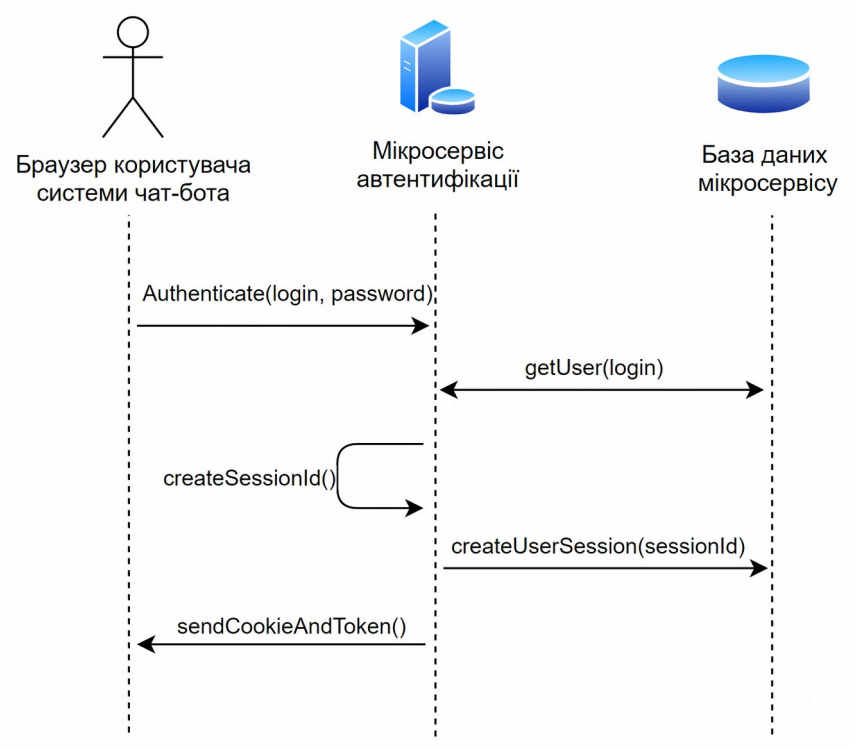

Рис.3.4. Процес успішної автентифікації користувача у системі керування чат-ботом

Для сторонніх протоколів (наприклад для синхронізації даних за допомогою механізму *WebWorker*) окремо генеруватимемо токен доступу. Термін дії сесії – одна година, при умові відсутності запитів від користувача протягом цього часу. З кожним запитом термін дії поновлюватиметься. Паролі користувачів зберігаються в БД у вигляді *MD5* хешу. З точки зору безпеки та захисту персональних даних користувачів забороняється зберігати їх у відкритому вигляді.

Усі дані зберігаються у БД *SQLite*, схема таблиць представлена на рис. 3.5.

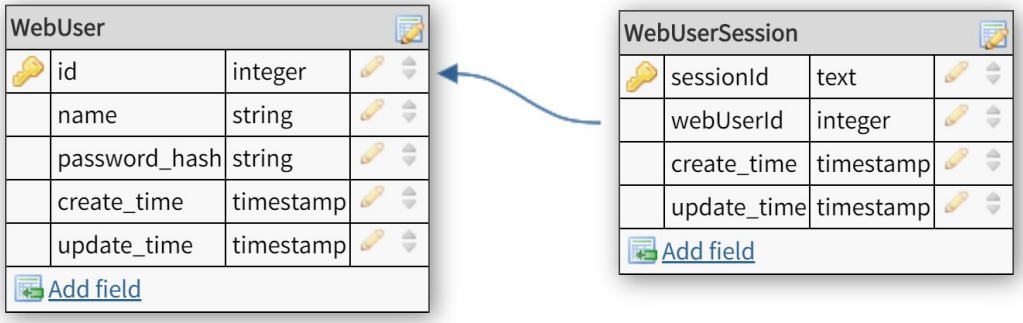

Рис.3.5. Схема БД сервісу авторизації

Визначимо *API* мікросервісу керування файлами (табл. 3.4).

Таблиця 3.4

API мікросервісу файлів

Операції	**Взаємодія з сервісами**	**Опис**
createWebServerListener(*params*) *createWebSocketsListener*(*params*) *createFileServiceListener*(*params*) *createAuthServiceConnection*(*params*)		Ініціалізація сервісу.
createPhoto(*photo*) *getPhotoById*(*photoId*) *updatePhoto*(*photo*) *createDocument*(*document*) *getDocumentById*(*documentId*) *updateDocument*(*document*)		Основні між сервісні операції.
createPhoto(*photo*) *getPhotoInline*(*photoId*) *getPhotoAttachment*(*photoId*) *updatePhoto*(*photo*)	Сервіс авторизації.	*HTTP* та *WebSockets API*.

createDocument(*document*) *getDocumentAttachment*(*documentId*) *updateDocument*(*document*) *getFilesByParams*(*params*)		
downloadFileByUrl(*url*)		Завантаження файлу з мережі.

Файл – це впорядкований набір байтів, що зберігається у постійній пам’яті комп’ютеру.

Обробка файлів та налаштування доступу до них, з огляду на необхідність враховувати можливості *Telegram Bot API*, має певні особливості. Будь-який файл можливо завантажити до системи двома способами:

1. За допомогою веб-додатку.

2. За допомогою чат-боту.

Для кожного файлу, який потрібно зберегти в системі, будемо генерувати унікальне ім’я. Розширення файлу при цьому вказувати не будемо, оскільки робота з файлами планується тільки в межах системи, їх знеособлення підвищить безпеку системи. Уся інформація по файлам зберігатиметься в БД: початкове ім’я файлу, *MIME*-тип, розмір в байтах, параметри доступу тощо. Окремо варто зазначити, що сервера *Telegram* визначають унікальне поле *file_id*, знання якого забезпечує доступ до файлу та можливість його пересилання у повідомленнях, без необхідності повторного завантаження. Ця можливість значно зекономить трафік та час при виконанні масових розсилок.

Дані сервісу зберігаються у БД *SQLite* та у постійній пам’яті серверу, схема таблиць БД представлена на рис. 3.6.

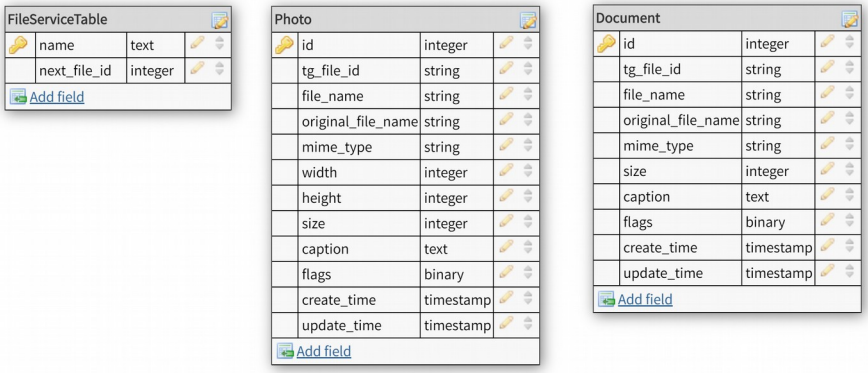

Рис.3.6. Схема БД сервісу файлів

Визначимо *API* сервісу розсилок (табл. 3.5). Написання реалізації даного мікросервісу є однією з найскладніших задач. Можливості:

1. Планування текстових та медіа повідомлень для груп користувачів, їх надсилання у встановлену дату та час.

2. Створення персоналізованих розсилок (повідомлень, які надсилаються новим користувачам чат-боту після встановленого проміжку часу).

3. Відображення інформації по новим та старим розсилкам у режимі реального часу.

4. Керування розсилками у будь-який момент часу, обробка помилок, відновлення роботи у випадку перезапуску сервісу.

Таблиця 3.5

API мікросервісу розсилок

Операції	**Взаємодія з сервісами**	**Опис**
createWebServerListener(*params*) *createWebSocketsListener*(*params*) *createMailingServiceListener*(*params*) *createAuthServiceConnection*(*params*) *createBotGateServiceConnection*(*params*) *createBotLogicServiceConnection*(*params*)		Ініціалізація сервісу.
createPersonalMailing(*personalMailing*)		Основні між сервісні операції.
createPlannedMailing(*plannedMailing*) *updatePlannedMailing*(*plannedMailing*) *deletePlannedMailing*(*plannedMailingId*) *createPersonalMailing*(*personalMailing*) *updatePersonalMailing*(*personalMailing*) *deletePersonalMailing*(*personalMailing*) *getPlannedMailingsByParams*(*params*) *getPersonalMailingsByParams*(*params*)	Сервіс авторизації. Сервіс *Telegram Bot API*.	*HTTP* та *WebSockets API*.
plannedMailingStarted(*plannedMailingId*) *plannedMailingUpdated*(*plannedMailingId*) *plannedMailingFinished*(*plannedMailingId*) *plannedMailingCancelled*(*plannedMailingId*)		Події *WebSockets*.
plannedMailingService()		Робочі задачі

personalMailingService()		по контролю розсилок.

Усі дані сервісу зберігаються у БД *SQLite*, схема таблиць представлена на рис. 3.7.

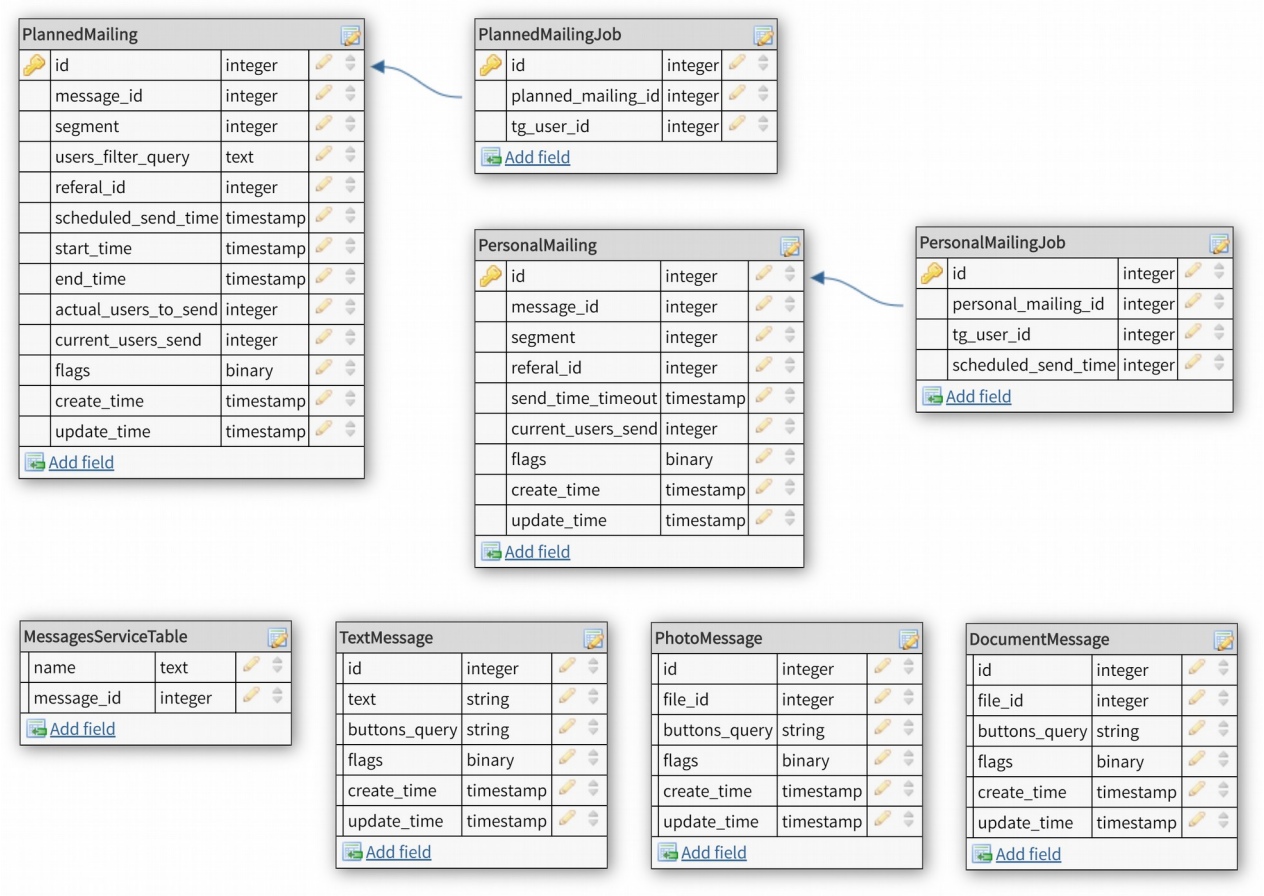

Рис.3.7. Схема БД сервісу розсилок

Даний сервіс створює власні таблиці для збереження даних повідомлень, а не делегує цю задачу сервісу повідомлень. Це є цілком прийнятним підходом у мікросервісній архітектурі. В таблицях *PhotoMessage* та *DocumentMessage* поле *file_id* – це посилання на файли, що розташовані у сервісі керування файлами.

Створення нового запису в таблиці *PlannedMailing* спричиняє планування нового завдання методом *plannedMailingService*(). На початку виконання розсилки сервіс звертається до сервісу логіки чат-бота з запитом списку користувачів, зазначаючи додаткові параметри (*segment*, *users_filter_query*, *referral_id*). Поле *segment* вказує на сегмент користувачів розсилки:

1. Усі користувачі чат-ботом (з можливим фільтром по даті створення користувачів, який зазначається у полі *users_filter_query*).

2. Користувачі, що прийшли з реферального посилання (конкретний *id* реферала зазначається у полі *referral_id*).

Після отримання списку користувачів в БД створюються нові записи в таблиці *PlannedMailingJob*, що гарантує його відновлення у разі перезапуску сервісу під час процесу розсилки. Наступним кроком новостворені записи беруться з таблиці та відбувається надсилання повідомлень до відповідних користувачів за допомогою сервісу взаємодії з *Telegram Bot API*. З кожним успішним надсиланням інформація про поточний стан розсилки в записі таблиці *PlannedMailing* оновлюється, а відповідні записи з таблиці *PlannedMailingJob* видаляються. Під час цього процесу генеруються внутрішні події у системі, завдяки яким інформація про оновлення таблиць надсилається клієнтам через протокол *WebSockets*. Таким чином реалізується можливість спостерігання за процесом розсилки у режимі реального часу.

Персональні розсилки працюють схожим чином. Нова персональна розсилка створюється за допомогою методу *createPersonalMailing()*, що зазвичай має відбуватись при створенні нового користувача чат-ботом у системі. Після чого з таблиці *PersonalMailing* дістаємо усі записи та плануємо завдання, попередньо створюючи записи в таблиці *PersonalMailingJob*. При настанні зазначеного часу відбувається надсилання повідомлення користувачу за допомогою сервісу *Telegram Bot API*. При успішному надсиланні інформація про поточний стан розсилки в записі таблиці *PersonalMailing* оновлюється, а відповідні записи з таблиці *PersonalMailingJob* видаляються. Аналогічно до запланованих розсилок, під час оновлення таблиць генеруються внутрішні події та інформація про оновлення надсилається клієнтам через протокол *WebSockets*.

Визначимо *API* сервісу повідомлень (табл. 3.6). Обмін повідомленнями є основною функцією будь-якого месенджеру, тому дуже важливо повноцінно реалізувати можливість обміну повідомленнями між чат-ботом та системою керування чат-ботом.

Необхідно врахувати типи повідомлень: текстові, фото та файлові, а також забезпечити можливість додаткового розширення підтримуваних типів: відео, голос, анімації тощо.

Таблиця 3.6

API мікросервісу повідомлень

Операції	**Взаємодія з сервісами**	**Опис**
createWebServerListener(*params*) *createWebSocketsListener*(*params*) *createMessageServiceListener*(*params*) *createAuthServiceConnection*(*params*) *createFileServiceConnection*(*params*)		
createTgTextMessage(*message*) *createTgPhotoMessage*(*message*) *createTgDocumentMessage*(*message*)		Основні між сервісні операції.
createTgTextMessage(*message*) *createTgPhotoMessage*(*message*) *createTgDocumentMessage*(*message*) *updateTgTextMessage*(*message*) *updateTgPhotoMessage*(*message*) *updateTgDocumentMessage*(*message*) *getTgMessages*() *getTgMessagesCount*() *getTgMessagesByParams*(*params*) *syncTgMessagesPages*() *syncTgMessages*(*params*)	Сервіс авторизації. Сервіс *Telegram Bot API*. Сервіс файлів.	*HTTP* та *WebSockets API*.
tgTextMessageCreated(*message*) *tgPhotoMessageCreated*(*message*) *tgDocumentMessageCreated*(*message*) *tgTextMessageUpdated*(*message*) *tgPhotoMessageUpdated*(*message*) *tgDocumentMessageUpdated*(*message*)		Події *WebSockets*.

Важливо зазначити, що дані повідомлень можуть займати значний об’єм постійної пам’яті, що може призвести до значних затримок при їх відправці

клієнтам. У подальшому ця проблема буде вирішена за допомогою створення механізму синхронізації даних між *backend* та *frontend*. Для спрощення роботи з даними повідомлень також потрібно пам’ятати про необхідність їх структуризації, в основному – по даті створення та оновлення, а також по стану (прочитане/не прочитане).

Усі дані мікросервісу зберігаються у БД *SQLite*, схема таблиць представлена на рис. 3.8.

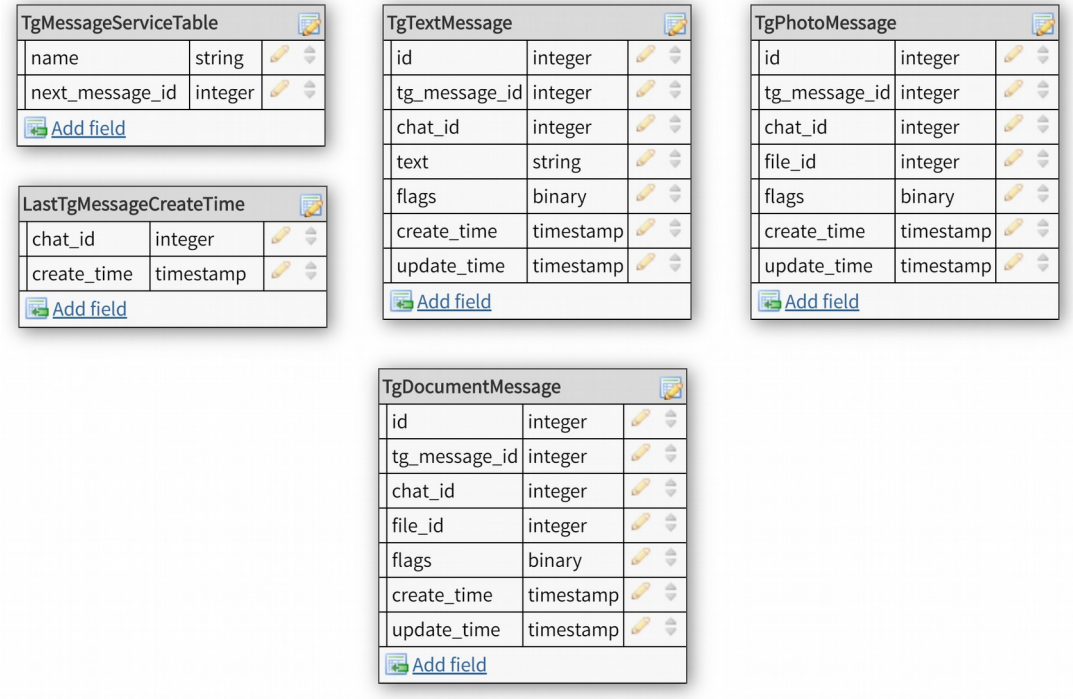

Рис.3.8. Схема БД сервісу повідомлень

3.3. Механізми обміну даними

Обмін даними – це звичайний процес для будь-якої інформаційної системи. Цей процес передбачає наявність:

1. Даних для передачі.
2. Каналу зв’язку.
3. Протоколів та механізмів передачі.

Детально розберемо протоколи та механізми передачі. Їх наявність передбачає певні правила, які розуміють обидві сторони, що задіяні у процесі обміну даними. Правила встановлюють коректність та узгодженість усього процесу.

Оскільки система керування чат-ботом потребує інтенсивного обміну даними, то доцільним рішенням є описати задіяні у цих процесах протоколи. Всього існує три сторони:

1. Клієнт (браузер).
2. Мікросервіси як постачальники даних для клієнта.
3. Мікросервіси як постачальники даних для інших мікросервісів.

Кожна сторона повинна використовувати ті механізми, які якнайкраще підходять для виконання поставлених завдань та максимально спрощують вищезазначені процеси.

Наприклад, використання протоколу *HTTP* для обміну даними між браузерами та веб-серверами – це де-факто стандарт у сфері вебу. Проте, деякі механізми даного протоколу не є вдалими для виконання певних задач.

Для прикладу – механізм встановлення з’єднання та подальшого пересилання інформації передбачає лише однонаправлену їх передачу, з можливістю обміну лише одним повідомленням у межах однієї сесії. Тому для виконання операцій у режимі реального часу використовувати даний протокол не доцільно.

Натомість для виконання пересилання інформації у режимі реального часу призначений протокол *WebSockets*, у якому аналогічний механізм працює методом встановлення з’єднання для подальшого двонаправленого обміну повідомленнями та файлами. Негативною стороною протоколу *WebSockets* є його молодість та обмежена підтримка з боку веб-браузерів (рис. 3.9).

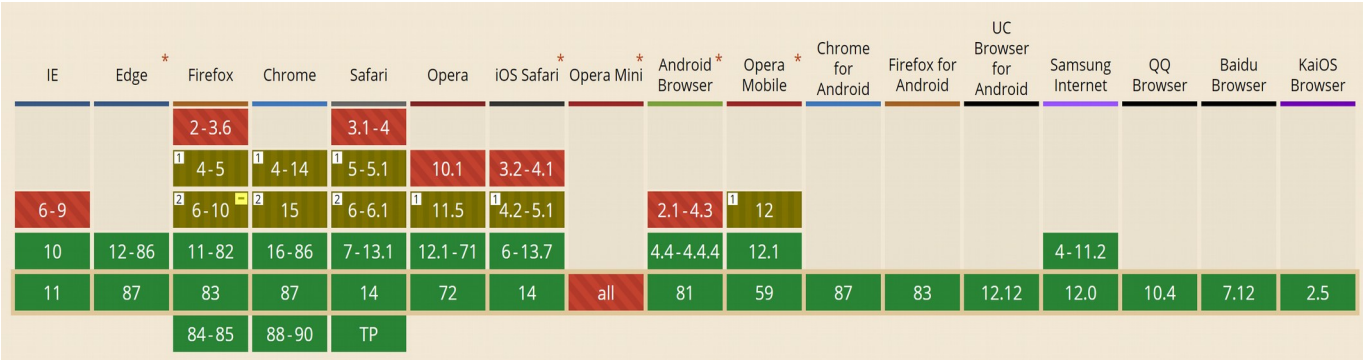

Рис.3.9. Підтримка протоколу *WebSockets* сучасними веб-браузерами

Також до обмежень можна віднести відсутність підтримки загальних механізмів веб-*API*, наприклад – роботу з *Cookies*, та складність процесу авторизації при вже існуючій авторизації по *HTTP*. Тим не менш, у системі керування чат-ботом цей протокол використовується для реалізації можливості обміну повідомленнями, а також для синхронізації даних.

Для обміну даними та командами між мікросервісами використовується механізм *RPI*, а саме – протокол *gRPC*. Використання даного протоколу обумовлюється кодуванням даних з використанням *Protocol Buffers*, що дозволяє формалізовано описати будь-які дані у межах одного *.proto* файлу, який у подальшому можливо використовувати різними мовами програмування на різних ОС. Також важливим аргументом для його впровадження у систему є наявність лише однієї реалізації.

У звичайних додатках часто використовуються механізми синхронізації даних, що передбачають повну або часткову дуплікацію даних на сервері та клієнті. Це роблять з декількох причин:

1. Зменшення часу очікування та кількості помилок.

2. Підвищення надійності.

3. Зникає необхідність у широкому каналі зв’язку, оскільки усі дані передаються частинами.

Відповідно до зазначених причин робимо висновок, що доцільно синхронізувати дані, які мають наступні властивості:

1. Одиниця даних займає невеликий об’єм постійної пам’яті, таких одиниць багато.

2. Часто використовуються.

3. Невелика зв’язність з іншими даними.

Теоретично, завдяки синхронізації даних, також можливо реалізувати офлайн режим роботи веб-додатку, що може використовуватися при деяких сценаріях роботи.

У системі керування чат-ботом доцільна синхронізація повідомлень та користувачів. Для реалізації цього процесу створимо механізм синхронізації даних (рис. 3.10).

Механізм використовує наступні поняття:

1. *Limit* – кількість одиниць даних у одній сторінці. *Limit* > 0.

2. *Page* – один набір даних (сторінка). Кількість сторінок визначається формулою: *Page* = (Кількість одиниць даних) / *Limit*.

3. *CRC32* – алгоритм для знаходження контрольної суми однієї сторінки даних.

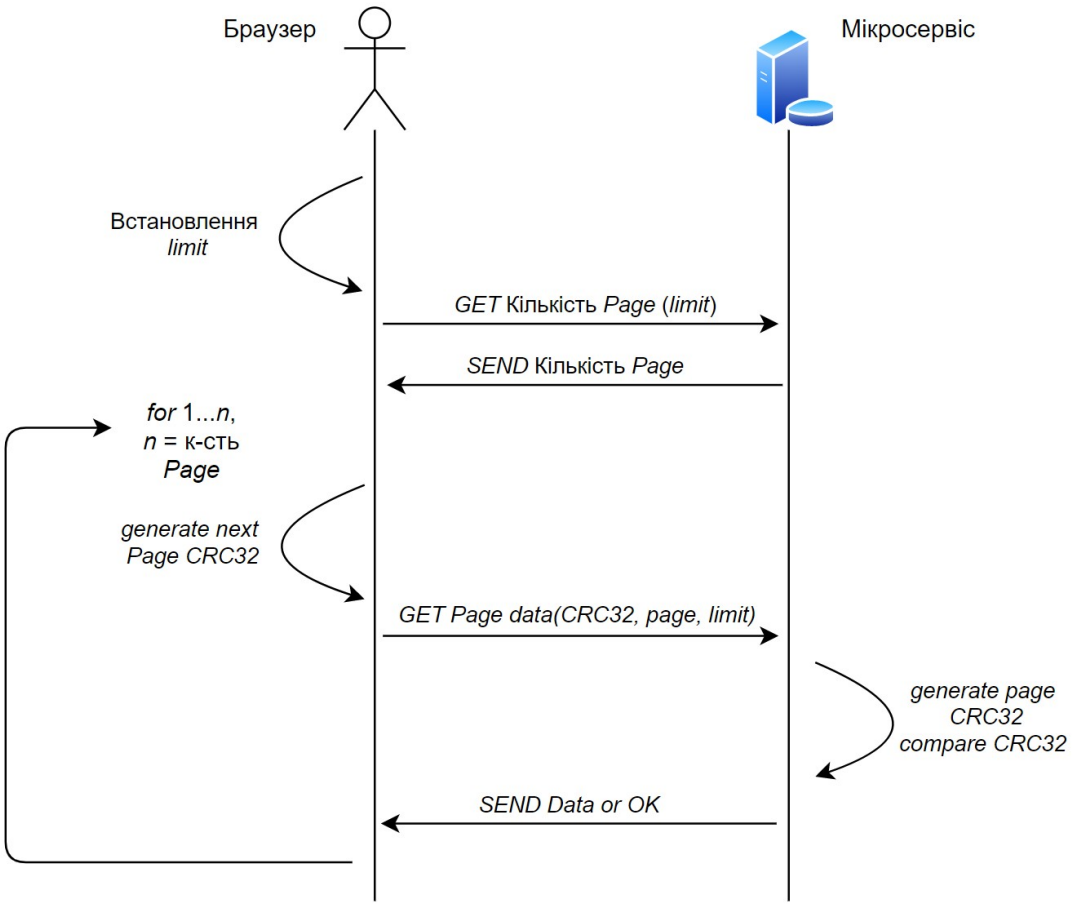

Рис.3.10. Візуалізація механізму синхронізації даних між клієнтом та мікросервісом

Механізм синхронізації буде автоматично виконуватись при ініціалізації веб-додатку. Сам процес є оптимальним з точки зору використання *CPU*, а також операцій *I/O* диску та мережі.

Рекомендується використовувати його окремо від основного потоку виконання *GUI* по двохнаправленому каналу передачі даних (*WebSockets*). У системі керування чат-ботом механізм буде використано у окремому потоці браузеру (*WebWorker*).

Використання запропонованого механізму забезпечить:

1. Ітеративне зменшення використовуваного трафіку.
2. Оптимізацію швидкості роботи веб-додатку.
3. Можливість використання веб-додатку у офлайн режимі.

3.4. Розгортання мікросервісів

Опишемо процес розгортання створеної серверної частини системи керування чат-ботом. Обрана архітектура дозволяє розгортання складових частин системи як на одній фізичній машині у рамках однієї ОС, так і на різних.

Наприклад, файловий сервіс вимагає великої кількості постійної пам’яті та високошвидкісне інтернет-з’єднання для збереження та передачі великих файлів, а сервіс розсилок, у порівнянні, – швидкодії *CPU* та збільшеної кількості оперативної пам’яті. Знання особливостей технічних вимог сервісів може зекономити ресурси при оренді хмарних сервісів або при купівлі власних.

Розглянемо варіант розгортання усіх сервісів на одній фізичній машині. Зазвичай будь-який сервер, що обслуговує веб-додатки або веб-сайти використовує для цілей балансування навантаження та перенаправлення запитів до потрібних процесів зворотній проксі. За допомогою зворотного проксі реалізуємо шифрування даних – обмін *SSL*-сертифікатами з клієнтами, а також відправлення статичних ресурсів (головна *HTML* сторінка веб-додатку та файли *JS*/*CSS*). Використаємо *NGINX* як зворотній проксі.

Кожному сервісу заздалегідь призначимо *IP* та порт (табл. 14). Оскільки запуск здійснюємо на єдиній ОС, то використаємо *IP – 127.0.0.1*. Порт самого сервісу та веб-служби сервісу – різні.

Таблиця 3.7

Мережеві налаштування мікросервісів

Сервіс	***port***	**Веб *port***
Авторизації	8081	9081
Взаємодії з *Telegram Bot API*	8082	-
Логіки чат-бота	8083	9083
Повідомлень	8084	9084
Файлів	8085	9085
Розсилок	8086	9086

Запит клієнта по протоколу *HTTP* отримує зворотній проксі – *NGINX*. Оскільки передбачається створення веб-додатку у вигляді *SPA*, то головна сторінка веб-додатку буде “/” та залишатиметься незмінною. Надалі *NGINX* з’ясовує, якому саме сервісу передбачався запит, та пересилає цей запит йому. Приклад конфігураційного файлу *NGINX* представлений у додатку А.

Кожний сервіс повинен унікально ідентифікуватися в мережевому оточенні для можливості доступу до нього.

Нехай основним доменом системи буде *domain.com*. Таким чином запит *domain.com/* – це запит головної сторінки веб-додатку. Наприклад, одним із варіантів є створення окремого *DNS* запису з піддоменом для кожного сервісу (*auth.domain.com*, *files.domain.com* тощо). В такому випадку цільовий сервіс дізнаємося з *HTTP HOST* заголовку. Іншим варіантом є доступ до сервісів через частину цільового *URI* (*domain.com/auth*, *domain.com/files* тощо). Даний варіант є простішим, оскільки при використанні різних піддоменів необхідно налаштовувати політику *CORS*.

Висновки за розділом

У цьому розділі розглянуті питання розробки серверної частини керування чат-ботом: здійснена декомпозиція системи на мікросервіси, спроектований *API* та схема БД для кожного мікросервісу.

Для декомпозиції системи на мікросервіси приведені два підходи: на основі бізнес можливостей та на основі субдоменної моделі. Для декомпозиції системи був обраний останній. У результаті отримано шість мікросервісів, кожний із яких має свою чітко окреслену зону відповідальності: взаємодія з *Telegram Bot API*, логіка чат-бота, авторизація користувачів, керування файлами, робота з повідомленнями, здійснення розсилок.

Після конкретизації списку мікросервісів була проведена робота по кожному: описано призначення, уточнені процеси та особливості роботи, створено *API*, розроблена схема БД. Сформовані та показані взаємозв’язки між мікросервісами.

Описані механізми обміну даними, що використовуються системою керування чат-ботом: *HTTP*, *WebSockets*, *gRPC*. Обґрунтоване використання стандартних протоколів, створений механізм для синхронізації даних між браузерами та мікросервісами.

У результаті створення мікросервісів окрема увага приділена процесу їх індивідуального запуску, а також процесу розгортання системи у цілому. Приведені два варіанти для розгортання системи: перший базується на єдиному домені з ідентифікацією сервісів по частині цільового *URI*, а другий – на створені окремих піддоменів для кожного сервісу. З’ясовано, що використання другого методу є недоцільним через необхідність налаштування політики *CORS*. Перший метод використовує *NGINX* для локації сервісів за єдиним *IP*.

РОЗДІЛ 4
РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ КЕРУВАННЯ ЧАТ-БОТОМ

4.1. Методологія проектування *UI*

Інтерфейс користувача *(UI)* – це дизайн програмного продукту, що визначає його структуру, форму, поведінку та стиль. Основною метою *UI* є представлення найпростішої та найпродуктивнішої взаємодії з ним для досягнення поставлених цілей користувача.

Список дисциплін дизайну користувацького інтерфейсу:

1. Дизайн *UX*. Користувацький досвід (*UX*) – послідовність дій користувача, що призводить його до бажаного результату. *UX* сконцентрований на послідовності дій, що відбуваються у процесі роботи з *UI*.

2. Дизайн взаємодії (*IxD*). Дослідження аспектів та закономірностей поведінки користувачів.

3. Зручність користування. Оцінка простоти вивчення функцій продукту.

4. Інформаційна архітектура (*IA*). Вивчення структури та представлення інформації. Основні задачі: пошук, навігація, організація, маркування.

5. Візуальний дизайн. Робота над кольорами, типографією, анімаціями та загальною естетикою продукту.

Процес дизайну *UI* – це ітеративний процес з чіткою послідовністю дій. Методологічно ці дії описуємо наступним чином:

1. Аналіз задачі. Визначаються основні завдання та компоненти пов’язані з ними. Для них описуються агенти (ті, хто виконує завдання), цілі та критерії для виконання завдання, вхідні дані (інформація, яка необхідна для виконання завдання), вихідні дані (інформація отримана у результаті виконання завдання),

методи (процедури для виконання завдання, кожна процедура описує низку задач для деталізації процесу).

2. Створення моделі користувача. Модель користувача базується на наборі концептуальних об’єктів та відповідного набору їх дій. З точки зору користувача ці об’єкти є абстрактними сутностями, поведінка яких є абсолютно послідовна та передбачувана у ході виконання будь-яких дій. Основні аспекти: закінченість, послідовність, використання конкретних об’єктів.

3. Відображення інформації. Цей етап включає: представлення та управління інформацією, отримання зворотного зв’язку. Основні аспекти: достовірність, економність та зручність.

4. Команди. Поділяються на два рівні: глобальні (вибір меню, підменю тощо) та локальні. Основні аспекти: послідовність та принцип найменших зусиль.

Основні концепції розробки сучасного *UI*, які будуть використані при розробці клієнтської частини системи:

1. *RWD* – це метод веб дизайну, який передбачає відображення інформації на пристроях з різними розмірами екрану у комфортний для людини спосіб. Таким чином зменшується необхідність постійного виконання таких операцій як масштабування та прокрутка. Адаптивний веб дизайн реалізується за допомогою використання сучасних допоміжних веб фреймворків для верстки, наприклад *Bootstrap*.

2. *MFD* – це метод, що передбачає створення веб верстки у першу чергу для мобільних пристроїв.

4.2. Порівняльний аналіз *SPA* та *SSR*

Односторінковий додаток (*SPA*) – це веб сайт, що оновлює свій вміст на основі навігаційних дій користувача без запиту до серверу для отримання нового *HTML* документу.

Життєвий цикл *SPA* представлений на рис. 4.1.

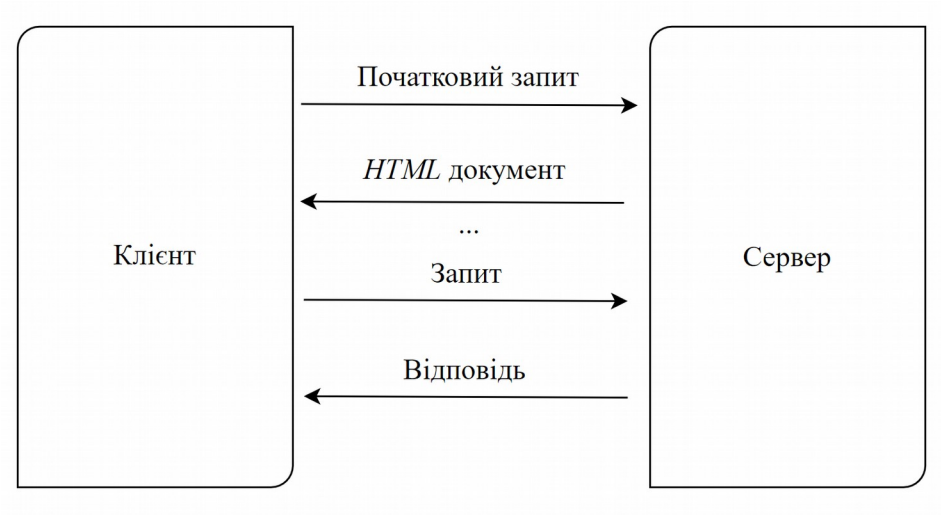

Рис.4.1. Життєвий цикл *SPA*

SPA повністю завантажується при першому запиті до серверу. В подальшій роботи частини сторінки оновлюються або замінюються новим вмістом, без потреби в завантаженні *HTML*-документів інших сторінок. При цьому частини даних, що потрібно відобразити, можуть завантажуватись з серверу у звичайному режимі. Це сповільнює початкове завантаження сайту, проте пришвидшує подальшу взаємодію з ним.

Переваги:

1. Пришвидшений час завантаження вмісту сторінок.

2. Покращений користувацький досвід.

3. Незалежність *back end* та *front end*, оскільки логіка відображення даних знаходиться в *SPA*.

Недоліки:

1. Активне використання *JavaScript* може призвести до проблем з швидкодією.

2. Відсутність реального вмісту сторінок при їх завантаженні ускладнює індексацію веб-сторінок пошуковиками.

3. Необхідність використання сторонніх веб-фреймворків вимагає їх додаткового підключення до додатку.

Візуалізація на стороні сервера (*SSR*) – це традиційний підхід до надання вмісту веб-сайту, який полягає у генерації *HTML*-документів на стороні серверу.

Переваги:

1. Традиційний підхід спрощує роботу пошуковиків, що покращує *SEO*.

2. Перший запит веб сторінок завжди швидший у порівнянні з *SPA*.

3. Ідеально підходить для статичних сайтів з невеликою кількістю оновлень даних на сторінках.

До недоліків можна віднести те, що генерування вмісту веб сторінок на стороні серверу вимагає більшого часу *CPU* та кількості оперативної пам’яті.

Оскільки система керування чат-ботом вимагатиме активного оновлення даних сторінок, то доцільно буде використати підхід *SPA*. Варто зазначити, що *React.js* можливо використати для його реалізації.

4.3. Структура проекту

Структура проекту – це впорядкований набір елементів, що відображають компонентні частини проекту та їх взаємозв’язки. Створення чіткої структури проекту – одна із найважливіших задач при розробці будь-якого ПЗ.

Компонентами системи можуть бути зовнішні залежності (завантажувальні модулі) та внутрішні компоненти (написані власноруч). При цьому зовнішні залежності зазвичай призначені для вирішення глобальних низкорівневих завдань, а внутрішні – для розв’язання конкретних прикладних проблем. Наприклад, у нашому випадку зовнішніми залежностями є: бібліотека *React.js*, компоненти *react-router-dom* та *history* для створення навігації у веб додатку, *socket-io-client*, бібліотека для спрощення процесу створення сучасного адаптивного *UI – mdbreact*.

Усі залежності та основні налаштування проекту прописуються у файлі *package.json*, який інтерпретується пакетним менеджером *npm* (див. додаток Б).

Проаналізуємо процес створення внутрішніх компонентів, взявши за основу вимоги до системи керування чат-ботом та особливості фреймворку *React.js*.

Для розробки веб додатку з використанням технології *React* необхідно створити основний *HTML* документ, що повинен мати базову розмітку та налаштовані мета-теги. При цьому до базової розмітки потрібно додати тег з унікальним атрибутом *id*. Цей тег стане вхідною точкою для відображення всього представлення веб додатку написаного на *React.js*.

Наступний крок полягає у створенні *js* файлу з підключенням головного компоненту веб додатку до вхідної точки, основних стилів та системи маршрутизації.

Окремою темою для розгляду є система маршрутизації по веб додатку. Оскільки використовується підхід *SPA*, то маршрутизація не може здійснюватися звичайним методом переходу по веб сторінкам. Проблеми, які необхідно вирішити наступні:

1. Забезпечити навігацію по веб додатку за допомогою стандартного тегу *HTML* – *<a>*.

2. Забезпечити перехід по сторінкам з коду.

Загальноприйнятий підхід до вирішення першої із проблеми – це використання спеціалізованої бібліотеки *React Router*.

Бібліотека *React Router* створює представлення навігації по веб додатку у вигляді розгалуженого набору компонентів з присвоєними їм *URL*-адресами. Цей механізм забезпечується використанням *API HTML5 – history*. Проте у загальному випадку неможливо вирішити другу проблему, оскільки механізм передбачає створення та використання внутрішнього об’єкту *history*, доступ до якого обмежений. Обмеження доступу веде до нераціональності використання загальноприйнятного підходу.

Використаємо специфіку *React Router* – можливість передавати параметром заздалегідь створений об’єкт *history*. Цей об’єкт створюємо та експортуємо з окремого модуля, у результаті отримуємо *Singleton* – тобто єдиний екземпляр класу. Таким чином єдиний екземпляр *history* буде доступний для перегляду та редагування з будь-якої точки додатку.

Стилі *CSS* підключаємо з метою редагування зовнішнього вигляду *HTML* елементів та для вказання параметрів адаптивної розмітки сторінок. Використовуємо стилі бібліотек: *Bootstrap*, *MDBootstrap*, *fontawesome*.

Представимо компоненти веб додатку у вигляді дерева (рис. 4.2).

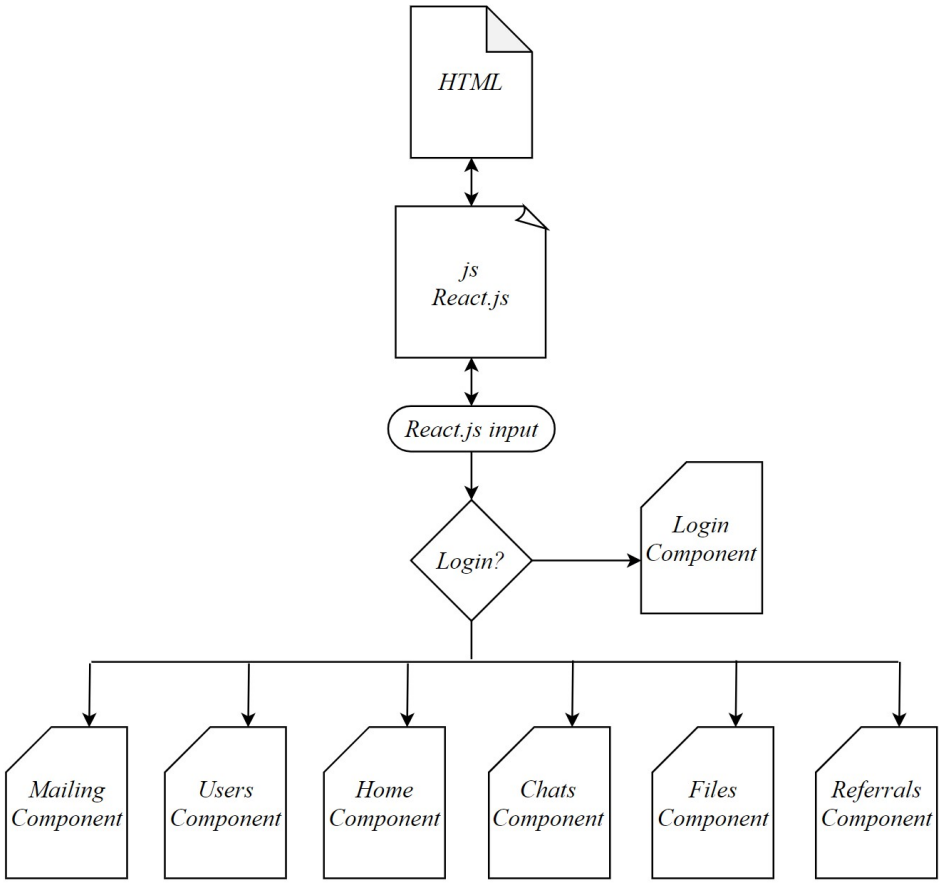

Рис.4.2. Дерево компонентів веб додатку

Кожен компонент є незалежною одиницею та існує для виконання конкретних завдань. Представлене дерево є достатньо високорівневим поглядом на компоненту складову системи, в дійсності ж дерево продовжує рости в глибину та ширину, розгалужуючись на компоненти з меншими зонами відповідальності. Компонентом з найменшою зоною відповідальності можна вважати базові *HTML* теги. В деяких випадках раціональним рішенням є використання низкорівневих компонентів одразу у декількох місцях, що зазвичай обумовлюється повторенням функціоналу у різних частинах системи. Прикладами таких компонентів є: *SearchBar*, *FileInfoDialog*.

4.4. Опис компонентів

При переході на веб додаток здійснюється його ініціалізація та відбувається процедура автентифікації користувача за допомогою виклику методу *API*

мікросервісу авторизації – *validateUserSession()*. На основі отриманої відповіді здійснюється відображення головних компонентів системи або компоненту *Login*.

Компонент *Login* містить форму з двома полями для вводу імені користувача та паролю (рис. 4.3).

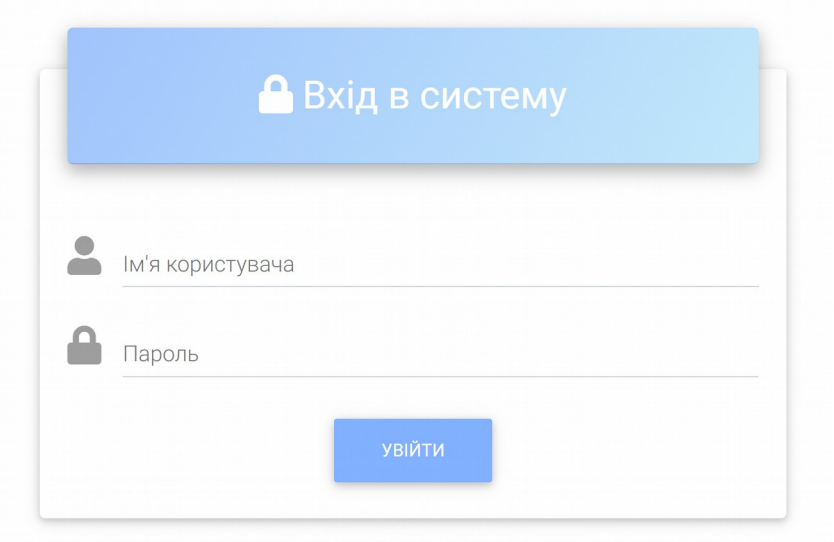

Рис.4.3. Компонент *Login*

Для відображення будь-якого виду помилок користувачу використовується компонент *Toast* – невелике повідомлення у кутку екрану з текстовим вмістом (рис. 4.4). Повідомлення мають таймер та показуються протягом декількох секунд поверх інших вікон.

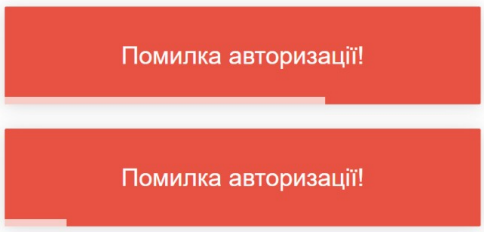

Рис.4.4. Компоненти *Toast*

Усі сторінки веб додатку мають компонент *Header* (рис. 4.5). Завдяки використанню *React.js* його вміст існує лише в одному екземплярі, та при зміні він автоматично оновлюється на усіх сторінках. Основне призначення даного компоненту – навігація та доступ до загальних операцій. Зазвичай у додатках на

основі технології *SSR* аналогічні компоненти прописуються окремо для кожної із сторінок, що при зміні вмісту компоненту веде до необхідності редагування кожної.

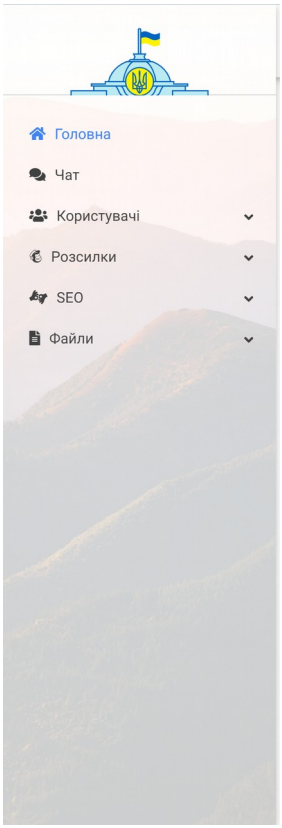

Рис.4.5. Компонент *Header*. Навігація *Sidebar*

У області *Sidebar* знаходяться посилання для навігації веб додатком. Область *Navbar* містить посилання до загальних операцій (рис. 4.6).

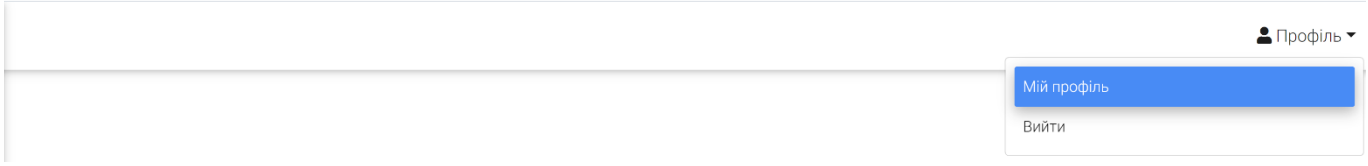

Рис.4.6. Компонент *Header*. Навігація *Navbar*

Компонент *UsersManagement* здійснює відображення даних користувачів чат-бота у вигляді таблиці.

Усі таблиці в додатку мають налаштування та інформацію про:

1. Кількість показаних сутностей.
2. Сторінки.
3. Пошук.
4. Сортування.

У випадку компоненту *UsersManagement* (рис. 4.7) дані користувачів наступні: *ID*, нікнейм, ім’я, статус, дата створення та оновлення.

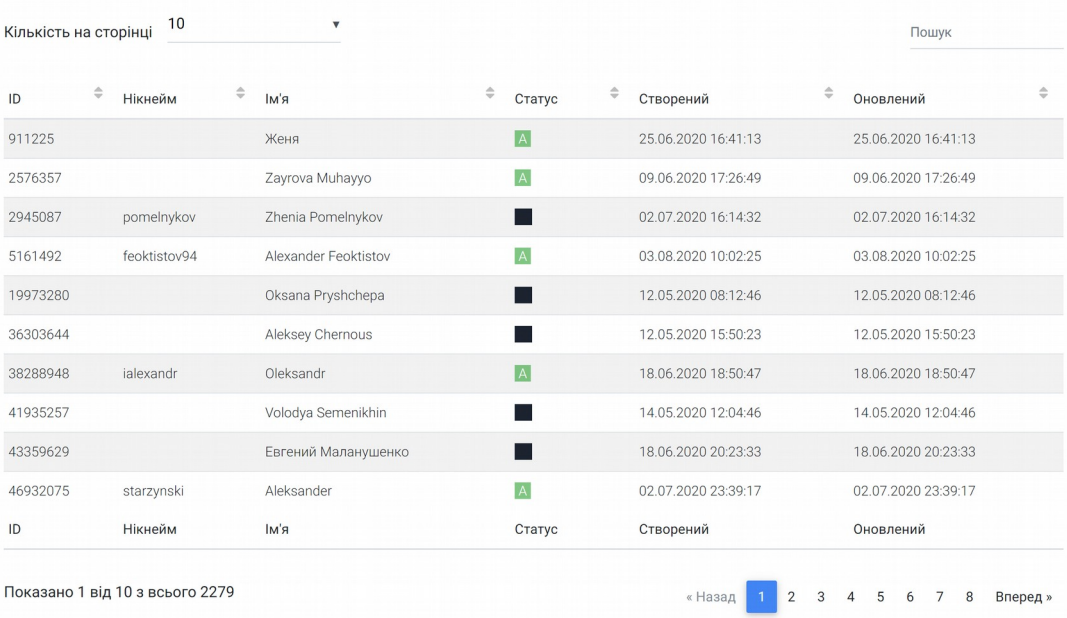

Рис.4.7. Компонент *UsersManagement*

Компонент *FileManagement* призначений для роботи з файлами (рис. 4.8). Основні можливості:

1. Перегляд, створення та редагування файлів.
2. Пошук файлів по ключовим фразам.
3. Фільтрація файлів по категоріям.
4. Зміна режиму публічності файлів.

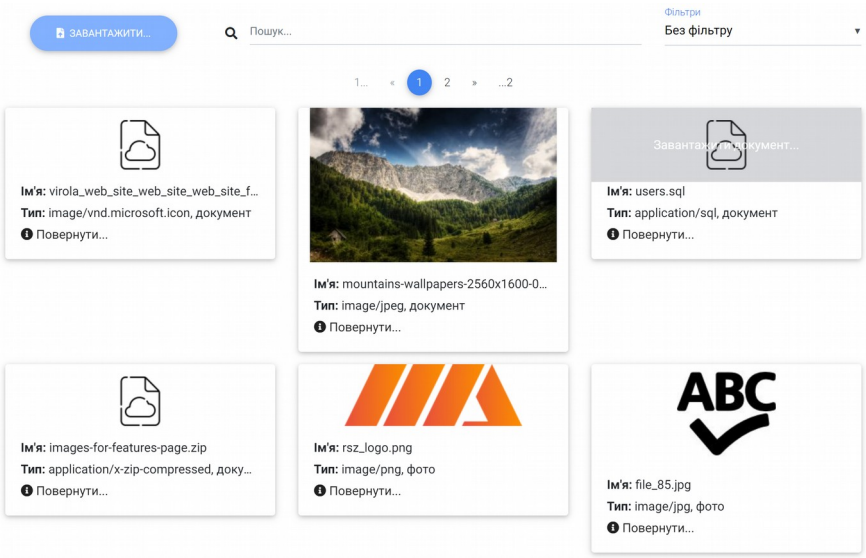

Рис.4.8. Компонент *FileManagement*

По кожному файлу зазначається інформація, яка представлена на рис. 4.9.

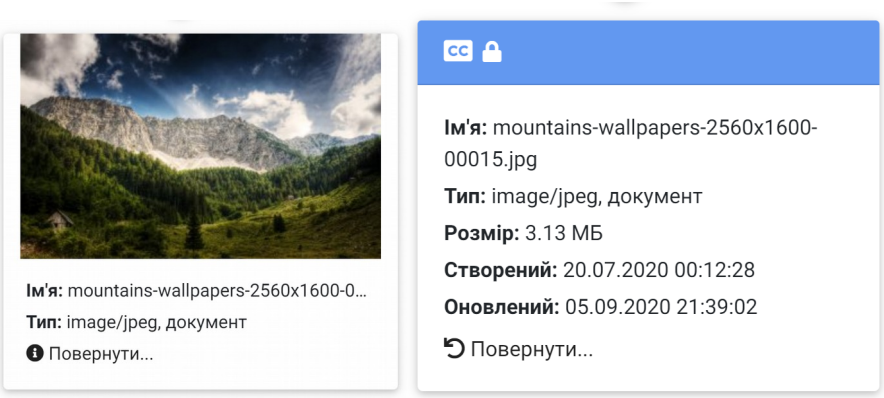

Рис.4.9. Основна інформація файлу

Додаткової уваги заслуговує функція зміни режиму публічності файлів (рис. 4.10). При ситуації, коли файл необхідно зробити доступним для завантаження без авторизації до системи, можливо поділитись посиланням на нього, попередньо увімкнувши режим файлу – “Публічний”. Це свого роду невеличкий файлообмінник.

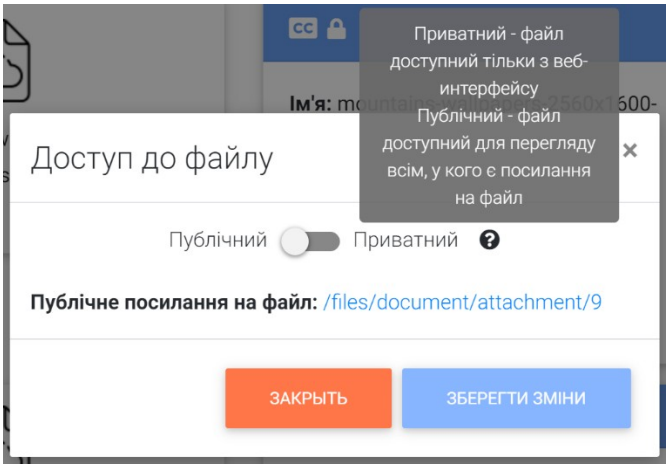

Рис.4.10. Зміна режиму публічності файлу

Для створення файлів використовується компонент *FileManagementCreateDialog* (рис. 4.11). У формі необхідно додати файл, за необхідності вказати підпис, обрати режим публічності. Максимальний розмір файлів – 20 МБ, що обумовлено обмеженнями *Telegram Bot API*, оскільки завантажені файли до системи заздалегідь вважаються такими, що будуть у подальшому використаними та завантаженими на сервери *Telegram*.

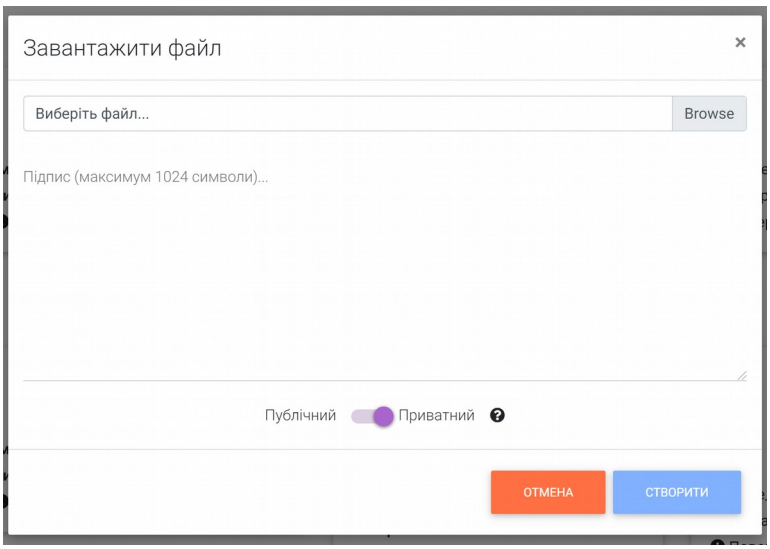

Рис.4.11. Компонент *FileManagementCreateDialog*

До додаткових можливостей відносимо пошук та фільтри, що дозволяють у зручний спосіб відобразити інформацію по обраним критеріям (рис. 4.12).

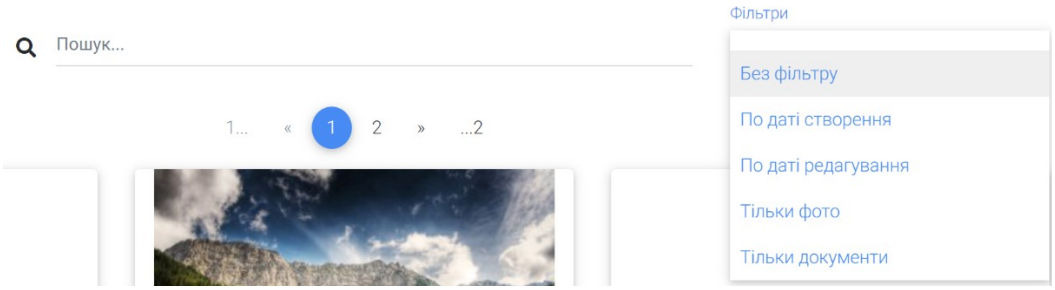

Рис.4.12. Пошук та фільтрація файлів

Компонент *ReferralManagement* призначений для відображення інформації про рефералів (рис. 4.13). Реферали у системі виконують функції *SEO*: створюють посилання для відслідковування користувачів.

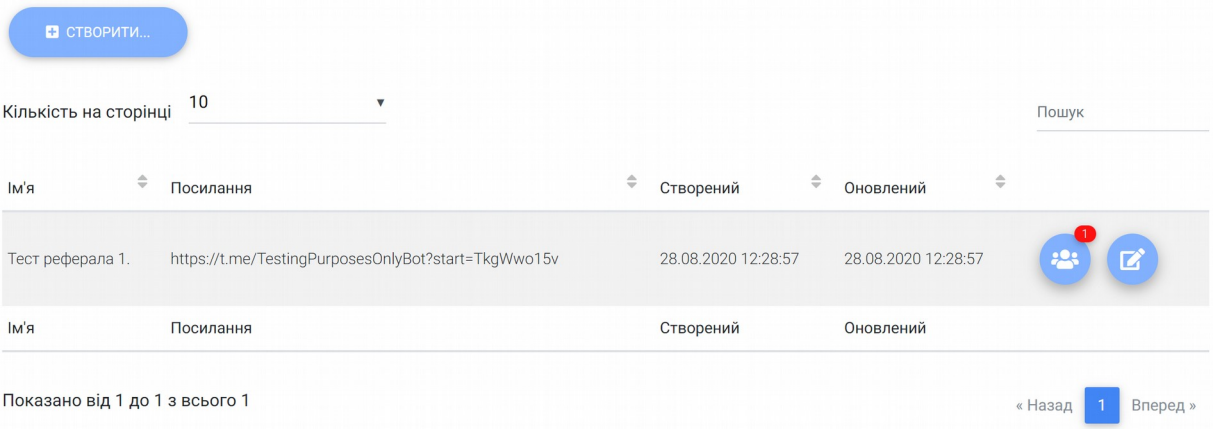

Рис.4.13. Компонент *ReferralManagement*

Створення реферaлу відбувається за допомогою компоненту *ReferralManagementCreateDialog* (рис. 4.14), що має форму для вводу імені реферaлу. При його створені в автоматичному режимі генерується унікальне посилання, яке прив’язане до нього.

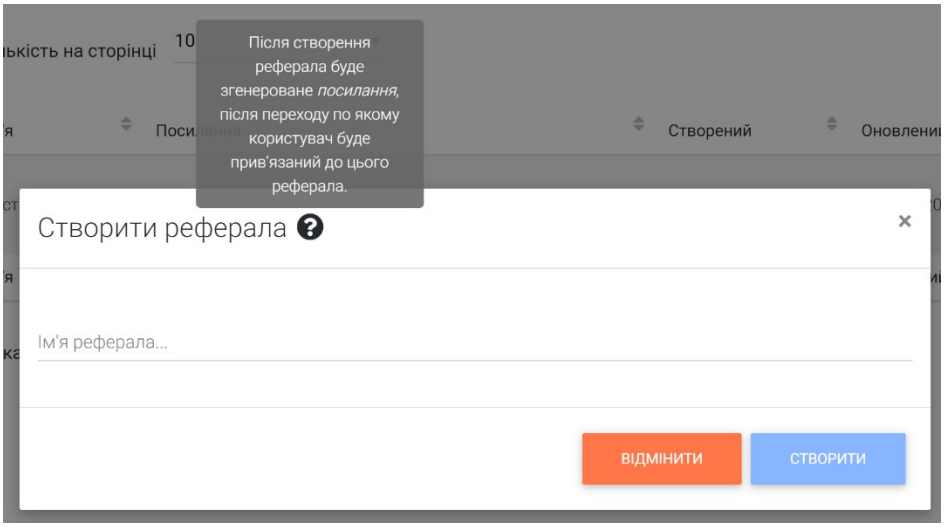

Рис.4.14. Компонент *ReferralManagementCreateDialog*

Кожен перехід за посиланням реферaлу записується, а у подальшому список користувачів відображається за допомогою компоненту *ReferralInfo* (рис. 4.15).

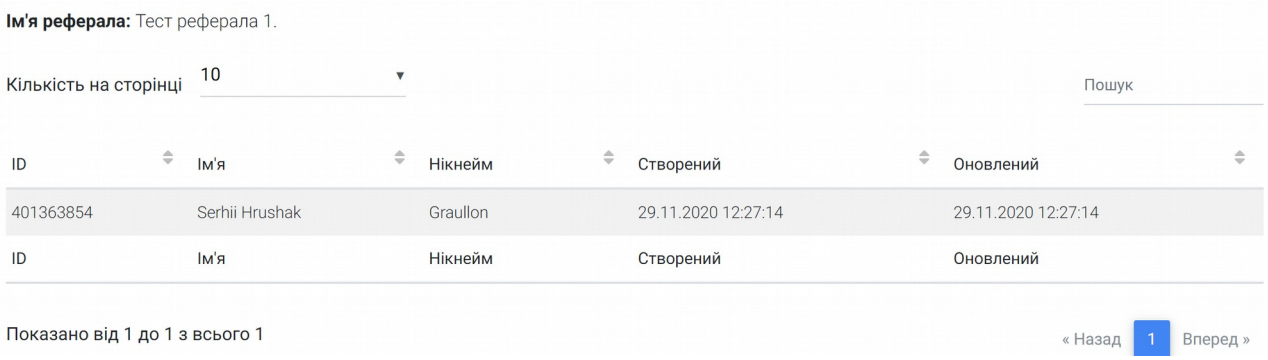

Рис.4.15. Компонент *ReferralInfo*

Розглянемо реалізацію розсилок. Існує два типи розсилок:

1. Заплановані.

2. Персоналізовані.

Для створення запланованих розсилок зазначаються наступні параметри:

1. Текст повідомлення/файл.

2. Клавіатура (опціонально).

3. Сегмент користувачів.

4. Фільтр по даті створення користувачів (опціонально).

5. Дата та час відправки.

Для персоналізованих розсилок замість дати та часу відправки зазначається відлік часу до відправки у хвилинах (рис. 4.16). Процес розсилки з усіма деталями представлено на рис. 4.17.

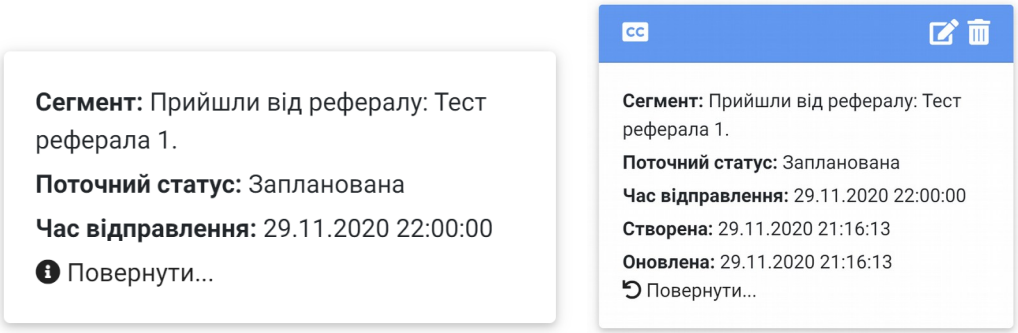

Рис.4.16. Інформація про заплановану розсилку

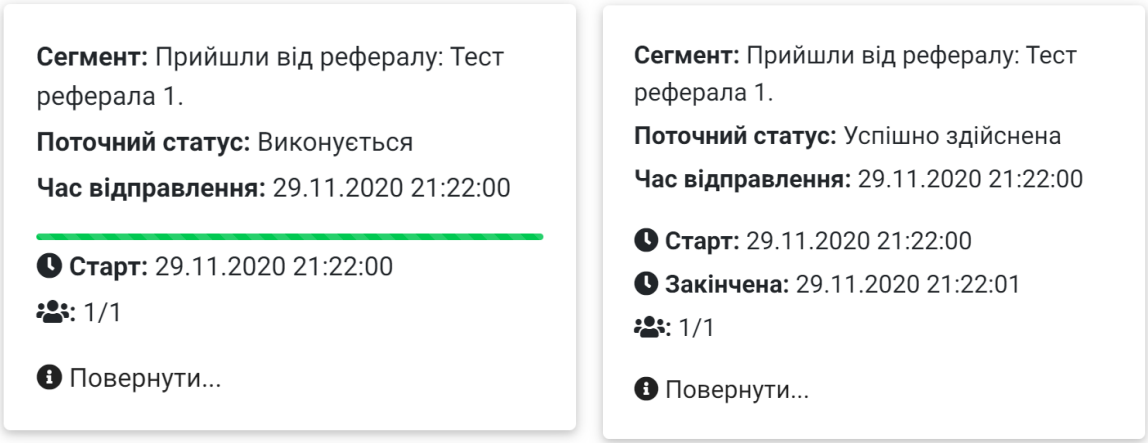

Рис.4.17. Процес здійснення розсилки

Компонент *ChatsManagement* (рис. 4.18) призначений для роботи з повідомленнями. Основні можливості:

1. Відображення інформації про користувачів, що написали повідомлення чат-боту, у вигляді кімнат.

2. Відображення історії повідомлень між чат-ботом та користувачем.

3. Можливість листування з обраними користувачами.

Варто зазначити, що *Telegram* не надає стандартизованого функціоналу по листуванню між чат-ботами та користувачами, тому реалізована можливість є його розширенням.

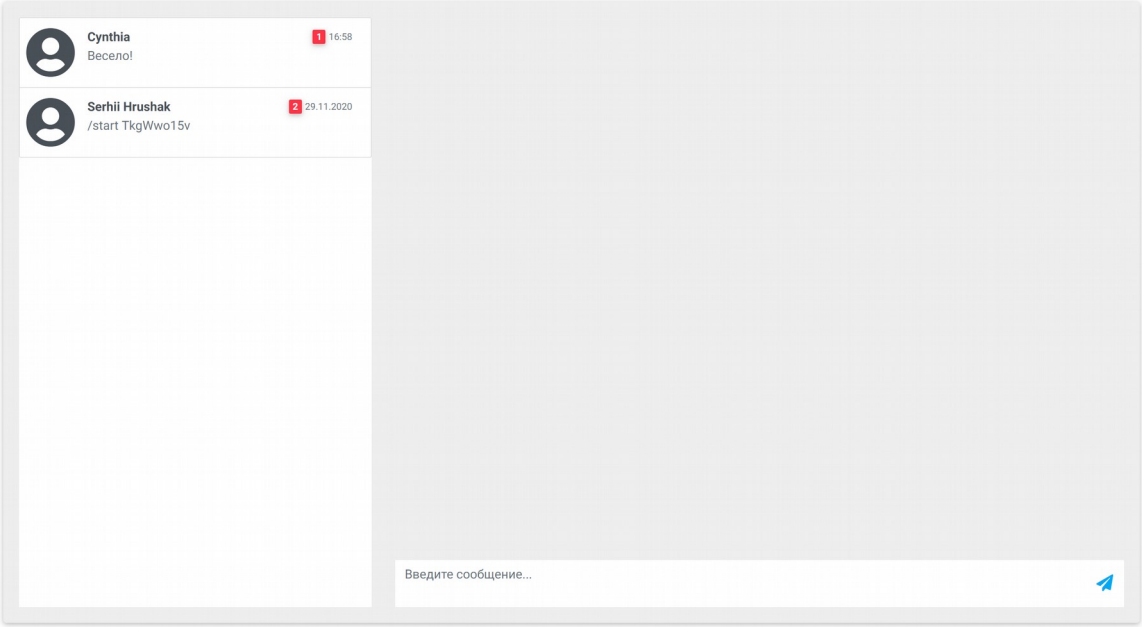

Рис.4.18. Компонент *ChatsManagement*

ChatsManagement поділяється на компоненти: *ChatRoomList*, *ChatRoom*, *ChatMessages*, імена яких пояснюють їх призначення та функціонал.

Повна топологія системи з серверною та клієнтською частинами представлена у додатку В. Програмний код системи прикладається разом з матеріалами дипломної роботи на диску.

Висновки за розділом

У розділі розглянуті питання розробки клієнтської частини системи керування чат-ботом: приведений огляд методологій проектування *UI*, здійснено порівняння *SPA* та *SSR*, створена структура проекту на *React.js*, проведений опис компонентів системи.

У результаті з’ясовані найкращі практики та підходи при проектуванні сучасних *UI*: *UX*, *IxD*, зручність користування, *IA*, візуальний дизайн. Приведений

опис процесу проектування будь-якого *UI* у вигляді етапів. У загальному вигляді представлені основні концепції при розробці *UI*: *RWD* та *MFD*.

Зроблений аналіз *SPA* та *SSR*, враховуючи усі переваги та недоліки, а також звернувши увагу на основну технологію розробки – *React.js*, показав, що найкращим вибором для створення системи керування чат-ботом буде *SPA*.

Перед початком програмування веб-інтерфейсу було проведене його компонентне проектування з узагальненим описом веб-технологій, що використані при розробці.

Приведений опис компонентів у останньому підрозділі – є успішним результатом програмування та створення клієнтського веб-додатку системи керування чат-ботом.

ВИСНОВКИ

У результаті виконання дипломної роботи досліджені технології чат-ботів та створена система керування чат-ботом на базі технології *React*: здійснене дослідження та порівняння сучасних платформ для розробки чат-ботів, проаналізовані та обґрунтовані технології для розробки системи керування чат-ботом, здійснена розробка серверної та клієнтської частин системи.

До створюваної системи керування чат-ботом поставлені чіткі функціональні вимоги, які були повністю реалізовані за результатами дипломної роботи:

1. Система дозволяє перегляд користувачів чат-ботом (загальну інформацію, дату останнього візиту тощо).
2. Система надає функціонал чату, що призначений для обміну повідомленнями між користувачами та чат-ботом, зберігає історію листувань.
3. Система реалізує окремий файловий сервіс для збереження будь-яких файлів.
4. Система надає функціонал розсилок з можливістю планування текстових та медіа повідомлень окремим групам користувачів, зберігає історію розсилок до архіву.
5. Система веде облік та надає доступ до перегляду загальних статистичних даних.

Додатково до зазначеного функціоналу система здійснює активний мережевий обмін даними з серверами *Telegram*, між мікросервісами та з клієнтами використовуючи сучасні протоколи та механізми для обміну даними.

Увесь зазначений функціонал працює у повній мірі. Розгортання системи можливе як у межах однієї фізичної машини та ОС, так і на різних машинах з різними ОС. Щодо процесу розгортання системи надані рекомендації та конфігураційні файли.

Перед початком роботи над дипломною роботою були поставлені конкретні задачі, у результаті виконані з них наступні:

1. Розкрита проблематика створення та використання чат-ботів.

2. Проведений аналіз сучасних сервісів для створення чат-ботів та обрана платформа *Telegram* для опрацювання.

3. Сформовані основні функціональні вимоги до створюваної системи.

4. Проведено аналіз технологій та підходів до розробки веб-серверів та веб-додатків. Детально розглянута технологія *React* як одна з провідних у сфері розробки сучасного *UI*.

5. Детально описані процеси передачі інформації в створеній системі та запропоновані свої.

6. Спроектовані схеми баз даних та обраний мікросервісний архітектурний підхід.

7. Розроблена серверна частина системи, зазначені використані технології та протоколи, описані мікросервіси та їх призначення, розглянуті процеси обміну даними.

8. Розроблена клієнтська частина системи на базі технології *React*, представлені результати розробки.

9. Надані рекомендації щодо процесу розгортання системи.

У першому розділі розглянута тема чат-ботів: надано визначення, основні класифікації, описані механізми обміну даними з платформами чат-ботів та надані рекомендації щодо їх використання, проведено дослідження сучасних платформ чат-ботів та здійснено порівняння їх функціональних можливостей, обраний месенджер *Telegram* для подальшого опрацювання. Встановлено, що існують два механізми зв’язку з платформами чат-ботів: *polling* та *webhook*. Процес порівняння месенджерів *Facebook*, *Telegram*, *Skype*, *Viber* містив дослідження їх технічних можливостей, переваг та недоліків. Проведення дослідження було здійснено за допомогою аналізу документацій кожного месенджеру та особистого тестування функціональних можливостей. У подальшому були зазначені основні функціональні можливості обраної платформи, розглянутий *Telegram Bot API* та основні правила і механізми передачі та кодування даних при роботі з ним. З’ясовано, що створення чат-ботів на платформі *Telegram* відбувається на безоплатній основі з невеликою

кількістю функціональних обмежень, а керування створеним ботом відбувається за допомогою спеціального токену доступу.

У другому розділі розглянуті питання постановки функціональних вимог до системи та аналізу і вибору технологій для розробки системи керування чат-ботом. У ньому було з’ясовано, що виконання поставленої мети по створенню системи керування чат-ботом вимагає значних інженерних навичок та умінь, а вибір технологій для розробки безпосередньо впливатиме на швидкість та легкість цього процесу. Розроблена система керування чат-ботом – це інформаційна система, що вимагає постійного перегляду, аналізу та пересилання даних. Тому основними технологіями стали: *Node.js*, *React.js*, *SQLite* та фреймворки *Express.js* та *Socket.io*. При цьому *Node.js* та *Express.js* використані для створення основної логіки та веб-серверів, *React.js* – для програмування клієнтського додатку, *SQLite* – для створення баз даних. Окрема увага приділена вибору архітектури ПЗ, а саме – мікросервісам. Здійснено порівняння мікросервісів та монолітного підходу та з’ясовано, що за допомогою мікросервісної архітектури можливо створити набір слабкозв’язних частин системи.

У третьому розділі здійснений процес декомпозиції системи на мікросервіси, що став першим етапом у розробці серверної частини системи керування чат-ботом. Після з’ясування основних компонентів системи був проведений етап опису основних функціональних можливостей сервісів та їх зон відповідальності, а також представлений детальний опис *API* кожного сервісу та схем баз даних. Також важливим етапом став опис механізмів обміну даними як всередині системи, так і за її межами. Було визначено, що учасниками процесів обміну даними є: *Telegram Bot API*, мікросервіси, браузери користувачів. На базі цього було запропоноване рішення, що використовує механізм синхронізації даних між мікросервісами та браузерами користувачів, і таким чином веде до поступового зменшення використовуваного трафіку та пришвидшення роботи системи. Останнім етапом став опис розгортання мікросервісів: описані мережеві адреси, надані рекомендації щодо цього процесу, розкрита роль зворотного проксі *NGINX* у топології системи.

Четвертий розділ розкриває суть реалізації клієнтського веб-додатку: узагальнено описані методології проектування *UI*, основним підходом до розробки обраний тип додатку SPA, уточнені використані бібліотеки стилів *CSS*: *MDBootstrap* та *fontawesome*, розроблений компонентний склад клієнтського веб-додатку, детально описане призначення кожного та представлені результати створення та роботи кожного.

Остаточним результатом роботи стало проведене дослідження технологій роботи чат-ботів, а також створення системи керування чат-ботом на базі технології *React*, що базується на платформі *Telegram*.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Провотар О.І. Особливості та проблеми віртуального спілкування за допомогою чат-ботів / О.І. Провотар, Х.А. Клочко // Наукові праці ВНТУ: Інформаційні технології та комп'ютерна техніка. – 2013. – № 3. – 6 с.

2. Сухас У. *Oracle Intelligent Bots*: Чат-боты с искусственным интеллектом [Інтернет ресурс]. – *Oracle*, 2017. – Режим доступу: *https://www.oracle.com/a/ocom/docs/chatbots.pdf* вільний.

3. Украинец И. Системы аналитики для чат-ботов [Інтернет-ресурс]. – Режим доступу: *http://ilyaukrainets.ru/chatbot/sistemy-analitiki-dlya-chat-botov/* вільний.

4. *A curated directory of chat bot resources & tools* [Інтернет-ресурс]. – Режим доступу: *http://www.botsfloor.com/botstash/products/?category=Development%20Platforms* вільний.

5. *Chatbot Report 2018: Current landscape of how people create chatbots and how users expect to interact with them* [Інтернет-ресурс]. – Режим доступу: *https://elearningindustry.com/chatbots-for-learning-support-10-reasonshttps://naiz.chat/NAIZ-report-18072018.pdf* вільний.

6. *Kumar C. 9 AI Platform to Help you in Creating Facebook Chatbot* [Інтернет-ресурс]. – Режим доступу: *https://geekflare.com/create-facebook-chatbot/* вільний.

7. *Messaging apps are now bigger than social networks* [Інтернет-ресурс]. – Режим доступу: *https://www.businessinsider.com/the-messaging-app-report-2015-11* вільний.

8. *Sheth B. The Bot Lifecycle. What to know before you make your chatbot* [Інтернет ресурс]. – Режим доступу: *https://chatbotsmagazine.com/the-bot-lifecycle-1ff357430db7* вільний.

9. *Michiels. E. Modelling Chatbots with a Cognitive System Allows for a Differentiating User Experience* [Інтернет-ресурс]. – Режим доступу: *http://ceur-ws.org/Vol-2027/paper24.pdf* вільний.

10. *Muldowney O. Chatbots. An Introduction And Easy Guide To Making Your Own* / *O. Muldowney*. – *Dublin*: *Curses & Magic*, 2017. – 69 p.

11. *RFC 7578 - Returning Values from Forms: multipart/form-data* [Інтернет-ресурс]. – Режим доступу: *https://tools.ietf.org/html/rfc7578* вільний.

12. *Telegram Bot API* [Інтернет-ресурс]. – Режим доступу: *https://core.telegram.org/bots/api* вільний.

13. Richardson C. Microservices Patterns / Chris Richardson. – 2018. – 520 p.

14. *RFC 6455 - The WebSocket Protocol* [Інтернет-ресурс]. – Режим доступу: *https://tools.ietf.org/html/rfc6455* вільний.

15. Ушакова Г. Д. Особенности виртуального общения посредством чатов / Г.Д. Ушакова, Ю.В. Балабанова // Филологический журнал : межвузовский сборник научных статей. – 2004. – Вып. XII. – С. 59 – 61.

16. Асмус Н. Г. Лингвистические особенности виртуального коммуникативного пространства : автореф. дис. ... канд. филол. наук : 10.02.19 «Теория язика» / Н. Г. Асмус. – Челябинск: Челябинский гос. ун-т, 2005. – 23 с.

17. Ясницкий Л.В. Интеллектуальные системы / Л.В. Ясницкий. – М.: Лаборатория знаний, 2016. – 221 с.

18. *Beaver L. The Chatbots Explainer* [Інтернет-ресурс] / *Web*-сайт: *BI Intelligence Copyright*. – Режим доступу: *https://www.businessinsider.com/intelligence/chatbots-explainer* вільний.

19. Шафер С. *HTML, XHTML* и *CSS*. Библия пользователя, 5-е издание / Стивен Шафер. – М.: «Диалектика», 2010. – 656 с.

20. Крейн Д. *Ajax in Practice* / Д. Крейн, Б. Бибо, Д. Сонневельд. – М.: Вильямс, 2007.

21. Макфарланд Д.С. Разработка сайтов на *JavaScript* и *jQuery* Исчерпывающее руководство / Д.С. Макфарланд. – С.: *O'Reilly*, 2012. – 587 с.

22. Аллан А. Программирование для мобильных устройств / А. Аллан. – СПБ.: *O'Reilly*, 2012. – 188 с.

23. Аллан А. Клиентская разработка для профессионалов. *Node.js* / А. Аллан. – СПБ.: Питер, 2017. – 220 с.

24. Гарднер Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби. – М.:Вильямс, 2014. – 268 с.

25. Хансен Г. Дж. Базы данных: разработка и управление / Г.Дж. Хансен. – М.: БИНОМ, 2010. – 704 с.

26. Грофф Д. *SQL*: полное руководство / Д. Грофф, П. Вайнберг. – К.: *BHV*, 2005. – 608 с.

27. *Nedelcu C. Nginx HTTP Server* / *C. Nedelcu. – N.*: *Packt*, 2010. – 348 *p*.

28. *Aivaliotis D. Mastering Nginx* / *D. Aivaliotis*. *– N.*: *Packt*, 2013. – 322 *p*.

29. Hrushak S., Pavlenko C. Advantages of DNS-over-HTTPS over DNS // Computer and information systems and technologies (Kharkiv, Ukraine, 2020). – 2020.

30. Hrushak S., Pavlenko C. Virtual Private Network and its use in secured corporative networks // Computer and information systems and technologies (Kharkiv, Ukraine, 2019). – 2019.

Додаток А

Приклад конфігураційного файлу *NGINX* для тестування системи

```
http {
        index index.html;
        sendfile on;
        include ./conf/mime.types;
        default_type application/octet-stream;

        server {

                        listen 127.0.0.1:8080;

                        proxy_http_version 1.1;
                        proxy_set_header Upgrade $http_upgrade;
                        proxy_set_header Connection 'upgrade';
                        proxy_set_header Host $host;
                        proxy_cache_bypass $http_upgrade;

                        location /auth/ {
                                proxy_pass http://127.0.0.1:9081/;
                        }

                        location /bot/ {
                                proxy_pass http://127.0.0.1:9083/;
                        }

                        location /messages/ {
                                proxy_pass http://127.0.0.1:9084/;
                        }

                        location /files/ {
                                proxy_pass http://127.0.0.1:9085/;
                        }

                        location /mailing/ {
                                proxy_pass http://127.0.0.1:9086/;
                        }

                        root ../www/;

                        location / {}
        }
}
```

Додаток Б

Конфігураційний файл *package.json* клієнтської частини системи

```
{
  "name": "webappfrontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.5.0",
    "@testing-library/user-event": "^7.2.1",
    "history": "^4.10.1",
    "js-cookie": "^2.2.1",
    "mdbreact": "git+ssh://git@github.com:Graullon/mdbreact.git",
    "react": "^16.13.1",
    "react-dom": "^16.13.1",
    "react-flatpickr": "^3.10.4",
    "react-router-dom": "^5.2.0",
    "react-scripts": "3.4.1",
    "socket.io-client": "^2.3.0"
  },
  "proxy": "http://127.0.0.1:8080",
  "scripts": {
    "start": "set HOST=127.0.0.1&&react-scripts start",
    "build": "react-scripts build"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Додаток В

Топологія створеної системи керування чат-ботом